CHILD COMPONENTS UPDATE THEIR SIBLINGS' PROPS

Child Components Update Sibling Components
Patterns within patterns within patterns!

In lesson 1, you learned your first React programming pattern: a *stateful*, parent component passes down a `prop` to a *stateless*, child component.

In lesson 2, you learned that lesson 1's pattern is actually part of a larger pattern: a stateful, parent component passes down an *event handler* to a stateless, child component. The child component then uses that *event handler* to update its parent's `state`.

In this lesson, we will expand the pattern one last time. A child component updates its parent's state, and the parent passes that state to a *sibling* component.

An understanding of this final pattern will be very helpful in the wild, not to mention in the next React course. Click Next and we'll build an example!

Instructions

In this video, the Like and Stats components are siblings under the Reactions parent component.

1. The Reactions component passes an event handler to the Like component.
2. When Like is clicked, the handler is called, which causes the parent Reactions component to send a new prop to Stats.
3. The Stats component updates with the new information.

One Sibling to Display, Another to Change

One of the very first things that you learned about components is that they should only have one job.

In the last lesson, `Child` had two jobs:

1 - `Child` displayed a name.

2 - `Child` offered a way to *change* that name.

You should divide `Child` in two: one component for displaying the name, and a different component for allowing a user to *change* the name.

In the code editor, select **Child.js**. Notice that these lines have vanished:

```
<h1>
  Hey, my name is {this.props.name}!
</h1>
```

The new version of `Child` renders a dropdown menu for *changing* the name, and *that's it*.

Select **Sibling.js** in the code editor.

Read through the render function's `return` statement.

Here, the name is displayed! Or at least it will be displayed, once you've done a little editing.

That brings us to the essential new concept for this lesson: you will have one stateless component *display* information, and a different stateless component offer the ability to *change* that information.

Instructions

1.
Click Run.

Select **Parent.js**, and look at the rendered `<Parent />` in the browser. Try selecting a name from the dropdown menu. Does it work?

Checkpoint 2 Passed

Hint
Run the code and try selecting a name from the dropdown menu. Does it work?

Parent.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Child } from './Child';
import { Sibling } from './Sibling';
```

```
class Parent extends React.Component {
  constructor(props) {
    super(props);

    this.state = { name: 'Frarthur' };

    this.changeName = this.changeName.bind(this);
  }

  changeName(newName) {
    this.setState({
      name: newName
    });
  }

  render() {
    return (
      <div>
        <Child
          name={this.state.name}
          onChange={this.changeName} />
        <Sibling />
      </div>
    );
  }
}

ReactDOM.render(
  <Parent />,
  document.getElementById('app')
);
```

Child.js

```
import React from 'react';

export class Child extends React.Component {
  constructor(props) {
    super(props);
```

```
    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(e) {
    const name = e.target.value;
    this.props.onChange(name);
  }

  render() {
    return (
      <div>
        <select
          id="great-names"
          onChange={this.handleChange}>

          <option value="Frarthur">Frarthur</option>
          <option value="Gromulus">Gromulus</option>
          <option value="Thinkpiece">Thinkpiece</option>
        </select>
      </div>
    );
  }
}
```

Sibling.js

```
import React from 'react';

export class Sibling extends React.Component {
  render() {

    return (
      <div>
        <h1>Hey, my name is Frarthur!</h1>
        <h2>Don't you think Frarthur is the prettiest name e
ver?</h2>
        <h2>Sure am glad that my parents picked Frarthur!</h
2>
      </div>
    );
```

```
  }
}
```

Pass the Right props to the Right Siblings

Look at **Parent.js** in the code editor.

Three things have changed in this file since the last Lesson:

1. `Sibling` has been required on line 4.
2. A `<Sibling />` instance has been added to the render function on line 27.
3. `<Sibling />` and `<Child />` have been wrapped in a `<div></div>`, since JSX expressions must have only one outer element.

Instructions

1.

`Sibling`'s job is to display the selected name.

That name is *stored* in `Parent`'s `state`. `Parent` will have to pass the name to `Sibling`, so that `Sibling` can display it.

In `Parent`'s render function, pass the name to `<Sibling />`:

```
name={this.state.name}
Checkpoint 2 Passed
```

Hint

`<Sibling />` starts looking like this:

```
<Sibling />
```

Find that line, and change it to:

```
<Sibling name={this.state.name} />
```

2.

`Child`'s job is to offer a way to *change* the chosen name. *Not* to display it!

In **Parent.js**, look at lines 25 and 26.

You are passing `Child` two pieces of dynamic information:

1. The chosen name (`this.state.name`).
2. A way to *change* the chosen name (`this.changeName`).

One of those is now unnecessary!

Remove an attribute from `<Child />`, so that it receives a way to *change* the chosen name and nothing more.

Checkpoint 3 Passed

Hint

In **Parent.js**, `<Child />` is currently being rendered with two attributes: `name` and `onChange`. Find the `name` attribute and remove it.

Parent.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Child } from './Child';
import { Sibling } from './Sibling';

class Parent extends React.Component {
  constructor(props) {
    super(props);

    this.state = { name: 'Frarthur' };

    this.changeName = this.changeName.bind(this);
  }

  changeName(newName) {
    this.setState({
      name: newName
    });
  }

  render() {
    return (
      <div>
        <Child
```

```
          onChange={this.changeName} />
        <Sibling
          name={this.state.name}
        />
      </div>
    );
  }
}

ReactDOM.render(
  <Parent />,
  document.getElementById('app')
);
```

Child.js

```
import React from 'react';

export class Child extends React.Component {
  constructor(props) {
    super(props);

    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(e) {
    const name = e.target.value;
    this.props.onChange(name);
  }

  render() {
    return (
      <div>
        <select
          id="great-names"
          onChange={this.handleChange}>

          <option value="Frarthur">Frarthur</option>
          <option value="Gromulus">Gromulus</option>
          <option value="Thinkpiece">Thinkpiece</option>
```

```
        </select>
      </div>
    );
  }
}
```

Sibling.js

```
import React from 'react';

export class Sibling extends React.Component {
  render() {

    return (
      <div>
        <h1>Hey, my name is Frarthur!</h1>
        <h2>Don't you think Frarthur is the prettiest name e
ver?</h2>
        <h2>Sure am glad that my parents picked Frarthur!</h
2>
      </div>
    );
  }
}
```

Display Information in a Sibling Component

You're on the last step!

You've passed the name down to `<Sibling />` as a `prop`. Now `<Sibling />` has to *display* that `prop`.

Instructions

1.
Select **Sibling.js**.

Notice that `Sibling` doesn't use any dynamic information at all. Every time that `Sibling` renders, it will always look exactly the same. That's not what you want!

On line 5, declare a new variable called `name`. Set `name` equal to `this.props.name`. You'll use it in the next checkpoint.

Checkpoint 2 Passed

Hint

Find the line that contains `render() {`. Then, find the line that starts with `return`.

You'll declare your variable in between those two lines.

2.

Inside of the render function's `return` statement, there are three instances of the word `Frarthur`.

You want to replace each instance of `Frarthur` with *whatever the chosen name is*.

Replace each `Frarthur` with `{name}`. Click Run. Try selecting a new name from the dropdown menu.

Checkpoint 3 Passed

Hint

Find all occurrences of `Frarthur` and replace it with `{name}`.

Sibling.js

```
import React from 'react';

export class Sibling extends React.Component {
  render() {
    const name = this.props.name;
    return (
      <div>
        <h1>Hey, my name is {name}!</h1>
        <h2>Don't you think {name} is the prettiest name eve
r?</h2>
        <h2>Sure am glad that my parents picked {name}!</h2>
      </div>
    );
  }
}
```

Parent.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Child } from './Child';
import { Sibling } from './Sibling';

class Parent extends React.Component {
  constructor(props) {
    super(props);

    this.state = { name: 'Frarthur' };

    this.changeName = this.changeName.bind(this);
  }

  changeName(newName) {
    this.setState({
      name: newName
    });
  }

  render() {
    return (
      <div>
        <Child
          onChange={this.changeName} />
        <Sibling
          name={this.state.name}
        />
      </div>
    );
  }
}

ReactDOM.render(
  <Parent />,
  document.getElementById('app')
);
```

Child.js

```
import React from 'react';

export class Child extends React.Component {
  constructor(props) {
    super(props);

    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(e) {
    const name = e.target.value;
    this.props.onChange(name);
  }

  render() {
    return (
      <div>
        <select
          id="great-names"
          onChange={this.handleChange}>

          <option value="Frarthur">Frarthur</option>
          <option value="Gromulus">Gromulus</option>
          <option value="Thinkpiece">Thinkpiece</option>
        </select>
      </div>
    );
  }
}
```

Stateless Components Inherit From Stateful Components Recap
You just executed your first complete React programming pattern!

Let's review. Follow each step in the code editor:

- A *stateful* component class defines a function that calls `this.setState`. (**Parent.js**, lines 15-19)

- The stateful component passes that function down to a stateless component. (**Parent.js**, line 24)
- That *stateless* component class defines a function that calls the passed-down function, and that can take an *event object* as an argument. (**Child.js**, lines 10-13)
- The stateless component class uses this new function as an event handler. (**Child.js**, line 20)
- When an event is detected, the parent's state updates. (A user selects a new dropdown menu item)
- The stateful component class passes down its state, distinct from the ability to *change* its state, to a different stateless component. (**Parent.js**, line 25)
- That stateless component class receives the state and displays it. (**Sibling.js**, lines 5-10)
- An instance of the stateful component class is rendered. One stateless child component displays the `state`, and a different stateless child component displays a way to *change* the `state`. (**Parent.js**, lines 23-26)

This pattern occurs in React all the time! The more that you see it, the more that its elegance will become clear.

Being introduced to this pattern is your first step towards understanding how React apps fit together! You'll get more practice using it throughout this course, as well as in the course after this one.

`Parent.js`

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Child } from './Child';
import { Sibling } from './Sibling';

class Parent extends React.Component {
  constructor(props) {
    super(props);

    this.state = { name: 'Frarthur' };

    this.changeName = this.changeName.bind(this);
  }

  changeName(newName) {
    this.setState({
      name: newName
    });
  }
```

```
  render() {
    return (
      <div>
        <Child onChange={this.changeName} />
        <Sibling name={this.state.name} />
      </div>
    );
  }
});

ReactDOM.render(
  <Parent />,
  document.getElementById('app')
);
```

Child.js

```
import React from 'react';

export class Child extends React.Component {
  constructor(props) {
    super(props);

    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(e) {
    const name = e.target.value;
    this.props.onChange(name);
  }

  render() {
    return (
      <div>
        <select
          id="great-names"
          onChange={this.handleChange}>

          <option value="Frarthur">Frarthur</option>
```

```
          <option value="Gromulus">Gromulus</option>
          <option value="Thinkpiece">Thinkpiece</option>
        </select>
      </div>
    );
  }
}
```

Sibling.js

```
import React from 'react';

export class Sibling extends React.Component {
  render() {
    const name = this.props.name;
    return (
      <div>
        <h1>Hey, my name is {name}!</h1>
        <h2>Don't you think {name} is the prettiest name ever?</h2>
        <h2>Sure am glad that my parents picked {name}!</h2>
      </div>
    );
  }
}
```