Chapter 13: React.createClass vs extends React.Component

Section 13.1: Create React Component

Let's explore the syntax differences by comparing two code examples.

React.createClass (deprecated)

Here we have a `const` with a React class assigned, with the `render` function following on to complete a typical base component definition.

```
import React from 'react';

const MyComponent = React.createClass({
  render() {
    return (
      <div></div>
    );
  }
});

export default MyComponent;
```

React.Component

Let's take the above React.createClass definition and convert it to use an ES6 class.

```
import React from 'react';

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

export default MyComponent;
```

In this example we're now using ES6 classes. For the React changes, we now create a class called **MyComponent** and extend from React.Component instead of accessing React.createClass directly. This way, we use less React boilerplate and more JavaScript.

PS: Typically this would be used with something like Babel to compile the ES6 to ES5 to work in other browsers.

Section 13.2: "this" Context

Using React.createClass will automatically bind `this` context (values) correctly, but that is not the case when using ES6 classes.

React.createClass

Note the `onClick` declaration with the `this.handleClick` method bound. When this method gets called React will apply the right execution context to the `handleClick`.

```
import React from 'react';

const MyComponent = React.createClass({
  handleClick() {
    console.log(this); // the React Component instance
  },
  render() {
    return (
      <div onClick={this.handleClick}></div>
    );
  }
});

export default MyComponent;
```

React.Component

With ES6 classes `this` is `null` by default, properties of the class do not automatically bind to the React class (component) instance.

```
import React from 'react';

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  handleClick() {
    console.log(this); // null
  }
  render() {
    return (
      <div onClick={this.handleClick}></div>
    );
  }
}

export default MyComponent;
```

There are a few ways we could bind the right `this` context.

Case 1: Bind inline:

```
import React from 'react';

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  handleClick() {
    console.log(this); // the React Component instance
  }
  render() {
    return (
      <div onClick={this.handleClick.bind(this)}></div>
    );
  }
}

export default MyComponent;
```

Case 2: Bind in the class constructor

Another approach is changing the context of `this.handleClick` inside the `constructor`. This way we avoid inline repetition. Considered by many as a better approach that avoids touching JSX at all:

```
import React from 'react';

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    console.log(this); // the React Component instance
  }
  render() {
    return (
      <div onClick={this.handleClick}></div>
    );
  }
}

export default MyComponent;
```

Case 3: Use ES6 anonymous function

You can also use ES6 anonymous function without having to bind explicitly:

```
import React from 'react';

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  handleClick = () => {
    console.log(this); // the React Component instance
  }
  render() {
    return (
      <div onClick={this.handleClick}></div>
    );
  }
}

export default MyComponent;
```

Section 13.3: Declare Default Props and PropTypes

There are important changes in how we use and declare default props and their types.

React.createClass

In this version, the `propTypes` property is an Object in which we can declare the type for each prop. The `getDefaultProps` property is a function that returns an Object to create the initial props.

```
import React from 'react';

const MyComponent = React.createClass({
  propTypes: {
    name: React.PropTypes.string,
    position: React.PropTypes.number
  },
  getDefaultProps() {
    return {
      name: 'Home',
      position: 1
    };
  },
  render() {
    return (
      <div></div>
    );
  }
});

export default MyComponent;
```

React.Component

This version uses `propTypes` as a property on the actual **MyComponent** class instead of a property as part of the `createClass` definition Object.

The `getDefaultProps` has now changed to just an Object property on the class called defaultProps, as it's no longer a "get" function, it's just an Object. It avoids more React boilerplate, this is just plain JavaScript.

```
import React from 'react';

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div></div>
    );
  }
}
MyComponent.propTypes = {
  name: React.PropTypes.string,
  position: React.PropTypes.number
};
MyComponent.defaultProps = {
  name: 'Home',
  position: 1
};

export default MyComponent;
```

Additionally, there is another syntax for `propTypes` and `defaultProps`. This is a shortcut if your build has ES7 property initializers turned on:

```
import React from 'react';

class MyComponent extends React.Component {
  static propTypes = {
    name: React.PropTypes.string,
    position: React.PropTypes.number
  };
  static defaultProps = {
    name: 'Home',
    position: 1
  };
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div></div>
    );
  }
}

export default MyComponent;
```

Section 13.4: Mixins

We can use `mixins` only with the React.createClass way.

React.createClass

In this version we can add `mixins` to components using the mixins property which takes an Array of available mixins. These then extend the component class.

```
import React from 'react';

var MyMixin = {
  doSomething() {

  }
};
const MyComponent = React.createClass({
  mixins: [MyMixin],
  handleClick() {
    this.doSomething(); // invoke mixin's method
  },
  render() {
    return (
      <button onClick={this.handleClick}>Do Something</button>
    );
  }
});

export default MyComponent;
```

React.Component

React mixins are not supported when using React components written in ES6. Moreover, they will not have support for ES6 classes in React. The reason is that they are considered harmful.

Section 13.5: Set Initial State

There are changes in how we are setting the initial states.

React.createClass

We have a `getInitialState` function, which simply returns an Object of initial states.

```
import React from 'react';

const MyComponent = React.createClass({
  getInitialState () {
    return {
      activePage: 1
    };
  },
  render() {
    return (
      <div></div>
    );
  }
});

export default MyComponent;
```

React.Component

In this version we declare all state as a simple **initialisation property in the constructor**, instead of using the `getInitialState` function. It feels less "React API" driven since this is just plain JavaScript.

```
import React from 'react';

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      activePage: 1
    };
  }
  render() {
    return (
      <div></div>
    );
  }
}

export default MyComponent;
```

Section 13.6: ES6/React “this” keyword with ajax to get data from server

```
import React from 'react';

class SearchEs6 extends React.Component{
    constructor(props) {
        super(props);
        this.state = {
            searchResults: []
        };
    }

    showResults(response){
        this.setState({
            searchResults: response.results
        })
    }

    search(url){
        $.ajax({
            type: "GET",
            dataType: 'jsonp',
            url: url,
            success: (data) => {
                this.showResults(data);
            },
            error: (xhr, status, err) => {
                console.error(url, status, err.toString());
            }
        });
    }

    render() {
        return (
            <div>
                <SearchBox search={this.search.bind(this)} />
                <Results searchResults={this.state.searchResults} />
            </div>
        );
    }
}
```