React Class Components: State

Before React Hooks, class components were superior to function components because they could be stateful. With a class constructor, we can set an initial state for the component. Also, the component's instance (`this`) gives access to the current state (`this.state`) and the component's state updater method (`this.setState`):

```
class App extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      searchTerm: 'React',
    };
  }

  render() {

    const { searchTerm } = this.state;
    return (
      <div>
        <h1>My Hacker Stories</h1>

        <SearchForm
          searchTerm={searchTerm}
          onSearchInput={() => this.setState({
            searchTerm: event.target.value
          })}
        />
      </div>
    );
  }
}
```

src/App.js

If the state has more than one property in its state object, the `setState` method performs only a shallow update. Only the properties passed to `setState` are overwritten, and all other properties in the state object stay intact. Since state management is important for frontend applications, there was no way around class components without hooks for function components.

In a React class component, there are two dedicated APIs (`this.state` and `this.setState`) to manage a component's state. In a function component, React's `useState` and `useReducer` hooks handle this. Related items are packed into one state hook, while a class component must use a general state API. This was one of the major reasons to introduce React Hooks, and move away from class components.

Exercises:

- Write a stateful class component (e.g. Input).