23. Class-based Comonents

github repo

React: Class-Based Components

- ES6 Classes
- Functional Components
- Class-Based Components
- Props
- State
- Events
- Lifecycle Methods

ES6 Classes

The `class` keyword is used to state we are building a class (an object blueprint.) Class names are usually singular and PascalCased. Basic class with property:

```
class MyClass {
    constructor() {
        this.property = "value";
    }
}
```

If creating a similar class, you can let JS know that the class has a parent blueprint via the `extends` keyword. Note that `super` is used to execute the parent class's constructor method, you are able to pass in arguments if needed.

```
class MyClass {
    constructor(val1) {
        this.prop1 = val1;
    }
}

class MySecondClass extends MyClass {
    constructor(val1, val2) {
        super(val1); // Run constructor with val1 passed in.
        this.prop2 = val2;
    }
}
```

To create an instance of this class (object blueprint), you must use the `new` keyword. For example:

```
myObj = new MyClass('Hello, World!');
console.log(myObj.prop1); // "Hello, World!"
```

Functional Components

What you're used to! This is the modern way to create React apps, hooks are intended to be the future. You've experienced first-hand how powerful custom hooks are, and how much they can assist in organization and slimming down component code. Newer companies, or very progressive ones, will have adopted this way of doing things.

```
// Basically a regular function.
function MyComponent {
    // Must return output.
    return (<p>Hello, World!</p>);
}
```

Class-Based Components

These are still supported, and very feature-full. They provide good functionality, and ensure critical methods all live inside of the component. The lifecycle is more clear and easy to poke at, and the structure will feel more natural for those that strive for an object-oriented approach to their problem-solving. Many businesses that haven't had a chance to update their approach, or their app is too far along to afford changing everything, continue to use this style of component. It isn't a wrong choice, and is still very common. It lends itself well to organizations and businesses that utilize TypeScript in their JavaScript stack.

```
import { Component } from 'react';

class MyComponent extends Component {
    constructor() {
        super(); // Execute constructor from React's "Component" class.
    }

    render() { // Must return output.
        return (<p>Hello, World!</p>);
    }
}
```

Props

Include `super()` in your component's `constructor()` method, and props will automatically be pulled in for your convenience. Props are accessed throughout the component via `this.props`. `this.props.children` can be used to grab any children components or content included inside of the component tags, should it be used in a non-self-closing fashion.

State

State should still not be edited directly☺afterall, React needs to know when state is updated so that it can, well, react! Default state values should be added to the `this.state` object in the `constructor()` method. To update state, pass in an object with a matching (or new) property name and the new value to the `this.setState({yourStatePropertyName: "Your state value."})` method.

Events

Watch out for the context of `this`, as its value changes depending on when and where JavaScript code is being run. This can be avoided a few ways, two we looked at today include...

```
// ...inside of your constructor:
this.methodName = this.methodName.bind(this); // Binds the keyword, so it can be used following the class context instead of event or otherwise.
```

```
// ...inside the root of your class:
methodName = () => { // Arrow functions and methods don't overwrite the context of "this".

}
```

Lifecycle Methods

- `componentDidMount` : Method that runs when the component is initialized (has been created and added to the browser.) Used often for intitial API request and interval set-up.
- `componentDidUpdate` : Method that runs anytime data in the component changes and re-rendered.
- `componentWillUnmount` : Method that runs just before the component is destroyed. Do your clean-up here! Consider removing any listeners, intervals, etc, that cannot be used once the component is gone.

Links and Resources

- Lifecycle Methods Diagram
- JS Class Reference
- super()
- Functional Components versus Class Components in React

View on Compass | Leave Feedback