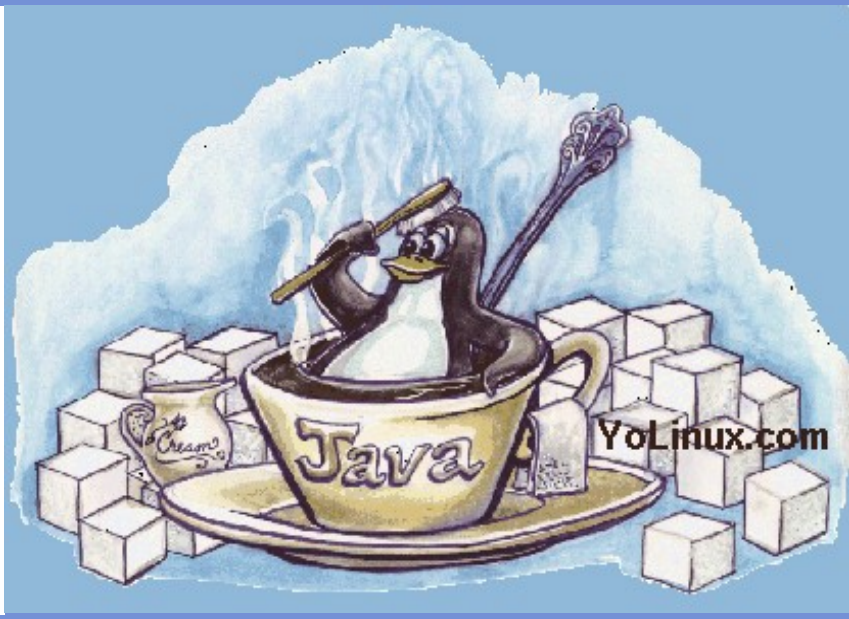
Fundación Código Libre Dominicano

Lic. Henry Terrero.
hterrero@codigolibre.org

Ing. Jose Paredes.
jparedes@codigolibre.org



Desarrollo De aplicaciones Con Java

TEMARIO CURSO DE JAVA.

Modulo I.

1. **Conceptos Básicos.**
    - Breve historia de Java
    - El compilador de Java
    - La Java Virtual Machine
    - Las variables PATH y CLASSPATH

2. **Características del Lenguaje.**
    - Variables y Tipos de Datos
    - Operadores
    - Control de Flujo
    - Arrays y Cadenas
    - Ejemplos

3. **Objetos, Clases e Interfaces.**
    - Conceptos de programación orientada a objetos
    - Crear y Utilizar Objetos
    - Declarar Clases
    - La Clase como generadora de objetos
    - Herencia
    - Métodos de objeto
    - Paso de argumentos a métodos
    - Métodos de clase (static)
    - Constructores
    - Métodos sobrecargados (overloaded)
    - Qué es un package
    - Ejemplos

Conceptos Básicos.

Breve historia de Java

Java surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. La reducida potencia de cálculo y memoria de los electrodomésticos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido.

Debido a la existencia de distintos tipos de CPUs y a los continuos cambios, era importante conseguir una herramienta independiente del tipo de CPU utilizada. Desarrollaron un código "neutro" que no dependía del tipo de electrodoméstico, el cual se ejecutaba sobre una "máquina hipotética o virtual" denominada Java Virtual Machine (JVM). Era la JVM quien interpretaba el código neutro convirtiéndolo a código particular de la CPU utilizada. Esto permitía lo que luego se ha convertido en el principal lema del lenguaje: "Write Once, Run Everywhere". A pesar de los esfuerzos realizados por sus creadores, ninguna empresa de electrodomésticos se interesó por el nuevo lenguaje.

Como lenguaje de programación para computadores, Java se introdujo a finales de 1995. La clave fue la incorporación de un intérprete Java en la versión 2.0 del programa Netscape Navigator, produciendo una verdadera revolución en Internet. Java 1.1 apareció a principios de 1997, mejorando sustancialmente la primera versión del lenguaje. Java 1.2, más tarde rebautizado como Java 2, nació a finales de 1998.

Al programar en Java no se parte de cero. Cualquier aplicación que se desarrolle "cuelga" (o se apoya, según como se quiera ver) en un gran número de clases preexistentes. Algunas de ellas las ha podido hacer el propio usuario, otras pueden ser comerciales, pero siempre hay un número muy importante de clases que forman parte del propio lenguaje (el API o Application Programming Interface de Java). Java incorpora en el propio lenguaje muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (threads, ejecución remota, componentes, seguridad, acceso a bases de datos, etc.). Por eso muchos expertos opinan que Java es el lenguaje ideal para aprender la informática moderna, porque incorpora todos estos conceptos de un modo estándar, mucho más sencillo y claro que con las citadas extensiones de otros lenguajes. Esto es consecuencia de haber sido diseñado más recientemente y por un único equipo.

El principal objetivo del lenguaje Java es llegar a ser el "nexo universal" que conecte a los usuarios con la información, esté ésta situada en el ordenador local, en un servidor de Web, en una base de datos o en cualquier otro lugar.

Java es un lenguaje muy completo (de hecho se está convirtiendo en un macro-lenguaje: Java 1.0 tenía 12 packages; Java 1.1 tenía 23 y Java 1.2 tiene 59). En cierta forma casi todo depende de casi todo. Por ello, conviene aprenderlo de modo iterativo: primero una visión muy general, que se va refinando en sucesivas iteraciones. Una forma de hacerlo es empezar con un ejemplo completo en el que ya aparecen algunas de las características más importantes.

La compañía Sun describe el lenguaje Java como "simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico". Además de una serie de halagos por parte de Sun hacia su propia criatura, el hecho es que todo ello describe bastante bien el lenguaje Java, aunque en algunas de esas características el lenguaje sea todavía bastante mejorable. Algunas de las anteriores ideas se irán explicando a lo largo de este manual.

Que es Java.

Java 2 (antes llamado Java 1.2 o JDK 1.2) es la tercera versión importante del lenguaje de programación Java.

No hay cambios conceptuales importantes respecto a Java 1.1 (en Java 1.1 sí los hubo respecto a Java 1.0), sino extensiones y ampliaciones, lo cual hace que a muchos efectos –por ejemplo, para esta introducción- sea casi lo mismo trabajar con Java 1.1 o con Java 1.2.

Los programas desarrollados en Java presentan diversas ventajas frente a los desarrollados en otros lenguajes como C/C++. La ejecución de programas en Java tiene muchas posibilidades: ejecución como aplicación independiente (Stand-alone Application), ejecución como applet, ejecución como servlet, etc. Un applet es una aplicación especial que se ejecuta dentro de un navegador o browser (por ejemplo Netscape Navigator o Internet Explorer) al cargar una página HTML desde un servidor Web. El applet se descarga desde el servidor y no requiere instalación en el ordenador donde se encuentra el browser. Un servlet es una aplicación sin interface gráfica que se ejecuta en un servidor de Internet. La ejecución como aplicación independiente es análoga a los programas desarrollados con otros lenguajes.

Además de incorporar la ejecución como Applet, Java permite fácilmente el desarrollo tanto de arquitecturas cliente-servidor como de aplicaciones distribuidas, consistentes en crear aplicaciones capaces de conectarse a otros ordenadores y ejecutar tareas en varios ordenadores simultáneamente, repartiendo por lo tanto el trabajo. Aunque también otros lenguajes de programación permiten crear aplicaciones de este tipo, Java incorpora en su propio API estas funcionalidades.

El entorno de desarrollo de Java

Existen distintos programas comerciales que permiten desarrollar código Java. La compañía Sun, creadora de Java, distribuye gratuitamente el Java(tm) Development Kit (JDK). Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en Java. Incorpora además la posibilidad de ejecutar parcialmente el programa, deteniendo la ejecución en el punto deseado y estudiando en cada momento el valor de cada una de las variables (con el denominado Debugger). Cualquier programador con un mínimo de experiencia sabe que una parte muy importante (muchas veces la mayor parte) del tiempo destinado a la elaboración de un programa se destina a la detección y corrección de errores. Existe también una versión reducida del JDK, denominada JRE (Java Runtime Environment) destinada únicamente a ejecutar código Java (no permite compilar).

Los IDEs (Integrated Development Environment), tal y como su nombre indica, son entornos de desarrollo integrados. En un mismo programa es posible escribir el código Java, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para realizar Debug gráficamente, frente a la versión que incorpora el JDK basada en la utilización de una consola (denominada habitualmente ventana de comandos de Linux)
Los entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con componentes ya desarrollados, los cuales se incorporan al proyecto o programa. Como inconvenientes se pueden señalar algunos fallos de compatibilidad entre plataformas, y ficheros resultantes de mayor tamaño que los basados en clases estándar.

El compilador de Java

Se trata de una de las herramientas de desarrollo incluidas en el JDK. Realiza un análisis de sintaxis del código escrito en los ficheros fuente de Java (con extensión *.java). Si no encuentra errores en el código genera los ficheros compilados (con extensión *.class). En otro caso muestra la línea o líneas erróneas. En el JDK de Sun dicho compilador se llama javac. Tiene numerosas opciones, algunas de las cuales varían de una versión a otra. Se aconseja consultar la documentación de la versión del JDK utilizada para obtener una información detallada de las distintas posibilidades.

La Java Virtual Machine

Tal y como se ha comentado al comienzo del capítulo, la existencia de distintos tipos de procesadores y ordenadores llevó a los ingenieros de Sun a la conclusión de que era muy importante conseguir un software que no dependiera del tipo de procesador utilizado. Se planteó la necesidad de conseguir un código capaz de ejecutarse en cualquier tipo de máquina. Una vez compilado no debería ser necesaria ninguna modificación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave consistió en desarrollar un código "neutro" el cual estuviera preparado para ser ejecutado sobre una "máquina hipotética o virtual", denominada Java Virtual Machine (JVM). Es esta JVM quien interpreta este código neutro convirtiéndolo a código particular de la CPU utilizada. Se evita tener que realizar un programa diferente para cada CPU o plataforma.

La JVM es el intérprete de Java. Ejecuta los "bytecodes" (ficheros compilados con extensión *.class) creados por el compilador de Java (javac). Tiene numerosas opciones entre las que destaca la posibilidad de utilizar el denominado JIT (Just-In-Time Compiler), que puede mejorar entre 10 y 20 veces la velocidad de ejecución de un programa.

Las variables PATH y CLASSPATH

El desarrollo y ejecución de aplicaciones en Java exige que las herramientas para compilar (javac) y ejecutar (java) se encuentren accesibles. El ordenador, desde una ventana de comandos de Linux, sólo es capaz de ejecutar los programas que se encuentran en los directorios indicados en la variable PATH del ordenador (o en el directorio activo). Si se desea compilar o ejecutar código en Java, el directorio donde se encuentran estos programas (java y javac) deberá encontrarse en el PATH. Tecleando $PATH en una ventana de comandos de Linux se muestran los nombres de directorios incluidos en dicha variable de entorno.

Java utiliza además una nueva variable de entorno denominada CLASSPATH, la cual determina dónde buscar tanto las clases o librerías de Java (el API de Java) como otras clases de usuario. A partir de la versión 1.1.4 del JDK no es necesario indicar esta variable, salvo que se desee añadir conjuntos de clases de usuario que no vengan con dicho JDK. La variable CLASSPATH puede incluir la ruta de directorios o ficheros *.zip o *.jar en los que se encuentren los ficheros *.class.

Si no la tienes agregadas en tus variables de entorno, la agregas con:

```
export JAVA_HOME=/opt/java/jdk1.6.0_06/
export PATH=/opt/java/jdk1.6.0_06//bin
```

Los parámetros son:

```
JAVA_HOME=/opt/java/jdk1.6.0_06/
PATH=/opt/java/jdk1.6.0_06//bin
```

CLASSPATH=/opt/java/jdk1.6.0_06/jre/lib/jce.jar

lo cual sería válido en el caso de que el JDK estuviera situado en el directorio /opt/java/jdk1.6.0_06/.

Cuando un fichero filename.java se compila y en ese directorio existe ya un fichero filename.class, se comparan las fechas de los dos ficheros. Si el fichero filename.java es más antiguo que el filename.class no se produce un nuevo fichero filename.class. Esto sólo es válido para ficheros *.class que se corresponden con una clase public.

Características del Lenguaje.

En este capítulo se presentan las características generales de Java como lenguaje de programación algorítmico. En este apartado Java es muy similar a C/C++, lenguajes en los que está inspirado. Se va a intentar ser breve, considerando que el lector ya conoce algunos otros lenguajes de programación y está familiarizado con lo que son variables, bifurcaciones, bucles, etc.

Variables

Una variable es un nombre que contiene un valor que puede cambiar a lo largo del programa. De acuerdo con el tipo de información que contienen, en Java hay dos tipos principales de variables:

1. Variables de tipos primitivos. Están definidas mediante un valor único que puede ser entero, de punto flotante, carácter o booleano. Java permite distinta precición y distintos rangos de valores para estos tipos de variables (char, byte, short, int, long, float, double, boolean). Ejemplos de variables de tipos primitivos podrían ser: 123, 3456754, 3.1415, 12e-09, 'A', True, etc.
2. Variables referencia. Las variables referencia son referencias o nombres de una información más compleja: arrays u objetos de una determinada clase.

Desde el punto de vista del papel o misión en el programa, las variables pueden ser:

1. Variables miembro de una clase: Se definen en una clase, fuera de cualquier método; pueden ser tipos primitivos o referencias.
2. Variables locales: Se definen dentro de un método o más en general dentro de cualquier bloque entre llaves {}. Se crean en el interior del bloque y se destruyen al finalizar dicho bloque. Pueden ser también tipos primitivos o referencias.

Nombres de Variables

Los nombres de variables en Java se pueden crear con mucha libertad. Pueden ser cualquier conjunto de caracteres numéricos y alfanuméricos, sin algunos caracteres especiales utilizados por Java como operadores o separadores (,.+-*/ etc.).

Existe una serie de palabras reservadas las cuales tienen un significado especial para Java y por lo tanto no se pueden utilizar como nombres de variables. Dichas palabras son:

abstract	boolean	break	byte	case	catch
char	class	const*	continue	default	do
double	else	extends	final	finally	float
for	goto*	if	implements	import	instanceof
int	interface	long	native	new	null
package	private	protected	public	return	short
static	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while

(*) son palabras reservadas, pero no se utilizan en la actual implementación del lenguaje Java.

Tipos Primitivos de Variables

Se llaman tipos primitivos de variables de Java a aquellas variables sencillas que contienen los tipos de información más habituales: valores boolean, caracteres y valores numéricos enteros o de punto flotante.

Java dispone de ocho tipos primitivos de variables: un tipo para almacenar valores true y false (boolean); un tipo para almacenar caracteres (char), y 6 tipos para guardar valores numéricos, cuatro tipos para enteros (byte, short, int y long) y dos para valores reales de punto flotante (float y double). Los rangos y la memoria que ocupa cada uno de estos tipos se muestran en la Tabla 2.1.

Tipo de variable	Descripción
Boolean	1 byte. Valores true y false
Char	2 bytes. Unicode. Comprende el código ASCII
Byte	1 byte. Valor entero entre -128 y 127
Short	2 bytes. Valor entero entre -32768 y 32767
Int	4 bytes. Valor entero entre -2.147.483.648 y 2.147.483.647
Long	8 bytes. Valor entre -9.223.372.036.854.775.808 y9.223.372.036.854.775.807
Float	4 bytes (entre 6 y 7 cifras decimales equivalentes). De -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38

Double 8 bytes (unas 15 cifras decimales equivalentes). De -1.79769313486232E308 a -4.94065645841247E-324 y de 4.94065645841247E-324 a 1.79769313486232E308

Tabla 2.1. Tipos primitivos de variables en Java.

Los tipos primitivos de Java tienen algunas características importantes que se resumen a continuación:

1. El tipo boolean no es un valor numérico: sólo admite los valores true o false. El tipo boolean no se identifica con el igual o distinto de cero, como en C/C++. El resultado de la expresión lógica que aparece como condición en un bucle o en una bifurcación debe ser boolean.
2. El tipo char contiene caracteres en código UNICODE (que incluye el código ASCII), y ocupan 16 bits por carácter. Comprende los caracteres de prácticamente todos los idiomas.
3. Los tipos byte, short, int y long son números enteros que pueden ser positivos o negativos, con distintos valores máximos y mínimos. A diferencia de C/C++, en Java no hay enteros unsigned.
4. Los tipos float y double son valores de punto flotante (números reales) con 6-7 y 15 cifras decimales equivalentes, respectivamente.
5. Se utiliza la palabra void para indicar la ausencia de un tipo de variable determinado.
6. A diferencia de C/C++, los tipos de variables en Java están perfectamente definidos en todas y cada una de las posibles plataformas. Por ejemplo, un int ocupa siempre la misma memoria y tiene el mismo rango de valores, en cualquier tipo de ordenador.
7. Existen extensiones de Java 1.2 para aprovechar la arquitectura de los procesadores Intel, que permiten realizar operaciones de punto flotente con una precisión extendida de 80 bits.

Cómo se definen e inicializan las variables

Una variable se define especificando el tipo y el nombre de dicha variable. Estas variables pueden ser tanto de tipos primitivos como referencias a objetos de alguna clase perteneciente al API de Java o generada por el usuario. Si no se especifica un valor en su declaración, las variable primitivas se inicializan a cero (salvo boolean y char, que se inicializan a false y '\0'). Análogamente las variables de tipo referencia son inicializadas por defecto a un valor especial: null.

Es importante distinguir entre la referencia a un objeto y el objeto mismo. Una referencia es una variable que indica dónde está guardado un objeto en la memoria del ordenador (a diferencia de C/C++, Java no permite acceder al valor de la dirección, pues en este lenguaje se han eliminado los punteros). Al declarar una referencia todavía no se encuentra “apuntando” a ningún objeto en particular (salvo que se cree explícitamente un nuevo objeto en la declaración), y por eso se le asigna el valor null. Si se desea que esta referencia apunte a un nuevo objeto es necesario crear el

objeto utilizando el operador new. Este operador reserva en la memoria del ordenador espacio para ese objeto (variables y funciones). También es posible igualar la referencia declarada a otra referencia a un objeto existente previamente.

Un tipo particular de referencias son los arrays o vectores, sean éstos de variables primitivas (por ejemplo, un vector de enteros) o de objetos. En la declaración de una referencia de tipo array hay que incluir los corchetes []. En los siguientes ejemplos aparece cómo crear un vector de 10 números enteros y cómo crear un vector de elementos MyClass. Java garantiza que los elementos del vector son inicializados a null o a cero (según el tipo de dato) en caso de no indicar otro valor.

Ejemplos de declaración e inicialización de variables:

```
int x;                    // Declaración de la variable primitiva x. Se inicializa a 0
int y = 5;                // Declaración de la variable primitiva y. Se inicializa a 5
MyClass unaRef;           // Declaración de una referencia a un objeto MyClass.
                          // Se inicializa a null
unaRef = new MyClass();   // La referencia "apunta" al nuevo objeto creado
                          // Se ha utilizado el constructor por defecto
MyClass segundaRef = unaRef;  // Declaración de una referencia a un objeto MyClass.
                          // Se inicializa al mismo valor que unaRef
int [] vector;            // Declaración de un array. Se inicializa a null
vector = new int[10];     // Vector de 10 enteros, inicializados a 0
double [] v = {1.0, 2.65, 3.1};   // Declaración e inicialización de un vector de 3
                          // elementos con los valores entre llaves
MyClass [] lista=new MyClass[5]; // Se crea un vector de 5 referencias a objetos
                          // Las 5 referencias son inicializadas a null
lista[0] = unaRef;        // Se asigna a lista[0] el mismo valor que unaRef
lista[1] = new MyClass(); // Se asigna a lista[1] la referencia al nuevo objeto
                          // El resto (lista[2]...lista[4] siguen con valor null
```

En el ejemplo mostrado las referencias unaRef, segundaRef y lista[0] actuarán sobre el mismo objeto. Es equivalente utilizar cualquiera de las referencias ya que el objeto al que se refieren es el mismo.

Visibilidad y vida de las variables

Se entiende por visibilidad, ámbito o scope de una variable, la parte de la aplicación donde dicha variable es accesible y por lo tanto puede ser utilizada en una expresión. En Java todas las variables deben estar incluidas en una clase. En general las variables declaradas dentro de unas llaves {}, es

decir dentro de un bloque, son visibles y existen dentro de estas llaves. Por ejemplo las variables declaradas al principio de una función existen mientras se ejecute la función; las variables declaradas dentro de un bloque if no serán válidas al finalizar las sentencias correspondientes a dicho if y las variables miembro de una clase (es decir declaradas entre las llaves {} de la clase pero fuera de cualquier método) son válidas mientras existe el objeto de la clase.

Las variables miembro de una clase declaradas como public son accesibles a través de una referencia a un objeto de dicha clase utilizando el operador punto (.). Las variables miembro declaradas como private no son accesibles directamente desde otras clases. Las funciones miembro de una clase tienen acceso directo a todas las variables miembro de la clase sin necesidad de anteponer el nombre de un objeto de la clase. Sin embargo las funciones miembro de una clase B derivada de otra A, tienen acceso a todas las variables miembro de A declaradas como public o protected, pero no a las declaradas como private. Una clase derivada sólo puede acceder directamente a las variables y funciones miembro de su clase base declaradas como public o protected. Otra característica del lenguaje es que es posible declarar una variable dentro de un bloque con el mismo nombre que una variable miembro, pero no con el nombre de otra variable local que ya existiera. La variable declarada dentro del bloque oculta a la variable miembro en ese bloque. Para acceder a la variable miembro oculta será preciso utilizar el operador this, en la forma this.varname.

Uno de los aspectos más importantes en la programación orientada a objetos (OOP) es la forma en la cual son creados y eliminados los objetos. En Java la forma de crear nuevos objetos es utilizando el operador new. Cuando se utiliza el operador new, la variable de tipo referencia guarda la posición de memoria donde está almacenado este nuevo objeto. Para cada objeto se lleva cuenta de por cuántas variables de tipo referencia es apuntado. La eliminación de los objetos la realiza el programa denominado garbage collector, quien automáticamente libera o borra la memoria ocupada por un objeto cuando no existe ninguna referencia apuntando a ese objeto. Lo anterior significa que aunque una variable de tipo referencia deje de existir, el objeto al cual apunta no es eliminado si hay otras referencias apuntando a ese mismo objeto.

Operadores de Java

Java es un lenguaje rico en operadores, que son casi idénticos a los de C/C++. Estos operadores se describen brevemente en los apartados siguientes.

Operadores aritméticos

Son operadores binarios (requieren siempre dos operandos) que realizan las operaciones aritméticas habituales: suma (+), resta (-), multiplicación (*), división (/) y resto de la división (%).

Operadores de asignación

Los operadores de asignación permiten asignar un valor a una variable. El operador de asignación por excelencia es el operador igual (=). La forma general de las sentencias de asignación con este operador es:

Operador	Utilización	Expresión equivalente
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 – op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

variable = expression;

Tabla 2.2. Otros operadores de asignación.

Java dispone de otros operadores de asignación. Se trata de versiones abreviadas del operador (=) que realizan operaciones “acumulativas” sobre una variable. La Tabla 2.2 muestra estos operadores y su equivalencia con el uso del operador igual (=).

Operadores unarios

Los operadores más (+) y menos (-) unarios sirven para mantener o cambiar el signo de una variable, constante o expresión numérica. Su uso en Java es el estándar de estos operadores.

Operador instanceof

El operador instanceof permite saber si un objeto pertenece o no a una determinada clase. Es un operador binario cuya forma general es,

objectName instanceof ClassName

y que devuelve true o false según el objeto pertenezca o no a la clase.

Operador condicional ?:

Este operador, tomado de C/C++, permite realizar bifurcaciones condicionales sencillas. Su forma general es la siguiente:

booleanExpression ? res1 : res2

donde se evalúa booleanExpression y se devuelve res1 si el resultado es true y res2 si el resultado es false. Es el único operador ternario (tres argumentos) de Java. Como todo operador que devuelve un valor puede ser utilizado en una expresión. Por ejemplo las sentencias:

x=1 ; y=10; z = (x<y)?x+3:y+8;

asignarían a z el valor 4, es decir x+3.

Operadores incrementales

Java dispone del operador incremento (++) y decremento (--). El operador (++) incrementa en una unidad la variable a la que se aplica, mientras que (--) la reduce en una unidad. Estos operadores se pueden utilizar de dos formas:

1. Precediendo a la variable (por ejemplo: ++i). En este caso primero se incrementa la variable y luego se utiliza (ya incrementada) en la expresión en la que aparece.
2. Siguiendo a la variable (por ejemplo: i++). En este caso primero se utiliza la variable en la expresión (con el valor anterior) y luego se incrementa.

En muchas ocasiones estos operadores se utilizan para incrementar una variable fuera de una expresión. En este caso ambos operadores son equivalente. Si se utilizan en una expresión más complicada, el resultado de utilizar estos operadores en una u otra de sus formas será diferente. La actualización de contadores en bucles for es una de las aplicaciones más frecuentes de estos operadores.

Operadores relacionales

Los operadores relacionales sirven para
realizar comparaciones de igualdad,
desigualdad y relación de menor o mayor.
El resultado de estos operadores es

siempre un valor boolean (true o false)
según se cumpla o no la relación
considerada.

La Tabla 2.3 muestra los operadores relacionales de Java.

Operador	Utilización	El resultado es true
>	op1 > op2	si op1 es mayor que op2
>=	op1 >= op2	si op1 es mayor o igual que op2
<	op1 < op2	si op1 es menor que op2
<=	op1 <= op2	si op1 es menor o igual que op2
==	op1 == op2	si op1 y op2 son iguales
!=	op1 != op2	si op1 y op2 son diferentes

Tabla 2.3. Operadores relacionales.
operadores relacionales de Java.

Estos operadores se utilizan con mucha frecuencia en las bifurcaciones y en los bucles, que se verán en próximos apartados de este capítulo.

Operadores lógicos

Los operadores lógicos se utilizan para construir expresiones lógicas, combinando valores lógicos (true y/o false) o los resultados de los operadores relacionales. La Tabla 2.4 muestra los operadores lógicos de Java. Debe notarse que en ciertos casos el segundo operando no se evalúa porque ya no es necesario (si ambos tienen que ser true y el primero es false, ya se sabe que la condición de que ambos sean true no se va a cumplir). Esto puede traer resultados no deseados y por eso se han añadido los operadores (&) y (|) que garantizan que los dos operandos se evalúan siempre.

Operador	Nombre	Utilización	Resultado
&&	AND	op1 && op2	true si op1 y op2 son true. Si op1 es false ya no se evalúa op2
||	OR	op1 || op2	true si op1 u op2 son true. Si op1 es true ya no se evalúa op2
!	negación	! op	true si op es false y false si op es true
&	AND	op1 & op2	true si op1 y op2 son true. Siempre se evalúa op2
|	OR	op1 | op2	true si op1 u op2 son true. Siempre se evalúa op2

Tabla 2.4. Operadores lógicos.

Operador de concatenación de cadenas de caracteres (+)

El operador más (+) se utiliza también para concatenar cadenas de caracteres. Por ejemplo, para escribir una cantidad con un rótulo y unas unidades puede utilizarse la sentencia:

```
System.out.println("El total asciende a " + result + " unidades");
```

donde el operador de concatenación se utiliza dos veces para construir la cadena de caracteres que se desea imprimir por medio del método println(). La variable numérica result es convertida automáticamente por Java en cadena de caracteres para poderla concatenar. En otras ocasiones se

deberá llamar explícitamente a un método para que realice esta conversión.

Operadores que actúan a nivel de bits

Java dispone también de un conjunto de operadores que actúan a nivel de bits. Las operaciones de bits se utilizan con frecuencia para definir señales o flags, esto es, variables de tipo entero en las que cada uno de sus bits indican si una opción está activada o no. La Tabla 2.5 muestra los operadores de Java que actúan a nivel de bits.

Operador	Utilización	Resultado
>>	op1 >> op2	Desplaza los bits de op1 a la derecha una distancia op2
<<	op1 << op2	Desplaza los bits de op1 a la izquierda una distancia op2
>>>	op1 >>> op2	Desplaza los bits de op1 a la derecha una distancia op2 (positiva)
&	op1 & op2	Operador AND a nivel de bits
|	op1 | op2	Operador OR a nivel de bits
^	op1 ^ op2	Operador XOR a nivel de bits (1 si sólo uno de los operandos es 1)
~	~op2	Operador complemento (invierte el valor de cada bit)

Tabla 2.5. Operadores a nivel de bits.

En binario, las potencias de dos se representan con un único bit activado. Por ejemplo, los números (1, 2, 4, 8, 16, 32, 64, 128) se representan respectivamente de modo binario en la forma (00000001, 00000010, 00000100, 00001000, 00010000, 00100000, 01000000, 10000000), utilizando sólo 8 bits. La suma de estos números permite construir una variable flags con los bits activados que se deseen. Por ejemplo, para construir una variable flags que sea 00010010 bastaría hacer flags=2+16. Para saber si el segundo bit por la derecha está o no activado bastaría utilizar la sentencia,

```
if (flags & 2 == 2) {...}
```

La Tabla 2.6 muestra los operadores de asignación a nivel de bits.

Operador	Utilización	Equivalente a
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2
^=	op1 ^= op2	op1 = op1 ^ op2
<<=	op1 <<= op2	op1 = op1 << op2
>>=	op1 >>= op2	op1 = op1 >> op2
>>>=	op1 >>>= op2	op1 = op1 >>> op2

Tabla 2.6. Operadores de asignación a nivel de bits.

Precedencia de operadores

El orden en que se realizan las operaciones es fundamental para determinar el resultado de una expresión. Por ejemplo, el resultado de x/y*z depende de qué operación (la división o el producto) se realice primero. La siguiente lista muestra el orden en que se ejecutan los distintos operadores en un sentencia, de mayor a menor precedencia:

postfix operators	[] . (params) expr++ expr--
unary operators	++expr --expr +expr -expr ~ !
creation or cast	new (type)expr
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
conditional	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

En Java, todos los operadores binarios, excepto los operadores de asignación, se evalúan de izquierda a derecha. Los operadores de asignación se evalúan de derecha a izquierda, lo que significa que el valor de la derecha se copia sobre la variable de la izquierda.

Estructuras de programación

En este apartado se supone que el lector tiene algunos conocimientos de programación y por lo tanto
no se explican en profundidad los conceptos que aparecen.

Las estructuras de programación o estructuras de control permiten tomar decisiones y realizar un proceso repetidas veces. Son los denominados bifurcaciones y bucles. En la mayoría de los lenguajes de programación, este tipo de estructuras son comunes en cuanto a concepto, aunque su sintaxis varía de un lenguaje a otro. La sintaxis de Java coincide prácticamente con la utilizada en C/C++, lo que hace que para un programador de C/C++ no suponga ninguna dificultad adicional.

Sentencias o expresiones

Una expresión es un conjunto variables unidos por operadores. Son órdenes que se le dan al computador para que realice una tarea determinada.

Una sentencia es una expresión que acaba en punto y coma (;). Se permite incluir varias sentencias en una línea, aunque lo habitual es utilizar una línea para cada sentencia. Por ejemplo:

```
i = 0; j = 5; x = i + j;// Línea compuesta de tres sentencias
```

Comentarios

Existen dos formas diferentes de introducir comentarios entre el código de Java (en realidad son tres, como pronto se verá). Son similares a la forma de realizar comentarios en el lenguaje C++. Los comentarios son tremendamente útiles para poder entender el código utilizado, facilitando de ese modo futuras revisiones y correcciones. Además permite que cualquier persona distinta al programador original pueda comprender el código escrito de una forma más rápida. Se recomienda acostumbrarse a comentar el código desarrollado. De esta forma se simplifica también la tarea de estudio y revisión posteriores.

Java interpreta que todo lo que aparece a la derecha de dos barras “//” en una línea cualquiera del código es un comentario del programador y no lo tiene en cuenta. El comentario puede empezar al comienzo de la línea o a continuación de una instrucción que debe ser ejecutada. La segunda forma de incluir comentarios consiste en escribir el texto entre los símbolos /*...*/. Este segundo método es válido para comentar más de una línea de código. Por ejemplo:

```
// Esta línea es un comentario
int a=1; // Comentario a la derecha de una sentencia
// Esta es la forma de comentar más de una línea utilizando
```

```
// las dos barras. Requiere incluir dos barras al comienzo de cada línea
/* Esta segunda forma es mucho más cómoda para comentar un número elevado de líneas
   ya que sólo requiere modificar
   el comienzo y el final. */
```

En Java existe además una forma especial de introducir los comentarios (utilizando /**...*/ más algunos caracteres especiales) que permite generar automáticamente la documentación sobre las clases y packages desarrollados por el programador. Una vez introducidos los comentarios, el programa javadoc.exe (incluido en el JDK) genera de forma automática la información de forma similar a la presentada en la propia documentación del JDK. La sintaxis de estos comentarios y la forma de utilizar el programa javadoc.exe se puede encontrar en la información que viene con el JDK.

Bifurcaciones

Las bifurcaciones permiten ejecutar una de entre varias acciones en función del valor de una expresión lógica o relacional. Se tratan de estructuras muy importantes ya que son las encargadas de controlar el flujo de ejecución de un programa. Existen dos bifurcaciones diferentes: if y switch.

Bifurcación if

Esta estructura permite ejecutar un conjunto de sentencias en función del valor que tenga la expresión de comparación (se ejecuta si la expresión de comparación tiene valor true). Tiene la forma siguiente:

```
if (booleanExpression) {
    statements;
}
```

Las llaves {} sirven para agrupar en un bloque las sentencias que se han de ejecutar, y no son necesarias si sólo hay una sentencia dentro del if.

Bifurcación if else

Análoga a la anterior, de la cual es una ampliación. Las sentencias incluidas en el else se ejecutan en
el caso de no cumplirse la expresión de comparación (false),

```
if (booleanExpression) {
```

```
    statements1;
} else {
    statements2;
}
```

Bifurcación if elseif else

Permite introducir más de una expresión de comparación. Si la primera condición no se cumple, se compara la segunda y así sucesivamente. En el caso de que no se cumpla ninguna de las comparaciones se ejecutan las sentencias correspondientes al else.

```
if (booleanExpression1) {
    statements1;
} else if (booleanExpression2) {
    statements2;
} else if (booleanExpression3) {
    statements3;
} else {
    statements4;
}
```

Véase a continuación el siguiente ejemplo:

```
int numero = 61;                        // La variable "numero" tiene dos dígitos
if(Math.abs(numero) < 10)                   // Math.abs() calcula el valor absoluto. (false)
    System.out.println("Numero tiene 1 digito ");
else if (Math.abs(numero) < 100) // Si numero es 61, estamos en este caso (true)
    System.out.println("Numero tiene 1 digito ");
else {                                  // Resto de los casos
    System.out.println("Numero tiene mas de 3 digitos ");
    System.out.println("Se ha ejecutado la opcion por defecto ");
}
```

Sentencia switch

Se trata de una alternativa a la bifurcación if elseif else cuando se compara la misma expresión con distintos valores. Su forma general es la siguiente:

```
switch (expression) {
```

```
    case value1: statements1;          break;
    case value2: statements2;          break;
    case value3: statements3;          break;
    case value4: statements4;          break;
    case value5: statements5;          break;
    case value6: statements6;          break;
    [default: statements7;]
}
```

Las características más relevantes de switch son las siguientes:

1. Cada sentencia case se corresponde con un único valor de expression. No se pueden establecer rangos o condiciones sino que se debe comparar con valores concretos. El ejemplo del Apartado 2.3.3.3 no se podría realizar utilizando switch.
2. Los valores no comprendidos en ninguna sentencia case se pueden gestionar en default, que es opcional.
3. En ausencia de break, cuando se ejecuta una sentencia case se ejecutan también todas las case que van a continuación, hasta que se llega a un break o hasta que se termina el switch.

Ejemplo:

```
char c = (char)(Math.random()*26+'a'); // Generación aleatoria de letras minúsculas
System.out.println("La letra " + c );
switch (c) {
   case 'a': // Se compara con la letra a
   case 'e': // Se compara con la letra e
   case 'i': // Se compara con la letra i
   case 'o': // Se compara con la letra o
   case 'u': // Se compara con la letra u
       System.out.println(" Es una vocal "); break;
   default:
       System.out.println(" Es una consonante ");
}
```

Bucles

Un bucle se utiliza para realizar un proceso repetidas veces. Se denomina también lazo o loop. El código incluido entre las llaves {} (opcionales si el proceso repetitivo consta de una sola línea), se ejecutará mientras se cumpla unas determinadas condiciones. Hay que prestar especial atención a

los bucles infinitos, hecho que ocurre cuando la condición de finalizar el bucle (booleanExpression) no se llega a cumplir nunca. Se trata de un fallo muy típico, habitual sobre todo entre programadores poco experimentados.

Bucle while

Las sentencias statements se ejecutan mientras booleanExpression sea true.

```
while (booleanExpression) {
    statements;
}
```

Bucle for

La forma general del bucle for es la siguiente:

```
for (initialization; booleanExpression; increment) {
    statements;
}
```

que es equivalente a utilizar while en la siguiente forma,

```
initialization;
while (booleanExpression) {
    statements;
    increment;
}
```

La sentencia o sentencias initialization se ejecuta al comienzo del for, e increment después de statements. La booleanExpression se evalúa al comienzo de cada iteración; el bucle termina cuando la expresión de comparación toma el valor false. Cualquiera de las tres partes puede estar vacía. La initialization y el increment pueden tener varias expresiones separadas por comas.

Por ejemplo, el código situado a la izquierda produce la salida que aparece a la derecha:

Código:

```
for(int i = 1, j = i + 10; i < 5; i++, j = 2*i) {
    System.out.println(" i = " + i + " j = " + j);
}
```

Salida:

```
i = 1  j = 11
i = 2  j = 4
i = 3  j = 6
i = 4  j = 8
```

Bucle do while

Es similar al bucle while pero con la particularidad de que el control está al final del bucle (lo que hace que el bucle se ejecute al menos una vez, independientemente de que la condición se cumpla o no). Una vez ejecutados los statements, se evalúa la condición: si resulta true se vuelven a ejecutar las sentencias incluidas en el bucle, mientras que si la condición se evalúa a false finaliza el bucle. Este tipo de bucles se utiliza con frecuencia para controlar la satisfacción de una determinada condición de error o de convergencia.

```
do {
    statements
} while (booleanExpression);
```

Sentencias break y continue

La sentencia break es válida tanto para las bifurcaciones como para los bucles. Hace que se salga inmediatamente del bucle o bloque que se está ejecutando, sin sin realizar la ejecución del resto de las sentencias.

La sentencia continue se utiliza en los bucles (no en bifurcaciones). Finaliza la iteración “i” que en ese momento se está ejecutando (no ejecuta el resto de sentencias que hubiera hasta el final del bucle). Vuelve al comienzo del bucle y comienza la siguiente iteración (i+1).

Sentencias break y continue con etiquetas

Las etiquetas permiten indicar un lugar donde continuar la ejecución de un programa después de un break o continue. El único lugar donde se pueden incluir etiquetas es justo delante de un bloque de código entre llaves {} (if, switch, do...while, while, for) y sólo se deben utilizar cuando se tiene uno o más bucles (o bloques) dentro de otro bucle y se desea salir (break) o continuar con la siguiente iteración (continue) de un bucle que no es el actual.

Por tanto, la sentencia break labelName finaliza el bloque que se encuentre a continuación de labelName. Por ejemplo, en las sentencias,

```
bucleI:       // etiqueta o label
for ( int i = 0, j = 0; i < 100; i++){
    while ( true ) {
        if( (++j) > 5) { break bucleI; }                    // Finaliza ambos bucles
        else { break; }                                     // Finaliza el bucle interior (while)
```

```
        }
    }
```

la expresión break bucleI; finaliza los dos bucles simultáneamente, mientras que la expresión break; sale del bucle while interior y seguiría con el bucle for en i. Con los valores presentados ambos bucles finalizarán con i = 5 y j = 6 (se invita al lector a comprobarlo).

La sentencia continue (siempre dentro de al menos un bucle) permite transferir el control a un bucle con nombre o etiqueta. Por ejemplo, la sentencia,

```
continue bucle1;
```

transfiere el control al bucle for que comienza después de la etiqueta bucle1: para que realice una nueva iteración, como por ejemplo:

```
bucle1:
for (int i=0; i<n; i++) {
    bucle2:
    for (int j=0; j<m; j++) {
        ...
        if (expression) continue bucle1; then continue bucle2;
        ...
    }
}
```

Sentencia return

Otra forma de salir de un bucle (y de un método) es utilizar la sentencia return. A diferencia de continue o break, la sentencia return sale también del método o función. En el caso de que la función devuelva alguna variable, este valor se deberá poner a continuación del return (return value;).

Bloque try {...} catch {...} finally {...}

Java incorpora en el propio lenguaje la gestión de errores. El mejor momento para detectar los errores es durante la compilación. Sin embargo prácticamente sólo los errores de sintaxis son detectados en esta operación. El resto de problemas surgen durante la ejecución de los programas.

En el lenguaje Java, una Exception es un cierto tipo de error o una condición anormal que se ha producido durante la ejecución de un programa. Algunas excepciones son fatales y provocan que se deba finalizar la ejecución del programa. En este caso conviene terminar ordenadamente y dar un

mensaje explicando el tipo de error que se ha producido. Otras excepciones, como por ejemplo no encontrar un fichero en el que hay que leer o escribir algo, pueden ser recuperables. En este caso el programa debe dar al usuario la oportunidad de corregir el error (definiendo por ejemplo un nuevo path del fichero no encontrado).

Los errores se representan mediante clases derivadas de la clase Throwable, pero los que tiene
que chequear un programador derivan de Exception (java.lang.Exception que a su vez deriva de Throwable). Existen algunos tipos de excepciones que Java obliga a tener en cuenta. Esto se hace mediante el uso de bloques try, catch y finally.

El código dentro del bloque try está “vigilado”. Si se produce una situación anormal y se lanza
como consecuencia una excepción, el control pasa al bloque catch, que se hace cargo de la situación y decide lo que hay que hacer. Se pueden incluir tantos bloques catch como se desee, cada uno de los cuales tratará un tipo de excepción. Finalmente, si está presente, se ejecuta el bloque finally, que es opcional, pero que en caso de existir se ejecuta siempre, sea cual sea el tipo de error.

En el caso en que el código de un método pueda generar una Exception y no se desee incluir en dicho método la gestión del error (es decir los bucles try/catch correspondientes), es necesario que el método pase la Exception al método desde el que ha sido llamado. Esto se consigue mediante la adición de la palabra throws seguida del nombre de la Exception concreta, después de la lista de argumentos del método. A su vez el método superior deberá incluir los bloques try/catch o volver a pasar la Exception. De esta forma se puede ir pasando la Exception de un método a otro hasta llegar al último método del programa, el método main().

En el siguiente ejemplo se presentan dos métodos que deben "controlar" una IOException relacionada con la lectura ficheros y una MyException propia. El primero de ellos (metodo1) realiza la gestión de las excepciones y el segundo (metodo2) las pasa al siguiente método.

```
void metodo1() {
    ...
    try {
        ... // Código que puede lanzar las excepciones IOException y MyException
    } catch (IOException e1) {// Se ocupa de IOException simplemente dando aviso
        System.out.println(e1.getMessage());
    } catch (MyException e2) {
        // Se ocupa de MyException dando un aviso y finalizando la función
        System.out.println(e2.getMessage()); return;
    } finally { // Sentencias que se ejecutarán en cualquier caso
        ...
```

```
    }]
    ...
} // Fin del metodo1
void metodo2() throws IOException, MyException {
    ...
    // Código que puede lanzar las excepciones IOException y MyException
    ...
} // Fin del metodo2
```

Arrays y Cadenas.

Arrays Unidimensionales.

Un array es una colección de valores de un mismo tipo engrosados en la misma variable. De forma que se puede acceder a cada valor independientemente. Para Java además un array es un objeto que tiene propiedades que se pueden manipular.

Los arrays solucionan problemas concernientes al manejo de muchas variables que se refieren a datos similares. Por ejemplo si tuviéramos la necesidad de almacenar las notas de una clase con 18 alumnos, necesitaríamos 18 variables, con la tremenda lentitud de manejo que supone eso. Solamente calcular la nota media requeriría una tremenda línea de código. Almacenar las notas supondría al menos 18 líneas de código.

Gracias a los arrays se puede crear un conjunto de variables con el mismo nombre. La diferencia será que un número (índice del array) distinguirá a cada variable.

En el caso de las notas, se puede crear un array llamado notas, que representa a todas las notas de la clase. Para poner la nota del primer alumno se usaría notas[0], el segundo sería notas[1], etc. (los corchetes permiten especificar el índice en concreto del array).

La declaración de un array unidimensional se hace con esta sintaxis.

```
tipo nombre[];
```

Ejemplo:

```
double cuentas[]; //Declara un array que almacenará valores
                  // doubles
```

Declara un array de tipo double. Esta declaración indica para qué servirá el array, pero no reserva espacio en la RAM al no saberse todavía el tamaño del mismo.

Tras la declaración del array, se tiene que iniciar. Eso lo realiza el operador new,

que es el que realmente crea el array indicando un tamaño. Cuando se usa new es cuando se reserva el espacio necesario en memoria. Un array no inicializado es un array null. Ejemplo:

```
int notas[]; //sería válido también int[] notas;
notas = new int[3]; //indica que el array constará de tres
                    //valores de tipo int
//También se puede hacer todo a la vez
//int notas[]=new int[3];
```

En el ejemplo anterior se crea un array de tres enteros (con los tipos básicos se crea en memoria el array y se inicializan los valores, los números se inician a 0).

Los valores del array se asignan utilizando el índice del mismo entre corchetes:

```
notas[2]=8;
```

También se pueden asignar valores al array en la propia declaración:

```
int notas[] = {8, 7, 9};
int notas2[]= new int[] {8,7,9};//Equivalente a la anterior
```

Esto declara e inicializa un array de tres elementos. En el ejemplo lo que significa es que notas[0] vale 8, notas[1] vale 7 y notas[2] vale 9.

En Java (como en otros lenguajes) el primer elemento de un array es el cero. El primer elemento del array notas, es notas[0]. Se pueden declarar arrays a cualquier tipo de datos (enteros, booleanos, doubles, ... e incluso objetos).

La ventaja de usar arrays (volviendo al caso de las notas) es que gracias a un simple bucle for se puede rellenar o leer fácilmente todos los elementos de un array:

```
//Calcular la media de las 18 notas
suma=0;
for (int i=0;i<=17;i++){
  suma+=nota[i];
}
media=suma/18;
```

A un array se le puede inicializar las veces que haga falta:

```
int notas[]=new notas[16];
...
notas=new notas[25];
```

Pero hay que tener en cuenta que el segundo new hace que se pierda el contenido anterior. Realmente un array es una referencia a valores que se almacenan en memoria

mediante el operador new, si el operador new se utiliza en la misma referencia, el anterior contenido se queda sin referencia y, por lo tanto se pierde.

Un array se puede asignar a otro array (si son del mismo tipo):

```
int notas[];
int ejemplo[]=new int[18];
notas=ejemplo;
```

En el último punto, notas equivale a ejemplo. Esta asignación provoca que cualquier cambio en notas también cambie el array ejemplos. Es decir esta asignación anterior, no copia los valores del array, sino que notas y ejemplo son referencias al mismo array.
Ejemplo:

```
int notas[]={3,3,3};
int ejemplo[]=notas;
ejemplo= notas;
ejemplo[0]=8;
System.out.println(notas[0]);//Escribirá el número 8
```

Arrays Multidimensionales.

Los arrays además pueden tener varias dimensiones. Entonces se habla de arrays de arrays (arrays que contienen arrays) Ejemplo:

```
int notas[][];
```

notas es un array que contiene arrays de enteros

```
notas = new int[3][12];//notas está compuesto por 3 arrays
                       //de 12 enteros cada uno
notas[0][0]=9;//el primer valor es 0
```

Puede haber más dimensiones incluso (notas[3][2][7]). Los arrays multidimensionales se pueden inicializar de forma más creativa incluso. Ejemplo:

```
int notas[][]=new int[5][];//Hay 5 arrays de enteros
notas[0]=new int[100]; //El primer array es de 100 enteros
notas[1]=new int[230]; //El segundo de 230
notas[2]=new int[400];
notas[3]=new int[100];
notas[4]=new int[200];
```

Hay que tener en cuenta que en el ejemplo anterior, notas[0] es un array de 100 enteros. Mientras que notas, es un array de 5 arrays de enteros.

Se pueden utilizar más de dos dimensiones si es necesario.

longitud de un array

Los arrays poseen un método que permite determinar cuánto mide un array. Se trata de length. Ejemplo (continuando del anterior):

```
System.out.println(notas.length); //Sale 5
System.out.println(notas[2].length); //Sale 400
```

la clase Arrays.

En el paquete java.utils se encuentra una clase estática llamada Arrays. Una clase estática permite ser utilizada como si fuera un objeto (como ocurre con Math). Esta clase posee métodos muy interesantes para utilizar sobre arrays.

Su uso es

```
Arrays.método(argumentos);
```

fill

Permite rellenar todo un array unidimensional con un determinado valor. Sus argumentos son el array a rellenar y el valor deseado:

```
int valores[]=new int[23];
Arrays.fill(valores,-1);//Todo el array vale -1
```

También permite decidir desde que índice hasta qué índice rellenamos:

```
Arrays.fill(valores,5,8,-1);//Del elemento 5 al 7 valdrán -1
```

equals

Compara dos arrays y devuelve true si son iguales. Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores.

sort

Permite ordenar un array en orden ascendente. Se pueden ordenar sólo una serie de elementos desde un determinado punto hasta un determinado punto.

```
int x[]={4,5,2,3,7,8,2,3,9,5};
Arrays.sort(x);//Estará ordenado
Arrays.sort(x,2,5);//Ordena del 2o al 4o elemento
```

binarySearch

Permite buscar un elemento de forma ultrarrápida en un array ordenado (en un array desordenado sus resultados son impredecibles). Devuelve el índice en el que está colocado el elemento. Ejemplo:

```
int x[]={1,2,3,4,5,6,7,8,9,10,11,12};
Arrays.sort(x);
System.out.println(Arrays.binarySearch(x,8));//Da
    7
```

El método System.arraysCopy

La clase System también posee un método relacionado con los arrays, dicho método permite copiar un array en otro. Recibe cinco argumentos: el array que se copia, el índice desde que se empieza a copia en el origen, el array destino de la copia, el índice desde el que se copia en el destino, y el tamaño de la copia (número de elementos de la copia).

```
int uno[]={1,1,2};
int dos[]={3,3,3,3,3,3,3,3,3};
System.arraycopy(uno, 0, dos, 0, uno.length);
for (int i=0;i<=8;i++){
    System.out.print(dos[i]+" ");
} //Sale 112333333
```

Clase String

Introducción

Para Java las cadenas de texto son objetos especiales. Los textos deben manejarse creando objetos de tipo String. Ejemplo:

```
String texto1 = “¡Prueba de texto!”;
```

Las cadenas pueden ocupar varias líneas utilizando el operador de concatenación “+”.

```
String texto2 =”Este es un texto que ocupa “ +
            “varias líneas, no obstante se puede “+
            “perfectamente encadenar”;
```

También se pueden crear objetos String sin utilizar constantes entrecomilladas, usando otros constructores:

```
char[]    palabra = {‘P’,’a’,’l’,’b’,’r’,’a’};//Array de char
String    cadena = new String(palabra);
byte[]    datos = {97,98,99};
String    codificada = new String (datos, “8859_1”);
```

En el último ejemplo la cadena codificada se crea desde un array de tipo byte que contiene números que serán interpretados como códigos Unicode. Al asignar, el valor 8859_1 indica la tabla de códigos a utilizar.

comparación entre objetos String

Los objetos String no pueden compararse directamente con los operadores de comparación. En su lugar se deben utilizar estas expresiones:

cadena1.equals(cadena2). El resultado es true si la cadena1 es igual a la cadena2. Ambas cadenas son variables de tipo String.

cadena1.equalsIgnoreCase(cadena2). Como la anterior, pero en este caso no se tienen en cuenta mayúsculas y minúsculas.

s1.compareTo(s2). Compara ambas cadenas, considerando el orden alfabético. Si la primera cadena es mayor en orden alfabético que la segunda devuelve 1, si son iguales devuelve 0 y si es la segunda la mayor devuelve -1. Hay que tener en cuenta que el orden no es el del alfabeto español, sino que usa la tabla ASCII, en esa tabla la letra ñ es mucho mayor que la o.

s1.compareToIgnoreCase(s2). Igual que la anterior, sólo que además ignora las mayúsculas (disponible desde Java 1.2)

String.valueOf

Este método pertenece no sólo a la clase String, sino a otras y siempre es un método que convierte valores de una clase a otra. En el caso de los objetos String, permite convertir valores que no son de cadena a forma de cadena. Ejemplos:

```
String numero = String.valueOf(1234);
String fecha = String.valueOf(new Date());
```

En el ejemplo se observa que este método pertenece a la clase String directamente, no hay que utilizar el nombre del objeto creado (como se verá más adelante, es un método estático).

Métodos de las variables de las cadenas

Son métodos que poseen las propias variables de cadena. Para utilizarlos basta con poner el nombre del método y sus parámetros después del nombre de la variable String. Es decir: variableString.método(argumentos)

length

Permite devolver la longitud de una cadena (el número de caracteres de la cadena):

```
String texto1=”Prueba”;
System.out.println(texto1.length());//Escribe 6
```

Concatenar cadenas

Se puede hacer de dos formas, utilizando el método concat o con el operador +. Ejemplo:

```
String s1=”Buenos ”, s2=”días”, s3, s4;
s3 = s1 + s2;
s4 = s1.concat(s2);
```

charAt

Devuelve un carácter de la cadena. El carácter a devolver se indica por su posición (el primer carácter es la posición 0) Si la posición es negativa o sobrepasa el tamaño de la cadena, ocurre un error de ejecución, una excepción tipo IndexOutOfBounds-Exception. Ejemplo:

```
String s1=”Prueba”;
char c1=s1.charAt(2); //c1 valdrá ‘u’
```

substring

Da como resultado una porción del texto de la cadena. La porción se toma desde una posición inicial hasta una posición final (sin incluir esa posición final). Si las posiciones indicadas no son válidas ocurre una excepción de tipo IndexOutOfBounds-Exception. Se empieza a contar desde la posición 0. Ejemplo:

```
String s1=”Buenos días”;

String s2=s1.substring(7,10); //s2 = día
```

indexOf

Devuelve la primera posición en la que aparece un determinado texto en la cadena. En el caso de que la cadena buscada no se encuentre, devuelve -1. El texto a buscar puede ser char o String. Ejemplo:

```
String s1=”Quería decirte que quiero que te vayas”;
System.out.println(s1.indexOf(“que”)); //Da 15
```

Se puede buscar desde una determinada posición. En el ejemplo anterior:

```
System.out.println(s1.indexOf(“que”,16)); //Ahora da 26
```

lastIndexOf

Devuelve la última posición en la que aparece un determinado texto en la cadena. Es casi idéntica a la anterior, sólo que busca desde el final. Ejemplo:

```
String s1=”Quería decirte que quiero que te vayas”;
System.out.println(s1.lastIndexOf(“que”); //Da 26
```

También permite comenzar a buscar desde una determinada posición.

endsWith

Devuelve true si la cadena termina con un determinado texto. Ejemplo:

```
String s1=”Quería decirte que quiero que te vayas”;
System.out.println(s1.endsWith(“vayas”); //Da true
```

startsWith

Devuelve true si la cadena empieza con un determinado texto.

replace

Cambia todas las apariciones de un carácter por otro en el texto que se indique y lo almacena como resultado. El texto original no se cambia, por lo que hay que asignar el resultado de replace a un String para almacenar el texto cambiado:

```
String s1=”Mariposa”;
System.out.println(s1.replace(‘a’,’e’));//Da Meripose
System.out.println(s1);//Sigue valiendo Mariposa
```

replaceAll

Modifica en un texto cada entrada de una cadena por otra y devuelve el resultado. El primer parámetro es el texto que se busca (que puede ser una expresión regular), el segundo parámetro es el texto con el que se reemplaza el buscado. La cadena original no se modifica.

```
String s1=”Cazar armadillos”;
System.out.println(s1.replace(“ar”,”er”));//Da Cazer ermedillos
System.out.println(s1);//Sigue valiendo Cazar armadilos
```

toUpperCase

Devuelve la versión en mayúsculas de la cadena.

toLowerCase

Devuelve la versión en minúsculas de la cadena.

toCharArray

Obtiene un array de caracteres a partir de una cadena.

Lista completa de métodos

método	descripción
char charAt(int index)	Proporciona el carácter que está en la posición dada por el entero index.
int compareTo(string s)	Compara las dos cadenas. Devuelve un valor menor que cero si la cadena s es mayor que la original, devuelve 0 si son iguales y devuelve un valor mayor que cero si s es menor que la original.
int compareToIgnoreCase(string s)	Compara dos cadenas, pero no tiene e cuenta si el texto es mayúsculas o no.
String concat(String s)	Añade la cadena s a la cadena original.

String copyValueOf(char[] data)	Produce un objeto String que es igual al array de caracteres data.
boolean endsWith(String s)	Devuelve true si la cadena termina con el texto s
boolean equals(String s)	Compara ambas cadenas, devuelve true si son iguales
boolean equalsIgnoreCase(String s)	Compara ambas cadenas sin tener en cuenta las mayúsculas y las minúsculas.
byte[] getBytes()	Devuelve un array de caracteres que toma a partir de la cadena de texto
void getBytes(int srcBegin, int srcEnd, char[] dest, int dstBegin);	Almacena el contenido de la cadena en el array de caracteres dest. Toma los caracteres desde la posición srcBegin hasta la posición srcEnd y les copia en el array desde la posición dstBegin
int indexOf(String s)	Devuelve la posición en la cadena del texto s
int indexOf(String s, int primeraPos)	Devuelve la posición en la cadena del texto s, empezando a buscar desde la posición PrimeraPos
int lastIndexOf(String s)	Devuelve la última posición en la cadena del texto s

método	descripción
int lastIndexOf(String s, int primeraPos)	Devuelve la última posición en la cadena del texto s, empezando a buscar desde la posición PrimeraPos
int length()	Devuelve la longitud de la cadena

String replace(char carAnterior, char ncarNuevo)	Devuelve una cadena idéntica al original pero que ha cambiando los caracteres iguales a carAnterior por carNuevo
String replaceFirst(String str1, String str2)	Cambia la primera aparición de la cadena str1 por la cadena str2
String replaceFirst(String str1, String str2)	Cambia la primera aparición de la cadena uno por la cadena dos
String replaceAll(String str1, String str2)	Cambia la todas las apariciones de la cadena uno por la cadena dos
String startsWith(String s)	Devuelve true si la cadena comienza con el texto s.
String substring(int primeraPos, int segundaPos)	Devuelve el texto que va desde primeraPos a segunaPos.
char[] toCharArray()	Devuelve un array de caracteres a partir de la cadena dada
String toLowerCase()	Convierte la cadena a minúsculas
String toLowerCase(Locale local)	Lo mismo pero siguiendo las instrucciones del argumento local
String toUpperCase()	Convierte la cadena a mayúsculas
String toUpperCase(Locale local)	Lo mismo pero siguiendo las instrucciones del argumento local
String trim()	Elimina los blancos que tenga la cadena tanto por delante como por detrás

Static String valueOf(tipo elemento)	Devuelve la cadena que representa el valor elemento. Si elemento es booleano, por ejemplo devolvería una cadena con el valor true o false

Objetos y Clases

Programación Orientada a Objetos

Se ha comentado anteriormente en este manual que Java es un lenguaje totalmente orientado a objetos. De hecho siempre hemos definido una clase pública con un método main que permite que se pueda visualizar en la pantalla el programa Java.

La gracia de la POO es que se hace que los problemas sean más sencillos, al permitir dividir el problema. Está división se hace en objetos, de forma que cada objeto funcione de forma totalmente independiente. Un objeto es un elemento del programa que posee sus propios datos y su propio funcionamiento.

Es decir un objeto está formado por datos (propiedades) y funciones que es capaz de realizar el objeto (métodos).

Antes de poder utilizar un objeto, se debe definir su clase. La clase es la definición de un tipo de objeto. Al definir una clase lo que se hace es indicar como funciona un determinado tipo de objetos. Luego, a partir de la clase, podremos crear objetos de esa

Propiedades de la POO

Encapsulamiento

Una clase se compone tanto de variables (propiedades) como de funciones y procedimientos (métodos). De hecho no se pueden definir variables (ni funciones) fuera de una clase (es decir no hay variables globales).
Ocultación. Hay una zona oculta al definir la clases (zona privada) que sólo es utilizada por esa clases y por alguna clase relacionada. Hay una zona pública (llamada también interfaz de la clase) que puede ser utilizada por cualquier parte del código.

Polimorfismo

Cada método de una clase puede tener varias definiciones distintas. En el caso del parchís: partida.empezar(4) empieza una partida para cuatro jugadores, partida.empezar(rojo, azul) empieza una partida de dos jugadores para los colores rojo y azul; estas son dos formas distintas de emplear el método empezar, que es polimórfico.

Herencia

Una clase puede heredar propiedades de otra.

Introducción al concepto de objeto

Un objeto es cualquier entidad representable en un programa informático, bien sea real (ordenador) o bien sea un concepto (transferencia). Un objeto en un sistema posee: una identidad, un estado y un comportamiento.

El estado marca las condiciones de existencia del objeto dentro del programa. Lógicamente este estado puede cambiar. Un coche puede estar parado, en marcha, estropeado, funcionando, sin gasolina, etc.

El comportamiento determina como responde el objeto ante peticiones de otros objetos. Por ejemplo un objeto conductor puede lanzar el mensaje arrancar a un coche. El comportamiento determina qué es lo que hará el objeto.

La identidad determina que cada objeto es único aunque tengan el mismo valor. No existen dos objetos iguales. Lo que sí existe es dos referencias al mismo objeto.

Los objetos se manejan por referencias, existirá una referencia a un objeto. De modo que esa referencia permitirá cambiar los atributos del objeto. Incluso puede haber varias referencias al mismo objeto, de modo que si una referencia cambia el estado del objeto, el resto (lógicamente) mostrarán esos cambios.

Los objetos por valor son los que no usan referencias y usan copias de valores concretos. En Java estos objetos son los tipos simples: int, char, byte, short, long, float, double y boolean. El resto son todos objetos (incluidos los arrays y Strings).

Clases

Por ejemplo, si quisiéramos crear el juego del parchís en Java, una clase sería la casilla, otra las fichas, otra el dado, etc., etc. En el caso de la casilla, se definiría la clase para indicar su funcionamiento y sus propiedades, y luego se crearía tantos objetos casilla como casillas tenga el juego.

Lo mismo ocurriría con las fichas, la clase ficha definiría las propiedades de la ficha (color y posición por ejemplo) y su funcionamiento mediante sus métodos (por ejemplo un método sería mover, otro llegar a la meta, etc., etc.,), luego se crearían tantos objetos ficha, como fichas tenga el juego.

Las clases son las plantillas para hacer objetos. Una clase sirve para definir una serie de objetos con propiedades (atributos), comportamientos (operaciones o métodos), y semántica comunes. Hay que pensar en una clase como un molde. A través de las clases se obtienen los objetos en sí.

Es decir antes de poder utilizar un objeto se debe definir la clase a la que pertenece, esa definición incluye:

Sus atributos. Es decir, los datos miembros de esa clase. Los datos pueden ser públicos (accesibles desde otra clase) o privados (sólo accesibles por código de su propia clase. También se las llama campos.

Sus métodos. Las funciones miembro de la clase. Son las acciones (u operaciones) que puede realizar la clase.

Código de inicialización. Para crear una clase normalmente hace falta realizar operaciones previas (es lo que se conoce como el constructor de la clase).

Otras clases. Dentro de una clase se pueden definir otras clases (clases internas, son consideradas como asociaciones dentro de UML).

Nombre de clase

Atributos

Métodos

Ilustración 5, Clase en notación UML

El formato general para crear una clase en Java es:

```
[acceso] class nombreDeClase {
  [acceso] [static] tipo atributo1;
  [acceso] [static] tipo atributo2;
```

```
    [acceso] [static] tipo atributo3;
    ...
    [access] [static] tipo método1(listaDeArgumentos) {
        ...código del método...
    }
    ...
}
```

La palabra opcional static sirve para hacer que el método o la propiedad a la que precede se pueda utilizar de manera genérica (más adelante se hablará de clases genéricas), los métodos o propiedades así definidos se llaman atributos de clase y métodos de clase respectivamente. Su uso se verá más adelante. Ejemplo;

```
class Noria {
    double radio;
    void girar(int velocidad){
        ...//definición del método
    }
    void parar(){...
}
```

Noria
radio:double
parar()
girar(int)

Objetos

Se les llama instancias de clase. Son un elemento en sí de la clase (en el ejemplo del parchís, una ficha en concreto). Un objeto se crea utilizando el llamado constructor de la clase. El constructor es el método que permite iniciar el objeto.

datos miembro (propiedades o atributos)

Para poder acceder a los atributos de un objeto, se utiliza esta sintaxis:

objeto.atributo

Por ejemplo:

Noria.radio;

Métodos

Los métodos se utilizan de la misma forma que los atributos, excepto porque los métodos poseen siempre paréntesis, dentro de los cuales pueden ir valore snecesarios para la ejecución del método (parámetros):

objeto.método(argumentosDelMétodo)

Los métodos siempre tienen paréntesis (es la diferencia con las propiedades) y dentro de los paréntesis se colocan los argumentos del método. Que son los datos que necesita el método para funcionar. Por ejemplo:

MiNoria.gira(5);

Lo cual podría hacer que la Noria avance a 5 Km/h.

Herencia

En la POO tiene mucha importancia este concepto, la herencia es el mecanismo que permite crear clases basadas en otras existentes. Se dice que esas clases descienden de las primeras. Así por ejemplo, se podría crear una clase llamada vehículo cuyos métodos serían mover, parar, acelerar y frenar. Y después se podría crear una clase coche basada en la anterior que tendría esos mismos métodos (les heredaría) y además añadiría algunos propios, por ejemplo abrirCapó o cambiarRueda.

Creación de objetos de la clase

Una vez definida la clase, se pueden utilizar objetos de la clase. Normalmente consta de dos pasos. Su declaración, y su creación. La declaración consiste en indicar que se va a utilizar un objeto de una clase determinada. Y se hace igual que cuando se declara una variable simple. Por ejemplo:

Noria noriaDePalencia;

Eso declara el objeto noriaDePalencia como objeto de tipo Noria; se supone que previamente se ha definido la clase Noria.

Para poder utilizar un objeto, hay que crearle de verdad. Eso consiste en utilizar el operador new. Por ejemplo:

noriaDePalencia = new Noria();

Al hacer esta operación el objeto reserva la memoria que necesita y se inicializa el objeto

mediante su constructor. Más adelante veremos como definir el constructor.

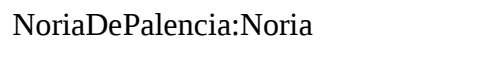


Ilustración 7, Objeto NoriaDePalencia de la clase Noria en notación UML

Especificadores de acceso

Se trata de una palabra que antecede a la declaración de una clase, método o propiedad de clase. Hay tres posibilidades: public, protected y private. Una cuarta posibilidad es no utilizar ninguna de estas tres palabras; entonces se dice que se ha utilizado el modificador por defecto (friendly).

Los especificadores determinan el alcance de la visibilidad del elemento al que se refieren. Referidos por ejemplo a un método, pueden hacer que el método sea visible sólo para la clase que lo utiliza (private), para éstas y las heredadas (protected), para todas las clases del mismo paquete (friendly) o para cualquier clase del tipo que sea (public).

En la siguiente tabla se puede observar la visibilidad de cada especificador:

zona	private (privado)	sin modificador (friendly)	protected (protegido)	public (público)
Misma clase	X	X	X	X
Subclase en el mismo paquete		X	X	X
Clase (no subclase) en el mismo paquete		X		X
Subclase en otro paquete			X	X
No subclase en otro paquete				X

Definir atributos de la clase (variables, propiedades o datos de la clases)

Cuando se definen los datos de una determinada clase, se debe indicar el tipo de propiedad que es (String, int, double, int[][],...) y el especificador de acceso (public, private,...). El especificador indica en qué partes del código ese dato será visible. Ejemplo:

```
class Persona {
  public String nombre;//Se puede acceder desde cualquier clase
  private int contraseña;//Sólo se puede acceder desde la
                                  //clase Persona
  protected String dirección; //Acceden a esta propiedad
                                  //esta clase y sus descendientes
```

Por lo general las propiedades de una clase suelen ser privadas o protegidas, a no ser que se trate de un valor constante, en cuyo caso se declararán como públicos.

Las variables locales de una clase pueden ser inicializadas.

```
class auto{
  public nRuedas=4;
```

Persona
+nombre:String
-contraseña:String
#direccion:String

Ilustración 8, La clase persona en UML. El signo + significa public, el signo # protected y el signo - private

Definir métodos de clase (operaciones o funciones de clase)

Un método es una llamada a una operación de un determinado objeto. Al realizar esta llamada (también se le llama enviar un mensaje), el control del programa pasa a ese método y lo mantendrá hasta que el método finalice o se haga uso de return.

Para que un método pueda trabajar, normalmente hay que pasarle unos datos en forma de argumentos o parámetros, cada uno de los cuales se separa por comas. Ejemplos de llamadas:

```
balón.botar(); //sin argumentos
miCoche.acelerar(10);
```

```
ficha.comer(posición15);posición 15 es una variable que se
                    //pasa como argumento
partida.empezarPartida("18:15",colores);
```

Los métodos de la clase se definen dentro de ésta. Hay que indicar un modificador de acceso (public, private, protected o ninguno, al igual que ocurre con las variables y con la propia clase) y un tipo de datos, que indica qué tipo de valores devuelve el método.

Esto último se debe a que los métodos son funciones que pueden devolver un determinado valor (un entero, un texto, un valor lógico,...) mediante el comando return. Si el método no devuelve ningún valor, entonces se utiliza el tipo void que significa que no devuelve valores (en ese caso el método no tendrá instrucción return).

El último detalle a tener en cuenta es que los métodos casi siempre necesitan datos para realizar la operación, estos datos van entre paréntesis y se les llama argumentos. Al definir el método hay que indicar que argumentos se necesitan y de qué tipo son.

Ejemplo:

```
public class vehiculo {
      /** Función principal */
      int ruedas;
      private double velocidad=0;
      String nombre;
      /** Aumenta la velocidad*/
        public void acelerar(double cantidad) {
            velocidad += cantidad;
      }
      /** Disminuye la velocidad*/
        public void frenar(double cantidad) {
            velocidad -= cantidad;
      }
      /** Devuelve la velocidad*/
      public double obtenerVelocidad(){
            return velocidad;
        }
public static void main(String args[]){
          vehiculo miCoche = new vehiculo();
```

```
        miCoche.acelerar(12);
        miCoche.frenar(5);
        System.out.println(miCoche.obtenerVelocidad());
    } // Da 7.0
```

En la clase anterior, los métodos acelerar y frenar son de tipo void por eso no tienen sentencia return. Sin embargo el método obtenerVelocidad es de tipo double por lo que su resultado es devuelto por la sentencia return y puede ser escrito en pantalla.

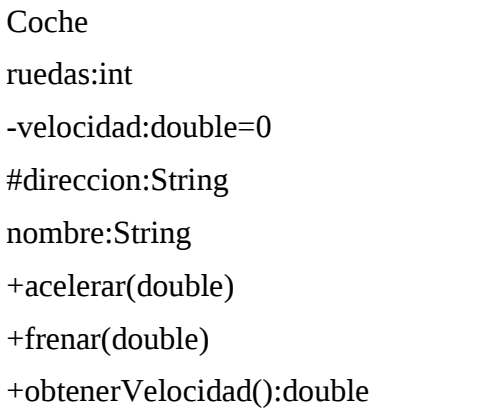


Ilustración 9, Versión UML de la clase Coche

Argumentos por valor y por referencia

En todos los lenguajes éste es un tema muy importante. Los argumentos son los datos que recibe un método y que necesita para funcionar. Ejemplo:

```
public class Matemáticas {
  public double factorial(int n){
        double resultado;
        for (resultado=n;n>1;n--) resultado*=n;
        return resultado;
  }
  ...
public static void main(String args[]){
  Matemáticas m1=new Matemáticas();
  double x=m1.factorial(25);//Llamada al método
  }
```

En el ejemplo anterior, el valor 25 es un argumento requerido por el método factorial

para que éste devuelva el resultado (que será el factorial de 25). En el código del método factorial, este valor 25 es copiado a la variable n, que es la encargada de almacenar y utilizar este valor.

Se dice que los argumentos son por valor, si la función recibe una copia de esos datos, es decir la variable que se pasa como argumento no estará afectada por el código.

Ejemplo:

```
class prueba {
   public void metodo1(int entero){
         entero=18;
...
   }
...
public static void main(String args[]){
         int x=24;
        prueba miPrueba = new prueba();
         miPrueba.metodo1(x);
         System.out.println(x); //Escribe 24, no 18
}
```

Este es un ejemplo de paso de parámetros por valor. La variable x se pasa como argumento o parámetro para el método metodo1, allí la variable entero recibe una copia del valor de x en la variable entero, y a esa copia se le asigna el valor 18. Sin embargo la variable x no está afectada por esta asignación.

Sin embargo en este otro caso:

```
class prueba {
   public void metodo1(int[] entero){
         entero[0]=18;
...
   }
...
public static void main(String args[]){
         int x[]={24,24};
        prueba miPrueba = new prueba();
```

```
        miPrueba.metodo1(x);
        System.out.println(x[0]); //Escribe 18, no 24
```

Aquí sí que la variable x está afectada por la asignación entero[0]=18. La razón es porque en este caso el método no recibe el valor de esta variable, sino la referencia, es decir la dirección física de esta variable. entero no es una replica de x, es la propia x llamada de otra forma.

Los tipos básicos (int, double, char, boolean, float, short y byte) se pasan por valor. También se pasan por valor las variables String. Los objetos y arrays se pasan por referencia.

Devolución de valores

Los métodos pueden devolver valores básicos (int, short, double, etc.), Strings, arrays e incluso objetos.

En todos los casos es el comando return el que realiza esta labor. En el caso de arrays y objetos, devuelve una referencia a ese array u objeto. Ejemplo:

```
class FabricaArrays {
    public int[] obtenArray(){
        int array[]= {1,2,3,4,5};
        return array;
    }
}
public class returnArray {
    public static void main(String[] args) {
        FabricaArrays fab=new FabricaArrays();
        int nuevoArray[]=fab.obtenArray();
    }
}
```

Sobrecarga de métodos

Una propiedad de la POO es el polimorfismo. Java posee esa propiedad ya que admite sobrecargar los métodos. Esto significa crear distintas variantes del mismo método.

Ejemplo:

```
class Matemáticas{
  public double suma(double x, double y) {
      return x+y;
  }
  public double suma(double x, double y, double z){
      return x+y+z;
  }
  public double suma(double[] array){
      double total =0;
      for(int i=0; i<array.length;i++){
          total+=array[i];
      }
      return total;
  }
```

La clase matemáticas posee tres versiones del método suma. una versión que suma dos números double, otra que suma tres y la última que suma todos los miembros de un array de doubles. Desde el código se puede utilizar cualquiera de las tres versiones según convenga.

La referencia this

La palabra this es una referencia al propio objeto en el que estamos. Ejemplo:

```
class punto {
  int posX, posY;//posición del punto
  punto(posX, posY){
      this.posX=posX;
      this.posY=posY;
  }
```

En el ejemplo hace falta la referencia this para clarificar cuando se usan las propiedades posX y posY, y cuando los argumentos con el mismo nombre. Otro ejemplo:

```
class punto {
  int posX, posY;
  ...
```

```
/**Suma las      coordenadas de otro punto*/
public void      suma(punto punto2){
      posX =    punto2.posX;
      posY =    punto2.posY;
}
/** Dobla el valor de las coordenadas del punto*/
      public void dobla(){
            suma(this);
      }
```

En el ejemplo anterior, la función dobla, dobla el valor de las coordenadas pasando el propio punto como referencia para la función suma (un punto sumado a sí mismo, daría el doble).

Los posibles usos de this son:

this. Referencia al objeto actual. Se usa por ejemplo pasarle como parámetro a un método cuando es llamado desde la propia clase.

this.atributo. Para acceder a una propiedad del objeto actual.

this.método(parámetros). Permite llamar a un método del objeto actual con los parámetros indicados.

this(parámetros). Permite llamar a un constructor del objeto actual. Esta llamada sólo puede ser empleada en la primera línea de un constructor.

Creación de constructores

Un constructor es un método que es llamado automáticamente al crear un objeto de una clase, es decir al usar la instrucción new. Sin embargo en ninguno de los ejemplos anteriores se ha definido constructor alguno, por eso no se ha utilizado ningún constructor al crear el objeto.

Un constructor no es más que un método que tiene el mismo nombre que la clase. Con lo cual para crear un constructor basta definir un método en el código de la clase que tenga el mismo nombre que la clase. Ejemplo:

```
class Ficha {
  private int casilla;
  Ficha() { //constructor
        casilla = 1;
```

```
  }
  public void avanzar(int n) {
        casilla += n;
  }
  public int casillaActual(){
        return casilla;
  }
}
public class app {
  public static void main(String[] args) {
        Ficha ficha1 = new Ficha();
        ficha1.avanzar(3);
        System.out.println(ficha1.casillaActual());//Da 4
```

En la línea Ficha ficha1 = new Ficha(); es cuando se llama al constructor, que es el que coloca inicialmente la casilla a 1. Pero el constructor puede tener parámetros:

```
class Ficha {
  private int casilla; //Valor inicial de la propiedad
  Ficha(int n) { //constructor
        casilla = n;
  }
  public void avanzar(int n) {
        casilla += n;
  }
  public int casillaActual(){
        return casilla;
  }
}

public class app {
  public static void main(String[] args) {
        Ficha ficha1 = new Ficha(6);
        ficha1.avanzar(3);
        System.out.println(ficha1.casillaActual());//Da 9
```

En este otro ejemplo, al crear el objeto ficha1, se le da un valor a la casilla, por lo que la

casilla vale al principio 6.

Hay que tener en cuenta que puede haber más de un constructor para la misma clase. Al igual que ocurría con los métodos, los constructores se pueden sobrecargar.

De este modo en el código anterior de la clase Ficha se podrían haber colocado los dos constructores que hemos visto, y sería entonces posible este código:

```
Ficha ficha1= new Ficha(); //La propiedad casilla de la
                           //ficha valdrá 1
Ficha ficha1= new Ficha(6); //La propiedad casilla de la
                           //ficha valdrá 6
```

Cuando se sobrecargan los constructores (se utilizan varias posibilidades de constructor), se pueden hacer llamadas a constructores mediante el objeto this

métodos y propiedades genéricos (static)

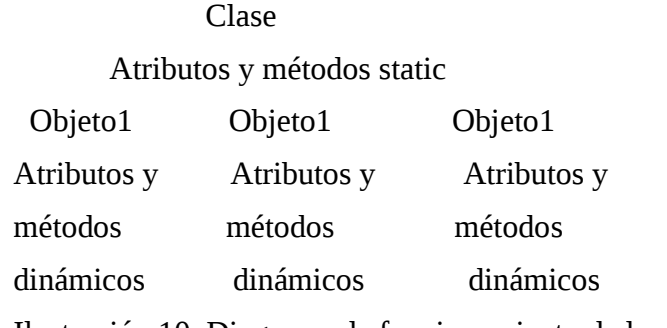


Ilustración 10, Diagrama de funcionamiento de los métodos y atributos static

Hemos visto que hay que crear objetos para poder utilizar los métodos y propiedades de una determinada clase. Sin embargo esto no es necesario si la propiedad o el método se definen precedidos de la palabra clave static. De esta forma se podrá utilizar el método sin definir objeto alguno, utilizando el nombre de la clase como si fuera un objeto. Así funciona la clase Math (véase la clase Math, página 23). Ejemplo:

```
class Calculadora {
  static public int factorial(int n) {
      int fact=1;
      while (n>0) {
          fact *=n--;
      }
      return fact;
```

```
  }
}
public class app {
  public static void main(String[] args) {
        System.out.println(Calculadora.factorial(5));
  }
}
```

En este ejemplo no ha hecho falta crear objeto alguno para poder calcular el factorial.

Una clase puede tener métodos y propiedades genéricos (static) y métodos y propiedades dinámicas (normales).

Cada vez que se crea un objeto con new, se almacena éste en memoria. Los métodos y propiedades normales, gastan memoria por cada objeto que se cree, sin embargo los métodos estáticos no gastan memoria por cada objeto creado, gastan memoria al definir la clase sólo. Es decir los métodos y atributos static son los mismos para todos los objetos creados, gastan por definir la clase, pero no por crear cada objeto.

Hay que crear métodos y propiedades genéricos cuando ese método o propiedad vale o da el mismo resultado en todos los objetos. Pero hay que utilizar métodos normales (dinámicos) cuando el método da resultados distintos según el objeto. Por ejemplo en un clase que represente aviones, la altura sería un atributo dinámico (distinto en cada objeto), mientras que el número total de aviones, sería un método static (es el mismo para todos los aviones).

El método main

Hasta ahora hemos utilizado el método main de forma incoherente como único posible mecanismo para ejecutar programas. De hecho este método dentro de una clase, indica que la clase es ejecutable desde la consola. Su prototipo es:

```
public static void main(String[] args){
  ...instruccionesejecutables....
}
```

Hay que tener en cuenta que el método main es estático, por lo que no podrá utilizar atributos o métodos dinámicos de la clase.

Los argumentos del método main son un array de caracteres donde cada elemento del array es un parámetro enviado por el usuario desde la línea de comandos. A este argumento se le llama comúnmente args. Es decir, si se ejecuta el programa con:

java claseConMain uno dos

Entonces el método main de esta clase recibe un array con dos elementos, el primero es la cadena “uno” y el segundo la cadena “dos” (es decir args[0]=”uno”; args[1]=”dos”).

destrucción de objetos

En C y C++ todos los programadores saben que los objetos se crean con new y para eliminarles de la memoria y así ahorrarla, se deben eliminar con la instrucción delete. Es decir, es responsabilidad del programador eliminar la memoria que gastaban los objetos que se van a dejar de usar. La instrucción delete del C++ llama al destructor de la clase, que es una función que se encarga de eliminar adecuadamente el objeto.

La sorpresa de los programadores C++ que empiezan a trabajar en Java es que no hay instrucción delete en Java. La duda está entonces, en cuándo se elimina la memoria que ocupa un objeto.

En Java hay un recolector de basura (garbage collector) que se encarga de gestionar los objetos que se dejan de usar y de eliminarles de memoria. Este proceso es automático e impredecible y trabajo en un hilo (thread) de baja prioridad.

Por lo general ese proceso de recolección de basura, trabaja cuando detecta que un objeto hace demasiado tiempo que no se utiliza en un programa. Esta eliminación depende de la máquina virtual, en casi todas la recolección se realiza periódicamente en un determinado lapso de tiempo. La implantación de máquina virtual conocida como HotSpot1 suele hacer la recolección mucho más a menudo

Se puede forzar la eliminación de un objeto asignándole el valor null, pero teniendo en cuenta que eso no equivale al famoso delete del lenguaje C++. Con null no se libera inmediatamente la memoria, sino que pasará un cierto tiempo (impredecible, por otro lado) hasta su total destrucción.

Se puede invocar al recolector de basura desde el código invocando al método estático System.gc(). Esto hace que el recolector de basura trabaje en cuanto se lea esa invocación.

Sin embargo puede haber problemas al crear referencias circulares. Como:

```
class uno {
  dos d;
  uno() { //constructor
        d = new dos();
  }
}
```

```
class dos {
  uno u;
  dos() {
        u = new uno();
  }
public class app {
```

Para saber más sobre HotSpot acudir a java.sun.com/products/hotspot/index.html.

```
   public static void main(Stgring[] args) {
        uno prueba = new uno();//referencia circular
        prueba = null; //no se liberará bien la memoria
   }
}
```

Al crear un objeto de clase uno, automáticamente se crea uno de la clase dos, que al crearse creará otro de la clase uno. Eso es un error que provocará que no se libere bien la memoria salvo que se eliminen previamente los objetos referenciados.

El método finalize

Es equivalente a los destructores del C++. Es un método que es llamado antes de eliminar definitivamente al objeto para hacer limpieza final. Un uso puede ser eliminar los objetos creados en la clase para eliminar referencias circulares. Ejemplo:

```
class uno {
  dos d;
  uno() {
        d = new dos();
  }
  protected void finalize(){
        d = null;//Se elimina d por lo que pudiera pasar
  }
}
```

finalize es un método de tipo protected heredado por todas las clases ya que está definido en la clase raíz Object.

La diferencia de finalize respecto a los métodos destructores de C++ estriba en que en Java no se llaman al instante (de hecho es imposible saber cuando son llamados). la llamada System.gc() llama a todos los finalize pendientes inmediatamente (es una forma de probar si el método finalize funciona o no).

Herencia

Introducción

Es una de las armas fundamentales de la programación orientada a objetos. Permite crear nuevas clases que heredan características presentas en clases anteriores. Esto facilita enormemente el trabajo porque ha permitido crear clases estándar para todos los programadores y a partir de ellas crear nuestras propias clases personales. Esto es más cómodo que tener que crear nuestras clases desde cero.

Para que una clase herede las características de otra hay que utilizar la palabra clave extends tras el nombre de la clase. A esta palabra le sigue el nombre de la clase cuyas características se heredarán. Sólo se puede tener herencia de una clase (a la clase de la que se hereda se la llama superclase y a la clase heredada se la llama subclase).
Ejemplo:

```
class coche extends vehiculo {
...
} //La clase coche parte de la definición de vehículo
                    superclase
                   vehículo
                   +ruedas:int;
                   +velocidad:double
                   +acelerar(int)
heredado
                   +frenar(int)
                            subclase
            coche
            +ruedas:int=4
redefinido
            +gasolina:int
   propio
            +repostar(int)
```

Ilustración 11, herencia

Métodos y propiedades no heredados

Por defecto se heredan todos los métodos y propiedades protected y public (no se heredan los private). Además si se define un método o propiedad en la subclase con el mismo nombre que en la superclase, entonces se dice que se está redefiniendo el método, con lo cual no se hereda éste, sino que se reemplaza por el nuevo.

Ejemplo:

```
class vehiculo {
  public int velocidad;
  public int ruedas;
  public void parar() {
        velocidad = 0;
  }
  public void acelerar(int kmh) {
        velocidad += kmh;
  }
class coche extends vehiculo{
  public int ruedas=4;
  public int gasolina;
  public void repostar(int litros) {
        gasolina+=litros;
  }
}
.....
public class app {
  public static void main(String[] args) {
        coche coche1=new coche();
        coche.acelerar(80);//Método heredado
        coche.repostar(12);
```

Anulación de métodos

Como se ha visto, las subclases heredan los métodos de las superclases. Pero es más, también los pueden sobrecargar para proporcionar una versión de un determinado método.

Por último, si una subclase define un método con el mismo nombre, tipo y argumentos que un método de la superclase, se dice entonces que se sobrescribe o anula el método de la superclase. Ejemplo:

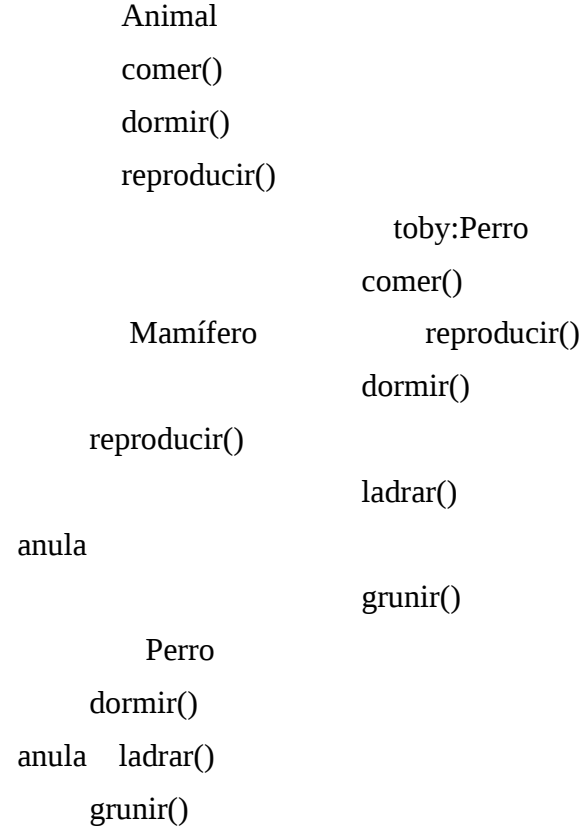


Ilustración 12, anulación de métodos

Super

A veces se requiere llamar a un método de la superclase. Eso se realiza con la palabra reservada super. Si this hace referencia a la clase actual, super hace referencia a la superclase respecto a la clase actual, con lo que es un método imprescindible para poder acceder a métodos anulados por herencia. Ejemplo

```
public class vehiculo{
  double velocidad;
```

```
    ...
    public void acelerar(double cantidad){
        velocidad+=cantidad;
    }
}
public class coche extends vehiculo{
    double gasolina;
    public void acelerar(double cantidad){
        super.acelerar(cantidad);
        gasolina*=0.9;
    }
```

En el ejemplo anterior, la llamada super.acelerar(cantidad) llama al método acelerar de la clase vehículo (el cual acelerará la marcha). Es necesario redefinir el método acelerar

en la clase coche ya que aunque la velocidad varía igual que en la superclase, hay que tener en cuenta el consumo de gasolina

Se puede incluso llamar a un constructor de una superclase, usando la sentencia super(). Ejemplo:

```
public class vehiculo{
    double velocidad;
    public vehiculo(double v){
        velocidad=v;
    }
}
public class coche extends vehiculo{
    double gasolina;
    public coche(double v, double g){
        super(v);//Llama al constructor de la clase vehiculo
        gasolina=g
    }
}
```

Por defecto Java realiza estas acciones:

Si la primera instrucción de un constructor de una subclase es una sentencia que no es ni super ni this, Java añade de forma invisible e implícita una llamada super() al constructor por defecto de la superclase, luego inicia las variables de la

subclase y luego sigue con la ejecución normal.
Si se usa super(..) en la primera instrucción, entonces se llama al constructor seleccionado de la superclase, luego inicia las propiedades de la subclase y luego sigue con el resto de sentencias del constructor.
Finalmente, si esa primera instrucción es this(..), entonces se llama al constructor seleccionado por medio de this, y después continúa con las sentencias del constructor. La inicialización de variables la habrá realizado el constructor al que se llamó mediante this.

Casting de clases

Como ocurre con los tipos básicos (ver conversión entre tipos (casting), página 18, es posible realizar un casting de objetos para convertir entre clases distintas. Lo que ocurre es que sólo se puede realizar este casting entre subclases. Es decir se realiza un casting para especificar más una referencia de clase (se realiza sobre una superclase para convertirla a una referencia de una subclase suya).

En cualquier otro caso no se puede asignar un objeto de un determinado tipo a otro.

Ejemplo:

```
Vehiculo vehiculo5=new Vehiculo();
Coche cocheDePepe = new Coche("BMW");
vehiculo5=cocheDePepe //Esto sí se permite
cocheDePepe=vehiculo5;//Tipos incompatibles
cocheDepepe=(coche)vehiculo5;//Ahora sí se permite
```

Hay que tener en cuenta que los objetos nunca cambian de tipo, se les prepara para su asignación pero no pueden acceder a propiedades o métodos que no les sean propios. Por ejemplo, si repostar() es un método de la clase coche y no de vehículo:

```
Vehiculo v1=new Vehiculo();
Coche c=new Coche();
v1=c;//No hace falta casting
v1.repostar(5);//¡¡¡Error!!!
```

Cuando se fuerza a realizar un casting entre objetos, en caso de que no se pueda realizar ocurrirá una excepción del tipo ClassCastingException. Realmente sólo se puede hacer un casting si el objeto originalmente era de ese tipo. Es decir la instrucción:

cocheDepepe=(Coche) vehiculo4;

Sólo es posible si vehiculo4 hace referencia a un objeto coche.

Instanceof

Permite comprobar si un determinado objeto pertenece a una clase concreta. Se utiliza de esta forma:

objeto instanceof clase

Comprueba si el objeto pertenece a una determinada clase y devuelve un valor true si es así. Ejemplo:

```
Coche miMercedes=new Coche();
if (miMercedes instanceof Coche)
  System.out.println("ES un coche");
if (miMercedes instanceof Vehículo)
  System.out.println("ES un coche");
if (miMercedes instanceof Camión)
  System.out.println("ES un camión");
```

En el ejemplo anterior aparecerá en pantalla:

ES un coche

ES un vehiculo

Clases abstractas

A veces resulta que en las superclases se desean incluir métodos teóricos, métodos que no se desea implementar del todo, sino que sencillamente se indican en la clase para que el desarrollador que desee crear una subclase heredada de la clase abstracta, esté obligado a sobrescribir el método.

A las clases que poseen métodos de este tipo (métodos abstractos) se las llama clases abstractas. Son clases creadas para ser heredadas por nuevas clases creadas por el programador. Son clases base para herencia. Las clases abstractas no deben de ser instanciadas (no se pueden crear objetos de las clases abstractas).

Una clase abstracta debe ser marcada con la palabra clave abstract. Cada método

abstracto de la clase, también llevará el abstract. Ejemplo:

```
abstract class vehiculo {
  public int velocidad=0;
  abstract public void acelera();
  public void para() {velocidad=0;}
}
class coche extends vehiculo {
  public void acelera() {
        velocidad+=5;
  }
}
public class prueba {
  public static void main(String[] args) {
        coche c1=new coche();
        c1.acelera();
        System.out.println(c1.velocidad);
        c1.para();
        System.out.println(c1.velocidad);
  }
}
```

Final

Se trata de una palabra que se coloca antecediendo a un método, variable o clase. Delante de un método en la definición de clase sirve para indicar que ese método no puede ser sobrescrito por las subclases. Si una subclase intentar sobrescribir el método, el compilador de Java avisará del error.

Si esa misma palabra se coloca delante de una clase, significará que esa clase no puede tener descendencia.

Por último si se usa la palabra final delante de la definición de una propiedad de clase, entonces esa propiedad pasará a ser una constante, es decir no se le podrá cambiar el valor en ninguna parte del código.

Clases internas

Se llaman clases internas a las clases que se definen dentro de otra clase. Esto permite simplificar aun más el problema de crear programas. Ya que un objeto complejo se puede descomponer en clases más sencillas. Pero requiere esta técnica una mayor pericia por parte del programador.

Al definir una clase dentro de otra, estamos haciéndola totalmente dependiente. Normalmente se realiza esta práctica para crear objetos internos a una clase (el motor de un coche por ejemplo), de modo que esos objetos pasan a ser atributos de la clase.

Por ejemplo:

```
public class Coche {
    public int velocidad;
    public Motor motor;
    public Coche(int cil) {
        motor=new Motor(cil);
        velocidad=0;
    }
    public class Motor{ //Clase interna
        public int cilindrada;
        public Motor(int cil){
            cilindrada=cil;
        }
    }
}
```

El objeto motor es un objeto de la clase Motor que es interna a Coche. Si quisiéramos acceder al objeto motor de un coche sería:

```
Coche c=new Coche(1200);
System.out.println(c.motor.cilindrada);//Saldrá 1200
```

Las clases internas pueden ser privadas, protegidas o públicas. Fuera de la clase contenedora no pueden crear objetos (sólo se pueden crear motores dentro de un coche), salvo que la clase interna sea static en ese caso sí podrían. Por ejemplo (si la clase motor fuera estática):

```
//suponiendo que la declaración del Motor dentro de Coche es
// public class static Motor{....
```

Coche.Motor m=new Coche.Motor(1200);

Pero eso sólo tiene sentido si todos los Coches tuvieran el mismo motor.

Dejando de lado el tema de las clases static, otro problema está en el operador this. El problema es que al usar this dentro de una clase interna, this se refiere al objeto de la clase interna (es decir this dentro de Motor se refiere al objeto Motor). Para poder referirse al objeto contenedor (al coche) se usa Clase.this (Coche.this). Ejemplo:

```
public class Coche {
    public int velocidad;
    public int cilindrada;
    public Motor motor;
    public Coche(int cil) {
        motor=new Motor(cil);
        velocidad=0;
    }
    public class Motor{
        public int cilindrada;
        public Motor(int cil){
            Coche.this.cilindrada=cil;//Coche
             this.cilindrada=cil;//Motor
        }
    }
}
```

Por último las clases internas pueden ser anónimas (se verán más adelante al estar más relacionadas con interfaces y adaptadores).

Creación de paquetes

Un paquete es una colección de clases e interfaces relacionadas. El compilador de Java usa los paquetes para organizar la compilación y ejecución. Es decir, un paquete es una biblioteca. De hecho el nombre completo de una clase es el nombre del paquete en el que está la clase, punto y luego el nombre de la clase. Es decir si la clase Coche está dentro del paquete locomoción, el nombre completo de Coche es locomoción.Coche.

A veces resulta que un paquete está dentro de otro paquete, entonces habrá que indicar la ruta completa a la clase. Por ejemplo locomoción.motor.Coche

Mediante el comando import (visto anteriormente), se evita tener que colocar el

nombre completo. El comando import se coloca antes de definir la clase. Ejemplo:

```
import locomoción.motor.Coche;
```

Gracias a esta instrucción para utilizar la clase Coche no hace falta indicar el paquete en el que se encuentra, basta indicar sólo Coche. Se puede utilizar el símbolo asterisco como comodín.

Ejemplo:

```
import locomoción.*;
//Importa todas las clase del paquete locomoción
```

Esta instrucción no importa el contenido de los paquetes interiores a locomoción (es decir que si la clase Coche está dentro del paquete motor, no sería importada con esa instrucción, ya que el paquete motor no ha sido importado, sí lo sería la clase locomoción.BarcoDeVela). Por ello en el ejemplo lo completo sería:

```
import locomoción.*;
import locomoción.motor.*;
```

Cuando desde un programa se hace referencia a una determinada clase se busca ésta en el paquete en el que está colocada la clase y, sino se encuentra, en los paquetes que se han importado al programa. Si ese nombre de clase se ha definido en un solo paquete, se usa. Si no es así podría haber ambigüedad por ello se debe usar un prefijo delante de la clase con el nombre del paquete.

Es decir:

```
paquete.clase
```

O incluso:

```
paquete1.paquete2.....clase
```

En el caso de que el paquete sea subpaquete de otro más grande.

Las clases son visibles en el mismo paquete a no ser que se las haya declarado con el modificador private.

Organización de los paquetes

Los paquetes en realidad son subdirectorios cuyo raíz debe ser absolutamente accesible por el sistema operativo. Para ello es necesario usar la variable de entorno CLASSPATH de la línea de comandos. Esta variable se suele definir en el archivo autoexec.bat o en MI PC en el caso de las últimas versiones de Windows (Véase proceso de compilación, página 9). Hay que añadirla las rutas a las carpetas que

contienen los paquetes (normalmente todos los paquetes se suelen crear en la misma carpeta), a estas carpetas se las llama filesystems.

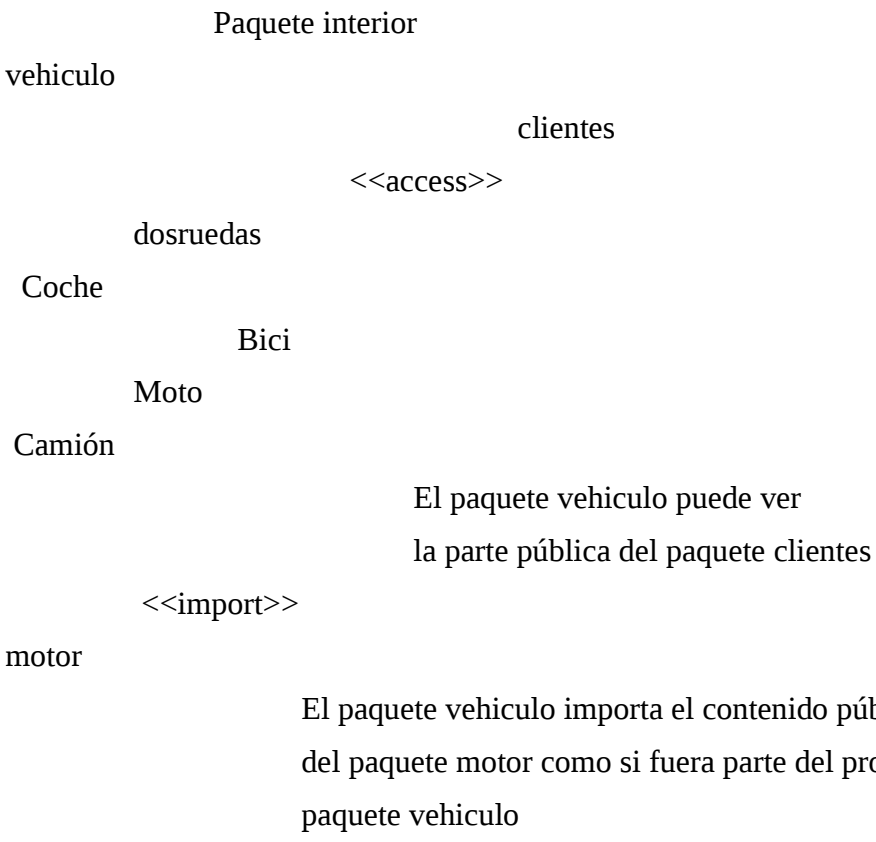
Así para el paquete prueba.reloj tiene que haber una carpeta prueba, dentro de la cual habrá una carpeta reloj y esa carpeta prueba tiene que formar parte del classpath.

Una clase se declara perteneciente aun determinado paquete usando la instrucción package al principio del código (sin usar esta instrucción, la clase no se puede compilar). Si se usa package tiene que ser la primera instrucción del programa Java:

```
//Clase perteneciente al paquete tema5 que está en ejemplos
package ejemplos.tema5;
```

En los entornos de desarrollo o IDEs (NetBeans, JBuilder,...) se puede uno despreocupar de la variable classpath ya que poseen mecanismos de ayuda para gestionar los paquetes. Pero hay que tener en cuenta que si se compila a mano mediante el comando java (véase proceso de compilación, página i) se debe añadir el modificador -cp para que sean accesibles las clases contenidas en paquetes del classpath (por ejemplo java -cp prueba.java).

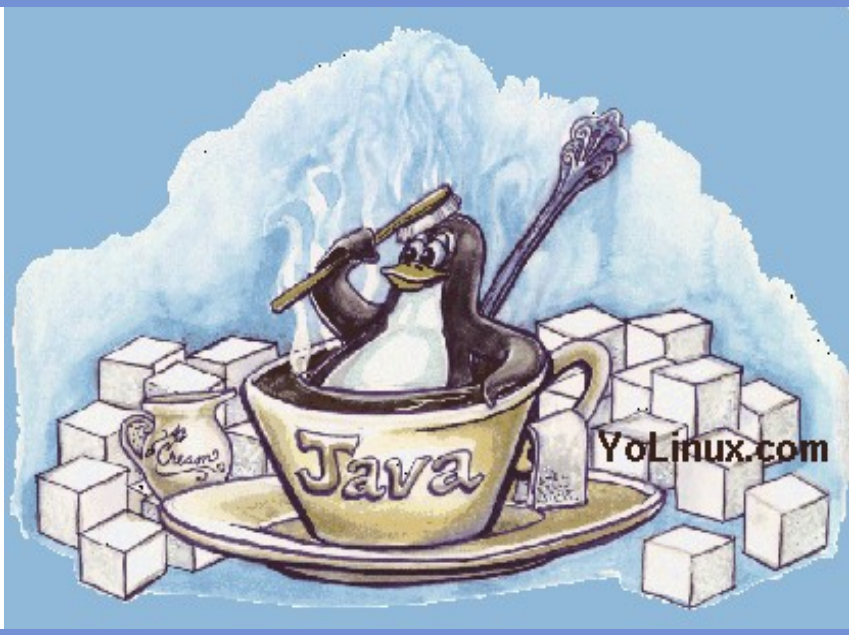
El uso de los paquetes permite que al compilar sólo se compile el código de la clase y de las clases importadas, en lugar de compilar todas las librerías. Sólo se compila lo que se utiliza.



Fundación Código Libre Dominicano

Lic. Henry Terrero.
hterrero@codigolibre.org

Ing. Jose Paredes.
jparedes@codigolibre.org



Desarrollo De aplicaciones Java Con Netbeans

EJERCICIO GUIADO. JAVA: INTRODUCCIÓN A LA POO

Introducción a la Programación Orientada a Objetos

La programación orientada a objetos es una nueva forma de entender la creación de programas. Esta filosofía de programación se basa en el concepto de objeto.

Objeto

Un objeto se define como un elemento que tiene unas **propiedades** y **métodos**.

Un objeto posee unas características (ancho, alto, color, etc…) A las características de un objeto se les llama **propiedades**.

Un objeto es un elemento "inteligente". A un objeto se le puede dar órdenes y él obedecerá. A estas órdenes se les llama **métodos**. Con los métodos podemos cambiar las características del objeto, pedirle información, o hacer que el objeto reaccione de alguna forma.

En Java todo son objetos. Veamos algunos ejemplos de uso de objetos en Java:

Ejemplo 1

Supongamos que tenemos una etiqueta llamada *etiTexto*. Esta etiqueta **es un objeto**.

Como objeto que es, la etiqueta *etiTexto* tiene una serie de características, como por ejemplo: el color de fondo, el tamaño, la posición que ocupa en la ventana, el ser opaca o no, el ser invisible o no, etc… Son las *propiedades* de la etiqueta.

A una etiqueta le podemos dar órdenes, a través de *métodos*.

A través de los métodos podemos por ejemplo cambiar las características del objeto. Por ejemplo, se puede cambiar el tamaño y posición de la etiqueta usando el método *setBounds*:

```
etiTexto.setBounds(10,20,100,20);
```

Normalmente, los métodos que permiten cambiar las características del objeto son métodos cuyo nombre empieza por *set*.

Los métodos también permiten pedirle al objeto que me de información. Por ejemplo, podríamos usar el conocido método *getText* para recoger el texto que contenga la etiqueta y almacenarlo en una variable:

```
String texto;
texto = etiTexto.getText();
```

Los métodos que le piden información al objeto suelen tener un nombre que empieza por *get*.

Los métodos también sirven para ordenarle al objeto que haga cosas. Por ejemplo, podemos ordenar a la etiqueta *etiTexto* que se vuelva a pintar en la ventana usando el método *repaint*:

```
etiTexto.repaint();
```

F d ió Códi Lib D i i

Ejemplo 2

Supongamos que tenemos un cuadro de texto llamado *txtCuadro*. Como todo en Java, un cuadro de texto es un **objeto**.

Un objeto tiene **propiedades**, es decir, características. Nuestro cuadro de texto *txtCuadro* tiene características propias: un color de fondo, un ancho, un alto, una posición en la ventana, el estar activado o no, el estar visible o no, etc…

A un objeto se le puede dar órdenes, llamadas **métodos**. Estas órdenes nos permiten cambiar las características del objeto, pedirle información, o simplemente pedirle al objeto que haga algo.

Por ejemplo, podemos cambiar el color de fondo del cuadro de texto *txtCuadro* usando el método llamado *setBackground*:

```
txtCuadro.setBackground(Color.RED);
```

Otros métodos que permiten cambiar las propiedades del objeto *txtCuadro* son:

setVisible - permite poner visible / invisible el cuadro de texto
setEnabled - permite activar / desactivar el cuadro de texto
setEditable - permite hacer que se pueda escribir o no en el cuadro de texto
setText - permite introducir un texto en el cuadro de texto
setBounds - permite cambiar el tamaño y posición del objeto
setToolTipText - permite asociar un texto de ayuda al cuadro de texto
etc…

Un objeto nos da información sobre él. Para pedirle información a un objeto usaremos métodos del tipo *get*. Por ejemplo, para pedirle al cuadro de texto el texto que contiene, usaremos el método *getText*:

```
String cadena = txtCuadro.getText();
```

Otros métodos que le piden información al cuadro de texto son:

getWidth - te dice la anchura que tiene el cuadro de texto
getHeight - te dice el alto que tiene el cuadro de texto
getSelectedText - te devuelve el texto que está seleccionado dentro del cuadro de texto
getToolTipText - te dice el texto de ayuda que tiene asociado el cuadro de texto
etc…

También se le puede dar al objeto simplemente órdenes para que haga algo. Por ejemplo, podemos ordenar al cuadro de texto *txtCuadro* que seleccione todo el texto que contiene en su interior a través del método *selectAll*:

```
txtCuadro.selectAll();
```

Otros métodos que ordenan al cuadro de texto son:

repaint - le ordena al cuadro de texto que se vuelva a pintar
copy - le ordena al cuadro de texto que copie el texto que tenga seleccionado
cut - le ordena al cuadro de texto que corte el texto que tenga seleccionado
paste - le ordena al cuadro que pegue el texto que se hubiera copiado o cortado
etc…

Clase

Todo objeto es de una “clase” determinada, o dicho de otra forma, tiene un “tipo de datos” determinado.

Por ejemplo, las etiquetas que se usan en las ventanas son objetos que pertenecen a la clase *JLabel*. Los cuadros de texto en cambio son objetos de la clase *JTextField*.

Para poder usar un objeto hay que *declararlo* y *construirlo*.

Declarar un Objeto

La declaración de un objeto es algo similar a la declaración de una variable. Es en este momento cuando se le da un nombre al objeto. Para declarar un objeto se sigue la siguiente sintaxis:

```
Clase nombreobjeto;
```

Por ejemplo, para declarar la etiqueta del ejemplo 1, se usaría el siguiente código:

```
JLabel etiTexto;
```

Para declarar, en cambio, el cuadro de texto del ejemplo 2, se usaría el siguiente código:

```
JTextField txtCuadro;
```

Construir un Objeto

En el momento de la “construcción” de un objeto, se le asignan a este una serie de propiedades iniciales. Es decir, unas características por defecto. Se puede decir que es el momento en que “nace” el objeto, y este nace ya con una forma predeterminada, que luego el programador podrá cambiar usando los métodos del objeto.

Es necesario construir el objeto para poder usarlo. La construcción del objeto se hace a través del siguiente código general:

```
nombreobjeto = new Clase();
```

Por ejemplo, para construir la etiqueta del ejemplo 1, se haría lo siguiente:

```
etiTexto = new JLabel();
```

Para construir el cuadro de texto del ejemplo 2, se haría lo siguiente:

```
txtCuadro = new JTextField();
```

NOTA. En algunos casos, la sintaxis de la declaración y la construcción se une en una sola línea. Por ejemplo, supongamos que queremos declarar la etiqueta *etiTexto* y construirla todo en una línea, entonces se puede hacer lo siguiente:

```
JLabel etiTexto = new JLabel();
```

En general, para declarar y construir un objeto en una sola línea se sigue la siguiente sintaxis:

```
Clase nombreobjeto = new Clase();
```

F d ió Códi Lib D i i

La Clase como generadora de objetos

Conociendo la Clase, se pueden crear tantos objetos de dicha Clase como se quiera. Es decir, la Clase de un objeto funciona como si fuera una plantilla a partir de la cual fabricamos objetos iguales. Todos los objetos creados a partir de una clase son iguales en un primer momento (cuando se construyen) aunque luego el programador puede dar forma a cada objeto cambiando sus propiedades.

Por ejemplo, la Clase JLabel define etiquetas. Esto quiere decir que podemos crear muchas etiquetas usando esta clase:

```
JLabel etiTexto = new JLabel();
JLabel etiResultado = new JLabel();
JLabel etiDato = new JLabel();
```

En el ejemplo se han declarado y construido tres etiquetas llamadas *etiTexto, etiResultado* y *etiDato*. Las tres etiquetas en este momento son iguales, pero a través de los distintos métodos podemos dar forma a cada una:

```
etiTexto.setBackground(Color.RED);
etiTexto.setText("Hola");
etiResultado.setBackground(Color.GREEN);
etiResultado.setText("Error");
etiDato.setBackground(Color.BLUE);
etiDato.setText("Cadena");
```

En el ejemplo se le ha dado, usando el método *setBackground*, un color a cada etiqueta. Y se ha cambiado el texto de cada una. Se le da forma a cada etiqueta.

EJERCICIO

Hasta ahora ha usado objetos aunque no tenga mucha conciencia de ello. Por ejemplo ha usado botones. Como ejercicio se propone lo siguiente:

- ¿Cuál es el nombre de la clase de los botones normales que usa en sus ventanas?
- ¿Cómo declararía un botón llamado *btnAceptar*, y otro llamado *btnCancelar*?
- ¿Cómo construiría dichos botones?
- Indique algunos métodos para cambiar propiedades de dichos botones (métodos set)
- Indique algunos métodos para pedirle información a dichos botones (métodos get)
- Indique algún método para dar órdenes a dichos botones (algún método que no sea ni set ni get)

CONCLUSIÓN

Un Objeto es un elemento que tiene una serie de características llamadas PROPIEDADES.

Por otro lado, al objeto se le pueden dar órdenes que cumplirá de inmediato. A dichas órdenes se les denomina MÉTODOS.

Los métodos se pueden dividir básicamente en tres tipos:

- **Para cambiar las propiedades del objeto (Métodos set)**
- **Para pedir información al objeto (Métodos get)**
- **Para dar órdenes al objeto.**

Todo objeto pertenece a una CLASE.

La CLASE nos permite declarar objetos y construirlos:

Declaración:

```
CLASE nombreobjeto;
```

Construcción:

```
nombreobjeto = new CLASE();
```

Declaración y Construcción en una misma línea

```
CLASE nombreobjeto = new CLASE(),
```

En la construcción de un objeto se asignan unas propiedades (características) por defecto al objeto que se construye, aunque luego, estas características pueden ser cambiadas por el programador.

EJERCICIO GUIADO. JAVA: POO. CLASES PROPIAS

Objetos propios del lenguaje

Hasta el momento, todos los objetos que ha usado a la hora de programar en Java, han sido objetos incluidos en el propio lenguaje, que se encuentran disponibles para que el programador los use en sus programas.

> *Estos objetos son: las etiquetas (JLabel), botones (JButton), los cuadros de texto (JTextField), cuadros de verificación (JCheckBox), botones de opción (JRadioButton), colores (Color), imágenes de icono (ImageIcon), modelos de lista (DefaultListModel), etc, etc.*

Todos estos objetos, por supuesto, pertenecen cada uno de ellos a una Clase:

> *Las etiquetas son objetos de la clase JLabel, los botones son objetos de la clase JButton, etc.*

Todos estos objetos tienen propiedades que pueden ser cambiadas en la ventana de diseño:

> *Ancho, alto, color, alineación, etc.*

Aunque también poseen métodos que nos permiten cambiar estas propiedades durante la ejecución del programa:

> *setText cambia el texto del objeto, setBackground cambia el color de fondo del objeto, setVisible hace visible o invisible al objeto, setBounds cambia el tamaño y la posición del objeto, etc.*

En cualquier momento le podemos pedir a un objeto que nos de información sobre sí mismo usando los métodos get:

> *getText obtenemos el texto que tenga el objeto, getWidth obtenemos la anchura del objeto, getHeight obtenemos la altura del objeto, etc.*

Los objetos son como “pequeños robots” a los que se les puede dar órdenes, usando los métodos que tienen disponible.

> *Por ejemplo, le podemos decir a un objeto que se pinte de nuevo usando el método repaint, podemos ordenarle a un cuadro de texto que coja el cursor, con el método requestFocus, etc.*

Todos estos objetos, con sus propiedades y métodos, nos facilitan la creación de nuestros programas. Pero a medida que nos vamos introduciendo en el mundo de la programación y nos especializamos en un tipo de programa en concreto, puede ser necesaria la creación de objetos propios, programados por nosotros mismos, de forma que puedan ser usados en nuestros futuros programas.

Objetos propios

A la hora de diseñar un objeto de creación propia, tendremos que pensar qué propiedades debe tener dicho objeto, y métodos serán necesarios para poder trabajar con él. Dicho de otra forma, debemos pensar en qué características debe tener el objeto y qué órdenes le podré dar.

Para crear objetos propios hay que programar la Clase del objeto. Una vez programada la Clase, ya podremos generar objetos de dicha clase, declarándolos y construyéndolos como si de cualquier otro objeto se tratara.

A continuación se propondrá un caso práctico de creación de objetos propios, con el que trabajaremos en las próximas hojas.

Lo que viene a continuación es un planteamiento teórico de diseño de una Clase de Objetos.

CASO PRÁCTICO: MULTICINES AVENIDA

Planteamiento

Los Multicines Avenida nos encargan un programa para facilitar las distintas gestiones que se realizan en dichos multicines.

El multicine cuenta con varias salas, y cada una de ellas genera una serie de información. Para facilitar el control de la información de cada sala programaremos una Clase de objeto a la que llamaremos **SalaCine**.

La Clase *SalaCine*

La Clase *SalaCine* definirá características de una sala de cine, y permitirá crear objetos que representen salas de cine. Cuando la Clase *SalaCine* esté programada, se podrán hacer cosas como las que sigue:

Los Multicines Avenida tienen una sala central donde se proyectan normalmente los estrenos. Se podría crear un objeto llamado *central* de la clase *SalaCine* de la siguiente forma:

```
SalaCine central;
```

Por supuesto, este objeto puede ser construido como cualquier otro:

```
central = new SalaCine();
```

El objeto *central* representará a la sala de cine central de los Multicines Avenida.

Otro ejemplo. Los Multicines Avenida tienen una sala donde proyectan versiones originales. Se podría crear un objeto llamado *salaVO* de la clase *SalaCine* de la siguiente forma:

```
SalaCine salaVO;  //declaración

salaVO = new SalaCine(); //construcción
```

Propiedades de los objetos de la clase *SalaCine*

A la hora de decidir las propiedades de un objeto de creación propia, tenemos que preguntarnos, ¿qué información me interesa almacenar del objeto? Trasladando esta idea a nuestro caso práctico, ¿qué información me interesaría tener de cada sala de cine?

Para este ejemplo supondremos que de cada sala de cine nos interesa tener conocimiento de las siguientes características (propiedades):

- **Aforo**: define el número de butacas de la sala (un número entero).
- **Ocupadas**: define el número de butacas ocupadas (un número entero).
- **Película**: define la película que se está proyectando en el momento en la sala (una cadena de texto)
- **Entrada**: define el precio de la entrada (un número double)

Valores por defecto de los objetos *SalaCine*

Cuando se construye un objeto, se asignan unos valores por defecto a sus propiedades. Por ejemplo, cuando se construye una etiqueta (Clase JLabel), esta tiene un tamaño inicial definido, un color, etc.

Lo mismo se tiene que hacer con los objetos propios que definimos. Es necesario decidir qué valores tendrá las propiedades del objeto al construirse.

En nuestro caso, las características de un objeto *SalaCine* en el momento de construirse serán las siguientes:

Aforo: 100
Ocupadas: 0
Película: “” *(la cadena vacía)*
Entrada: 5,00

Observa este código, en él construimos el objeto correspondiente a la sala central del multicine:

```
SalaCine central;

central = new SalaCine();
```

En este momento (en el que el objeto *central* está recién construido) este objeto tiene asignado un aforo de 100, el número de butacas ocupadas es 0, la película que se proyecta en la sala central es “” y la entrada para esta sala es de 5 euros.

Los valores por defecto que se asignan a los objetos de una clase son valores arbitrarios que el programador decidirá según su conveniencia.

Métodos de los objetos *SalaCine*

Para comunicarnos con los objetos de la Clase *SalaCine* que construyamos, tendremos que disponer de un conjunto de métodos que nos permitan asignar valores a las propiedades de los objetos, recoger información de dichos objetos y que le den órdenes al objeto.

Será el programador el que decida qué métodos le interesará que posea los objetos de la Clase que está programando.

Para nuestro caso particular, supondremos que los objetos de la Clase *SalaCine* deberán tener los siguientes métodos:

Métodos de cambio de propiedades (Métodos *set*)

setAforo - asignará un aforo a la sala de cine
setOcupadas - asignará una cantidad de butacas ocupadas a la sala de cine
setLibres - asignará una cantidad de butacas libres a la sala de cine
setPelicula - asignará un título de película a la sala de cine
setEntrada - fijará el precio de las entradas a la sala de cine

Gracias a estos métodos podemos dar forma a un objeto *SalaCine* recién creado.

Por ejemplo, supongamos que queremos crear el objeto que representa la sala de versiones originales. Resulta que esta sala tiene de aforo 50 localidades, que se está proyectando la película “Metrópolis” y que la entrada para ver la película es de 3 euros. La sala está vacía de momento.

Para crear el objeto, se usaría el siguiente código:

```
//Se construye el objeto
SalaCine salaVO;
salaVO = new SalaCine();

//Se le asignan características
salaVO.setAforo(50);                //aforo 50
salaVO.setPelicula("Metrópolis");  //la película que se proyecta
salaVO.setEntrada(3);               //entrada a 3 euros
```

Al construir el objeto *salaVO* tiene por defecto los valores siguientes en sus propiedades:

Aforo: 100
Ocupadas: 0
Película: ""
Entrada: 5,00

Gracias a los métodos disponibles hemos asignados estos nuevos valores:

Aforo: 50
Ocupadas: 0
Película: "Metrópolis"
Entrada: 3,00

Métodos para pedirle información al objeto (Métodos get)

Se programarán los siguientes métodos get en la clase *SalaCine*:

getAforo - devuelve el aforo que tiene el objeto
getOcupadas - devuelve el número de butacas ocupadas que tiene el objeto
getLibres - devuelve el número de butacas que tiene libres el objeto
getPorcentaje - devolverá el porcentaje de ocupación de la sala
getIngresos - devolverá los ingresos producidos por la sala (entradas vendidas por precio)
getPelicula - devolverá el título de la película que se está proyectando
getEntrada - devolverá el precio de la entrada asignado al objeto

Estos métodos nos permitirán obtener información de un objeto del tipo *SalaCine*. Por ejemplo, supongamos que tenemos el objeto llamado *central* (correspondiente a la sala principal del multicine), para obtener la película que se está proyectando en dicha sala solo habría que usar este código:

```
String peli;      //una variable de cadena
peli = central.getPelicula();
```

O, por ejemplo, para saber los ingresos producidos por la sala central…

```
double ingresos;
ingresos = central.getIngresos();
```

Métodos para dar órdenes al objeto

Se programarán los siguientes métodos para dar órdenes a los objetos de la clase *SalaCine*.

F d ió Códi Lib D i i

vaciar

- Este método pondrá el número de plazas ocupadas a cero y le asignará a la película la cadena vacía.

entraUno

- Este método le dice al objeto que ha entrado una nueva persona en la sala. (Esto debe producir que el número de plazas ocupadas aumente en uno)

RESUMEN SALACINE

He aquí un resumen de la Clase de objetos *SalaCine*, la cual se programará en la próxima hoja:

Clase de objetos: ***SalaCine***

Propiedades de los objetos *SalaCine*:

Aforo - número entero (int)
Ocupadas - número entero (int)
Película - cadena (String)
Entrada - número decimal (double)

Valores por defecto de los objetos del tipo *SalaCine*:

Aforo: **100**
Ocupadas: **0**
Película: **(cadena vacía)**
Entrada: **5**

Métodos de los objetos del tipo *SalaCine*:

Métodos de asignación de propiedades (set)

setAforo - modifica la propiedad Aforo
setOcupadas - modifica la propiedad Ocupadas
setLibres - modifica la propiedad Ocupadas también
setPelicula - modifica la propiedad Película
setEntrada - modifica la propiedad Entrada

Métodos de petición de información (get)

getAforo - devuelve el valor de la propiedad Aforo
gnetOcupadas - devuelve el valor de la propiedad Ocupadas
getLibres - devuelve el número de butacas libres
getPorcentaje - devuelve el porcentaje de ocupación de la sala
getIngresos - devuelve los ingresos obtenidos por la venta de entradas
getPelicula - devuelve el valor de la propiedad Película
getEntrada - devuelve el valor de la propiedad Entrada

Métodos de orden

Vaciar - vacía la ocupación de la sala y borra la película
entraUno - le indica al objeto que ha entrado una persona más en la sala

F d ió Códi Lib D i i

EJERCICIO PRÁCTICO

Supongamos que programamos una Clase de objetos llamada *Rectangulo*, la cual permitirá construir objetos que representen a rectángulos.

- ¿Cómo declararía y construiría un objeto llamado *suelo* del tipo *Rectangulo*?

- Supongamos que las propiedades de los objetos de la clase *Rectangulo* sean las siguientes:

    - Base (double)
    - Altura (double)

  ¿Qué métodos de tipo *set* programaría para cambiar las propiedades de los objetos del tipo *Rectangulo*?

- Como ejemplo de los métodos anteriores, suponga que quiere asignar al objeto *suelo* una base de 30 y una altura de 50. ¿Qué código usaría?

- Teniendo en cuenta que nos puede interesar conocer el área de un objeto *Rectangulo* y su perímetro, y teniendo en cuenta que los objetos *Rectangulo* tienen dos propiedades (Base y Altura), ¿qué cuatro métodos *get* serían interesantes de programar para los objetos del tipo *Rectangulo*?

- Teniendo en cuenta los métodos del punto 4. Supongamos que quiero almacenar en una variable double llamada *area* el área del objeto *suelo*. ¿Qué código usaría? Y si quiero almacenar el perímetro del objeto *suelo* en una variable llamada *peri*?

CONCLUSIÓN

Para crear un objeto propio, necesita tener claro lo siguiente:

Nombre de la Clase del objeto.

Propiedades que tendrán los objetos de dicha clase.

Valores iniciales que tendrán las propiedades cuando se construya cada objeto.

Métodos que serán interesantes de programar:

Métodos de ajuste de propiedades (set)
Métodos de petición de información (get)
Métodos de orden

Una vez que se tenga claro lo anterior, se podrá empezar la programación de la Clase de objetos.

EJERCICIO GUIADO. JAVA: POO. PROGRAMACIÓN DE UNA CLASE

Programación de una Clase

En este ejercicio guiado, crearemos la Clase *SalaCine*, que hemos descrito en la hoja anterior. Luego, a partir de esta clase, fabricaremos objetos representando salas de cine, y los usaremos en un proyecto Java.

Recuerda las características que hemos decidido para la Clase *SalaCine* en la hoja anterior:

CLASE SALACINE

Nombre de la Clase: *SalaCine*

Propiedades de los objetos *SalaCine*:

Aforo - número entero (int)
Ocupadas - número entero (int)
Película - cadena (String)
Entrada - número decimal (double)

Valores por defecto de los objetos del tipo *SalaCine*:

Aforo: **100**
Ocupadas: **0**
Película: **(cadena vacía)**
Entrada: **5**

Métodos de los objetos del tipo *SalaCine*:

Métodos de asignación de propiedades (set)

setAforo - modifica la propiedad Aforo
setOcupadas - modifica la propiedad Ocupadas
setLibres - modifica la propiedad Ocupadas también
setPelicula - modifica la propiedad Película
setEntrada - modifica la propiedad Entrada

Métodos de petición de información (get)

getAforo - devuelve el valor de la propiedad Aforo
getOcupadas - devuelve el valor de la propiedad Ocupadas
getLibres - devuelve el número de butacas libres
getPorcentaje - devuelve el porcentaje de ocupación de la sala
getIngresos - devuelve los ingresos obtenidos por la venta de entradas
getPelicula - devuelve el valor de la propiedad Película
getEntrada - devuelve el valor de la propiedad Entrada

Métodos de orden

Vaciar - vacía la ocupación de la sala y borra la película
entraUno - le indica al objeto que ha entrado una persona más en la sala

PROGRAMACIÓN DE UNA CLASE

Fichero de la Clase

La programación de una clase de objetos se realiza en un fichero aparte, cuyo nombre es exactamente el mismo que el de la propia clase, y cuya extensión es .java.

Por ejemplo, si queremos programar la clase SalaCine, esto se debe hacer en un fichero llamado:

`SalaCine.java`

Cuando programemos esta clase dentro de NetBeans, veremos las facilidades que nos proporciona este para la creación de la clase. De momento, solo veremos de forma teórica como hay que programar la clase. (No tiene que introducir lo que viene a continuación en ningún sitio)

Estructura básica de la Clase

Dentro del fichero de la clase, comenzará la programación de esta de la siguiente forma:

```
public class SalaCine {



}
```

La programación de una clase comienza siempre con una línea de código como la que sigue:

```
public class NombreDeLaClase {


}
```

Toda la programación de la clase se introducirá dentro de las dos llaves.

Propiedades de la Clase

Lo primero que se debe introducir en la clase que se está programando son las propiedades. Las propiedades de una clase son básicamente variables globales de ésta. Si introducimos las propiedades de la clase *SalaCine*, esta nos quedaría así:

```
public class SalaCine {

      int Aforo;
      int Ocupadas;
      String Película;
      double Entrada;

}
```

Constructor de la Clase

Cuando se planteó la clase *SalaCine*, se tuvo que decidir qué valores iniciales deberían tener las propiedades de la clase. Para asignar estos valores iniciales, es necesario programar lo que se denomina el *Constructor*.

F d ió Códi Lib D i i

El *Constructor* de una clase es un método (un procedimiento para entendernos) un poco especial, ya que debe tener el mismo nombre de la clase y no devuelve nada, pero no lleva la palabra *void*. Dentro del constructor se inicializan las propiedades de la clase.

En general, la programación del constructor sigue la siguiente sintaxis:

```
public NombreDeLaClase() {
      propiedad1 = valor;
      propiedad2 = valor;
      etc…
}
```

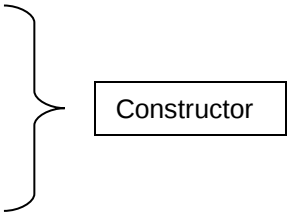
La clase *SalaCine*, añadiendo el *Constructor*, tendrá el siguiente aspecto:

```
public class SalaCine {

      int Aforo;
      int Ocupadas;
      String Película;
      double Entrada;

      //Constructor
      public SalaCine() {
            Aforo = 100;
            Ocupadas = 0;
            Pelicula = "";
            Entrada = 5.0;
      }

}
```



Observa como usamos el constructor de la clase *SalaCine* para asignar a cada propiedad los valores por defecto decididos en el diseño de la clase que se hizo en la hoja anterior.

Métodos del tipo set

Todas las clases suelen contener métodos del tipo *set*. Recuerda que estos métodos permiten asignar valores a las propiedades de la clase.

Debes tener en cuenta también que cuando se habla de método de una clase, en realidad se está hablando de un procedimiento o función, que puede recibir como parámetro determinadas variables y que puede devolver valores.

Los métodos del tipo *set* son básicamente procedimientos que reciben valores como parámetros que introducimos en las propiedades. Estos métodos no devuelven nada, así que son *void*.

Se recomienda, que el parámetro del procedimiento se llame de forma distinta a la propiedad que se asigna.

Veamos la programación del método *setAforo*, de la clase *SalaCine*:

```
public void setAforo(int afo) {

      Aforo = afo;

}
```

F d ió Códi Lib D i i

Observa este método:

1. Es void, es decir, no devuelve nada (*el significado de la palabra public se verá más adelante*)
2. El método recibe como parámetro una variable del mismo tipo que la propiedad que queremos modificar (en este caso *int*) y un nombre que se recomienda que no sea igual al de la propiedad (en nuestro caso, *afo*, de aforo)

3. Puedes observar que lo que se hace simplemente en el método es asignar la variable pasada como parámetro a la propiedad.

La mayoría de los procedimientos *set* usados para introducir valores en las propiedades tienen la misma forma. Aquí tienes la programación de los demás procedimientos *set* de la clase *SalaCine*.

```
//Método setOcupadas
public void setOcupadas(int ocu) {
      Ocupadas = ocu;
}

//Método setPelicula
public void setPelicula(String peli) {
      Pelicula = peli;
}

//Método setEntrada
public void setEntrada(double entra) {
      Entrada = entra;
}
```

Hay un método *set* de la clase *SalaCine* llamado *setLibres* cuya misión es asignar el número de localidades libres del cine. Sin embargo la clase *SalaCine* no tiene una propiedad “Libres”. En realidad, este método debe modificar el número de localidades ocupadas. Observa su programación:

```
//Método setLibres
public void setLibres(int lib) {
      int ocu;

      ocu = Aforo – lib;
      Ocupadas = ocu;
}
```

Al asignar un número de localidades ocupadas, estamos asignando indirectamente el número de localidades libres. Como puedes observar en el método, lo que se hace es calcular el número de localidades ocupadas a partir de las libres, y asignar este valor a la propiedad Ocupadas.

No se pensó en crear una propiedad de la clase llamada *Libres* ya que en todo momento se puede saber cuantas localidades libres hay restando el Aforo menos las localidades Ocupadas.

La clase *SalaCine*, añadiendo los métodos *set*, quedaría de la siguiente forma:

F d ió Códi Lib D i i

```
public class SalaCine {

	int Aforo;
	int Ocupadas;
	String Película;
	double Entrada;

	//Constructor
	public SalaCine() {
		Aforo = 100;
		Ocupadas = 0;
		Pelicula = “”;
		Entrada = 5.0;
	}

	//Métodos set

	//Método setAforo
	public void setAforo(int afo) {
		Aforo = afo;
	}

	//Método setOcupadas
	public void setOcupadas(int ocu) {
		Ocupadas = ocu;
	}

	//Método setPelicula
	public void setPelicula(String peli) {
		Pelicula = peli;
	}

	//Método setEntrada
	public void setEntrada(double entra) {
		Entrada = entra;
	}

	//Método setLibres
	public void setLibres(int lib) {
		int ocu;

		ocu = Aforo – lib;
		Ocupadas = ocu;
	}

}
```

Métodos Set

Métodos del tipo get

Al igual que los métodos *set*, los métodos *get* son muy fáciles de programar ya que suelen tener siempre la misma forma.

Estos métodos no suelen llevar parámetros y devuelven el valor de la propiedad correspondiente usando la típica instrucción *return* usada tanto en las funciones. Por tanto, un método get nunca es void. Siempre será del mismo tipo de datos que la propiedad que devuelve.

F d ió Códi Lib D i i

Veamos la programación del método *getAforo*:

```
//Método getAforo
public int getAforo() {
      return Aforo;
}
```

Como puedes ver el método simplemente devuelve el valor de la propiedad Aforo. Como esta propiedad es int, el método es int.

Los métodos que devuelven el resto de las propiedades son igual de sencillos de programar:

```
//Método getOcupadas
public int getOcupadas() {
      return Ocupadas;
}

//Método getPelicula
public String getPelicula() {
      return Película;
}

//Método getEntrada
public double getEntrada() {
      return Entrada;
}
```

Todos estos métodos son iguales. Solo tienes que fijarte en el tipo de datos de la propiedad que devuelven.

Existen otros métodos get que devuelven cálculos realizados con las propiedades. Estos métodos realizan algún cálculo y luego devuelven el resultado. Observa el siguiente método *get*:

```
//Método getLibres
public int getLibres() {
      int lib;
      lib = Aforo – Ocupadas;
      return lib;
}
```

No existe una propiedad *Libres*, por lo que este valor debe ser calculado a partir del Aforo y el número de localidades Ocupadas. Para ello restamos y almacenamos el valor en una variable a la que hemos llamado *lib*. Luego devolvemos dicha variable.

Los dos métodos *get* que quedan por programar de la clase *SalaCine* son parecidos:

```
//Método getPorcentaje
public double getPorcentaje() {
      double por;
      por = (double) Ocupadas / (double) Aforo * 100.0;
      return por;
}
```

Este método calcula el porcentaje de ocupación de la sala (es un valor double)

```
//Método getIngresos
public double getIngresos() {
      double ingre;
```

F d ió Códi Lib D i i

```
      ingre = Ocupadas * Entrada;
      return ingre;
}
```

Los ingresos se calculan multiplicando el número de entradas por lo que vale una entrada.

La clase *SalaCine* una vez introducidos los métodos get quedaría de la siguiente forma:

```
public class SalaCine {

      int Aforo;
      int Ocupadas;
      String Película;
      double Entrada;

      //Constructor
      public SalaCine() {
            Aforo = 100;
            Ocupadas = 0;
            Pelicula = “”;
            Entrada = 5.0;
      }

      //Métodos set

      //Método setAforo
      public void setAforo(int afo) {
            Aforo = afo;
      }

      //Método setOcupadas
      public void setOcupadas(int ocu) {
            Ocupadas = ocu;
      }

      //Método setPelicula
      public void setPelicula(String peli) {
            Pelicula = peli;
      }

      //Método setEntrada
      public void setEntrada(double entra) {
            Entrada = entra;
      }

      //Método setLibres
      public void setLibres(int lib) {
            int ocu;

            ocu = Aforo – lib;
            Ocupadas = ocu;
      }


      //Métodos get

      //Método getAforo
      public int getAforo() {
            return Aforo;
```

Métodos Get

F d ió Códi Lib D i i

```
	}

	//Método getOcupadas
	public int getOcupadas() {
		return Ocupadas;
	}


	//Método getPelicula
	public String getPelicula() {
		return Película;
	}

	//Método getEntrada
	public double getEntrada() {
		return Entrada;
	}

	//Método getLibres
	public int getLibres() {
		int lib;
		lib = Aforo – Ocupadas;
		return lib;
	}


	//Método getPorcentaje
	public double getPorcentaje() {
		double por;
		por = (double) Ocupadas / (double) Aforo * 100.0;
		return por;
	}


	//Método getIngresos
	public double getIngresos() {
		double ingre;
		ingre = Ocupadas * Entrada;
		return ingre;
	}
```

Métodos Get

```
}
```

Métodos de orden

Para finalizar la programación de la clase *SalaCine*, se programarán los dos métodos de orden que hemos indicado en el planteamiento de la clase. Estos métodos suelen realizar alguna tarea que involucra a las propiedades de la clase, modificándola internamente. No suelen devolver ningún valor, aunque pueden recibir parámetros.

F d ió Códi Lib D i i

Veamos la programación del método *Vaciar*, cuyo objetivo es vaciar la sala y quitar la película en proyección:

```
//Método Vaciar
public void Vaciar() {
      Ocupadas = 0;
      Película = “”;
}
```

Como se puede observar, es un método muy sencillo, ya que simplemente cambia algunas propiedades de la clase.

El método entraUno es también muy sencillo de programar. Este método le indica al objeto que ha entrado un nuevo espectador. Sabiendo esto, el objeto debe aumentar en uno el número de localidades ocupadas:

```
//Método entraUno
public void entraUno() {
      Ocupadas++;
}
```

Añadiendo estos dos últimos métodos, la programación de la clase *SalaCine* quedaría finalmente como sigue:

```
public class SalaCine {

      int Aforo;
      int Ocupadas;
      String Película;
      double Entrada;

      //Constructor
      public SalaCine() {
            Aforo = 100;
            Ocupadas = 0;
            Pelicula = “”;
            Entrada = 5.0;
      }

      //Métodos set

      //Método setAforo
      public void setAforo(int afo) {
            Aforo = afo;
      }

      //Método setOcupadas
      public void setOcupadas(int ocu) {
            Ocupadas = ocu;
      }

      //Método setPelicula
      public void setPelicula(String peli) {
            Pelicula = peli;
      }

      //Método setEntrada
      public void setEntrada(double entra) {
            Entrada = entra;
      }
```

Propiedades (variables globales)

Constructor

Métodos Set

F d ió Códi Lib D i i

```
//Método setLibres
public void setLibres(int lib) {
      int ocu;

      ocu = Aforo – lib;
      Ocupadas = ocu;
}
```

F d ió Códi Lib D i i

```
//Métodos get

//Método getAforo
public int getAforo() {
      return Aforo;
}

//Método getOcupadas
public int getOcupadas() {
      return Ocupadas;
}


//Método getPelicula
public String getPelicula() {
      return Película;
}

//Método getEntrada
public double getEntrada() {
      return Entrada;
}

//Método getLibres
public int getLibres() {
      int lib;
      lib = Aforo – Ocupadas;
      return lib;
}


//Método getPorcentaje
public double getPorcentaje() {
      double por;
      por = (double) Ocupadas / (double) Aforo * 100.0;
      return por;
}

//Método getIngresos
public double getIngresos() {
      double ingre;
      ingre = Ocupadas * Entrada;
      return ingre;
}
```

Métodos Get

```
//Métodos de orden

//Método Vaciar
public void Vaciar() {
      Ocupadas = 0;
      Película = “”;
}

//Método entraUno
public void entraUno() {
      Ocupadas++;
}
```

Métodos de orden y otros métodos.

F d ió Códi Lib D i i

}

EJERCICIOS RECOMENDADOS

Supongamos que tenemos una clase llamada *Rectangulo* que nos permitirá generar objetos de tipo rectángulo.

Sea el planteamiento de la clase *Rectangulo* el que sigue:

CLASE RECTANGULO

Nombre de la clase: Rectangulo

Propiedades de los objetos de la clase Rectangulo:

Base (double)
Altura (double)

Valores iniciales de las propiedades de los objetos de la clase Rectangulo:

Base – 100
Altura – 50

Métodos:

Métodos set:

setBase – permite asignar un valor a la propiedad Base.
setAltura – permite asignar un valor a la propiedad Altura.

Métodos get:

getBase – devuelve el valor de la propiedad Base
getAltura – devuelve el valor de la propiedad Altura
getArea – devuelve el área del rectángulo
getPerímetro – devuelve el perímetro del rectángulo

Otros métodos:

Cuadrar – este método debe hacer que la Altura tenga el valor de la Base.

SE PIDE:

Realiza (en papel) la programación de la clase Rectangulo a partir del planteamiento anterior.

CONCLUSIÓN

La programación de una clase se realiza en un fichero que tiene el mismo nombre que la clase y extensión .java

La estructura general de una clase es la siguiente:

public class NombreClase {

Propiedades (variables globales)

Constructor

Métodos set

Métodos get

Métodos de orden y otros métodos

}

El Constructor es un procedimiento que no devuelve nada pero que no es void. El constructor debe llamarse igual que la clase y se usa para asignar los valores iniciales a las propiedades.

Los métodos set son void y reciben como parámetro un valor que se asigna a la propiedad correspondiente.

Los métodos get no tienen parámetros y devuelven el valor de una propiedad de la clase, aunque también pueden realizar cálculos y devolver sus resultados.

Los métodos de orden realizan alguna tarea específica y a veces modifican las propiedades de la clase de alguna forma.

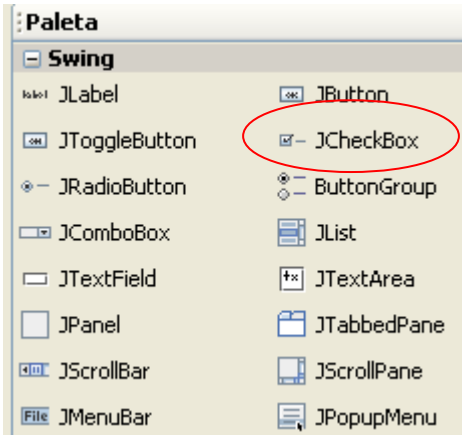
EJERCICIO GUIADO. JAVA: CUADROS DE VERIFICACIÓN

4. Realiza un nuevo proyecto.

5. En la ventana principal debes añadir lo siguiente:

    f. Un botón “Aceptar” llamado btnAceptar.
    g. Una etiqueta con borde llamada etiResultado.

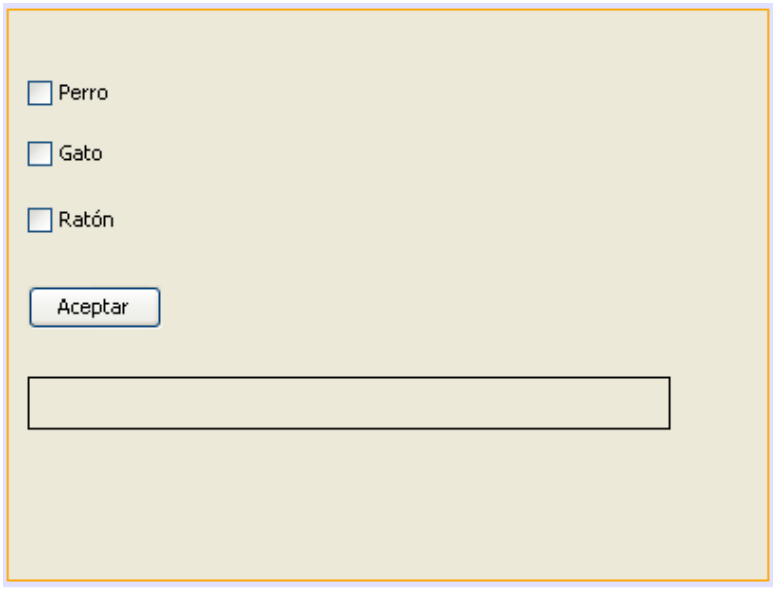
8. Añade también tres cuadros de verificación. Estos cuadros son objetos del tipo JCheckBox.



9. Añade tres JCheckBox y cambia el texto de ellos, de forma que aparezca “Perro”, “Gato” y “Ratón”.

10. Debe cambiar el nombre de cada uno de ellos. Se llamarán: chkPerro, chkGato, chkRaton.

11. La ventana tendrá el siguiente aspecto cuando termine:



12. El programa debe funcionar de la siguiente forma:

    Cuando el usuario pulse aceptar, en la etiqueta aparecerá un mensaje indicando qué animales han sido “seleccionados”. Para ello hay que programar el evento *actionPerformed* del botón Aceptar. En ese evento añada el siguiente código:

```
String mensaje="Animales elegidos: ";
if (chkPerro.isSelected()) {
    mensaje=mensaje+"Perro ";
}

if (chkGato.isSelected()) {
    mensaje=mensaje+"Gato ";
}

if (chkRaton.isSelected()) {
    mensaje=mensaje+"Raton ";
}

etiResultado.setText(mensaje);
```

13. Observa el código. En él se hace lo siguiente:

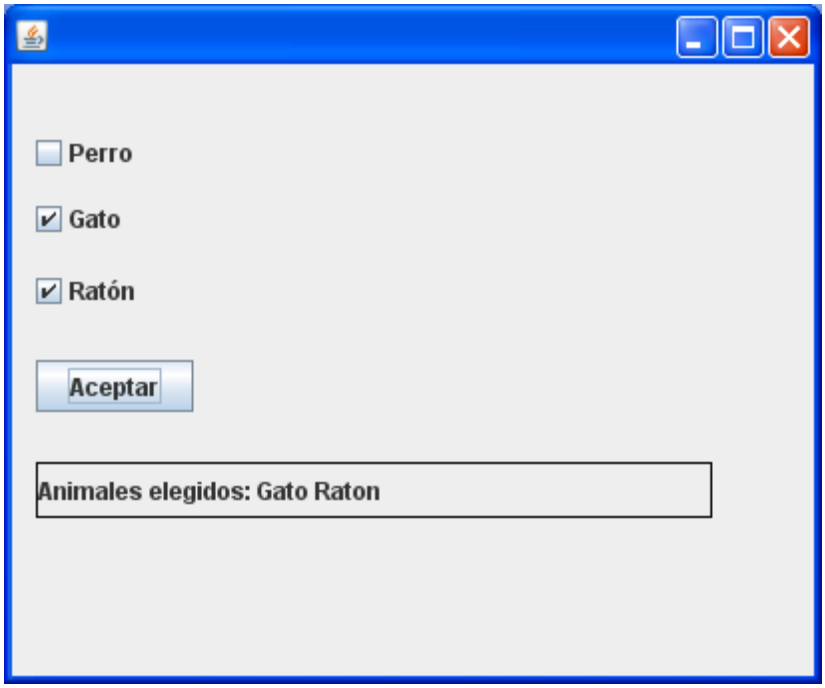
    n. Se crea una variable de cadena llamada *mensaje*.
    o. En esa variable se introduce el texto “Animales elegidos: “
    p. Luego, compruebo si está seleccionada la casilla de verificación chkPerro. Si es así concateno a la cadena mensaje la palabra “Perro”.
    q. Luego compruebo si está seleccionada la casilla de verificación chkGato y hago lo mismo.

F d ió Códi Lib D i i

r. Lo mismo con la casilla chkRaton.
s. Finalmente presento la cadena mensaje en la etiqueta etiResultado.

20. Observa el método isSelected() propio de las casillas de verificación, permiten saber si una casilla está activada o no.

21. Ejecute el programa. Seleccione por ejemplo las casillas Gato y Ratón. Al pulsar Aceptar el resultado debe ser el siguiente:



CONCLUSIÓN

Los cuadros de verificación (JCheckBox) se usan cuando quieres seleccionar varias opciones.

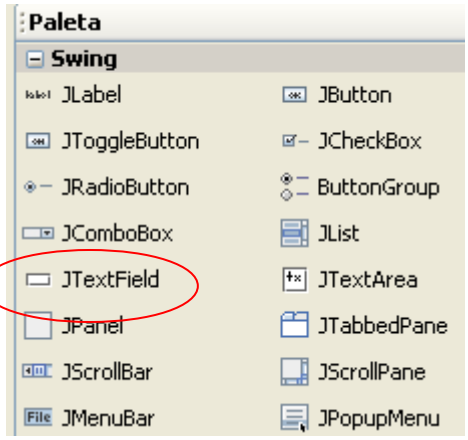
EJERCICIO GUIADO. JAVA: BOTONES DE OPCIÓN

22. Realiza un nuevo proyecto.

23. En la ventana principal debes añadir lo siguiente:

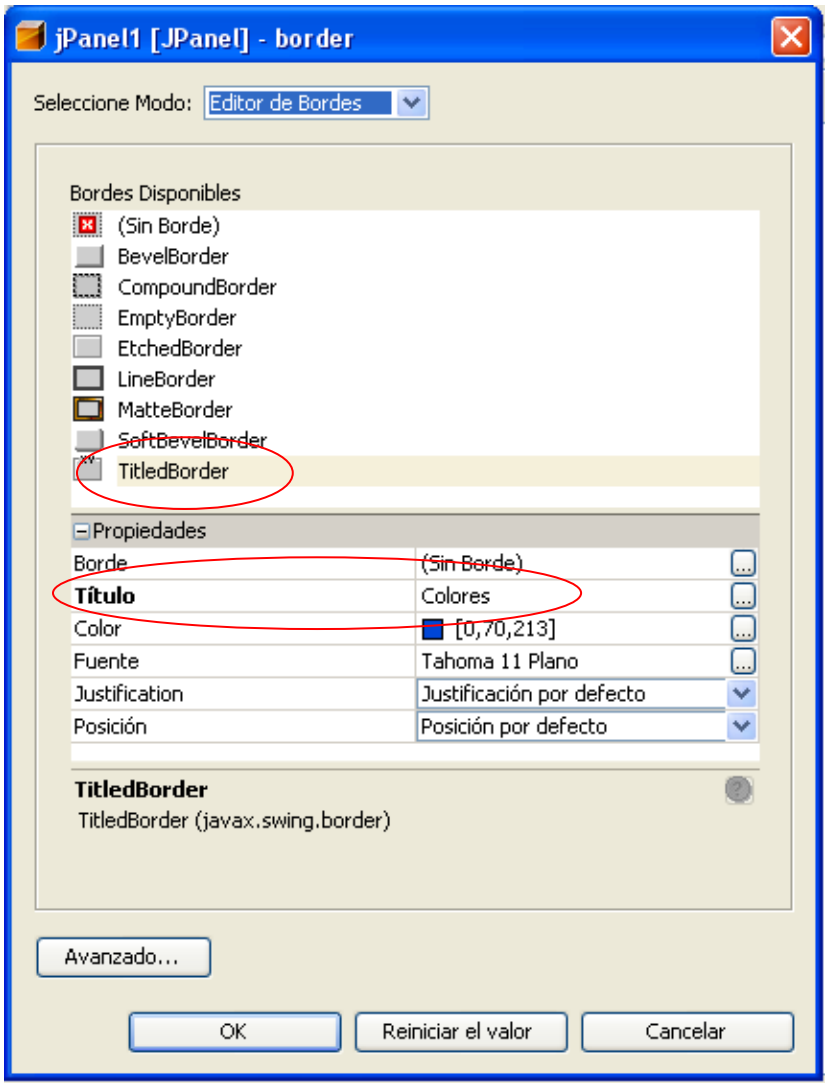
    x. Un botón “Aceptar” llamado btnAceptar.
    y. Una etiqueta con borde llamada etiResultado.

26. Añade un panel. Un panel es una zona rectangular que puede contener elementos (botones, etiquetas, etc) La forma de poner un panel es a través del objeto JPanel.

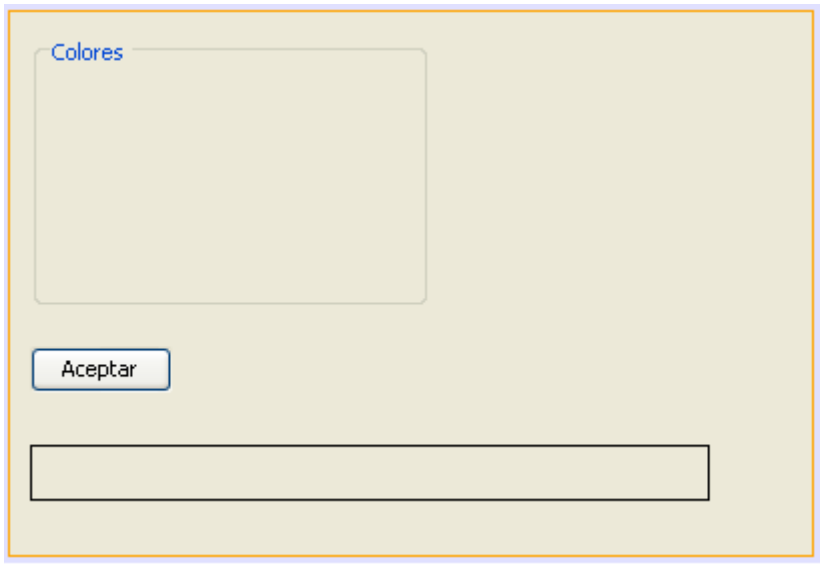


27. Una vez añadido el panel en el JFrame, le pondremos un borde para poder localizarlo fácilmente. Debes hacer lo siguiente:

    bb. Selecciona el panel que has añadido.
    cc. Activa la propiedad Border (botón con tres puntos)
    dd. Busca el tipo de borde llamado TitledBorder (borde con título) y pon el título colores.

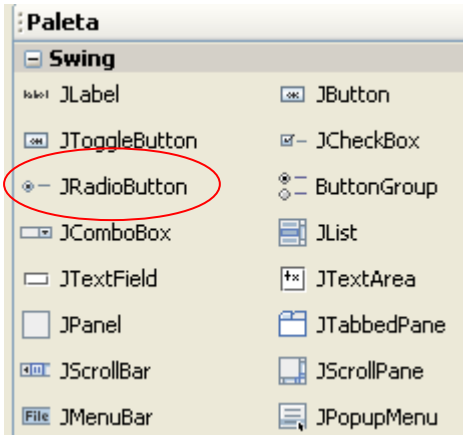


31. Tu ventana debe quedar más o menos así:



F d ió Códi Lib D i i

32. Ahora debes añadir tres botones de opción (botones de radio) dentro del panel. Estos botones son objetos del tipo JRadioButton.



33. Añade tres JRadioButton y cambia el texto de ellos, de forma que aparezca “Rojo”, “Verde” y “Azul”.

34. Debe cambiar el nombre de cada uno de ellos. Se llamarán: optRojo, optVerde, optAzul.

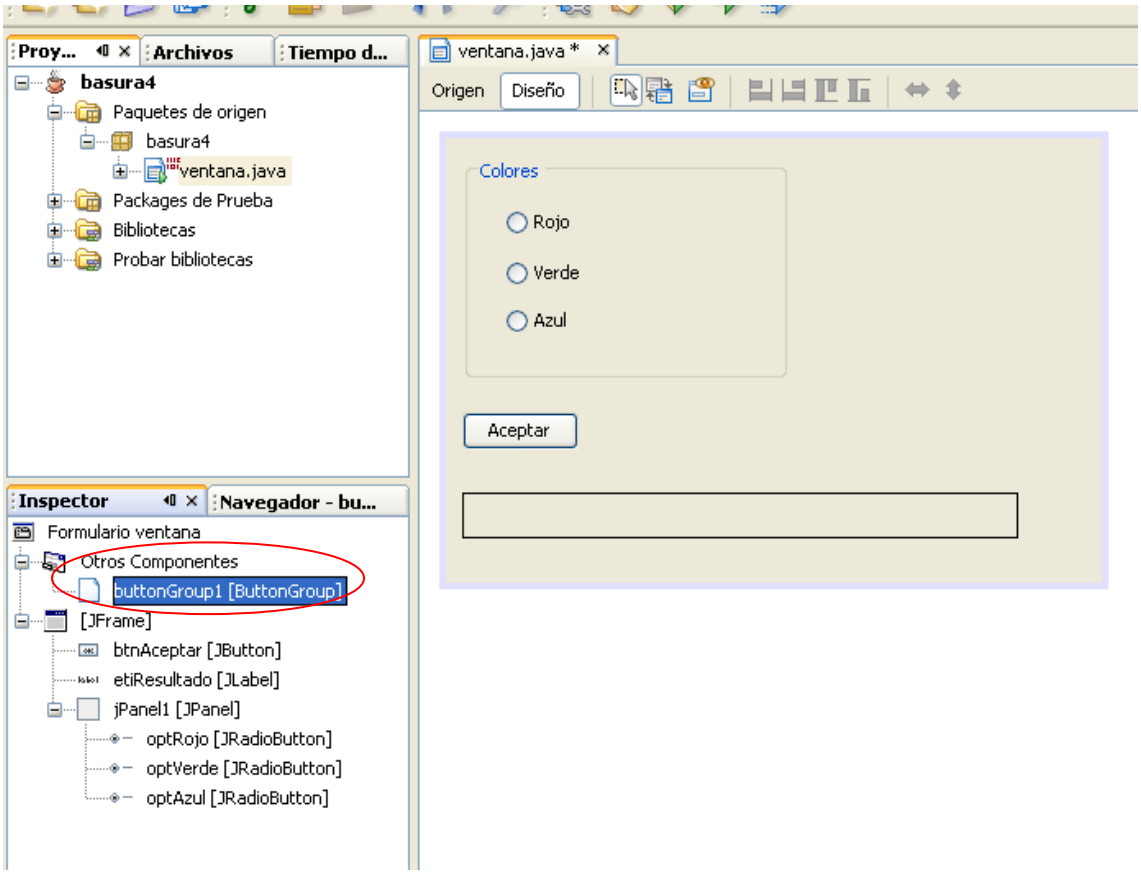
35. La ventana tendrá el siguiente aspecto cuando termine:

36. Si ejecuta el programa, observará que pueden seleccionarse varios colores a la vez. Esto no es interesante, ya que los botones de opción se usan para activar solo una opción entre varias.

F d ió Códi Lib D i i

37. Hay que hacer que solo un botón de opción pueda estar seleccionado a la vez. Para ello, debe añadir un nuevo objeto. Realice los siguientes pasos:

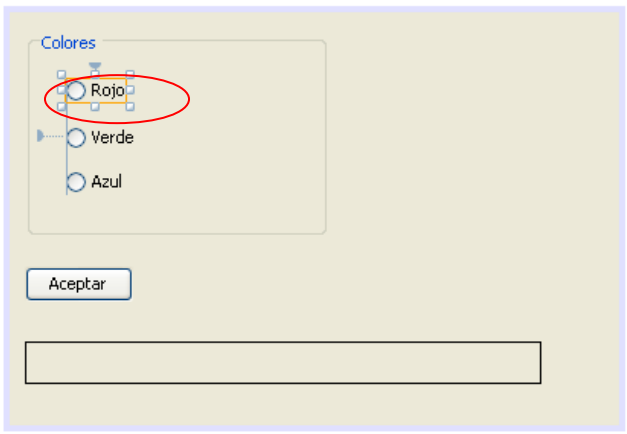
    ll. Añada un objeto del tipo ButtonGroup al formulario. ¡Atención! Este objeto es invisible, y no se verá en el formulario, sin embargo, lo podréis ver en el Inspector, en la parte de “Otros Componentes”:

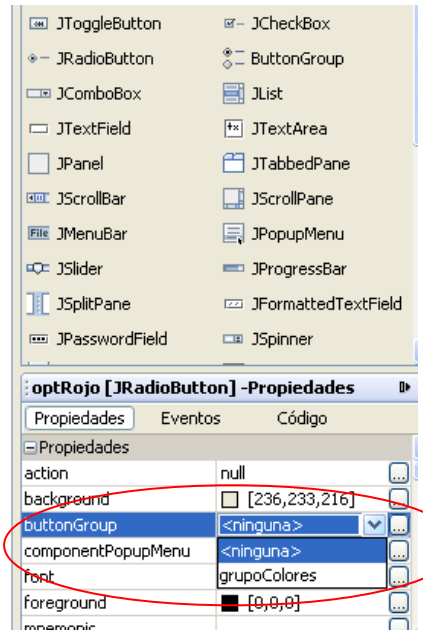


    mm. Tienes que darle un nombre al ButtonGroup. El nombre será “grupoColores”.

    nn. Ahora, hay que conseguir que los tres botones pertenezcan al mismo grupo. Es decir, que pertenezcan al grupo grupoColores.

    oo. Selecciona el botón de opción optRojo y cambia su propiedad buttonGroup, indicando que pertenece al grupo colores (observa la imagen):





pp. Haz lo mismo con los botones optVerde y optAzul.

43. Acabas de asociar los tres botones de opción a un mismo grupo. Esto produce que solo una de las tres opciones pueda estar activada. Pruébelo ejecutando el programa.

44. Ahora interesa que la opción “Rojo” salga activada desde el principio. Una forma de hacer esto es programando en el “Constructor” lo siguiente:

```
optRojo.setSelected(true);
```

El método setSelected hace que se pueda activar o desactivar un botón de opción.

Prueba el programa. Observa como la opción Rojo está activada inicialmente.

45. El programa no está terminado aún. Interesa que cuando el usuario pulse el botón Aceptar, en la etiqueta aparezca el color elegido. Para ello, en el *actionPerformed* del botón Aceptar programe lo siguiente:

```
String mensaje="Color elegido: ";

 if (optRojo.isSelected()) {
     mensaje=mensaje+"Rojo";
 } else if (optVerde.isSelected()) {
     mensaje=mensaje+"Verde";
 } else if (optAzul.isSelected()) {
     mensaje=mensaje+"Azul";
 }
```

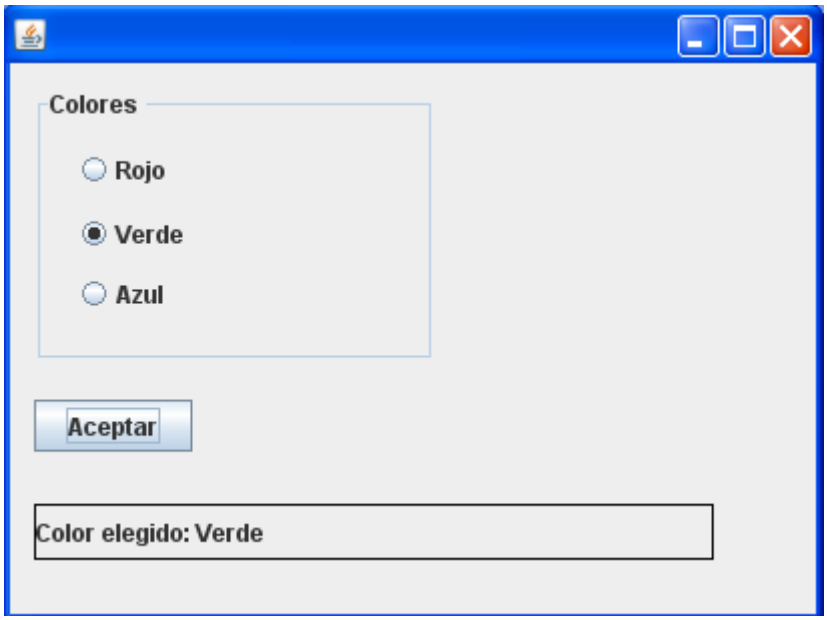
F d ió Códi Lib D i i

```
etiResultado.setText(mensaje);
```

46. Observa el código. En él se hace lo siguiente:

uu. Se crea una variable de cadena llamada *mensaje*.
vv. En esa variable se introduce el texto “Color elegido: “
ww. Luego se comprueba que opción está seleccionada, usando el método isSelected de los botones de opción. Este método te dice si un botón está seleccionado o no.
xx. Según la opción que esté seleccionada, se añade un texto u otro a la cadena *mensaje*.
yy. Finalmente se muestra la cadena *mensaje* en la etiqueta etiResultado.

52. Ejecute el programa. Seleccione por ejemplo la Verde. Al pulsar Aceptar el resultado debe ser el siguiente:



CONCLUSIÓN

Los botones de opción, también llamados botones de radio (JRadioButton) se usan cuando quieres que el usuario pueda elegir una opción de entre varias.

Es interesante que los botones de radio aparezcan dentro de un panel JPanel. Se recomienda colocar un borde al panel.

Es totalmente necesario añadir un objeto del tipo ButtonGroup, y hacer que los botones de radio pertenezcan a dicho grupo. En caso contrario, será posible activar varios botones de opción a la vez.

F d ió Códi Lib D i i

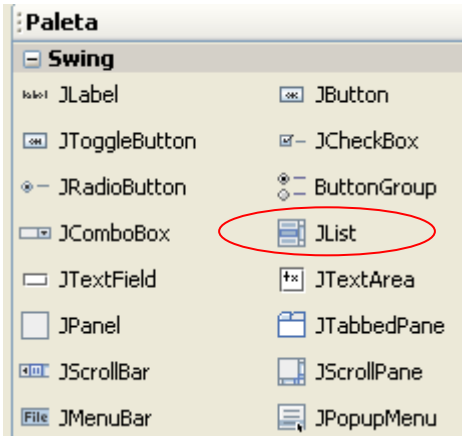
EJERCICIO GUIADO. JAVA: CUADROS DE LISTA

53. Realiza un nuevo proyecto.

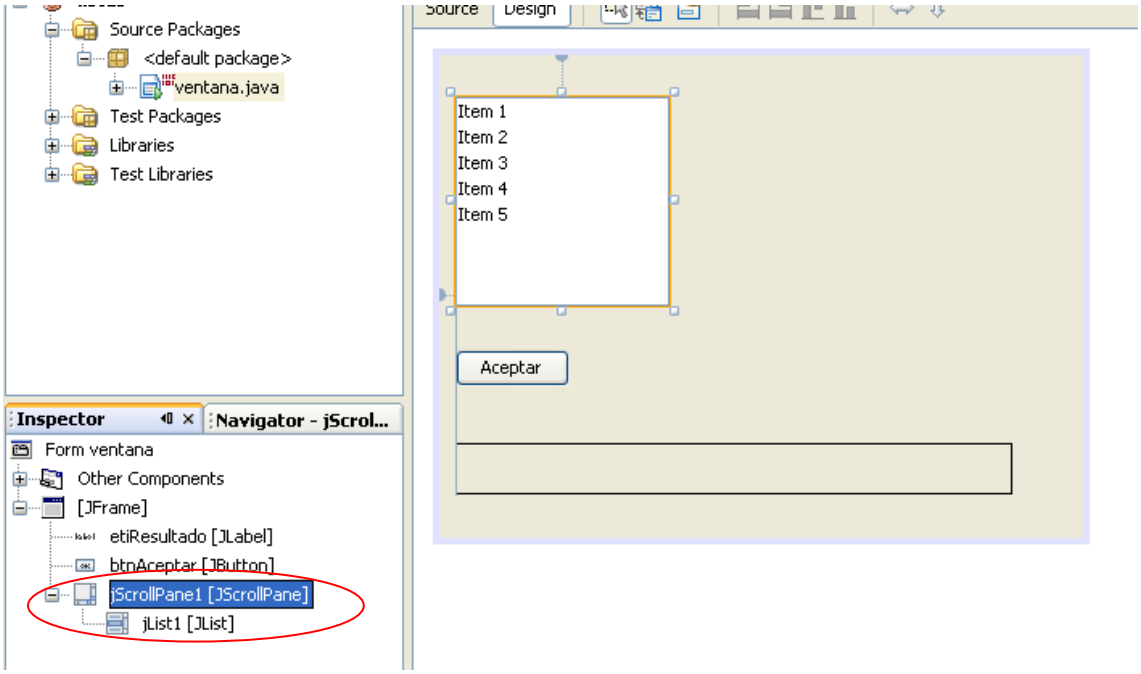
54. En la ventana principal debes añadir lo siguiente:

    ccc. Un botón “Aceptar” llamado btnAceptar.
    ddd. Una etiqueta con borde llamada etiResultado.

57. Añade un cuadro de lista. Los cuadros de listas son objetos JList.



58. Cámbiale el nombre al JList. Ten cuidado, ya que en los JList aparecen siempre dentro de otro objeto llamado jScrollPane. Si miras en el Inspector, verás que al pulsar en el botón + del jScrollPane aparecerá tu JList:

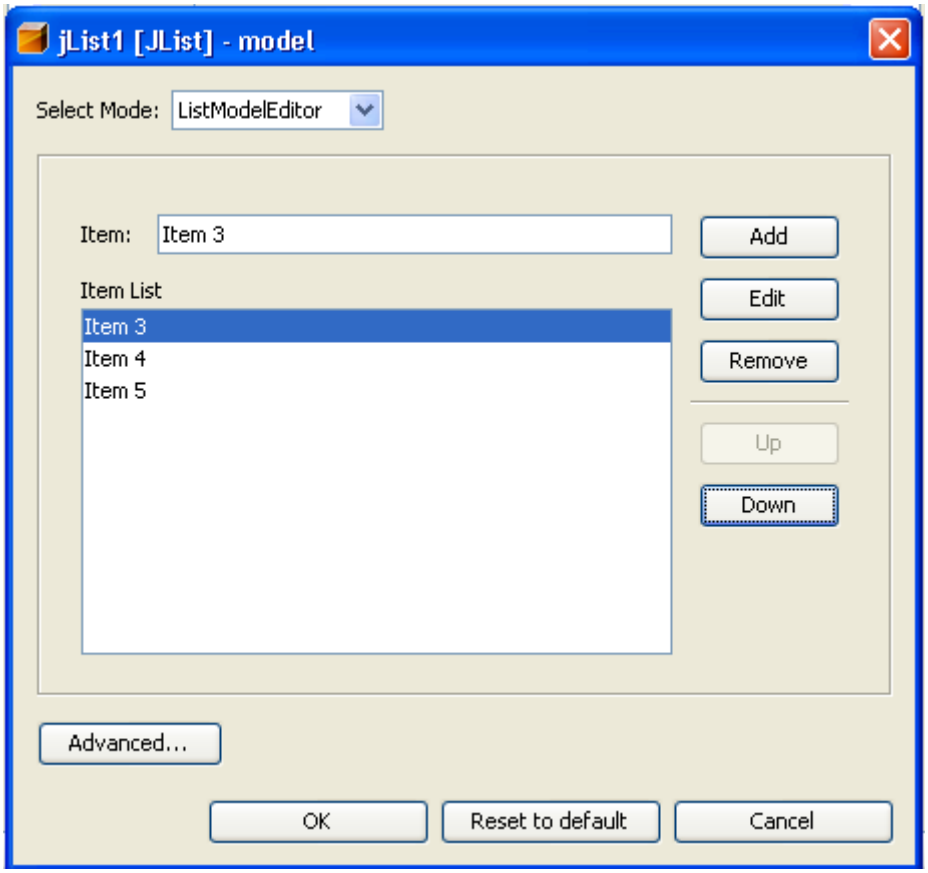


59. Aprovecha para cambiarle el nombre al JList. El nuevo nombre será lstColores.

60. Si te fijas en el JList, consiste en un cuadro que contiene una serie de Items. Estos elementos pueden ser cambiados a través de la propiedad Model del JList.

61. Busca la propiedad Model y haz clic en el botón de los tres puntos. Aparecerá un cuadro de diálogo parecido al siguiente. Solo tienes que seleccionar los elementos que quieras y pulsar el botón “Borrar” (Remove) para eliminarlos de la lista.

62. Puedes añadir elementos escribiéndolos en el cuadro Artículo y luego pulsando el botón “Añadir” (Add).



63. Debes hacer que la lista sea la siguiente:

    Rojo
    Verde
    Azul

64. Ahora programaremos el *actionPerformed* del botón Aceptar. Debes introducir el siguiente código:

```
String mensaje;

mensaje="El color seleccionado es: "+lstColores.getSelectedValue().toString();
etiResultado.setText(mensaje);
```

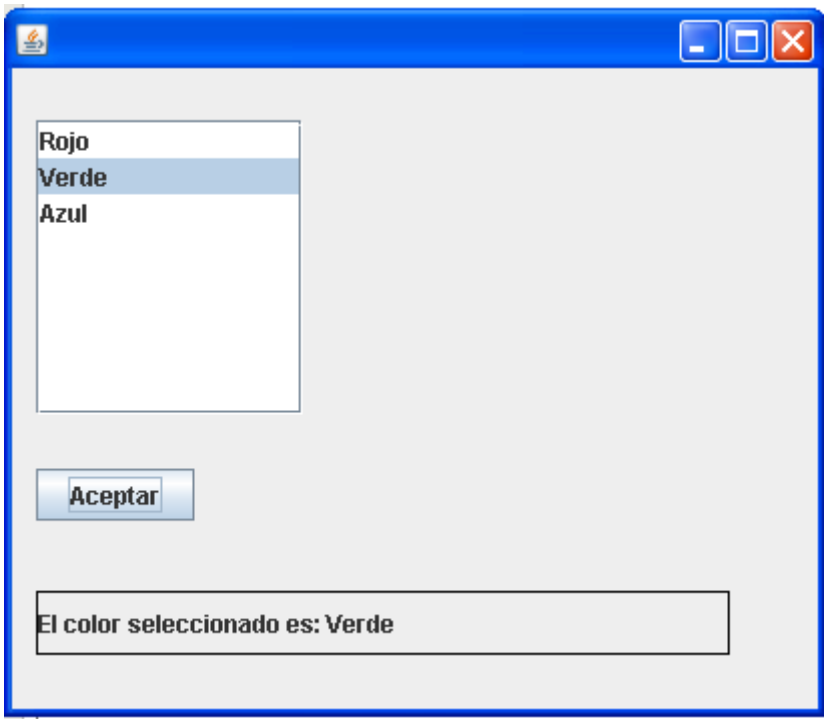
65. Observa el código:

    nnn. Se crea una variable de cadena llamada *mensaje*.
    ooo. Y dentro de esta variable se introduce una concatenación de cadenas.
    ppp. Observa la parte: `lstColores.getSelectedValue()`, esta parte devuelve el valor seleccionado de la lista.
    qqq. Hay que tener en cuenta que este valor no es una cadena, por eso hay que convertirla a cadena añadiendo `.toString()`.

rrr. De esta manera puedes extraer el elemento seleccionado de un cuadro de lista.

sss. Luego simplemente ponemos la cadena mensaje dentro de la etiqueta.

72. Ejecuta el programa y observa su funcionamiento. Por ejemplo, si seleccionas el color verde y pulsas aceptar el resultado será el siguiente:



73. Vamos a mejorar el programa. Puede suceder que el usuario no seleccione ningún valor del cuadro de lista, y sería interesante en este caso que el programa avisara de ello. Cambie el código del botón Aceptar por este otro código:

```
String mensaje;

if (lstColores.getSelectedIndex()==-1) {
   mensaje="No hay un color seleccionado.";
} else {
   mensaje="El color seleccionado es: "+lstColores.getSelectedValue().toString();
}
etiResultado.setText(mensaje);
```

74. Observa el código:

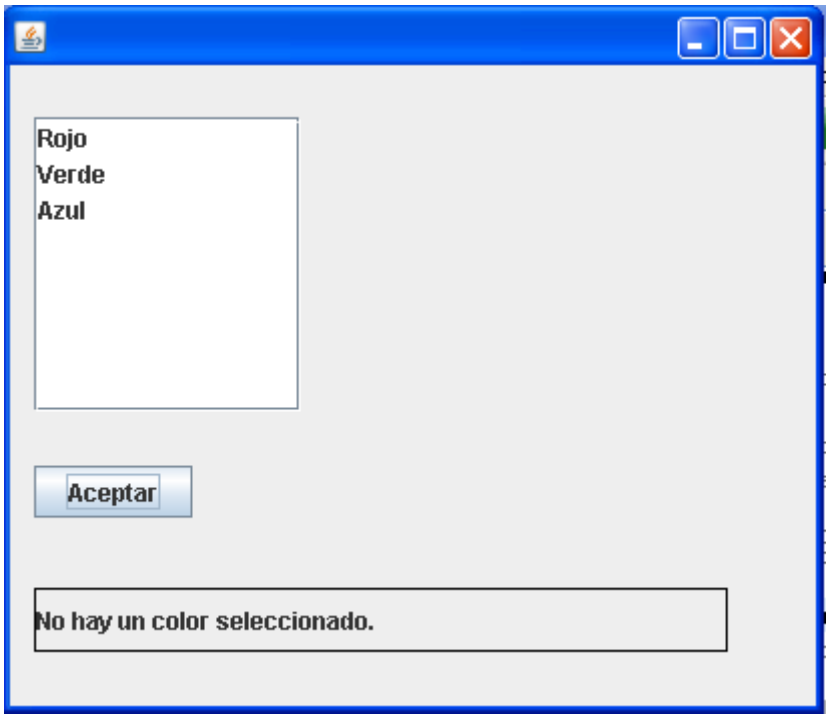
www. El método getSelectedIndex me dice el índice del elemento que está seleccionado.

xxx. Por ejemplo, si está seleccionado el primero el índice es 0, si está seleccionado el segundo el índice es 1, etc.

yyy. Si este método devuelve -1, entonces es señal de que no hay ningún elemento seleccionado.

zzz. Aprovecho esto para mostrar un mensaje indicando lo sucedido.

79. Si ejecuta el programa y pulsa el botón Aceptar sin seleccionar nada el resultado debería ser el siguiente:



80. Se podría haber prescindido del botón aceptar si el código anterior se hubiera puesto en el evento mouseClicked del cuadro de lista en vez de en el *actionPerformed* del botón Aceptar. En este caso, cada vez que se seleccionara un elemento de la lista, automáticamente aparecería el mensaje en la etiqueta.

    Se anima a que realice esta modificación.

CONCLUSIÓN

El objeto JList permite crear cuadros de lista. Estos objetos contienen una serie de elementos que pueden ser seleccionados.

A través del método getSelectedValue se puede obtener el elemento que está seleccionado. (Recuerda convertirlo a cadena con toString)

A través del método getSelectedIndex se puede saber la posición del elemento seleccionado. Si este índice es -1, entonces sabremos que no hay ningún elemento seleccionado.

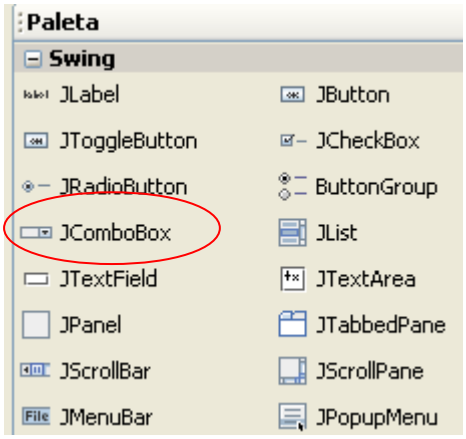
EJERCICIO GUIADO. JAVA: CUADROS COMBINADOS

81. Realiza un nuevo proyecto.

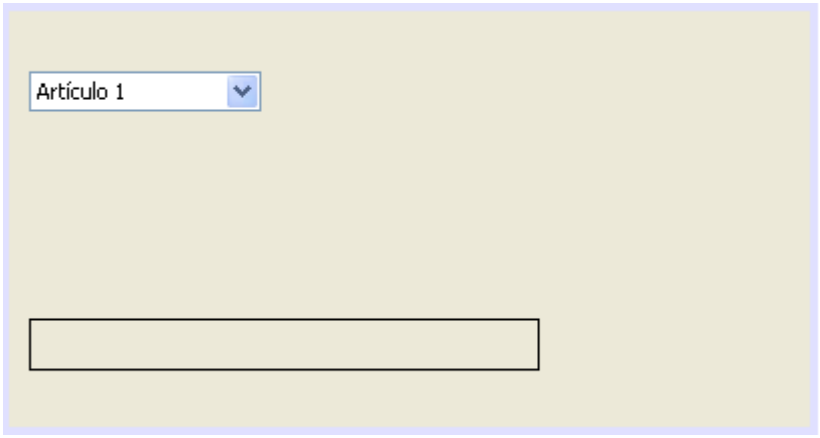
82. En la ventana principal debes añadir lo siguiente:

    eeee. Una etiqueta con borde llamada etiResultado.

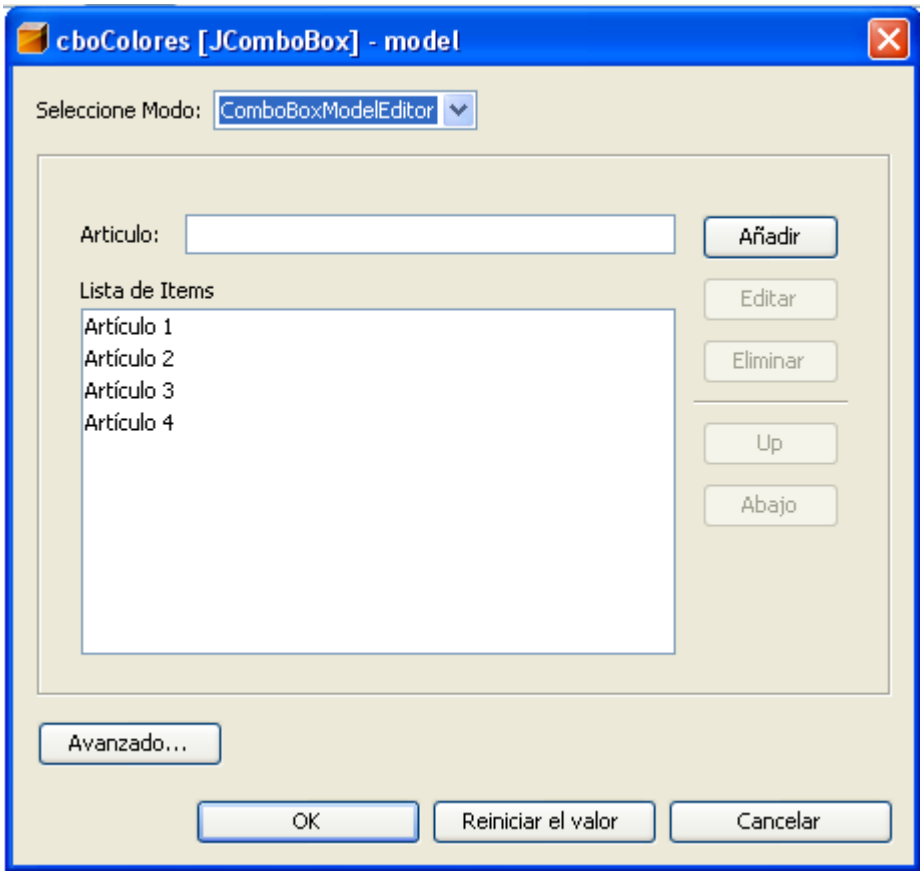
84. Añade un cuadro combinado (combo). Los cuadros combinados son objetos del tipo JComboBox. Básicamente, un combo es una lista desplegable.



85. Cámbiale el nombre al JComboBox. El nombre será cboColores. Tu programa debe tener más o menos este aspecto.



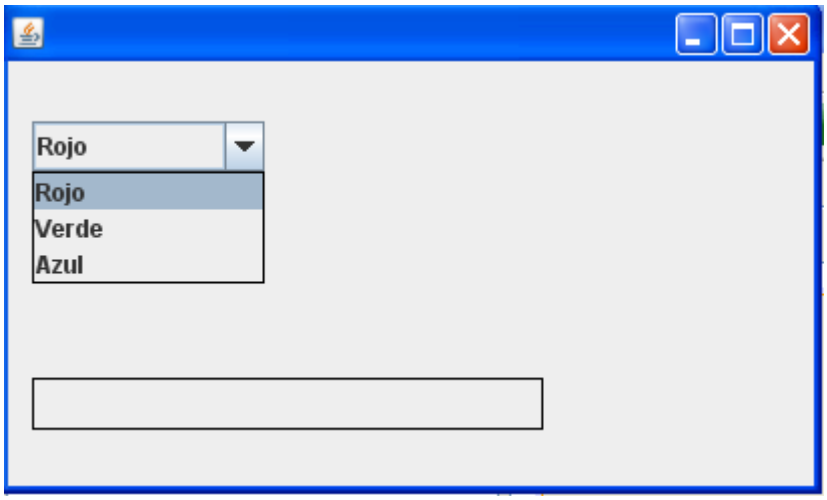
86. Los elementos del cboColores pueden ser cambiados a través de la propiedad Model. Selecciona el combo y activa la propiedad Model (el botoncito con los tres puntos) Aparecerá lo siguiente:



87. Al igual que pasaba con los cuadros de lista, se pueden eliminar los elementos que contiene el combo y añadir elementos propios. Use los botones Añadir y Eliminar para añadir la siguiente lista de elementos:

    Rojo
    Verde
    Azul

88. Ejecuta el programa y observa el funcionamiento del desplegable...



F d ió Códi Lib D i i

89. Vamos a hacer que cuando se elija un elemento del desplegable, en la etiqueta aparezca un mensaje indicando el color elegido.

    Para ello, debes programar el evento *actionPerformed* del combo y añadir el siguiente código:

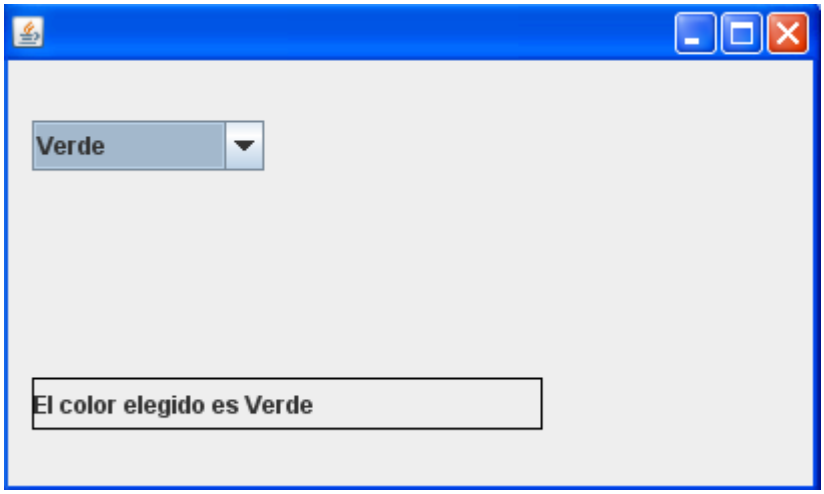
```
String mensaje="El color elegido es ";

mensaje=mensaje+cboColores.getSelectedItem().toString();
etiResultado.setText(mensaje);
```

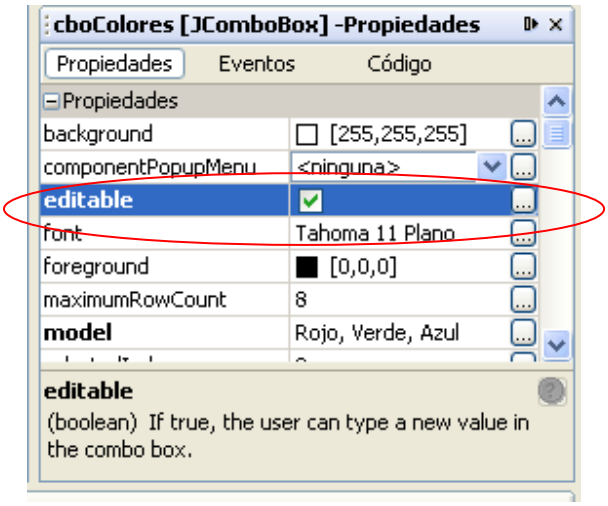
90. Este código hace lo siguiente:

    mmmm. Crea una variable de cadena.
    nnnn. Concatena dentro de ella el mensaje “El color elegido es” con el color seleccionado.
    oooo. Observa el método getSelectedItem, se usa para saber el elemento seleccionado del combo. Es necesario convertirlo a texto con toString.
    pppp. Finalmente se coloca el mensaje en la etiqueta.

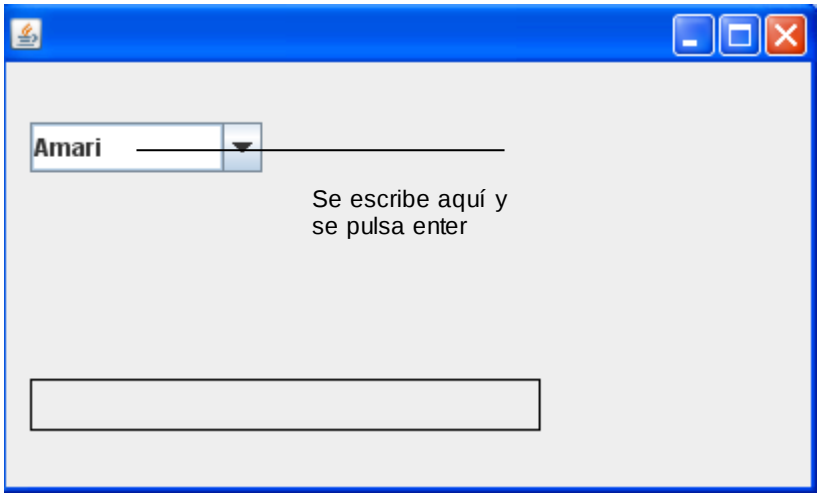
95. Ejecuta el programa y comprueba su funcionamiento. Por ejemplo, si elegimos el color verde, el aspecto del programa será el siguiente:



96. Los cuadros combinados pueden funcionar también como cuadros de texto. Es decir, pueden permitir que se escriba texto dentro de ellos. Para hacer esto, basta con cambiar su propiedad “editable” y activarla.



97. Ejecuta el programa y observa como se puede escribir dentro del combo. Al pulsar Enter, el programa funciona igualmente con el texto escrito.



CONCLUSIÓN

Los combos son listas desplegables donde se puede elegir una de las opciones propuestas.

Los combos pueden funcionar también como cuadros de textos, si se activa la opción editable.

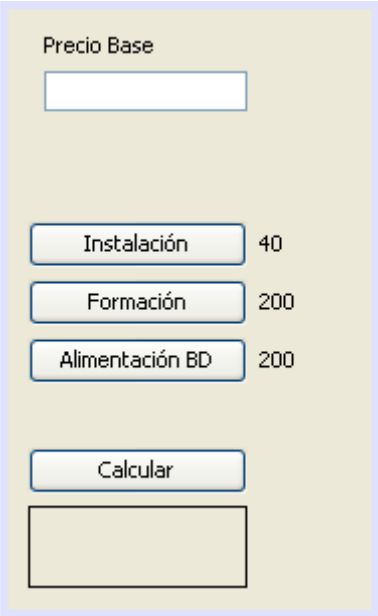
A través del método getSelectedItem se puede extraer la opción seleccionada o el texto escrito en el combo.

F d ió Códi Lib D i i

EJERCICIO GUIADO. JAVA: TOGGLEBUTTONS

98. Realiza un nuevo proyecto.

99. Crearás una ventana como la que sigue teniendo en cuenta lo siguiente:



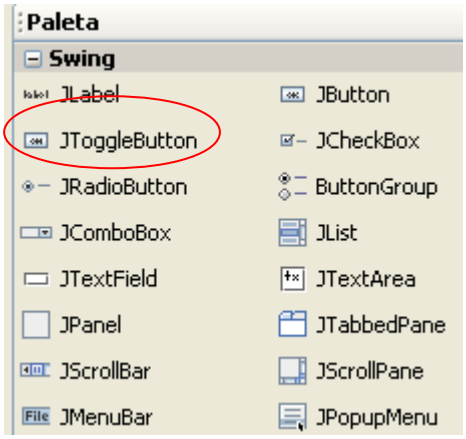
vvvv. Se añadirá una etiqueta con el texto "Precio Base". No hace falta cambiarle su nombre.

wwww. Se añadirá un cuadro de texto llamado txtPrecioBase.

xxxx. Se creará un botón "Calcular", llamado btnCalcular.

yyyy. Se creará una etiqueta vacía y con borde llamada etiTotal. Use la propiedad *font* de esta etiqueta para hacer que el texto tenga un mayor tamaño.

zzzz. Debes añadir también tres botones, con el texto "Instalación", "Formación" y "Alimentación BD" respectivamente.



Estos botones no son botones normales, son botones del tipo JToggleButton. Usa este tipo de objeto para crearlos.

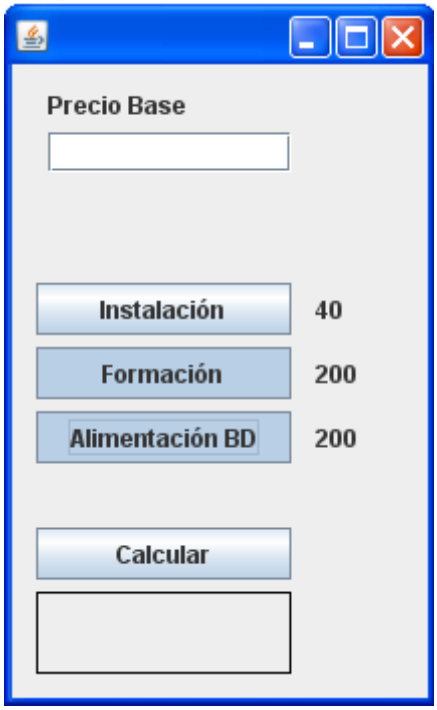
Estos botones, se diferencian de los botones normales en que se quedan pulsados cuando se hace un clic sobre ellos, y no vuelven a su estado normal hasta que no se vuelve a hacer clic sobre ellos.

Los tres botones se llamarán respectivamente: tbtnInstalacion, tbtnFormacion, tbtnAlimentacionBD.

aaaaa. Añade finalmente tres etiquetas conteniendo los números 40, 200, 200. La primera se llamará etiPrecioInstalacion, la segunda etiPrecioFormacion y la tercera etiPrecioAlimentacionBD.

106. Prueba el programa y comprueba el funcionamiento de los botones JToggleButton:

F d ió Códi Lib D i i



Observa como al pulsar los JToggledButton estos se quedan pulsados.

Si se vuelven a activar se "*despulsan*".

107. Se pretende que el programa funcione de la siguiente forma:

ddddd. El usuario introducirá un precio base para el servicio que se vende.
eeeee. A continuación, si el cliente quiere la instalación, activará el botón Instalación.
fffff. Si el cliente quiere la formación, activará el botón Formación.
ggggg. Si el cliente quiere la Alimentación de Base de Datos, activará el botón Alimentación BD.
hhhhh. Ten en cuenta que el cliente puede querer una o varias de las opciones indicadas.
iiiii. Finalmente se pulsará el botón calcular y se calculará el precio total. Este precio se calcula de la siguiente forma:

Precio Total = Precio Base + Precio Extras.

El precio de los Extras dependerá de las opciones elegidas por el usuario. Por ejemplo, si el usuario quiere Instalación y Formación, los extras costarán 240 euros.

114. Así pues, se programará el *actionPerformed* del botón Calcular de la siguiente forma:

```
double precio_base;
double precio_instal; //precio instalación
double precio_for; //precio formacion
double precio_ali; //precio alimentacion

//Recojo datos desde la ventana:

precio_base = Double.parseDouble(txtPrecioBase.getText());
precio_instal = Double.parseDouble(etiPrecioInstalacion.getText());
precio_for = Double.parseDouble(etiPrecioFormacion.getText());
precio_ali = Double.parseDouble(etiPrecioAlimentacionBD.getText());

//Ahora que tengo los datos, puedo hacer cálculos.

//Al precio base se le van añadiendo precio de extras
//según estén o no activados los JToggleButtons

double precio_total;

precio_total = precio_base;

if (tbtnInstalacion.isSelected()) { //Si se seleccionó instalación
   precio_total = precio_total+precio_instal;
}

if (tbtnFormacion.isSelected()) { //Si se seleccionó formación
   precio_total = precio_total+precio_for;
}

if (tbtnAlimentacionBD.isSelected()) { //Si se seleccionó Alimentación BD
   precio_total = precio_total+precio_ali;
}


//Finalmente pongo el resultado en la etiqueta
etiTotal.setText(precio_total+" €");
```

115. Veamos una explicación del código:

lllll. Primero se crean variables doubles (ya que se admitirán decimales) para poder hacer los cálculos.

mmmmm. Se extraerán los datos de la ventana y se almacenarán en dichas variables. Para ello, hay que convertir desde cadena a double:

```
precio_base = Double.parseDouble(txtPrecioBase.getText());
precio_instal = Double.parseDouble(etiPrecioInstalacion.getText());
precio_for = Double.parseDouble(etiPrecioFormacion.getText());
precio_ali = Double.parseDouble(etiPrecioAlimentacionBD.getText());
```

nnnnn. Una vez obtenidos los datos en forma numérica, ya se pueden hacer cálculos con ellos. El programa declara una nueva variable *precio_total* donde se introducirá el resultado. En primer lugar se introduce en esta variable el precio base.

```
double precio_total;

precio_total = precio_base;
```

ooooo. A continuación se le suma al precio_total los precios de los extras si el botón correspondiente está seleccionado. Esto se hace a través de if. Por ejemplo, para sumar el extra por instalación:

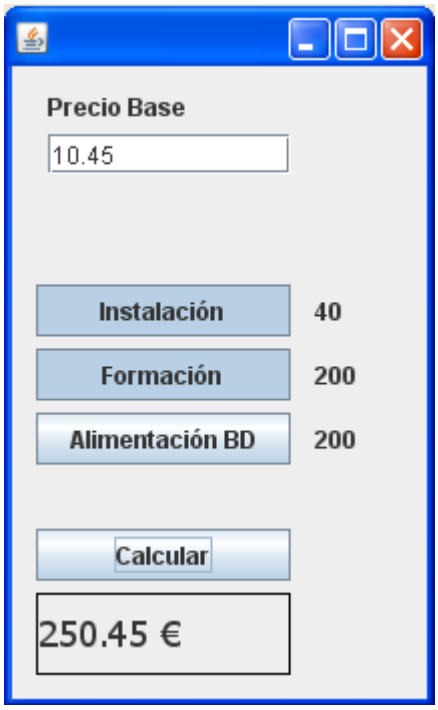
```
if (tbtnInstalacion.isSelected()) { //Si se seleccionó instalación
   precio_total = precio_total+precio_instal;
}
```

Esto significa: “Si el botón instalación está seleccionado, añade al precio total el precio por instalación”

Observa el uso del método *isSelected* para saber si el botón tbtnInstalacion ha sido seleccionado.

ppppp. Finalmente el resultado se muestra en la etiqueta de total.

121. Comprueba el funcionamiento del programa…



Introduce una cantidad (usa el punto para los decimales)

Selecciona los extras que desees.

Pulsa Calcular y obtendrás el resultado.

122. Supongamos que normalmente (en el 90 por ciento de los casos) la instalación es solicitada por el usuario. Podría ser interesante que el botón Instalación ya saliera activado al ejecutarse el programa. Para ello, añade en el *Constructor* la siguiente línea.

```
tbtnInstalacion.setSelected(true);
```

Esta línea usa el método *setSelected* para activar al botón tbtnInstalación.

Comprueba esto ejecutando el programa.

CONCLUSIÓN

Los JToggleButton son botones que pueden quedarse pulsados.

A través del método isSelected podemos saber si un JToggleButton está seleccionado.

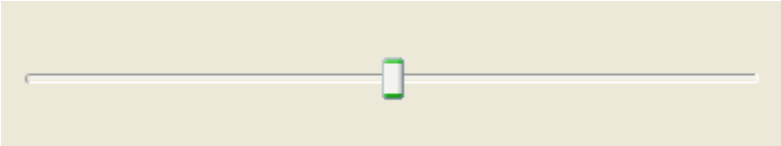
También puedes usar el método setSelected para seleccionar o no un botón de este tipo.

Realmente, estos botones no suelen ser muy usados, ya que pueden ser sustituidos por Cuadros de Verificación (JCheckBox) que son más conocidos.

EJERCICIO GUIADO. JAVA: SLIDERS

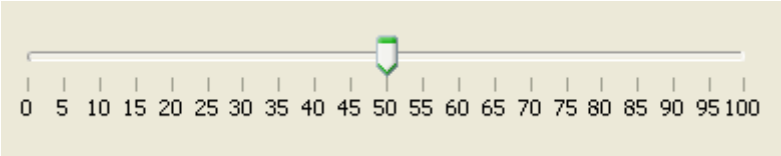
Introducción a los JSliders

La clase JSlider permite crear objetos como el siguiente:

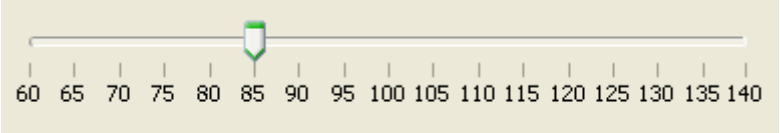


Estos elementos tienen un pequeño recuadro que se puede arrastrar a derecha o izquierda. Según la posición del recuadro, el JSlider tendrá un valor concreto.

El JSlider se puede configurar para que muestre los distintos valores que puede tomar:



También se puede configurar de forma que los valores mínimo y máximo sean distintos:

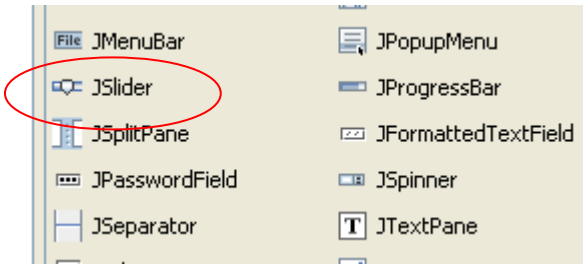


El valor que tiene un JSlider es el valor al que apunta el recuadro del JSlider. En la imagen anterior, el JSlider tiene un valor de 85.

Se verá a continuación las características más interesantes de los JSlider y como programarlos.

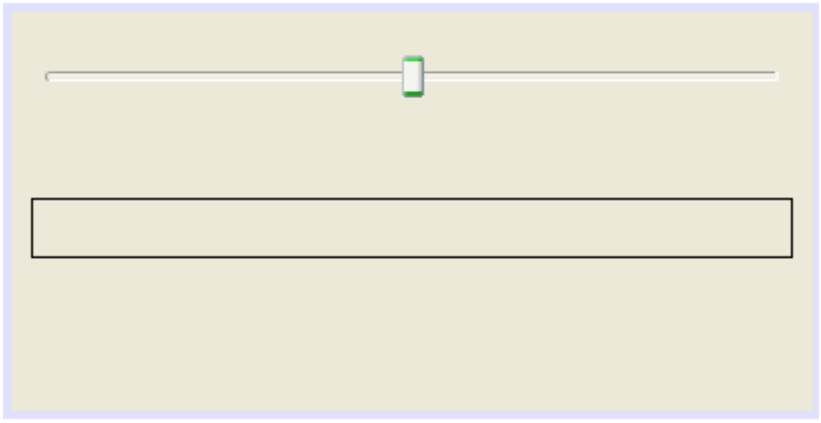
Ejercicio guiado

- Crea un nuevo proyecto.
- Añade en él un JSLider. Su nombre será `slDeslizador`.



- Añade una etiqueta con borde. Su nombre será *etiValor*.

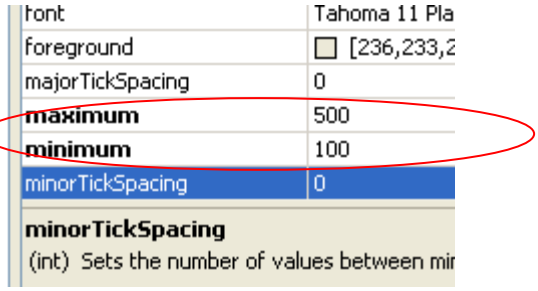
- La ventana tendrá el siguiente aspecto:



- Un JSlider tiene un valor mínimo y un valor máximo. El valor mínimo es el valor que tiene cuando el recuadro está pegado a la parte izquierda, mientras que el valor máximo es el valor que tiene cuando el recuadro está pegado a la parte derecha.

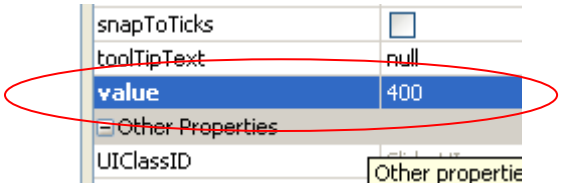
  El valor mínimo y máximo del JSlider se puede cambiar. Busca las propiedades *maximum* y *minimum* del JSlider y asigna los siguientes valores:

  Máximo: 500
  Mínimo: 100

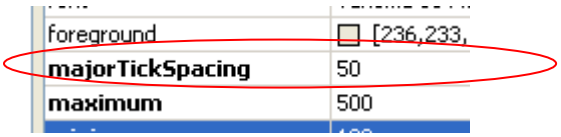


F d ió Códi Lib D i i

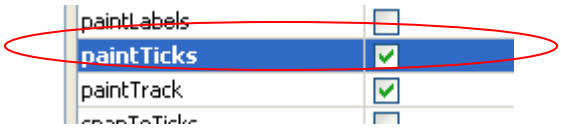
- Se puede asignar un valor inicial al JSlider a través de su propiedad *value*. Busque esta propiedad y asigne un valor de 400. Observe donde se sitúa el recuadro del JSlider.

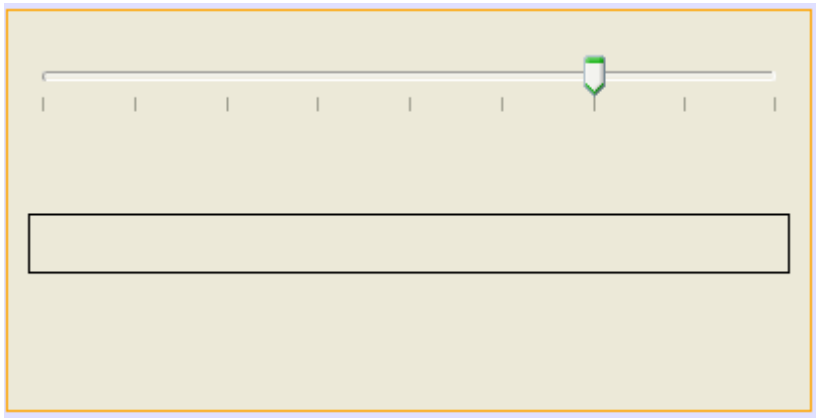


- Se puede mejorar el JSlider definiendo unas divisiones (medidas) Por ejemplo, haremos que cada 50 unidades aparezca una división. Para ello use la propiedad *majorTickSpacing* y asigne un 50.



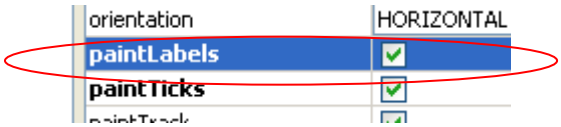
- Esto, en realidad, no produce ningún cambio en el JSlider. Para que las divisiones se vean, es necesario que active también la propiedad *paintTicks*. Esta propiedad pintará divisiones en el JSlider:





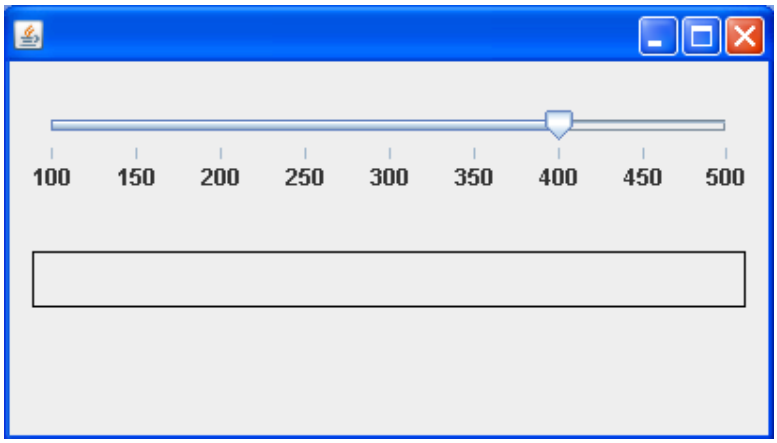
Medidas cada 50 unidades

- Aún se puede mejorar la presentación del JSlider, si hacemos que aparezca el valor de cada división. Para ello debes activar la propiedad *paintLabel*.



F d ió Códi Lib D i i

- Ejecuta el programa para ver el funcionamiento del Deslizador y su aspecto. Debe ser parecido al siguiente:



- Bien. Ahora se pretende que cuando el usuario arrastre el deslizador, en la etiqueta aparezca el valor correspondiente. Para ello tendrá que programar el evento *stateChanged* del JSlider.

  El evento *stateChanged* sucede cuando el usuario arrastra el recuadro del deslizador.

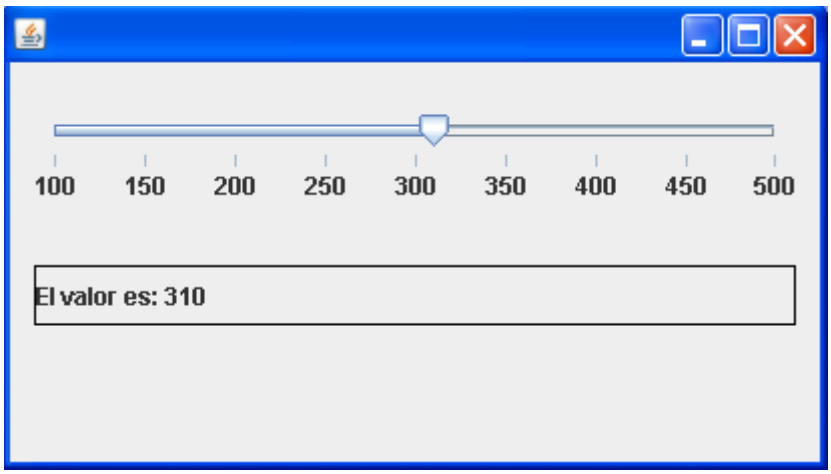
  En este evento programe lo siguiente:

```
etiValor.setText("El valor es: "+slDeslizador.getValue());
```

- Ejecute el programa y observe lo que sucede cuando arrastra el deslizador.

- La explicación del código es la siguiente:

  - El método *getValue* del deslizador nos devuelve el valor que tiene actualmente el deslizador.
  - Este valor es concatenado a la cadena “El valor es:” y es mostrado en la etiqueta a través del conocido *setText*.



Movemos aquí.

Y aparece el valor correspondiente aquí.

F d ió Códi Lib D i i

- A continuación se mencionan otras propiedades interesantes de los JSlider que puedes probar por tu cuenta:

  orientation
  Permite cambiar la orientación del JSlider. Podrías por ejemplo hacer que el JSlider estuviera en vertical.

  minorTickSpacing
  Permite asignar subdivisiones a las divisiones ya asignadas. Prueba por ejemplo a asignar un 10 a esta propiedad y ejecuta el programa. Observa las divisiones del JSlider.

  snapToTicks
  Cuando esta propiedad está activada, no podrás colocar el deslizador entre dos divisiones. Es decir, el deslizador siempre estará situado sobre una de las divisiones. Prueba a activarla.

  paintTrack
  Esta propiedad permite pintar o no la línea sobre la que se desliza el JSlider. Prueba a desactivarla.

CONCLUSIÓN

Los JSliders son objetos "deslizadores". Permiten elegir un valor arrastrando un pequeño recuadro de derecha a izquierda o viceversa.

El valor de un JSliders puede ser obtenido a través de su método *getValue*.

Si quieres programar el cambio (el arrastre) en el deslizador, tienes que programar el evento llamado *stateChanged*.

EJERCICIO GUIADO. JAVA: SPINNER

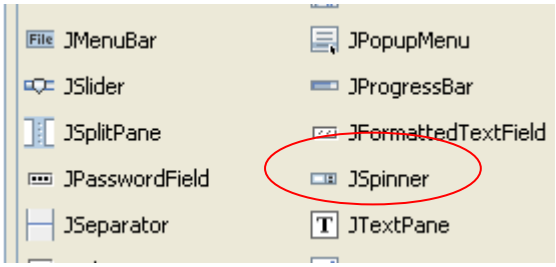
Introducción a los JSpinner

La clase JSpinner permite crear cuadros como el siguiente:

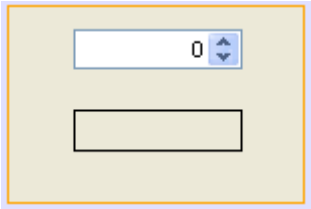
Son elementos muy comunes en los programas. A través de los dos botones triangulares se puede hacer que el valor del cuadro aumente o disminuya. También se puede escribir directamente un valor dentro del cuadro.

Ejercicio guiado

- Crea un nuevo proyecto.
- Añade en él un JSpinner. Su nombre será `spiValor`.



- Añade una etiqueta con borde. Su nombre será *etiValor*.
- La ventana tendrá el siguiente aspecto:



- Interesa que cuando cambie el JSpinner (ya sea porque se pulsaron los botones triangulares o porque se escribió dentro) aparezca el valor correspondiente dentro de la etiqueta. Para ello, tendrá que programar el evento *stateChanged* del JSpinner.

  En el evento *stateChanged* introduzca el siguiente código:

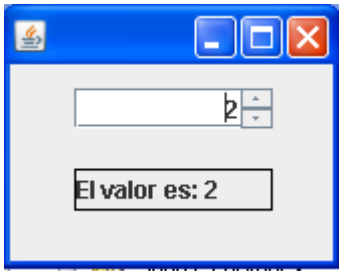
F d ió Códi Lib D i i

```
etiValor.setText("El valor es: "+spiValor.getValue().toString());
```

- Como puedes observar, lo que hace el programa es recoger el valor que tiene el JSpinner a través del método *getValue* y luego se lo asigna a la etiqueta con el clásico *setText*. (Es muy parecido a los deslizadores)

  Debes tener en cuenta que el valor devuelto no es un número ni una cadena, así que en el ejemplo se ha usado el método *toString()* para convertirlo a una cadena.

- Prueba el programa y observa su funcionamiento:



El usuario modifica el valor del JSpinner...

Y aquí aparece el valor seleccionado.

- Observa como los valores del JSpinner aumentan o disminuyen en 1. Por otro lado, no parece haber un límite para los valores del JSpinner.

  La pregunta ahora es: ¿Se puede modificar el contenido del JSpinner de forma que tenga unos valores concretos? La respuesta es sí. Veamos como hacerlo.

- Entra dentro del código del programa y, dentro del constructor, añade este código debajo de *initComponents*:

```
SpinnerNumberModel nm = new SpinnerNumberModel();
nm.setMaximum(10);
nm.setMinimum(0);
spiValor.setModel(nm);
```

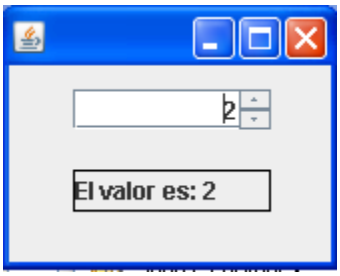
- Este código hace lo siguiente:

  - El JSpinner, al igual que los JList y los JComboBox, es un objeto que contiene otro objeto “modelo”, y es el objeto “modelo” el que contiene los números visualizados en el JSpinner.

  - En el código anterior se crea un “modelo” para el JSpinner, se definen los valores que contendrá, y luego se asigna al JSpinner. Estudiemos las líneas del código.

- La primera línea crea un “modelo” llamado *nm*. Los modelos de los JSpinner son del tipo *SpinnerNumberModel*. Necesitarás incluir el import correspondiente (atento a la bombilla)
- En la segunda línea se define como valor máximo del modelo el 10, a través de un método llamado *setMaximum*.
- En la tercera línea se define como valor mínimo del modelo el 0, a través de un método llamado *setMinimum*.
- Finalmente se asigna el modelo creado al JSpinner.
- Este código, en definitiva, hará que el JSpinner muestre los valores comprendidos entre 0 y 10.

- Prueba el programa y observa los valores que puede tomar el JSpinner.



Ahora los valores están comprendidos entre 0 y 10

- Vamos a añadir otra mejora. Cambie el código del constructor por este otro. (Observa que solo se ha añadido una línea):

```
SpinnerNumberModel nm = new SpinnerNumberModel();
nm.setMaximum(10);
nm.setMinimum(0);
nm.setStepSize(2);
spiValor.setModel(nm);
```

- La línea añadida es:

```
nm.setStepSize(2);
```

Esta línea usa un método del modelo del JSpinner que permite definir el valor de cambio del JSPinner. Dicho de otra forma, esta línea hace que los valores del JSpinner salten de 2 en 2.

- Ejecuta el programa de nuevo y observa como cambian los valores del JSpinner.

- El modelo del JSpinner tiene también un método llamado *setValue* que permite asignar un valor inicial al modelo. Pruebe a usar este método para hacer que el JSpinner muestre desde el principio el valor 4.

F d ió Códi Lib D i i

CONCLUSIÓN

Los JSpinners son objetos que permiten seleccionar un número, ya sea escribiéndolo en el recuadro, o bien a través de dos botones triangulares que permiten aumentar o disminuir el valor actual.

Los JSpinners son objetos con "modelo". Es decir, este objeto contiene a su vez otro objeto "modelo" que es el que realmente contiene los datos.

Datos → Modelo → JSpinner

Para definir el contenido del JSpinner es necesario crear un modelo del tipo SpinnerNumberModel. Se le asigna al modelo los números deseados, y finalmente se une el modelo con el JSpinner.

El objeto modelo del JSpinner permite definir el valor mínimo y el valor máximo, así como el intervalo de aumento de los valores.

EJERCICIO GUIADO. JAVA: SCROLLBARS

Introducción a las JscrollBars (Barras de desplazamiento)

La clase JScrollBar permite crear barras de desplazamiento independientes, como la que se muestra a continuación:

La barra tiene un valor mínimo, que se consigue haciendo que el recuadro de la barra de desplazamiento esté pegado a la parte izquierda.

Valor mínimo

Cuando se pulsa algunos de los botones de la barra de desplazamiento, el valor de la barra se incrementa / decrementa poco a poco. A este incremento / decremento lo llamaremos *incremento unitario*.

Decrementa el valor poco a poco (*incremento unitario*)

Incrementa el valor poco a poco (*incremento unitario*)

Cuando se pulsa directamente sobre la barra, el valor de la barra se incrementa / decrementa en mayor cantidad. A este incremento / decremento lo llamaremos *incremento en bloque*.

Al pulsar directamente sobre la barra se decrementa en mayor cantidad (*incremento en bloque*)

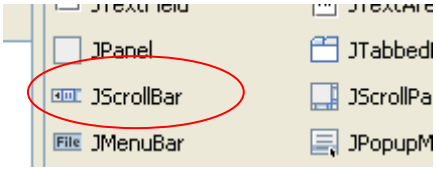
Al pulsar directamente sobre la barra se incremente en mayor cantidad (*incremento en bloque*)

F d ió Códi Lib D i i

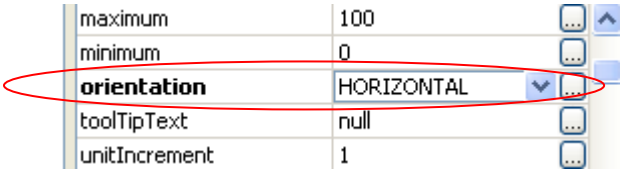
Ejercicio guiado

1. Para comprender mejor el funcionamiento de las barras de desplazamiento se creará un proyecto nuevo.

2. Añade en el proyecto una barra de desplazamiento (JScrollBar) y llámala *desValor*.

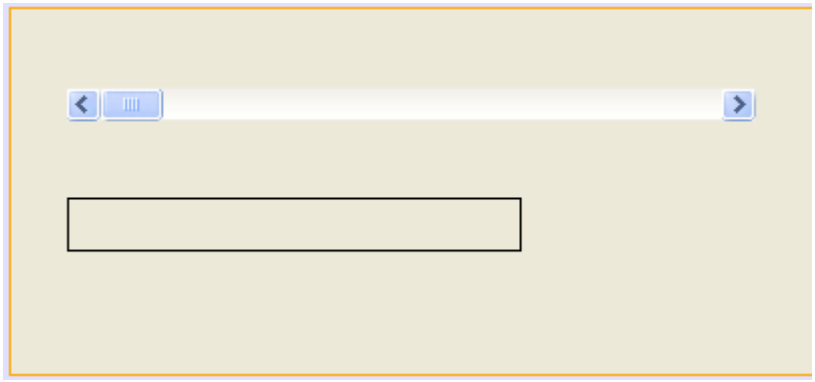


3. La barra de desplazamiento aparecerá en vertical. Use la propiedad de la barra llamada *Orientation* para hacer que la barra aparezca en posición horizontal.



4. Añade también una etiqueta con borde y llámala etiValor.

5. La ventana debe quedar más o menos así:



6. Interesa que cuando el usuario cambie de alguna manera la barra de desplazamiento, en la etiqueta aparezca el valor de la barra.

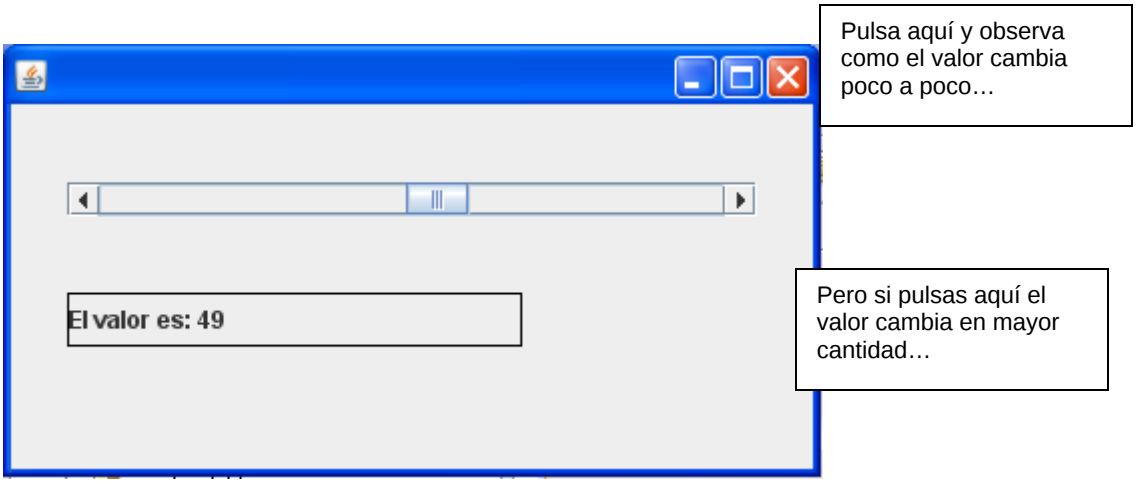
   Para ello, se debe programar el evento *AdjustmentValueChanged* de la barra de desplazamiento.

   En este evento programa lo siguiente:

```
etiValor.setText("El valor es: "+desValor.getValue());
```

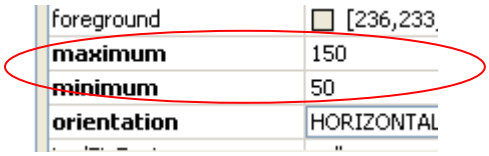
F d ió Códi Lib D i i

7. Como ves, se coloca en la etiqueta el valor de la barra. El valor de la barra se obtiene con el método *getValue*. Ejecuta el programa para ver su funcionamiento.

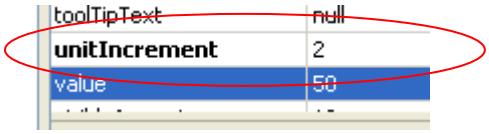


8. Sigamos estudiando el programa. Se pide que cambies las siguientes propiedades de tu barra:

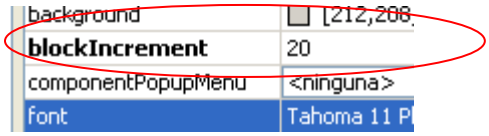
   Minimum – Permite asignar el valor mínimo de la barra. Escribe un 50
   Maximum – Permite asignar el valor máximo de la barra. Escribe un 150



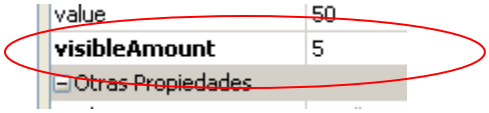
   UnitIncrement – Permite cambiar el *incremento unitario*. Escribe un 2.



   BlockIncrement – Permite cambiar el *incremento en bloque*. Escribe un 20.



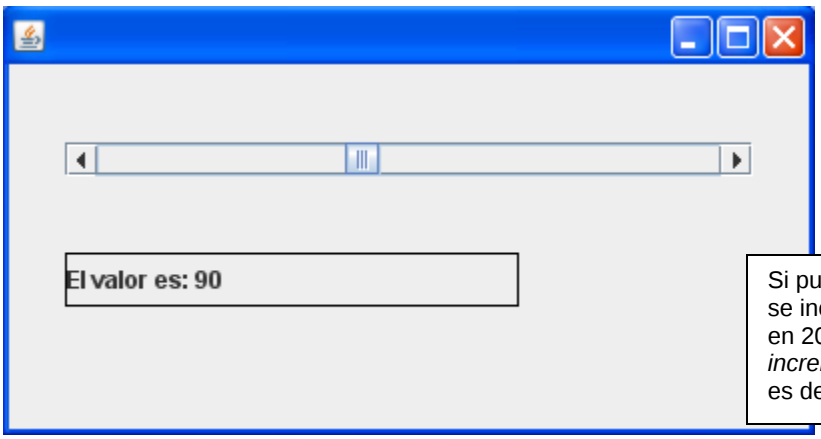
   VisibleAmount – Permite cambiar el ancho del recuadro de la barra. Escribe un 5.



5

F d ió Códi Lib D i i

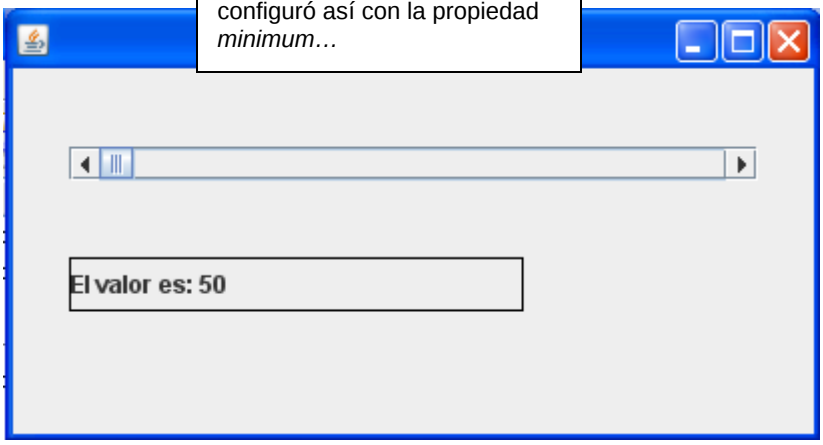
9. Ejecuta ahora el programa y comprueba su funcionamiento:

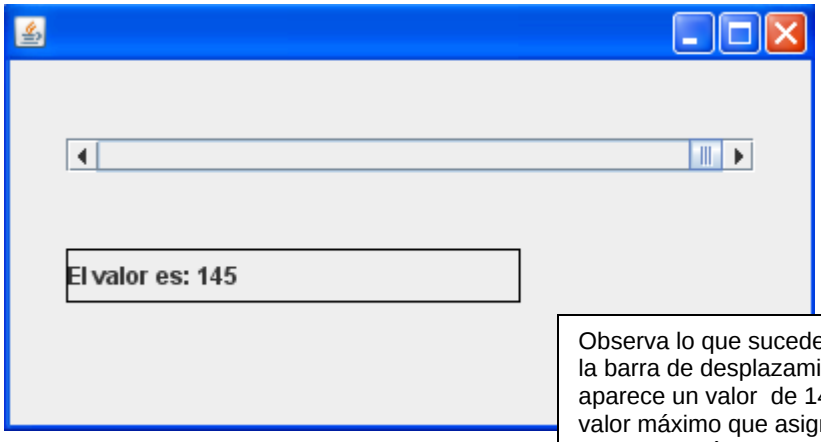


Si pulsas aquí, el valor se incrementa de 2 en 2, ya que el *incremento unitario* se configuró en 2.

Si pulsas aquí, el valor se incrementa de 20 en 20, ya que el *incremento en bloque* es de 20.

Si llevas la barra de desplazamiento al mínimo, su valor será de 50, ya que se configuró así con la propiedad *minimum…*

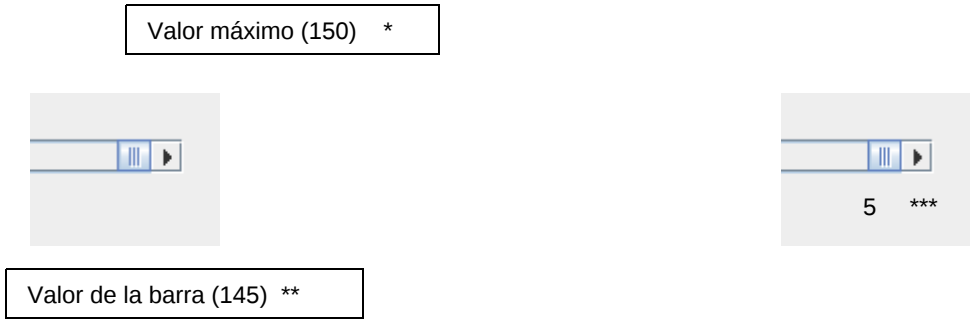




Observa lo que sucede cuando llevas la barra de desplazamiento al máximo: aparece un valor de 145, cuando el valor máximo que asignamos fue de 150 ¿por qué?

F d ió Códi Lib D i i

10. Tal como se ha indicado anteriormente, pasa algo raro con la barra de desplazamiento cuando esta está al máximo. Se esperaba que alcanzara el valor 150, y sin embargo, el valor máximo alcanzado fue de 145. La explicación es la siguiente:



* Nuestra barra tiene un valor máximo de 150.

** Sin embargo, el valor de la barra viene indicado por el lado izquierdo del recuadro interno.

*** Como el recuadro interno tiene un ancho definido a través de la propiedad *VisibleAmount*, el valor máximo que la barra puede alcanzar es de:

Valor = ValorMáximo – Ancho del recuadro.

Es decir,

Valor alcanzable = 150 – 5 = 145

11. A través del método *setValue* de la barra de desplazamiento se puede asignar un valor inicial a la barra. Programe en el constructor de su programa lo necesario para que la barra de desplazamiento tenga un valor de 70 al empezar el programa.

CONCLUSIÓN

Las JScrollBars son barras de desplazamiento independientes. Al igual que los JSliders, las JScrollBars tienen un valor concreto, que puede ser obtenido a través del método *getValue*.

Entre las características programables de una barra de desplazamiento, tenemos las siguientes:

- Valor mínimo (propiedad Minimum)
- Valor máximo (propiedad Maximum)

- **Incremento unitario (propiedad UnitIncrement)**
- **Incremento en bloque (propiedad BlockIncrement)**
- **Tamaño del recuadro de la barra (propiedad VisibleAmount)**

EJERCICIO GUIADO. JAVA: BARRA DE MENUS

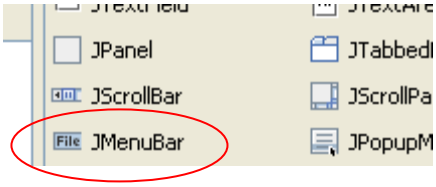
Barras de Menús

La barra de menús nos permitirá acceder a las opciones más importantes del programa. Todo programa de gran envergadura suele tener una barra de menús.

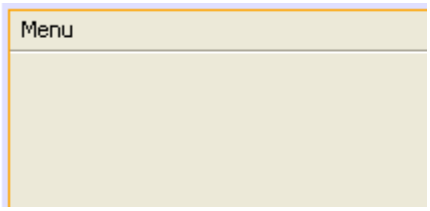
Ejercicio guiado

12. Veamos como añadir una barra de menús a nuestras aplicaciones. En primer lugar, crea un proyecto con el NetBeans.

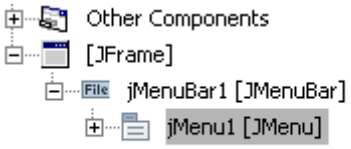
13. Añade a tu ventana un objeto JMenuBar



14. En la parte superior de tu ventana aparecerá esto:



15. En el inspector (parte inferior izquierda) observarás como aparece un objeto JMenuBar, y, dentro de él, un objeto del tipo JMenu. Los objetos JMenu representan las opciones principales contenidas dentro de la barra de menús.

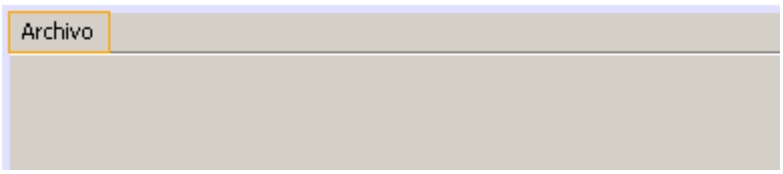


16. Aprovecha el *Inspector* para cambiar el nombre al objeto JMenuBar. Llámalo *barraMenus*.

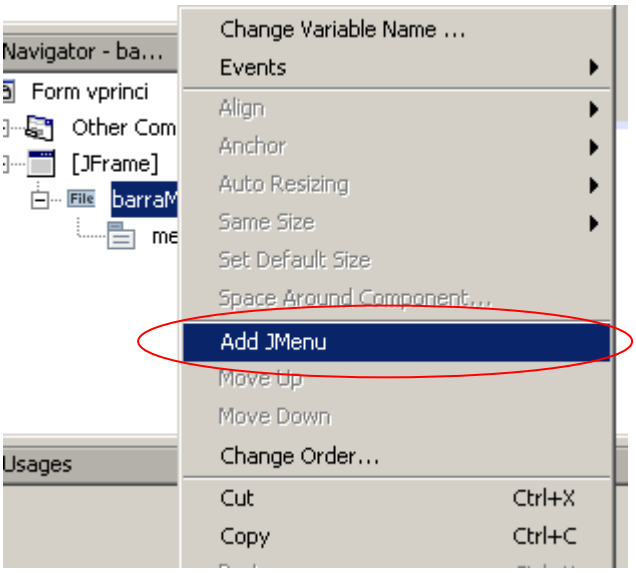
17. Cambia también el nombre al objeto JMenu. Asígnale el nombre *menuArchivo*. El *Inspector* tendrá el siguiente aspecto:

18. Ahora, la única opción de la barra de menús muestra el texto “Menu”. Esto se puede cambiar seleccionándola y cambiando su propiedad *text*. Asígnale el texto “Archivo” a la opción del menú:

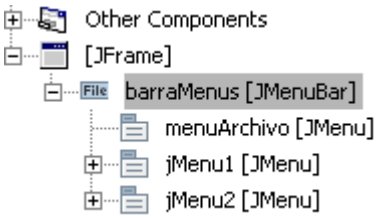
19. Ahora el aspecto de la barra de menús será el siguiente:



20. Puedes añadir más opciones principales a la barra de menús haciendo clic con el derecho sobre el objeto de la barra de menús y activando la opción “Añadir JMenu”.

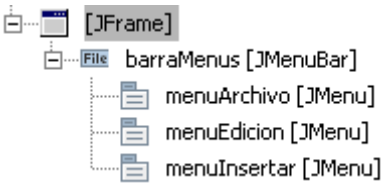


21. Añada dos opciones más a la barra de menús. El inspector debe tener ahora el siguiente aspecto:

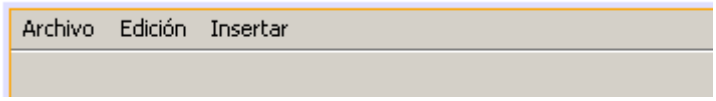


22. Y la barra de menús presentará este otro aspecto:

23. Cambia los nombres de las dos nuevas opciones. Sus nombres serán: menuEdicion y menuInsertar.

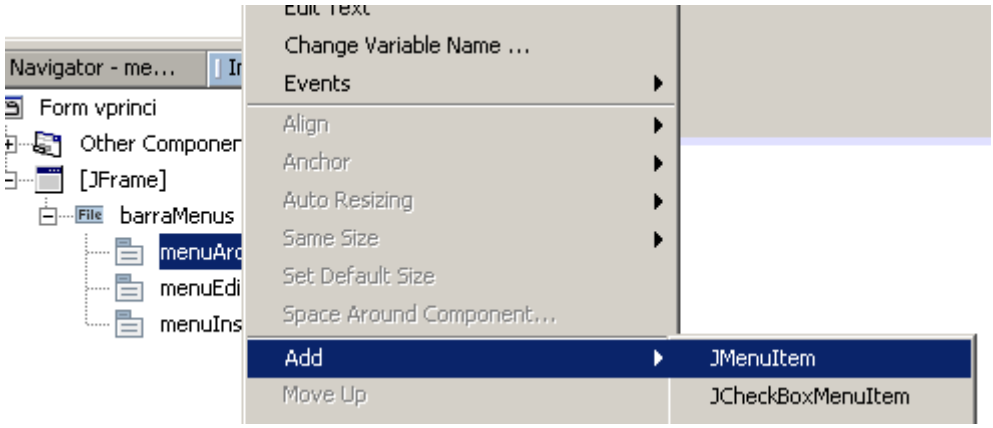


24. Cambia los textos de ambas opciones. Sus textos serán: “Edición” e “Insertar”.



25. Ya tenemos creada la barra de menús (JMenuBar) con sus opciones principales (JMenu). Ahora se tendrán que definir las opciones contenidas en cada opción principal. Por ejemplo, crearemos las opciones contenidas en el menú Archivo.

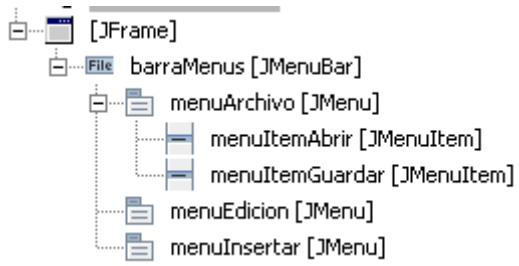
26. Haz clic con el botón derecho sobre el objeto menuArchivo y activa la opción “Añadir – JMenuItem”.



F d ió Códi Lib D i i

Los JMenuItem son objetos que representan las opciones contenidas en los menús desplegables de la barra de menús.

27. Añade un JMenuItem más al menuArchivo y luego cambia el nombre a ambos. Sus nombres serán *menuItemAbrir* y *menuItemGuardar*. El aspecto del *Inspector* será el siguiente:

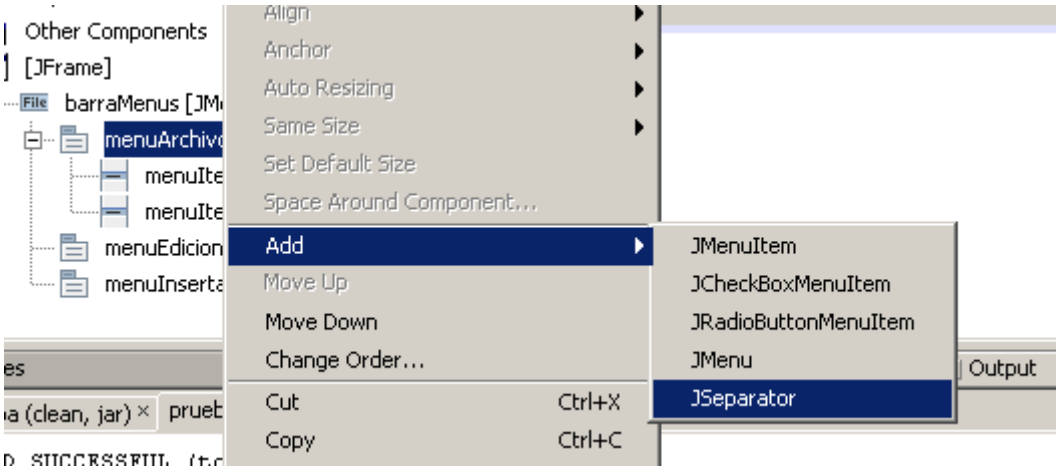


28. Usa ahora la propiedad *Text* de ambos JMenuItem para asignarles un texto. El primero tendrá el texto “Abrir” y el segundo el texto “Guardar”.

29. Ya podemos ejecutar el programa para ver que es lo que se ha conseguido. Use el menú:

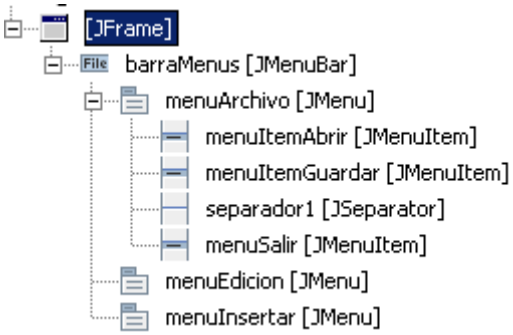
Observa como la opción Archivo se despliega mostrando dos submenús: Abrir y Guardar.

30. Seguiremos añadiendo elementos al menú. Ahora haga clic con el derecho sobre el elemento *menuArchivo* y añada un JSeparator.

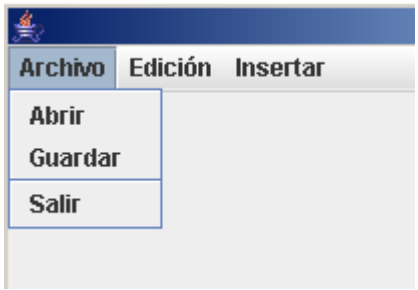


Los JSeparator son objetos que definen una separación entre las opciones de un menú. Cámbiele el nombre y llámelo “separador1”:

31. Añada un nuevo JMenuItem al menú Archivo y ponle el nombre menuSalir. El texto de esta opción será “Salir” (use su propiedad *text*) El aspecto del *Inspector* será el siguiente:



32. Ejecuta el programa y observa el contenido de la opción Archivo del menú:

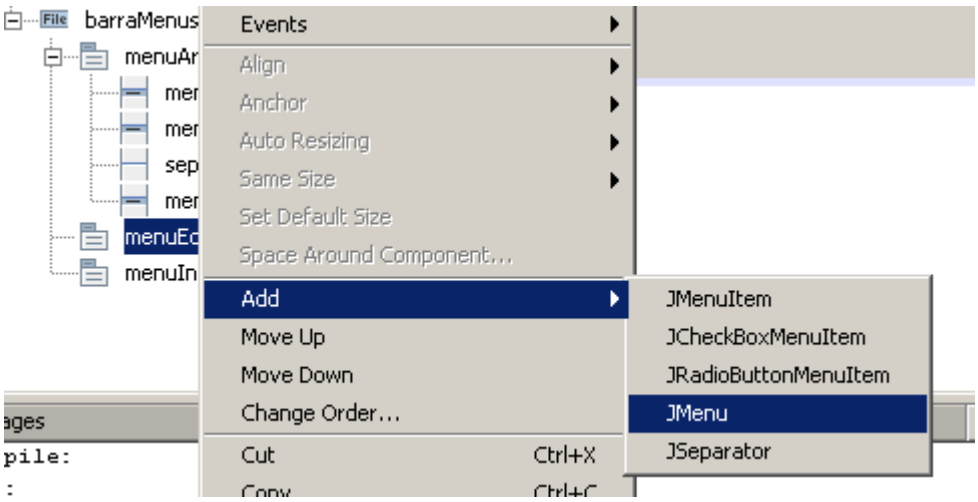


Observa el efecto que produce el separador.

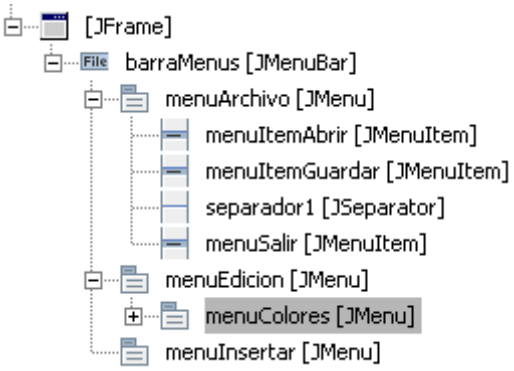
33. Un JMenu representa las opciones principales de la barra de menús. A su vez, un JMenu contiene JMenuItem, que son las opciones contenidas en cada opción principal, y que se ven cuando se despliega el menú.

F d ió Códi Lib D i i

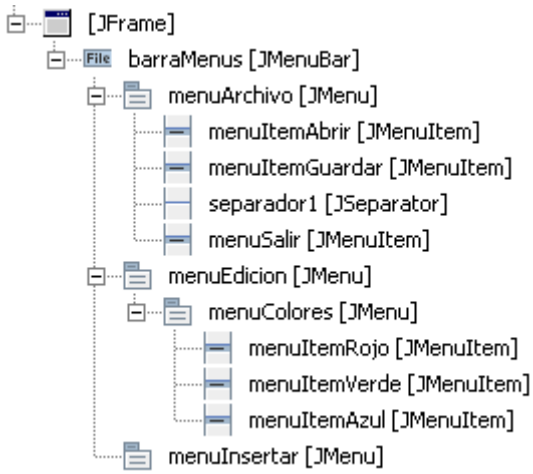
Sin embargo, un JMenu puede contener a otros JMenu, que a su vez contendrán varios JMenuItem. Usando el botón derecho del ratón y la opción “Añadir”, añade un JMenu dentro de menuEdicion:



34. Llama al nuevo JMenu *menuColores* y asignale el texto “Colores”.



35. Ahora añade dentro del *menuColores* tres JMenuItem llamados respectivamente: *menuItemRojo*, *menuItemVerde*, *menuItemAzul*. Sus textos serán “Rojo”, “Verde” y “Azul”.



36. Ejecuta el programa y observa como ha quedado el menú Edición:

La opción Edición (JMenu) contiene una opción Colores (JMenu) que a su vez contiene las opciones Rojo, Verde y Azul (JMenuItems)

37. De nada sirve crear un menú si luego este no reacciona a las pulsaciones del ratón. Cada objeto del menú tiene un evento ActionPerformed que permite programar lo que debe suceder cuando se active dicha opción del menú.

38. Marque en el inspector el objeto menuItemRojo y acceda a su evento ActionPerformed. Dentro de él programe este sencillo código:

```
this.getContentPane().setBackground(Color.RED);
```

Este código cambia el color de fondo de la ventana a rojo.

39. Compruebe el funcionamiento de la opción “Rojo” del menú ejecutando el programa.

40. Programa tu mismo las opciones “Verde” y “Azul”.

CONCLUSIÓN

Las barras de menús son un conjunto de objetos de distinto tipo que se contienen unos a los otros:

La barra en sí está representada por un objeto del tipo JMenuBar.

La barra contiene opciones principales, representadas por objetos JMenu.

Las opciones principales contienen opciones que aparecen al desplegarse el menú. Esta opciones son objetos del tipo JMenuItem.

Un JMenu también puede contener otros JMenu, que a su vez contendrán JMenuItems.

También puede añadir separadores (JSeparator) que permiten visualizar mejor las opciones dentro de un menú.

EJERCICIO GUIADO. JAVA: BARRA DE HERRAMIENTAS

Barras de herramientas

Una barra de herramientas es básicamente un contenedor de botones y otros elementos propios de la ventana.

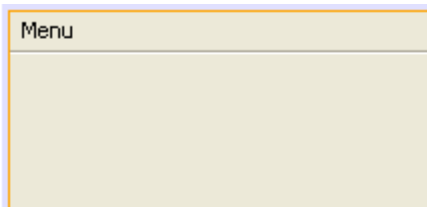
A través de estos botones se pueden activar de forma rápida las opciones del programa, las cuales suelen estar también incluidas dentro de la barra de menús.

Ejercicio guiado

41. Veamos como añadir una barra de herramientas a nuestras aplicaciones. En primer lugar, crea un proyecto con el NetBeans.

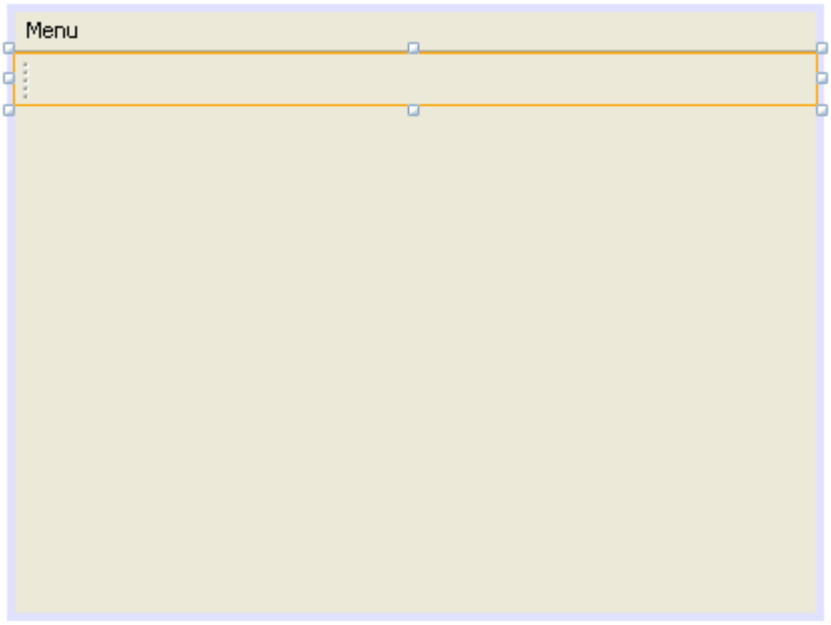
42. Añade a tu ventana un objeto JmenuBar (una barra de menús)

43. En la parte superior de tu ventana aparecerá esto:



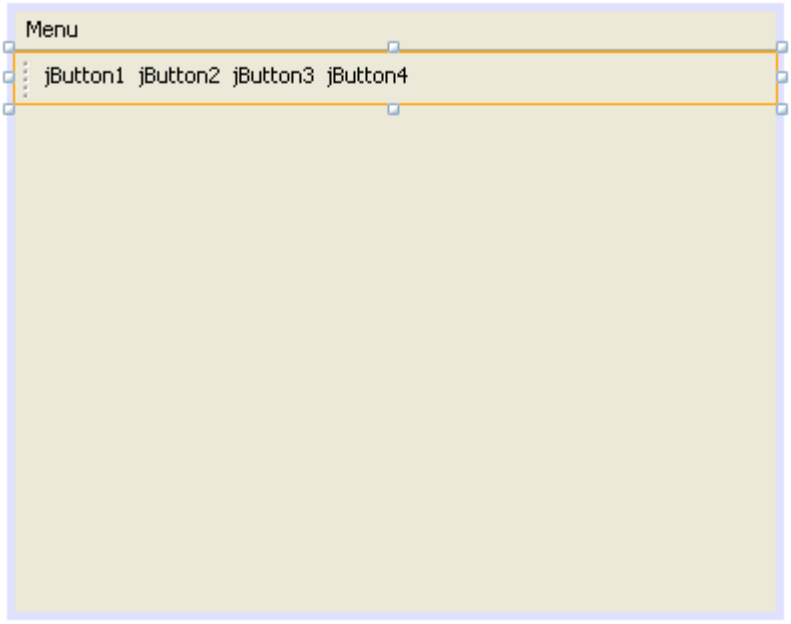
44. Debajo de la barra de menús colocaremos una barra de herramientas, así que añade un objeto del tipo JToolBar. Haz que la barra se coloque debajo de la barra de menús y que alcance desde la parte izquierda de la ventana a la parte derecha.

    La ventana quedará así:

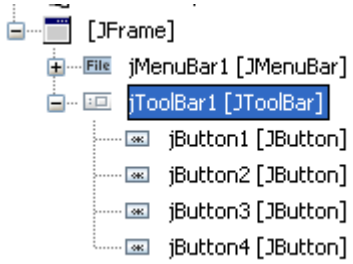


45. Las barras de herramientas son simples contenedoras de objetos. Dentro de ellas se pueden colocar botones, combos, etiquetas, etc.

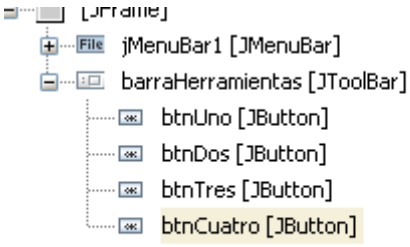
    Normalmente, las barras de herramientas contienen botones. Así que añade cuatro botones (JButton) dentro de la barra. Solo tienes que colocarlos dentro de ella.



46. Puedes ver si los botones están bien colocados observando el *Inspector*: Observa como los botones colocados se encuentran dentro de la barra.



47. Aprovecharemos el inspector para cambiar el nombre a la barra y a cada botón. A la barra la llamaremos *barraHerramientas*, y a los botones los llamaremos *btnUno, btnDos, btnTres y btnCuatro*:



48. Cambia el texto de los botones. Estos contendrán el texto: “Uno”, “Dos”, “Tres” y “Cuatro”.

49. Ejecuta el programa y observa el resultado.

50. La forma de programar cada botón no varía, aunque estos se encuentren dentro de la barra herramientas. Solo hay que seleccionar el botón y acceder a su evento *actionPerformed*.

51. Solo como demostración de esto último, entra en el *actionPerformed* del primer botón y programa esto:

```
JOptionPane.showMessageDialog(null,"Activaste el botón uno");
```

Luego ejecuta el programa y comprueba el funcionamiento del botón.

52. Los botones de la barra de herramientas normalmente no contienen texto, sino que contienen un icono que representa la función que realiza. La forma de colocar un icono dentro de un botón es a través de su propiedad *icon*.

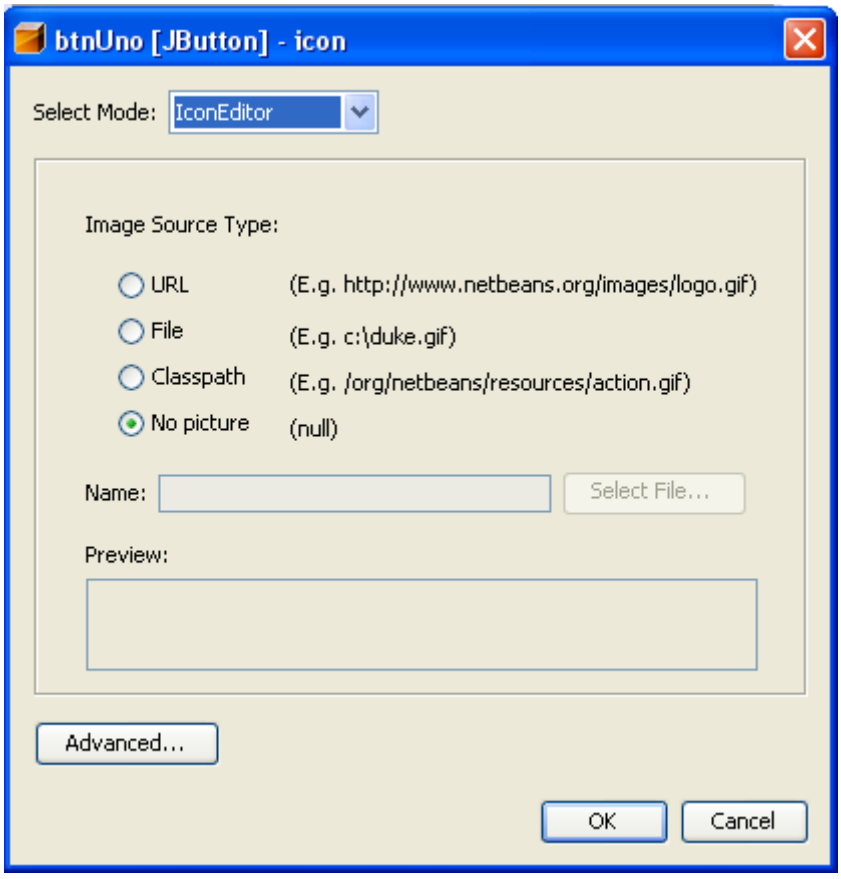
53. A través de la propiedad *icon* de un botón podrá seleccionar un fichero de imagen que contenga la imagen a mostrar en el botón.

54. Activa la propiedad icon del primer botón. Luego elige la opción *Fichero* y pulsa el botón *Seleccionar Fichero* para buscar un fichero con imagen.

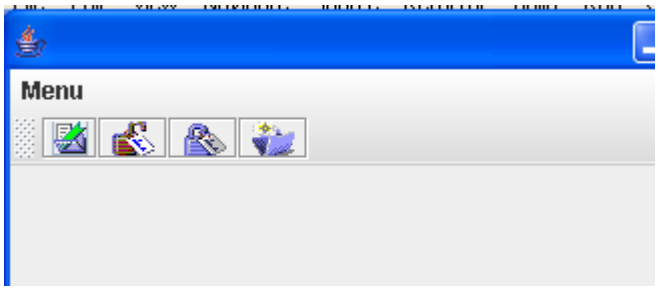
    Nota: Busca un fichero de imagen que sea del tipo .gif o .jpg.

    Nota: Procura que la imagen sea pequeña.

    Nota: Se recomienda buscar imágenes .gif en Internet para practicar.



55. Una vez colocadas las imágenes a los botones, se puede quitar el texto de estos. Un ejemplo de cómo podría quedar la barra de herramientas es este:



CONCLUSIÓN

Las barras de herramientas son simplemente contenedores de objetos. Normalmente botones.

Los elementos de la barra de herramientas se manejan de la misma forma que si no estuvieran dentro de la barra.

Lo normal es hacer que los botones de la barra no tengan texto y tengan iconos asociados.

EJERCICIO GUIADO. JAVA: MENUS EMERGENTES

El evento mouseClicked

El evento mouseClicked es capaz de capturar un clic del ratón sobre un determinado elemento de la ventana.

Este evento recibe como parámetro un objeto del tipo MouseEvent, y gracias a él se puede conseguir información como la siguiente:

1. Qué botón del ratón fue pulsado.
2. Cuantas veces (clic, doble clic, etc)
3. En qué coordenadas fue pulsado el botón.
4. Etc.

Se puede usar esta información para saber por ejemplo si se pulsó el botón derecho del ratón, y sacar en este caso un menú contextual en pantalla.

En este ejercicio guiado se estudiarán las posibilidades del evento mouseClicked y se aplicarán a la creación y visualización de menús contextuales (o emergentes)

Ejercicio guiado

56. Crea un nuevo proyecto.

57. No hace falta que añada nada a la ventana.

58. Programaremos la pulsación del ratón sobre el formulario, así que haga clic sobre el formulario y active el evento *mouseClicked*.

59. Observe el código del evento:

```
private void formMouseClicked(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:

    }
```

Este evento recibe como parámetro un objeto llamado *evt* del tipo MouseEvent (en rojo en el código) que nos permite saber en qué condiciones se hizo clic.

60. Dentro del evento programe lo siguiente:

```
if (evt.getButton()==1) {
            JOptionPane.showMessageDialog(null,"Pulso el izquierdo");
} else if (evt.getButton()==2)  {
            JOptionPane.showMessageDialog(null,"Pulso el central");
} else if (evt.getButton()==3) {
            JOptionPane.showMessageDialog(null,"Pulso el derecho");
}
```

F d ió Códi Lib D i i

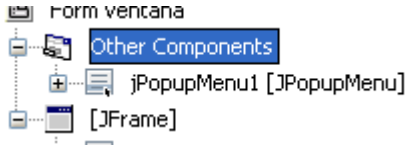
61. Ejecuta el programa y haz clic sobre el formulario con el botón derecho, con el izquierdo y con el central. Observa el resultado.

62. Ahora quizás puedas comprender el código anterior. En él, se usa el método *getButton* del objeto *evt* para saber qué botón se pulsó. El método *getButton* devuelve un entero que puede ser 1, 2 o 3 según el botón pulsado.

63. Se puede aprovechar el método *getButton* para controlar la pulsación del botón derecho del ratón y así sacar un menú contextual. Pero antes, es necesario crear el menú.

64. Agrega a tu formulario un objeto del tipo *JPopupMenu*. Estos objetos definen menús emergentes.

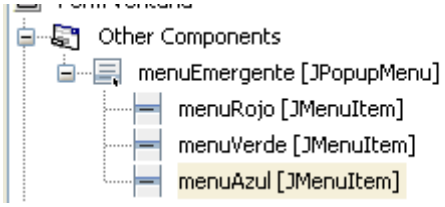
65. Los objetos JPopupMenu no se muestran en el formulario, pero puedes verlo en el *Inspector* dentro de la rama de *Otros Componentes*:



66. Aprovecharemos el inspector para cambiar el nombre al menú. Llámalo *menuEmergente*.

67. Los menús emergentes se crean igual que las opciones de menús normales, añadiendo con el botón derecho del ratón objetos JMenuItem.

68. Añada al menú emergente tres JMenuItem, y asígneles los siguientes nombres a cada uno: *menuRojo, menuVerde, menuAzul*. El *inspector debería tener el siguiente aspecto:*



69. Tienes que cambiar la propiedad *text* de cada opción del menú. Recuerda que esta propiedad define lo que aparece en el menú. Asignarás los siguientes textos: “Rojo”, “Verde” y “Azul”.

70. El menú emergente ya está construido. Ahora tenemos que hacer que aparezca cuando el usuario pulse el botón derecho del ratón sobre el formulario. Para ello,

entraremos de nuevo en el evento *mouseClicked* del formulario y cambiaremos su código por el siguiente:

```
menuEmergente.show(this,evt.getX(),evt.getY());
```

71. Este código significa lo siguiente:

5. El método show le da la orden al menuEmergente para que se muestre.
6. El método show recibe tres elementos: por un lado la ventana donde actúa (this)
7. Por otro lado la posición x donde debe mostrarse el menú. Esta posición es aquella donde se pulsó el ratón, y se puede conseguir gracias al método *getX* del objeto evt.
8. Por último se necesita la posición y. Esta posición se puede conseguir gracias al método *getY* del objeto evt.

   Es decir, decidimos mostrar el menú emergente justo en las coordenadas donde se hizo clic.

72. Ejecuta el programa y observa el resultado.

Al hacer clic con el derecho se mostrará el menú contextual.

73. Para hacer que al pulsarse una opción suceda algo, solo hay que activar el método *actionPerformed* del JMenuItem correspondiente. Por ejemplo, active el *actionPerformed* del menuRojo y dentro programe lo siguiente:

```
this.getContentPane().setBackground(Color.RED);
```

74. Ejecuta el programa y comprueba lo que sucede al pulsar la opción Rojo del menú contextual.

CONCLUSIÓN

Los menús contextuales son objetos del tipo JPopupMenu. Estos objetos contienen JMenuItem al igual que las opciones de menú normales.

Cuando se asigna un JPopupMenu a un formulario, no aparece sobre la ventana, pero sí en el *inspector*.

Para hacer que aparezca el menú emergente, es necesario programar el evento *mouseClicked* del objeto sobre el que quiera que aparezca el meú.

Tendrá que usar el método *show* del menú emergente para mostrar dicho menú.

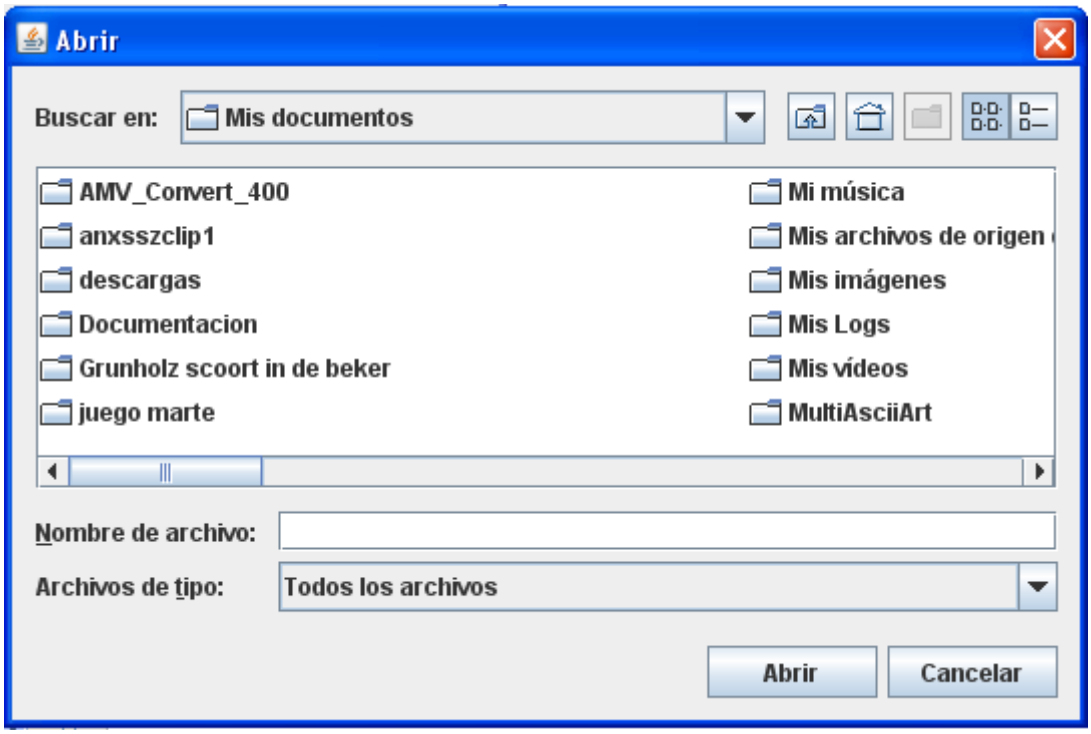
EJERCICIO GUIADO. JAVA: FILECHOOSER

Cuadros de diálogo Abrir y Guardar

Las opciones Abrir y Guardar son opciones muy comunes en las aplicaciones. Estas opciones permiten buscar en el árbol de carpetas del sistema un fichero en concreto y abrirlo, o bien guardar una información dentro de un fichero en alguna carpeta.

Java proporciona una clase llamada JFileChooser (*elegir fichero*) que permite mostrar la ventana típica de Abrir o Guardar:

Ventana Abrir fichero:



(La ventana de guardar es la misma, solo que muestra en su barra de título la palabra *Guardar*)

El objeto JFileChooser nos facilita la labor de elegir el fichero, pero no realiza la apertura o la acción de guardar la información en él. Esto tendrá que ser programado.

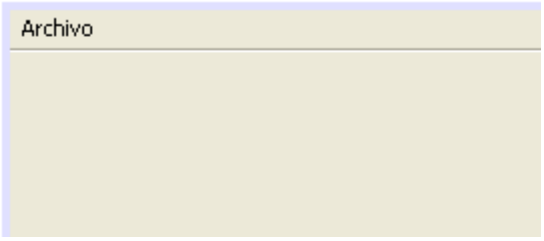
Ejercicio guiado

- Vamos a practicar con el JFileChooser. Para ello, crea un nuevo proyecto.

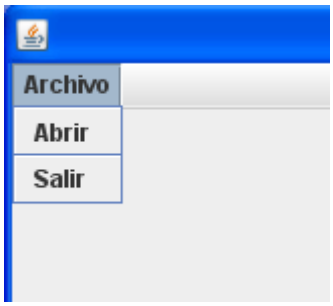
- Añade en el proyecto los siguientes elementos:

    - Una barra de menús. Llámala *barraMenus*.
    - Dentro de ella una opción “Archivo” llamada *menuArchivo*.
    - Dentro de la opción “Archivo”, introduce los siguientes elementos:
        - Una opción “Abrir”, llamada *menuAbrir*.
        - Un separador (llámalo como quieras)
        - Una opción “Salir”, llamada *menuSalir*.

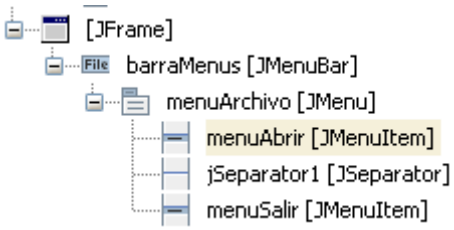
- Una vez hecho esto tu formulario tendrá la siguiente forma:



- Si ejecutas el programa el menú se verá así:



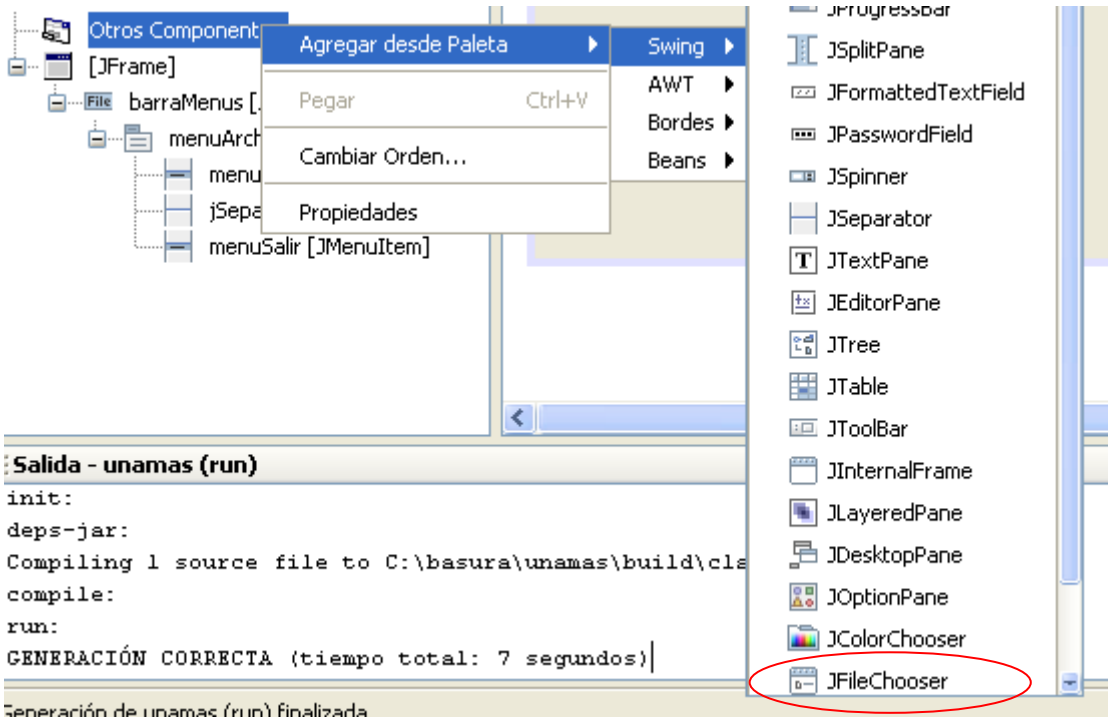
- Si observas el *Inspector*, tendrá un aspecto parecido al siguiente:



- Para que al pulsar la opción “Abrir” de nuestro programa aparezca el diálogo de apertura de ficheros, es necesario añadir a nuestro programa un objeto del tipo JFileChooser.

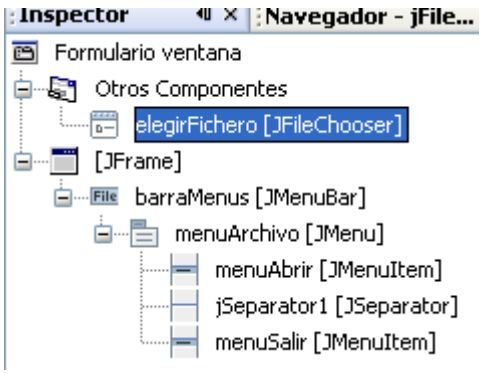
Los objetos JFileChooser se añadirán el la zona de *"Otros Componentes"* del inspector.

- Haz clic con el derecho sobre la zona de "otros componentes" y activa la opción *Agregar desde Paleta – Swing – JFileChooser*:



- Aparecerá entonces un objeto JFileChooser dentro de *Otros Componentes*. Aprovecha para cambiarle el nombre a este objeto. Su nombre será *elegirFichero*.

  El *inspector* quedará así:



- Una vez hecho esto, ya podemos programar la opción Abrir del menú. Activa el evento *actionPerformed* de la opción “Abrir” y programa dentro de él lo siguiente:

```
int resp;

resp=elegirFichero.showOpenDialog(this);

if (resp==JFileChooser.APPROVE_OPTION) {
                  JOptionPane.showMessageDialog(null,elegirFichero.getSelectedFile(
    ).toString());

} else if (resp==JFileChooser.CANCEL_OPTION) {

    JOptionPane.showMessageDialog(null,"Se pulsó la opción Cancelar");
}
```

- Ejecuta el código y prueba la opción “Abrir” del menú. Prueba a elegir algún fichero y abrirlo. Prueba a cancelar la ventana de apertura. Etc

- Analicemos el código anterior:

```
int resp;

resp=elegirFichero.showOpenDialog(this);
```

75. Estas dos líneas crean una variable entera *resp* (respuesta) y a continuación hacen que se muestre la ventana “Abrir Fichero”. Observa que para conseguirlo hay que usar el método *showOpenDialog* del objeto elegirFichero. Este método lleva como parámetro la ventana actual (this)

76. El método showOpenDialog no solo muestra la ventana “Abrir Fichero” sino que también devuelve un valor entero según el botón pulsado por el usuario en esta ventana. Esto es: botón “Abrir” o botón “Calcelar”.

77. Se pueden usar dos if para controlar lo que sucede si el usuario pulsó el botón “Abrir” o el botón “Calcelar” de la ventana “Abrir Fichero”:

```
if (resp==JFileChooser.APPROVE_OPTION) {
                  JOptionPane.showMessageDialog(null,elegirFichero.getSelectedFile(
    ).toString());

} else if (resp==JFileChooser.CANCEL_OPTION) {

    JOptionPane.showMessageDialog(null,"Se pulsó la opción Cancelar");
}
```

78. En el primer if se compara la variable *resp* con la constante *JFileChooser.APPROVE_OPTION*, para saber si el usuario pulsó “Abrir”.

79. En el segundo if se compara la variable *resp* con la constante *JFileChooser.CANCEL_OPTION*, para saber si el usuario pulsó “Calcelar”.

F d ió Códi Lib D i i

80. En el caso de que el usuario pulsara “Abrir”, el programa usa el método *getSelectedFile* del objeto *elegirFichero* para recoger el camino del fichero elegido. Este camino debe ser convertido a cadena con el método *toString*.

81. El programa aprovecha esto para mostrar dicho camino en pantalla gracias al típico JOptionPane.

82. En el caso del que el usuario pulsara el botón “Cancelar” el programa muestra un mensaje indicándolo.

- Hay que volver a dejar claro que el cuadro de diálogo “Abrir” realmente no abre ningún fichero, sino que devuelve el camino del fichero elegido usando el código:

  ```
  elegirFichero.getSelectedFile().toString()
  ```

  Luego queda en manos del programador el trabajar con el fichero correspondiente de la forma que desee.

CONCLUSIÓN

Los objetos JFileChooser permiten mostrar el cuadro de diálogo “Abrir Fichero” o “Guardar Fichero”.

Estos objetos no abren ni guardan ficheros, solo permiten al usuario elegir el fichero a abrir o guardar de forma sencilla.

El JFileChooser devuelve el camino del fichero elegido, y luego el programador trabajará con dicho fichero como mejor le interese.

EJERCICIO GUIADO. JAVA: PANELES DE DESPLAZAMIENTO

Paneles de Desplazamiento

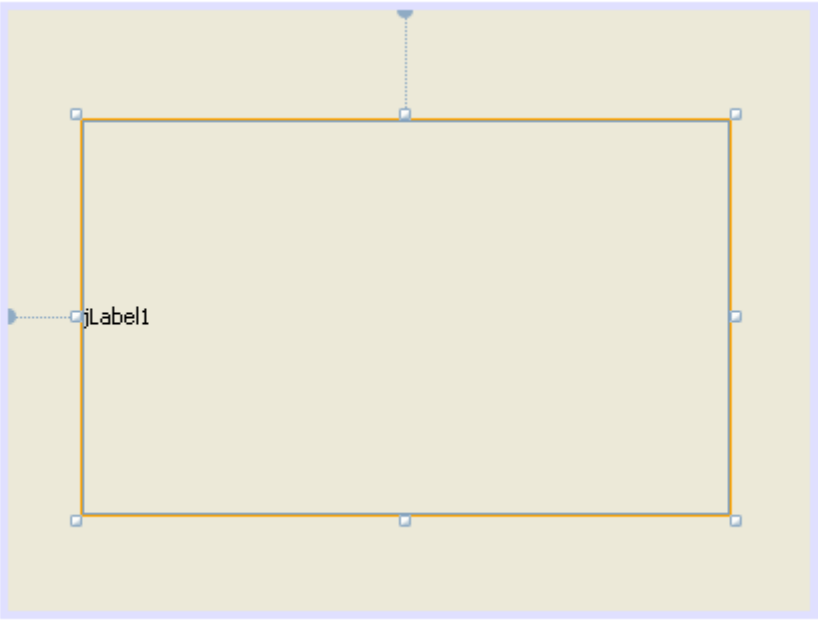
Llamaremos paneles de desplazamiento a paneles que contienen elementos tan grandes que no pueden ser mostrados en su totalidad. Estos paneles contienen entonces dos barras de desplazamiento que permiten visualizar el interior del panel de desplazamiento correctamente.

Por ejemplo, un panel de desplazamiento podría contener una imagen tan grande que no se viera entera:

Los paneles de desplazamiento son objetos del tipo JScrollPane.

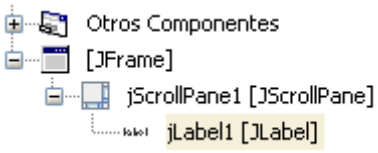
Ejercicio guiado 1

- Vamos a practicar con los JScrollPane. Para ello, crea un nuevo proyecto.
- Añade en el proyecto un JScrollPane.
- Un JScrollPane, por sí mismo, no contiene nada. Es necesario añadir dentro de él el objeto que contendrá. Para nuestro ejemplo añadiremos dentro de él una etiqueta (JLabel)
- El formulario debe tener ahora este aspecto:



- Si observas el *Inspector* verás claramente la distribución de los objetos:

F d ió Códi Lib D i i



Observa como tienes un JScrollPane que contiene una etiqueta.

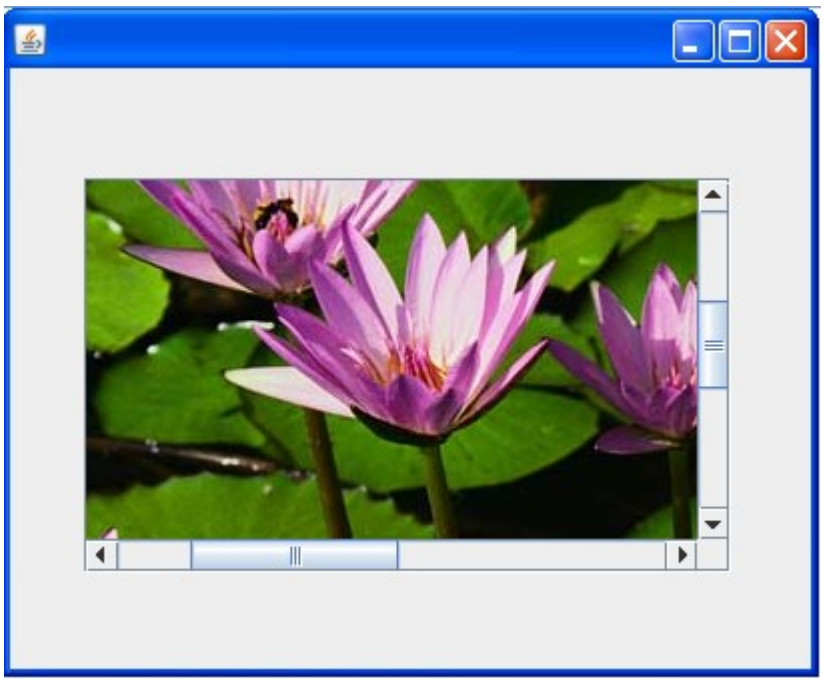
- Aprovechemos el *Inspector* para cambiar el nombre a cada objeto. Al JScrollPane le llamaremos *scpImagen* y a la etiqueta *etiImagen*.

- Elimina el texto contenido en la etiqueta *etiImagen*. Solo tienes que borrar el contenido de la propiedad *text*.

- Luego introduciremos una imagen dentro de la etiqueta, a través de la propiedad *icon*. La imagen la introduciremos desde fichero, y elegiremos la siguiente imagen de tu disco duro:

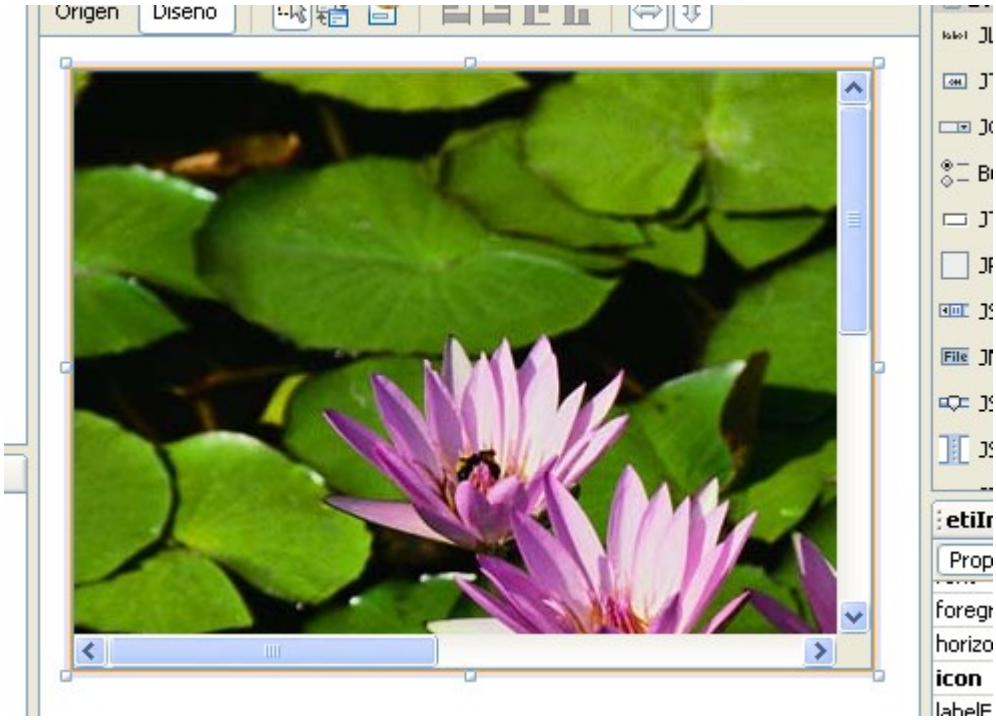
  Mis Documentos / Mis Imágenes / Imágenes de Muestra / Nenúfares.jpg

- Esta imagen es tan grande que no se podrá ver entera dentro del panel de desplazamiento. Ejecuta el programa y observarás el uso de las barras de desplazamiento dentro del panel.



F d ió Códi Lib D i i

- Puedes mejorar el programa si agrandas el panel de desplazamiento de forma que ocupe todo el formulario:



De esta forma, cuando ejecutes el programa, al agrandar la ventana, se agrandará el panel de desplazamiento, viéndose mejor la imagen contenida.

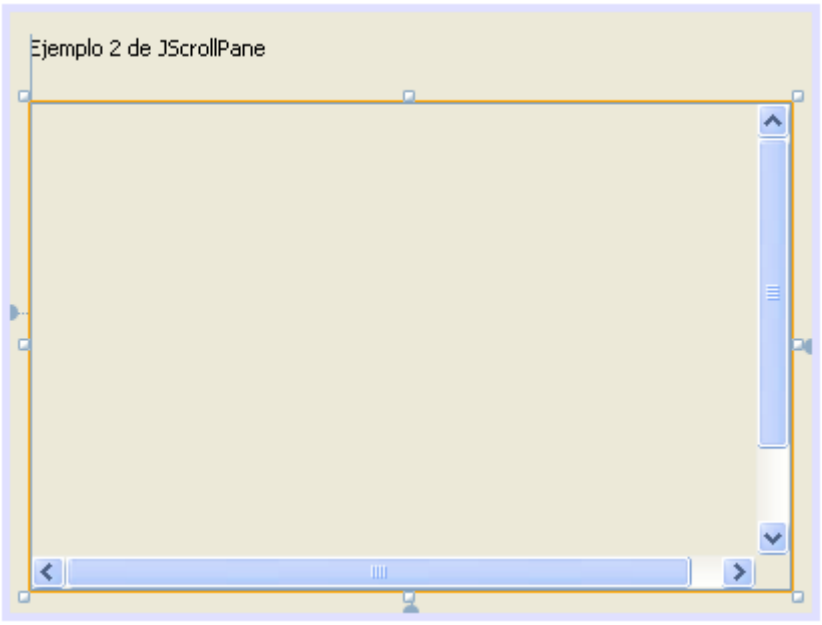
- Ejecuta el programa y compruébalo.

Ejercicio guiado 2

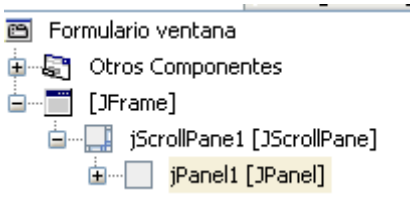
Los JScrollPane no solo están diseñados para contener imágenes. Pueden contener cualquier otro elemento. Vamos a ver, con otro proyecto de ejemplo, otro uso de los JScrollPane.

1. Crea un nuevo proyecto.

2. Añade a la ventana una etiqueta con el texto “Ejemplo 2 de JScrollPane” y un JScrollPane de forma que esté asociado a los límites de la ventana. Observa la imagen:

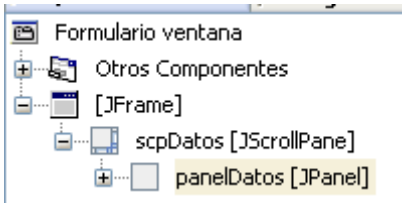


3. Ahora añade dentro del JScrollPane un panel normal (JPanel). En la ventana no notarás ninguna diferencia, pero en el *Inspector* debería aparecer esto:



4. Como ves, el JScrollPane contiene a un objeto JPanel.

5. Aprovechemos para cambiar el nombre a ambos objetos. Al JScrollPane lo llamaremos *scpDatos* y al JPanel lo llamaremos *panelDatos*.



6. Los JPanel son objetos contenedores. Es decir, pueden contener otros objetos como por ejemplo botones, etiquetas, cuadros de texto, etc.

   Además, los JPanel pueden ser diseñados independientemente de la ventana. Haz doble clic sobre el *panelDatos* en el *Inspector* y observa lo que ocurre:

7. En la pantalla aparecerá únicamente el JPanel, para que puede ser diseñado aparte de la ventana completa:

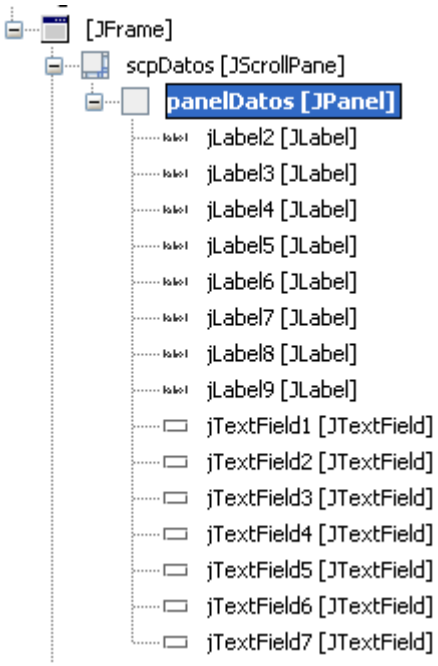
F d ió Códi Lib D i i

8. Para distinguirlo de lo que es en sí la ventana, haremos las siguientes cosas con el panel:

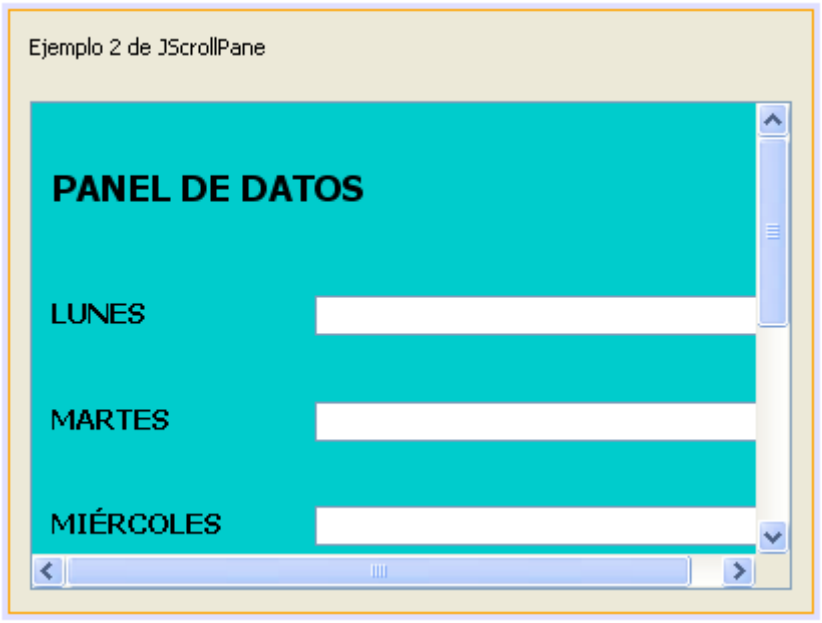
123. Cambia el color de fondo y asígnale un color verde.
124. Añade en él una etiqueta con el texto “Panel de Datos”.
125. Añade varias etiquetas y cuadros de textos correspondientes a los días de la semana.
126. Agranda el panel.

El panel debería quedar así. Toma como referencia esta imagen:

Es muy interesante que observes el *Inspector*. En él podrás observar la distribución de los objetos en la ventana. Podrás ver como el JFrame contiene un JScrollPane (scpDatos) que a su vez contiene un JPanel (panelDatos) que a su vez contiene una serie de etiquetas y cuadros de textos a los que aún no les hemos asignado un nombre:



9. Haz doble clic sobre el JFrame (en el *Inspector*) para poder ver globalmente la ventana. En la pantalla debería aparecer esto:



   Como ves, el JPanel contenido en el JScrollPane es más grande que él, por lo que no se podrá visualizar completamente. Será necesario usar las barras de desplazamiento del JScrollPane.

10. Ejecuta el programa para entender esto último.

F d ió Códi Lib D i i

CONCLUSIÓN

Los objetos JScrollPane son paneles de desplazamiento. Estos paneles pueden contener objetos mayores que el propio panel de desplazamiento. Cuando esto sucede, el panel muestra barras de desplazamiento para poder visualizar todo el contenido del panel.

Los JScrollPane son ideales para mostrar imágenes, paneles y otros elementos cuyo tamaño pueda ser mayor que la propia ventana.

EJERCICIO GUIADO. JAVA: VARIABLES GLOBALES

Variables Globales / Propiedades de la Clase

Las propiedades de la clase en java es el equivalente a las variables globales en lenguajes estructurados como el C.

Una propiedad es una variable que puede ser accedida desde cualquier evento programado. Esta variable se inicializa a un valor cuando se ejecuta el programa y los distintos eventos pueden ir cambiando su valor según se necesite.

Veamos un ejemplo para entender el funcionamiento de las propiedades de la clase / variables globales.

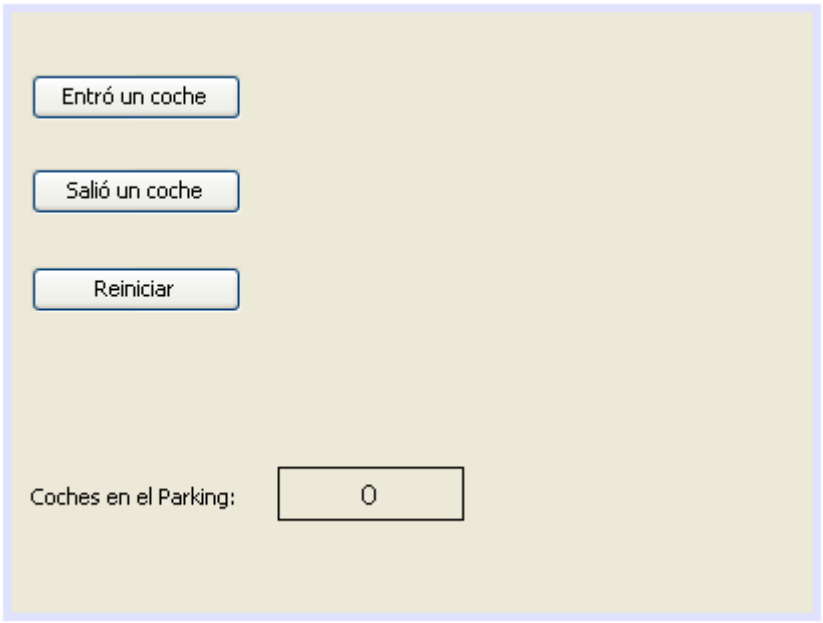
Ejercicio guiado 1

- Crea un nuevo proyecto llamado *ProyectoParking*. Dentro de él añade un paquete llamado *paqueteParking.* Y finalmente añade un JFrame llamado *Parking*. El aspecto de tu proyecto será el siguiente:

Clase Principal

- Tu clase principal es la clase *Parking*.

- Se pretende hacer un pequeño programa que controle los coches que van entrando y van saliendo de un parking. En todo momento el programa debe decir cuantos coches hay dentro del parking. Para ello debes crear una ventana como la que sigue:



- Esta ventana contiene lo siguiente:

  - Un botón “Entró un coche” llamado btnEntro.
  - Un botón “Salió un coche” llamado btnSalio.
  - Un botón “Reiniciar” llamado btnReiniciar.
  - Una etiqueta con el texto “Coches en el Parking”.
  - Una etiqueta con borde llamada etiCoches que contenga un “0”.

- Se pretende que el programa funcione de la siguiente forma:

  - Cuando el usuario pulse el botón “Entró un coche”, el número de coches del parking aumenta en 1, mostrándose en la etiqueta etiCoches.
  - Si el usuario pulsa el botón “Salió un coche”, el número de coches del parking disminuye en 1, mostrándose en la etiqueta etiCoches.
  - El botón Reiniciar coloca el número de coches del parking a 0.

- Para poder controlar el número de coches en el Parking, es necesario que nuestra clase principal *Parking* tenga una propiedad (variable global) a la que llamaremos *coches*. Esta variable será entera (int).

  Esta variable contendrá en todo momento los coches que hay actualmente en el *Parking*. Esta variable podrá ser usada desde cualquier evento.

- Para crear una variable global haz clic en el botón “Origen” para acceder al código:

- Luego busca, al comienzo del código una línea que comenzará por

  ```
  public class
  ```

F d ió Códi Lib D i i

Seguida del nombre de tu clase principal "*Parking*".

Debajo de dicha línea es donde se programarán las propiedades de la clase (las variables globales)

Aquí se declaran las variables globales, también llamadas propiedades de la clase.

```
/**
 *
 * @author  didact
 */
public class Parking extends javax.swing.JFrame {

    /**
     * Creates new form Parking
     */
    public Parking() {
        initComponents();
    }
```

- En dicho lugar declararás la variable *coches* de tipo int:

```
public class Parking extends javax.swing.JFrame {

    int coches;

    /**
     * Creates new form Parking
     */
    public Parking() {
```

Declaración de una variable global int llamada coches.

- Cuando el programa arranque, será necesario que la variable global *coches* tenga un valor inicial. O dicho de otra forma, será necesario inicializar la variable. Para inicializar la variable iremos al constructor. Añade lo siguiente al código:

```
public class Parking extends javax.swing.JFrame {

    int coches;

    /**
     * Creates new form Parking
     */
    public Parking() {
        initComponents();
        coches=0;
    }

    /** This method is called from within the construct
```

Inicialización de la propiedad coches.

Inicializamos a cero ya que se supone que cuando arranca el programa no hay ningún coche dentro del parking.

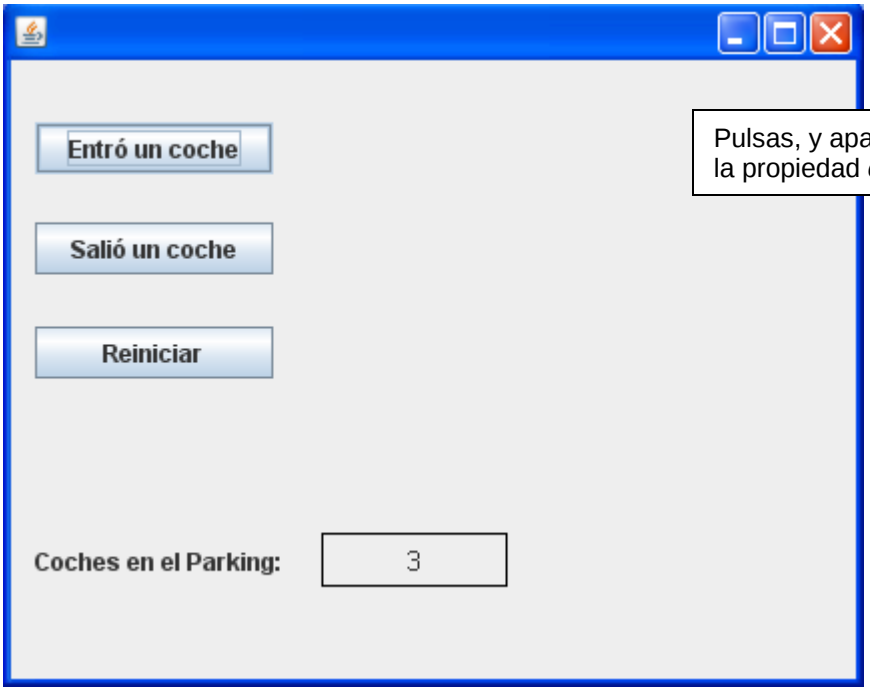
- Ahora que ya tenemos declarada e inicializada la variable global *coches*, esta puede ser usada sin problemas desde cualquier evento que programemos.

  Por ejemplo, empezaremos programando la pulsación del botón “Entró un coche”. Acceda a su evento *actionPerformed* y programe esto:

```
coches=coches+1;
etiCoches.setText(“”+coches);
```

  Como ves, se le añade a la variable *coches* uno más y luego se coloca su valor actual en la etiqueta.

- Ejecuta el programa y prueba este botón varias veces.



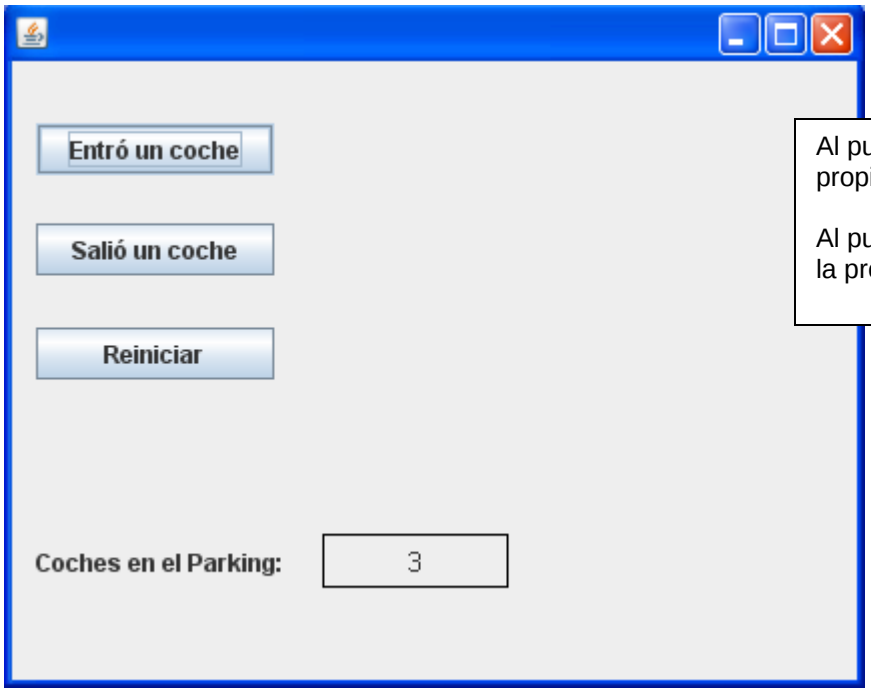
- Ahora programaremos el botón “Salió un coche” de la siguiente forma:

```
if (coches>0) {
  coches=coches-1;
  etiCoches.setText(“”+coches);
}
```

  Como ves, se accede igualmente a la propiedad *coches* pero esta vez para restarle una unidad. Luego se muestra el valor actual de *coches* en la etiqueta correspondiente.

Se usa un if para controlar que no pueda restarse un coche cuando el parking esté vacío. (Si hay cero coches en el parking no se puede restar uno)

- Ejecuta el programa y prueba los dos botones. Observa como la cantidad de coches del parking aumenta o disminuye.



Al pulsar aquí aumentas en uno la propiedad coches.

Al pulsar aquí disminuyes en uno la propiedad coches.

Lo realmente interesante de esto es que desde dos eventos distintos (desde dos botones) se puede usar la variable *coches*. Esto es así gracias a que ha sido creada como variable global, o dicho de otro modo, ha sido creada como propiedad de la clase *Parking*.

- Finalmente se programará el botón *Reiniciar*. Este botón, al ser pulsado, debe colocar la propiedad *coches* a cero. Programa dentro de su *actionPerformed* lo siguiente:

```
coches=0;
etiCoches.setText("0");
```

Simplemente introduzco el valor cero en la variable global y actualizo la etiqueta.

- Ejecuta el programa y comprueba el funcionamiento de este botón.

CONCLUSIÓN

Las variables globales, también llamadas propiedades de la clase, son variables que pueden ser usadas desde cualquier evento del programa. Estas variables mantienen su valor hasta que otro evento lo modifique.

Las variables globales se declaran justo después de la línea *public class*.

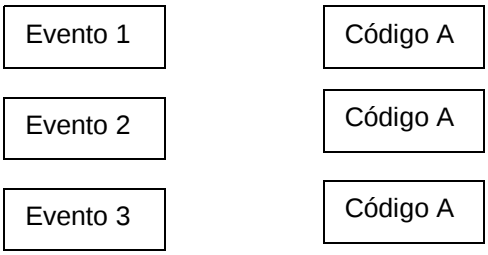
La inicialización de estas variables se realiza en el constructor.

F d ió Códi Lib D i i

EJERCICIO GUIADO. JAVA: CENTRALIZAR CÓDIGO

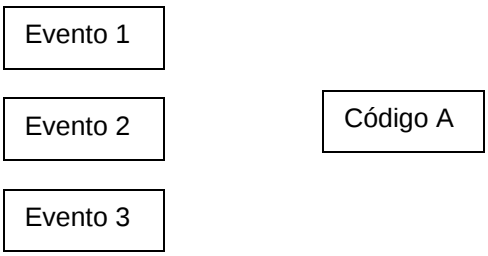
El problema de la repetición de código

Es muy habitual en Java que varios eventos tengan que ejecutar el mismo código. En este caso se plantea la necesidad de "copiar y pegar" ese código en los distintos eventos a programar:



Esta es una mala forma de programación, ya que se necesitara modificar el código, sería necesario realizar la modificación en cada copia del código. Es muy fácil que haya olvidos y aparezcan errores en el programa que luego son muy difíciles de localizar.

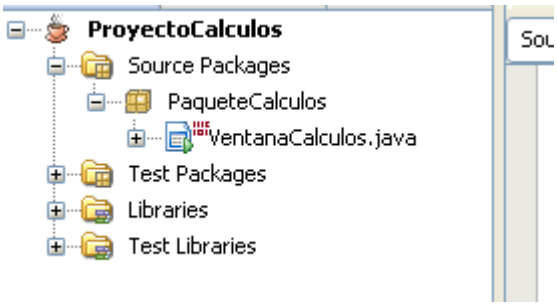
Lo mejor es que el código que tenga que ser ejecutado desde distintos eventos aparezca solo una vez, y sea llamado desde cada evento:



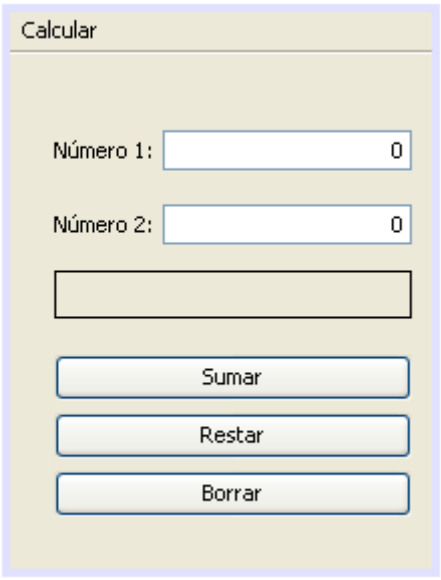
Veamos algunos ejemplos en los que el código se puede repetir y como evitar esta repetición.

Ejercicio guiado 1

- Crea un nuevo proyecto en java que se llame *ProyectoCalculos*. Este proyecto tendrá un paquete llamado *PaqueteCalculos*. Y dentro de él creará un JFrame llamado *VentanaCalculos*. El proyecto tendrá el siguiente aspecto:



- La *VentanaCalculos* debe estar diseñada de la siguiente forma:



Esta ventana contiene los siguientes elementos:

- Una barra de menús a la que puede llamar *menuBarra*.
- La barra de menús contiene un JMenu con el texto “Calcular” y que se puede llamar *menuCalcular*
- El *menuCalcular* contendrá tres JMenuItem, llamados respectivamente: *menuSumar, menuRestar, menuBorrar* y con los textos *“Sumar”, “Restar” y “Borrar”*.
- Una etiqueta con el texto “Número 1”. (no importa su nombre)
- Una etiqueta con el texto “Número 2”. (no importa su nombre)
- Un cuadro de texto con un 0 y con el nombre *txtNumero1*.
- Un cuadro de texto con un 0 y con el nombre *txtNumero2*.
- Una etiqueta con el nombre *etiResultado*.
- Un botón “Sumar” con el nombre *btnSumar*.
- Un botón “Restar” con el nombre *btnRestar*.
- Un botón “Borrar” con el nombre *btnBorrar*.

- Aquí puedes ver la ventana en ejecución con el menú “Calcular” desplegado:

F d ió Códi Lib D i i

- El objetivo de programa es el siguiente:
    - El usuario introducirá dos números en los cuadros de texto.
    - Si pulsa el botón Sumar, se calculará la suma.
    - Si pulsa el botón Restar, se calculará la resta.
    - Si pulsa el botón Borrar, se borrarán ambos cuadros de texto.
    - Si elige la opción del menú Calcular-Sumar entonces se calculará la suma.
    - Si elige la opción del menú Calcular-Restar entonces se calculará la resta.
    - Si elige la opción del menú Calcular-Borrar entonces se borrarán ambos cuadros de texto.
    - Si se pulsa enter en alguno de los dos cuadros de texto se debería calcular la suma.

- Este es un ejemplo en el que al activarse uno de varios eventos distintos se tiene que ejecutar el mismo código. Observa el caso de la suma:

Pulsar Botón Sumar

Activar Calcular – Sumar en el menú

Pulsar enter en el primer cuadro de texto

Pulsar enter en el segundo cuadro de texto

Calcular la suma y mostrarla en la etiqueta de resultado

F d ió Códi Lib D i i

- Para que el código esté “centralizado”, es decir, que aparezca solo una vez, será necesario construir en la clase un *método*. Un *método* en java es el equivalente de una función o procedimiento en C. Veamos como hacerlo:

- Accede al código de tu programa a través del botón *Origen*.

- Un buen sitio para programar tus procedimientos puede ser debajo del constructor. Puedes distinguir fácilmente al constructor porque tiene el mismo nombre que la clase que estás programando, o dicho de otro modo, tiene el mismo nombre que la ventana que estás programando: *VentanaCalculos*.

```
 */
public class VentanaCalculos extends javax.swing.JFrame {

    /**
     * Creates new form VentanaCalculos
     */
    public VentanaCalculos() {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this m
```

Este es el constructor

Este es un buen sitio para crear tus propios procedimientos

- Se va a programar un procedimiento que se encargue de recoger los valores de los cuadros de texto. Calculará la suma de dichos valores, y luego mostrará la suma en la etiqueta de resultados.

  Los procedimientos en java tienen prácticamente la misma estructura que en C. Programe lo siguiente:

```
public class VentanaCalculos extends javax.swing.JFrame {

    /**
     * Creates new form VentanaCalculos
     */
    public VentanaCalculos() {
        initComponents();
    }

    void Sumar() {
        String cad1, cad2;
        int a,b,s;

        cad1 = txtNumero1.getText();
        cad2 = txtNumero2.getText();
        a = Integer.parseInt(cad1);
        b = Integer.parseInt(cad2);
        s=a+b;
        etiResultado.setText(""+s);
    }

    /** This method is called from within the constructor to
```

Este es el procedimiento que tienes que introducir en el programa.

- Si observas el código, es el típico procedimiento de C, cuya cabecera comienza con *void* y el nombre que le hayas asignado (en nuestro caso *Sumar*)

```
void Sumar() {
      ....
}
```

  Si estudias las líneas del código, verás que lo que hace es recoger el contenido de los dos cuadros de texto en dos variables de cadena llamadas *cad1 y cad2*.

  Luego convierte dichas cadenas en números que almacena en dos variables enteras llamadas a y b.

  Finalmente calcula la suma en una variable s y presenta el resultado en la etiqueta etiResultado.

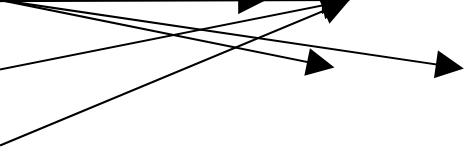
- Hay que destacar que este código no pertenece ahora mismo a ningún evento en concreto, por lo que no tiene efecto ninguno sobre el programa. Será necesario pues asociar los eventos correspondientes con este procedimiento.

- Interesa que al pulsar el botón “Sumar” se ejecute la suma, así pues entre en el evento *actionPerformed* del botón “Sumar” y añada la siguiente línea:

```
Sumar();
```

- Como también interesa que al pulsar la opción del menú “Calcular-Sumar” se ejecute la suma, entre en el evento *actionPerformed* de la opción del menú “Sumar” y añade de nuevo la siguiente línea:

F d ió Códi Lib D i i



```
Sumar();
```

- También se quiere que al pulsar la tecla enter en el cuadro de texto del número 1 se ejecute la suma. Por lo tanto, en el evento *actionPerformed* del cuadro de texto txtNumero1 hay que añadir la siguiente línea:

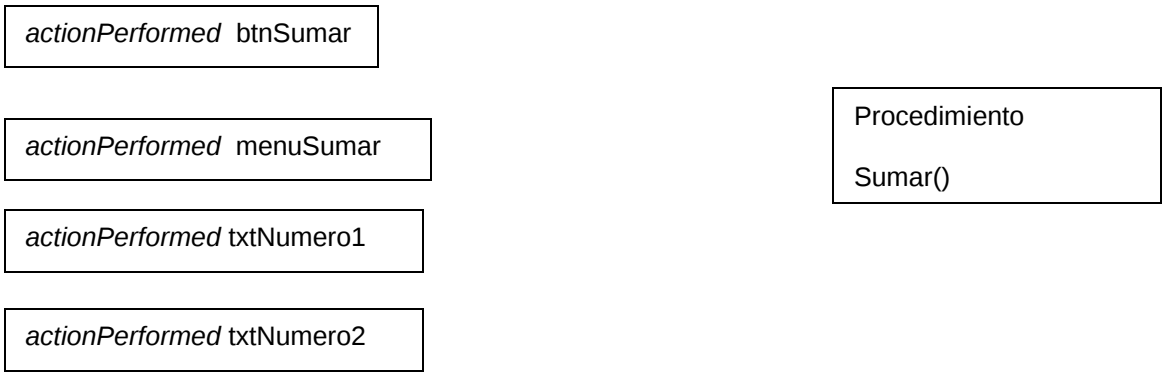
```
Sumar();
```

- Y como también se quiere que al pulsar la tecla enter en el cuadro de texto del número 2 se ejecute la suma, también habrá que introducir en su *actionPerformed* la siguiente línea:

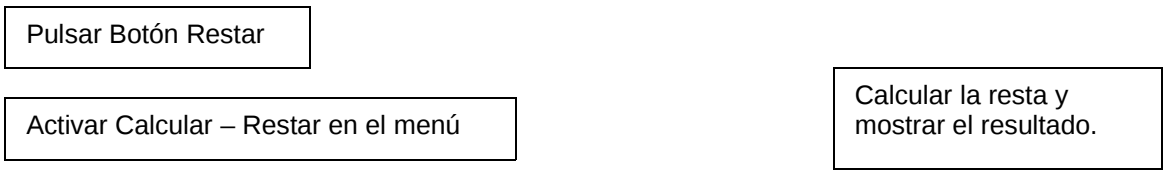
```
Sumar();
```

- Antes de continuar, ejecute el programa, introduzca dos números, y compruebe como se calcula la suma al pulsar el botón Sumar, o al activar la opción del menú Calcular–Sumar, o al pulsar Enter en el primer cuadro de texto, o al pulsar Enter en el segundo cuadro de texto.

  En cada uno de los eventos hay una llamada al procedimiento *Sumar*, que es el que se encarga de realizar la suma.



- En el caso de la resta sucede igual. Tenemos que varios eventos distintos deben provocar que se realice una misma operación. En este caso tenemos lo siguiente:



- Para centralizar el código, crearemos un método *Restar* que se encargará de hacer la resta de los números introducidos en los cuadros de texto. Este método se puede colocar debajo del anterior método *Sumar*:

```
    void Sumar() {
        String cad1, cad2;
        int a,b,s;

        cad1 = txtNumero1.getText();
        cad2 = txtNumero2.getText();
        a = Integer.parseInt(cad1);
        b = Integer.parseInt(cad2);
        s=a+b;
        etiResultado.setText(""+s);
    }

    void Restar() {
        String cad1, cad2;
        int a,b,r;

        cad1 = txtNumero1.getText();
        cad2 = txtNumero2.getText();
        a = Integer.parseInt(cad1);
        b = Integer.parseInt(cad2);
        r=a-b;
        etiResultado.setText(""+r);
    }
```

Programa este procedimiento.

- El código de este procedimiento es prácticamente idéntico al del procedimiento Sumar, así que no se comentará.

- Ahora, es necesario que cuando se activen los eventos indicados antes, estos hagan una llamada al procedimiento Restar para que se efectúe la resta. Así pues, entre en el evento *actionPerformed* del botón "Restar" y añada esta línea de código:

```
Restar();
```

- Igualmente, entre en el evento *actionPerformed* de la opción del menú "Calcular – Restar" y añada la misma llamada:

```
Restar();
```

- Ejecute el programa y compruebe como funciona el cálculo de la resta, da igual que lo haga pulsando el botón "Restar" o la opción del menú "Restar". Ambos eventos llaman al mismo método:

actionPerformed btnRestar

actionPerformed menuRestar

Procedimiento

Restar()

F d ió Códi Lib D i i

- Finalmente se programará el borrado de los cuadros de texto a través del botón “Borrar” y de la opción del menú “Borrar”. En primer lugar, programa el siguiente método (puedes hacerlo debajo del método “Restar”):

```
}

void Restar() {
    String cad1, cad2;
    int a,b,r;

    cad1 = txtNumero1.getText();
    cad2 = txtNumero2.getText();
    a = Integer.parseInt(cad1);
    b = Integer.parseInt(cad2);
    r=a-b;
    etiResultado.setText(""+r);
}

void Borrar() {
    txtNumero1.setText("");
    txtNumero2.setText("");
}
```

Programa el procedimiento Borrar...

- Ahora programa las llamadas al procedimiento borrar desde los distintos eventos. En el evento *actionPerformed* del botón “Borrar” y en el evento *actionPerformed* de la opción del menú “Borrar” programa la siguiente llamada:

```
Borrar();
```

- Ejecuta el programa y prueba su funcionamiento.

CONCLUSIÓN

En java se pueden programar procedimientos al igual que en C. Normalmente, estos procedimientos se programarán debajo del constructor, y tienen la misma estructura que en C.

Se puede llamar a un mismo procedimiento desde distintos eventos, evitando así la repetición de código.

F d ió Códi Lib D i i

EJERCICIO GUIADO. JAVA: LAYOUTS

El problema de la distribución de elementos en las ventanas

Uno de los problemas que más quebraderos de cabeza da al programador es el diseño de las ventanas y la situación de los distintos componentes en ellas.

Para diseñar más cómodamente las ventanas, Java proporciona una serie de objetos denominados Layouts, que definen la forma que tendrán los elementos de situarse en las ventanas.

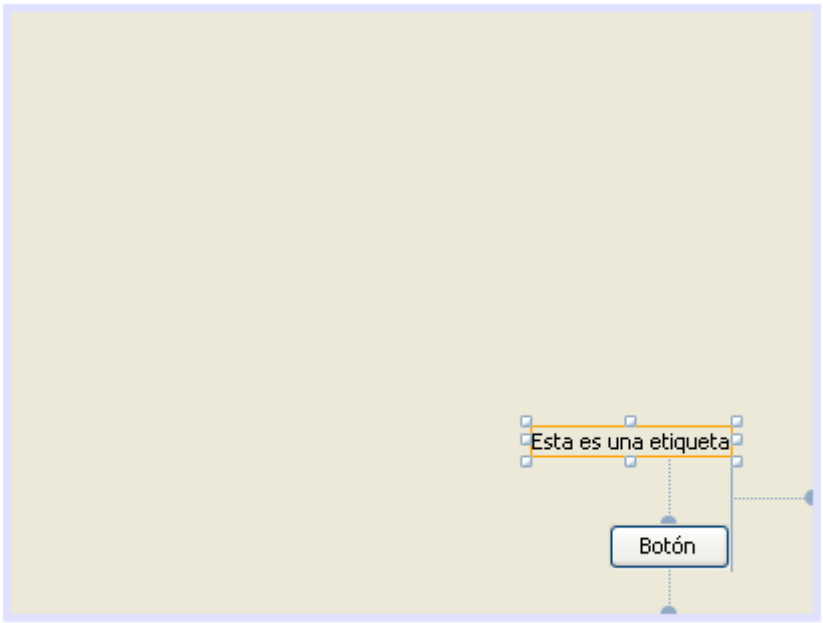
Así pues, un Layout define de qué forma se colocarán las etiquetas, botones, cuadros de textos y demás componentes en la ventana que diseñamos.

Ejercicio guiado

“Diseño Libre”

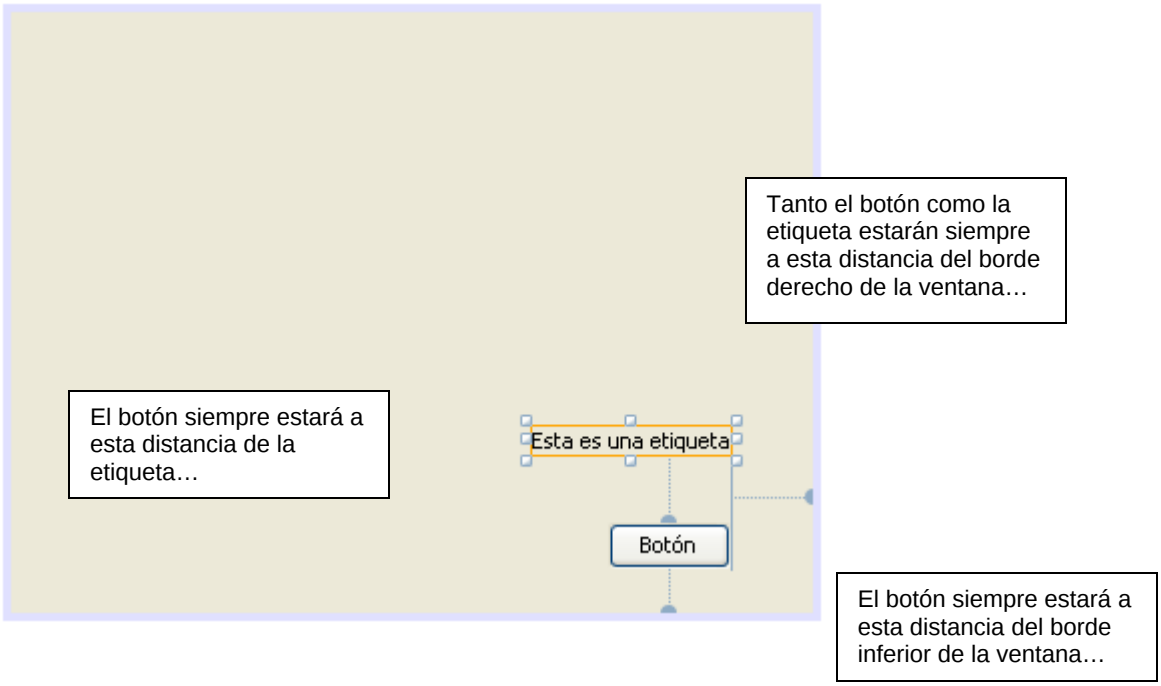
- Crea un nuevo proyecto en java.

- Añade una etiqueta y un botón. Muévelos a la posición que se indica en la imagen. Deben aparecer las líneas “guía” de color azul que se muestran:

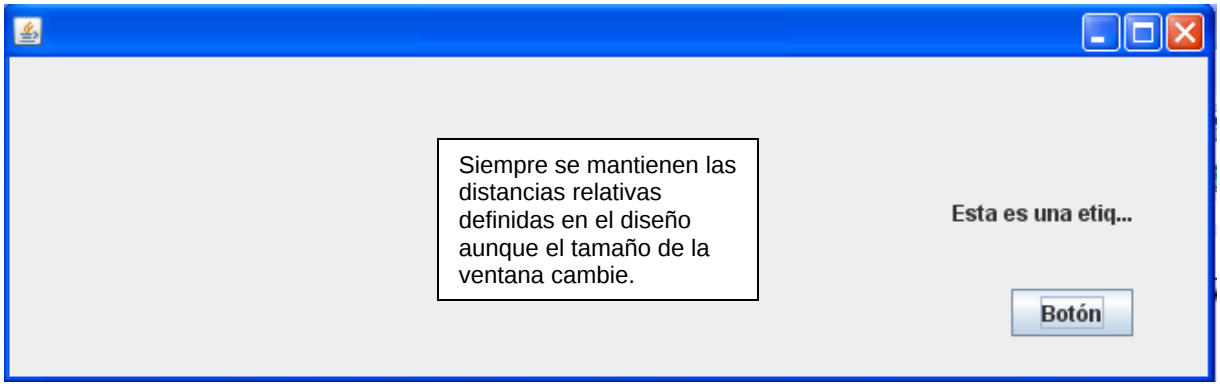


- Las líneas azules que aparecen indican con qué otro elemento está relacionado un componente de la ventana. La situación de un elemento dependerá siempre de la situación del otro.

  Dicho de otra forma, las líneas azules indican las distancias que siempre se respetarán. Observa la siguiente imagen:



- Ejecuta el programa y prueba a ensanchar (o achicar) la ventana por el lado derecho y por el lado inferior. Debes observar como la etiqueta y el botón mantienen sus distancias relativas entre sí y con los bordes derecho e inferior de la ventana.



- Este comportamiento de los elementos en la ventana viene dado por una opción del NetBeans llamada *Diseño Libre* (Free Design)

  El Diseño Libre permite que los elementos de una ventana mantengan una distribución relativa da igual el tamaño que tenga la ventana. Dicho de otra forma, los elementos se redistribuyen al cambiar el tamaño de la ventana.

  El problema del Diseño Libre es el poco control que se tiene sobre los elementos que se añaden a la ventana.

  Se puede observar como a veces los elementos no se colocan en la posición que deseamos o como cambian de tamaño de forma inesperada. Todo esto es debido a la necesidad de dichos elementos de mantener unas distancias relativas con otros

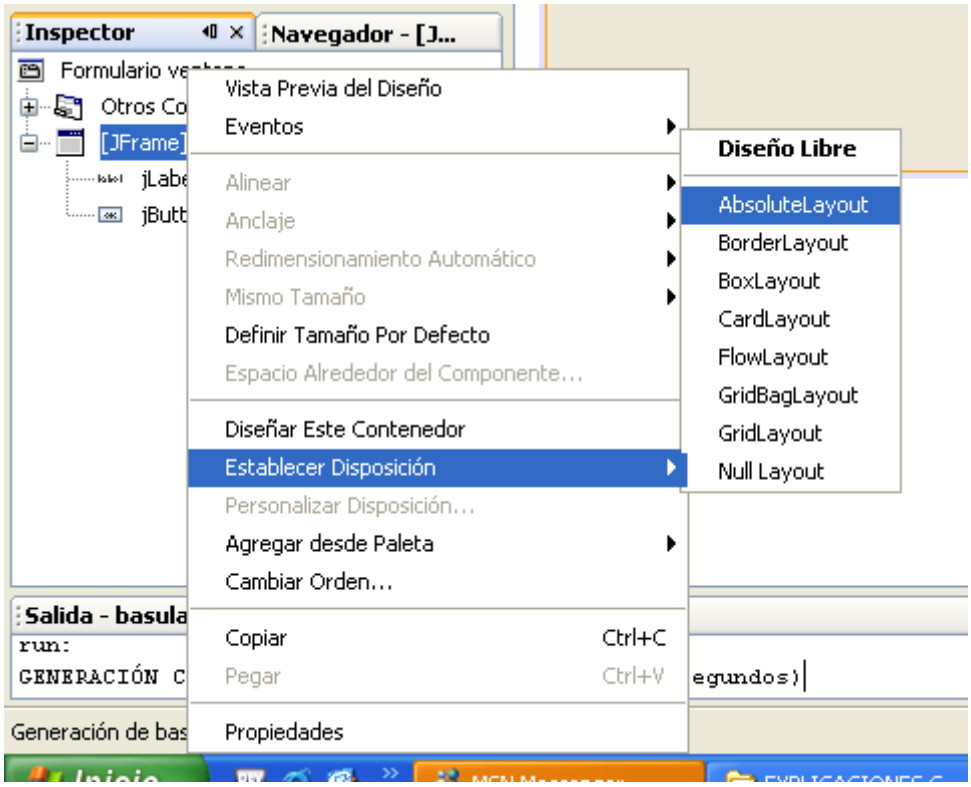
F d ió Códi Lib D i i

elementos de la ventana. Cuantos más elementos tengamos en una ventana, más difícil será el colocarlos usando el Diseño Libre.

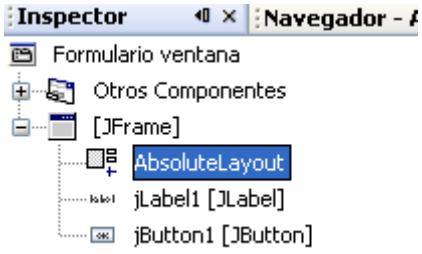
“AboluteLayout. Posiciones Absolutas”

- El Diseño Libre es la opción que está activada por defecto cuando se crea un proyecto en NetBeans. Sin embargo, esta opción se puede cambiar por distintos “Layouts” o “Distribuciones”.

- En el *Inspector* de tu proyecto pulsa el botón derecho del ratón sobre el objeto JFrame y activa la opción *Establecer Disposición – AbsoluteLayout*.



- El *Inspector* tendrá la siguiente forma ahora:



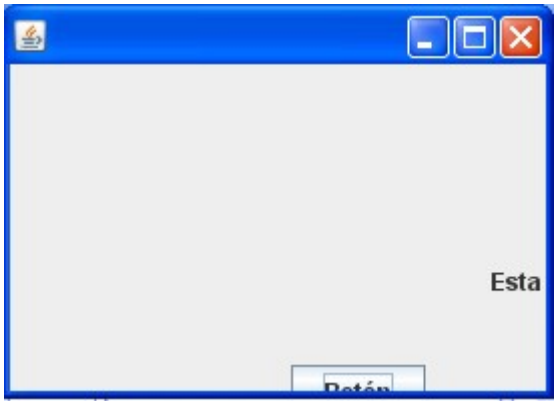
Como ves, aparece un objeto dentro del JFrame llamado AbsoluteLayout. Este objeto define otra forma de situar los elementos en la ventana. Concretamente, la distribución AbsoluteLayout permite al programador colocar cada elemento donde él quiera, sin restricciones, sin tener en cuenta distancias relativas.

F d ió Códi Lib D i i

- Sitúa la etiqueta y el botón donde quieras. Observa que no aparece ninguna línea guía que defina distancias relativas:

- La ventana de definir una distribución AbsoluteLayout es la facilidad para colocar cada elemento en la ventana (no tendrás los problemas del Diseño Libre). Sin embargo, la desventaja es que los elementos no mantienen una distribución relativa respecto al tamaño de la ventana.

- Ejecuta el programa y reduce su ancho. Observa lo que ocurre:

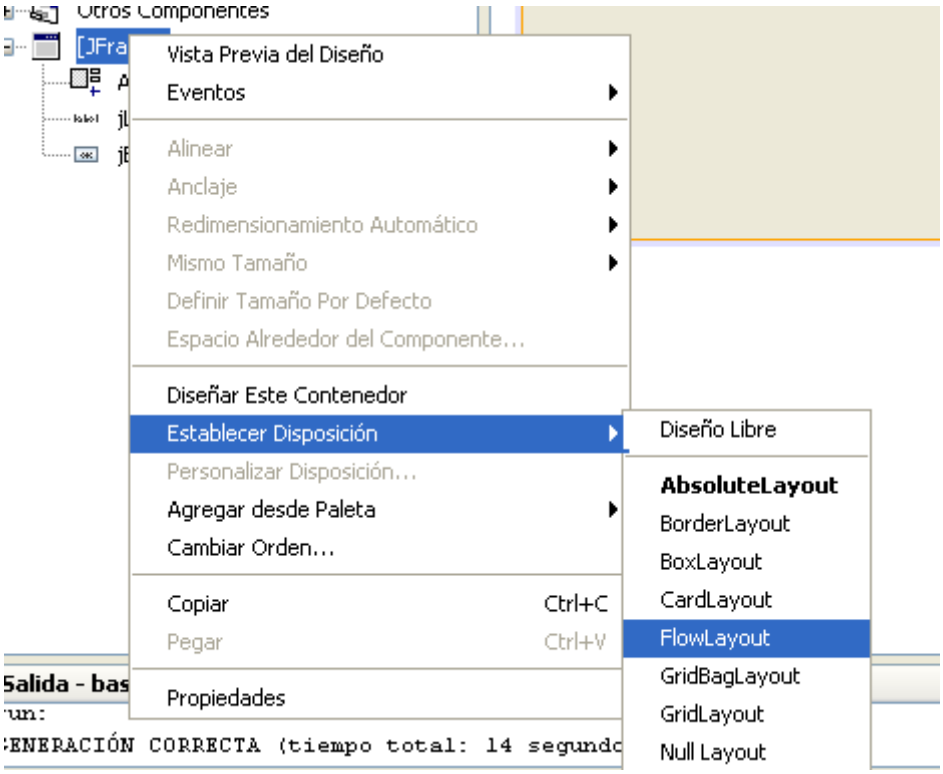


Verás que los elementos de la ventana son inamovibles aunque la ventana cambie de tamaño. En cambio, en el Diseño Libre los elementos intentaban siempre estar dentro de la ventana.

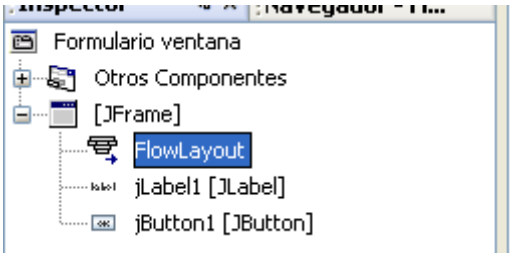
“Distribución en línea. FlowLayout”

F d ió Códi Lib D i i

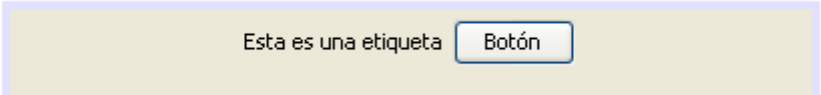
- Practiquemos ahora con otro tipo de distribución. Accede al *Inspector* y pulsa el botón derecho del ratón sobre el objeto JFrame. Activa la opción *Establecer Disposición – FlowLayout.*



- Observa como el layout “AbsoluteLayout” es sustituido por la distribución “FlowLayout”. Una elemento solo puede tener un tipo de distribución a la vez.

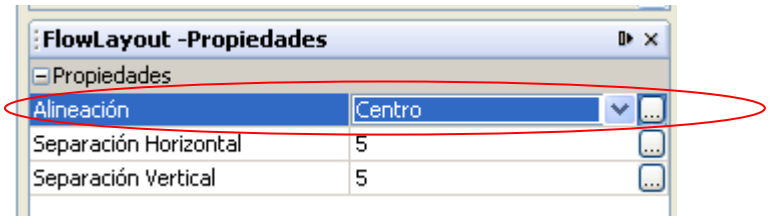


- Observa la ventana. Los elementos se han colocado uno detrás de otro. Se han colocado “en línea”. Esto es lo que hace el “FlowLayout”. Fuerza a los distintos elementos a que se coloquen en fila.

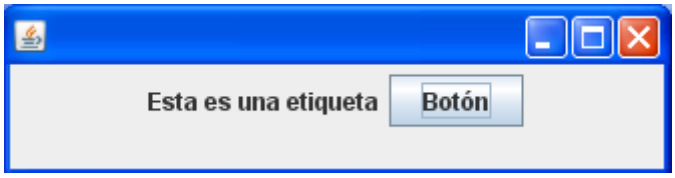


- Si seleccionas el FlowLayout en el *Inspector*, podrás acceder a sus propiedades (los layout son objetos como los demás) Una de las propiedades del FlowLayout se llama

alineación y permite que los elementos estén alineados a la izquierda, derecha o centro. El FlowLayout tiene una alineación centro por defecto.



- El FlowLayout no permite controlar la posición de los elementos en la ventana, pero sí procura que los elementos estén siempre visibles aunque la ventana se cambie de tamaño. Ejecuta el programa y observa el comportamiento del FlowLayout al agrandar o achicar la ventana:

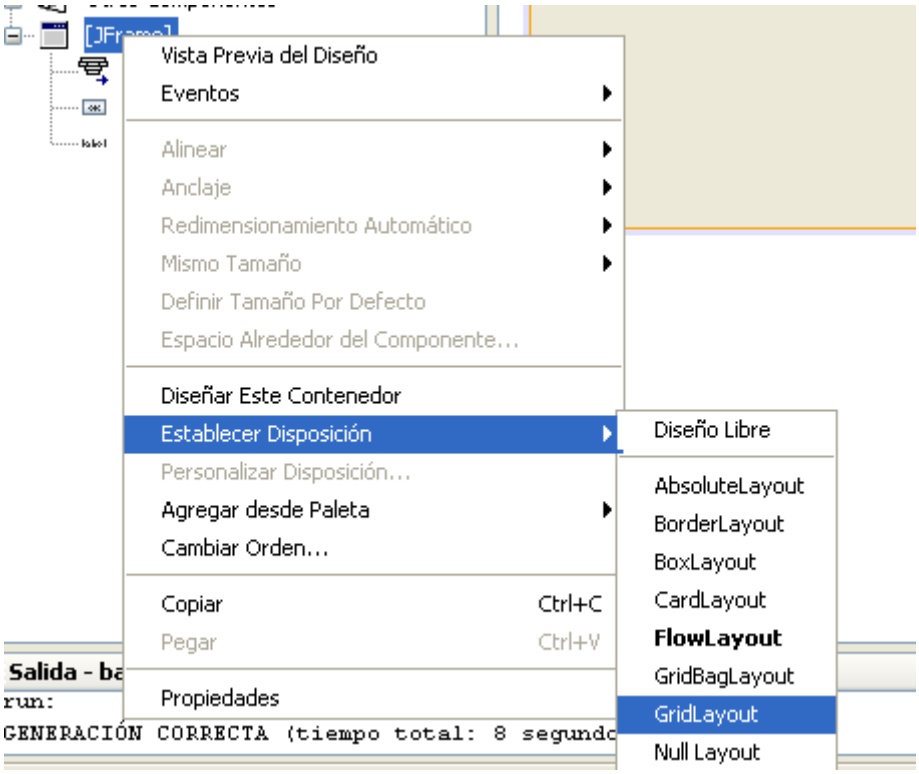


En el FlowLayout, los elementos intentan siempre estar dentro de la ventana, aunque esta se cambie de tamaño…

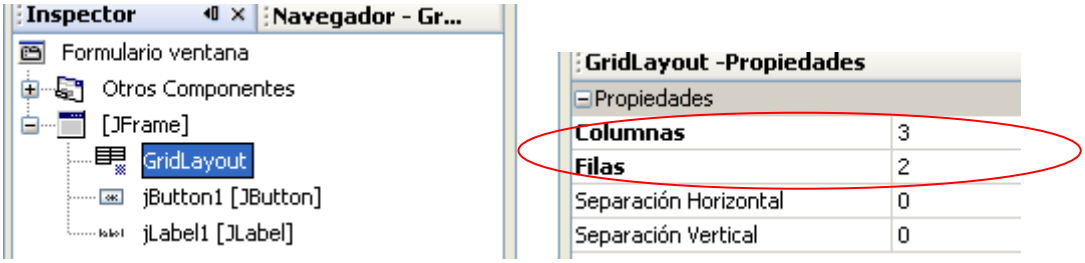
"Distribución en rejilla. GridLayout"

- Otra distribución que se puede usar es la distribución GridLayout. Esta distribución coloca a los elementos en filas y columnas, como si formaran parte de una tabla. Al añadir esta distribución es necesario indicar cuantas filas y columnas tendrá la rejilla.

- Cambia el layout del JFrame por un GridLayout:



- Marca el GridLayout y cambia sus propiedades Filas y Columnas. Asigna a la propiedad Filas un 2 y a la propiedad Columnas un 3.



- Al asignar 2 filas y 3 columnas al GridLayout, conseguiremos que los elementos de la ventana se distribuyan en una tabla como la siguiente:

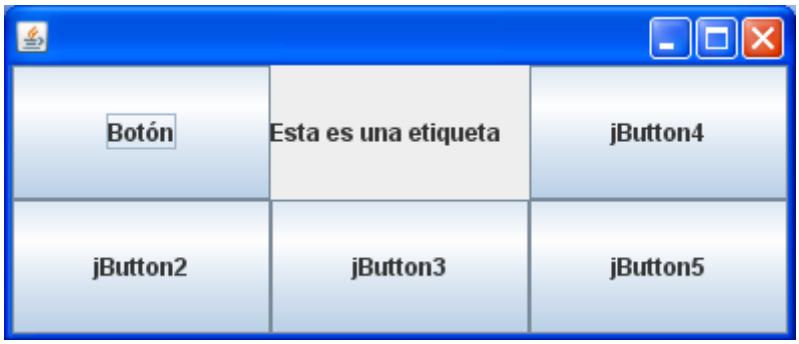
Los distintos elementos se adaptarán al espacio que tienen asignado, y cambiarán de tamaño.

F d ió Códi Lib D i i

- Ya que solo tienes dos elementos en la ventana (una etiqueta y un botón), añade otros cuatro elementos más (cuatro botones) para observar como se distribuyen en la cuadrícula.

- En un GridLayout, los elementos estarán situados siempre en una casilla de la rejilla, ocupando todo su espacio. El programador no tiene mucho control sobre la disposición de los elementos.

- Ejecuta el programa y agranda y achica la ventana. Observa como los elementos siempre mantienen su disposición en rejilla y siempre aparecen dentro de la ventana aunque el tamaño de esta varíe.



Con un GridLayout los elementos aparecen en filas y columnas.

Siempre aparecen dentro de la ventana aunque el tamaño de esta cambie.

“BorderLayout”

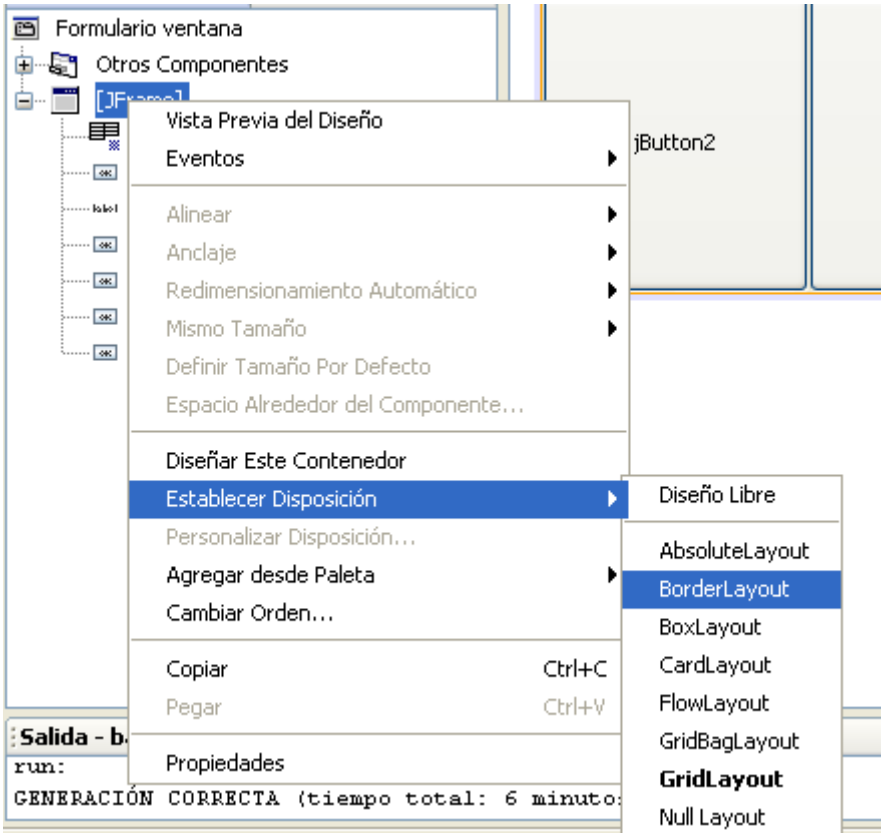
- Otra de las distribuciones posibles es la llamada BorderLayout. Esta distribución coloca los elementos de la ventana en cinco zonas:

  - Zona norte (parte superior de la ventana)
  - Zona sur (parte inferior de la ventana)
  - Zona este (parte derecha de la ventana)

F d ió Códi Lib D i i

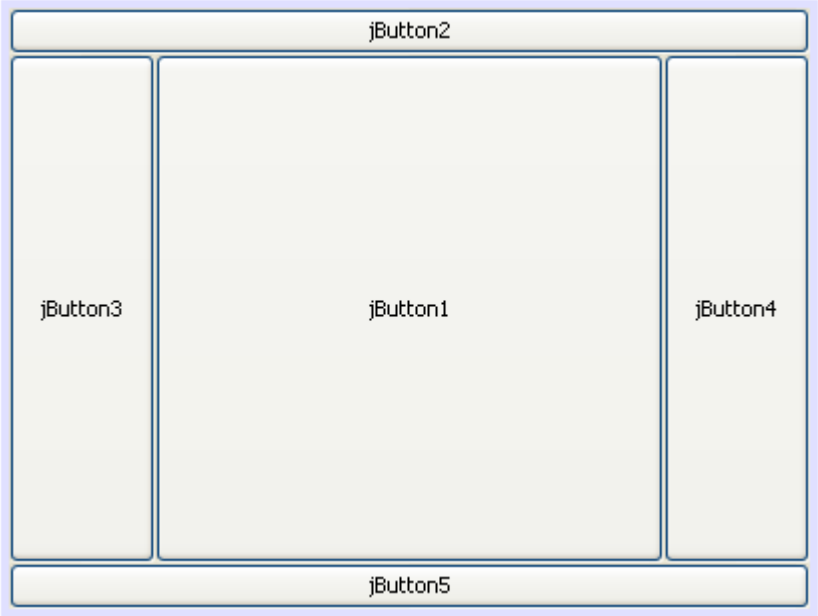
- Zona oeste (parte izquierda de la ventana)
- Zona centro.

- Haz clic con el derecho sobre el JFrame y asigna una distribución “BorderLayout”.



- Para poder entender el funcionamiento del BorderLayout, se recomienda que el JFrame contenga únicamente 5 botones (elimine los elementos que tiene ahora y añada cinco botones)

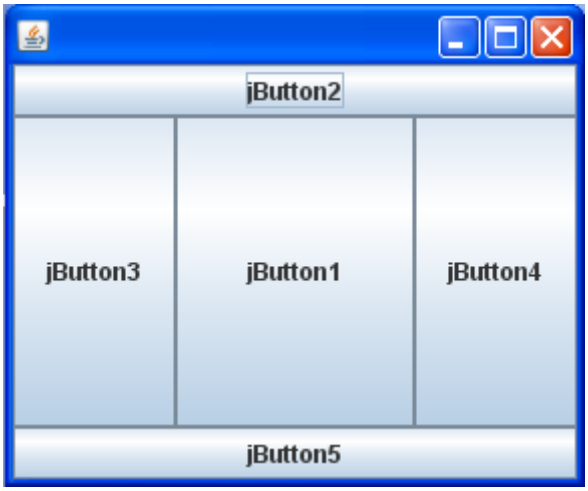
  La ventana tendrá un aspecto parecido al siguiente:



- Como se puede observar, cada botón se ha colocado en una zona, y su tamaño ha variado hasta ocupar la zona entera. Tenemos un botón en el norte, otro al sur, uno al este, otro al oeste y uno en el centro.

  El programador no tiene mucho control sobre la disposición de los elementos en la ventana al usar esta distribución.

- Ejecuta el programa y observa como los elementos siempre se mantienen dentro de la ventana aunque esta cambie de tamaño.



Con un GridLayout los elementos aparecen zonas.

Siempre aparecen dentro de la ventana aunque el tamaño de esta cambie.

F d ió Códi Lib D i i

CONCLUSIÓN

El diseño de la ventana viene definido por los Layouts o distribuciones.

Diseño Libre – Esta distribución viene activada por defecto en el NetBeans, y define una distribución de componentes en la que se respetan las distancias entre ellos cuando la ventana cambia de tamaño.

AbsoluteLayout – En esta distribución el programador puede colocar cada elemento en la posición que desee de la ventana. Los distintos elementos mantienen su posición aunque la ventana cambie de tamaño, lo que puede hacer que si la ventana se reduce de tamaño algunos elementos no se vean.

FlowLayout – En esta distribución los elementos se colocan uno detrás de otro. Los elementos intentarán estar dentro de la ventana aunque esta se reduzca de tamaño.

GridLayout – Esta distribución coloca a los elementos en filas y columnas. Los elementos siempre estarán dentro de la ventana aunque esta se reduzca de tamaño.

BorderLayout – Esta distribución coloca a los elementos en zonas. Los elementos siempre estarán dentro de la ventana aunque esta se reduzca de tamaño.

EJERCICIO GUIADO. JAVA: LAYOUTS Y PANELES

Técnicas de distribución de elementos en las ventanas

A la hora de diseñar una ventana se tienen en cuenta dos cosas:

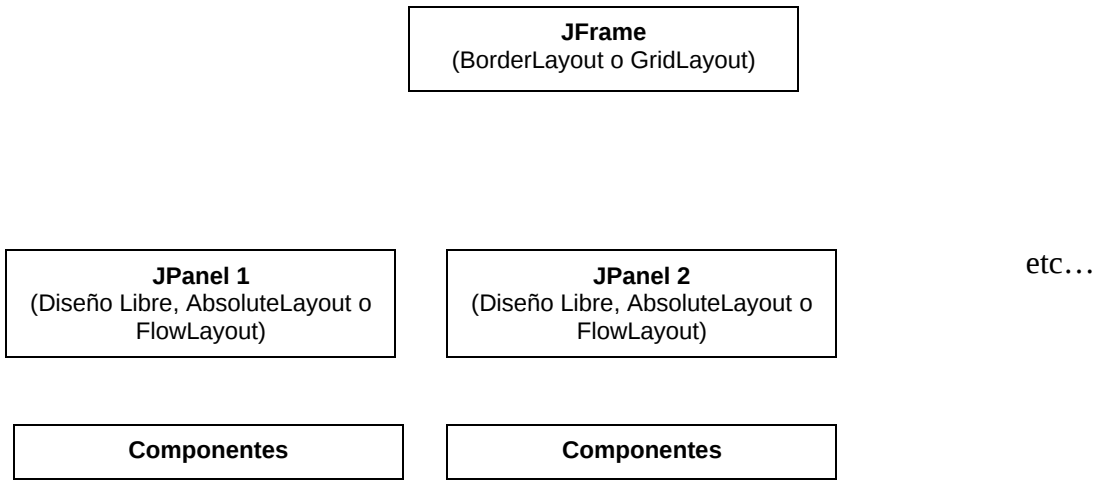
- La facilidad a la hora de colocar muchos componentes en la ventana.
- Que dichos componentes estén siempre visibles independientemente del tamaño de la ventana.

La distribución AbsoluteLayout por ejemplo nos da mucha facilidad a la hora de colocar los elementos en la ventana, pero sin embargo los componentes no se adaptan a los cambios de tamaño.

El Diseño Libre en cambio permite crear ventanas en las que sus componentes se "recolocan" según el tamaño de estas pero a cambio crece la dificultad del diseño.

Para aprovechar las ventajas de los distintos layouts y minimizar sus inconvenientes, es habitual en java crear una estructura de paneles cada uno de ellos con un layout distinto, según nuestras necesidades.

Normalmente, al JFrame se le asigna un layout que lo divida en zonas, como puede ser el BorderLayout o el GridLayout. Luego, dentro de cada una de estas zonas se introduce un panel (objeto JPanel). Y a cada uno de estos paneles se le asigna el layout que más le convenga al programador (FlowLayout, Diseño Libre, AbsoluteLayout, etc…) Finalmente, dentro de cada panel se añaden los componentes de la ventana.

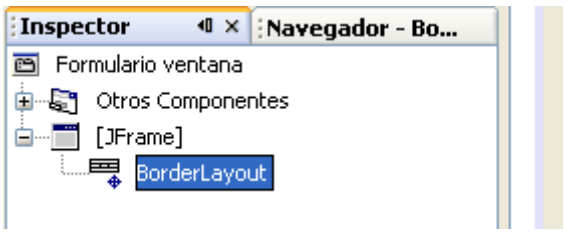


Ejercicio guiado

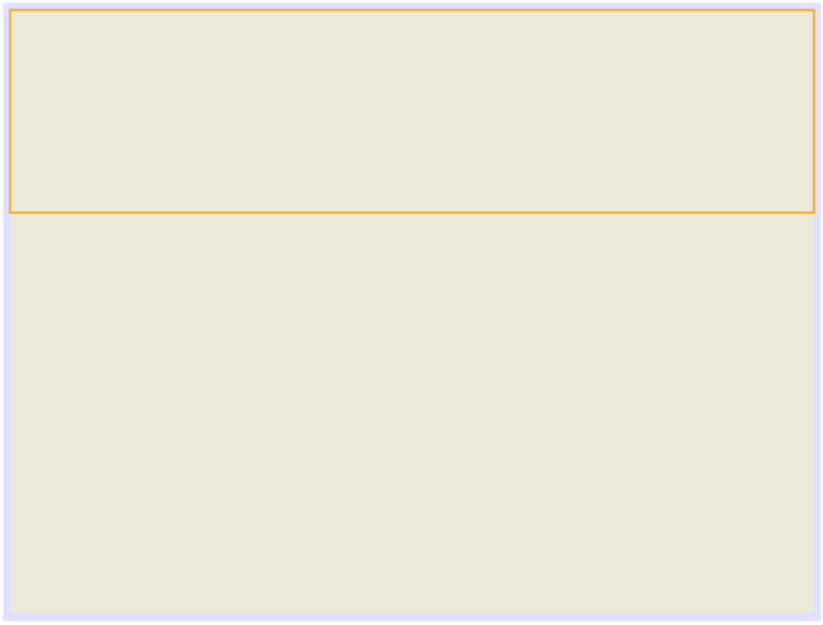
- Crea un nuevo proyecto en java.

  Se pretende crear un proyecto con una ventana de diseño complejo. Para ello sigue los siguiente pasos:

- En primer lugar, asigna un BorderLayout al JFrame:

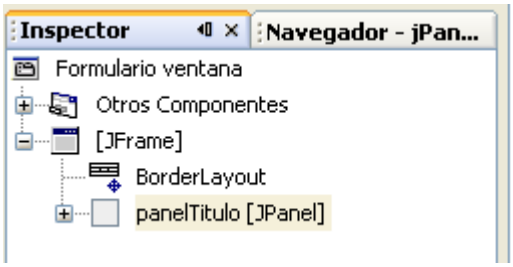


- El BorderLayout divide la ventana principal en zonas. Ahora añade un panel (JPanel) a la zona norte de la ventana.

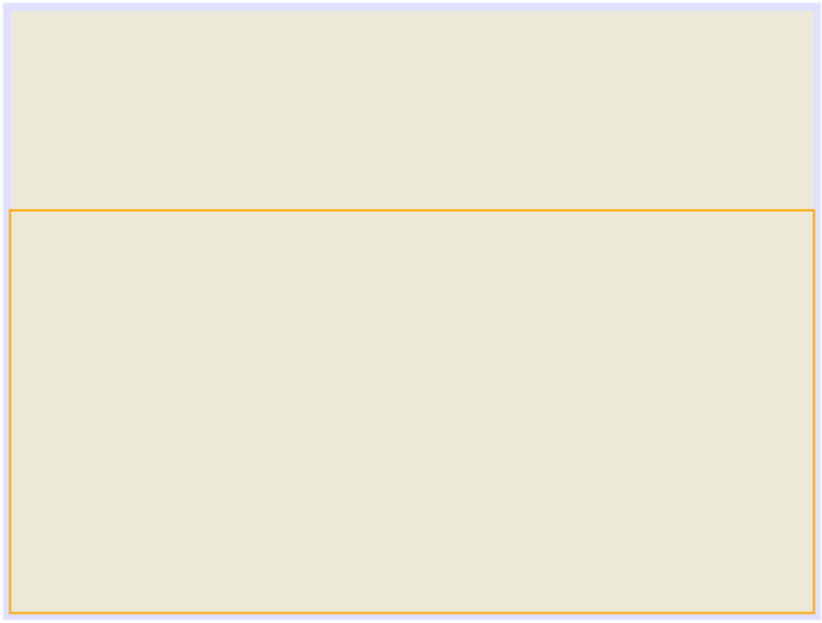


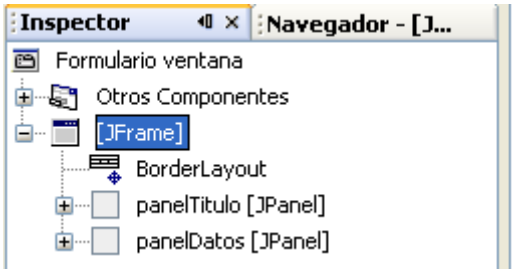
Panel en la zona norte.

- Cambia el nombre a este panel y llámalo *panelTitulo*, ya que contendrá el nombre del programa.



- Añade otro panel, esta vez a la parte central. El panel se llamará *panelDatos*:

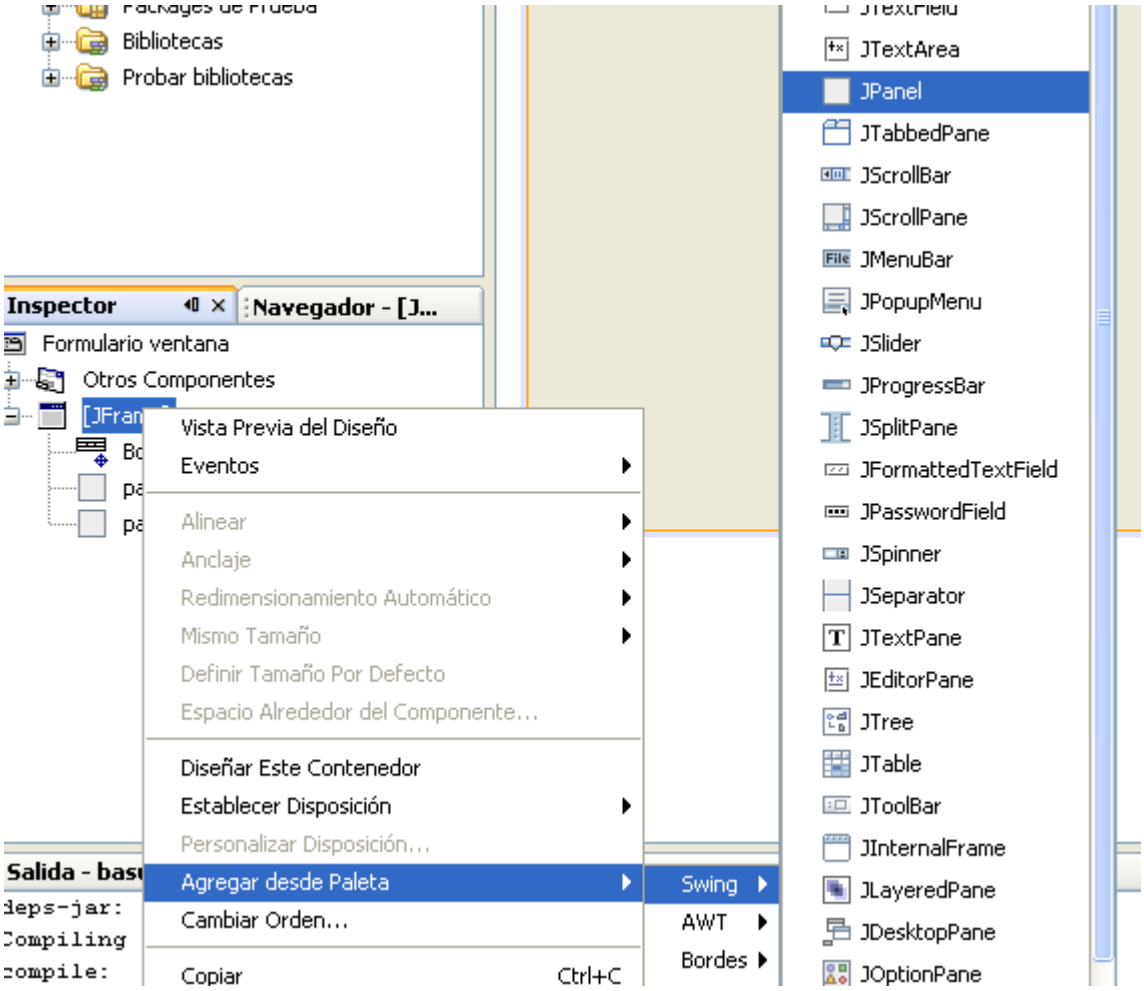




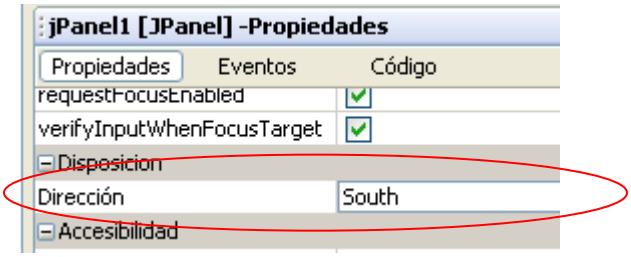
- Añade un nuevo panel en la parte sur de la ventana. Su nombre será *panelEstado*.

NOTA. A veces resulta complicado agregar un panel en una zona de la ventana cuando tenemos un BorderLayout. Puedes entonces hacer clic con el derecho sobre JFrame en el *Inspector* y activar la opción *Agregar desde paleta – Swing – JPanel*.

F d ió Códi Lib D i i

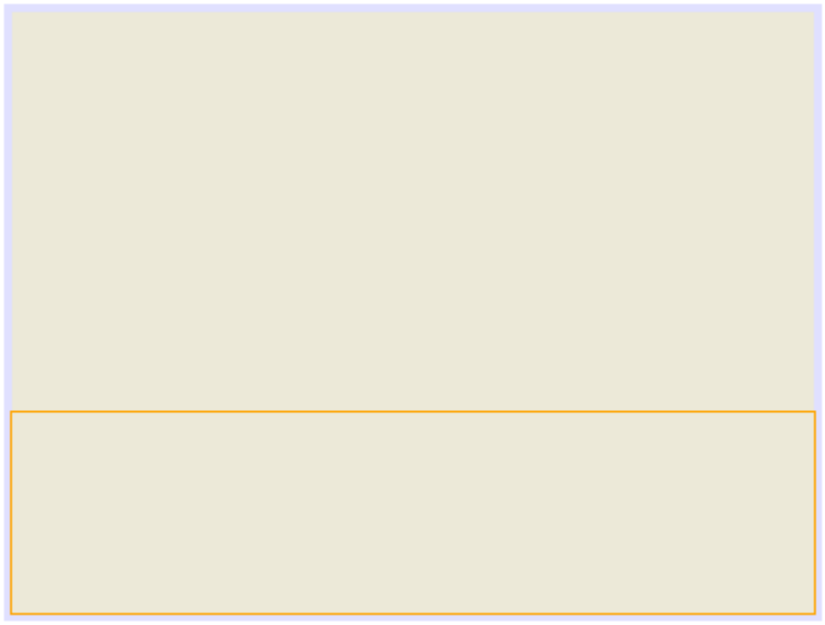


- Si el panel no se coloca en el sitio deseado, se puede seleccionar en el *Inspector* y activar su propiedad *Dirección*, e indicar la zona donde se quiere colocar:

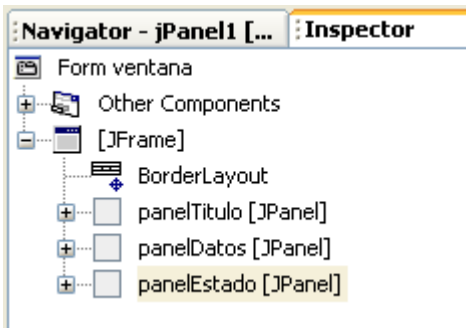


El panel debería estar situado finalmente en el sur de la ventana:

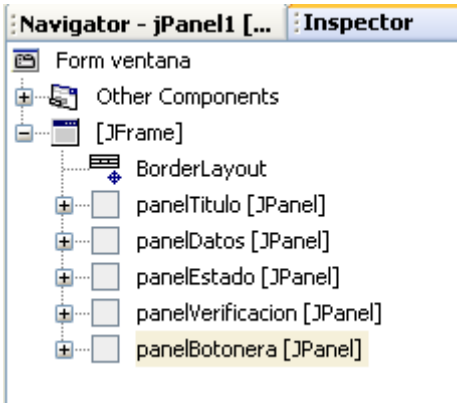
F d ió Códi Lib D i i



- El *Inspector* tendrá la siguiente forma ahora:



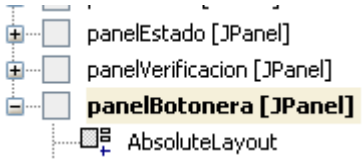
- Añade ahora tu solo un panel en la zona oeste llamado *panelBotonera* y otro en la zona esta llamado *panelVerificacion*. El *Inspector* debería tener la siguiente forma al finalizar:



F d ió Códi Lib D i i

- Cada panel puede ser diseñado de forma individual, simplemente haciendo doble clic sobre él. Así pues, empezaremos diseñando el panel *panelBotonera*. Haz doble clic sobre él.
- En la parte izquierda del NetBeans aparecerá únicamente el *panelBotonera*. Agrándalo para que tenga la siguiente forma:

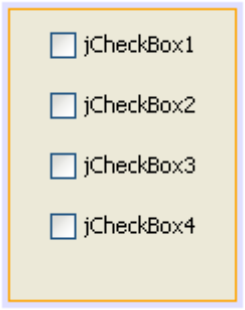
- A cada panel se le puede asignar un Layout distinto. A este panel le asignaremos un AbsoluteLayout para poder colocar cada elemento donde quiera. Asigna un AbsoluteLayout al panel haciendo clic con el derecho sobre él en el *Inspector*. El *Inspector* debería quedar así:



- Ahora añade cuatro botones al panel. Observa como tienes libertad total para colocar cada botón donde quieras. Procura que el panel quede así:

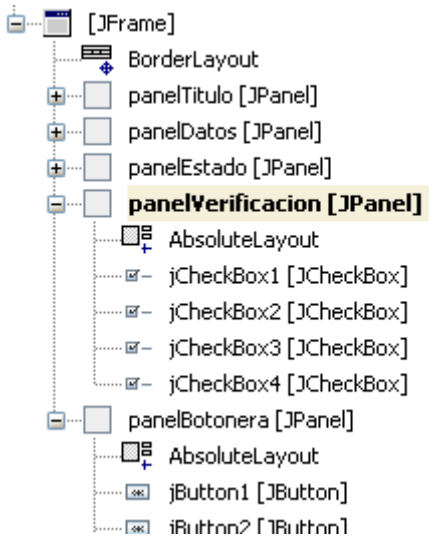
(No nos vamos a preocupar en este ejercicio de los nombres de los componentes)

- Ahora diseña el panel *panelVerificación* haciendo doble clic sobre él.
- Asígnale también un layout AbsoluteLayout.
- Coloca en él cuatro casillas de verificación. El aspecto del panel al terminar debe ser parecido al siguiente:



Y el *Inspector* debe tener un estado similar a este:

F d ió Códi Lib D i i

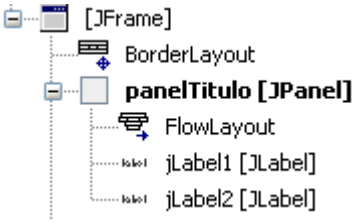


- Ahora se diseñará el *panelTitulo*. Haz doble clic sobre él.

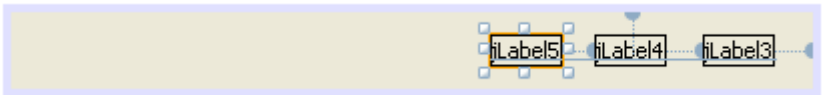
- En este caso se le añadirá un FlowLayout. Recuerda que este layout hace que cada elemento se coloque uno detrás de otro.

- Añade al panel dos etiquetas como las que siguen. Ponle un borde a cada una:

El *Inspector* tendrá este aspecto en lo que se refiere al *panelTitulo*...



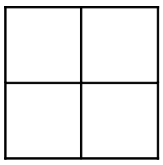
- El *panelEstado* lo diseñaremos sin asignar ningún layout, es decir, usando el *Diseño Libre*. En él añadiremos tres etiquetas de forma que estas mantengan una distancia relativa con respecto al límite derecho del panel. Dicho de otra forma, que siempre estén pegadas a la parte derecha del panel:



Observa las líneas “guía”. Indican que las etiquetas dependen de la parte derecha del panel. A su vez cada una depende de la otra. Es como si estuvieran “enganchadas”, como los vagones de un tren.

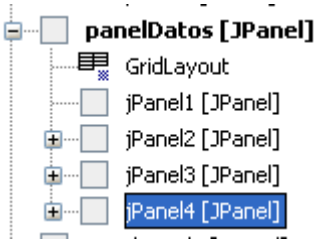
- El *panelDatos* lo vamos a complicar un poco. Haz doble clic sobre él para diseñarlo y asígnale un GridLayout.

- Marca el GridLayout y asígnale 2 filas y 2 columnas, para que interiormente tenga forma de una rejilla como esta:

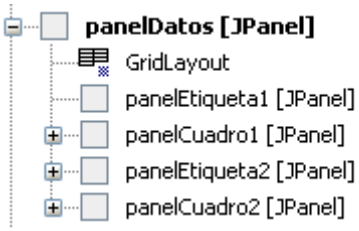


- A cada una de las divisiones del GridLayout del *panelDatos* le asignaremos un nuevo panel. Añade al *panelDatos* cuatro paneles. Esto lo puedes hacer fácilmente haciendo clic con el botón derecho del ratón sobre el *panelDatos* en el *Inspector* y eligiendo la opción *Añadir desde paleta – Swing – JPanel*.

  El aspecto del inspector debería ser como el que sigue, en lo que se refiere al *panelDatos*:



- Asignaremos a cada uno de los cuatro paneles los siguientes nombres: *panelEtiqueta1, panelCuadro1, panelEtiqueta2, panelCuadro2*. El panel quedará así en el *Inspector*.



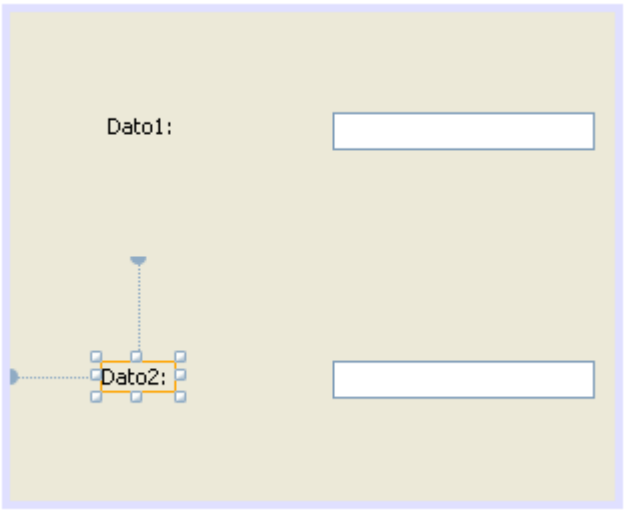
Así pues, el panel *panelDatos* tiene forma de rejilla con cuatro celdas, y en cada celda hay un panel. Puede imaginarse el panelDatos así:

panelDatos

F d ió Códi Lib D i i

PanelEtiqueta1	PanelCuadro1
PanelEtiqueta2	PanelCuadro2

- Ahora añada al panelEtiqueta1 y al panelEtiqueta2 sendas etiquetas. Y al panelCuadro1 y panelCuadro2 sendos cuadros de textos. El panel *panelDatos* debe quedar así:



- Finalmente ejecuta el programa y comprueba como se comportan los elementos según el panel donde se encuentre y el layout asignado a cada uno.

CONCLUSIÓN

Para el diseño de ventanas muy complejas, ser suelen definir layouts que dividan en zonas el JFrame, como por ejemplo el BorderLayout o el GridLayout.

Dentro de cada una de dichas zonas se añade un JPanel, al que se le asigna un AbsoluteLayout, un FlowLayout o se mantiene el Diseño Libre.

Es posible asignar a un panel un layout de zonas, como el GridLayout, y, a su vez, introducir en él nuevos paneles, y así sucesivamente.

EJERCICIO GUIADO. JAVA: DIÁLOGOS

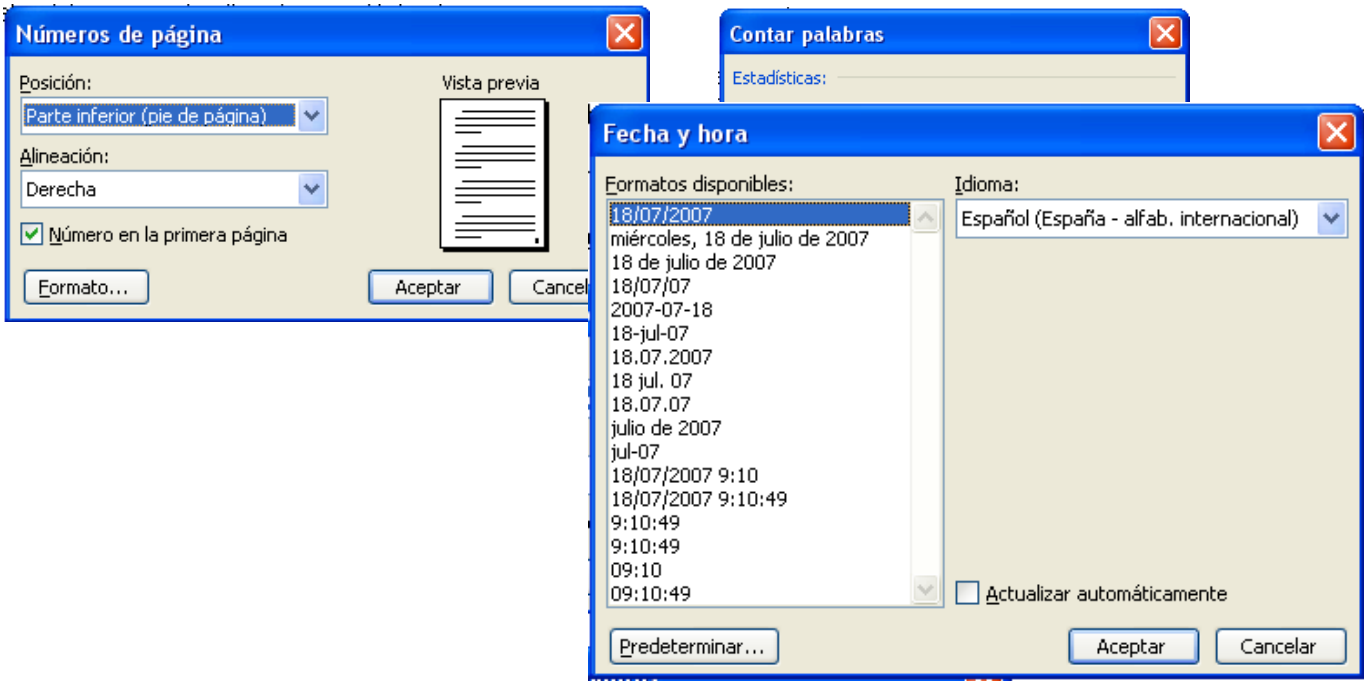
Cuadros de Diálogo

Un cuadro de diálogo es un cuadro con opciones que aparece normalmente cuando se activa una opción del menú principal del programa.

Los cuadros de diálogo tienen forma de ventana aunque no poseen algunas características de estas. Por ejemplo, no pueden ser minimizados ni maximizados.

Los cuadros de diálogo, aparte de las opciones que muestran, suelen contener dos botones típicos: el botón Aceptar y el botón Cancelar. El primero de ellos da por válidas las opciones elegidas y cierra el cuadro de diálogo. El segundo simplemente cierra el cuadro de diálogo sin hacer ninguna modificación.

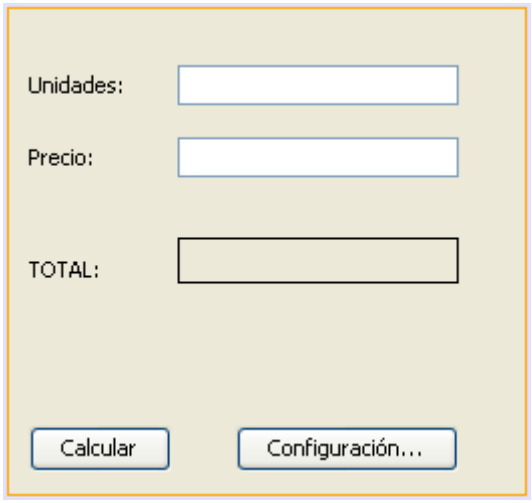
He aquí algunos ejemplos de cuadros de diálogo del programa Word:



Para crear cuadros de diálogo en Java, se usa un tipo de objetos llamado JDialog. Estos objetos pueden ser diseñados como si fueran ventanas, aunque representan realmente cuadros de diálogo.

Ejercicio guiado

- Crea un nuevo proyecto en java.
- Diseña el JFrame de forma que la ventana tenga el siguiente aspecto:



Los elementos de la ventana tendrán los siguientes nombres:

- Cuadro de texto de unidades: txtUnidades.
- Cuadro de texto de precio: txtPrecio.
- Etiqueta con borde del total: etiTotal.
- Botón Calcular: btnCalcular.
- Botón Configuración: btnConfiguracion.

- Se pretende que cuando se pulse el botón Calcular se calcule el total de la venta (esto se hará luego) Para hacer el cálculo se tendrán en cuenta el IVA y el Descuento a aplicar. Estos dos valores serán variables globales, ya que se usarán en distintos lugares del programa.

- Así pues entra en el código y declara una variable global *iva* y otra *descuento* tal como se indica a continuación (recuerda que las variables globales se colocan justo después de la línea donde se define la clase principal *public class*):

F d ió Códi Lib D i i

```
 * @author  didact
 */
public class ventanaprincipal extends javax.swing.JFrame {

    double iva;
    double descuento;


    /** Creates new form ventanaprincipal */
    public ventanaprincipal() {
        initComponents();
    }
```

Variables globales

- Cuando el programa arranque, interesará que el *iva* por defecto sea 0, y que el *descuento* por defecto sea 0 también, así que en el constructor, inicializaremos las variables globales *iva* y *descuento* a 0:

```
 */
public class ventanaprincipal extends javax.swing.JFrame {

    double iva;
    double descuento;


    /** Creates new form ventanaprincipal */
    public ventanaprincipal() {
        initComponents();
        iva=0;
        descuento=0;
    }
```

Inicialización de variables globales

- Estamos ya preparados para programar el botón btnCalcular. Entra en su actionPerformed y allí se programará la realización del cálculo de la siguiente forma:

```
double unidades;
double precio;
double total;    //total
double cantiva;  //cantidad iva
double cantdes;  //cantidad descuento
double totalsiniva; //total sin iva

//Recojo los datos de los cuadros de textos (convirtiendolos a números)
unidades = Double.parseDouble(txtUnidades.getText());
precio = Double.parseDouble(txtPrecio.getText());

//Calculo el total sin iva, la cantidad de iva y la cantidad de descuento
totalsiniva=precio*unidades;
cantiva=totalsiniva*iva/100;
cantdes=totalsiniva*descuento/100;
```

F d ió Códi Lib D i i

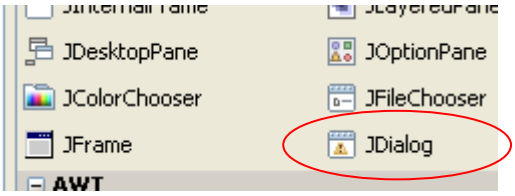
```
//Ahora calculo el precio total:
total = totalsiniva+cantiva-cantdes;

//Coloco el total en la etiqueta:
etiTotal.setText(""+total);
```

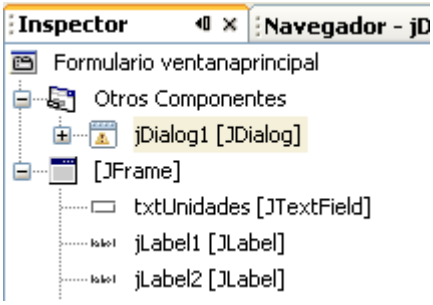
- Puedes ya ejecutar el programa y comprobar que el botón Calcular funciona, aunque el cálculo que realiza lo hace con un iva 0 y un descuento 0.

- A continuación se programará el botón Configuración de forma que nos permita decidir qué iva y qué descuento queremos aplicar. Este botón mostrará un CUADRO DE DIÁLOGO que permita al usuario configurar estos datos.

- Para añadir un cuadro de diálogo al proyecto, se tiene que añadir un objeto del tipo JDialog sobre el JFrame.

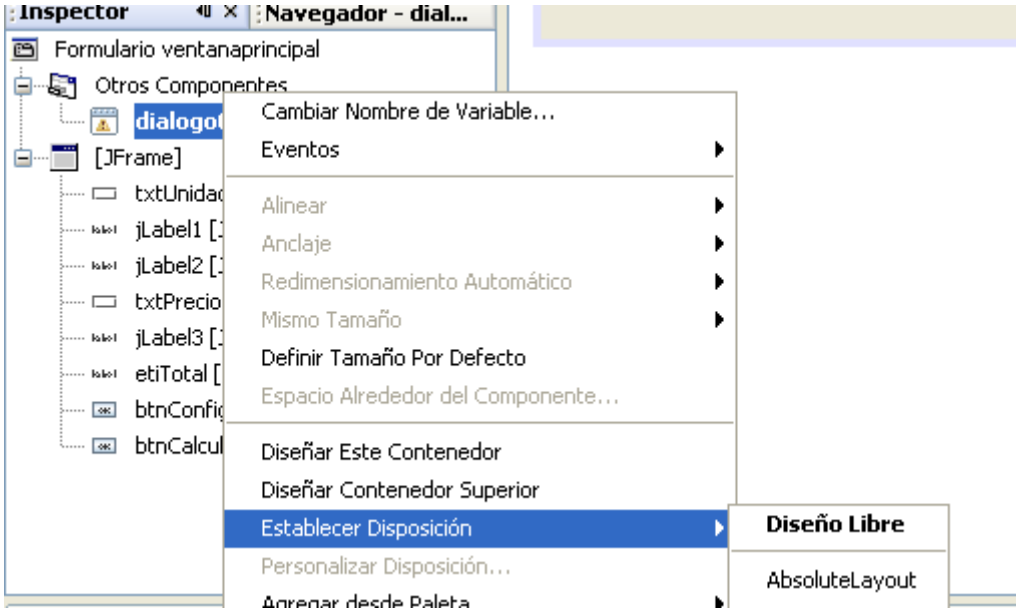


- Los JDialog son objetos ocultos, es decir, objetos que se colocan en la parte del *Inspector* llamada *Otros Componentes*, al igual que sucede con los menús contextuales o los JFileChooser. Observa tu inspector, allí verás el JDialog que has añadido:

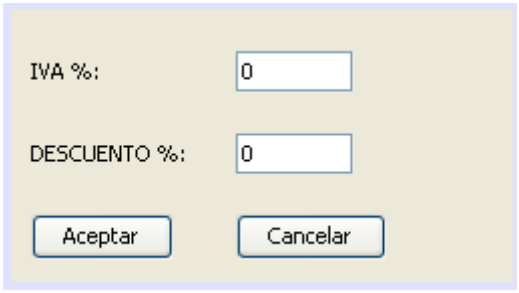


- Cámbiale el nombre. Lo llamaremos *dialogoConfiguracion*.

- Los diálogos normalmente traen por defecto el layout BorderLayout. Para nuestro ejemplo cambiaremos el layout del JDialog por el Diseño Libre:



- Los JDialog se pueden diseñar independientemente, al igual que los JPanel. Solo tienes que hacer doble clic sobre el *dialogoConfiguracion* (en el *inspector*) y este aparecerá en el centro de la ventana.
- Así pues debes diseñar el *dialogoConfiguracion* para que quede de la siguiente forma:

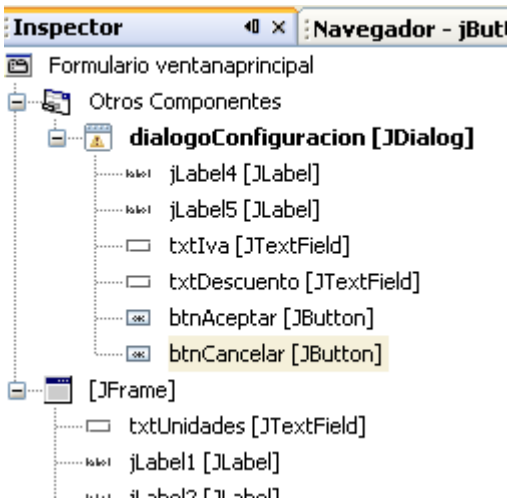


Los elementos del cuadro de diálogo tienen los siguientes nombres:

- El cuadro de texto del Iva: txtIva.
- El cuadro de texto del Descuento: txtDescuento.
- El botón Aceptar: btnAceptar.
- El botón Cancelar: btnCancelar.

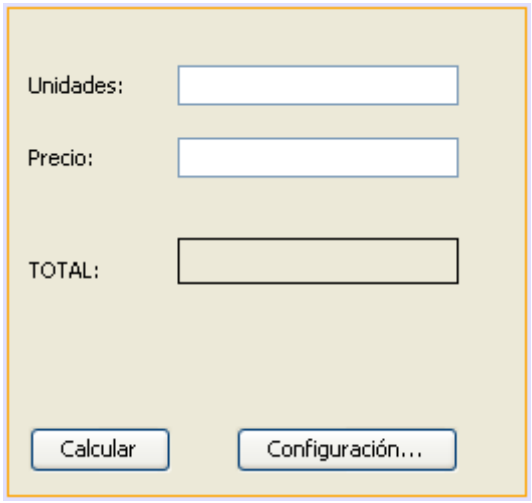
Si observas el *Inspector* debe tener el siguiente aspecto:

F d ió Códi Lib D i i

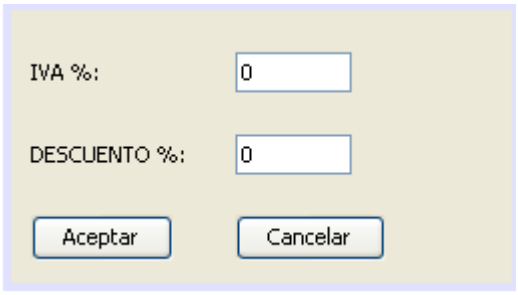


F d ió Códi Lib D i i

- Se ha dicho que cuando se pulse el botón Configuración en la ventana principal, debe aparecer el cuadro de diálogo *dialogoConfiguracion*, que acabas de diseñar:



Haces clic sobre Configuración y aparece el diálogo



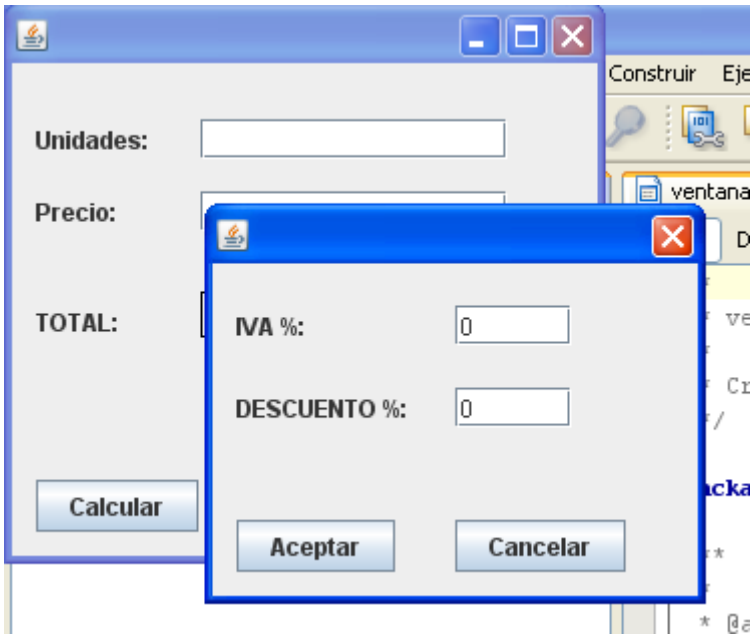
- Para conseguir esto, debes programar el *actionPerformed* del botón *btnConfiguracion* de la siguiente forma:

```
dialogoConfiguracion.setSize(250,200);
dialogoConfiguracion.setLocation(100,100);
dialogoConfiguracion.setVisible(true);
```

- El código anterior hace lo siguiente:

  - A través del método *setSize* se asigna un tamaño de 250 x 200 al cuadro de diálogo.
  - A través del método *setLocation* se determina que el cuadro de diálogo aparecerá en la posición (100, 100) de la pantalla.
  - A través del método *setVisible* hacemos que el cuadro de diálogo se muestre.

- Ejecuta el programa y observa lo que sucede cuando pulsas el botón Configurar. Debería aparecer el cuadro de diálogo en la posición programada y con el tamaño programado:



- Los botones Aceptar y Cancelar del cuadro de diálogo aún no hacen nada. Así que los programaremos. Empezaremos por el más sencillo, el botón Cancelar.

- El botón Cancelar de un cuadro de diálogo simplemente cierra dicho cuadro de diálogo. Para ello, debes añadir el siguiente código en el *actionPerformed* del botón Cancelar del diálogo:

  ```
  dialogoConfiguracion.dispose();
  ```

  El método *dispose* se usa para cerrar un cuadro de diálogo. También se puede usar con un JFrame para cerrarlo.

- Ejecuta el programa de nuevo y comprueba el funcionamiento del botón Cancelar del cuadro de diálogo.

- Ahora se programará el botón Aceptar. Cuando el usuario pulse este botón, se confirmará el valor del *iva* y del *descuento* que haya introducido. Es decir, se traspasarán los valores introducidos en los cuadros de texto txtIva y txtDescuento a las variables globales *iva* y *descuento*.

  Una vez que se haya hecho esto, el cuadro de diálogo se debe cerrar.

- Este es el código que hace lo anterior. Debe programarlo en el *actionPerformed* del botón Aceptar:

F d ió Códi Lib D i i

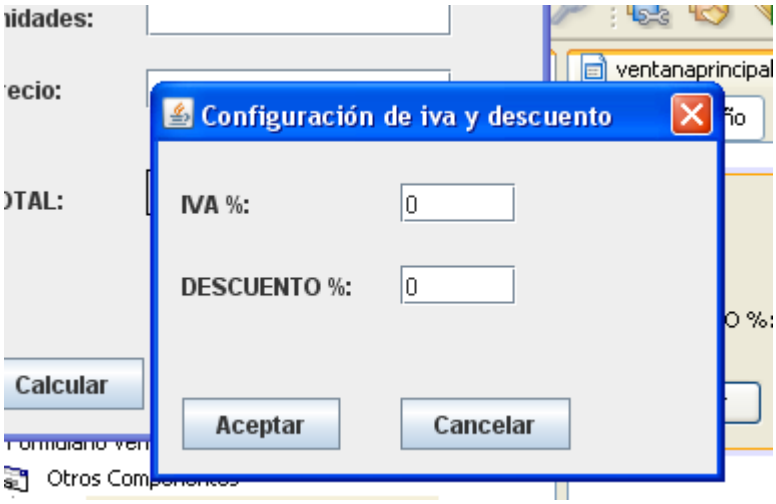
```
 iva = Double.parseDouble(txtIva.getText());
descuento=Double.parseDouble(txtDescuento.getText());
dialogoConfiguracion.dispose();
```

- Observe el código. Primero se traspasa los valores de los cuadros de texto a las variables globales y luego se cierra el cuadro de diálogo.
- Compruebe el funcionamiento del programa de la siguiente forma:

    - Ejecute el programa.
    - Introduzca 5 unidades y 20 de precio.
    - Si pulsa calcular, el total será 100. (No hay ni iva ni descuento al empezar el programa)
    - Ahora pulse el botón Configuración, e introduzca un iva del 16. El descuento déjelo a 0. Acepte.
    - Ahora vuelva a calcular. Observe como ahora el total es 116, ya que se tiene en cuenta el iva configurado.
    - Pruebe a configurar un descuento y vuelva a calcular.

- Se pretende ahora mejorar un poco el cuadro de diálogo, añadiéndole un título. Seleccione el cuadro de diálogo en el *Inspector* y luego busque su propiedad *title*. En ella escriba "*Configuración de iva y descuento*".

- Vuelva a ejecutar el programa. Observe la barra de título del cuadro de diálogo:



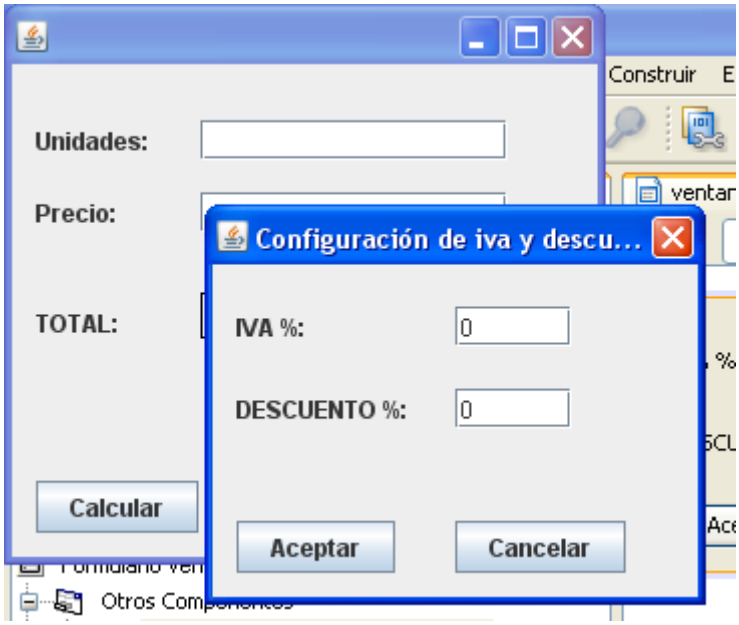
- Ahora se estudiará el concepto de *cuadro de diálogo modal* y *cuadro de diálogo no modal*.

1. Un cuadro de diálogo *no modal*. Es aquel que permite activar la ventana desde la que apareció. Los cuadros de diálogo añadidos a un proyecto son por defecto *no modales*.

- Ejecuta el programa y prueba a hacer lo siguiente:

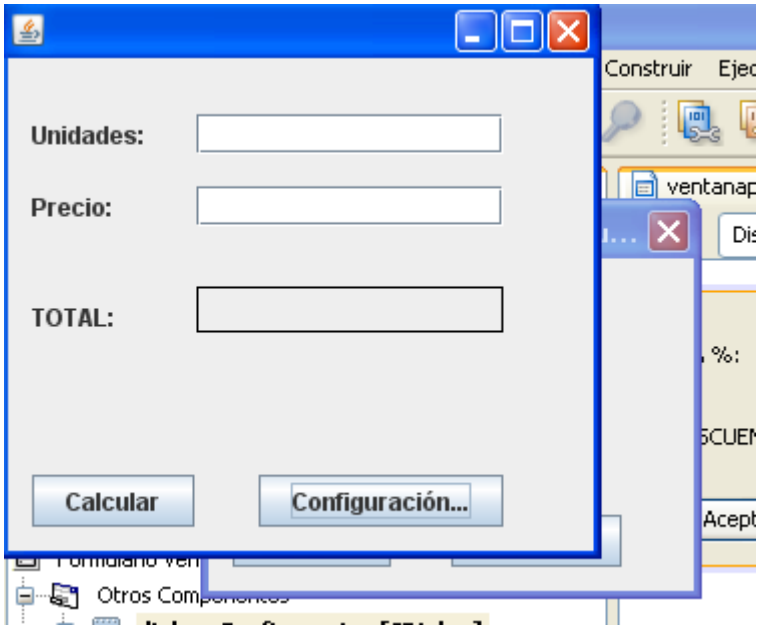
    - Pulsa el botón Configurar. Aparecerá el cuadro de diálogo.

- Pulsa sobre la ventana.



Pulsa sobre la ventana.

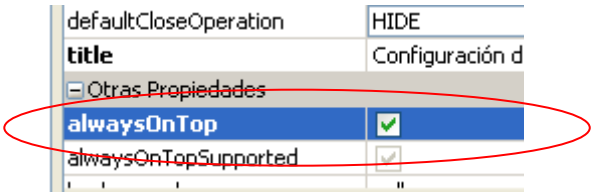
- Observarás que la ventana se activa, colocándose sobre el cuadro de diálogo.



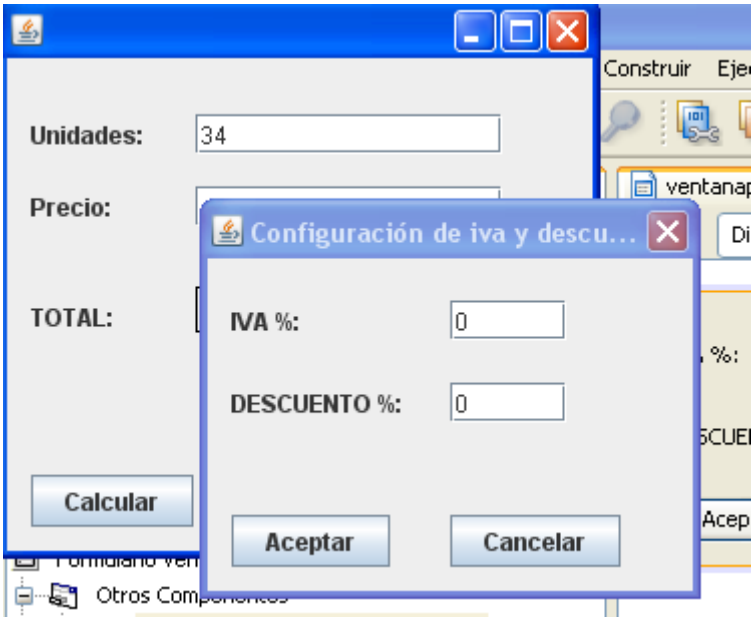
La ventana se activa colocándose por encima del cuadro de diálogo.

- Esto es posible gracias a que el cuadro de diálogo es *no modal*.

- A veces, puede ser interesante que se active la ventana pero que el cuadro de diálogo siga delante de ella. Para conseguir esto, es necesario activar la propiedad del cuadro de diálogo llamada *alwaysOnTop*. Activa esta propiedad:



- Ahora ejecuta el programa de nuevo y haz que se visualice el cuadro de diálogo de configuración. Podrás comprobar que se puede activar la ventana e incluso escribir en sus cuadros de textos, y que el cuadro de diálogo sigue visible:



Se puede activar la ventana trasera, e incluso escribir en ella. Esto es gracias a que el cuadro de diálogo es *no modal.*

Por otro lado, el cuadro de diálogo sigue mostrándose delante de la ventana. Esto es gracias a la propiedad *alwaysOnTop*

- Es muy común, cuando tenemos un cuadro de diálogo *no modal*, usar la propiedad *alwaysOnTop*, para que siempre aparezca delante de la ventana.

- Ahora se estudiará el concepto de cuadro de diálogo *modal*.

2. Un cuadro de diálogo *modal* es aquel que no permite que se active otra ventana hasta que este no se haya cerrado.

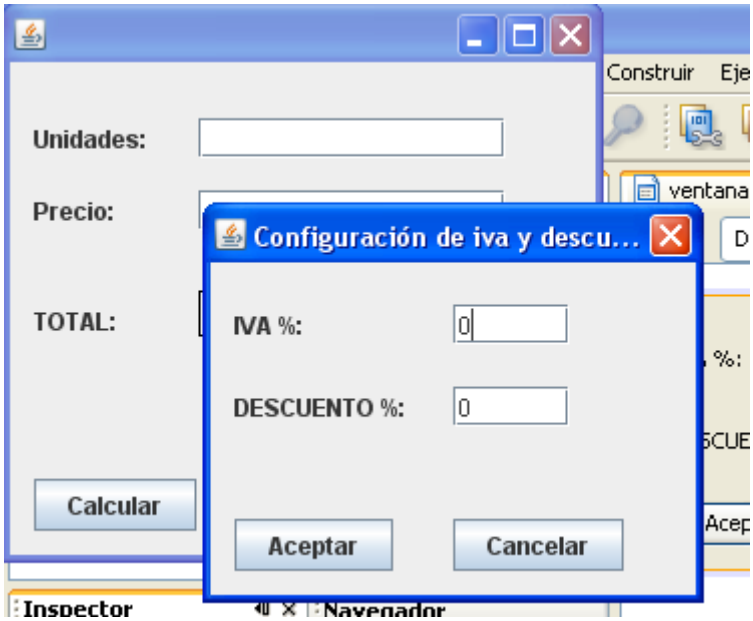
- Para convertir nuestro cuadro de diálogo en *modal*, será necesario que lo selecciones en el *inspector* y busques la propiedad *modal*. Debes activar esta propiedad.

F d ió Códi Lib D i i

- Ahora ejecuta el programa comprueba lo siguiente:

    - Haz que se visualice el cuadro de diálogo de configuración.

    - A continuación intenta activar la ventana haciendo clic sobre ella. Verás como no es posible activarla. Es más, intenta escribir en sus cuadros de texto. No será posible hacerlo. (Incluso observarás un parpadeo en el cuadro de diálogo avisándote de ello). Esto es debido a que ahora nuestro cuadro de diálogo es *modal*.



Aunque intentes activar la ventana o escribir en ella, no podrás, ya que el cuadro de diálogo es *modal*.

Incluso verás un parpadeo en el cuadro de diálogo cuando intentas activar la otra ventana.

Se podría decir que un cuadro de diálogo *modal* es un acaparador, y que no te deja usar otro elemento hasta que no acabes con él.

Solo cuando cierres el cuadro de diálogo podrás seguir trabajando con la ventana.

    - Solo cuando pulses, Aceptar, o Cancelar, o cierres el cuadro de diálogo, podrás seguir trabajando con tu ventana.

CONCLUSIÓN

Los Cuadros de Diálogo son ventanas simplificadas que muestran distintas opciones al usuario.

Los objetos JDialog son los que permiten la creación y uso de cuadros de diálogo en un proyecto java.

Para visualizar un JDialog será necesario llamar a su método *setVisible*. También son interesantes los métodos *setSize* para asignarles un tamaño y *setLocation* para situar el cuadro de diálogo en la pantalla.

Para cerrar un JDialog será necesario invocar a su método dispose.

F d ió Códi Lib D i i

Existen dos tipos de cuadros de diálogo: los modales y no modales.

Los cuadros de diálogo modales no permiten que se active otra ventana hasta que el cuadro de diálogo no se haya cerrado.

Los cuadros de diálogo no modales permiten trabajar con otra ventana a pesar de que el propio cuadro de diálogo no haya sido cerrado.

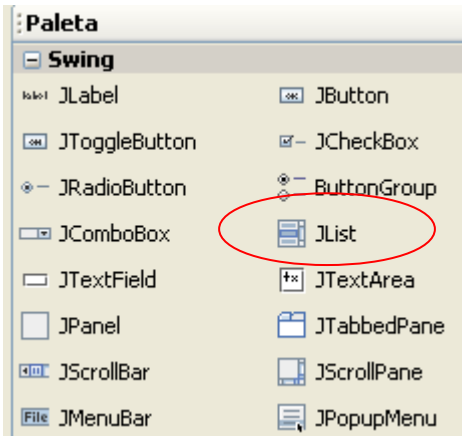
EJERCICIO GUIADO. JAVA: MODELOS DE CUADRO DE LISTA

127. Realiza un nuevo proyecto.

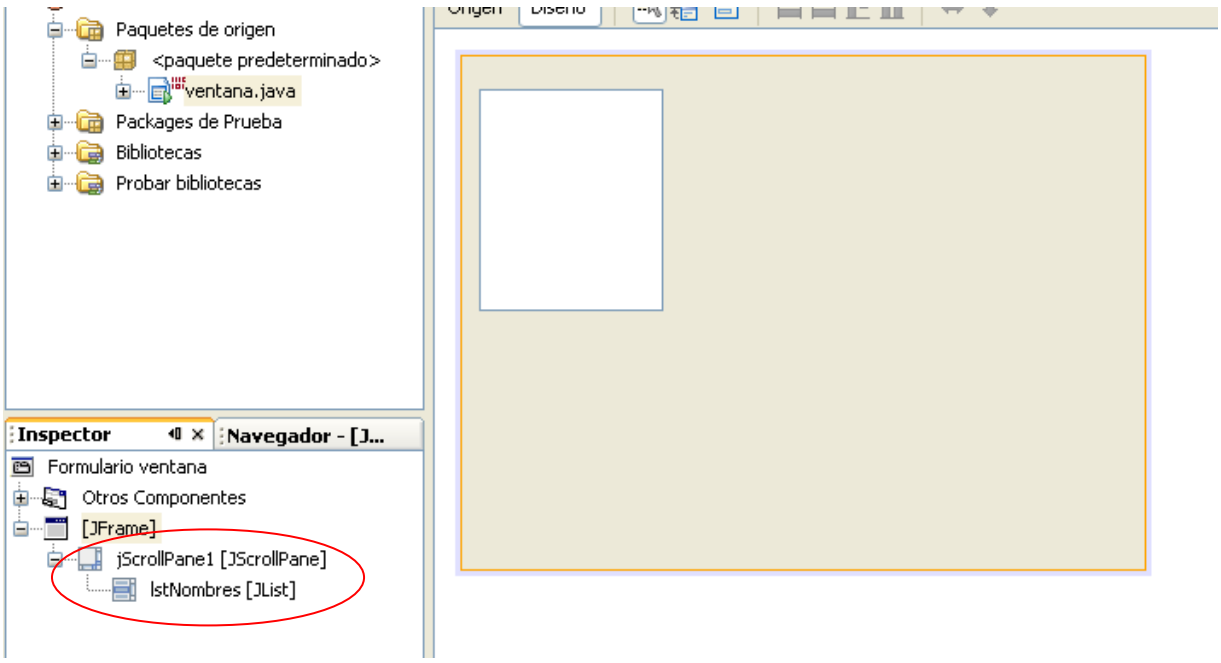
128. En la ventana principal debes añadir lo siguiente:

yyyyy. Una etiqueta con borde llamada etiResultado.

130. Añade un cuadro de lista al formulario (JList).

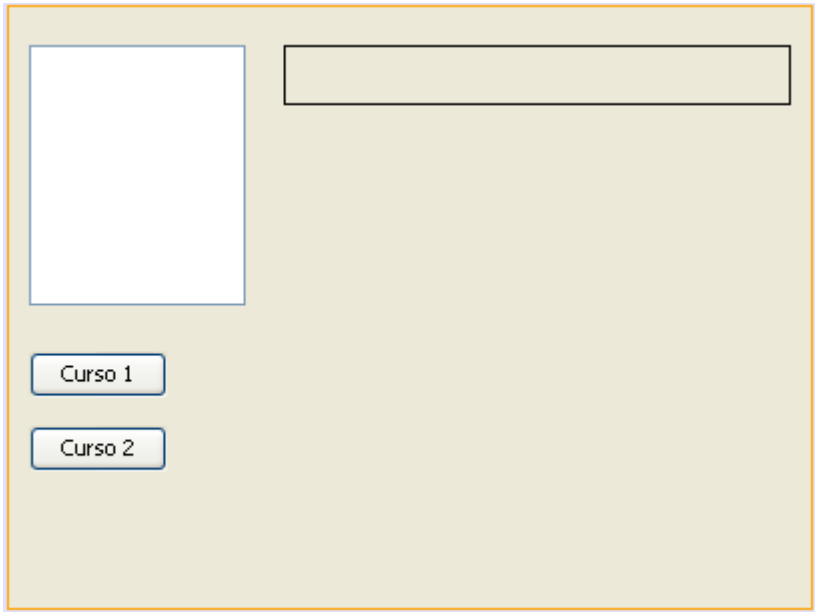


131. Borra todo el contenido de la lista (propiedad model) y cámbiale el nombre a la lista. La lista se llamará *lstNombres*. Recuerda que las listas aparecen dentro de un objeto del tipo JScrollPane.



F d ió Códi Lib D i i

132. Añade dos botones al formulario. Uno de ellos tendrá el texto “Curso 1” y se llamará btnCurso1 y el otro tendrá el texto “Curso 2” y se llamará btnCurso2.



133. En el evento *actionPerformed* del botón “Curso 1” programa lo siguiente:

```
DefaultListModel modelo = new DefaultListModel();
modelo.addElement("Juan");
modelo.addElement("María");
modelo.addElement("Luis");
lstNombres.setModel(modelo);
```

134. En el evento *actionPerformed* del botón “Curso 2” programa lo siguiente:

```
DefaultListModel modelo = new DefaultListModel();
modelo.addElement("Ana");
modelo.addElement("Marta");
modelo.addElement("Jose");
lstNombres.setModel(modelo);
```

135. Explicación de los códigos anteriores:

ffffff. Lo que hace cada botón es rellenar el cuadro de lista con una serie de nombres. En el caso del botón “Curso 1”, la lista se rellena con los nombres Juan, María y Luis, mientras que en el caso del botón “Curso 2”, la lista se rellena con los nombres Ana, Marta y Jose.

gggggg. El contenido de un cuadro de lista es lo que se denomina un “modelo”. El “modelo” es un objeto que contiene el listado de elementos de la lista.

hhhhhh. Los modelos de las listas son objetos del tipo *DefaultListModel*.

iiiiii. Lo que hace el programa es crear un “modelo”. Luego rellena el “modelo” con datos, y finalmente asocia el “modelo” al cuadro de lista. Veamos como se hace todo esto.

jjjjjj. Primero se crea el “modelo”, a través de la siguiente instrucción (será necesario añadir el *import* correspondiente, atento a la bombillita):

```
DefaultListModel modelo = new DefaultListModel();
```

kkkkkk. El “modelo” tiene un método llamado *addElement* que permite introducir datos dentro de él. Así pues usamos este método para añadir los datos al modelo.

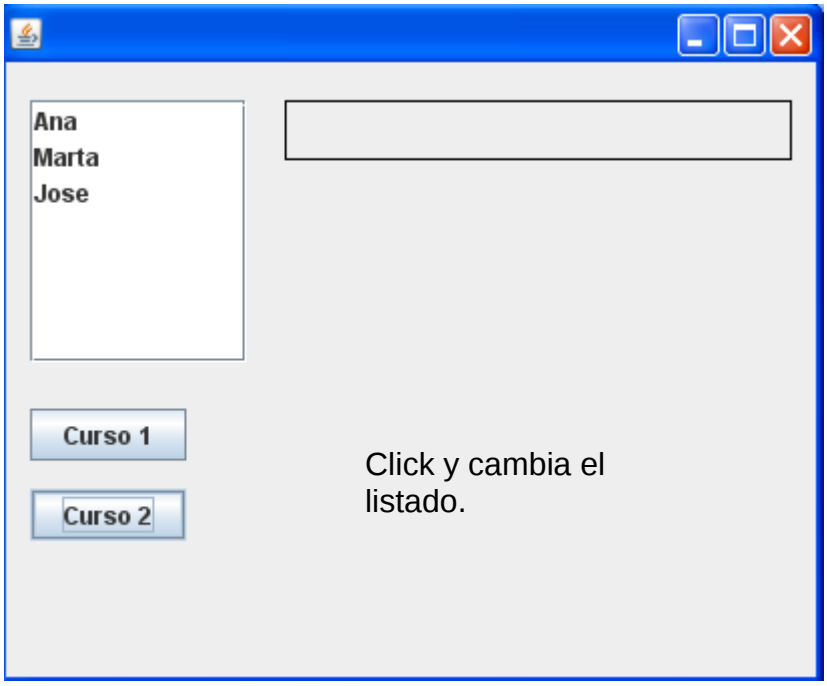
```
modelo.addElement("Ana");
modelo.addElement("Marta");
modelo.addElement("Jose");
```

llllll. Finalmente asociamos el “modelo” creado al cuadro de lista de la siguiente forma:

```
lstNombres.setModel(modelo);
```

mmmmmm. Así pues, aquí tienes una forma de cambiar el contenido de un cuadro de lista desde el propio programa.

144. Prueba a ejecutar el programa. Observa como cuando pulsas cada botón cambia el contenido de la lista:

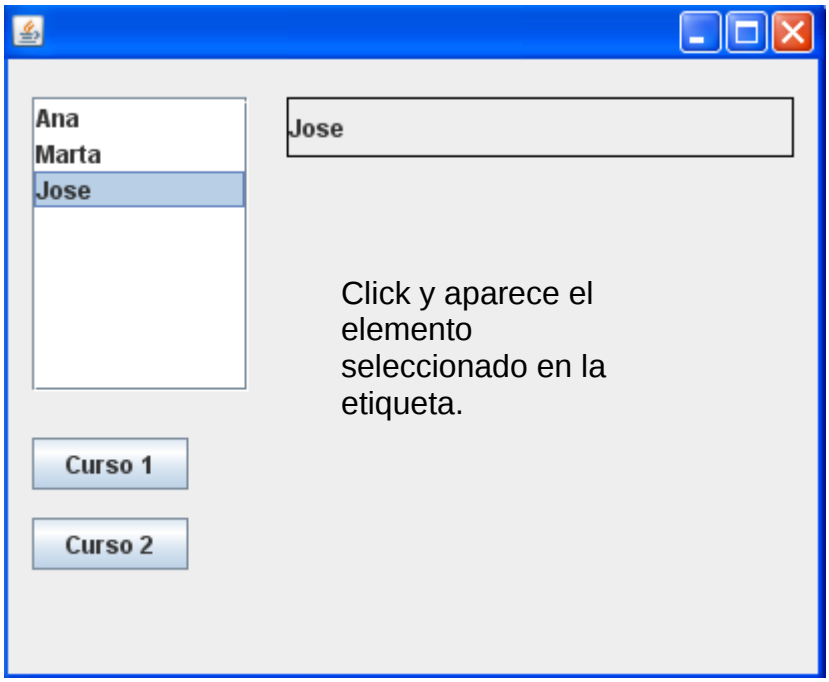


145. Ahora añade el siguiente código al evento *mouseClicked* del cuadro de lista:

```
etiResultado.setText(lstNombres.getSelectedValue().toString());
```

Esta instrucción hace que al seleccionar un elemento del cuadro de lista éste aparezca en la etiqueta etiResultado. Recuerda que el método getSelectedValue permite recoger el elemento seleccionado (hay que convertirlo a cadena con toString)

146. Ejecuta el programa:



147. Una propuesta. Añada un botón "Vaciar" llamado btnVaciar. Este botón vaciará el contenido de la lista. Para esto lo único que tiene que hacer es crear un modelo y, *sin introducir ningún valor en él*, asociarlo al cuadro de lista.

CONCLUSIÓN

Un cuadro de lista es un objeto que contiene a su vez otro objeto denominado "modelo".

El objeto "modelo" es el que realmente contiene los datos de la lista.

Cuadro de lista → Modelo → Datos

Se puede crear un "modelo" y luego introducir datos en él. Luego se puede asociar ese "modelo" a la lista. De esta manera se puede cambiar el contenido de la lista en cualquier momento.

EJERCICIO GUIADO. JAVA: MODELOS DE CUADRO DE LISTA

148. Realiza un nuevo proyecto.

149. En la ventana principal debes añadir lo siguiente:

ttttt. Un combo llamado cboNumeros.
uuuuuu. Un botón “Pares” llamado btnPares.
vvvvvv. Un botón “Impares” llamado btnImpares.
wwwwww. Una etiqueta con borde llamada etiResultado.

154. Elimina todos los elementos que contenga el combo. Recuerda, debes usar la propiedad “model” del combo para cambiar sus elementos.

155. Después de haber hecho todo esto, tu ventana debe quedar más o menos así:

156. En el evento *actionPerformed* del botón Pares, programa lo siguiente:

```
int i;

DefaultComboBoxModel modelo = new DefaultComboBoxModel();

for (i=0;i<10;i+=2) {
   modelo.addElement("Nº "+i);
}

cboNumeros.setModel(modelo);
```

157. Observa lo que hace este código:
bbbbbbb. Crea un objeto “modelo” para el combo.

Al igual que pasa con los cuadros de lista, los combos tienen un objeto “modelo” que es el que realmente contiene los datos. En el caso de los combos, para crear un objeto “modelo” se usará esta instrucción:

F d ió Códi Lib D i i

```
DefaultComboBoxModel modelo = new DefaultComboBoxModel();
```

ccccccc. A continuación, se usa el objeto “modelo” creado y se rellena de datos. Concretamente, se rellena con los números pares comprendidos entre 0 y 10.

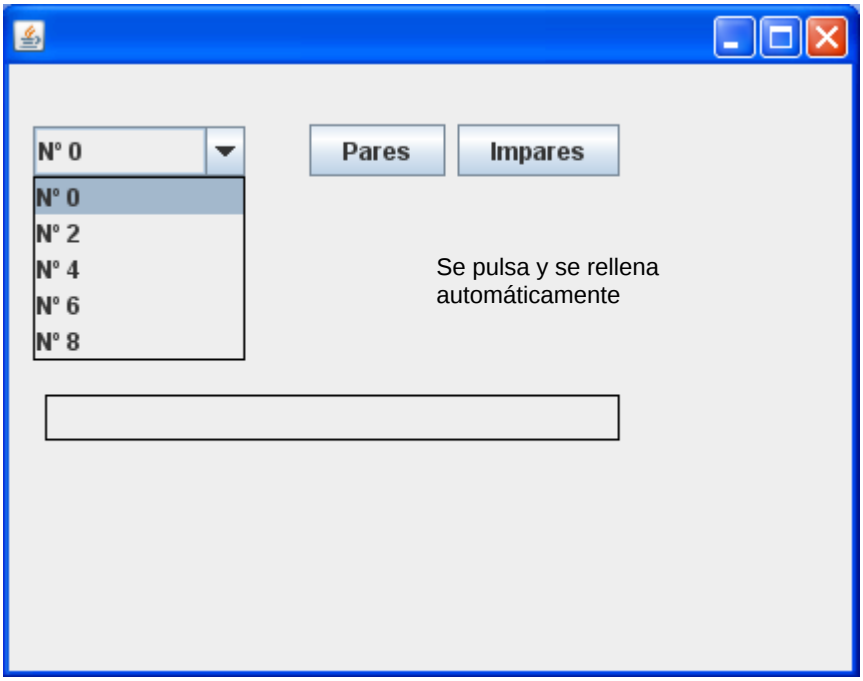
ddddddd. Observa el uso de la propiedad addElement para añadir un elemento al modelo del combo.

eeeeeee. Se ha usado un bucle for para hacer la introducción de datos en el modelo más fácil.

fffffff. Finalmente, se asocia el modelo al combo a través de la siguiente línea, con lo que el combo aparece relleno con los elementos del modelo:

```
cboNumeros.setModel(modelo);
```

163. Ejecuta el programa y observa el funcionamiento del botón Pares.



164. El botón Impares es similar. Programa su *actionPerformed* como sigue:

```
int i;
DefaultComboBoxModel modelo = new DefaultComboBoxModel();

for (i=1;i<10;i+=2) {
   modelo.addElement("Nº "+i);
}

cboNumeros.setModel(modelo);
```

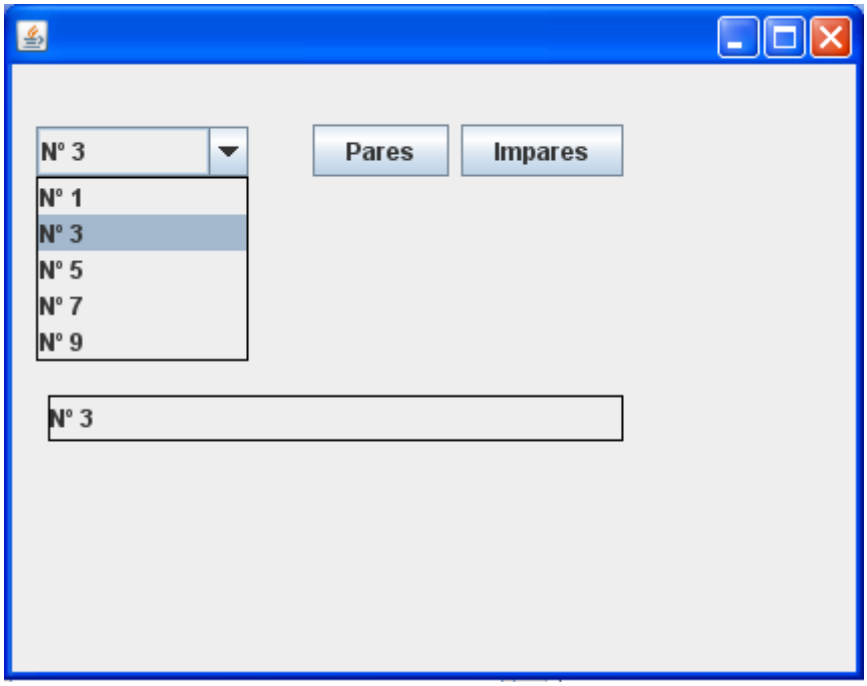
165. La única diferencia de este código es el for, que está diseñado para que se introduzcan los números impares comprendidos entre 0 y 10 dentro del modelo.

166. Finalmente se programará el *actionPerformed* del combo para que al seleccionar un elemento este aparezca en la etiqueta. Esto se hace con una simple instrucción:

```
etiResultado.setText(cboNumeros.getSelectedItem().toString());
```

Recuerda el uso de getSelectedItem() para recoger el elemento seleccionado, y el uso de toString() para convertirlo a texto.

167. Prueba el programa. Prueba los botones Pares e Impares y prueba el combo.



168. Sería interesante añadir un botón “Vaciar” llamado btnVaciar que vaciara el contenido del combo. Esto se haría simplemente creando un modelo vacío y asignarlo al combo. Se anima al alumno a que realice esta mejora.

CONCLUSIÓN

Un combo, al igual que los cuadros de lista, es un objeto que contiene a su vez otro objeto denominado “modelo”.

El objeto “modelo” es el que realmente contiene los datos del combo.

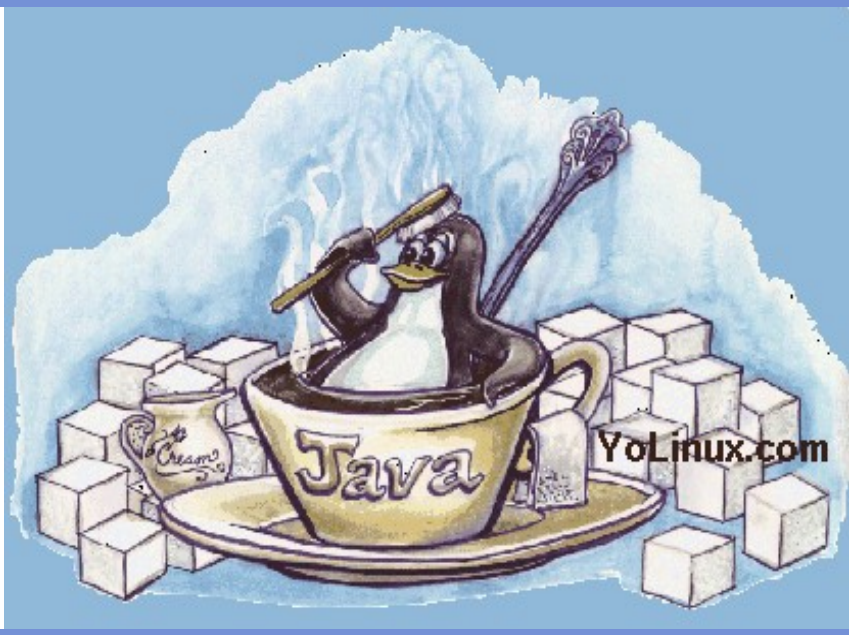
Combo → Modelo → Datos

Se puede crear un “modelo” y luego introducir datos en él. Luego se puede asociar ese “modelo” al combo. De esta manera se puede cambiar el contenido del combo en cualquier momento.

Fundación Código Libre Dominicano

Lic. Henry Terrero.
hterrero@codigolibre.org

Ing. Jose Paredes.
jparedes@codigolibre.org



Desarrollo De aplicaciones Java Con Netbeans Ejercicios

METODOLOGÍA DE LA PROGRAMACIÓN

JAVA

PRIMEROS PROGRAMAS

Ejercicio 1

Realice un programa en java con las siguientes características:

La ventana principal tendrá los siguientes elementos:

- Una etiqueta que contenga su nombre.
  Nombre de la etiqueta: etiNombre
- Una etiqueta que contenga su ciudad.
  Nombre de la etiqueta: etiCiudad
- Un botón con el texto “Ocultar Nombre”.
  Nombre del botón: btnOcultarNombre
- Un botón con el texto “Visualizar Nombre”.
  Nombre del botón: btnVisuNombre
- Un botón con el texto “Ocultar Ciudad”.
  Nombre del botón: btnOcultarNombre
- Un botón con el texto “Visualizar Ciudad”.
  Nombre del botón: btnVisuCiudad

El funcionamiento del programa será el siguiente:

- Cuando se pulse el botón *btnOcultarNombre*, se debe ocultar la etiqueta *etiNombre*.
- Cuando se pulse el botón *btnVisuNombre*, se debe visualizar la etiqueta *etiNombre*.
- Cuando se pulse el botón *btnOcultarCiudad*, se debe ocultar la etiqueta *etiCiudad*.
- Cuando se pulse el botón *btnVisuCiudad*, se debe visualizar la etiqueta *etiCiudad*.

Para hacer esto debe programar el evento *actionPerformed* de cada botón.

Usará el método *setVisible* de cada etiqueta para visualizarlas u ocultarlas.

Ejercicio 2

Creará un programa java cuya ventana principal contenga los siguientes componentes:

- Un cuadro de texto. El nombre será *txtTexto*.
- Una etiqueta vacía (sin texto dentro). El nombre será *etiTexto1*.
- Otra etiqueta vacía (sin texto dentro). El nombre será *etiTexto2*.
- Un botón con el texto “Traspasa 1”. El nombre será *btnTraspasa1*.
- Un botón con el texto “Traspasa 2”. El nombre será *btnTraspasa2*.

El programa funcionará de la siguiente forma:

- Cuando el usuario pulse el botón “Traspasa 1”, lo que se haya escrito dentro del cuadro de texto se copiará dentro de la etiqueta 1.
- Cuando el usuario pulse el botón “Traspasa 2”, lo que se haya escrito dentro del cuadro de texto se copiará dentro de la etiqueta 2.

Para hacer esto programará el evento *actionPerformed* de cada botón.

Usará el método *setText* para situar el texto en la etiqueta, y *getText* para recoger el texto del cuadro de texto.

Ejercicio 3

Se pide que realice un programa en java cuya ventana principal incluya los siguientes componentes:

- Un cuadro de texto llamado *txtTexto*.
- Una etiqueta vacía llamada *etiTexto*.
- Un botón con el texto "Vaciar" llamado *btnVaciar*.

El programa funcionará de la siguiente forma:

- Cuando el usuario escriba en el cuadro de texto *txtTexto*, lo que escriba pasará inmediatamente a la etiqueta *etiTexto*. (Tendrá que programar el evento *keyPressed* del cuadro de texto)

- Cuando el usuario pulse el botón "Vaciar" el texto contenido en el cuadro de texto y en la etiqueta se borrará. (Tendrá que programar el evento *actionPerformed*)

Tendrá que usar el método *setText* para colocar texto en la etiqueta y usar el método *getText* para recoger el texto escrito en el cuadro de texto.

Ejercicio 4

Se pide que realice un programa en java cuya ventana principal contenga los siguientes elementos:

- Un cuadro de texto llamado *txtNombre*.
- Un cuadro de texto llamado *txtCiudad*.
- Una etiqueta llamada *txtFrase*.
- Un botón con el texto "Aceptar" llamado *btnAceptar*.
- Un botón con el texto "Desactivar" llamado *btnDesactivar*.
- Un botón con el texto "Activar" llamado *btnActivar*.

El programa funcionará de la siguiente forma:

- El usuario introducirá un nombre en el cuadro de texto *txtNombre*. Por ejemplo "Juan".
- El usuario introducirá una ciudad en el cuadro de texto *txtCiudad*. Por ejemplo "Jerez".
- Cuando el usuario pulse el botón "Aceptar", entonces aparecerá un mensaje en la etiqueta llamada *txtFrase* indicando como se llama y donde vive. Por ejemplo:

  Usted se llama Juan y vive en Jerez.

- Cuando el usuario pulse el botón "Desactivar", entonces los dos cuadros de texto se desactivarán. Cuando el usuario pulse el botón "Activar", los dos cuadros de texto se volverán a activar.

Tendrá que programar el evento *actionPerformed* de los botones.

Tendrá que usar la concatenación de cadenas. Recuerde que se pueden concatenar cadenas a través del operador +.

Tendrá que usar el método *setText* para colocar el texto en la etiqueta y el método *getText* para recoger el texto de cada cuadro de texto.

Tendrá que usar el método *setEnabled* para activar o desactivar cada cuadro de texto.

Ejercicio 5

Debe realizar un programa java cuya ventana principal contenga los siguientes elementos:

- Cuatro etiquetas, conteniendo distintas palabras cada una. Puede llamarlas: *etiPal1, etiPal2, etiPal3* y *etiPal4*.

- Otra etiqueta que contenga el texto “Ocultar”. Esta etiqueta se llamará *etiOcultar* y tendrá un tamaño más grande que las otras y un color de fondo.

El programa debe funcionar de la siguiente forma:

- Cuando el puntero del ratón “sobrevuele” la etiqueta *etiOcultar*, las etiquetas con las palabras deben desaparecer.

- Cuando el puntero del ratón salga de la superficie de la etiqueta *etiOcultar*, entonces las etiquetas con las palabras volverán a aparecer.

Tendrá que programar los eventos *mouseEntered* y *mouseExited* para que el programa detecte cuando el ratón entra o sale de la etiqueta *etiOcultar*.

Debe usar el método *setVisible* de las etiquetas para ocultarlas o visualizarlas.

Ejercicio 6

Debe realizar un programa java cuya ventana principal contenga los siguientes elementos:

- Una etiqueta que contenga su nombre. Esta etiqueta se llamará *etiNombre*. Esta etiqueta debe tener un color de fondo.
- Un botón con el texto “Esquina”. Este botón se llamará *btnEsquina*.
- Un botón con el texto “Centro”. Este botón se llamará *btnCentro*.
- Un botón con el texto “Agrandar”. Este botón se llamará *btnAgrandar*.
- Un botón con el texto “Achicar”. Este botón se llamará *btnAchicar*.

El programa debe funcionar de la siguiente forma:

- Cuando el usuario pulse el botón “Esquina”, la etiqueta *etiNombre* se colocará en la esquina de la ventana.

- Cuando el usuario pulse el botón “Centro”, la etiqueta *etiNombre* se colocará en el centro de la ventana.

- Cuando el usuario pulse el botón “Agrandar”, la etiqueta *etiNombre* cambiará de tamaño, agrandándose.

- Cuando el usuario pulse el botón “Achicar”, la etiqueta *etiNombre* cambiará de tamaño, empequeñeciéndose.

- Cuando el usuario lleve el puntero sobre uno de los botones, el botón cambiará de tamaño agrandándose. Cuando el puntero salga del botón, el botón volverá a su tamaño normal.

Debe programar los eventos *actionPerformed, mouseEntered* y *mouseExited* para los botones.

Tendrá que usar el método *setLocation* para cambiar la posición de la etiqueta. Este método recibe como parámetro la posición x e y donde situará el componente.

F d ió Códi Lib D i i

Tendrá que usar el método *setSize* para cambiar el tamaño de la etiqueta y de los botones. Este método recibe como parámetro el ancho y alto del componente.

La propiedad *preferredSize* permite definir un tamaño inicial a cada componente de la ventana.

Ejercicio 7

Se pide realizar un programa java que tenga los siguientes elementos en la ventana principal:

- Una etiqueta con el texto que quiera. Esta etiqueta se llamará *etiTexto*.
- Un botón con el texto “Azul”. Este botón se llamará *btnAzul*.
- Un botón con el texto “Rojo”. Este botón se llamará *btnRojo*.
- Un botón con el texto “Verde”. Este botón se llamará *btnVerde*.
- Un botón con el texto “Fondo Azul”. Este botón se llamará *btnFondoAzul*.
- Un botón con el texto “Fondo Rojo”. Este botón se llamará *btnFondoRojo*.
- Un botón con el texto “Fondo Verde”. Este botón se llamará *btnFondoVerde*.
- Un botón con el texto “Transparente”. Este botón se llamará *btnTransparente*.
- Un botón con el texto “Opaca”. Este botón se llamará *btnOpaca*.

El programa debe funcionar de la siguiente forma:

- Cuando se pulse el botón “Azul”, el texto de la etiqueta se pondrá de color azul.
- Cuando se pulse el botón “Rojo”, el texto de la etiqueta se pondrá de color rojo.
- Cuando se pulse el botón “Verde”, el texto de la etiqueta se pondrá de color verde.
- Cuando se pulse el botón “Fondo Azul”, el fondo de la etiqueta se pondrá de color azul.
- Cuando se pulse el botón “Fondo Rojo”, el fondo de la etiqueta se pondrá de color rojo.
- Cuando se pulse el botón “Fondo Verde”, el fondo de la etiqueta se pondrá de color verde.
- Cuando se pulse el botón “Transparente”, la etiqueta dejará de ser opaca.
- Cuando se pulse el botón “Opaca”, se activará la propiedad opaca de la etiqueta.

Debe programar los eventos *actionPerformed* de cada botón.

Para cambiar el color de fondo de una etiqueta, usará el método *setBackground*.

Para cambiar el color del texto de una etiqueta, usará el método *setForeground*.

Para indicar el color azul usará Color.BLUE. Para indicar el color rojo, usará Color.RED. Y para indicar el color verde usará Color.GREEN.

Para hacer que una etiqueta sea opaca o no, usará el método *setOpaque*.

PROGRAMACIÓN

JAVA

PROGRAMAS CON CÁLCULOS

Ejercicio 1

Realice un programa para calcular el total de una venta.

El programa tendrá una ventana con al menos los siguientes elementos:

(1) Un cuadro de texto llamado txtUnidades donde el usuario introducirá las unidades vendidas.
(2) Un cuadro de texto llamado txtPrecio donde el usuario introducirá el precio unidad.
(3) Una etiqueta llamada etiTotalSinIva.
(4) Una etiqueta llamda etiIva.
(5) Una etiqueta llamada etiTotalMasIva.
(6) Un botón llamado btnCalcular.

El programa funcionará de la siguiente forma:

(7) El usuario introducirá las unidades y el precio y luego pulsará el botón Calcular.
(8) El programa entonces calculará el total de la venta, el iva y el total más iva y presentará dichos resultados en cada etiqueta.

Ejercicio 2

Realice un programa para calcular la nota final de un alumno.

El programa tendrá una ventana con al menos los siguientes elementos:

(9) Un cuadro de texto llamado txtPrimerTrimestre.
(10) Un cuadro de texto llamado txtSegundoTrimestre.
(11) Un cuadro de texto llamado txtTercerTrimestre.
(12) Una etiqueta llamada etiNotaFinal.
(13) Una etiqueta llamada etiResultado.
(14) Un botón llamado btnCalcular.

El programa funcionará de la siguiente forma:

(15) El usuario introducirá las notas de los tres trimestres en los cuadros de texto correspondientes.
(16) Cuando se pulse el botón Calcular, el programa calculará la nota media y la mostrará en la etiqueta llamada etiNotaFinal.
(17) Si la nota final es menor de 5, entonces en la etiqueta etiResultado aparecerá la palabra SUSPENSO.
(18) Si la nota final es 5 o más, entonces en la etiqueta etiResultado aparecerá la palabra APROBADO.
(19) Si la nota final fuera un suspenso, entonces las etiquetas etiNotaFinal y etiResultado deben aparecer de color rojo. En caso contrario aparecerán de color negro.

Ejercicio 3

Realizar un programa para calcular el área y el perímetro de un círculo.

El programa debe tener una ventana con al menos los siguientes elementos:

(20) Un cuadro de texto llamado txtRadio, donde el usuario introducirá el radio.
(21) Una etiqueta llamada etiArea, donde se mostrará el área del círculo.
(22) Una etiqueta llamada etiPerimetro, donde se mostrará el perímetro del círculo.

El programa funcionará de la siguiente forma:

(23) El usuario introducirá un radio dentro del cuadro de texto llamado radio. Al pulsar la tecla Enter sobre dicho cuadro de texto, el programa calculará el área y el perímetro y los mostrará en las etiquetas correspondientes.

(24) Si el usuario introduce un radio negativo, en las etiquetas debe aparecer la palabra "Error" en color rojo.

(25) Use el elemento PI de la clase Math para hacer este ejercicio.

Ejercicio 4

Realizar un programa para calcular potencias de un número.

El programa le permitirá al usuario introducir una base y un exponente. Luego el programa podrá calcular la potencia de la base elevada al exponente.

El programa deberá usar la clase Math.

El diseño de la ventana queda a su elección.

Ejercicio 5

Realizar un programa que calcule la raíz cuadrada de un número. El programa tendrá los siguientes elementos en la ventana únicamente:

(26) Un cuadro de texto llamado txtNumero.
(27) Una etiqueta llamada etiRaiz.

El programa funcionará de la siguiente forma: cuando el usuario escriba un número en el cuadro de texto txtNumero, inmediatamente aparecerá su raíz cuadrada en la etiqueta. Para ello, tendrá que programar el evento keyReleased del cuadro de texto.

Use la clase Math para realizar el cálculo de la raíz cuadrada.

Ejercicio 6

Realice un programa que contenga dos cuadros de texto: txtPalabra1, y txtPalabra2. La ventana tendrá también un botón llamado btnConcatena y una etiqueta llamada etiTexto.

El usuario introducirá las dos palabras dentro de los cuadros de texto y luego pulsará el botón Concatena. El resultado será que en la etiqueta etiTexto aparecerán las dos palabras escritas concatenadas.

Por ejemplo, si el usuario escribe en el primer cuadro de texto “Lunes” y en el segundo “Martes”, en la etiqueta aparecerá: “LunesMartes”.

Ejercicio 7

Realizar un programa que muestre 10 etiquetas llamadas respectivamente: etiUna, etiDos, etiTres, etc...

Estas etiquetas contendrán los números del 0 al 9.

Aparte, la ventana tendrá un cuadro de texto llamado txtNumero.

Se pide que cuando el usuario lleve el ratón sobre una de las etiquetas, aparezca en el cuadro de texto el número correspondiente.

Los números se van añadiendo al cuadro de texto. Por ejemplo, si el usuario lleva el puntero sobre la etiqueta Uno, luego sobre la etiqueta Tres y luego sobre la etiqueta Uno, en el cuadro de texto debería haber aparecido lo siguiente: 131.

Añada un botón llamado btnBorrar, que al ser pulsado borre el contenido del cuadro de texto.

Ejercicio 8

Se pide realizar un programa para resolver las ecuaciones de segundo grado.

Las ecuaciones de segundo grado tienen la siguiente forma:

$$ax^2+bx+c=0$$

Dados unos valores para a, b y c, se debe calcular cuanto vale x.

Una ecuación de segundo grado puede tener 0, 1 o 2 soluciones.

Para saber el número de soluciones de una ecuación de segundo grado se debe realizar el siguiente cálculo:

$$R = b^2 - 4ac$$

Si R es menor de cero, la ecuación no tiene solución.
Si R da 0, la ecuación tiene una solución.
Si R es mayor de cero, la ecuación tiene dos soluciones.

Cuando existe una solución, esta se calcula de la siguiente forma:

```
x = -b / (2a)
```

Cuando existen dos soluciones, estas se calculan de la siguiente forma:

```
x = (-b + raiz(R) ) / (2a)

x = (-b – raiz(R) ) / (2a)
```

(*raíz* es la raíz cuadrada)

Realice el programa para resolver las ecuaciones de segundo grado. El diseño de la ventana queda a su elección. Procure que el programa indique cuando hay o no soluciones, y que las muestre de forma adecuada.

PROGRAMACIÓN

JAVA

CUADROS DE MENSAJE, CONFIRMACIÓN E INTRODUCCIÓN DE DATOS

Ejercicio 1

Realice un programa para calcular la división de un número A entre un número B. El programa tendrá los siguientes elementos en la ventana:

- Dos cuadros de texto llamados txtA y txtB donde se introducirán los dos números.
- Un botón "Calcular División" llamado btnDivision.
- Una etiqueta llamada etiResultado donde aparecerá el resultado.

El programa debe funcionar de la siguiente forma:

- Cuando se pulse el botón "Calcular División" se calculará la división del número A entre el B y el resultado aparecerá en la etiqueta etiResultado.
- Si el usuario introduce un valor 0 dentro del cuadro de texto del número B, entonces el programa mostrará un mensaje de error (Use un JOptionPane.showMessage)
- Si el usuario introduce un valor menor que cero en cualquiera de los dos cuadros de texto, entonces también se mostrará un error.

Ejercicio 2

Realice un programa que permita calcular el sueldo total de un empleado. Para ello, el programa tendrá los siguientes elementos en la ventana:

- Un cuadro de texto llamado txtSueldoBase.
- Un cuadro de texto llamado txtMeses.
- Un botón llamado btnCalcular.
- Una etiqueta llamada etiResultado.

El programa funcionará de la siguiente forma:

- El usuario introducirá en el cuadro de texto txtSueldoBase la cantidad bruta que cobra el trabajador al mes.
- En el cuadro de texto txtMeses introducirá el número de meses trabajados.
- Al pulsar el botón calcular se calculará el sueldo a percibir por el empleado. Se calculará así:

  Total a percibir = (SueldoBase – 10% del SueldoBase) * Meses

- Cuando se pulse el botón calcular, antes de que aparezca el sueldo en la etiqueta de resultado, el programa debe pedirle al usuario que introduzca una contraseña. Solo si la contraseña es correcta el programa mostrará el sueldo total.
- Para introducir la contraseña use un JOptionPane.showInputDialog.
- Si el usuario introduce una contraseña incorrecta, el programa mostrará un aviso (JOptionPane.showMessage) y el resultado no se mostrará.

Ejercicico 3

Realizar un programa para calcular el índice de masa corporal de una persona.

Para ello, creará una ventana con los siguientes elementos:

- Un cuadro de texto llamado txtPeso, donde se introducirá el peso de la persona.
- Un cuadro de texto llamado txtTalla, donde se introducirá la talla.
- Una etiqueta llamada etiIMC donde aparecerá el Indice de masa corporal calculado.
- Un botón llamado btnCalcular y otro llamado btnLimpiar.

El programa funcionará de la siguiente forma:

- El usuario introducirá un peso y una talla en los cuadros de texto. Luego pulsará el botón calcular para calcular el índice de masa corporal, el cual se calcula así:

  $$IMC = Peso / Talla^2$$

- El IMC calculado aparecerá en la etiqueta, y además, aparecerá un mensaje indicando la conclusión a la que se llega, la cual puede ser una de las siguientes según el IMC:

IMC	CONCLUSIÓN
<18	Anorexia
>=18 y <20	Delgadez
>=20 y <27	Normalidad
>=27 y <30	Obesidad (grado 1)
>=30 y <35	Obesidad (grado 2)
>=35 y <40	Obesidad (grado 3)
>=40 y	Obesidad mórbida

  El mensaje puede ser algo así: “Su IMC indica que tiene anorexia”, por ejemplo. Este mensaje aparecerá en un JOptionPane.showMessage.

- Cuando pulse el botón Limpiar, se borrarán los cuadros de texto Peso y Talla. Antes de que esto ocurra, el programa debe pedir confirmación, con un cuadro de diálogo de confirmación (JOptionPane.showConfirmDialog). El cuadro de confirmación tendrá el siguiente aspecto:

  ¿Desea borrar los datos?
  SI NO

  Según lo elegido por el usuario se borrarán los cuadros de texto o no.

PROGRAMACIÓN

JAVA

CADENAS

Ejercicio 1

Realice un programa cuya ventana tenga los siguientes elementos:

(1) Un cuadro de texto llamado txtFrase.
(2) Varias etiquetas. (Llámelas como quiera)
(3) Un botón “Analizar” llamado btnAnalizar.

El programa funcionará de la siguiente forma:

(4) El usuario introducirá una frase en el cuadro de texto, y luego pulsará el botón Analizar.

(5) Al pulsar Analizar, el programa mostrará la siguiente información en las etiquetas:

f. La frase en mayúsculas.
g. La frase en minúsculas.
h. Número de caracteres de la frase.
i. Número de caracteres de la frase sin contar los espacios.

(10) Si el usuario pulsa Analizar cuando no hay ninguna frase introducida en el cuadro de texto, el programa debe mostrar un error emergente (JOptionPane)

Ejercicio 2

Realice un programa cuya ventana tenga los siguientes elementos:

- Un cuadro de texto llamado txtFrase y otro llamado txtSubcadena.
- Varias etiquetas.
- Un botón “Analizar” llamado btnAnalizar.

El programa funcionará de la siguiente forma:

- El usuario introducirá una frase en el cuadro de texto txtFrase, y luego introducirá una palabra en el cuadro de texto txtSubcadena, y luego pulsará el botón Analizar.

- Al pulsar el botón, el programa debe mostrar la siguiente información:

  - La posición en la que se encuentra la primera aparición de la palabra en la frase.
  - La posición en la que se encuentra la última aparición de la palabra en la frase.
  - Mostrará el texto que hay en la frase antes de la primera palabra.
  - Mostrará el texto que hay en la frase después de la última palabra.

- Por ejemplo, si la frase fuera:

  Un globo, dos globos, tres globos. La luna es un globo que se me escapó.

  Y la palabra fuera *globo*, entonces la información a mostrar sería:
  Posición inicial: 3
  Posición final: 49
  Texto anterior: Un
  Texto posterior: que se me escapó

- Si la palabra no se encuentra en la frase, el programa mostrará un error emergente y no se presentará nada en las etiquetas.

Ejercicio 3

Realice un programa que tenga los siguientes elementos:

- Un cuadro de texto llamado txtFrase.
- Un cuadro de texto llamado txtPalabra1.
- Un cuadro de texto llamado txtPalabra2.
- Un cuadro de texto llamado txtPalabra3.
- Un cuadro de texto llamado txtPalabra4.
- Varias etiquetas.
- Un botón “Analizar” llamado btnAnalizar.

El programa funcionará de la siguiente forma:

- El usuario introducirá una frase en el cuadro de texto txtFrase, y tres palabras en los cuadros de texto de las palabras.

- Al pulsar el botón Analizar, el programa debe indicar la siguiente información en las etiquetas:

    - Indicará si la frase es igual a la palabra 1.
    - Indicará si la frase empieza por la palabra 2.
    - Indicará si la frase termina por la palabra 3.
    - Indicará si la palabra 4 está contenida en la frase, y en el caso de que esté contenida, se indicará la posición inicial en la que se encuentra.

PROGRAMACIÓN

JAVA

CADENAS (CONTINUACIÓN)

Ejercicio 1

Realizar un programa cuya ventana tenga los siguientes elementos:

(1) Un cuadro de texto llamado txtFrase.
(2) Varias etiquetas.
(3) Un botón “Analizar” llamado btnAnalizar.

El programa debe contar cuantas vocales tiene la frase. El funcionamiento será el siguiente:

(4) El usuario escribirá una frase en el cuadro de texto txtFrase. Luego se pulsará el botón Analizar.

(5) El programa mostrará en las etiquetas el número de a, de e, de i de o y de u que se encuentran en la frase.

(6) Tenga en cuenta que puede haber vocales en mayúsculas y en minúsculas.

(7) Si el usuario no introduce nada en el cuadro de texto txtFrase, entonces el programa debería mostrar un error.

Ejercicio 2

Realizar un programa cuya ventana tenga los siguientes elementos:

(8) Un cuadro de texto llamado txtDNI.
(9) Una etiqueta llamada etiDNI.
(10) Un botón “Preparar DNI” llamado btnPrepararDNI.

El programa funcionará de la siguiente forma:

(11) El usuario introducirá un DNI en el cuadro de texto llamado txtDNI y luego pulsará el botón Preparar DNI. El resultado será que el DNI introducido aparecerá “preparado” en la etiqueta etiDNI.

(12) A continuación se explica como preparar el DNI:

El usuario puede introducir un DNI con uno de estos formatos:

31.543.234-A
31.543.234 A
31.543.234A
31 543 234 A
etc.

Sin embargo, cuando el usuario pulse el botón Preparar DNI, en la etiqueta etiDNI debe aparecer el DNI con el siguiente formato:

31543234A

Es decir, sin ningún espacio y sin puntos ni guiones.

(13) Si el usuario no introduce nada en el cuadro de texto del DNI y pulsa el botón, entonces debe aparecer un error emergente (JOptionPane).

Ejercicio 3

Se pide hacer un programa que le permita al usuario introducir una palabra en latín de la primera declinación, y a continuación generar sus "casos" en plural y singular.

Las palabras de la primera declinación en latín son sustantivos femeninos (la mayoría), que terminan en a, como por ejemplo: ROSA o ANIMA.

Estas palabras tienen las siguientes variantes o "casos":

Por ejemplo, para la palabra ROSA, sus casos son los siguientes:

CASO	**SINGULAR**	**PLURAL**
NOMINATIVO	Rosa	Rosae
VOCATIVO	Rosa	Rosae
ACUSATIVO	Rosam	Rosas
GENITIVO	Rosae	Rosarum
DATIVO	Rosae	Rosis
ABLATIVO	Rosa	Rosis

Por ejemplo, para la palabra ANIMA, sus casos son los siguientes:

CASO	**SINGULAR**	**PLURAL**
NOMINATIVO	Anima	Animae
VOCATIVO	Anima	Animae
ACUSATIVO	Animam	Animas
GENITIVO	Animae	Animarum
DATIVO	Animae	Animis
ABLATIVO	Anima	Animis

Debes observar que algunos casos son exactamente iguales a la palabra inicial, como por ejemplo el Nominativo Singular.

Otros casos, en cambio, se construyen añadiendo algunas letras al final de la palabra inicial. Por ejemplo, el Acusativo singular se construye añadiendo una "m" a la palabra inicial.

Para construir el Dativo y Ablativo plural, es necesario concatenar la palabra inicial (sin la a final) con "is". Por ejemplo, en el caso de la palabra Rosa, se concatenaría: Ros + is.

TENIENDO EN CUENTA LO ANTERIOR, SE PIDE REALIZAR EL SIGUIENTE PROGRAMA:

Realice un programa cuya ventana tenga los siguientes elementos:

(14) Un cuadro de texto txtPalabra.
(15) Doce etiquetas al menos correspondientes a los 6 casos en singular y plural.
(16) Un botón "Declinar" llamado btnDeclinar.

El programa funcionará de la siguiente forma:

(17) El usuario introducirá una palabra en latín de la primera declinación en el cuadro de texto, y luego pulsará el botón "Declinar". Entonces en las etiquetas aparecerán los casos declinados de la palabra.

(18) Si el usuario introduce una palabra que no termine en “a”, entonces el programa dará un error, ya que dicha palabra no es de la primera declinación.

(19) Tenga en cuenta que la palabra introducida puede estar en mayúsculas o minúsculas.

PROGRAMACIÓN

JAVA

EXCEPCIONES

Ejercicio 1

Realizar un programa que le permita al usuario introducir una frase, una posición inicial y una posición final. (Tres cuadros de texto llamados *txtFrase, txtPosIni* y *txtPosFin*.

El programa debe mostrar la subcadena contenida entre la posición inicial y la posición final (use una etiqueta *etiSubcadena*)

Esto sucederá cuando se pulse un botón “Analizar” llamado *btnAnalizar*.

El código de este programa puede generar errores de ejecución (excepciones), en el caso de que el usuario no introduzca nada en el cuadro de texto de la frase, o en el caso de que el usuario introduzca un valor incorrecto de los cuadros de texto de las posiciones.

El programa debe ser capaz de capturar las excepciones producidas y mostrar un mensaje de error.

Ejercicio 2

Realizar un programa que pida las coordenadas de una recta, es decir, las coordenadas del punto p1 (x1, y1) y las coordenadas del punto p2 (x2,y2).

Así pues el programa tendrá cuatro cuadros de texto: *txtX1*, *txtY1*, *txtX2* y *txtY2* donde el usuario introducirá las coordenadas.

El programa debe calcular la distancia de la recta, usando la siguiente fórmula:

$$Raíz((x2-x1)^2 + (y2-y1)^2)$$

Añada un botón “Calcular” llamado *btnCalcular* que realice el cálculo. El código que introduzca en este botón debe ser capaz de capturar cualquier tipo de excepción producida. Interesa que aparezca un mensaje indicando el tipo de error producido, y el tipo de excepción.

Ejercicio 3

Realice un programa que le pida dos números al usuario. Ambos números deben estar comprendidos entre 0 y 100.

El programa debe calcular la división y el resto del mayor entre el menor.

El programa debe capturar todo tipo de excepciones producidas en el código.

Además, debe hacer que si el usuario introduce un número no comprendido entre 0 y 100, el código genere una excepción propia

F d ió Códi Lib D i i

PROGRAMACIÓN

JAVA

LA CLASE JFRAME

Ejercicio 1

Realizar un programa cuya ventana contanga los siguientes elementos:

(11) Un cuadro de texto llamado txtTitulo.
(12) Un botón “Cambiar Título” llamado btnCambiarTitulo.
(13) Un botón “Maximizar” llamado btnMaximizar.
(14) Un botón “Minimizar” llamado btnMinimizar.
(15) Un botón “Restaurar” llamado btnRestaurar.
(16) Un cuadro de texto llamado txtPosicionX.
(17) Un cuadro de texto llamado txtPosicionY.
(18) Un botón “Mover” llamado btnMover.
(19) Un botón “Centrar” llamado btnCentrar.
(20) Un botón “Estilo Java” llamado btnEstiloJava.
(21) Un botón “Estilo Windows” llamado btnEstiloWindows.
(22) Un botón “Rojo” llamado btnRojo.
(23) Un botón “Verde” llamado btnVerde.
(24) Un botón “Azul” llamado btnAzul.

El funcionamiento del programa será el siguiente:

(25) Si el usuario pulsa el botón “Cambiar Titulo”, el título de la ventana cambiará, colocándose como título lo que el usuario haya escrito en el cuadro de texto txtTitulo.

(26) Si el usuario pulsa el botón “Maximizar”, la ventana se maximizará.

(27) Si el usuario pulsa el botón “Minimizar”, la ventana se minimizará.

(28) Si el usuario pulsa el botón “Restaurar”, la ventana se restaurará.

(29) Si el usuario pulsa el botón “Mover”, la ventana se colocará en la posición de la pantalla que venga indicada por los valores x, y, introducidos respectivamente en los cuadros de texto txtPosicionX y txtPosicionY.

(30) Si el usuario pulsa el botón “Centrar”, la ventana se colocará en el centro de la pantalla.

(31) Si el usuario pulsa el botón “Estilo Java”, entonces toda la ventana cambiará a la visualización Java.

(32) Si el usuario pulsa el botón “Estilo Windows”, entonces toda la ventana cambiará a la visualización Windows.

(33) Si el usuario pulsa uno de los botones de color, entonces el fondo de la ventana se cambiará al color indicado.

Además, el programa contará con las siguientes características:

(34) Al iniciarse el programa, este debe pedir una contraseña al usuario. Si el usuario introduce la contraseña correcta, entonces el usuario entrará en el programa. En caso contrario el programa se cerrará.

(35) Al finalizar el programa, este debe pedir una confirmación al usuario:

¿Desea salir del programa?

Si el usuario responde Sí, entonces el programa finalizará. En caso contrario, el programa seguirá funcionando.

F d ió Códi Lib D i i

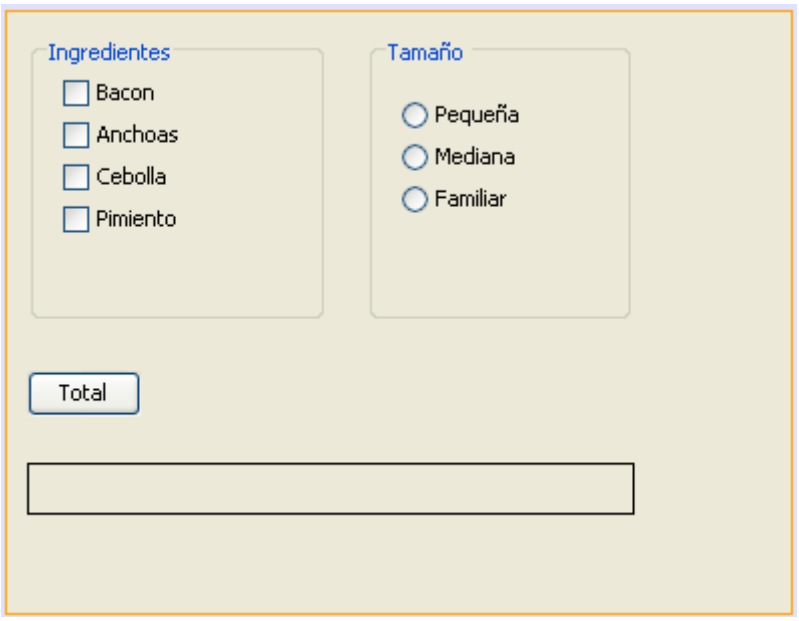
PROGRAMACIÓN

JAVA

CUADROS DE VERIFICACIÓN, BOTONES DE OPCIÓN

Ejercicio 1

Se pide realizar un programa que tenga los siguientes elementos en la ventana principal:



(1) Un panel con el título “Ingredientes”. No hace falta que le de un nombre.
(2) Un panel con el título “Tamaño”. No hace falta que le de un nombre.
(3) Cuatro JCheckBox con los textos:
    d. “Bacon” – nombre: chkBacon
    e. “Anchoas” – nombre: chkAnchoas
    f. “Cebolla” – nombre: chkCebolla
    g. “Pimiento” – nombre: chkPimiento
(8) Tres JRadioButtons con los textos:
    i. “Pequeña” – nombre: optPequenia
    j. “Mediana” – nombre: optMediana
    k. “Familiar” – nombre: optFamiliar

(12) Un botón “Total” llamado btnTotal.
(13) Una etiqueta con borde llamada etiResultado.

El programa funcionará de la siguiente forma:

(14) El usuario elegirá un tamaño para la pizza que quiere pedir. Este tamaño puede ser uno solo de los siguientes: pequeña, mediana o familiar.
(15) El usuario elegirá también los ingredientes que desee. Puede seleccionar uno o varios ingredientes.
(16) Al pulsar el botón Total, el programa calculará y mostrará en la etiqueta etiResultado el precio de la pizza, teniendo en cuenta lo siguiente:

Una pizza pequeña cuesta 7 euros.

F d ió Códi Lib D i i

Una pizza mediana cuesta 9 euros.
Una pizza familiar cuesta 11 euros.

Si se le añade como ingrediente Bacon, hay que aumentar el precio de la pizza en 1,50 euros.
Si se le añade como ingrediente Anchoas, hay que aumentar el precio de la pizza en 1,80 euros.
Si se le añade como ingrediente Cebolla, hay que aumentar el precio de la pizza en 1,00 euros.
Si se le añade como ingrediente Pimiento, hay que aumentar el precio de la pizza en 1,20 euros.

Tenga en cuenta esto otro:

(17) Al ejecutar el programa, debe estar seleccionada la opción pizza familiar por defecto, y no debe estar seleccionada ninguno de los ingredientes.

(18) Al iniciarse el programa, debe aparecer un cuadro indicando el nombre del programa. Por ejemplo: PIZZERÍA JAVA, y el nombre del programador.

(19) Al intentar cerrar el programa, este debe pedir confirmación para salir. Solo si el usuario acepta salir del programa este se cerrará.

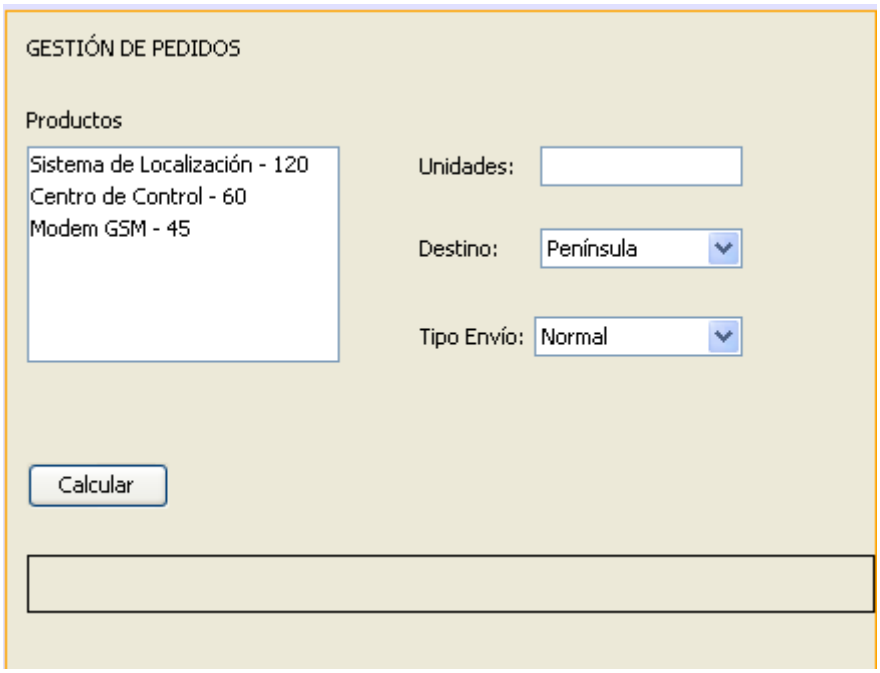
PROGRAMACIÓN

JAVA

LISTAS Y COMBOS

Ejercicio 1

Se pide realizar un programa que tenga el siguiente aspecto:



Esta ventana contiene lo siguiente:

(20) Un cuadro de lista con los siguientes productos (y sus precios):

Sistema de Localización – 120
Centro de Control – 60
Modem GSM – 45

Este cuadro de lista se llamará lstProductos.

(21) Un cuadro de texto donde se introducirán las unidades que se envían, llamado txtUnidades.

(22) Un combo llamado cboDestino donde tendremos los siguientes elementos:

Península
Canarias
Extranjero

(23) Un combo llamado cboTipoEnvio donde tendremos los siguientes elementos:

Normal
Urgente

F d ió Códi Lib D i i

(24) Un botón “Calcular” llamado btnCalcular.
(25) Una etiqueta con un borde llamada etiResultado.
(26) Otras etiquetas informativas.

El programa funcionará de la siguiente forma:

(27) El usuario marcará un producto de los tres que aparecen en la lista.
(28) El usuario indicará el número de unidades que se enviarán del producto.
(29) El usuario indicará el tipo de destino.
(30) El usuario indicará el tipo de envío.
(31) Al pulsar Calcular, el programa mostrará en la etiqueta etiResultado el total del envío, teniendo en cuenta lo siguiente:

El total del envío se calcula así:

Total = (Unidades * Precio del producto) + coste destino + coste tipo

El Coste de Destino puede ser uno de los siguientes:

Península – 20
Canarias – 25
Extranjero – 30

El Coste del Tipo de Envío puede ser uno de los siguientes:

Normal – 3
Urgente – 10

Tenga en cuenta esto otro:

(32) Si se pulsa calcular cuando no hay seleccionado ningún producto, el programa mostrará un mensaje de error emergente indicándolo.

(33) Si las unidades son incorrectas, debería mostrarse también un error.

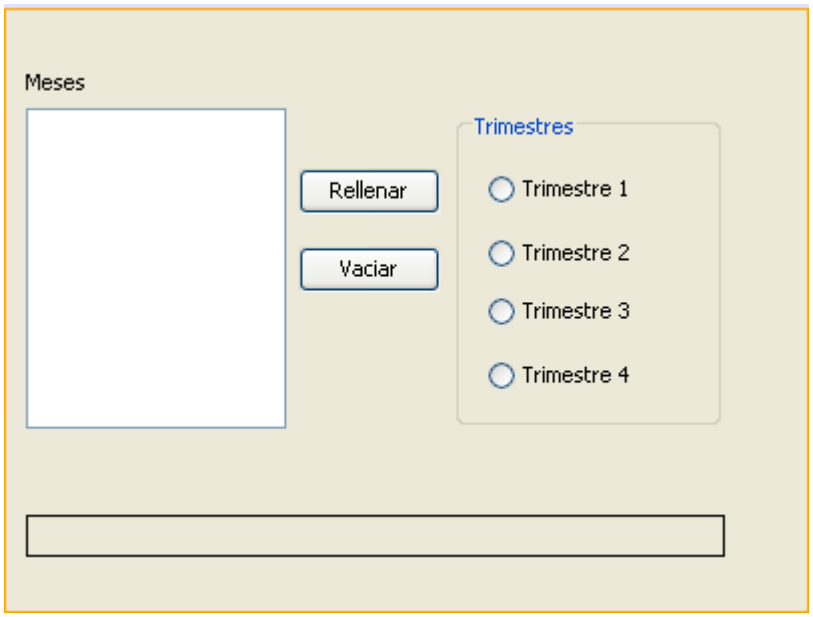
PROGRAMACIÓN

JAVA

LISTAS, COMBOS, MODELOS

Ejercicio 1

Se pide realizar un programa que tenga el siguiente aspecto:



Esta ventana contiene lo siguiente:

(28) Un cuadro de lista llamado lstMeses.
(29) Varios botones de opción con los siguientes nombres:
- dd. Un botón “Trimestre 1” llamado optTri1.
- ee. Un botón “Trimestre 2” llamado optTri2.
- ff. Un botón “Trimestre 3” llamado optTri3.
- gg. Un botón “Trimestre 4” llamado optTri4.
- hh. Todos estos botones deben estar agrupados a través de un objeto ButtonGroup al que llamaremos grupoTrimestres.
- ii. Interesará que estos botones estén dentro de un panel.

(36) Un botón “Rellenar” llamado btnRellenar.
(37) Un botón “Vaciar” llamado btnVaciar.
(38) Una etiqueta etiMes con un borde.

El programa funcionará de la siguiente forma:

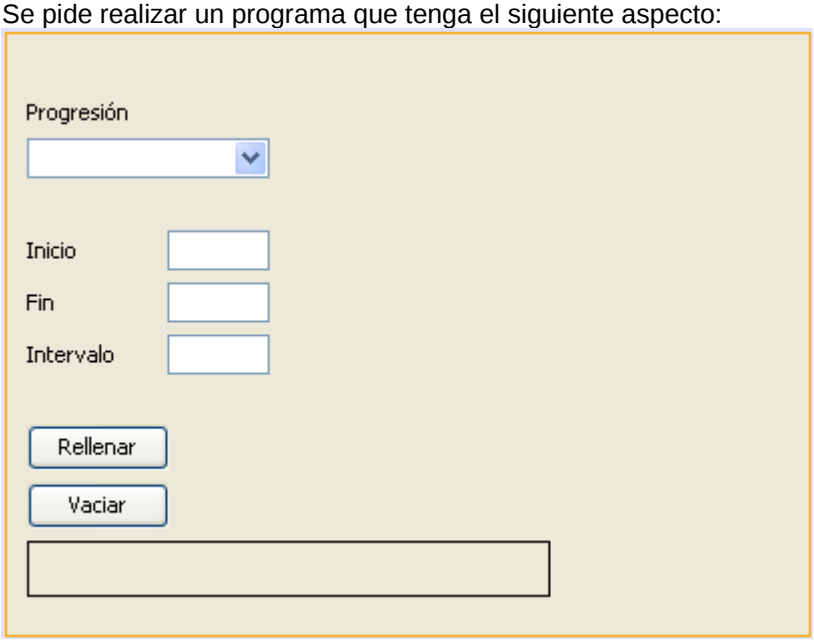
(39) El usuario elegirá una de las opciones: Trimestre 1, Trimestre 2, Trimestre 3, Trimestre 4.
(40) A continuación el usuario pulsará el botón Rellenar, y entonces el cuadro de lista se rellenará con los meses correspondientes al trimestre elegido.
(41) Por ejemplo, si el usuario elige el Trimestre 2 y pulsa el botón, entonces el cuadro de lista contendrá: Abril, Mayo, Junio.
(42) Cuando el usuario pulse el botón Vaciar, el cuadro de lista se vaciará.

F d ió Códi Lib D i i

(43) Cuando el usuario haga clic sobre un elemento de la lista, este debe aparecer en la etiqueta etiMes.

Ejercicio 2

Se pide realizar un programa que tenga el siguiente aspecto:



Esta ventana contiene lo siguiente:

(44) Un combo llamado cboProgresion.
(45) Un cuadro de texto llamado txtInicio.
(46) Un cuadro de texto llamado txtFin.
(47) Un cuadro de texto llamado txtIntervalo.
(48) Un botón “Rellenar” llamado btnRellenar.
(49) Una etiqueta llamada etiResultado.
(50) Un botón “Vaciar” llamado btnVaciar.

El programa funcionará de la siguiente forma:

(51) El usuario introducirá un número en txtInicio.
(52) Luego introducirá otro número en txtFin.
(53) También introducirá un número en txtIntervalo.
(54) Al pulsar el botón Rellenar, el combo se rellenará con el listado de números comprendidos entre el número inicial y el número final con intervalo el indicado.
(55) Un ejemplo:
    ddd. El usuario introduce en txtInicio un 2
    eee. El usuario introduce en txtFin un 12
    fff. El usuario introduce en txtIntervalo un 3
    ggg. Al pulsar el botón Rellenar, el combo debe rellenarse con los siguientes números: 2, 5, 8, 11
    hhh. Observa, del 2 al 12 saltando de 3 en 3.

(61) Al elegir cualquiera de los números en el combo, este debe mostrarse en la etiqueta etiResultado.
(62) Al pulsarse el botón “Vaciar” el combo se vacia.

A tener en cuenta.

(63) Si el usuario introduce algo que no sea un número en los cuadros de texto, el programa debería mostrar un error.

(64) El número inicial debe ser menor que el número final. En caso contrario el programa debe mostrar un error.

F d ió Códi Lib D i i

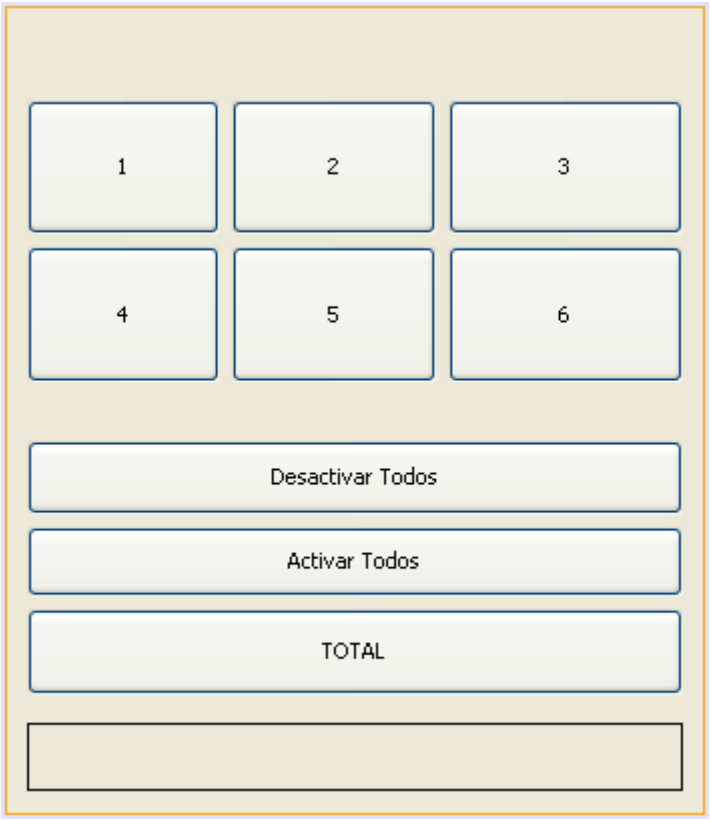
PROGRAMACIÓN

JAVA

JTOGGLEBUTTONS

Ejercicio 1

Realice un programa cuya ventana tenga el siguiente aspecto:



- Los botones numerados desde el 1 al 6 son JToggleButtons. Sus nombres son respectivamente: botonUno, botonDos, etc…
- Tres botones normales:
  - Desactivar Todos. Nombre: btnDesactivarTodos
  - Activar Todos. Nombre: btnActivarTodos
  - Total. Nombre: btnTotal
- Y una etiqueta con borde llamada etiResultado.

El programa funcionará de la siguiente forma:

- Cuando el usuario pulse Total, en la etiqueta debe aparecer la suma de los números de los botones seleccionados. Por ejemplo, si están seleccionados el 2 y el 4, aparecerá un 6.
- Si se pulsa el botón Desactivar Todos, todos los botones activados se desactivarán.
- Si se pulsa el botón Activar Todos, se activarán todos los botones.

F d ió Códi Lib D i i

PROGRAMACIÓN

JAVA

JSLIDER

Ejercicio 1

Una empresa de productos químicos necesita calcular la cantidad de agua y sales que necesita mezclar para fabricar un detergente. Para ello hay que tener en cuenta tres factores:

- Litros que se quieren fabricar. (Es un valor entre 1 y 100)
- Calidad del detergente. (Es un valor entre 0 y 10)
- Densidad deseada. (Es un valor entre 50 y 200)

La cantidad de agua necesaria viene dada por la siguiente fórmula:

Litros de agua = 2 * Litros a Fabricar + Calidad / Densidad

La cantidad de sales necesaria viene dada por la siguiente fórmula:

Sales = (Calidad * Litros a Fabricar) / (100 * Densidad)

Se pide realizar un programa cuya ventana contenga lo siguiente:

- Un JSlider llamado *deslizadorLitros* que represente los litros a fabricar. Sus valores estarán comprendidos entre 1 y 100. El valor inicial del deslizador será 50.

- Un JSlider llamado *deslizadorCalidad* que represente la calidad del detergente. Sus valores estarán comprendidos entre 0 y 10. El valor inicial será 5.

- Un JSlider llamado *deslizadorDensidad* que represente la densidad deseada. Sus valores estarán comprendidos entre 50 y 200. El valor inicial será 100.

- Una etiqueta con borde llamada *etiLitros* donde aparecerá la cantidad de litros elegida en el deslizador de litros.

- Una etiqueta con borde llamada *etiCalidad* donde aparecerá la cantidad de calidad elegida en el deslizador de calidad.

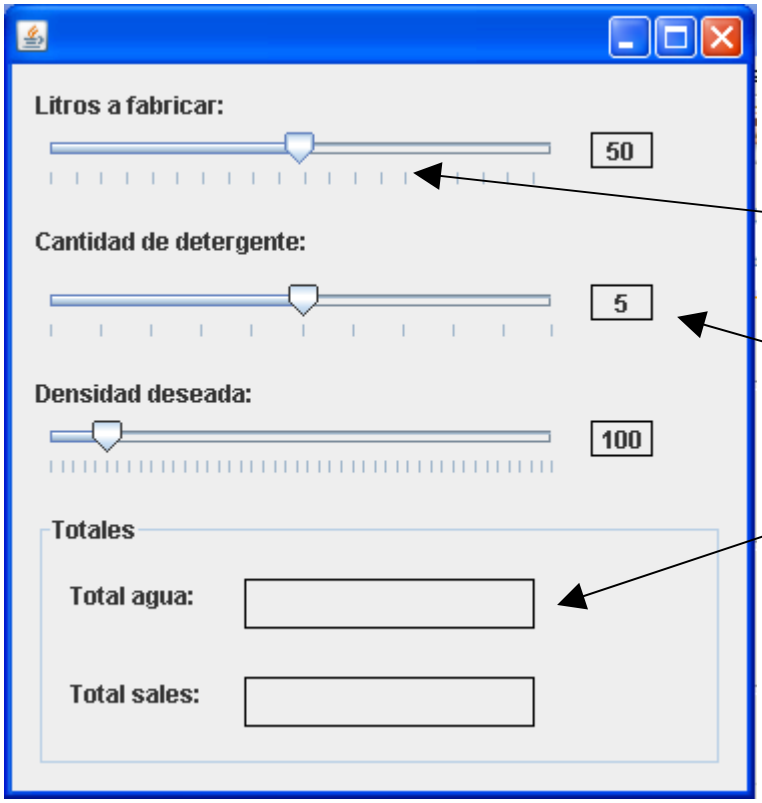
- Una etiqueta con borde llamada *etiDensidad* donde aparecerá la cantidad de densidad elegida en el deslizador de calidad.

- Una etiqueta con borde *etiLitrosAgua*, que contenga el total de litros de agua calculados.

- Una etiqueta con borde *etiSales*, que contenga el total de sales calculados.

- Varias etiquetas informativas / algún panel.

La ventana puede tener el siguiente aspecto:



El usuario arrastra los deslizadores...

...Los valores de las etiquetas cambian...

...Se calcula automáticamente los totales...

El programa funcionará de la siguiente forma:

- El usuario arrastrará los deslizadores y automáticamente aparecerá en las etiquetas los valores de cada deslizador y la cantidad total de aguas y sales calculada.

F d ió Códi Lib D i i

PROGRAMACIÓN

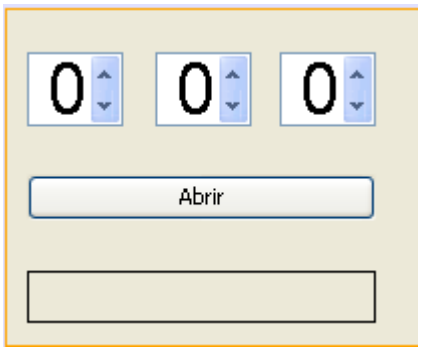
JAVA

JSPINNER

Ejercicio 1

Se propone hacer un juego sencillo, que simule la apertura de una caja a través de una combinación.

Para ello, debes crear una ventana como la que sigue:



Esta ventana contiene los siguientes elementos:

(1) Tres JSpinner a los que se les llamará: spiCentenas, spiDecenas y spiUnidades. Estos JSpinner solo admitirán los valores entre 0 y 9.

(2) Un botón btnAbrir.

(3) Una etiqueta con borde llamada etiResultado.

Funcionamiento del programa:

(4) La clave de apertura será la siguiente: 246 (Esto no lo sabe el usuario)

(5) El usuario modificará los valores de los JSpinner y luego pulsará el botón Abrir.

(6) Si los valores de los JSpinner coinciden con la clave, 246, entonces en la etiqueta debe aparecer el mensaje “Caja Abierta”.

(7) Si los valores de los JSpinner forman un número menor que 246, entonces en la etiqueta debe aparecer el mensaje: “El número secreto es mayor”.

(8) Si los valores de los JSpinner forman un número mayor que 246, entonces en la etiqueta debe aparecer el mensaje: “El número secreto es menor”.

Nota:

(9) Ten en cuenta que el valor obtenido de un JSpinner no es un número. Si quieres obtener el número entero del JSpinner tienes que usar un código como este:

```
int x;
```

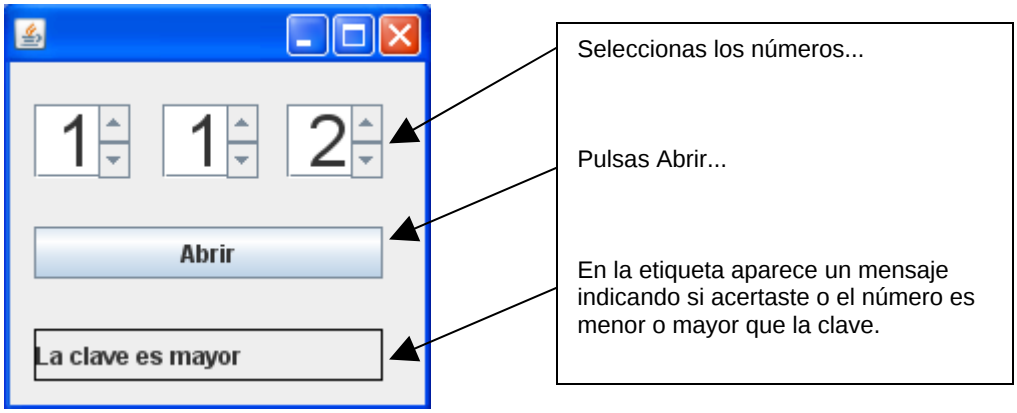
F d ió Códi Lib D i i

```
x = Integer.parseInt(spiValor.getValue().toString()) ;
```

(10) Sea el número A las centenas, el B las decenas y el C las unidades. Para calcular el número correspondiente hay que hacer el siguiente cálculo:

N = A * 100 + B * 10 + C

Ejemplo de funcionamiento:



F d ió Códi Lib D i i

PROGRAMACIÓN

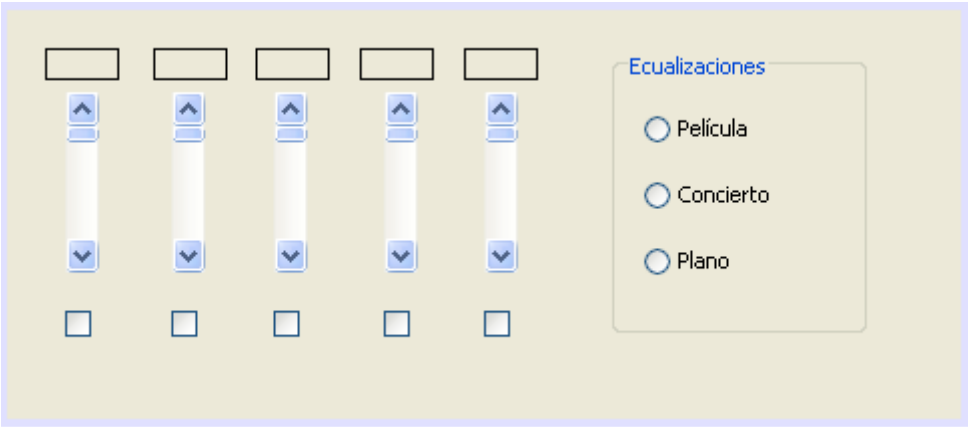
JAVA

JSROLLBAR

Ejercicio 1

Imagine que le encargan realizar un programa que contenga un ecualizador. Para permitir al usuario el control de las distintas frecuencias se usarán varias barras de desplazamiento.

La ventana del programa que debe construir debe ser similar a la siguiente:



Esta ventana contiene lo siguiente:

(1) Cinco etiquetas con borde que puede llamar etiFrecuencia1, etiFrecuencia2, etc…

(2) Cinco barras de desplazamiento que tienen de nombre desFrecuencia1, desFrecuencia2, etc…

(3) Cinco cuadros de verificación que puede llamar chkFrecuencia1, chkFrecuencia2, etc…

(4) Un panel con título y dentro de él tres botones de opción a los que puede llamar optPelícula, optConcierto y optPlano. (Para estos tres botones de radio necesitará un objeto del tipo ButtonGroup al que puede llamar grupoEcualización)

El programa funcionará de la siguiente forma:

(5) Las cinco barras de desplazamiento deben tener valores comprendidos entre un mínimo de 0 y un máximo de 10. El incremento unitario debe ser de 1 y el incremento en bloque de 2.

(6) Cada vez que se mueva una barra, en su etiqueta correspondiente aparecerá el valor de dicha barra (un valor que estará entre 0 y 10) Comprueba que cuando el usuario active al máximo una barra en la etiqueta aparezca un 10.

(7) Las casillas de verificación indican el bloqueo o no de cada barra. Si el usuario activa una casilla de verificación, entonces su barra correspondiente quedará bloqueada de forma que no pueda ser modificada. Cuando se vuelva a desactivar la casilla la barra se volverá a activar (Use el método *setEnabled*)

(8) Las opciones de ecualización predefinidas permitirán colocar las barras de desplazamiento en unos valores predefinidos. Concretamente:

F d ió Códi Lib D i i

a. Si el usuario activa la opción *Película*, las barras quedarán con los siguientes valores respectivamente: 2, 5, 8, 5, 2
b. Si el usuario activa la opción *Concierto*, las barras quedarán con los siguientes valores respectivamente: 9, 5, 1, 5, 9

c. Si el usuario activa la opción *Plano*, todas las barras se colocarán a 5.

(9) Cuando el programa se inicie, todas las barras deben estar a 5 y la opción *Plano* debe estar activada.

PROGRAMACIÓN

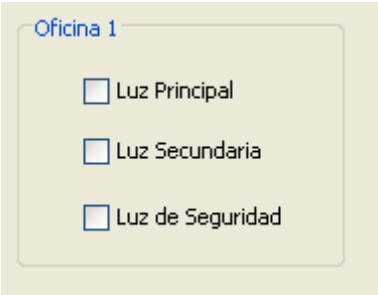
JAVA

BARRA DE MENÚS

Ejercicio 1

Le encargan un programa que controle las luces de las oficinas de un edificio. Concretamente, se tienen que controlar tres oficinas, y cada una de ellas tiene dos luces principales y una de seguridad.

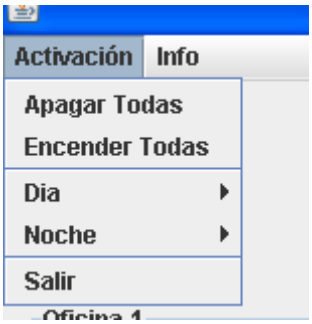
La ventana principal del programa debe mostrar tres paneles como el que sigue:



Se supondrá que las distintas luces de cada oficina se pueden encender o apagar activando o desactivando los cuadros de verificación.

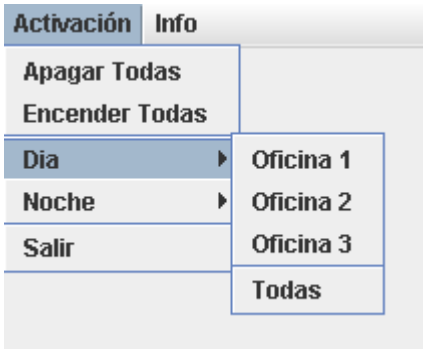
El programa además debe de contar con un menú con una opción llamada “Activación” y otra llamada “Info”

La opción Activación contendrá las siguientes opciones:



(20)La opción “Apagar Todas” desactivará todos los cuadros de verificación de las luces.
(21)La opción “Encender Todas” activará todos los cuadros de verificación de las luces.

(22)La opción “Dia” contiene las siguientes subopciones:



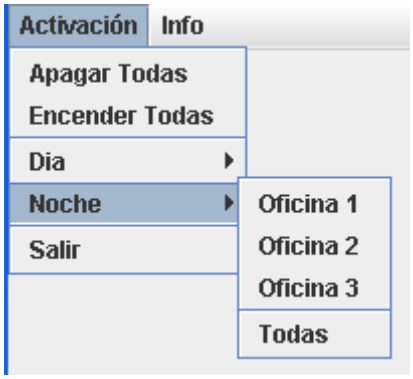
(23) Si se activa la opción “Dia – Oficina 1”, entonces se encenderán (activarán) las luces principal y secundaria de la oficina 1, y se apagará (desactivará) la luz de seguridad de dicha oficina.

(24) Si se activa la opción “Dia – Oficina 2”, entonces se encenderán (activarán) las luces principal y secundaria de la oficina 2, y se apagará (desactivará) la luz de seguridad de dicha oficina.

(25) Si se activa la opción “Dia – Oficina 3”, entonces se encenderán (activarán) las luces principal y secundaria de la oficina 3, y se apagará (desactivará) la luz de seguridad de dicha oficina.

(26) Si se activa la opción “Dia – Todas”, entonces todas las oficinas tendrán las luces principal y secundarias encendidas, y apagadas las luces de seguridad.

La opción Noche contiene las siguientes opciones:



(27) Si se activa la opción “Noche – Oficina 1”, entonces se apagarán las luces principal y secundaria de la oficina 1 y se encenderá la luz de seguridad.

(28) Si se activa la opción “Noche – Oficina 2”, entonces se apagarán las luces principal y secundaria de la oficina 2 y se encenderá la luz de seguridad.

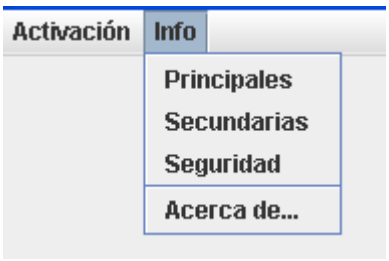
(29) Si se activa la opción “Noche – Oficina 3”, entonces se apagarán las luces principal y secundaria de la oficina 3 y se encenderá la luz de seguridad.

(30) Si se activa la opción “Noche – Todas”, entonces se apagarán todas las luces principales y secundarias de todas las oficinas y se encenderán todas las luces de seguridad.

La opción Salir permitirá finalizar el programa.

F d ió Códi Lib D i i

La opción Info del menú contendrá lo siguiente:



(31) La opción Principales mostrará un JOptionPane donde se indique cuantas luces principales hay encendidas ahora mismo y cuantas apagadas.

(32) La opción Secundarias mostrará un JOptionPane donde se indique cuantas luces secundarias hay encendidas ahora mismo y cuantas apagadas.

(33) La opción Seguridad mostrará un JOptionPane donde se indique cuantas luces de seguridad hay encendidas y cuantas apagadas.

(34) La opción Acerca de... mostrará un JOptionPane que contendrá el nombre del programa y del programador.

MEJORA OPCIONAL

Sería interesante que cada cuadro de verificación estuviera acompañado de una pequeña imagen que representara una bombilla encendida o apagada según el estado de la luz.

Use etiquetas para contener las imágenes. En el **EJERCICIO DE INVESTIGACIÓN 2** puede encontrar más información sobre esto.

PROGRAMACIÓN

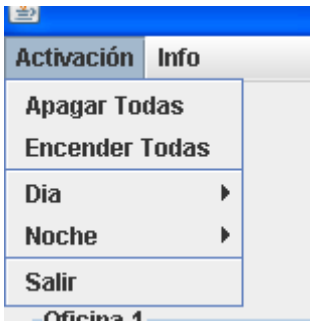
JAVA

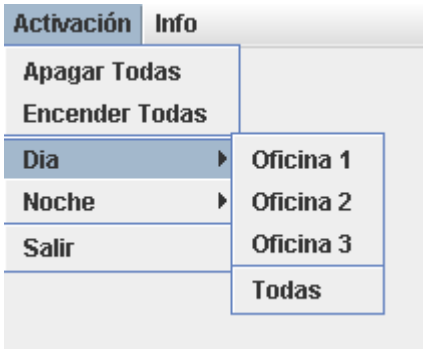
BARRA DE HERRAMIENTAS

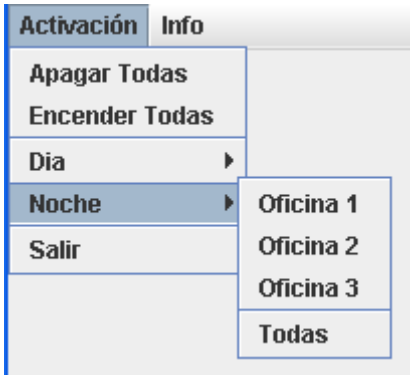
Ejercicio 1

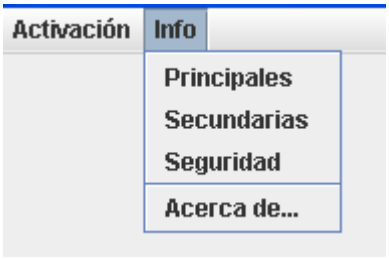
Se va a mejorar el programa de control de luces de la hoja anterior, añadiéndole una barra de herramientas.

El programa anterior tenía un menú con las siguientes opciones:









El objetivo del ejercicio es crear una barra de herramientas que contenga un botón para algunas de las opciones principales del menú. Concretamente, la barra de herramientas contendrá los siguientes botones:

(1) Un botón para la opción Apagar Todas.
(2) Un botón para la opción Encender Todas.
(3) Un botón para activar todas las oficinas en modo “día” (es equivalente a la opción *Activación – Día – Todas*)
(4) Un botón para activar todas las oficinas en modo “noche” (es equivalente a la opción *Activación – Noche – Todas*)
(5) Un botón para conseguir información sobre las luces principales (*Info – Principales*)
(6) Un botón para conseguir información sobre las luces secundarias (*Info – Secundarias*)
(7) Un botón para conseguir información sobre las luces de seguridad (*Info – Seguridad*)

En cada *actionPerformed* de cada botón de la barra de herramientas se tiene que programar el mismo código que el de la opción del menú correspondiente.

PROGRAMACIÓN

JAVA

MENÚS CONTEXTUALES

Ejercicio 1

Se va a añadir una mejora más en el programa de luces realizado en las hojas anteriores. Concretamente se añadirá un menú contextual que contenga las siguientes opciones:

Apagar Todas
Encender Todas
---------- (Separador)
Modo Día
Modo Noche
---------- (Separador)
Principales
Secundarias
Seguridad

Este menú aparecerá cuando el usuario haga clic con el botón derecho del ratón sobre el fondo de la ventana de su programa. Es decir, tendrá que programar el clic del ratón sobre el formulario.

Las opciones del menú contextual coinciden con las opciones de la barra de menús de la siguiente forma:

Apagar Todas	– Activación / Apagar Todas
Encender Todas	– Activación / Encender Todas
Modo Día	– Activación / Día / Todas
Modo Noche	– Activación / Día / Noche
Principales	– Info / Principales
Secundarias	– Info / Secundarias
Seguridad	– Info / Seguridad

Programe dichas opciones de forma que se realice la operación correspondiente a su opción del menú hermana.

PROGRAMACIÓN

JAVA

DIALOGO ABRIR/GUARDAR FICHERO

Notas Iniciales

Supongamos que tiene en su proyecto un objeto del tipo JFileChooser llamado *elegirFichero*.

A través del método *getSelectedFile* obtiene el fichero que ha elegido el usuario. El método *getSelectedFile* devuelve en realidad un objeto del tipo File.

Así pues, puede hacer lo siguiente:

```
int resp;   //una respuesta
File f;     //un fichero

resp=elegirFichero.showOpenDialog(this);
if (resp==JFileChooser.APPROVE_OPTION) {
   f = elegirFichero.getSelectedFile();
}
```

Si observa este código, verá que se crea una variable f de tipo File. Y f es el fichero elegido por el usuario en el cuadro de diálogo Abrir.

Los objetos del tipo File poseen diversos métodos que permiten obtener información sobre el fichero y realizar algunas operaciones con ellos. He aquí algunos métodos que se pueden usar:

getName() **- Devuelve el nombre del fichero.**

Por ejemplo:

```
String nombre = f.getName();
//La variable nombre contendría el nombre del fichero f
```

getPath() **- Devuelve el camino completo del fichero.**

Por ejemplo:

```
String camino = f.getPath();
//La variable camino contendría el camino del fichero f
```

exists() **- Devuelve verdadero o falso según exista o no.**

Por ejemplo:

```
if (f.exists() == true ) {
      JOptionPane.showMessageDialog(null, “El fichero f existe”);
} else {
      JOptionPane.showMessageDialog(null, “El fichero f no existe”);
}
```

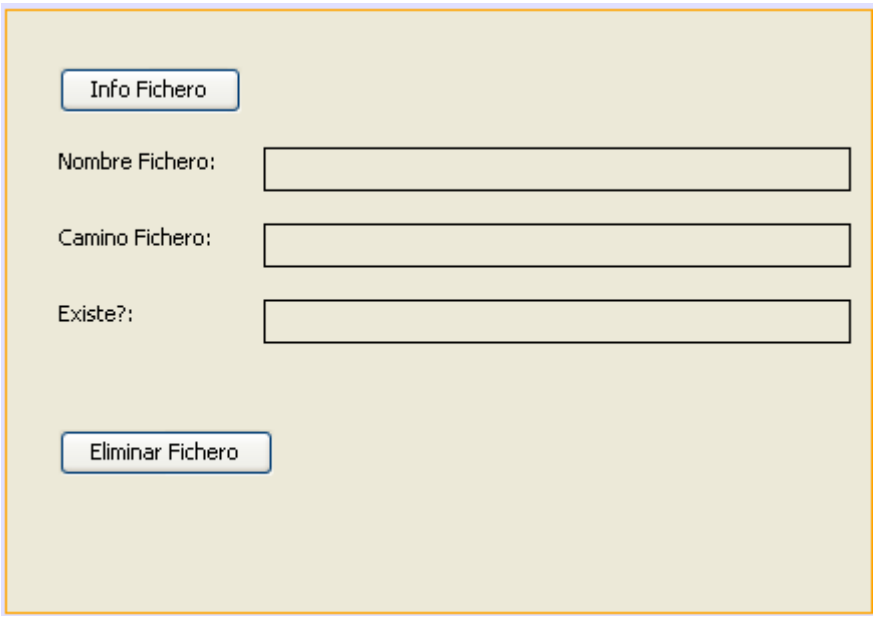
delete() **- Borra el fichero**

Por ejemplo:

```
f.delete();  //Borra el fichero f
```

Ejercicio

Teniendo en cuenta las ideas expuestas, realizar un programa cuya ventana tenga el siguiente aspecto:



El programa funcionará de la siguiente forma:

(11) Si el usuario pulsa el botón “Info Fichero”, aparecerá el cuadro de diálogo (JFileChooser) Abrir, donde el usuario podrá elegir un fichero. Una vez que el usuario elija un fichero y pulse el botón “Abrir”, aparecerán los datos del fichero en las distintas etiquetas.

(12) En la etiqueta nombre del fichero aparecerá el nombre del fichero (use *getName()*)

(13) En la etiqueta camino del fichero aparecerá el camino completo del fichero (use *getPath()*)

(14) En la etiqueta existe? aparecerá un mensaje indicando si el fichero existe o no (use *exists()*) Queda claro que si el fichero no existe no se puede visualizar ni su nombre ni su camino.

(15) Si el usuario, en cambio, pulsa el botón “Eliminar Fichero”, entonces el programa mostrará el cuadro de diálogo “Abrir” y una vez que el usuario elija un fichero, el programa lo borrará (use *delete()*)

Pida una confirmación antes de eliminar el fichero. Tenga cuidado al comprobar esta opción.

PROGRAMACIÓN

JAVA

VARIABLES GLOBALES

Ejercicio 1

Realizar un programa cuya ventana contenga los siguientes elementos:

- Un cuadro de texto llamado txtNumero.
- Un botón “Acumular” llamado btnAcumular.
- Un botón “Resultados” llamado btnResultados.
- Un botón “Reiniciar” llamado btnReiniciar.
- Una etiqueta con borde llamada etiMayor.
- Una etiqueta con borde llamada etiSuma.
- Una etiqueta con borde llamada etiMedia.
- Una etiqueta con borde llamada etiCuenta.

El programa funcionará de la siguiente forma:

- El usuario introducirá un número en el cuadro de texto txtNumero y luego pulsará el botón “Acumular”. En ese momento se borrará el número introducido en el cuadro de texto.

- Este proceso lo repetirá el usuario varias veces.

- Cuando el usuario pulse el botón “Resultados”, deben aparecer en las etiquetas los siguientes datos:

    - El número mayor introducido hasta el momento.
    - La suma de los números introducidos hasta el momento.
    - La media de los números introducidos hasta el momento.
    - Cuantos números ha introducido el usuario hasta el momento.

- El botón “Reiniciar” reinicia el proceso borrando todo lo que hubiera en las etiquetas de resultados y reiniciando las variables globales.

Variables globales a usar

Para que sirva de ayuda, se recomienda que use las siguientes variables globales:

- Una variable double llamada *mayor*. Contendrá en todo momento el número mayor introducido.
- Una variable double llamada *suma*. Contendrá en todo momento la suma de los números introducidos.
- Una variable int llamada *cuenta*. Contendrá en todo momento la cuenta de todos los números introducidos hasta ahora.

Al comenzar el programa, y al pulsar el botón “Reiniciar”, será necesario que estas tres variables se inicien a 0.

Nota. Se supondrá que todos los número introducidos serán mayores o iguales a 0.

Ejercicio 2

Realizar un programa que contenga los siguientes elementos en su ventana:

- Un cuadro de texto llamado txtNumero.
- Un cuadro de texto llamado txtApuesta
- Un botón “Jugar” llamado btnJugar.
- Una etiqueta llamada etiNumero.
- Una etiqueta llamada etiResultado que contendrá inicialmente un “100”.

El programa funcionará de la siguiente forma:

- Se trata de un juego de adivinación. El usuario introducirá un número entre 1 y 10 en el cuadro de texto txtNumero.

- Luego introducirá una apuesta en el cuadro de texto txtApuesta.

- Y a continuación pulsará el botón “Jugar”.

- El programa calculará entonces un número al azar entre 1 y 10 y lo mostrará en la etiqueta etiNumero.

- Si el número introducido por el usuario coincide con el número elegido al azar por la máquina, entonces el usuario gana y se le debe sumar lo apostado a la cantidad que tenía en dicho momento. Si el usuario pierde entonces se le debe restar lo apostado a la cantidad que tenía. El total que le quede debe aparecer en la etiqueta resultado.

- Al empezar el programa el usuario tiene 100 euros de bote.

Las variables globales a usar son las siguientes:

- Solo se necesita una variable global de tipo double a la que puedes llamar *ganancias*. Esta variable estará inicializada al comienzo del programa a 100.

Para hacer que el ordenador calcule un número aleatorio entre 1 y 10 debes usar el siguiente código:

```
int n;  //el numero
double aleatorio;
aleatorio = Math.random();
aleatorio = aleatorio * 10;
aleatorio = Math.floor(aleatorio);
aleatorio = aleatorio + 1;
n = (int) aleatorio;
```

(Más información sobre la generación de números aleatorios en el Ejercicio de Investigación Nº3)

PROGRAMACIÓN

JAVA

CENTRALIZAR CÓDIGO

Ejercicio 1

Realice un programa cuya ventana tenga el siguiente aspecto:

Esta ventana consta de los siguientes elementos:

(1) Un cuadro de texto txtHoras que contiene un 0.
(2) Un cuadro de texto txtMinutos que contiene un 0.
(3) Varias etiquetas de información. Una contiene la palabra “Horas”, otra contiene la palabra “Minutos” y otra contiene el símbolo dos puntos (:).
(4) Un cuadro de texto vacío al que se le llamará txtCantidadMin
(5) Un botón con el signo + al que se le llamará btnSumar.
(6) Un botón con el signo – al que se le llamará btnRestar.

Además:

(7) Una barra de herramientas con el nombre *barraHerramientas*, que contenga tres botones:
    h. Un botón herramientasSumar.
    i. Un botón herramientasRestar.
    j. Un botón herramientasReiniciar.

(11)Asigne a estos botones el icono que quiera. Por ejemplo, iconos de relojes.

Además:

F d ió Códi Lib D i i

(12) La ventana contendrá una barra de menús a la que puede llamar *barraMenus*. Esta barra contiene una opción “Acción” a la que puede llamar *menuAccion*.

(13) Dentro de la opción Acción, tendrá estas otras opciones:

(14) Una opción “Sumar” a la que llamará *menuSumar*.
(15) Una opción “Restar” a la que llamará *menuRestar*.
(16) Una opción “Reiniciar” a la que llamará *menuReiniciar*.
(17) Y una opción “Salir” a la que llamará *menuSalir*.
(18) Además tendrá una serie de separadores para mejorar la presentación del menú.

El programa funcionará de la siguiente forma:

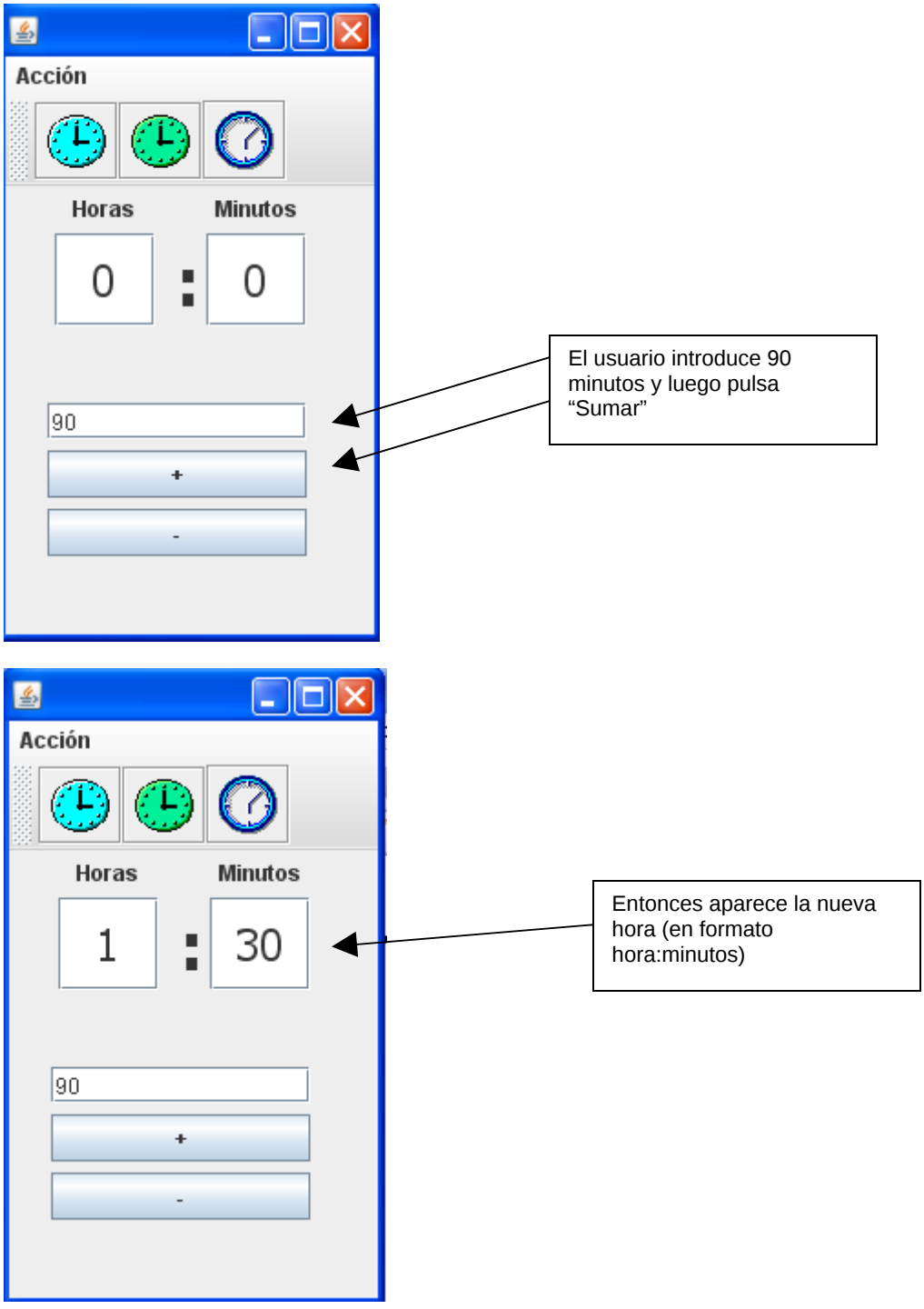
(19) El usuario introducirá en el cuadro de texto txtCantidadMin una cantidad de minutos.

(20) Si luego, pulsa el botón sumar, o bien activa el botón sumar de la barra de herramientas, o bien activa la opción sumar del menú, entonces los minutos escritos se suman a la hora que se muestra.

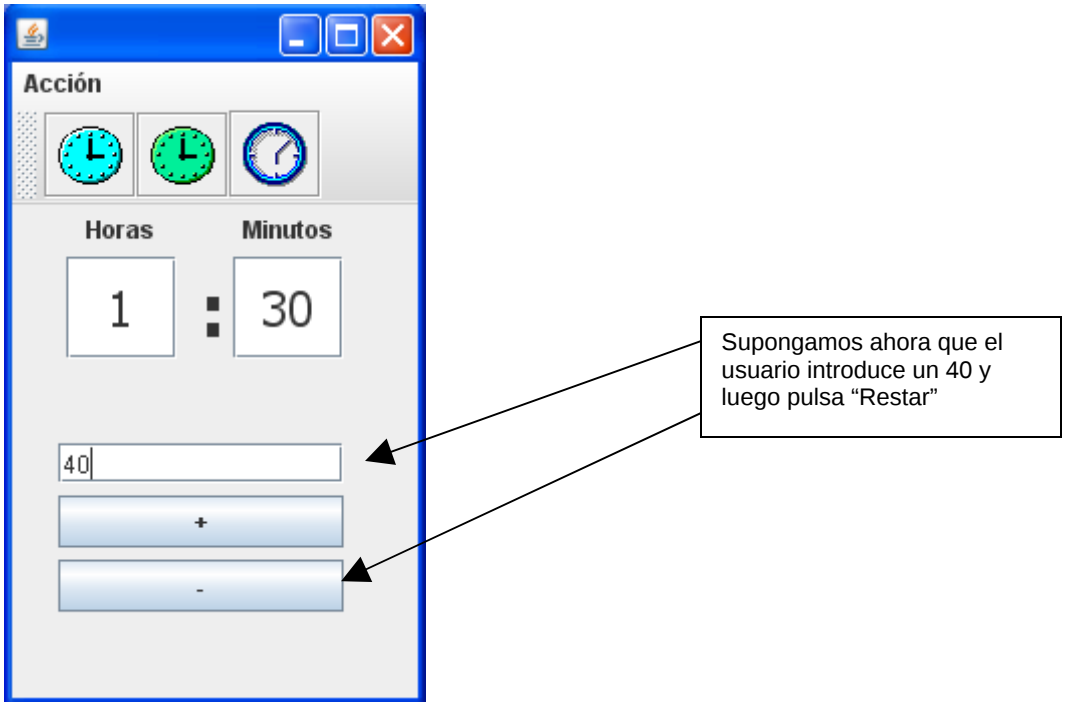
(21) En cambio, si el usuario pulsa el botón restar, o bien activa el botón restar de la barra de herramientas, o bien activa la opción restar de la barra de menús, entonces los minutos escritos se restan a la hora que se muestra.

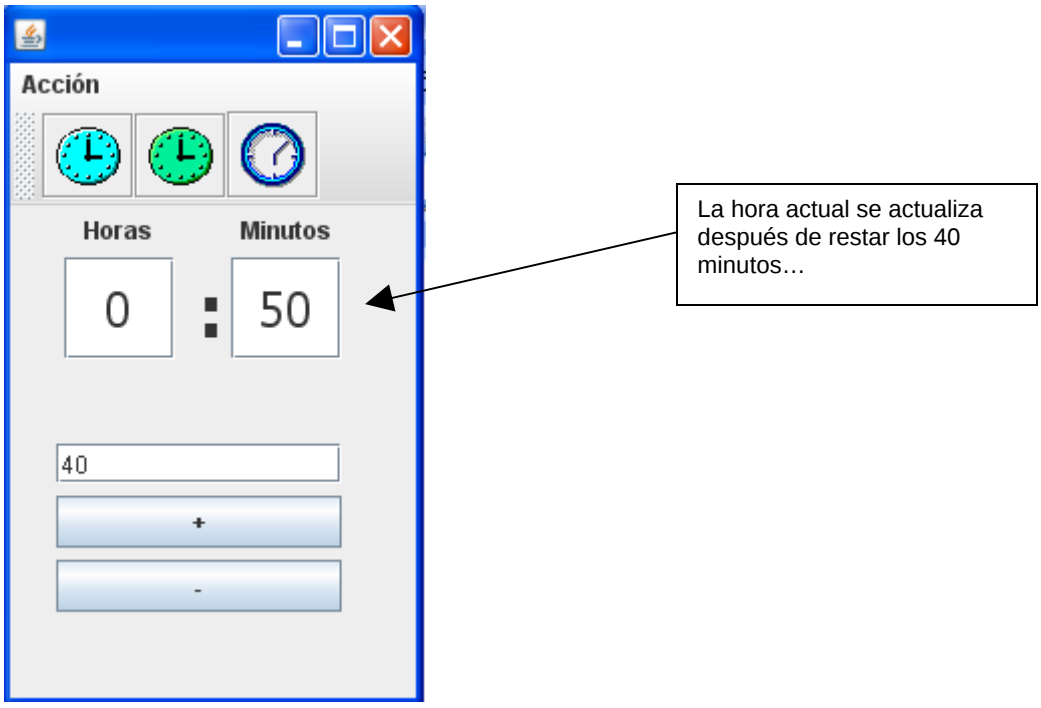
(22) Si el usuario pulsa el botón reiniciar de la barra de menús o pulsa la opción reiniciar del menú, entonces la hora que se muestra se reinicia a las 00:00 horas.

Un ejemplo:



F d ió Códi Lib D i i





A TENER EN CUENTA

(23) Se aconseja que realice un procedimiento Sumar que se encargue de hacer la suma de minutos.

(24) Realice también un procedimiento Restar que se encargue de hacer la resta de minutos.

(25) Realice un procedimiento Reiniciar que se encargue de reiniciar la hora.

F d ió Códi Lib D i i

(26) Llame desde los distintos eventos al procedimiento que corresponda.

(27) Se aconseja tener una variable global *hora* que contenga en todo momento la hora actual.

(28) Se aconseja tener una variable global *minutos* que contenga en todo momento los minutos actuales.

(29) Para calcular la nueva hora use estos algoritmos:

Sumar minutos…

Supongamos que la hora actual sea las 14:25
Y que se quiera sumar 70 minutos

Primero se hace el cálculo total de minutos: 14 * 60 + 25 = 865

Luego se suman los minutos: 865 + 70 = 935

El resultado se divide entre 60 y ya tenemos la nueva hora: 935 / 60 = **15**

El resto de la división son los minutos: 935 % 60 = **35**

La nueva hora por tanto es las **15 : 35**

Restar minutos…

Supongamos que la hora actual sea las 14:25
Y que se quiera restar 70 minutos

Primero se hace el cálculo total de minutos: 14 * 60 + 25 = 865

Luego se le restan los minutos: 865 – 70 = 795

El resultado se divide entre 60 y ya tenemos la nueva hora: 795 / 60 = **13**

El resto de la división son los minutos: 795 % 60 = **15**

La nueva hora por tanto es las **13 : 15**

PROGRAMACIÓN

JAVA

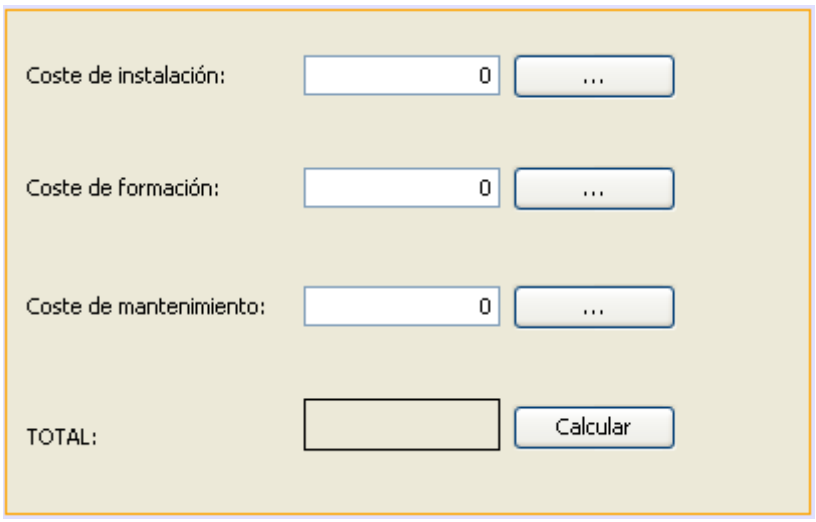
CUADROS DE DIÁLOGO

Ejercicio 1

Se pide realizar un programa que facilite el cálculo del precio de implantación de un determinado software en una empresa.

Se pretende calcular el coste de instalación, el coste de formación y el coste de mantenimiento.

Para ello, debe realizar un proyecto con una ventana como la que sigue:

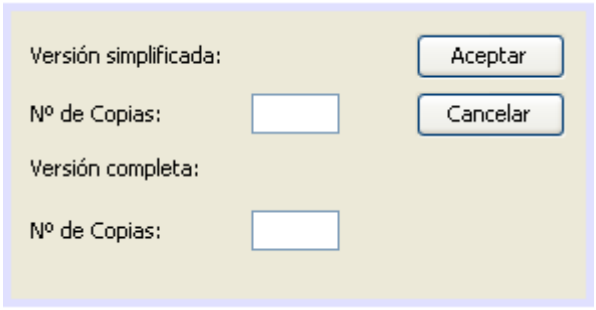


Esta ventana tiene tres cuadros de textos, para cada uno de los costes. Si se desea, se puede introducir directamente cada uno de los costes en estos cuadros de textos y al pulsar el botón Calcular aparecerá la suma de los tres en la etiqueta Total.

Por otro lado, cada cuadro de texto tiene asignado un botón con tres puntos que permitirá facilitar el cálculo de cada coste a través de un cuadro de diálogo.

Cuadro de diálogo Coste de Instalación

En el caso de que se pulse el botón correspondiente al Coste de Instalación, el cuadro de diálogo que debe aparecer tiene que ser el siguiente (haz que sea modal):



Aquí se indicará el número de copias de la versión simplificada del programa a instalar, y el número de copias de la versión completa. Al pulsar el botón Aceptar, se mostrará el coste total

por *instalación* en el cuadro de texto correspondiente de la ventana principal y luego se cerrará el cuadro de diálogo.

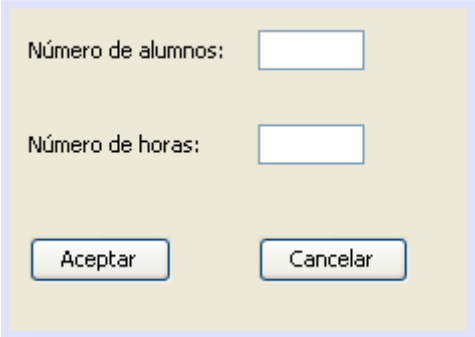
Para calcular el coste de instalación, ten en cuenta lo siguiente:
Cada copia de la versión simplificada cuesta 120 euros, y cada copia de la versión completa cuesta 180 euros. Si por ejemplo, el usuario introduce 1 copia simplificada y 2 de la versión completa, el coste total será:

coste por instalación = 120 * 1 + 180 * 2 = 480

Si se pulsa el botón Cancelar, el cuadro de diálogo se cierra y no se muestra nada en el cuadro de texto del coste de instalación de la ventana principal.

Cuadro de diálogo Coste de Formación

En el caso de que se pulse el botón correspondiente al Coste de Formación, el cuadro de diálogo que debe aparecer tiene que ser el siguiente (haz que sea modal):



El coste de formación se calculará multiplicando el número de alumnos por el número de horas por 10 euros.

Por ejemplo, si el usuario introduce 3 alumnos y 12 horas, el coste por formación será:

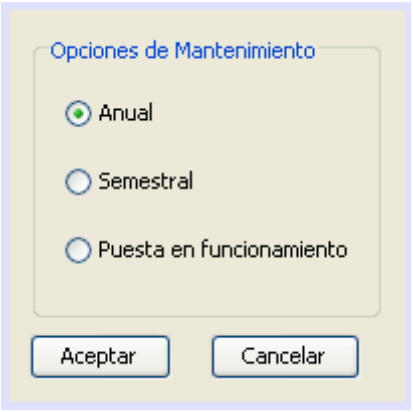
coste por formación = 2 * 12 * 10 = 240

Al pulsar el botón Aceptar, se calculará dicho coste y se introducirá en el cuadro de texto del coste de formación de la ventana principal (luego el cuadro de texto se cerrará).

Si se pulsa el botón Cancelar, el cuadro de diálogo simplemente se cerrará, sin que ocurra nada más.

Cuadro de diálogo Coste de Mantenimiento

En el caso de que se pulse el botón correspondiente al Coste de Mantenimiento, el cuadro de diálogo que debe aparecer tiene que ser el siguiente (haz que sea modal):



Al pulsar Aceptar, el programa calculará el coste de mantenimiento y lo presentará en el cuadro de texto correspondiente de la ventana principal (y luego se cerrará el cuadro de diálogo)

La forma de calcular el coste de mantenimiento es la siguiente:

(1) Si se elige un mantenimiento Anual, entonces el coste será de 600 euros.
(2) Si se elige un mantenimiento Semestral, entonces el coste será de 350 euros.
(3) Si se elige un mantenimiento del tipo *Puesta en funcionamiento* entonces el coste será de 200 euros.

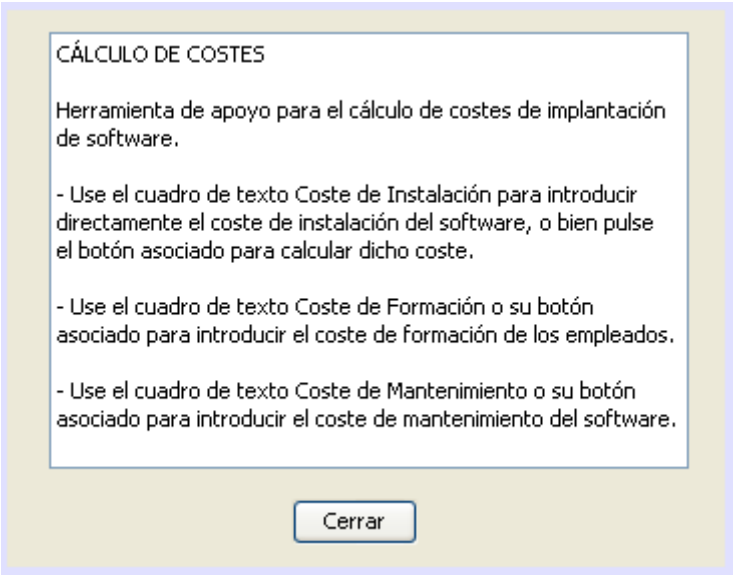
Si se pulsa el botón Cancelar, el cuadro de diálogo se cierra sin más.

MEJORAS

Ayuda

Añade un botón a la ventana principal con esta forma:

Al pulsar este botón, aparecerá un cuadro de diálogo de ayuda con este aspecto (haz que no sea modal y que esté siempre por encima de la ventana principal:



Al pulsar el botón Cerrar, el cuadro de diálogo simplemente se cerrará.

Para hacer el texto, usa el objeto *JTextPane*, y su propiedad *text*. Este objeto permite crear cuadros con gran cantidad de texto. Ten en cuenta que cuando añadas un *JTextPane* este aparecerá dentro de un *JScrollPane*.

Pantalla de Splash (Presentación)

Se añadirá un nuevo cuadro de diálogo al proyecto que servirá como pantalla de presentación (a estos cuadros de diálogo se les suele denominar *Splash*)

Diseñe este cuadro de diálogo como quiera. Puede añadir varias imágenes, colocar el texto donde quiera, etc.

Debe mostrar este cuadro de diálogo al comenzar el programa (en el evento *windowOpened* del formulario)

Si quiere mostrar el cuadro de diálogo de la presentación en el centro de la pantalla, puede usar este código (se supone que el cuadro de diálogo se llama *dialogoPres*):

```
int x=(int) (Toolkit.getDefaultToolkit().getScreenSize().getWidth()/2 -  dialogoPres.getWidth()/2);
int y=(int) (Toolkit.getDefaultToolkit().getScreenSize().getHeight()/2 -  dialogoPres.getHeight()/2);
dialogoPres.setLocation(x,y);
```

El cuadro de diálogo debe tener un botón Cerrar, o bien, debe tener la posibilidad de cerrarse cuando el usuario haga clic sobre él en cualquier parte.

Para mejorar su cuadro de diálogo de presentación, se recomienda que active la propiedad del cuadro de diálogo llamada *undecorated*. Esta propiedad oculta la barra de título del cuadro de diálogo.

PROGRAMACIÓN

JAVA

DISEÑO DE VENTANA DESDE CÓDIGO

Ejercicio 1

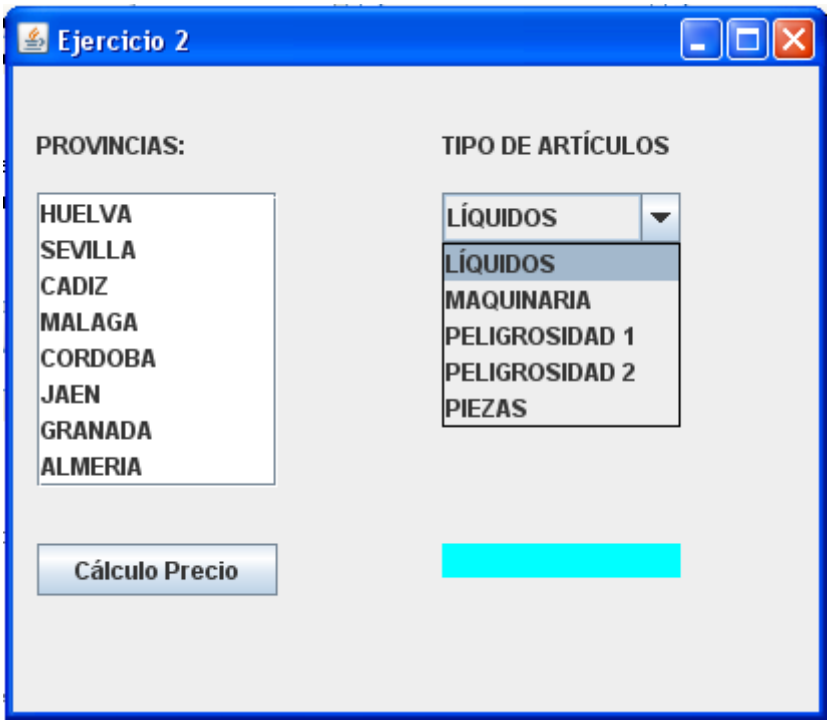
Realice un proyecto cuya ventana tenga el siguiente diseño. Este diseño lo tiene que hacer **totalmente desde código**, sin usar la ventana de diseño del NetBeans:

Tenga en cuenta lo siguiente:

- La ventana debe tener como título “Ejercicio 1”
- El tamaño de la ventana al arrancar el programa debe ser de 350 x 400
- Haga que la ventana aparezca en el centro de la pantalla.
- El recuadro de color verde con el texto RESULTADO es una etiqueta.
- El botón con el texto “metros/seg” es un JToggleButton y debe estar activado al arrancar el programa.
- En la parte inferior de la ventana hay un JSlider. Su valor mínimo será 0 y su valor máximo será 100. El valor inicial del JSlider tiene que ser 20.
- El JSlider debe mostrar sus marcas (método *setPaintTicks*) y la separación entre marcas debe ser de 5 (método *setMinorTickSpacing*)

Ejercicio 2

Realice un proyecto cuya ventana principal tenga el siguiente diseño. Este diseño lo tiene que hacer **directamente desde código**, sin usar la ventana de diseño del NetBeans.



Tenga en cuenta lo siguiente:

- La ventana debe tener el título “Ejercicio 2”
- El tamaño de la ventana debe ser de 300x300
- La ventana debe aparecer en el centro de la pantalla al arrancar el programa.
- En la parte derecha la ventana contiene un JList con las ocho provincias andaluzas.
- En la parte izquierda la ventana contiene un JComboBox con unos tipos de artículos. (El combo se muestra desplegado para que puedas ver la lista de tipos de artículos)
- En la parte inferior hay una etiqueta vacía con color CYAN de fondo.

F d ió Códi Lib D i i

PROGRAMACIÓN

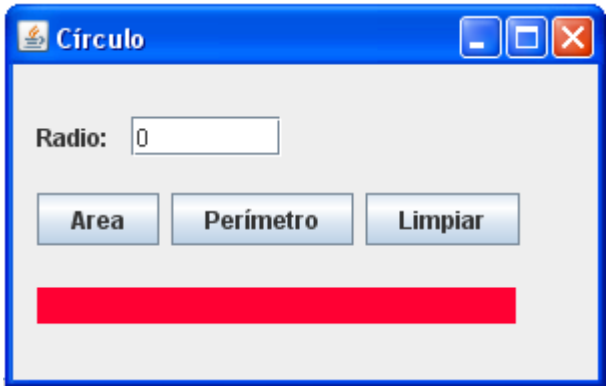
JAVA

ASIGNACIÓN DE EVENTOS DESDE CÓDIGO

Ejercicio 1

Realice un programa para calcular el área y el perímetro de un círculo.

La ventana del programa debe tener un aspecto parecido al siguiente:



El usuario introducirá un radio y…

- Si pulsa el botón Área se calculará el área del círculo. (*actionPerformed* del botón de Área)
- Si pulsa el botón Perímetro se calculará el perímetro del círculo. (*actionPerformed* del botón de Perímetro)
- Si pulsa el botón Limpiar en el cuadro de texto aparecerá un 0 y la etiqueta de resultado (la de color rojo) se vaciará. (*actionPerformed* del botón Limpiar)

Todo el programa debe ser realizado directamente desde código, sin usar la ventana de diseño de NetBeans.

Ejercicio 2

Realice un programa para calcular la velocidad de un vehículo. La ventana del programa tendrá el siguiente diseño:

F d ió Códi Lib D i i

El programa funcionará de la siguiente forma:

- El usuario introducirá un Espacio y un Tiempo, y al pulsar Enter (*actionPerformed*) en cualquiera de los dos cuadros de textos aparecerá la velocidad en la etiqueta de resultados de color amarillo.

  La velocidad se calcula así: velocidad = espacio / tiempo

- El usuario también puede introducir una Velocidad y un Tiempo en los cuadros de textos de la parte derecha. Si pulsa Enter en cualquiera de estos cuadros de texto (*actionPerformed*) entonces se calculará el Espacio en la etiqueta roja de resultados.

  El espacio se calcula así: espacio = velocidad * tiempo

Tendrá que programar por tanto el evento actionPerformed de cada cuadro de texto.

Tanto el diseño como la programación de eventos debe realizarla directamente desde código, sin usar la ventana de diseño del NetBeans.

F d ió Códi Lib D i i

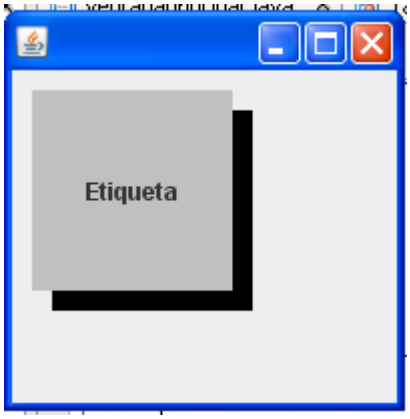
PROGRAMACIÓN

JAVA

ASIGNACIÓN DE EVENTOS DESDE CÓDIGO. EVENTOS DE RATÓN

Ejercicio 1

Realice un programa que tenga el siguiente aspecto:



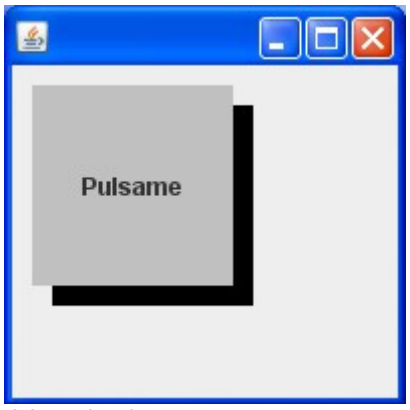
En la ventana hay dos etiquetas. Una de ellas es de color negro y la otra es de color gris. Están situadas de tal forma que la etiqueta de color negro simula la sombra de la otra etiqueta.

La etiqueta de color gris debe tener asociados los siguientes eventos del ratón:

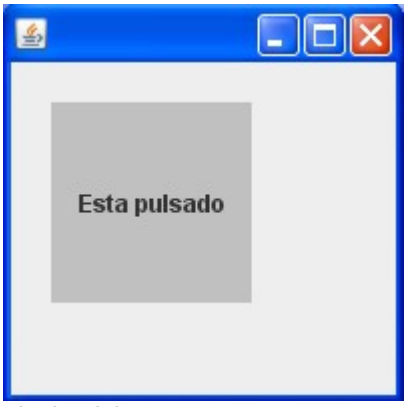
- mouseEntered. Cuando el ratón entre en la superficie de la etiqueta debe cambiar su texto por “Pulsame”.

- mouseExited. Cuando el ratón salga de la superficie de la etiqueta debe cambiar el texto de esta otra vez por “Etiqueta”.

- mousePressed. Cuando se pulse un botón del ratón sobre la etiqueta, esta debe moverse de sitio de forma que parezca que se ha pulsado como si fuera un botón (observa la imagen)



(sin pulsar)



(pulsada)

Para conseguir esto, debes colocar la etiqueta gris sobre la etiqueta negra.

Cuando suceda este evento también debes cambiar el texto de la etiqueta por “Está pulsado”

F d ió Códi Lib D i i

- **mouseReleased. Cuando se suelte el botón del ratón, la etiqueta volverá a su posición normal, y el texto cambiará a “Pulsado”.**

F d ió Códi Lib D i i

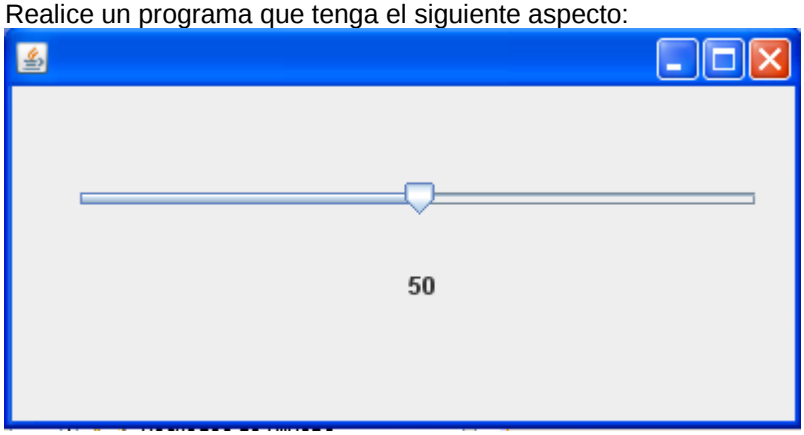
PROGRAMACIÓN

JAVA

ASIGNACIÓN DE EVENTOS DESDE CÓDIGO. GENERALIDADES

Ejercicio 1

Realice un programa que tenga el siguiente aspecto:



En la ventana hay simplemente un JSlider (deslizador) y una etiqueta que contiene inicialmente el valor 50.

Se pide lo siguiente:

1. Realice el diseño de la ventana desde código.

2. Interesa que cuando se modifique el valor del JSlider, dicho valor aparezca en la etiqueta. Esto se tiene que realizar programando el evento correspondiente desde código. Aquí tiene la información necesaria:

    a. El evento a programar del JSlider se llama: *stateChanged*
    b. Pertenece al oyente llamado *ChangeListener*
    c. Asignarás el oyente al JSlider con el método *addChangeListener*
    d. El evento lleva como parámetro un objeto evt del tipo *ChangeEvent*

3. Interesa que al girar la rueda del ratón cambie el valor del JSlider. Para controlar el movimiento de la rueda del ratón debe asignar un evento al formulario (**this**). Aquí tiene la información necesaria del evento para la rueda del ratón:

    a. El evento a programar de el formulario (this) se llamad *mouseWheelMoved*
    b. Pertenece al oyente llamado *MouseWheelListener*
    c. Se asigna a la ventana (this) con el método *addMouseWheelListener*
    d. El evento lleva como parámetro un objeto evt del tipo *MouseWheelEvent*

    PISTA:
    El objeto evt pasado como parámetro del evento de la rueda del ratón contiene información sobre como se ha movido la rueda. Este objeto tiene un método llamado *getUnitsToScroll* que devuelve un número entero que indica cuanto se ha movido la rueda. Este número puede ser positivo o negativo según hacia donde se movió la rueda.

    Aprovechando esto, el evento de la rueda del ratón se puede programar así:

```
int valor = slider.getValue();  //slider es el objeto JSlider
valor = valor + evt.getUnitsToScroll();
slider.setValue(valor);
```

F d ió Códi Lib D i i

PROGRAMACIÓN

JAVA

ASIGNACIÓN DE EVENTOS DESDE CÓDIGO. EVENTOS DE VENTANA

Ejercicio 1

La ventana sobre la que trabajamos también tiene sus propios eventos. Estos eventos son los siguientes:

(65)windowOpened – Sucede cuando la ventana se abre.
(66)windowClosing – Sucede cuando la ventana va a cerrarse.
(67)windowActivated – Sucede cuando la ventana se activa.
(68)windowDeactivated – Sucede cuando la ventana se desactiva.

Los eventos de ventana pertenecen al adaptador WindowAdapter.

El adaptador se asigna a la ventana (this) a través del método addWindowListener.

Todos los eventos llevan como parámetro un objeto evt del tipo WindowEvent.

Teniendo en cuenta esto, realizar un programa cuya ventana tenga únicamente una etiqueta.

El programa funcionará de la siguiente forma:

(69)Cuando se active la ventana, aparecerá un mensaje “La ventana se ha activado” en la etiqueta central.
(70)Cuando se desactive la ventana, aparecerá un mensaje “La ventana se ha desactivado” en la etiqueta central.

Nota: Una forma de desactivar la ventana es hacer clic sobre la barra de tareas. Luego se puede volver a activar haciendo clic sobre la ventana.

Además, el programa debe hacer esto otro:

(71)Cuando se abra la ventana, debe aparecer un mensaje de bienvenida (un JOptionPane)
(72)De la misma forma, cuando se cierre la ventana, debe aparecer un mensaje de despedida.

PROGRAMACIÓN

JAVA

VECTORES DE COMPONENTES

Ejercicio 1

Realizar un programa en cuya ventana aparezcan 7 JToggleButton (use un vector)

Cada uno de los botones debe contener como texto un día de la semana.

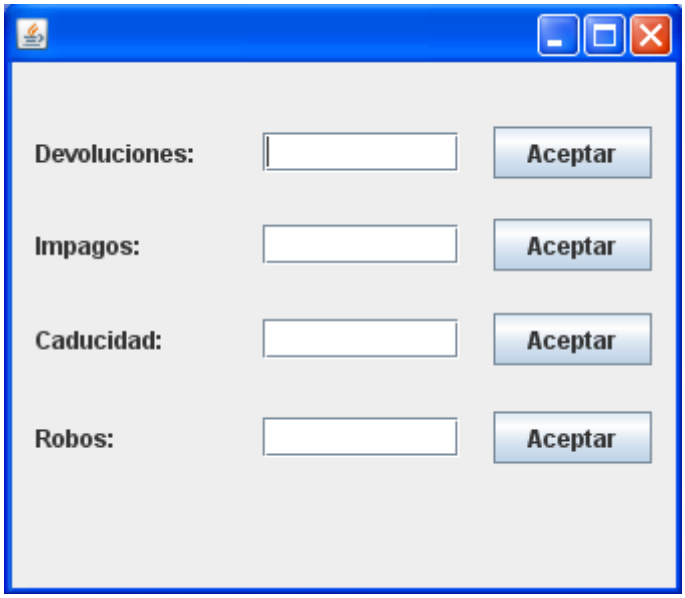
Los botones deben aparecer en horizontal:

Añada un botón “Aceptar” a la ventana (un botón normal) y dos etiquetas.

Al pulsar el botón “Aceptar” debe aparecer en una de las etiquetas el número de botones activados. También debe aparecer en la otra etiqueta los días de la semana elegidos (activos)

Ejercicio 2

Se necesita hacer un programa que muestre la siguiente ventana:



Diseñe la ventana totalmente desde código y usando vectores. Tendrá que usar los siguientes vectores:

- Un vector de etiquetas (JLabel) para cada etiqueta. Necesitarás también un vector auxiliar de String que contenga los textos de las etiquetas: “Devoluciones”, “Impagos”, “Caducidad” y “Robos”.

- Un vector de cuadros de texto (JTextField)

- Un vector de botones (JButton)

Tendrás que construir cada vector y luego tendrás que construir los elementos de dichos vectores, colocándolos en la ventana.

El programa no tiene que hacer nada, solo limítese a diseñar la ventana.

F d ió Códi Lib D i i

PROGRAMACIÓN

JAVA

VECTORES DE COMPONENTES

Ejercicio 1

Realizar un programa donde aparezcan diez botones conteniendo los números entre el 0 y 9.

Todos estos botones pertenecerán a un vector de JButton, y tendrán asociado un evento *actionPerformed*.

Cada vez que se pulse uno de los botones, en un cuadro de texto de la ventana se añadirá el dígito correspondiente, como si estuvieramos usando una calculadora.

Añadir también un botón “Borrar” (no perteneciente al vector) que permita borrar el contenido del cuadro de texto.

El aspecto del programa puede ser similar al siguiente:

Se pulsó el 9 y luego el 5

COLOCACIÓN DE LOS BOTONES

Para la colocación de los botones en el JFrame, se puede usar un vector de posiciones X y un vector de posiciones Y que contengan las posiciones (X,Y) de cada botón:

```
int vectorx[]={10,40,70,10,40,70,10,40,70,10};
int vectory[]={10,40,70,10,40,70,10,40,70,10};
```

Y luego se puede aplicar los valores de estos vectores en el momento de usar *setBounds*:

```
for (…) {
      …
      vBotones[i].setBounds(vectorx[i],vectory[i], 20,20);
      …
}
```

Ejercicio 2

Se pide hacer un programa que muestre 8 etiquetas, cada una con el nombre de una provincia andaluza. Estas etiquetas estarán definidas al principio con color azul de fondo y texto negro.

Creará un vector para hacer el programa que contenga las ocho etiquetas (vector de JLabel) y las situará en el formulario como desee.

El programa debe funcionar de la siguiente forma:

- Al hacer clic sobre una etiqueta (*mouseClicked*), el color de fondo de esta cambiará a verde, mientras que el color de fondo de todas las demás se colocará en azul (para cambiar el color de fondo: *setBackground* y *setOpaque*)

- Al sobrevolar el ratón la etiqueta (evento *mouseEntered*) el color del texto de la etiqueta se pondrá en amarillo (para cambiar el color de texto: *setForeground*).

- Al abandonar el ratón la etiqueta (evento *mouseExited*) el color del texto de la etiqueta volverá a ser de color negro.

PROGRAMACIÓN

JAVA

POO. CREACIÓN Y USO DE CLASES PROPIAS

Ejercicio 1

Crear un proyecto Java en NetBeans cuya ventana principal tenga el siguiente aspecto:

Ten en cuenta lo siguiente:

(1) Al iniciarse el programa, debe estar activada por defecto la opción *Pared Norte*.

(2) El botón *Asignar* asignará el ancho y alto que se haya introducido a la pared que esté seleccionada en ese momento.

(3) El botón *Area* mostrará en un JOptionPane el área de la pared seleccionada en ese momento.

(4) El botón *Perímetro* mostrará en un JOptionPane el perímetro de la pared seleccionada en ese momento.

CLASE RECTANGULO

PARA HACER ESTE EJERCICIO USARÁ UNA CLASE DE CREACIÓN PROPIA LLAMADA RECTÁNGULO.

La clase Rectangulo debe tener las siguientes características:

F d ió Códi Lib D i i

CLASE RECTANGULO

Nombre de la clase: Rectangulo

Propiedades de los objetos de la clase Rectangulo:

Base (double)
Altura (double)

Valores iniciales de las propiedades de los objetos de la clase Rectangulo:

Base – 100
Altura – 50

Métodos:

Métodos set:

setBase – permite asignar un valor a la propiedad Base.
setAltura – permite asignar un valor a la propiedad Altura.

Métodos get:

getBase – devuelve el valor de la propiedad Base
getAltura – devuelve el valor de la propiedad Altura
getArea – devuelve el área del rectángulo
getPerímetro – devuelve el perímetro del rectángulo

Otros métodos:

Cuadrar – este método debe hacer que la Altura tenga el valor de la Base.

TENDRÁ QUE PROGRAMAR ESTA CLASE E INCLUIRLA EN SU PROYECTO

UNA VEZ INCLUIDA ESTA CLASE, CREARÁ CUATRO OBJETOS DE LA CLASE RECTÁNGULO, CADA UNO DE LOS CUALES HARÁ REFERENCIA A UNA DE LAS CUATRO PAREDES:

Objetos de la clase Rectángulo que usará en el proyecto:

(5) ParedNorte
(6) ParedSur
(7) ParedOeste
(8) ParedEste

Para programar cada botón de la ventana principal, solo tiene que ayudarse dando órdenes a las cuatro paredes o pidiéndoles información.

F d ió Códi Lib D i i

Ejercicio 2

PRIMERA PARTE

Trabajamos de programador para una empresa de venta por correo. Esta empresa recibe pedidos de clientes y necesita controlar la situación de cada pedido.

Para facilitar la realización de aplicaciones para esta empresa se decide crear una CLASE de objetos llamada PEDIDO.

La Clase Pedido permitirá crear objetos de tipo pedido. Estos objetos nos proporcionarán información sobre el estado del pedido y nos facilitará la programación de los proyectos para la empresa.

SE PIDE PROGRAMAR LA CLASE **PEDIDO** TENIENDO EN CUENTA SUS CARACTERISTICAS, LAS CUALES SE MENCIONAN A CONTINUACIÓN:

F d ió Códi Lib D i i

CLASE PEDIDO

Nombre de la Clase: **Pedido**

Propiedades de los objetos de la Clase Pedido:

Articulo: una cadena que indica el nombre del artículo que se ha pedido.
Unidades: un entero indicando las unidades pedidas.
Precio: un double indicando el precio unidad.
GastosEnvio: un double indicando los gastos de envío.
Descuento: un double indicando el tanto por ciento de descuento.

Valores iniciales de las propiedades de los objetos de la Clase Pedido:

Articulo: **(cadena vacía)**
Unidades: **1**
Precio: **0**
GastosEnvio: **3**
Descuento: **0**

Métodos *set*

setArticulo – permite asignar el nombre del artículo al objeto pedido
setUnidades – permite asignar el número de unidades pedidas
setPrecio – permite asignar un precio unidad al artículo del pedido
setGastosEnvio – permite asignar la cantidad de gastos de envío del pedido
setDescuento – permite asignar el tanto por ciento de descuento del pedido

Métodos *get*

getArticulo – devuelve el nombre del artículo del pedido
getUnidades – devuelve el número de unidades del artículo
getPrecio – devuelve el precio del artículo del pedido
getGastosEnvio – devuelve los gastos de envío del pedido
getDescuento – devuelve el tanto por ciento de descuento del pedido

getTotalSinIva – devuelve el total sin iva del pedido. Se calcula así:

```
TotalSinIva = (Unidades * Precio) + gastos de envio
```

getIva – devuelve la cantidad de Iva del pedido. Se calcula así:

```
Iva = TotalSinIva * 0,16
```

getTotalMasIva – devuelve el total del pedido más el Iva. Se calcula así:

```
TotalMasIva = TotalSinIva + Iva
```

getTotalDescuento – devuelve el total del descuento. Se calcula así:

```
TotalDescuento = TotalMasIva * Descuento / 100
```

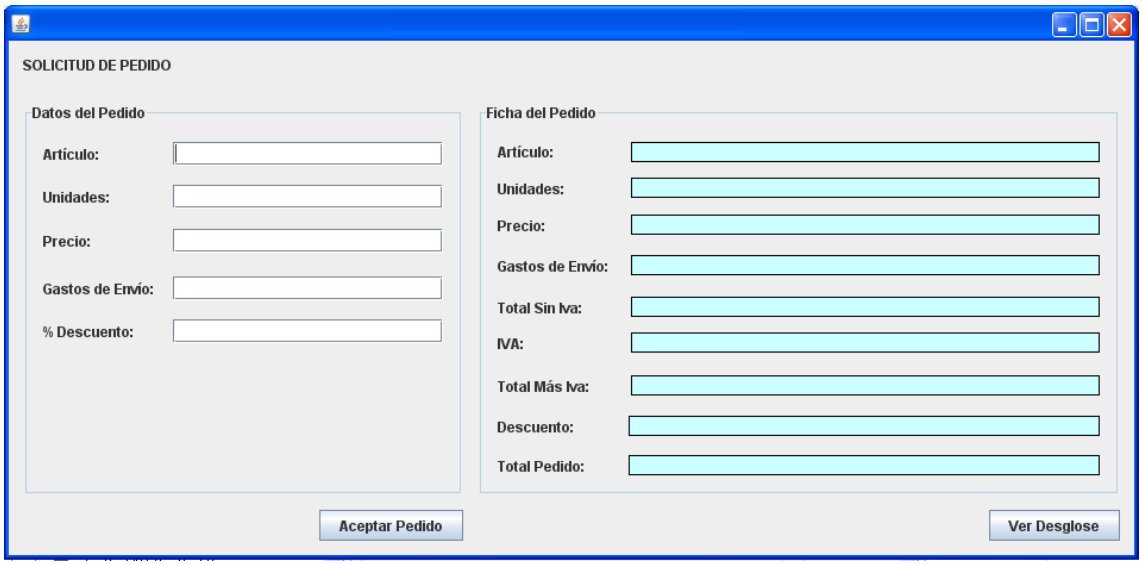
getTotalPedido – devuelve el total del pedido. Se calcula así:

```
TotalPedido = TotalMasIva – TotalDescuento
```

F d ió Códi Lib D i i

SEGUNDA PARTE

Realizar un proyecto cuya ventana principal tenga el siguiente aspecto:



El programa funcionará de la siguiente forma:

(9) En el panel “Datos del Pedido” se introducirán los siguientes datos del pedido a enviar:
   a. Nombre del Artículo
   b. Unidades pedidas
   c. Precio unidad del artículo
   d. Gastos de envío
   e. Tanto Por Ciento de descuento.

(10) Al pulsar el botón “Aceptar Pedido”, todos estos datos deben asignarse a un objeto llamado ***ped* de tipo Pedido**.

(11) Al pulsar el botón “Ver Desglose”, deben aparecer en las distintas etiquetas (de color verde en la imagen) los siguientes datos del pedido:

   a. Nombre del Artículo
   b. Unidades pedidas
   c. Precio unidad
   d. Gastos de envío
   e. Total Sin Iva del pedido
   f. Iva del pedido
   g. Total Más iva
   h. Total de Descuento
   i. Total del Pedido.

(12) Para hacer esto solo tendrá que pedirle información al objeto *ped* usando sus métodos *get* y luego colocar esta información en cada etiqueta.

PARA PODER REALIZAR ESTE PROYECTO SERÁ NECESARIO INCLUIR LA PROGRAMACIÓN DE LA CLASE PEDIDO REALIZADA EN EL APARTADO ANTERIOR DEL EJERCICIO. SI NO SE AÑADE LA PROGRAMACIÓN DE LA CLASE, SERÁ IMPOSIBLE CREAR EL OBJETO *PED* QUE NOS PERMITE MANEJAR LAS CARACTERÍSTICAS DEL PEDIDO.

F d ió Códi Lib D i i

PROGRAMACIÓN

JAVA

POO. HERENCIA

EJERCICIO 1

Interesa crear una etiqueta propia cuya función sea la de mostrar temperaturas.

Este tipo de etiqueta se llamará **EtiquetaTemperatura** y tendrá las siguientes características:

Propiedades

temperatura – double

Métodos

setTemperatura()

Este método recibe como parámetro un double con la temperatura a mostrar. Esta temperatura se almacena en la propiedad *temperatura*.

Además, este método muestra la temperatura en la etiqueta, añadiendo ºC. Por ejemplo, si la temperatura asignada fuera 10, entonces en la etiqueta aparecería:

10 ºC

getTemperatura()

Este método devuelve un double con la temperatura actual.

mostrarRangoColor()

Este método asignará un color a la etiqueta según la temperatura que contenga. Aquí está la escala de colores a usar:

< 0	Azul
>= 0 y < 10	Cyan
>= 10 y < 25	Magenta
>= 25 y < 35	Naranja
>= 35	Rojo

Si la etiqueta no contuviera un valor numérico válido, entonces se mostrará transparente (es decir, *setOpaque(false)*)

cambiarTemperatura()

Este método recibe un valor double como parámetro. Si este valor es positivo, entonces la temperatura aumenta en este valor. Si el valor es negativo, la temperatura disminuye.

Un ejemplo de uso de una etiqueta de este tipo:

```
etiTemp.setTemperatura(12);  //en la etiqueta se mostrará 12 ºC
etiTemp.mostrarRangoColor(); //la etiqueta se mostrará con color
                             //de fondo Magenta
etiTemp.cambiarTemperatura(20); //la temperatura sube 20 ºC
etiTemp.mostrarRangoColor(); //la etiqueta se muestra naranja
double t = etiTemp.getTemperatura(); //t contendrá un 32
```

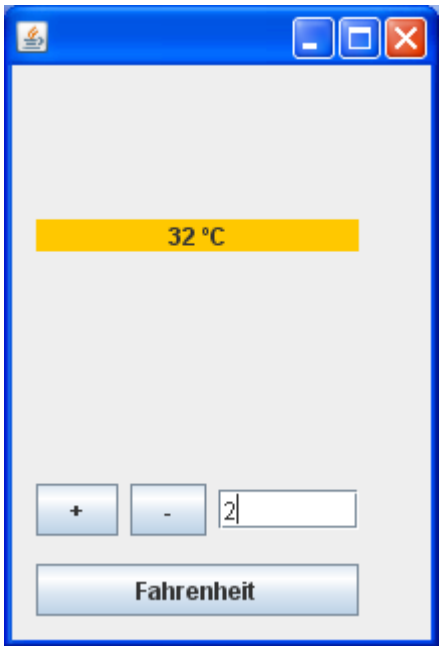
Objetivo del ejercicio

Debe programar la clase *EtiquetaTemperatura* de forma que herede de las etiquetas normales de java (JLabel)

Tendrá que añadir la propiedad *temperatura* a la clase *EtiquetaTemperatura* y programar los métodos antes comentados.

EJERCICIO 2

Realiza un proyecto cuya ventana principal tenga el siguiente aspecto:



En la parte superior añadirá (a través de código) una etiqueta del tipo *EtiquetaTemperatura*, que estará inicializada a 0 ºC

En la parte inferior añadirá tres botones y un cuadro de texto (esto lo puede hacer desde la ventana de diseño). Cada vez que se pulse el botón +, la temperatura de la etiqueta aumentará en la cantidad de grados indicados en el cuadro de texto. Y cuando se pulse el botón -, la temperatura disminuirá.

Cada vez que varíe la temperatura de la etiqueta, deberá variar su color.

Al pulsar el botón "Fahrenheit", aparecerá en un JOptionPane la temperatura que tiene actualmente la etiqueta convertida a grados Fahrenheit.

F d ió Códi Lib D i i

PROGRAMACIÓN

JAVA

POO. DIALOGOS PROPIOS

Planteamiento Inicial

Todos los proyectos que hacemos en el trabajo tienen que llevar un cuadro de diálogo de presentación que contenga:

(1) El nombre del programa.
(2) La versión.
(3) El nombre del programador.

Resulta un rollo tener que programar el cuadro de presentación cada vez que nos encargan un nuevo proyecto, así que programaremos una Clase Propia que derive de la clase JDialog, y que represente un cuadro de diálogo que contenga el nombre del programa, la versión y el programador.

Gracias a esta clase que vamos a programar, no tendremos que perder tiempo cada vez que necesitemos añadir una presentación a los futuros proyectos que hagamos en la empresa.

Clase DialogoPresentacion

Se programará una clase propia que se llamará *DialogoPresentacion*. Esta clase será un cuadro de diálogo, o, dicho de otra forma, heredará de la clase JDialog.

El aspecto de un cuadro de diálogo de este tipo en funcionamiento podría ser el siguiente:

Las tres etiquetas con bordes se llaman respectivamente: *etiNombrePrograma, etiVersion, etiNombreProgramador*.

El botón se llamará *btnEmpezar*.

Todos estos elementos están dentro de un JPanel al que se le ha asignado un borde con relieve.

Métodos de la clase DialogoPresentacion

Esta clase tendrá los siguientes métodos:

setNombrePrograma

(4) Recibirá una cadena con el nombre del programa, y esta cadena se introducirá en la etiqueta *etiNombrePrograma*.

setVersion

(5) Recibirá una cadena con la versión del programa, y esta cadena se introducirá en la etiqueta *etiVersion*.

setNombreProgramador

(6) Recibirá una cadena con el nombre del programador, y esta cadena se introducirá en la etiqueta *etiNombreProgramador*.

Cuando el usuario pulse el botón “Empezar”, lo único que tiene que suceder es que se cierre el cuadro de diálogo.

Proyecto Ejemplo

Debes crear un programa simple que permita calcular el área de un círculo.

Este proyecto debe incluir una presentación que aparezca al principio. Esta pantalla de presentación debe contener los siguientes datos:

Nombre del programa: Cálculo del Área del Círculo
Versión: v 1.0
Nombre del programador: *su nombre*.

Tendrá por supuesto, que incluir la clase *DialogoPresentacion* en su proyecto, y crear un objeto del tipo *DialogoPresentacion*. Debe asignar a este objeto los tres datos: nombre del programa, versión y programador. Y hacer que aparezca la presentación cuando se ejecute el programa.

Pista. La aparición de la presentación se programará en el evento *windowOpened* de la ventana principal.

PROGRAMACIÓN

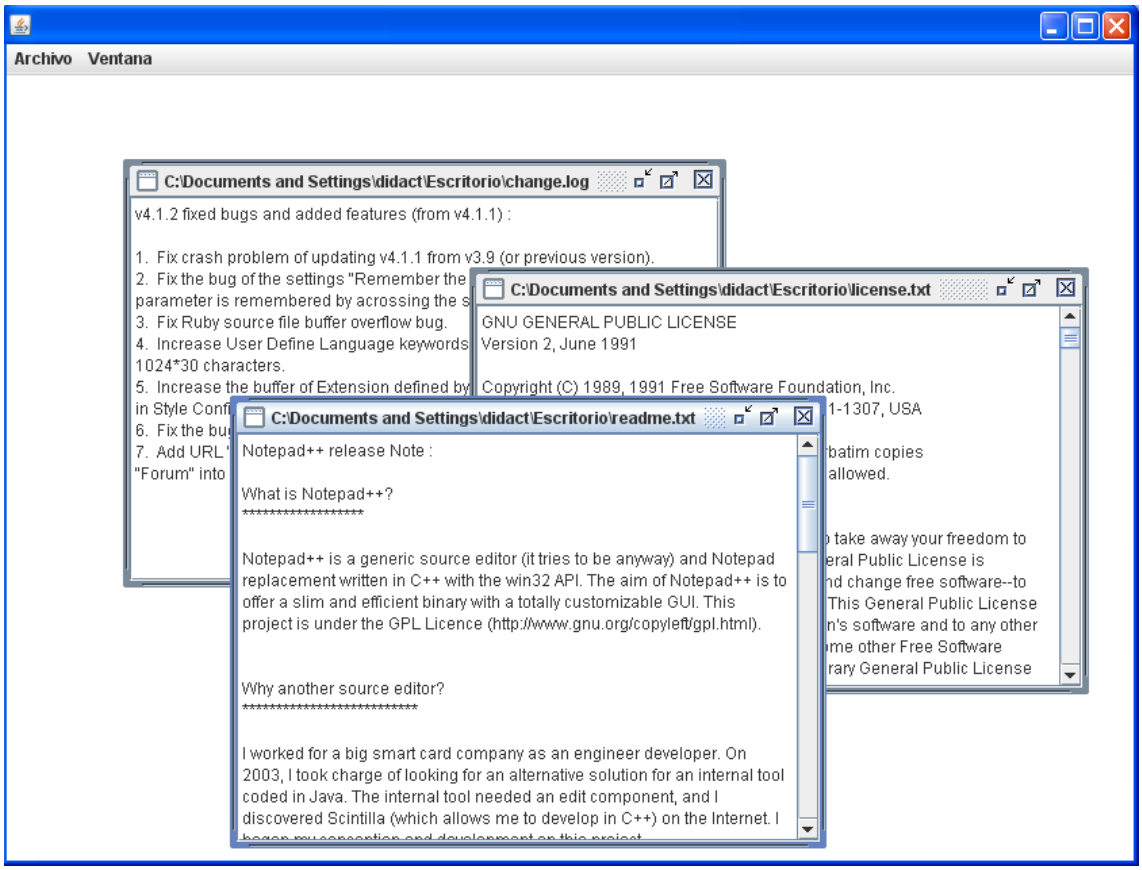
JAVA

PROGRAMACIÓN MDI

Bloc de Notas MDI

Planteamiento Inicial

Se pretende realizar un programa capaz de abrir ficheros de texto (.txt). Este programa será MDI, es decir, será capaz de abrir varios ficheros, mostrando sus contenidos en distintas ventanas.



Ventana Principal

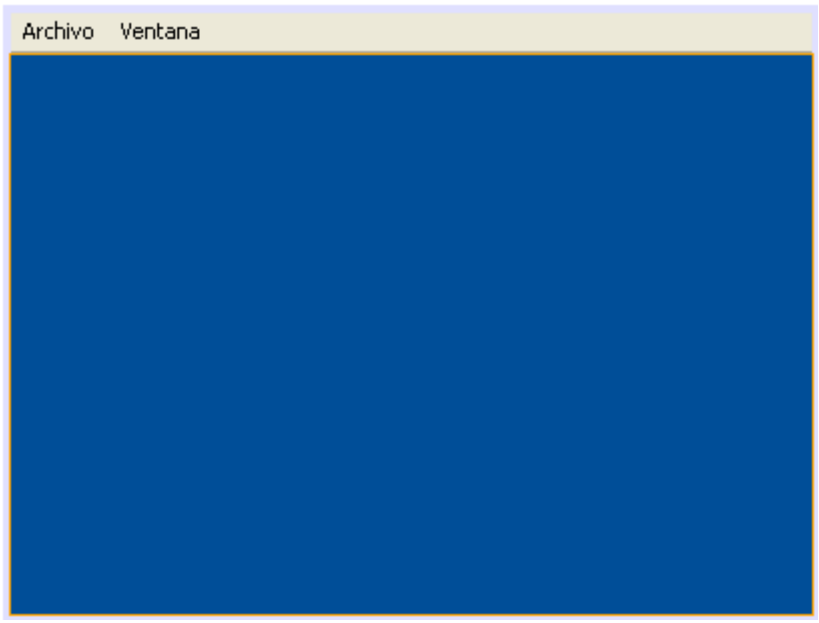
La ventana principal del programa constará de un menú con las siguientes opciones:

Archivo
- *Abrir*
- *Cerrar*
- *Cerrar Todos*
- *Salir*

Ventana
- *Cascada*
- *Mosaico Horizontal*
- *Mosaico Vertical*

En el centro de la ventana (se recomienda un Layout del tipo *BorderLayout*) se colocará un panel del tipo JDesktopPane que será el encargado de contener las ventanas internas. Puede llamar a este panel *panelInterno*.

Aspecto de la ventana principal:



panelInterno
(JDesktopPane)

Ventana Interna

Debe añadir a su proyecto una clase JInternalFrame para diseñar las ventanas internas.

Una ventana interna constará únicamente de un objeto JTextPane que ocupará toda la ventana.

Los objetos JTextPane son similares a los cuadros de texto (JTextField) con la diferencia de que pueden contener grandes cantidades de texto en varias líneas. Llame al objeto JTextPane con el nombre *txtTexto*.

F d ió Códi Lib D i i

txtTexto
(JTextPane)

Estas ventanas internas contendrán el texto del fichero que se abra. Para facilitar la tarea de abrir un fichero y colocar su texto en la ventana interna debe añadir el siguiente método a la clase ventana interna:

```
public void ponerTexto(String caminofichero) {
        try {
            File fi = new File(caminofichero);
            FileReader lectura = new FileReader(fi);
            BufferedReader entrada = new BufferedReader(lectura);
            String linea;
            String texto="";
            linea = entrada.readLine();
            while(linea!=null) {
                texto = texto+linea+"\n";
                linea = entrada.readLine();
            }
            entrada.close();
            lectura.close();
            txtTexto.setText(texto);
        } catch(Exception e) {
            JOptionPane.showMessageDialog(null,"Error al leer fichero.");
        }
    }
```

Este método recibe el camino de un fichero, y coloca dentro del JTextPane (el panel de texto) el texto contenido en el fichero. Aunque no es necesario entender el código de este método, en la parte final del enunciado se da una explicación de su funcionamiento.

F d ió Códi Lib D i i

Opciones principales del programa

Se comenta a continuación la forma en que deben funcionar las distintas opciones del programa:

Opción Archivo – Abrir
Esta opción servirá para abrir un fichero .txt y mostrar su contenido en una ventana interna. Para ello, tendrá que mostrar un JFileChooser para abrir un fichero.

Si el usuario selecciona un fichero, entonces tendrá que crear una ventana interna llamada por ejemplo *vi*, y mostrarla en el panel Interno.

Finalmente, usando el método ponerTexto de la ventana interna *vi* deberá mostrar el contenido del fichero seleccionado en el JFileChooser.

```
vi.ponerTexto(caminoficheroseleccionado)
```

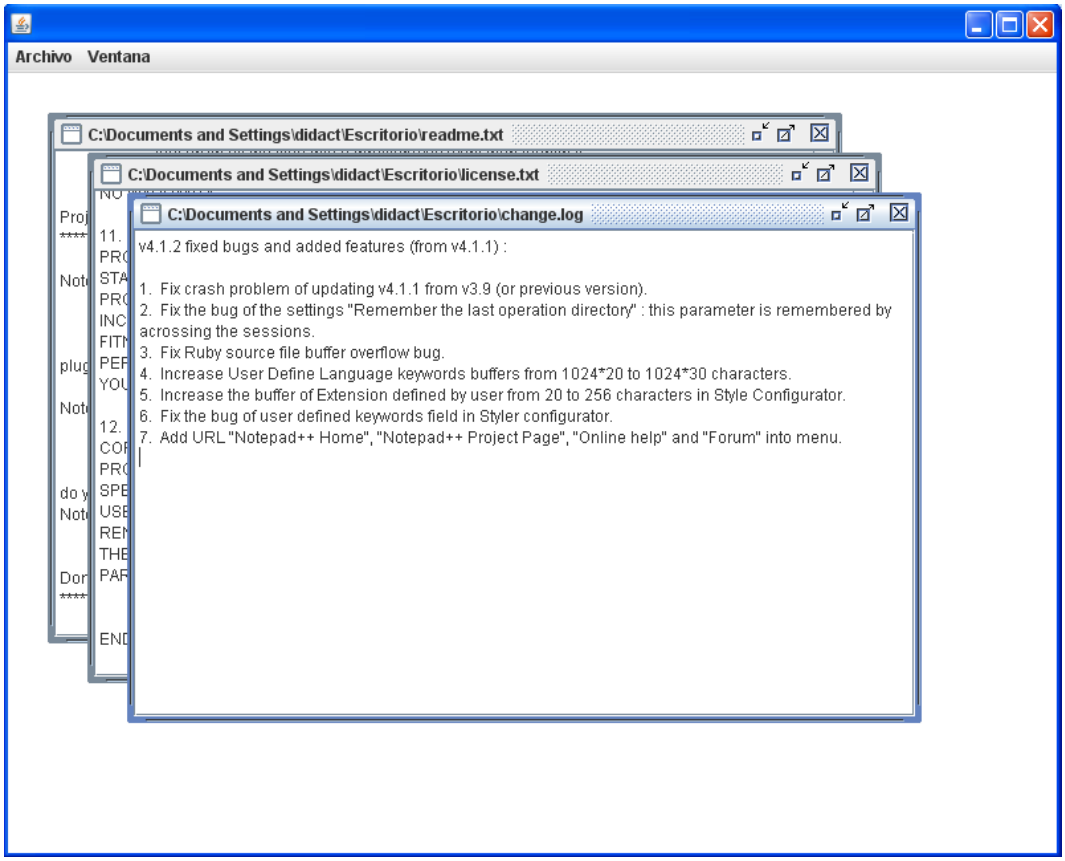
Al abrir un fichero de texto en una ventana, deberá aparecer el camino del fichero en la barra de título de la ventana interna.

Opción Archivo – Cerrar
La opción Cerrar cerrará la ventana interna que esté activa (si es que hay alguna)

Opción Archivo – Cerrar Todas
Esta opción cerrará todas las ventanas internas abiertas en el panel interno.

Opción Archivo – Salir
Permitirá salir del programa.

Opción Ventana – Cascada
Esta opción colocará todas las ventanas internas que haya abiertas en cascada. Esto es, una encima de otra, mostrando el título de cada una.

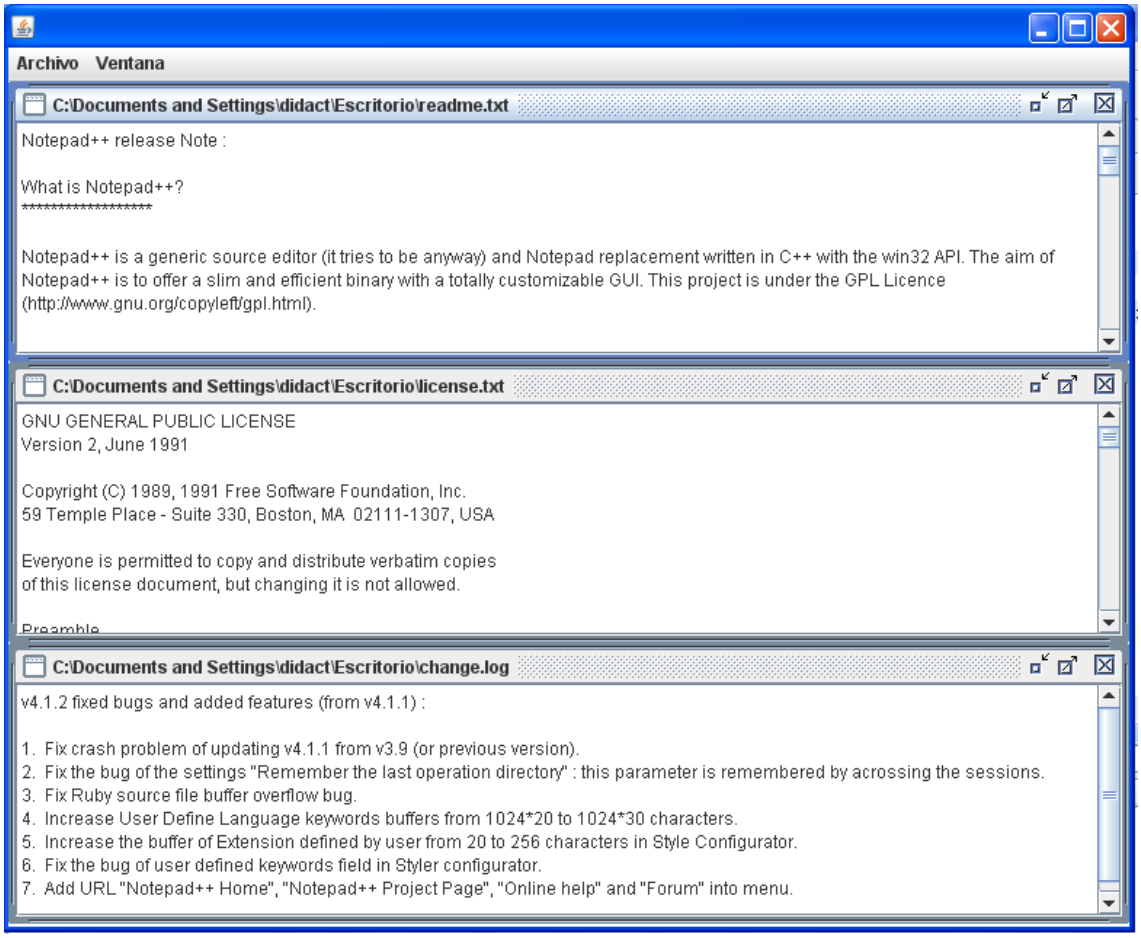


Opción Ventana – Mosaico Horizontal
Esta opción hará visible todas las ventanas mostrándolas en horizontal, ocupando todo el panel interno.

Para ello, tendrás que obtener el tamaño vertical del panel interno, y dividirlo por el número de ventanas abiertas en él. Esa cantidad será la altura de cada ventana.

El ancho de cada ventana será el ancho del panel interno.

Una vez calculado el ancho y alto de cada ventana debes colocar cada ventana una debajo de otra.



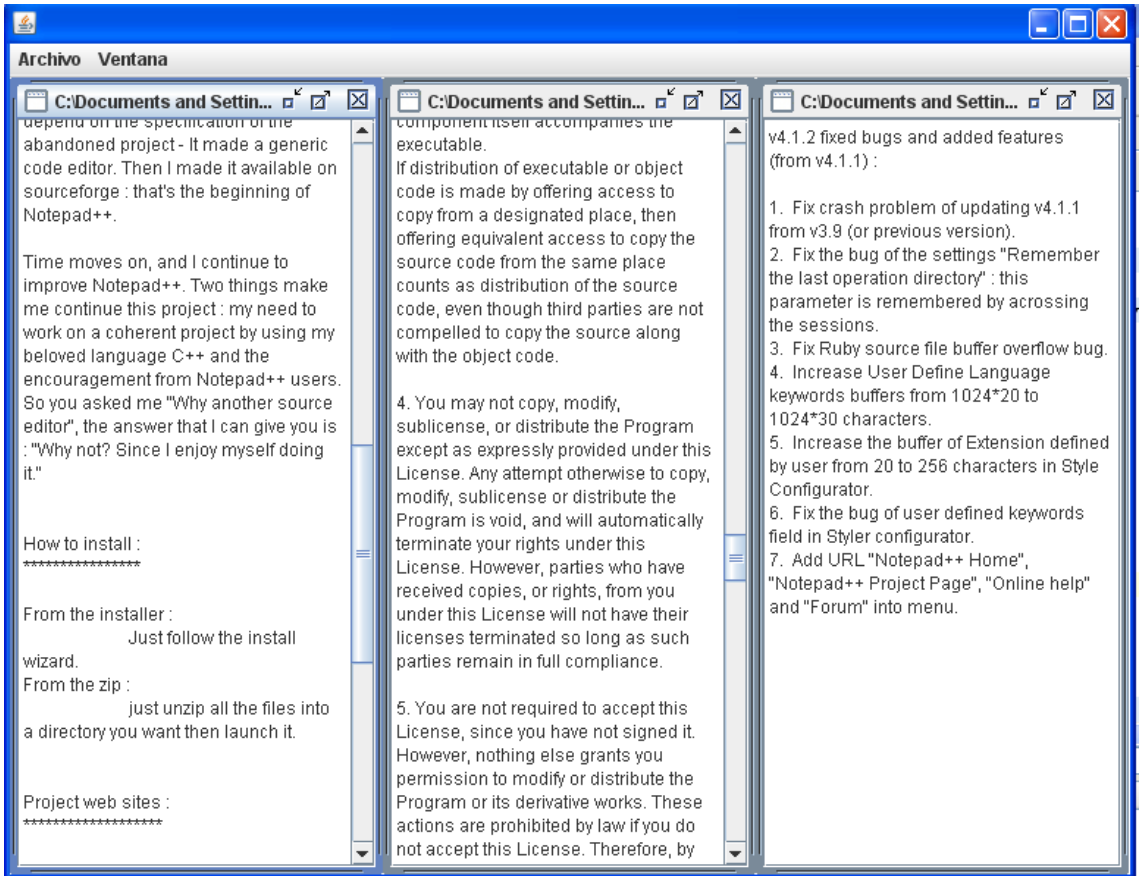
F d ió Códi Lib D i i

Opción Ventana – Mosaico Vertical
Esta opción hará visible todas las ventanas mostrándolas en vertical, ocupando todo el panel interno.

Para ello, tendrás que obtener el tamaño horizontal del panel interno, y dividirlo por el número de ventanas internas abiertas en él. Esa cantidad será el ancho de cada ventana.

Al alto de cada ventana será el alto del panel interno.

Una vez calculado el ancho y alto de cada ventana debes colocar cada ventana una al lado de otra.



EXPLICACIÓN MÉTODO ponerTexto

Para realizar este ejercicio es necesario añadir a la clase de la ventana interna el siguiente método. No es necesario entender el código de este procedimiento para hacer el ejercicio, pero en cualquier caso, aquí hay una explicación de este:

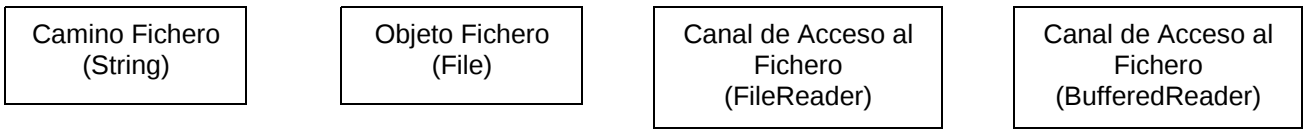
```
public void ponerTexto(String caminofichero) {
        try {
            File fi = new File(caminofichero);
            FileReader lectura = new FileReader(fi);
            BufferedReader entrada = new BufferedReader(lectura);
            String linea;
            String texto="";
            linea = entrada.readLine();
            while(linea!=null) {
                texto = texto+linea+"\n";
                linea = entrada.readLine();
            }
            entrada.close();
            lectura.close();
            txtTexto.setText(texto);
        } catch(Exception e) {
            JOptionPane.showMessageDialog(null,"Error al leer fichero.");
        }
    }
```

Este método recibe un camino de fichero de texto y muestra dentro de un JTextPane el texto que contiene dicho fichero.

La forma de acceder a un fichero para extraer su contenido es a través de “canales”. Los canales son objetos que se “enganchan” al fichero uno detrás de otro. Cada canal ofrece una serie de posibilidades al programador.

Para los ficheros de texto, la forma de acceder a ellos es la siguiente:

- Se crea un objeto File a partir del camino del fichero (El camino del fichero es un String)
- Se engancha al objeto File un objeto del tipo FileReader. (Los objetos FileReader son canales de lectura, que permiten extraer información del fichero)
- Se engancha al canal FileReader un objeto BufferedReader. (Los objetos BufferedReader dan facilidad al programador para extraer el texto del fichero)
- Finalmente, a través del método *readLine* del objeto BufferedReader se pueden extraer una a una las líneas de texto del fichero.
- Una vez finalizado el trabajo con el fichero se cierra el canal FileReader y el canal BufferedReader.



La creación de un objeto File a partir del camino del fichero se hace con la siguiente instrucción:

```
File fi = new File(caminofichero);
```

La creación de un canal FileReader y la conexión con el fichero anterior se hace a través de la siguiente instrucción:

F d ió Códi Lib D i i

```
FileReader lectura = new FileReader(fi);
```

La creación de un canal BufferedReader y la conexión con el canal FileReader anterior se hace a través de la siguiente instrucción:

```
BufferedReader entrada = new BufferedReader(lectura);
```

Una vez hecho esto, tenemos un objeto llamado *entrada* del tipo BufferedReader que posee un método llamado *readLine* que permite leer una línea de texto del fichero. A través de este método vamos leyendo líneas de texto del fichero.

Hay que tener en cuenta que si al usar *readLine* obtenemos el valor null, es que ya no hay más texto en el fichero.

Todo esto se aprovecha para crear una cadena larga con el contenido del fichero:

```
String linea;
String texto="";
linea = entrada.readLine();
while(linea!=null) {
    texto = texto+linea+"\n";
    linea = entrada.readLine();
}
```

El código anterior crea una variable *texto* que contiene el texto completo del fichero. Este código extrae una a una las líneas de texto del fichero y las va concatenando en la variable *texto*.

Como hemos terminado con la manipulación del fichero, es bueno cerrar los canales de comunicación con el fichero (el canal FileReader y el canal BufferedReader creados):

```
entrada.close();
lectura.close();
```

Y finalmente metemos el texto extraído en el panel de texto de la ventana, el cual se llama txtTexto:

```
txtTexto.setText(texto);
```

Todo esto debe estar dentro de un try ... catch ya que es un código susceptible de tener errores de ejecución (es decir, de lanzar excepciones)

PROGRAMACIÓN

JAVA - BASES DE DATOS

MODELO E-R

SUPUESTO Nº 1. "CONTROL DE VENTAS"

Una empresa necesita un programa para controlar las ventas que realiza diariamente.

La empresa tiene una lista de clientes cuyos datos quiere controlar. Los datos que la empresa quiere guardar de cada cliente es : el CIF, el nombre, la ciudad y un teléfono de contacto.

La empresa tiene una serie de comerciales que son los que realizan las ventas. La empresa quiere controlar la información de sus comerciales. Concretamente necesita almacenar de ellos sus nombres y apellidos y su móvil. A cada comercial se le asigna un número en la empresa para distinguirlos.

Diariamente, los comerciales realizan ventas de productos a los clientes. Interesa almacenar información sobre dichas ventas. De cada venta interesa almacenar el nombre del producto que se vende, las unidades vendidas del artículo, el precio del producto y la fecha en que se efectuó la venta. También interesa saber la forma de pago.

Debes tener en cuenta también la siguiente información :

Un comercial realiza ventas. Una venta es realizada por un solo comercial.

Una venta en concreto se realiza a un solo cliente. Por otro lado, un cliente comprará muchas cosas a la empresa. Es decir, se le pueden hacer muchas ventas a un cliente.

SE PIDE :

- Identifique las entidades que participan, teniendo en cuenta el enunciado.
- Identifique los atributos de cada entidad.
- Indique el atributo clave para cada entidad. Si fuera necesario, añada un atributo clave a la entidad.
- Indique las relaciones que existen entre las entidades.
- Indique el tipo de relaciones existentes (Nota : no existen relaciones Muchas a Muchas)
- Averigue las claves foráneas según las relaciones existentes.
- Realice el gráfico correspondiente al Modelo E-R.

SUPUESTO Nº 2. « TRANSPORTES »

Una empresa cuenta con una serie de camiones que usa para transportar diversos productos. La empresa necesita un programa que gestione toda la información producida por esta actividad.

Concretamente, la empresa necesita almacenar información sobre los camiones que posee. Necesita almacenar la matrícula de cada camión, la marca y el modelo y el año de compra.

Por otro lado, la empresa tiene una serie de conductores. De cada conductor se quiere almacenar el nombre, los apellidos, el móvil y el sueldo que gana. A cada conductor se le asigna un número en la empresa. No hay dos conductores con dos números iguales.

La empresa quiere controlar la información correspondiente a cada transporte que se efectúa. Concretamente quiere saber la ciudad de origen del transporte, la ciudad de destino, el material que se transporta, cuantos kilos se transporta y la fecha del transporte.

Cada transporte concreto está encargado por un cliente. A la empresa le interesa tener información de los clientes que encargan transportes. La información que interesa almacenar de cada cliente es : el CIF del cliente, el nombre de la empresa, la dirección de la empresa, el teléfono de la empresa.

Hay que tener en cuenta también la siguiente información :

En un transporte en concreto solo participa un camión. Ahora bien, un camión se usa en muchos transportes.

En cada transporte participa un solo conductor, y, por supuesto, un conductor realiza muchos transportes mientras trabaja para la empresa.

Cada transporte está encargado por un solo cliente. Pero ten en cuenta que un cliente puede encargar muchos transportes.

SE PIDE :

1. Identifique las entidades que participan, teniendo en cuenta el enunciado.
2. Identifique los atributos de cada entidad.
3. Indique el atributo clave para cada entidad. Si fuera necesario, añada un atributo clave a la entidad.
4. Indique las relaciones que existen entre las entidades.
5. Indique el tipo de relaciones existentes (Nota : no existen relaciones Muchas a Muchas)
6. Averigue las claves foráneas según las relaciones existentes.
7. Realice el gráfico correspondiente al Modelo E-R.

SUPUESTO Nº 3. « MEDICIONES »

Una empresa química realiza mediciones constantemente de los materiales que contiene la tierra de distintas parcelas de cultivo, y le interesaría tener un control de toda la información que esto genera. Para ello le encargan un programa para gestionar esta información.

A esta empresa le interesaría guardar información sobre cada medición que se hace. Interesa almacenar lo siguiente: fecha en que se hace la medición, hora, temperatura ambiente en el momento de hacer la medición, humedad del aire y un comentario sobre la medición (en este comentario se indicará el resultado de la medición) Cada medición se numera, de forma que no puede haber dos mediciones con el mismo número.

Las mediciones las realizan químicos especializados contratados por la empresa. Esta empresa desea almacenar los datos de estos químicos: nombre, apellidos y teléfono de contacto.

Las mediciones se realizan en terrenos particulares, e interesa almacenar información sobre dichos terrenos. Concretamente interesa almacenar: el número de hectáreas del terreno, nombre del propietario, DNI del propietario, teléfono de contacto del propietario, dirección del terreno. A cada terreno se le asignará un código único para distinguirlo de los demás.

Las mediciones se plasman finalmente en informes recopilatorios que se envían a la empresa que ha encargado las medidas. Interesa almacenar de cada informe lo siguiente: número de informe (no hay dos informes con el mismo número), nombre de la empresa que encarga el informe, fecha en que se lanza el informe, conclusiones del informe (aquí se redactan una serie de conclusiones finales deducidas a partir de las distintas mediciones)

Debe tener en cuenta también lo siguiente:

Un químico realiza muchas mediciones, pero una medición la realiza solo un químico.

En un mismo terreno se pueden realizar diversas mediciones, pero tenga en cuenta que una medición está realizada en un solo terreno.

Un informe de mediciones plasmará el resultado de muchas mediciones, pero una medición aparecerá en un solo informe.

SE PIDE :

1. Identifique las entidades que participan, teniendo en cuenta el enunciado.
2. Identifique los atributos de cada entidad.
3. Indique el atributo clave para cada entidad. Si fuera necesario, añada un atributo clave a la entidad.
4. Indique las relaciones que existen entre las entidades.
5. Indique el tipo de relaciones existentes (Nota : no existen relaciones Muchas a Muchas)
6. Averigue las claves foráneas según las relaciones existentes.
7. Realice el gráfico correspondiente al Modelo E-R.

F d ió Códi Lib D i i

SUPUESTO Nº4. “REVISIONES MÉDICAS”

En un hospital se realizan distintas revisiones médicas a los pacientes ingresados, y les interesa almacenar información sobre estas revisiones.

De cada revisión interesa almacenar la fecha y hora en que se hizo, peso y altura del paciente y otros resultados de dicha revisión. Cada revisión se numera de forma que no haya dos revisiones con el mismo número.

Cada revisión se realiza a un solo paciente, aunque a un paciente se le pueden hacer varias revisiones mientras está ingresado.

Interesa almacenar la información de cada paciente: nombre, apellidos, DNI y fecha de nacimiento del paciente.

Los médicos realizan revisiones a sus pacientes. Hay que tener en cuenta que un mismo médico puede realizar revisiones a distintos pacientes, pero un paciente tiene asignado un solo médico para las revisiones.

Interesa almacenar la información de cada médico: nombre, apellidos, especialidad. A cada médico se le asigna un código que es único.

SE PIDE :

(1) Identifique las entidades que participan, teniendo en cuenta el enunciado.
(2) Identifique los atributos de cada entidad.
(3) Indique el atributo clave para cada entidad. Si fuera necesario, añada un atributo clave a la entidad.
(4) Indique las relaciones que existen entre las entidades.
(5) Indique el tipo de relaciones existentes (Nota : no existen relaciones Muchas a Muchas)
(6) Averigue las claves foráneas según las relaciones existentes.
(7) Realice el gráfico correspondiente al Modelo E-R.

F d ió Códi Lib D i i

PROGRAMACIÓN

JAVA - BASES DE DATOS

TRASPASO A TABLAS

SUPUESTO Nº 1. "CONTROL DE VENTAS"

A partir del Modelo Entidad-Relación correspondiente a este supuesto de la hoja anterior, realice el traspaso a tablas de modelo.

Introduzca luego algunos datos en las tablas. Invéntese los datos, pero procure que tengan cierto sentido. Tenga en cuenta que los datos correspondientes a las claves y claves foráneas tendrán que coincidir para que la base de datos tenga coherencia.

SUPUESTO Nº 2. « TRANSPORTES »

A partir del Modelo Entidad-Relación correspondiente a este supuesto de la hoja anterior, realice el traspaso a tablas de modelo.

Introduzca luego algunos datos en las tablas. Invéntese los datos, pero procure que tengan cierto sentido. Tenga en cuenta que los datos correspondientes a las claves y claves foráneas tendrán que coincidir para que la base de datos tenga coherencia.

SUPUESTO Nº 3. « MEDICIONES »

A partir del Modelo Entidad-Relación correspondiente a este supuesto de la hoja anterior, realice el traspaso a tablas de modelo.

Introduzca luego algunos datos en las tablas. Invéntese los datos, pero procure que tengan cierto sentido. Tenga en cuenta que los datos correspondientes a las claves y claves foráneas tendrán que coincidir para que la base de datos tenga coherencia.

SUPUESTO Nº4. “REVISIONES MÉDICAS”

A partir del Modelo Entidad-Relación correspondiente a este supuesto de la hoja anterior, realice el traspaso a tablas de modelo.

Introduzca luego algunos datos en las tablas. Invéntese los datos, pero procure que tengan cierto sentido. Tenga en cuenta que los datos correspondientes a las claves y claves foráneas tendrán que coincidir para que la base de datos tenga coherencia.

PROGRAMACIÓN

JAVA - BASES DE DATOS

RELACIONES MUCHAS A MUCHAS

SUPUESTO Nº 1 « LIMUSINAS »

Una empresa de alquiler de limusinas quiere gestionar la siguiente información :

Por un lado se quiere almacenar la información de los clientes que solicitan el uso de una limusina. Se almacenará el nombre, apellidos, DNI y teléfono del cliente.

Por otro lado se quiere almacenar la información de las distinas limusinas propiedad de la empresa. Se almacenará la matrícula, la marca y el modelo.

Una limusina es alquilada por muchos clientes (en momentos distintos claro está, un día la alquila un cliente y otro día la alquila otro)

Por otro lado, un cliente alquila muchas limusinas (en momentos distintos también, un día alquila una limusina, y a la semana siguiente puede alquilar otra, etc)

Interesa saber la fecha en que un cliente alquila una limusina, y el nombre del chófer que condujo en el viaje.

SE PIDE :

(8) Este supuesto produce una relación Muchas a Muchas entre las entidades limusinas y clientes. Represente dicha relación.
(9) A continuación identifique la entidad intermedia que elimina la relación Muchas a Muchas.
(10) Añada los atributos que considere necesarios a la entidad intermedia.
(11) Realice el Modelo Entidad Relación completo del supuesto teniendo en cuenta la nueva entidad introducida y sus atributos.
(12) Realice el traspaso a tabla de dicho modelo.

SUPUESTO Nº 2 « RESERVAS HOTEL »

Un hotel quiere guardar un histórico con todas las reservas realizadas en el hotel por sus clientes. Concretamente se quiere guardar lo siguiente :

Información sobre sus clientes : nombre del cliente, apellidos, DNI y país.

Información sobre sus habitaciones : número de la habitación, número de camas, precio de la habitación por noche.

Un cliente puede haber reservado varias habitaciones (no tiene por qué ser a la vez : un verano puede alojarse en la habitación nº 124, pero al verano siguiente puede alojarse en la habitación nº 535 del mismo hotel)

En una habitación se alojan muchos clientes (no a la vez, claro está : una semana está alojado el cliente *Juan González* y a la semana siguiente está alojado el cliente *Ana Pérez*)

Interesaría almacenar también como se ha pagado la reserva de la habitación, y la fecha de entrada y de salida del cliente en la habitación.

SE PIDE :

(13) Este supuesto produce una relación Muchas a Muchas entre las entidades habitaciones y clientes. Represente dicha relación.
(14) A continuación identifique la entidad intermedia que elimina la relación Muchas a Muchas.
(15) Añada los atributos que considere necesarios a la entidad intermedia.
(16) Realice el Modelo Entidad Relación completo del supuesto teniendo en cuenta la nueva entidad introducida y sus atributos.
(17) Realice el traspaso a tabla de dicho modelo.

F d ió Códi Lib D i i

SUPUESTO Nº 3 « TALLER MECÁNICO »

Un taller mecánico quiere guardar información sobre todas las reparaciones que se realizan en él. Para ello se quiere guardar información sobre lo siguiente :

Mecánicos : Interesa almacenar la información del mecánico que realiza la reparación. Se almacenará el nombre del mecánico, apellidos, móvil y un código que identifica a cada mecánico.

Coches : Interesa almacenar la información de los coches que han sido reparados en el taller. Se almacenará la matrícula, marca y modelo del coche.

Debes tener en cuenta que un mecánico repara muchos coches (no a la vez, se supone, primero uno y luego otro)

Por otro lado, un coche puede ser reparado por varios mecánicos (es de suponer que un coche puede sufrir varias averías a lo largo de su vida, y cada una de estas avería puede ser reparada por un mecánico distinto)

Interesa almacenar también el coste de cada reparación, así como el número de horas que se usaron para hacer la reparación.

SE PIDE :

(16) Este supuesto produce una relación Muchas a Muchas. Represente dicha relación.
(17) A continuación identifique la entidad intermedia que elimina la relación Muchas a Muchas.
(18) Añada los atributos que considere necesarios a la entidad intermedia.
(19) Realice el Modelo Entidad Relación completo del supuesto teniendo en cuenta la nueva entidad introducida y sus atributos.
(20) Realice el traspaso a tabla de dicho modelo.

SUPUESTO Nº 4 « VIDEOCLUB »

Un videoclub quiere almacenar información sobre los alquileres de películas que se hicieron. Le interesa almacenar información sobre las películas y sobre sus socios :

Películas : se almacenará el número del DVD (no hay dos DVD con el mismo número), el título de la película que contiene, el nombre del director y el tipo de película.

Socios : se almacenará el número de socio (no hay dos socios con el mismo número), el nombre y apellidos del socio, su teléfono y dirección.

Hay que tener en cuenta que una película se alquila a muchos socios (No a la vez, claro está : el DVD es alquilado a un socio, y cuando este lo devuelve, se vuelve a alquilar a otro, y así sucesivamente)

Por otro lado, un socio alquila muchas películas (No a la vez, claro está : primero alquila una, y al día siguiente alquila otra, y así sucesivamente)

Interesaría almacenar también la fecha en que produce cada alquiler y lo que pagó el socio por el alquiler.

SE PIDE :

(36) Este supuesto produce una relación Muchas a Muchas. Represente dicha relación.
(37) A continuación identifique la entidad intermedia que elimina la relación Muchas a Muchas.
(38) Añada los atributos que considere necesarios a la entidad intermedia.
(39) Realice el Modelo Entidad Relación completo del supuesto teniendo en cuenta la nueva entidad introducida y sus atributos.
(40) Realice el traspaso a tabla de dicho modelo.

F d ió Códi Lib D i i

PROGRAMACIÓN

JAVA - BASES DE DATOS - ACCESS

CREACIÓN DE BASES DE DATOS EN ACCESS

En las hojas de ejercicios anteriores ha diseñado las bases de datos para distintos supuestos. Ha realizado el Modelo Entidad – Relación y luego ha realizado el traspaso a Tablas.

Ahora se pide que cree las bases de datos correspondientes a dichos traspasos a Tablas en Access para los siguientes supuestos de hojas anteriores:

SUPUESTO "CONTROL DE VENTAS"

Crear un fichero de base de datos llamado CONTROLVEN usando el programa Access. Introducir en él las tablas diseñadas para este supuesto. Introducir algunos datos en las tablas (Intente que los datos sean coherentes)

SUPUESTO « TRANSPORTES »

Crear un fichero de base de datos llamado TRANSPORTES usando el programa Access. Introducir en él las tablas diseñadas para este supuesto. Introducir algunos datos en las tablas (Intente que los datos sean coherentes)

SUPUESTO « MEDICIONES »

Crear un fichero de base de datos llamado MEDICIONES usando el programa Access. Introducir en él las tablas diseñadas para este supuesto. Introducir algunos datos en las tablas (Intente que los datos sean coherentes)

SUPUESTO « RESERVAS HOTEL »

Crear un fichero de base de datos llamado HOTEL usando el programa Access. Introducir en él las tablas diseñadas para este supuesto. Introducir algunos datos en las tablas (Intente que los datos sean coherentes)

SUPUESTO « TALLER MECÁNICO »

Crear un fichero de base de datos llamado TALLER usando el programa Access. Introducir en él las tablas diseñadas para este supuesto. Introducir algunos datos en las tablas (Intente que los datos sean coherentes)

PROGRAMACIÓN

JAVA - BASES DE DATOS – ACCESS – CONSULTAS

CREACIÓN DE CONSULTAS EN ACCESS

Entre en la base de datos MANEMPSA (usada en las hojas de ejercicios guiados) y añada registros a las tablas hasta que estas queden así:

Tabla Trabajadores

Trabajadores : Tabla

DNI	Nombre	Apellidos	Sueldo	Fecha	Matricula
12.321.567-B	Juan	Pérez	1120	04/05/2002	4433-ABB
21.123.123-A	Ana	Ruiz	1200	02/03/2002	3322-ASR
22.333.444-C	Francisco	López	1000	01/06/2006	1144-BBB
			0		

Tabla Coches

Coches : Tabla

Matricula	Marca	Modelo	Año	DNI
1144-BBB	CITROEN	C3	2005	22.333.444-C
3322-ASR	SEAT	Ibiza	2000	21.123.123-A
4433-ABB	CITROEN	Saxo	2001	12.321.567-B
			0	

Tabla Clientes

Clientes : Tabla

CIF	Nombre	Direccion	Tfno1	Tfno2
B11223212	Seguros Segasa	C/Ancha 2	956344334	629234323
B22334466	Academia La Plata	C/La Plata 10	956302323	
B33221111	Papelería Cuatro	C/Larga 8	956305060	
B44556666	Seguros Cruces	C/Lealas 5	956309020	633212342
B44557777	Academia de Pintura La Vid	C/Cerezas 7	956101010	600121212
B55112233	Papelería El Bolígrafo	Parque Atlántic		670787878
B55334433	Seguros La Paz	C/Larga 3	956202020	

Tabla Servicios

Sevicios : Tabla

Numero	Fecha	Tipo	Cantidad	Comentario	DNI	CIF
1	12/04/2004	Limpieza	300		21.123.123-A	B11223212
2	22/05/2005	Fontanería	238	Arreglo tuberías	12.321.567-B	B22334466
3	21/12/2005	Electricidad	130	Revisión cableado	21.123.123-A	B33221111
4	10/11/2006	Fontanería	250		12.321.567-B	B11223212
5	03/06/2006	Fontanería	214	Desatascos servici	21.123.123-A	B55112233
6	12/06/2006	Limpieza	265	Limpieza cristales	22.333.444-C	B55334433
7	20/07/2006	Limpieza	170		12.321.567-B	B44556666
8	01/08/2006	Electricidad	160	Recambio paneles	22.333.444-C	B44557777
9	05/08/2006	Limpieza	250	Limpieza fachada	12.321.567-B	B44556666
10	08/08/2006	Fontanería	265		12.321.567-B	B44557777
11	09/08/2006	Limpieza	139	Limpieza cristales	22.333.444-C	B44556666
(Autonumérico)			0			

Ahora, realice las siguientes consultas en la base de datos. Guarde cada una con el nombre que se indica. Se recomienda que compruebe cada consulta antes de guardarla.

(21) Crear una consulta llamada *Servicios de Limpieza*.

En ella deben aparecer los campos *fecha del servicio, tipo, cantidad y comentario* de aquellos servicios cuyo tipo sea *Limpieza*.

(22) Crear una consulta llamada *Servicios Baratos*.

En ella deben aparecer los campos *número del servicio, fecha, tipo y cantidad* de aquellos servicios que hayan costado menos de 180 euros.

(23) Crear una consulta llamada *Servicios anteriores 2006*.

En ella deben aparecer los campos *número del servicio, fecha, tipo y cantidad* de aquellos servicios que fueron realizados antes del 1 – 1 – 2006

(24) Crear una consulta llamada *Servicios de Fontanería*

En ella deben aparecer los campos *número de servicio, cantidad, tipo y comentario* de todos aquellos servicios que fueron de fontanería y costaron 250 o más euros.

(25) Crear una consulta llamada *Listado de Servicios No Limpieza*

En ella deben aparecer los campos *número de servicio, cantidad, tipo y comentario* de los servicios de Fontanería y los servicios de Electricidad.

(26) Crear una consulta llamada *Listado de Servicios de Electricidad*

En ella deben aparecer los campos *fecha del servicio, cantidad, tipo, comentario, nombre y apellidos del trabajador y nombre del cliente* de aquellos servicios que sean del tipo *Electricidad*.

Debes tener en cuenta que los campos *nombre y apellidos* del trabajador pertenecen a la tabla *Trabajadores*, mientras que el *nombre del cliente* pertenece a la tabla *Clientes*. Los demás campos pertenecen a la tabla *Servicios*.

F d ió Códi Lib D i i

(27) Crear una consulta llamada *Servicios realizados por Juan*

En ella deben aparecer los campos *fecha del servicio, cantidad, tipo, comentario, nombre del cliente y nombre y apellidos del trabajador* de todos aquellos servicios realizados por el trabajador con DNI 12.321.567-B

Ten en cuenta que tendrás que usar varias tablas para hacer la consulta.

(28) Crear una consulta llamada *Servicios a Academias*

En ella deben aparecer los campos *fecha del servicio, tipo, cantidad, nombre del cliente y nombre y apellidos del trabajador* de todos aquellos servicios que se hayan realizado a una Academia (es decir, el nombre del cliente debe contener la palabra "academia")

(29) Crear una consulta llamada *Servicios del año 2006*

En ella aparecerán los campos *fecha del servicio, tipo, cantidad, apellidos del trabajador*, *nombre del cliente, CIF del cliente* de todos aquellos servicios que se hayan realizado entre el 1 del 1 de 2006 y el 31 del 12 del 2006

(30) Crear una consulta llamada *Servicios en la calle Larga*

En ella aparecerán los campos *fecha del servicio, tipo, cantidad, nombre del cliente, dirección del cliente, DNI del trabajador* para todos aquellos servicios realizados en la calle *Larga*.

(31) Crear una consulta llamada *Servicios trabajadores 2006*

En ella deben aparecer los campos *fecha del servicio, tipo cantidad, nombre y apellidos del trabajador y fecha de entrada del trabajador* de todos aquellos servicios realizados por los trabajadores que entraron en la empresa a partir del 1 – 1 – 2006.

Para crear esta consulta tienes que usar las tablas Trabajadores y Servicios.

(32) Crear una consulta llamada *Clientes de Seguros*

En ella deben aparecer los campos *CIF, nombre del cliente, dirección del cliente* de todos aquellos clientes que gestionen seguros (deben contener en el nombre la palabra "seguros")

(33) Crear una consulta llamada *Listado de Academias y Papelerías*

En ella deben aparecer los campos *CIF, nombre del cliente, dirección del cliente y teléfono fijo* de todos aquellos clientes que sean academias o papelerías.

(34) Crear una consulta llamada *Listado de SEAT y trabajadores*

En ella deben aparecer los campos *matrícula, marca y modelo* de los seat de la empresa. También interesa que aparezca el *nombre y apellido* del conductor de cada coche.

(35) Crear una consulta llamada *Servicios realizados con CITROEN*

En ella debe aparecer el listado de servicios que han sido realizados usando alguno de los citroen de la empresa.

En esta consulta deben aparecer los siguientes campos: *matrícula del vehículo, marca y modelo*. *Nombre y apellidos del trabajador que hizo el servicio*. *Nombre y dirección del cliente al que se le hizo el servicio*. *Tipo de servicio y cantidad*.

En esta consulta participan todas las tablas de la base de datos.

PROGRAMACIÓN

JAVA - BASES DE DATOS – SQL – CONSULTAS

CREACIÓN DE CONSULTAS USANDO SQL

EJERCICIO 1

Las siguientes consultas son las realizadas en la hoja anterior. Se le pide al alumno que vuelva a hacer dichas consultas pero esta vez usando el lenguaje SQL (se le ha añadido a los nombres la palabra SQL para distinguirlas de las consultas de la hoja anterior)

(36) Crear una consulta llamada *Servicios de Limpieza SQL*.

En ella deben aparecer los campos *fecha del servicio, tipo, cantidad y comentario* de aquellos servicios cuyo tipo sea *Limpieza*.

(37) Crear una consulta llamada *Servicios Baratos SQL*.

En ella deben aparecer los campos *número del servicio, fecha, tipo y cantidad* de aquellos servicios que hayan costado menos de 180 euros.

(38) Crear una consulta llamada *Servicios anteriores 2006 SQL*.

En ella deben aparecer los campos *número del servicio, fecha, tipo y cantidad* de aquellos servicios que fueron realizados antes del 1 – 1 – 2006

(39) Crear una consulta llamada *Servicios de Fontanería SQL*

En ella deben aparecer los campos *número de servicio, cantidad, tipo y comentario* de todos aquellos servicios que fueron de fontanería y costaron 250 o más euros.

(40) Crear una consulta llamada *Listado de Servicios No Limpieza SQL*

En ella deben aparecer los campos *número de servicio, cantidad, tipo y comentario* de los servicios de Fontanería y los servicios de Electricidad.

(41) Crear una consulta llamada *Listado de Servicios de Electricidad SQL*

En ella deben aparecer los campos *fecha del servicio, cantidad, tipo, comentario, nombre y apellidos del trabajador y nombre del cliente* de aquellos servicios que sean del tipo *Electricidad*.

Debes tener en cuenta que los campos *nombre y apellidos* del trabajador pertenecen a la tabla *Trabajadores*, mientras que el *nombre del cliente* pertenece a la tabla *Clientes*. Los demás campos pertenecen a la tabla *Servicios*.

(42) Crear una consulta llamada *Servicios realizados por Juan SQL*

En ella deben aparecer los campos *fecha del servicio, cantidad, tipo, comentario, nombre del cliente y nombre y apellidos del trabajador* de todos aquellos servicios realizados por el trabajador con DNI 12.321.567-B

Ten en cuenta que tendrás que usar varias tablas para hacer la consulta.

(43) Crear una consulta llamada *Servicios a Academias SQL*

En ella deben aparecer los campos *fecha del servicio, tipo, cantidad, nombre del cliente y nombre y apellidos del trabajador* de todos aquellos servicios que se hayan realizado a una Academia (es decir, el nombre del cliente debe contener la palabra "academia")

(44) Crear una consulta llamada *Servicios del año 2006 SQL*

En ella aparecerán los campos *fecha del servicio, tipo, cantidad, apellidos del trabajador, nombre del cliente, CIF del cliente* de todos aquellos servicios que se hayan realizado entre el 1 del 1 de 2006 y el 31 del 12 del 2006

(45) Crear una consulta llamada *Servicios en la calle Larga SQL*

En ella aparecerán los campos *fecha del servicio, tipo, cantidad, nombre del cliente, dirección del cliente, DNI del trabajador* para todos aquellos servicios realizados en la calle *Larga*.

(46) Crear una consulta llamada *Servicios trabajadores 2006 SQL*

En ella deben aparecer los campos *fecha del servicio, tipo cantidad, nombre y apellidos del trabajador y fecha de entrada del trabajador* de todos aquellos servicios realizados por los trabajadores que entraron en la empresa a partir del 1 – 1 – 2006.

Para crear esta consulta tienes que usar las tablas Trabajadores y Servicios.

(47) Crear una consulta llamada *Clientes de Seguros SQL*

En ella deben aparecer los campos *CIF, nombre del cliente, dirección del cliente* de todos aquellos clientes que gestionen seguros (deben contener en el nombre la palabra "seguros")

(48) Crear una consulta llamada *Listado de Academias y Papelerías SQL*

En ella deben aparecer los campos *CIF, nombre del cliente, dirección del cliente y teléfono fijo* de todos aquellos clientes que sean academias o papelerías.

(49) Crear una consulta llamada *Listado de SEAT y trabajadores SQL*

En ella deben aparecer los campos *matrícula, marca y modelo* de los seat de la empresa. También interesa que aparezca el *nombre y apellido* del conductor de cada coche.

(50) Crear una consulta llamada *Servicios realizados con CITROEN SQL*

En ella debe aparecer el listado de servicios que han sido realizados usando alguno de los citroen de la empresa.

En esta consulta deben aparecer los siguientes campos: *matrícula del vehículo, marca y modelo*. *Nombre y apellidos del trabajador que hizo el servicio*. *Nombre y dirección del cliente al que se le hizo el servicio*. *Tipo de servicio y cantidad*.

En esta consulta participan todas las tablas de la base de datos.

EJERCICIO 2

Realice también estas otras consultas usando el lenguaje SQL (asígneles el nombre que quiera):

(1) Se pide mostrar un listado de clientes con las siguientes características:

Campos a mostrar: nombre del cliente, dirección, teléfono 1.

La condición es que no tengan teléfono 2 (dicho de otra forma, que el campo teléfono 2 sea nulo)

(2) Se pide mostrar un listado de clientes con las siguientes características:

Campos a mostrar: nombre, dirección, teléfono 1.

La condición es que vivan en una calle, es decir, que su dirección comience por “C/”

(3) Se pide mostrar un listado de trabajadores con las siguientes características:

Campos a mostrar: nombre, sueldo, fecha de entrada.

La condición es que hayan entrado en la empresa en el año 2002. Es decir, entre el 1/1/2002 y el 31/12/2002.

(4) Se pide mostrar un listado de trabajadores con las siguientes características:

Campos a mostrar: nombre, apellidos, sueldo.

Condición: ninguna.

Se pide que el listado salga ordenado por sueldo de mayor a menor

(5) Se pide mostrar un listado de trabajadores con las siguientes características:

Campos a mostrar: todos

Condición: Que no se llamen “Ana”

Ordenados por nombre ascendentemente.

(6) Se pide mostrar un listado de coches con las siguientes características:

Mostrar todos los campos.

Ordenado por año de compra, de más antiguo a más moderno.

(7) Mejore la consulta anterior de forma que también se muestre el nombre del trabajador que conduce cada coche.

F d ió Códi Lib D i i

(8) Se pide mostrar un listado de servicios con las siguientes características:

Campos a mostrar: tipo de servicio, fecha, cantidad, comentario.

Condición: mostrar los servicios de limpieza que hayan costado menos de 250 euros.

Ordenado por cantidad de menor a mayor

(9) Mejore la consulta anterior de forma que también se muestre la dirección donde se hizo el servicio.

(10) Se pide mostrar un listado de servicios con las siguientes características:

Campos a mostrar: tipo de servicio, fecha, cantidad, comentario.

Condición: se pide mostrar los servicios de limpieza y los de fontanería todos en la misma consulta.

Ordenado por fecha de más reciente a más antigua.

(11) Se pide mostrar un listado de servicios con las siguientes características:

Campos a mostrar: el nombre del trabajador, la fecha en que se hizo el servicio, la cantidad que costó el servicio, el tipo de servicio, el nombre del cliente, la dirección del cliente.

Condición: se pide mostrar aquellos servicios de fontanería realizados por el trabajador con nombre Juan que hayan costado menos de 240 euros.

El listado debe aparecer ordenado por cantidad de mayor a menor.

PROGRAMACIÓN

JAVA - BASES DE DATOS – SQL – ALTAS/BAJAS/MODIFICACIONES

CREACIÓN DE CONSULTAS DE ACCIÓN USANDO SQL

PREPARACIÓN

Los ejercicios que vienen a continuación modificarán el contenido de las tablas. En el caso de equivocación es posible perder todos los datos contenidos en la tabla con la que se está trabajando. Para evitar problemas, se harán copias de las tablas y se trabajará con las copias.

En este ejercicio de preparación se explica brevemente como realizar una copia de una tabla. Siga los pasos que se indican a continuación:

(13) Entre en la base de datos MANEMPSA que tiene guardada en MIS DOCUMENTOS.

(14) En la zona de tablas, haga clic con el botón derecho sobre la tabla *Trabajadores* y active la opción *Copiar*.

(15) En una zona en blanco haga clic con el botón derecho y active la opción *Pegar*.

(16) Aparecerá un cuadro de diálogo indicando que va a copiar una tabla. Tiene que indicar el nombre que tendrá la copia. En nuestro ejemplo, el nombre será *Trabajadores2*. Pulse aceptar.

(17) Si todo ha ido bien, verás que ha aparecido una nueva tabla llamada *Trabajadores2*. Si la abres, observarás que su contenido es el mismo que *Trabajadores*.

EJERCICIO:

Crear una copia de las demás tablas:

Clientes → Clientes2
Coches → Coches2
Servicios → Servicios2

(Nota: para los siguientes ejercicios usaremos las copias de las tablas, para así mantener intacto el contenido de las tablas originales)

INSERCIÓN DE REGISTROS

NOTA: COMPROBAR QUE LOS EJERCICIOS SE HAN REALIZADO CORRECTAMENTE ENTRANDO EN LAS TABLAS Y OBSERVANDO SU CONTENIDO

EJERCICIOS:

(8) En la tabla *Trabajadores2* añada los siguientes trabajadores:

Rosa González, con DNI 11.567.789-D, cobra 1100,60 euros y entró en la empresa el 2 de marzo del 2006. No tiene asignado un coche.

Benito Rodríguez, con DNI 56.456.345-W, cobra 500 euros y entró en la empresa el 10 de Diciembre de 2005. El coche que tiene asignado tiene de matrícula 4454-EEE

Eva Ramos, con DNI 33.987.987-F. Entró en la empresa el 20 de enero de 2007. No tiene asignado de momento un sueldo y tampoco tiene asignado coche.

(9) En la tabla *Coches2* añada el siguiente coche:

SEAT Córdoba con matrícula 4454-EEE. Año 2004. Está asignado al trabajador con DNI 56.456.345-W

MODIFICACIÓN DE REGISTROS

NOTA: COMPROBAR QUE LOS EJERCICIOS SE HAN REALIZADO CORRECTAMENTE ENTRANDO EN LAS TABLAS Y OBSERVANDO SU CONTENIDO

EJERCICIOS:

(51) Se pide asignar a los trabajadores de la tabla *Trabajadores2* que hayan entrado antes del año 2005 un sueldo de 1400 euros.

(52) En la tabla *Trabajadores2* asigne al trabajador con DNI 33.987.987-F el coche con matrícula 4454-EEE.

(53) La trabajadora llamada *Ana* ha sido suspendida de empleo. Asigne un valor 0 al sueldo de esta trabajadora en la tabla *Trabajadores2*.

(54) Se ha averiguado el teléfono móvil del gerente de la empresa *Academia La Plata*. El móvil es el 633 45 54 45. Se pide que en la tabla *Clientes2* asigne ese teléfono móvil a dicho cliente.

(55) Se han detectado algunos errores de datos en el cliente con CIF B44556666. Concretamente, el nombre del cliente es *Seguros La Cruz*, su dirección es *C/Lealas 10*, y su teléfono fijo es el *956 30 90 90*. Se pide que en la tabla *Clientes2* haga dichos cambios a dicho cliente.

ELIMINACIÓN DE REGISTROS

NOTA: COMPROBAR QUE LOS EJERCICIOS SE HAN REALIZADO CORRECTAMENTE ENTRANDO EN LAS TABLAS Y OBSERVANDO SU CONTENIDO

(41) Eliminar de la tabla *Servicios2* a todos los servicios del año 2004 (es decir, cuya fecha esté comprendida entre 1-1-2004 y 31-12-2004)

Comprueba el resultado observando el contenido de la tabla *Servicios2*

(42) Eliminar de la tabla *Servicios2* a todos los servicios de fontanería de más de 250 euros.

Comprueba el resultado observando el contenido de la tabla *Servicios2*

(43) Eliminar de la tabla *Servicios2* aquellos servicios en los que se haya instalado algún recambio (es decir, que el campo comentario contenga la palabra "recambio")

Comprueba el resultado observando el contenido de la tabla *Servicios2*

(44) Finalmente, elimina todos los registros de la tabla *Servicios2*.

Comprueba el resultado observando el contenido de la tabla *Servicios2*

(45) En la tabla *Clientes2*, eliminar a todos los clientes que no tengan móvil (es decir, que el campo móvil sea nulo)

Comprueba el resultado observando el contenido de la tabla *Servicios2*

(46) En la tabla *Clientes2*, eliminar a todos los clientes que tengan una empresa de seguros (es decir, que contengan la palabra "seguros" en el nombre del cliente)

Comprueba el resultado observando el contenido de la tabla *Servicios2*

(47) Finalmente, eliminar de la tabla *Clientes2* el resto de registros.

Comprueba el resultado observando el contenido de la tabla *Servicios2*

EJERCICIO FINAL

Para la realización de los ejercicios anteriores se han hecho copias de las tablas de la base de datos. Una vez realizados dichos ejercicios estas tablas no son necesarias. Puede borrarlas simplemente haciendo clic con el botón derecho sobre ellas y activar la opción *eliminar*.

Así pues, elimine las tablas: *Trabajadores2, Clientes2, Servicios2, Coches2*.

PROGRAMACIÓN

JAVA – APLICACIONES DE BASES DE DATOS

CREACIÓN DE APLICACIONES JAVA USANDO BASE DE DATOS

EJERCICIO 1

Crear un programa en Java que manipulará la base de datos MANEMPSA.

La ventana principal de este proyecto contendrá los siguientes botones:

(1) "Coches de la Empresa" – *btnCoches*

Cuando se pulse este botón, deberá aparecer en un JOptionPane el listado de coches de la empresa.

Debe aparecer la marca, modelo y año de cada coche, y además deben aparecer ordenados por año ascendentemente.

(2) "Listado de Clientes" – *btnClientes*

Cuando se pulse este botón, deberá aparecer en un JOptionPane el listado de clientes de la empresa.

Debe aparecer el CIF del cliente, el nombre y la dirección. Los clientes deben aparecer ordenados por nombre.

(3) "Listado de Servicios" – *btnServicios*

Cuando se pulse este botón aparecerá en un JOptionPane el listado de servicios realizados.

Debe aparecer el tipo de servicio, la cantidad del servicio y el CIF del cliente. Deben aparecer ordenados por cantidad de mayor a menor.

(4) "Mil Euristas" – *btnMil*

Cuando se pulse este botón, deberá aparecer en un JOptionPane el listado de trabajadores de la empresa que cobren 1000 o menos euros.

Debe aparecer el DNI del trabajador, nombre, apellidos y sueldo.

(5) "Limpiezas" – *btnLimpiezas*

Cuando se pulse este botón, deberá aparecer en un JOptionPane el listado de servicios de limpieza realizados.

Debe aparecer el número del servicio, el DNI del trabajador que realizó el servicio, el tipo de servicio y el coste. Debe aparecer ordenado por número de servicio de menor a mayor.

(6) "Electricidad" – *btnElectricidad*

Este botón es igual que el anterior, solo que en vez de mostrar los servicios de limpieza se tienen que mostrar los servicios de electricidad.

(7) “Ingresos Empresa” – *btnIngresos*

Cuando se pulse este botón, interesa que aparezca en un JOptionPane la suma de las cantidades de todos los servicios realizados.

(8) “Citroen” – *btnCitroen*

Cuando se pulse este botón, interesa que aparezca en un JOptionPane el número de coches citroen que tiene la empresa.

(9) “Estadisticas Limpieza” – *btnEstLim*

Cuando se pulse este botón, interesa que aparezca en un JOptionPane lo siguiente:

j. El número de servicios de limpieza que se han realizado.
k. La suma de los costes de todos los servicios de limpieza.
l. El coste medio de un servicio de limpieza.

(13)“El quinto servicio” – *btnQuinto*

Cuando se pulse este botón interesa que aparezca en un JOptionPane los datos del quinto servicio realizado.

Interesa que aparezca el número del servicio, el tipo de servicio y el coste de dicho servicio.

(14)“Listado de servicios de fontanería” – *btnFont*

Cuando se pulse este botón interesa que aparezca en un JOptionPane un listado con los servicios de fontanería.

Interesa que aparezca el nombre y apellidos del trabajador que hizo el servicio, el tipo de servicio y el coste del servicio.

(15)“Listado de servicios de menos de 200 euros” – *btnMenosDos*

Cuando se pulse este botón interesa que aparezca en un JOptionPane un listado con los servicios con una cantidad menor de 200 euros.

Interesa que aparezca el nombre del trabajador que hizo el servicio, el tipo de servicio, el coste y el nombre del cliente al que se le hizo el servicio. Haz que el listado salga ordenado por coste de mayor a menor.

PROGRAMACIÓN

JAVA – APLICACIONES DE BASES DE DATOS

FECHAS, NÚMEROS REALES, VALORES NULOS

EJERCICIO 1

Crear un programa en Java que manipulará la base de datos MANEMPSA.

La ventana principal de esta aplicación tendrá dos botones y un JTextPane:

Botón Datos Trabajadores

Al pulsar este botón debe aparecer el listado de trabajadores en el JTextPane.

Concretamente deben aparecer el nombre, apellidos, sueldo, fecha de entrada y matrícula del coche que conduce.

El listado debe aparecer ordenado por fechas ascendentemente (es decir, primero los trabajadores con más antigüedad, y luego los más recientes)

Las fechas deben aparecer en el siguiente formato:

Día / Mes / Año

Los sueldos deben aparecer con la coma decimal.

F d ió Códi Lib D i i

Si el trabajador no tuviera asignada un coche, en el campo matrícula debería aparecer la cadena “*sin coche*”.

Botón Datos Servicios

Al pulsar este botón aparecerá el listado de servicios en el JTextPane.

El listado aparecerá ordenado por fechas de más reciente a más antiguo.

Debe aparecer la fecha del servicio, el coste, el tipo y el comentario.

La fecha debe aparecer en formato: *Dia del Mes del Año*. Por ejemplo: *3 del 5 del 2001*

El coste aparecerá con coma decimal.

Si no hay comentario para el servicio, debe aparecer la cadena “*sin comentarios*”.

F d ió Códi Lib D i i

PROGRAMACIÓN

JAVA – APLICACIONES DE BASES DE DATOS

CONSULTAS DEL USUARIO

Realizar una pequeña aplicación de base de datos que maneje la base de datos MANEMPSA.

La ventana principal del programa contendrá un JTextPane y los siguientes elementos:

Botón Ver Servicios

Al pulsar este botón aparecerá en el JTextPane el listado con todos los servicios realizados por la empresa ordenados por fecha de forma ascendente.

Deben aparecer en el listado los siguientes campos: fecha del servicio, tipo, coste y comentario.

La fecha aparecerá en formato dia-mes-año.
El coste aparecerá con coma decimal.
No interesa que aparezcan los valores null. Cuando esto suceda, sustituirlos por un simple guión "-".

Selección por costes:

Se deben añadir a la ventana los siguientes elementos:

(34)Un cuadro de texto.
(35)Un botón "Mayor que"
(36)Un botón "Menor que"
(37)Un botón "Igual a"
(38)Un botón "Distinto de"

En el cuadro de texto el usuario introducirá una cantidad.

Al pulsar el botón "Mayor que" aparecerá el listado de servicios cuyo coste sea mayor a dicha cantidad.
Al pulsar el botón "Menor que" aparecerá el listado de servicios cuyo coste sea menor a dicha cantidad.
Al pulsar el botón "Igual a" aparecerá el listado de servicios cuyo coste sea igual a dicha cantidad.
Al pulsar el botón "Distinto de" aparecerá el listado de servicios cuyo coste sea distinto a dicha cantidad.

Selección por tipo:

Se deben añadir a la ventana los siguientes elementos:

(39)Un cuadro de texto.
(40)Un botón "Igual a".
(41)Un botón "Contiene a".

El usuario introducirá un texto en el cuadro de texto.

Al pulsar el botón “Igual a”, el programa mostrará el listado de servicios cuyo tipo sea igual al texto introducido.

Por ejemplo, si el usuario introduce el texto “Limpieza”, el programa mostrará los servicios cuyo tipo sea igual a “Limpieza”.

Al pulsar el botón “Contiene a”, el programa mostrará el listado de servicios cuyo tipo contenga el texto del cuadro de texto.

Por ejemplo, si el usuario introduce el texto “ía”, el programa mostrará el listado de servicios de Fontaner**ía**, Carpinter**ía**, etc. Es decir, todos los servicios que contengan “ía” en el tipo.

Selección por fecha:

Se deben añadir a la ventana los siguientes elementos:

(42)Un cuadro de texto para el día, otro para el mes y otro para el año.
(43)Un botón “Anterior a”
(44)Un botón “Posterior a”
(45)Un botón “En el año”

El usuario introducirá una fecha en los cuadros día-mes-año.

Luego, si el usuario pulsa el botón *Anterior a* el programa mostrará el listado de servicios que hayan sido anteriores a la fecha indicada.

Si el usuario pulsa *Posterior a* el programa mostrará el listado de servicios que hayan sido posteriores a la fecha indicada.

Si el usuario pulsa el botón *En el año*, el programa mostrará el listado de servicios que hayan sido realizados en el año indicado en el cuadro año. (En este caso no se tienen en cuenta los cuadros día y mes)

PROGRAMACIÓN

JAVA – APLICACIONES DE BASES DE DATOS

ALTAS DE REGISTROS

Realizar una pequeña aplicación de base de datos que maneje la base de datos MANEMPSA.

La ventana principal de dicha aplicación contendrá los siguientes elementos:

Botón Listado de coches

Al pulsar este botón, aparecerá en un panel de texto el listado de coches de la empresa. Deben aparecer todos los campos de los coches.

Panel de Alta de coches

Este panel contendrá una serie de cuadros de texto que permitirán introducir los datos de un nuevo coche que se quiera añadir.

Dentro del panel también tendremos un botón *Alta* que al ser pulsado efectuará la introducción del nuevo coche en la tabla.

Al pulsarse el botón *Alta,* deberá mostrarse el contenido de la tabla *coches* en el panel, y allí deberá aparecer el listado incluyendo el nuevo coche introducido.

(El código del botón *Alta* deberá construir una consulta SQL válida del tipo INSERT INTO que permita introducir los datos del nuevo coche)

PROGRAMACIÓN

JAVA – APLICACIONES DE BASES DE DATOS

BAJA DE REGISTROS

Mejorar la aplicación realizada en la hoja de ejercicios anterior de forma que posibilite la eliminación de coches de la tabla coches de la base de datos MANEMPSA.

Para ello, debe añadir a la ventana principal lo siguiente:

(16) Un cuadro de texto donde el usuario introducirá una matrícula.
(17) Un botón *Eliminar coche*.

Al pulsar el botón *Eliminar coche*, se borrará de la tabla el coche cuya matrícula se haya escrito en el cuadro de texto.

Una vez realizado el borrado, aparecerá en el panel de la ventana el contenido de la tabla coches para que el usuario pueda confirmar que el coche ha sido realmente borrado.

Hay que tener en cuenta varias cosas:

(18) El programa avisará si la matrícula introducida en el cuadro de texto no se corresponde con ningún coche de la base de datos. (En este caso no se hará el borrado, claro está)

(19) En el caso de que la matrícula exista, el programa pedirá confirmación antes de efectuar el borrado.

PROGRAMACIÓN

JAVA – APLICACIONES DE BASES DE DATOS

MODIFICACIÓN DE REGISTROS

Mejorar la aplicación realizada en la hoja de ejercicios anterior de forma que posibilite la modificación de coches de la tabla coches de la base de datos MANEMPSA.

Para ello, debe añadir a la ventana principal lo siguiente:

(20) Un panel de modificación que contenga un cuadro de texto por campo de la tabla coches (Este panel será muy parecido al panel de introducción de coches)
(21) Un botón *Buscar* al lado del cuadro de texto de la matrícula.
(22) Un botón *Efectuar Modificación*.

El usuario tendrá que introducir una matrícula y luego pulsará el botón *Buscar*. Al pulsar este botón, el programa rellenará los cuadros de texto con los datos del coche que tenga dicha matrícula. Si no existe un coche con dicha matrícula entonces el programa mostrará un mensaje de error.

Una vez rellenos los cuadros de texto con los datos del coche, el usuario realizará modificaciones en dichos datos y luego pulsará el botón *Efectuar Modificación*.

Al pulsar este botón se efectuará la modificación de los datos de este coche en la tabla coches y se presentará el listado completo de la tabla coches en el panel del programa.

PROGRAMACIÓN

JAVA – JTABLE

EJERCICIO Nº1

Se pide que realice una aplicación que muestre una tabla con las siguientes columnas:

Nombre	Examen 1	Examen 2	Examen 3

Al comenzar el programa la tabla estará vacía y solo se mostrará la cabecera.

El programa constará además de la tabla de los siguientes elementos:

(23) Un botón “Añadir Blanco”.

Este botón añadirá una nueva fila en blanco a la tabla.

(24) Un botón “Añadir Alumno”.

Este botón añadirá una nueva fila a la tabla con los siguientes valores:

Nombre: “No definido”
Examen 1: 5
Examen 2: 5
Examen 3: 5

(25) Un botón “Eliminar”

Este botón eliminará la fila donde que esté seleccionada en ese momento.

En el caso de que no hubiera ninguna fila seleccionada al pulsar este botón, el programa debe mostrar un mensaje de error.

(26) Un botón “Resultados”

Este botón mostrará en un JOptionPane la nota media del alumno de la fila que esté seleccionada. También debe indicar en ese JOptionPane si el alumno ha aprobado o no.

En el caso de que no hubiera ninguna fila seleccionada al pulsar este botón, el programa debe mostrar un mensaje de error.

(27) Un botón “Resultados Trimestre 1”

Este botón mostrará en un JOptionPane el número de alumnos que aprobaron el examen 1 y el número de alumnos que suspendieron dicho examen.

PROGRAMACIÓN

JAVA – JTABLE

EJERCICIO Nº1

Realiza un programa que contenga un JTable y un botón. El botón tendrá el texto “Clientes” y al pulsarse se mostrará en la tabla el contenido de la tabla *clientes* de la base de datos MANEMPSA.

EJERCICIO Nº2

Realiza un programa igual al anterior pero que permita mostrar el contenido de la tabla *servicios* de la base de datos MANEMPSA.

PROGRAMACIÓN

JAVA – JTABLE

EJERCICIO Nº1

Realiza un programa que permita realizar Altas, Bajas y Modificaciones en la tabla *clientes* de la base de datos MANEMPSA.

Este programa debe mostrar en todo momento un JTable con el contenido actualizado de la tabla *clientes*.

El programa debe tener tres botones. Un botón *Eliminar*, otro botón *Nuevo Cliente* y un botón *Modificar Cliente*.

Al pulsar *Eliminar* se eliminará el cliente seleccionado en el JTable.

Al pulsar *Nuevo Cliente* aparecerá un cuadro de diálogo donde el usuario introducirá los datos de un nuevo cliente.

Al pulsar *Modificar Cliente* aparecerá un cuadro de diálogo donde el usuario podrá cambiar los datos del cliente que haya seleccionado en el JTable.

PROGRAMACIÓN

JAVA – JTABLE

EJERCICIO Nº1

El programa realizado en la hoja de ejercicios anterior permitía el alta, baja y modificación de los clientes de la base de datos MANEMPSA.

Se pide ahora que añada un cuadro de diálogo de filtrado al ejercicio de la hoja anterior que permita filtrar por los distintos campos de la tabla de clientes.

PROGRAMACIÓN

JAVA – JTABLE

EJERCICIO Nº1

El programa de la hoja de ejercicios anterior permitía la gestión de la tabla *clientes* de la base de datos MANEMPSA.

Se pide que mejore el programa de la hoja anterior de forma que el cuadro de diálogo de filtrado permita al usuario decidir la ordenación del listado.

Por otro lado, se pide que junto al JTable donde aparece el listado de clientes aparezca el número de clientes que se está mostrando en todo momento en el JTable.

PROGRAMACIÓN

JAVA – iREPORT

EJERCICIO Nº1

Crear una DSN llamada *EjercicioDSN* que haga referencia a la base de datos MANEMPSA que tiene almacenada en Mis Documentos.

EJERCICIO Nº2

En el programa iReport, crear una conexión llamada *Conexión Ejercicio* que esté creada a partir de la DSN del ejercicio anterior.

EJERCICIO Nº3

Crear un informe usando la conexión *Conexión Ejercicio* del ejercicio anterior donde aparezca el listado de clientes de la base de datos MANEMPSA ordenado por nombre de cliente.

El título de este listado debe ser: *Listado de Clientes* y luego guarde el informe con el nombre *clientes*.

PROGRAMACIÓN

JAVA – iREPORT

EJERCICIO Nº1

Crea un listado de clientes usando la opción *mago de informes* de iReport. Este listado debe mostrar los siguientes campos: nombre, dirección y los dos teléfonos. El listado debe salir ordenado por nombre.

Una vez creado el listado, debe realizar las siguientes modificaciones en él:

(48) El título del listado debe ser: *"Clientes de MANEMPSA"*. Y debe tener las siguientes características:
   ww. Color de fondo: Azul.
   xx. Tamaño de la letra: 18.
   yy. Color de letra: Blanco.
   zz. En cursiva.

(1) La cabecera del listado debe tener las siguientes características:
   a. Tamaño de letra: 10
   b. Cada campo de la cabecera debe tener una alineación Centro.
   c. Los campos deben estar en negrita y subrayados
   d. Deben ser de color Azul.

(2) Los datos del listado deben tener las siguientes características:
   a. Deben tener alineación centro.
   b. Deben tener un tamaño de letra de 10 y el tipo de letra debe ser *Comic Sans*.
   c. Los nombres deben aparecer en color verde, al igual que las direcciones.
   d. Los teléfonos deben aparecer en color rojo y en negrita.

Una vez hecho esto, si quiere experimentar realizando algún cambio más, hágalo.

PROGRAMACIÓN

JAVA – iREPORT

EJERCICIO Nº1

Se pide realizar un informe en el que aparezcan los servicios realizados al cliente “Seguros Segasa”, cuyo CIF es ‘B11223212’.

Interesa que el informe tenga el siguiente diseño:

(30) El título será NOTA DE SERVICIOS SEGASA

(31) En el encabezado de página debe aparecer una ficha como la que sigue:

Ficha de Cliente.

CIF: ---el cif del cliente---
NOMBRE: ---el nombre del cliente---
DIRECCIÓN: ---dirección del cliente---
TELEFONO 1: ---teléfono 1 del cliente---
TELEFONO 2: ---teléfono 2 del cliente---

(32) Incluya también en el encabezado de página un logotipo (bájese una imagen de internet)

(33) El listado de servicios debe mostrar los datos: *Fecha del servicio, tipo y cantidad* ordenado ascendentemente por fechas. Por supuesto, este listado debe aparecer con su encabezado. Sería interesante que destacara el encabezado colocándolo en negrita y separándolo del listado de datos a través de una línea.

(34) En la zona del pie de columna, añada simplemente una línea horizontal.

(35) En la zona del pie de página y del pie de última página debe aparecer el siguiente texto centrado:

MANEMPSA
Mantenimiento Integral a Empresas

Polígono Industrial Norte, C/ Bonanza 3
11404 Jerez (Cádiz)

F d ió Códi Lib D i i