Chapter 15: Communication Between Components

Section 15.1: Child to Parent Components

Sending data back to the parent, to do this we simply **pass a function as a prop from the parent component to the child component**, and **the child component calls that function**.

In this example, we will change the Parent state by passing a function to the Child component and invoking that function inside the Child component.

```
import React from 'react';

class Parent extends React.Component {
    constructor(props) {
        super(props);
        this.state = { count: 0 };

        this.outputEvent = this.outputEvent.bind(this);
    }
    outputEvent(event) {
        // the event context comes from the Child
        this.setState({ count: this.state.count++ });
    }

    render() {
        const variable = 5;
        return (
            <div>
                Count: { this.state.count }
                <Child clickHandler={this.outputEvent} />
            </div>
        );
    }
}

class Child extends React.Component {
    render() {
        return (
            <button onClick={this.props.clickHandler}>
                Add One More
            </button>
        );
    }
}

export default Parent;
```

Note that the Parent's `outputEvent` method (that changes the Parent state) is invoked by the Child's button `onClick` event.

Section 15.2: Not-related Components

The only way if your components does not have a parent-child relationship (or are related but too further such as a grand grand grand son) is to have some kind of a signal that one component subscribes to, and the other writes into.

Those are the 2 basic operations of any event system: **subscribe/listen** to an event to be notify, and **send/trigger/publish/dispatch** a event to notify the ones who wants.

There are at least 3 patterns to do that. You can find a comparison here.

Here is a brief summary:

- Pattern 1: **Event Emitter/Target/Dispatcher**: the listeners need to reference the source to subscribe.
    - to subscribe: `otherObject.addEventListener('click', () => { alert('click!'); });`
    - to dispatch: `this.dispatchEvent('click');`
- Pattern 2: **Publish/Subscribe**: you don't need a specific reference to the source that triggers the event, there is a global object accessible everywhere that handles all the events.
    - to subscribe: `globalBroadcaster.subscribe('click', () => { alert('click!'); });`
    - to dispatch: `globalBroadcaster.publish('click');`
- Pattern 3: **Signals**: similar to Event Emitter/Target/Dispatcher but you don't use any random strings here. Each object that could emit events needs to have a specific property with that name. This way, you know exactly what events can an object emit.
    - to subscribe: `otherObject.clicked.add( () => { alert('click'); });`
    - to dispatch: `this.clicked.dispatch();`

Section 15.3: Parent to Child Components

That the easiest case actually, very natural in the React world and the chances are - you are already using it.

You can **pass props down to child components**. In this example `message` is the prop that we pass down to the child component, the name message is chosen arbitrarily, you can name it anything you want.

```
import React from 'react';

class Parent extends React.Component {
    render() {
        const variable = 5;
        return (
            <div>
                <Child message="message for child" />
                <Child message={variable} />
            </div>
        );
    }
}

class Child extends React.Component {
    render() {
        return <h1>{this.props.message}</h1>
    }
}

export default Parent;
```

Here, the `<Parent />` component renders two `<Child />` components, passing `message for child` inside the first component and `5` inside the second one.

In summary, you have a component (parent) that renders another one (child) and passes to it some props.