Creating strongly-typed class state

In this lesson, we are going to learn how to create a strongly-typed state in a class component.

WE'LL COVER THE FOLLOWING

Component without state type specified

We are going to use an exercise in CodeSandbox to work on a `Counter` component. The task will be similar to what we did for the function-based `Counter` component earlier in this course.

Click the link below to open the exercise in CodeSandbox:

CodeSandbox project

We are going to add the current count as a state to the `Counter` component. We'll start by initializing the `count` state in the component's `constructor`:

```
class Counter extends React.Component {
  constructor(props: {}) {
    super(props);
    this.state = {
      count: 0
    }
  }
  render() {
    return <button>Click to start counter</button>;
  }
}
```

Let's also increment the `count` when the button is clicked and output the count in the button:

```
render() {
  return (
    <button onClick={() => this.setState((state) => ({count: state.count +
 1}))}>
      {this.state.count ? this.state.count : "Click to start counter"}
    </button>
  );
}
```

The component does function as we want, but we have type errors where the `count` state is referenced:

Why is the TypeScript compiler reporting this error?

Show Answer

Specifying the state type

So, how can we define the type for the state in a class component? Well, we can do this in the second generic parameter on the `Component` class:

```
class MyComponent extends React.Component<Props, State> { ... }
```

Let's specify the type for the `Counter` components state inline:

```
class Counter extends React.Component<{}, { count: number }> { ... }
```

Notice that we specify the props type as `{}` in the first generic parameter.

This is because we can't specify the second generic parameter without specifying the first.

The TypeScript compiler warnings should now disappear, and the `Counter` component should still function correctly.

The state type can also be specified using a type alias or an interface. Refactor the `Counter` components state type to be a type alias.

Show Answer

Default state from props

Let's start to implement a prop that allows the consumer of the `Counter` to specify the initial value for the `count` state:

```
type Props = {
  initialCount?: number;
}
type State = {
  count: number;
};
class Counter extends React.Component<Props, State> {
  static defaultProps = {
    initialCount: 0
  }
  constructor(props: Props) {
    super(props);
    this.state = {
      count: 0
    };
  }
  ...
}
```

Finish the implementation by setting the initial `count` state in the constructor to the `initialCount` prop value.

Show Answer

Try this out by passing the `initialCount` prop in the call to the `render`

Try this out by passing the `initialCount` prop in the call to the `render` function:

```
const rootElement = document.getElementById("root");
render(<Counter initialCount={5} />, rootElement);
```

We'll see that the buttons content starts at 5:

Wrap up

Well done, you can now create class components with a strongly-typed state! We specify the type for the class state in the second generic parameter of the `Component` base class. If the class has no props, we can use `{}` as the props type.