Creating strongly-typed class props

Earlier in this course we learned how to create strongly-typed props for function components. In this lesson, we'll learn how to do this for class components.

Specifying props

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

Click the link below to open the exercise in CodeSandbox:

CodeSandbox exercise

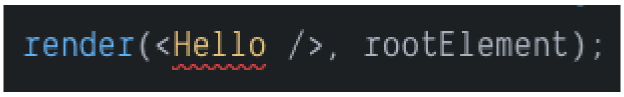
Class components inherit from the `Component` class in React. `Component` is a generic class that takes the props type in as a generic parameter:

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

Let's add a `who` prop to the `Hello` component by specifying the props type inline. Let's also output the value of `who` after the *Hello* message.

```
class Hello extends React.Component<{ who: string }> {
  render() {
    return <p>Hello, {this.props.who}</p>;
  }
}
```

A red squiggly line should appear under the `Hello` component reference in the call to the `render` function. What is the problem that is being highlighted to us?



```
render(<Hello />, rootElement);
```

Show Answer

Update `Hello` in the `render` function to pass in *Bob* into the `who` prop:

```
render(<Hello who="Bob" />, rootElement);
```

The red squiggly line should disappear, and *Hello, Bob* should be output to the browser.

Congratulations, you have just implemented and consumed a strongly typed prop in a class component!

Is it possible to implement the props type as a type alias? If so, implement the props as a type alias.

Show Answer

Is it possible to implement the props type as an interface? If so, implement the props as an interface.

Show Answer

Optional props

An optional prop can be implemented in the same way as a function component by putting a `?` before the type annotation.

In our CodeSandbox project that we are working on, add an optional `message` prop to the `Hello` component. Render the value of the `message` prop on the line after the hello message in a `p` tag.

Show Answer

Default prop values

We can provide default values for props in a class component using a `defaultProps` static property:

```
class MyComponent extends React.Component<P> {
  static defaultProps = {
    prop1: value1,
    prop2: value2,
    ...
  }
  ...
}
```

In our CodeSandbox project, set the `message` prop to *“How are you?”* by default.

Show Answer

Destructuring

If a method within a class uses some of the props multiple times, destructuring the props at the start of the method can arguably improve the methods readability.

In our CodeSandbox project, destructure the props in the `render` method and reference the destructured variables in the JSX.

Show Answer

No props

If a component doesn't take in any props, should we leave the generic parameter in the `Component` class blank? Is this type-safe?

Show Answer

Object props

A prop can be a complex object. Let's change the `who` prop to be an object that has a `name` property and `friend` boolean property. The `Hello` component will be as follows:

```
class Hello extends React.Component<Props> {
  static defaultProps = {
    message: "How are you?"
  };
  render() {
    const { who, message } = this.props;
    return (
      <React.Fragment>
        <p>{`Hello, ${who.name} ${who.friend && " my friend"}`}</p>
        {message && <p>{message}</p>}
      </React.Fragment>
    );
  }
}
```

What will be the definition of the Props type?

Show Answer

Function props

A prop can also be a function. A *render prop* is a common use case of a function prop.

Let’s add a render prop to our `Hello` component to allow the consumer to control how the message is rendered. The prop will be called `renderMessage` and the `Hello` component will be as follows:

```
class Hello extends React.Component<Props> {
  static defaultProps = {
    message: "How are you?"
  };
  render() {
    const { who, message, renderMessage } = this.props;
    return (
      <React.Fragment>
        <p>{`Hello, ${who.name} ${who.friend && " my friend"}`}</p>
        {message && (renderMessage ? renderMessage(message) : <p>{message}
</p>)}
      </React.Fragment>
    );
  }
}
```

An example consumption of `Hello` is as follows:

```
<Hello
  who={{ name: "Bob", friend: true }}
  message="Hey, how are you?"
  renderMessage={m => <i>{m}</i>}
/>
```

So, what should the definition of the Props type be now?

Show Answer

So, the type annotation for a function prop is:

```
(param1: type1, param2: type2, ...) => React.ReactNode;
```

Wrap up

Well done, we can now create strongly-typed props for class components!

Well done, we can now create strongly-typed props for class components. Creating the type itself is the same as for function components. The difference

is where we define a class component props type, which is the first generic parameter in the base `Component` class.

In the next lesson, we'll learn how to create a state that is strongly-typed in a class component.