Stateless Components Inherit From Stateful Components

Let's learn our first *programming pattern!*

In this lesson, we'll take a look at a simple version of a programming pattern. The following lessons will expand upon that lesson, and by the end, we'll have a programming pattern in its full complexity.

Our programming pattern uses two React components: a *stateful* component, and a *stateless* component. "Stateful" describes any component that has a `state` property; "stateless" describes any component that does not.

In our pattern, a *stateful* component passes its `state` down to a *stateless* component.

Click Next to walk through an example!

Instructions

In this video, in the "Before" scene, the user interface is defined by a single, complex component.

In the "After" scene, the user interface is defined by a clear hierarchy of components in which the stateful component, Post, passes state information to stateless components, like User info, Content, and Stats.

Build a Stateful Component Class

Let's make a *stateful* component pass its `state` to a *stateless* component.

To make that happen, you need two component classes: a *stateful* class, and a *stateless* class.

Instructions

1.

We'll build the stateful class first.

On line 1 of **Parent.js**, `import` the React.js library. Store the library in a variable named `React`.

On line 2, `import` the ReactDOM library. Store it in a variable named `ReactDOM`.

Make lines 3 and 4 empty. You'll add code to line 3 later.

On line 5, declare a new `Parent` component. `Parent` will represent your *stateful* component class.

Ensure `Parent` `extends` `React.Component`. Add this method to your `Parent` component class:

```
render() {
  return <div></div>;
}
```

Checkpoint 2 Passed

Hint

Lines 1 and 2 will import the libraries that you need. To import React, use `import React from 'react';`. You'll also need to import `'react-dom'` and store it in the `ReactDOM` variable.

On line 5, you'll declare a new `Parent` component that renders an empty `<div>`. Here's a similar-looking component that renders an empty `<span>`:

```
class MyComponent extends React.Component {
  render() {
    return <span></span>;
  }
}
```

2.

Since `Parent` is supposed to be *stateful*, it will need to set its initial state. That means that it will need a constructor method.

Before the render method, give `Parent` a method named `constructor`. Give `constructor` one parameter named `props`.

Inside of `constructor()`'s body, call `super(props)`. On the next line, still inside of `constructor()`'s body, declare a property named `this.state` set equal to `{ name: 'Frarthur' }`.

Checkpoint 3 Passed

Hint

Here’s a similar component that sets a similar state.

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { name: 'Alex' };
  }
  render() {
    return <span></span>;
  }
}
```

Your component will probably look similar to this.

Parent.js

```
import React from 'react';
import ReactDOM from 'react-dom';


class Parent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { name: 'Frarthur' };
  }
  render() {
    return <div></div>;
  }
}
```

Build a Stateless Component Class
Great! You just made a *stateful* component class named `Parent`.

Now, let’s make our *stateless* component class.

Instructions

1.
Select **Child.js**.

On line 1, `import` the React.js library. Store the library in a variable named `React`.

Leave line 2 blank. On line 3, declare a new component named `Child`. `Child` will represent your *stateless* component class.

Add the following method to your `Child` component class:

```
render()                                                    {
  return                                                  <h1></h1>;
}
```

Checkpoint 2 Passed

Hint

You'll need to import React in this file. That will look very similar to how you did it in **Parent.js**.

`<Child>` will look similar to `<Parent>`, but without the constructor. Again, you can can refer to **Parent.js** for an example.

2.

`Child` is going to receive a `prop` called `name`, and display that `prop` on the screen.

How can you make a component *display* a `prop` called `name`?

- To access a `prop`, use the expression `this.props.name-of-prop`.
- To make a component *display* something, include that thing in the render function's `return` statement.

You need to include `this.props.name` inside of `Child`'s render function's `return` statement.

Add this expression in between the `<h1></h1>` tags:

```
Hey, my name is {this.props.name}!
```

Checkpoint 3 Passed

Hint

In `<Child>`'s `render()` function, there should be an empty `<h1>` element. But not for long! You'll fill it in with `Hey, my name is {this.props.name}!`.

3.

A `<Parent />` is going to pass a `prop` to a `<Child />`.

That means that a `<Parent />` is going to *render* a `<Child />`. Rendering is the only way for a component to pass `props` to another component.

Any component rendered by a different component must be included in an `export` statement.

On line 3, put the word `export` before the word `class`, so that the line begins: `export class Child`.

Checkpoint 4 Passed

Hint

In **Child.js**, put the word `export` before the word `class`, so that the line begins with `export class Child`.

4.

That's it! `Child` is ready to inherit a `prop` and display it.

Child.js

```
import React from 'react';

export class Child extends React.Component {
  render() {
    return <h1>Hey, my name is {this.props.name}!</h1>;
  }
}
```

Pass a Component's State

A `<Parent />` is supposed to pass its `state` to a `<Child />`.

Before a `<Parent />` can pass anything to a `<Child />`, you need to `import` `Child` into **Parent.js**.

Instructions

1.

To import a local component, we will need to modify our `import` syntax to use local files and named exports. For example, if we wanted to import a component called `ComponentName` from a local file called **Component.js** we would write

```
import { ComponentName } from './Component';
```

On line 3, import the `Child` component from **Child.js**.

Parent.js and **Child.js** share the same parent directory.

Checkpoint 2 Passed

Hint

To import `MyComponent` from `./MyComponent.js`, you'd write something like this:

```
import { MyComponent } from './MyComponent';
```

Your import of `Child` will look similar.

2.

Great! Now `Parent` is ready to pass its `state` to a `<Child />`.

Inside of `Parent`'s `.render()` method's `return` statement, get rid of the `<div></div>`.

Replace it with a `<Child />` instance.

Give `<Child />` an attribute with a *name* of `name`. The attribute's *value* should be the `name` property stored in `this.state`.

Checkpoint 3 Passed

Hint

`<Parent>` should render a `<Child>` instead of an `<h1>`.

To render `<Child>` with the right prop, you might do something like this:

```
return <Child name={this.state.name} />;
```

3.

All that's left is to render your components!

At the bottom of **Parent.js**, call `ReactDOM.render();`.

For `ReactDOM.render()`'s first argument, pass in `<Parent />`.

For `ReactDOM.render`'s second argument, pass in `document.getElementById('app')`.

Rendering `<Parent />` will render *both* components, because `Parent`'s render function returns a `<Child />`. Click Run, and see the rendered information that you passed down from `Parent`.

Checkpoint 4 Passed

Hint

To render a component called `<MyComponent />`, you'd write something like this at the bottom of your file:

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

Parent.js

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

class Parent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { name: 'Frarthur' };
  }
  render() {
    return (
      <Child
        name={this.state.name}
      />
    )
  }
}

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

Don't Update props

Great work! You just passed information from a *stateful* component to a *stateless* component. You will be doing a lot of that.

You learned earlier that a component can *change* its state by calling `this.setState()`. You may have been wondering: how does a component *change* its props?

The answer: it doesn't!

A component should never update `this.props`. Look at **Bad.js** to see an example of what not to do.

This is potentially confusing. `props` and `state` store *dynamic* information. Dynamic information can change, by definition. If a component can't change its `props`, then what are `props` for?

A React component should use `props` to store information that can be changed, but can only be changed by a *different* component.

A React component should use `state` to store information that the component itself can change.

If that's a bit confusing, don't worry! The next two lessons will be examples.

Bad.js

```
import React from 'react';

class Bad extends React.Component {
  render() {
    this.props.message = 'yo'; // NOOOOOOOOOOOOO!!!
    return <h1>{this.props.message}</h1>;
  }
}
```