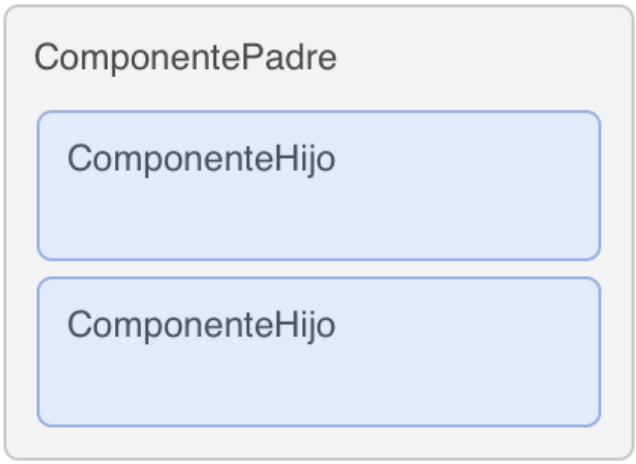


Hasta ahora, la comunicación entre componentes que hemos visto es la de padre a hijo, es decir, de un componente a los que renderiza. Cuando un componente quiere pasar un valor a otro que va a renderizar utiliza los props. De esta manera, el hijo podrá utilizar estos valores durante su ciclo de vida. Además, el padre podrá actualizar dichos valores cuando sea necesario.

Pero, ¿qué ocurre cuando es el hijo el que tiene que comunicarse con el padre?

# Hijo a padre

**La respuesta es la misma: utilizamos los props**. Para que un hijo sea capaz de comunicarse con su padre, este le envía distintos métodos que podrá ejecutar durante su ciclo de vida. Estos métodos podrán llamar a otras funciones en el padre así como actualizar su estado.

```
class ParentComponent extends React.Component {
  // Inicializamos el estado
  constructor(props) {
    super(props);

    this.state = {
      counter: 0
    }
  }

  // Este método se envía al hijo para que lo ejecute cuando el usuario
  // haga click en el botón
  onClick = () => {
    this.setState({ counter: this.state.counter + 1 });
  };

  render() {
    return <div>
      <p>Number of clicks: <b>{ this.state.counter }</b></p>
```

```
      <ChildComponent onClick={ this.onClick } />
    </div>;
  }
}

// La clase hijo no almacena estado, por lo que podemos definirla como un
// componente stateless
const ChildComponent = props => <button onClick={ props.onClick }>+ 1</button>;
```

Código en Codepen.

Recordad que siempre debemos de asociar los métodos al contexto del componente. Al definir `onClick` con el operador `=>` la asignación de `this` al contexto del componente padre va implícita. No obstante, si definimos el método `onClick` con la notación de `function`, necesitaremos llamar al método `bind` en el constructor o cuando pasemos la función al hijo:

```
// En el constructor
this.onClick = this.onClick.bind(this);
// Cuando pasamos el valor al hijo
<ChildComponent onClick={ this.onClick.bind(this) } />
```