- ⏱️ 7 minutes
- ✅ Completed

Refactoring Class Components

With React version 16, Class Components were replaced by Function Components. However, many code bases still use Class Components in parts that have not been completed migrated to use Function Components. You will still see Class Components being used in React, and one of your regular tasks as a developer will be to convert Class Components into Function Components.

When you finish this article, you should be able to:

- Recognize a React Class Component
- Convert the use of component props in a Class Component to a Function Component
- Convert the use of component state in a Class Component to a Function Component using `useState`
- Understand what lifecycle methods are, the types of lifecycle methods, and when they are called when a Class Component is rendered
- Convert the use of lifecycle methods in a Class Component to a Function Component using `useEffect`

What are Class Components?

Before React version 16, Function Components were used only for components that (1) didn't need component state, and/or (2) didn't need lifecycle methods (similar to `useEffect`). Otherwise Class Components needed to be used if you wanted any of those functionalities. With React version 16 came React hooks. Hooks allowed Function Components to have component state using `useState` and imitate lifecycle methods using `useEffect`.
A Class Component is a component that is defined using JavaScript `Class` syntax and extends the `React.Component` class from the `react` package.

Rendering elements

To render elements in a Class Component, you must define a `render` method on the class. The return value of the `render` method is what will be rendered by the component. Here's an example of converting a Class Component's `render` method into a Function Component

```
import React from 'react';

class ClassComponent extends React.Component {
  render() {
    return (
      <div></div>
    );
  }
}
function FunctionComponent() {
  return (
    <div></div>
  );
}
```

Component props

To access props in a Class Component, you can get the `props` object on the instance of the class.
Here's an example of accessing the `title` prop and rendering it in a Class Component:

```
import React from 'react';

class ClassComponent extends React.Component {
  render() {
    return (
      <>
        <h1>{this.props.title}</h1>
      </>
    );
  }
}
```

You can convert it to a Function Component like so:

```
function FunctionComponent({ title }) {
  return (
    <>
```

```
      <h1>{title}</h1>
    </>
  );
}
```

If the `React.Component`'s `constructor` method is overwritten with a `constructor` method defined on the component class, the first argument of the `constructor` method is passed into `React.Component`'s `constructor` method. You'll see this pattern in Class Components often because Class Components must be initialized with props this way to have proper access to props within the rest of the component.

Here's an example of the pattern:

```
import React from 'react';

class ClassComponent extends React.Component {
  constructor(props) {
    super(props); // must be called if creating a constructor
method
  }

  render() {
    return (
      <>
        <h1>{this.props.title}</h1>
      </>
    );
  }
}
```

Component state

Now let's take a look at how component state in a Class Component is initialized and used.

In a Class Component, component state must be initialized in the `constructor` method on the class. The component state is always an object with its keys being the state variables that you would normally create using `useState`. To manipulate state key's values, you must use a single method on the class called `setState`. `setState` takes in an object with the key value pairs to change on the class's state object.

Here's an example of converting a Class Component with component state into a Function Component.

```
import React from 'react';

class ClassComponent extends React.Component {
  constructor(props) {
    super(props); // must be called if creating a constructor
method

    // Initialize the component state object
    this.state = {
      count: 0
    };
  }

  render() {
    return (
      <>
        <h1>{this.props.title}</h1>
        <div>{count}</div>
        <button onClick={() => this.setState((state) => ({ count:
state.count + 1 }))}>
          Increment
        </button>
      </>
    );
  }
}
import { useState } from 'react';

function FunctionComponent({ title }) {
  const [count, setCount] = useState(0)

  return (
    <>
      <h1>{title}</h1>
      <div>{count}</div>
      <button onClick={() => setCount(count + 1)}>
        Increment
      </button>
    </>
  );
}
```

Lifecycle methods

Lifecycle methods of a Class Component are methods that will be invoked after the rendering of a component. There are three types of lifecycle methods. `componentDidMount` will only run once, after the component's first render. `componentDidUpdate` will run after every render that isn't the first render. `componentWillUnmount` will run right before the component is removed from the component tree. The `useEffect` hook can be used to imitate the behavior of the lifecycle methods for a Class Component.
Here's an example of converting a Class Component with lifecycle methods into a Function Component with `useEffect`.

```
import React from 'react';

class ClassComponent extends React.Component {
  constructor(props) {
    super(props); // must be called if creating a constructor
method

    // Initialize the component state object
    this.state = {
      count: 0
    };
  }

  componentDidMount() {
    setTimeout(() => {
      this.setState({ count: 0 });
    }, 1000);
  }

  componentDidUpdate(prevProps, prevState) {
    if (prevState.count !== this.state.count) {
      console.log('hello world!');
    }
  }

  componentWillUnmount() {
    console.log('cleanup')
  }

  render() {
    return (
      <>
        <div>{count}</div>
        <button onClick={() => this.setState((state) => ({ count:
state.count + 1 }))}>
          Increment
        </button>
      </>
    );
  }
```

```
}
import { useState, useEffect } from 'react';

function FunctionComponent({ title }) {
  const [count, setCount] = useState(0);

  useEffect(() => {
    setTimeout(() => {
      setCount(0);
    }, 1000);
    return () => console.log('cleanup');
  }, []);

  useEffect(() => {
    console.log('hello world!');
  }, [count]);

  return (
    <>
      <h1>{title}</h1>
      <div>{count}</div>
      <button onClick={() => setCount(count + 1)}>
        Increment
      </button>
    </>
  );
}
```

The lifecycle method `componentDidMount` will run once and after the first render of the `ClassComponent`. This function gets converted into a `useEffect` call in the `FunctionComponent` with an empty dependency array. A `useEffect` function with an empty dependency array will be called after the first render, just like the `componentDidMount` lifecycle method.
`componentDidUpdate` runs after every single render besides the first render. An almost equivalent `useEffect` call is created that will run after the first render and after every single re-render that has a change in the `count` state variable. If you take a look at the `componentDidUpdate` function, there is a conditional comparing the previous `count` state to the current `count` state. This is converted by the `useEffect` call into the dependency array with the `count` state variable as a dependency.
`componentWillUnmount` function runs right before the component is removed from the component tree. The `useEffect` equivalent of this is the return function of the `useEffect` function with an empty dependency array.

Understanding when a Class Component will run a lifecycle method is the key to being able to convert it effectively!

What you've learned

In this lesson, you learned how to recognize a React Class Component in older React code. You also learned how to convert the use of component props and component state in a Class Component to a Function Component. Finally, you learned what lifecycle methods are in a Class Component and how to convert them into `useEffect` calls in a Function Component.

Did you find this lesson helpful?

No

Yes

Continue To Next Page →

Good job! You've previously marked this page as completed. 🎉