React Class Components

React has changed a lot since 2013. The iterations of its library, how React applications are written, and especially its components have all changed drastically. However, many React applications were built over the last few years, so not everything was created with the current status quo in mind. This section of the book covers React's legacy.

I won't cover all that's considered legacy in React, because some features have been revamped more than once. You may see the previous iteration of the feature in older React applications, but will probably be different than the current.

Throughout this section we will compare a **modern React application** to its **legacy version**. We'll discover that most differences between modern and legacy React are due to class components versus function components.

> **Note:** The live execution of modern React application and its legacy version is shown at the end of this chapter.

React Class Components

React components have undergone many changes, from **createClass components** over **class components**, to **function components**. Going through a React application today, it's likely that we'll see class components next to the modern function components.

```
class InputWithLabel extends React.Component {
  render() {
    const {
      id,
      value,
      type = 'text',
      onInputChange,
      children,
```

```
    children,
  } = this.props;


  return (
    <>
      <label htmlFor={id}>{children}</label>
       
      <input
        id={id}
        type={type}
        value={value}
        onChange={onInputChange}
      />
    </>
  );
 }
}
```

src/App.js

A typical class component is a JavaScript class with a mandatory **render method** that returns the JSX. The class extends from a `React.Component` to inherit (class inheritance) all React's component features (e.g. state management for state, lifecycle methods for side-effects). React props are accessed via the class instance (`this`).

For a while class components were the popular choice for writing React applications. Eventually, function components were added, and both co-existed with their distinct purposes side by side:

```
const InputWithLabel = ({
  id,
  value,
  type = 'text',
  onInputChange,
  children,
}) => (
  <>
    <label htmlFor={id}>{children}</label>
     
    <input
      id={id}
      type={type}
      value={value}
      onChange={onInputChange}
    />
  </>
);
```

src/App.js

If no side-effects and no state were used in legacy apps, we'd use a function

component instead of a class component. Before 2018–before React Hooks were introduced–React's function components couldn't handle side-effects (`useEffect` hooks) or state (`useState`/`useReducer` hooks). As a result, these components were known as **functional stateless components**, there only to input props and output JSX. To use state or side-effects, it was necessary to refactor from a function component to a class component. When neither state nor side-effects were used, we used class components or the more lightweight function component.

With the addition of React Hooks, function components worked the same as class components, with state and side-effects. And since there was no longer any practical difference between them, the community chose function components since they are more lightweight.

Exercises:

- Read more about JavaScript Classes.
- Read more about how to refactor from a class component to a function component.
- Learn more about a different class component syntax which wasn't popular but more effective.
- Read more about class components in depth.
- Read more about other legacy and modern component types in React.