Tema 3

Sentencias de Control de Java

Sentencias

Las sentencias son las instrucciones de las que está formado un programa. Las sentencias en un programa en Java se encuentran en el cuerpo de los métodos, incluyendo los constructores de las clases.

Las sentencias en Java se clasifican de la siguiente manera:

- Sentencia de expresión o simple
- Sentencia compuesta o bloque
- Sentencias de selección o condicionales
- Sentencias de iteración o repetitivas
- Sentencias de salto

Sentencia de Expresión o Simple

La mayoría de las sentencias de un programa en Java son sentencias de expresión. Una **sentencia de expresión** tiene la siguiente sintaxis:

```
[expresión];
```

Esto es, una expresión seguida de un punto y coma (;). La mayoría de las sentencias de expresión son asignaciones o invocaciones a métodos. Si *expresión* se omite, la construcción se llama **sentencia nula**.

El punto y coma (;) es un terminador de sentencia.

Algunos ejemplos de sentencias expresión son:

```
y = x + 3;
n++;
this.titulo = título;
System.out.println("\nHola");
;     // sentencia nula
```

Sentencia compuesta o bloque

Una **sentencia compuesta** o **bloque** son construcciones que contienen otras sentencias las cuales se deberán ejecutar en secuencia. La sintaxis de una sentencia compuesta es:

```
{
      [Declaración local] ...

      sentencia ...
}
```

Una sentencia compuesta está formada de declaraciones locales y sentencias. Los objetos declarados localmente son sólo conocidos dentro de la sentencia compuesta en que fueron declarados. Las sentencias dentro de una sentencia compuesta pueden se de cualquier tipo: sentencias de expresión, sentencias compuesta, sentencias de selección, sentencias repetitivas, sentencias de salto. Note que dentro de una sentencia compuesta podemos tener anidadas sentencias compuestas.

Una sentencia compuesta va encerrada entre llaves { y }.

Una sentencia compuesta es sintácticamente equivalente a una sentencia simple, esto es, donde puede ir una sentencia simple podemos poner una sentencia compuesta y viceversa.

Un ejemplo de una sentencia compuesta es el cuerpo de un método.

Ejemplo sobre la sentencia Compuesta

Como un ejemplo de una sentencia compuesta, considere el constructor de la clase Canción, del problema sobre el amante de la música y el cine, que recibe como parámetros los valores para inicializar los atributos de la clase:

```
public Cancion(String clave, String titulo, Genero genero,
               String interprete, String autor, String album, int duracion,
               Fecha fecha) {
  this.titulo = titulo;
  this.genero = genero;
  this.interprete = interprete;
  this.autor = autor;
  this.album = album;
  this.duracion = duracion;
  this.fecha = fecha;
}
```

Cada una de las líneas de código del cuerpo del constructor es una sentencia simple y todas juntas constituyen una sentencia compuesta.

Sentencias de Selección o Condicionales

Hay ocasiones en las que deseamos que, dependiendo del valor de una expresión, la computadora ejecute una de dos o más sentencias compuestas. Las sentencias que nos permiten realizar tal tarea se conocen como **sentencias de selección o condicionales**. En Java existen dos sentencias condicionales: La sentencia **if** y la sentencia **switch**.

Sentencia if

La sentencia **if** nos permite seleccionar entre ejecutar una de dos sentencias compuestas, dependiendo del resultado de una expresión. La sintaxis de la sentencia **if** es la siguiente:

```
if(expresión) sentencia1 [else sentencia2]
```

expresión debe ser de tipo booleano. *sentencia1* y *sentencia2* son sentencias compuestas.

Primero se evalúa *expresión*, si su valor es verdadero, se ejecuta *sentencia1* y después el programa continúa con la sentencia que sigue a la sentencia **if**; si el valor de *expresión* es falso se ejecuta *sentencia2* y luego continúa con la sentencia después de la sentencia **if**.

Si **else** *sentencia2* no aparece y *expresión* es verdadera se ejecuta *sentencia1* y el programa continúa con la sentencia después de la sentencia **if**. De otro modo no se ejecuta *sentencia1* y el programa salta a ejecutar la sentencia después de la sentencia **if**.

Una sentencia **if** puede estar anidada dentro de otra sentencia **if**, por ejemplo:

```
if(expresión1)
   if(expresión2) sentencia1
   else sentencia2
else
   if(expresión3) sentencia3
   else sentencia4;
```

ó escalonados como en el siguiente ejemplo:

```
if(expresión1) sentencia1
else if(expresión2) sentencia2
   else if(expresión3) sentencia3
       else sentencia4;
```

Esta última construcción se acostumbra escribir de la siguiente manera para que los sangrados no crezcan demasiado:

```
if(expresión1) sentencia1
else if(expresión2) sentencia2
else if(expresión3) sentencia3
else sentencia4
```

En los casos donde tenemos sentencias **if** anidadas podría surgir la pregunta de a qué **if** le corresponde cuál **else**. La regla en estos casos es que un **else** se asocia con el **if** anterior más cercano si es que no está ya asociado con un **else**.

Ejemplo sobre la sentencia if

El costo de un telegrama ordinario es de $25.00 si el número de palabras es hasta 10, por cada palabra adicional se cobra $2.50. Si el telegrama es urgente los costos son de $40,00 y $4.00 respectivamente. Escribir una clase permita crear telegramas y determine su costo.

El código de la clase Telegrama es la siguiente:

```
/*
 * Telegrama.java
 *
 * Creada el 10 de octubre de 2005, 12:36 PM
 */

package telegrama;

/**
 * Esta clase permite calcular el costo de un telegrama
 * Ilustra el uso de la sentencia if
 *
 * @author mdomitsu
 */
public class Telegrama {
  private final double COSTO_ORDINARIO = 25.0;
  private final double COSTO_URGENTE = 40.0;
  private final double COSTO_ADICIONAL_ORDINARIO = 2.5;
  private final double COSTO_ADICIONAL_URGENTE = 4.0;
  private String tipoTelegrama;
  private int numPalabras;
  private double costo;

  public Telegrama(String tipoTelegrama, int numPalabras) {
    this.tipoTelegrama = tipoTelegrama;
    this.numPalabras = numPalabras;
    costo = calculaCosto();
  }

  public double calculaCosto() {
    if(tipoTelegrama.charAt(0) == 'O' || tipoTelegrama.charAt(0) == 'o')
```

```
      if(numPalabras <= 10) return COSTO_ORDINARIO;
      else return COSTO_ORDINARIO +
                  COSTO_ADICIONAL_ORDINARIO * (numPalabras - 10);
    else if(tipoTelegrama.charAt(0) == 'U' || tipoTelegrama.charAt(0) == 'u')
      if(numPalabras <= 10) return COSTO_URGENTE;
      else return COSTO_URGENTE + COSTO_ADICIONAL_URGENTE * (numPalabras - 10);
    else return 0;
  }

  public String toString() {
    return tipoTelegrama + ", " + numPalabras + ", " + costo;
  }
}
```

Sentencia switch

Un caso especial de una construcción con varios **if** escalonados y que aparece con bastante frecuencia en los programas, tiene la siguiente forma:

```
if(expresión == cte1) sentencia1
else if(expresión == cte2) sentencia2
else if(expresión == cte3) sentencia3
...
else sentencia
```

donde *expresión* es una expresión de tipo entero, *cte1*, *cte2*, *cte3*, ... son constantes de tipo entero y *sentencia1*, *sentencia2*, ... , *sentencia* son sentencias compuestas. La construcción anterior puede simplificarse utilizando la **sentencia switch**. La sintaxis de una sentencia **switch** es la siguiente:

```
switch(expresión) {
   case cte1: [sentencia1]
              [break;]
  [case cte2: [sentencia2]
              [break;]] ...

  [default:   [sentencia]
              [break;]]
}
```

expresión es una expresión de tipo char, byte, short, o int, y se conoce como el selector del **switch**.

cte1, *cte2*, ... son constantes de tipo char, byte, short, o int. En una sentencia **switch** no puede haber dos constantes iguales.

Primero se evalúa *expresión* y luego se ejecuta la sentencia compuesta que está asociada a la constante que es igual al valor de *expresión*. Después de ejecutar esa sentencia, se ejecutan en secuencia todas las sentencias que le siguen hasta encontrar

una sentencia **break**. Al encontrarse con una sentencia **break**, el programa continúa con la sentencia después de la sentencia switch.

Si ninguna constante coincide con el valor de *expresión* entonces se ejecuta la sentencia asociado con la etiqueta **default**. Si ésta no existe, el programa continúa con la sentencia después del **switch**.

La sentencia asociada a la etiqueta default no tiene que ser la última sentencia dentro de una sentencia **switch**. Puede ir en cualquier lugar, por ejemplo:

```
switch(expression {
   case cte1: sentencia1
              break;
   case cte2: sentencia2
              break;
   default:   sentencia
              break;
   case cte3: sentencia3
              break;
   case cte4: sentencia4
}
```

ó

```
switch(expresión) {
   default:   sentencia]
              break;
   case cte1: sentencia1
              break;
   case cte2: sentencia2
              break;
   case cte3: sentencia3
              break;
   case cte4: sentencia4
}
```

Ejemplo sobre la sentencia switch

El siguiente código es una modificación de la clase telegrama para que el método calculaCosto() emplee una sentencia **switch** en lugar de **ifs** anidados.

```
/*
 * Telegrama2.java
 *
 * Creada el 10 de octubre de 2005, 12:36 PM
 */

package telegrama;

/**
 * Esta clase permite calcular el costo de un telegrama.
```

```
 * Ilustra el uso de la sentencia switch
 *
 * @author mdomitsu
 */
public class Telegrama2 {
  private final double COSTO_ORDINARIO = 25.0;
  private final double COSTO_URGENTE = 40.0;
  private final double COSTO_ADICIONAL_ORDINARIO = 2.5;
  private final double COSTO_ADICIONAL_URGENTE = 4.0;
  private String tipoTelegrama;
  private int numPalabras;
  private double costo;

  public Telegrama2(String tipoTelegrama, int numPalabras) {
    this.tipoTelegrama = tipoTelegrama;
    this.numPalabras = numPalabras;
    costo = calculaCosto();
  }

  public double calculaCosto() {
    switch(tipoTelegrama.charAt(0)) {
      case 'O':
      case 'o':
        if (numPalabras <= 10)
          return COSTO_ORDINARIO;
        else
          return COSTO_ORDINARIO + COSTO_ADICIONAL_ORDINARIO *
                 (numPalabras - 10);
      case 'U':
      case 'u':
        if (numPalabras <= 10)
          return COSTO_URGENTE;
        else
          return COSTO_URGENTE + COSTO_ADICIONAL_URGENTE * (numPalabras - 10);
      default:
        return 0;
    }
  }

  public String toString() {
    return tipoTelegrama + ", " + numPalabras + ", " + costo;
  }
}
```

Sentencias de Iteración o Repetitivas

Hay ocasiones que se requiere que una sentencia compuesta se ejecute varias veces, posiblemente cambiando los valores de las expresiones contenidas en ésta. Por ejemplo, supongamos que deseamos crear una tabla del capital acumulado de una cantidad invertida a una tasa de interés anual y recapitalizado mensualmente. En este ejemplo, para cada mes es necesario determinar el capital acumulado e imprimir el resultado. Este proceso se repetirá tantas veces como el número de meses que deseemos tabular. Las sentencias que nos permiten realizar tal tarea se conocen como **sentencias de iteración**

o repetitivas. En Java existen tres tipos de sentencias repetitivas: La sentencia **while**, la sentencia **for** y la sentencia **do - while**.

Sentencia while

La sentencia **while** nos permite ejecutar una sentencia compuesta, mientras se cumpla una condición. La sintaxis para esta sentencia es la siguiente:

```
while(expresión) sentencia
```

expresión debe ser de tipo booleano. *sentencia* es una sentencia compuesta.

La sentencia **while** opera de la siguiente manera: Primero se evalúa *expresión*. Si su valor es verdadero, se ejecuta *sentencia* y *expresión* es evaluada de nuevo, repitiéndose el ciclo hasta que *expresión* resulta falsa. Cuando esto último ocurre el ciclo termina y el programa continúa con la siguiente instrucción después del ciclo.

Si *expresión* es falsa al principio del ciclo, *sentencia* no se ejecuta ni una vez.

Ejemplo sobre la sentencia while

Se desea una clase llamada Capital para estudiar el comportamiento del capital acumulado de una inversión, a una tasa de interés anual, con recapitalización mensual. Se requiere un método que tabule el capital acumulado mensualmente. La salida del programa será una tabla de la forma:

```
Mes Capital
---------------
1 dddd.dd
2 dddd.dd
...
```

```
/*
 * Capital.java
 *
 * Creada el 11 de octubre de 2005, 12:36 PM
 */

package capital;

/**
 * Esta clase permite tabular el capital acumulado de una inversión,
 * a una tasa de interés anual, con recapitalización mensual.
 * Ilustra el uso de la sentencia while
 *
 * @author mdomitsu
 */
public class Capital {
```

```
  private double capital;
  private double tasa;
  private int meses;

  public Capital(double capital, double tasa, int meses) {
    this.capital = capital;
    this.tasa = tasa;
    this.meses = meses;
  }

  public void tabla() {
    int mes = 1;
    double capital = this.capital;

    System.out.println("Mes     Capital");
    while(mes <= meses) {
       capital *= (1 + tasa / 12);
       System.out.println(mes + ": " + capital);
       mes++;
    }
  }
}
```

Sentencia for

La sentencia for es una sentencia de repetición cuya sintaxis tiene una forma más compacta utilizando la sentencia while. La sintaxis de la **sentencia for** es la siguiente:

for([*expresión1*]; [*expresión2*]; [*expresión3*]**)** *sentencia*

Aquí la expresión *expresión2* es una expresión booleana y *expresión1 y expresión3* son, cada una, una expresión simple o varias expresiones simples separadas por comas.

La sentencia **for** opera de la siguiente manera:

1. Se evalúa *expresión1*. Si *expresión1* está formada de varias expresiones simples, éstas se evalúan en secuencia.

2. Se evalúa *expresión2*. Si el resultado es verdadero entonces se ejecuta la sentencia compuesta, *sentencia* y luego se evalúa *expresión3*. Si *expresión3* está formada de varias expresiones simples, éstas se evalúan en secuencia.

3. Repite el paso 2 hasta que el resultado de la comparación es falso, en cuyo caso el programa brinca a la siguiente sentencia después de la sentencia **for**.

Si inicialmente el valor de *expresión2* es falso, *sentencia* no se ejecuta ni una sola vez y el ciclo termina.

En la sentencia **for**, cualesquiera de las tres expresiones se pueden omitir. En particular si omitimos *expresión2* se considera que es verdadera y tendremos un ciclo infinito. A continuación se muestran algunas construcciones usando la sentencia **for**:

```
for(i = 0; i < N; i++) {
   ...            /* Uso más común */
}

for(i = 0, j = 1; i < N; i++, j += 3) {
   ...            /* Doble inicialización y doble incremento */
}

for( ; ; ) {
   ...            /* Un ciclo infinito */
}
```

Ejemplo sobre la sentencia for

El siguiente código muestra la clase Capital del ejemplo sobre la proposición while modificado para que utilice en su lugar una proposición for.

```
*
 * Capital2.java
 *
 * Creada el 11 de octubre de 2005, 12:36 PM
 */

package capital;

/**
 * Esta clase permite tabular el capital acumulado de una inversión,
 * a una tasa de interés anual, con recapitalización mensual.
 *
 * Ilustra el uso de la sentencia for
 *
 * @author mdomitsu
 */
public class Capital2 {
  private double capital;
  private double tasa;
  private int meses;

  public Capital2(double capital, double tasa, int meses) {
    this.capital = capital;
    this.tasa = tasa;
    this.meses = meses;
  }

  public void tabla() {
    double capital = this.capital;

    System.out.println("Mes     Capital");
    for(int mes = 1; mes <= meses; mes++) {
       capital *= (1 + tasa / 12);
```

```
        System.out.println(mes + ": " + capital);
    }
  }
}
```

Sentencia do - while

La **sentencia do - while** es otra variante de la sentencia **while** cuya sintaxis es la siguiente:

do *sentencia* **while(***expresión***)**

expresión debe ser de tipo booleano. *sentencia* es una sentencia compuesta.

La sentencia **do - while** opera de la siguiente manera: Primero se ejecuta *sentencia* y luego se evalúa *expresión*, si su valor es verdadero, se ejecuta *sentencia* y *expresión* es evaluada de nuevo, repitiéndose el ciclo hasta que *expresión* resulta falsa. Cuando esto último ocurre el ciclo termina y el programa continúa con la siguiente instrucción después del ciclo.

La diferencia básica entre las sentencias **do - while** y **while** es que la sentencia compuesta en el ciclo **do - while** se ejecuta por lo menos una vez, ya que la prueba sobre *expresión* se realiza hasta que la sentencia compuesta se ha ejecutado; mientras que la sentencia compuesta en la sentencia **while** no se ejecutará si *expresión* es falsa desde el principio.

Ejemplo sobre la sentencia do - while

El siguiente código muestra la clase Capital del ejemplo sobre la proposición while modificado para que utilice en su lugar una do-while.

```
/*
 * Capital3.java
 *
 * Creada el 11 de octubre de 2005, 12:36 PM
 */

package capital;

/**
 * Esta clase permite tabular el capital acumulado de una inversión,
 * a una tasa de interés anual, con recapitalización mensual.
 *
 * Ilustra el uso de la sentencia do-while
 *
 * @author mdomitsu
 */
public class Capital3 {
  private double capital;
```

```
  private double tasa;
  private int meses;

  public Capital3(double capital, double tasa, int meses) {
    this.capital = capital;
    this.tasa = tasa;
    this.meses = meses;
  }

  public void tabla() {
    int mes = 1;
    double capital = this.capital;

    System.out.println("Mes     Capital");
    do {
       capital *= (1 + tasa / 12);
       System.out.println(mes + ": " + capital);
       mes++;
    }
    while(mes <= meses);
  }
}
```

Sentencias de Salto

Hay ocasiones en que es conveniente que el programa salte de una instrucción a otra en forma incondicional, esto es, no sujeto a una condición como lo vimos con las sentencias de selección y de iteración. En Java existen cinco sentencias de salto: Las sentencias **break**, **continue**, **break etiquetada**, **continue etiquetada** y **return**.

Sentencia break

Cuando estudiamos la sentencia **switch** vimos que la **sentencia break** transfiere el control del programa a la siguiente sentencia después de la sentencia **switch**. Esto es, termina la ejecución de una sentencia **switch**. La **sentencia break** también se utiliza para terminar en forma incondicional la ejecución de cualquier sentencia de iteración. La sintaxis de esta sentencia es:

```
break;
```

La ejecución de una sentencia **break** se encuentra dentro de una sentencia **switch** o iterativa que a su vez está dentro de otra sentencia **switch** o iterativa, sólo termina la sentencia **switch** o iterativa más interna.

Ejemplo sobre la sentencia break

Un número es primo sólo si es divisible entre 1 y si mismo. Para determinar si un número n es primo podemos probar si es divisible entre i, donde i = 2, 3, ... , n-1. Si encontramos

que el número es divisible entre uno de los valores de i, el número ya no es primo y no es necesario seguir probando para los valores restantes de i.

El código siguiente muestra el método esPrimo() que determina si su argumento es primo o no.

```
/*
 * Primo.java
 *
 * Creada el 11 de octubre de 2005, 12:36 PM
 */

package primo;

import java.util.Vector;

/**
 * Esta clase permite determinar si un número entero es primo.
 * También determina los números primos que hay en un rango de números
 *
 * Ilustra el uso de las sentencias break y continue
 *
 * @author mdomitsu
 */
public class Primo {

  /**
   * Determina si un número es primo
   * @param n Número a determinar si es primo
   * @return true si el número es primo, false en caso contrario
   */
  public static boolean esPrimo(int n) {
    int i;

    // Prueba si sólo es divisible entre 1 y si mismo
    for(i = 2; i < n; i++)
      // Si el número es divisible entre i no es primo, rompe el ciclo.
      if(n % i == 0) break;

    // Si se recorrió todo el ciclo si es primo, falso en caso contrario
    if(i == n) return true;
    else return false;
  }

  ...
}
```

Sentencia continue

La **sentencia continue** sólo se utiliza con las sentencias de iteración. La sintaxis de esta sentencia es:

```
continue;
```

Al ejecutarse la sentencia **continue**, genera un salto al final de la última sentencia dentro del cuerpo de una sentencia de iteración. En las sentencias **while** y **do - while**, la sentencia **continue** hace que el programa salte a evaluar la expresión de control del ciclo. En la sentencia **for** la sentencia **continue** hace que el programa salte a evaluar la tercera expresión y luego la expresión de control del ciclo.

La ejecución de una sentencia **continue** que se encuentra dentro de una sentencia iterativa que a su vez está dentro de otra sentencia iterativa, sólo termina la sentencia iterativa más interna.

Ejemplo sobre la sentencia continue

El siguiente método de la clase Primo determina los números primos entre dos números.

```
/*
 * Primo.java
 *
 * Creada el 11 de octubre de 2005, 12:36 PM
 */

package primo;

import java.util.Vector;

/**
 * Esta clase permite determinar si un número entero es primo.
 * También determina los números primos que hay en un rango de números
 *
 * Ilustra el uso de las sentencias break y continue
 *
 * @author mdomitsu
 */
public class Primo {

  ...

  /**
   * Determina los números primos que hay en un rango de números
   * @param n1 Límite inferior del rango
   * @param n2 Límite superior del rango
   * @return Vector con los primos encontrados.
   */
  public static Vector primos(int n1, int n2) {
    int i, j;
    Vector listaPrimos = new Vector();

    // Prueba para cada número en el rango, si es primo
    for(j = n1; j <= n2; j++) {
      // Si no es primo, prueba el siguiente número
      if(!esPrimo(j))continue;

      // Es primo, agrégalo a la lista de primos
```

```
      else listaPrimos.add(new Integer(j));
    }

    return listaPrimos;
  }
}
```

Sentencia break Etiquetada

Ya se vio que la sentencia break sólo termina la sentencia **switch** o iterativa más interna que la contiene. Si deseamos salir de un conjunto de sentencias iterativas anidadas podemos usar una forma extendida de la sentencia **break**, la sentencia **break etiquetada**, cuya sintaxis es:

```
break etiqueta;
```

donde *etiqueta* es un identificador que identifica un bloque (sentencia compuesta) y debe contener a la sentencia **break etiquetada**,. Para identificar el bloque lo precedemos de la etiqueta y dos puntos:

```
etiqueta: bloque
```

Al ejecutarse la sentencia **break etiquetada**, el control del programa pasa a la siguiente instrucción después del bloque etiquetado.

Por ejemplo, en el siguiente código suma los números del 0 al 100. Cuando `i + j` toma el valor de 100, el control del programa termina la ejecución de ambos ciclos:

```
suma = 0;

bloque: for(j = 0; j < 20; j += 10) {
  for(i = 0; i < 10; i++) {
    suma += j + i;
    if(j + i == 100) break bloque;
  }
}
// Al ejecutarse la sentencia break bloque, el control del programa
// salta a la instrucción en la siguiente línea.
```

Sentencia continue Etiquetada

Un bloque etiquetado también puede contener una sentencia **continue etiquetada** cuya sintaxis es:

```
continue etiqueta;
```

donde *etiqueta* es un identificador que identifica al bloque etiquetado. Al ejecutarse la sentencia **continue etiquetada**, el control del programa pasa al final de la última sentencia dentro del bloque etiquetado.

Por ejemplo, en el siguiente código suma los números del 0 al 199 menos el 100. Cuando `i + j` toma el valor de 100, no se hace la suma:

```
suma = 0;

bloque: for(j = 0; j <= 20; j += 10) {
  for(i = 0; i < 10; i++) {
    if(j + i == 100) continue bloque;
    suma += j + i;
    // Al ejecutarse la sentencia continue bloque, el control
    // del programasalta aquí sin realizar la suma.
  }
}
```

Sentencia return

La sentencia **return** termina la ejecución de un método regresando el control del programa al método llamante. La sintaxis de la sentencia **return** es la siguiente:

```
return [expresión];
```

- Si el método es de tipo void, puede haber 0, 1 o más sentencias **return** y todas las sentencias aparecen sin *expresión.*

- Si un método es de un tipo diferente a void, debe tener al menos una sentencia return y todas las sentencias **return** del método llevan una *expresión.* Y el tipo de la expresión debe coincidir con el tipo del método.

- Aunque un método tenga varias sentencias **return**, sólo una de ellas se ejecuta.

- Si en el cuerpo de un método no hay una sentencia **return**, la ejecución del método termina después de ejecutar la última sentencia del método.

- Al ejecutar la sentencia **return** el control del programa pasa al método llamante. Si la sentencia return aparece con *expresión*, antes de regresar el control se evalúa *expresión* y se regresa su valor al método llamante.

Por ejemplo el siguiente método no tiene una sentencia **return** y por lo tanto termina después de ejecutar la última sentencia dentro del método:

```
public void setTitulo(String titulo) {
  this.titulo = titulo;
}
```

El siguiente método tiene cinco sentencias **return** y cómo el método es del tipo doble (regresa un valor de tipo doble), cada una de ellas va seguida de una expresión:

```
public double calculaCosto() {
  switch(tipoTelegrama.charAt(0)) {
    case 'O':
    case 'o':
      if (numPalabras <= 10)
        return costoOrdinario;
      else
        return costoOrdinario + costoAdicionalOrdinario *
               (numPalabras - 10);
    case 'U':
    case 'u':
      if (numPalabras <= 10)
        return costoUrgente;
      else
        return costoUrgente + costoAdicionalUrgente *
               (numPalabras - 10);
    default:
      return 0;
  }
}
```