React Components

- Props and State
- Lifecycle
- State in a Component Tree

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a UI Component?

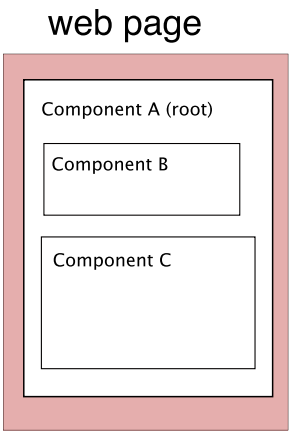
- Over the years, there have been several web-based, server-side component frameworks including Microsoft's ASP.NET and Java Server Faces (JSF)
- React provides a UI component architecture that runs on the client

The notion of UI components actually predates the web – for pure client apps, Microsoft defined the MFC framework and Java had Swing.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Composing Components

- Instead of defining a single "king" component, it's usually best to subdivide into smaller, more focused, easier-to-understand and more reusable subcomponents
- Many React apps define a **root** component and organize subcomponents into a **component tree**
- A common React design pattern is to store the bulk of the app's state in the root component



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Component Props

- React components are conceptually black boxes that can accept inputs referred to as **props**
- The *Single Strict Rule*: React components shall treat props as read-only



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining React Components

- React provides three ways to define components:
  - React.createClass()
  - Writing stateless functions
  - Extending React.Component
- Facebook no longer recommends using createClass(), so we will not cover it

When React was released, ES6 was not yet available so there was no standard syntax for defining classes in JavaScript, thus React invented its own with React.createClass(). But with ES6 adoption, Facebook has decided not to recommend using React.createClass() and may remove it in a future release.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Stateless Function Components

- For components that don't need to maintain any state, it's convenient to define them as functions that return JSX markup for the component's UI

App.js

```
class App extends Component
{
  render()
  {
    return (
      <section>
        <StatelessFunction myValue="Good stuff"/>
      </section>
    );
  }
}
```

StatelessFunction.js

```
function StatelessFunction (props)
{
   return(
     <section>
       <h2>
         Stateless Function Component!
       </h2>
       <div>
         My props: {props.myValue}
       </div>
    </section>);
}
```


Many React programmers consider using stateless functional components as a best practice since they are easier to reuse, test, understand and perform better than stateful components.

Note that JSX requires that all component markup be within a single root element, so if you want to return multiple HTML elements, you need to wrap them within a single element. Here we used <section> wrappers.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using ES6 Arrow Functions

- ES6 provides a convenient **arrow** syntax for defining functions that you can use to define React components

App.js

```
class App extends Component
{
  render()
  {
    return (
      <section>
        <StatelessArrow myValue="Good stuff"/>
      </section>
    );
  }
}
```

StatelessArrow.js

```
const StatelessArrow = (props) =>
{
   return(
     <section>
       <h2>
         Stateless Arrow Function Component!
       </h2>
       <div>
         My props: {props.myValue}
       </div>
    </section>);
}
```


For more info on arrow functions, see https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions.

One interesting feature of arrow functions is that they do NOT bind a local "this" in the way that normal JavaScript functions do. We will use that feature in a later example.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using ES6 Object Destructuring

- ES6 can automatically destructure objects during assignment - you can use this to write functional components that don't need to explicitly reference **props**

App.js

```
class App extends Component
{
  render()
  {
    return (
      <section>
        <StatelessArrowDestructured myValue="Good stuff"/>
      </section>
    );
  }
}
```

StatelessArrowDestructured.js

```
const StatelessArrowDestructured = ({myValue}) =>
{
   return(
     <section>
       <h2>
         Stateless Arrow Function with ES6 Destructured Object Syntax!
       </h2>
       <div>
         My props: {myValue}
       </div>
    </section>);
}
```


Note that for descructuring to work, you must use curly braces. If your input props has multiple properties, separate their names here with commas:

```
{myValue, anotherValue}
```

For more on "destructuring assignment", see:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Extending React.Component

- For components that need to maintain state, you can use ES6 classes
- We will discuss React component state in more detail later in this chapter
- Note that to reference *props*, use **this.props.xxx**

App.js

```
class App extends Component
{
  render()
  {
    return (
      <section>
        <StatefulClassComponent myValue="Good stuff"/>
      </section>
    );
  }
}
```

StatefulClassComponent.js

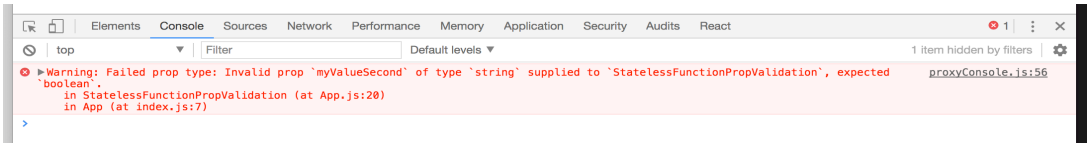
```
class StatefulClassComponent extends Component
{
   render()
   {
      return (
        <section>
          <h2>Stateful Class Component!</h2>
          <div>
            My props: {this.props.myValue}
          </div>
        </section>);
   }
}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Property Validation

- JavaScript is a loosely typed language - there's no compiler type-checking and variables can silently change type during assignment
- Loose typing makes developing and running programs easy, but can result in hard-to-debug runtime errors
- React lets you specify the type of a component's **props** - React will issue console warnings if you supply an incorrect value

```
Type     Validator
---      ---------
array    PropTypes.array
bool     PropTypes.bool
function PropTypes.func
number   PropTypes.number
object   PropTypes.object
string   PropTypes.string
```



Note that we are not talking about user-input validation here. React's property validation support is all about making it easier to develop and debug programs.

You can make any validation a required property by adding the text ".isRequired" to the validator name.

Also note that there are a few more validators available than shown here.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Property Validation, cont'd

- To perform property validation, first install the **prop-types** npm package and import it (or write a *script* element that references the library)
- Then define a **propTypes** property on the component that specifies types

App.js

```
class App extends Component
{
  render()
  {
    return (
      <section>
        <StatelessFunctionPropValidation
          myValueSecond="Good stuff"/>
      </section>
    );
  }
}
```

StatelessFunctionPropValidation .js

```
function StatelessFunctionPropValidation (props)
{
   return(
     <section>
       <div>
         My second value (should be a boolean):
           {props.myValueSecond}
       </div>
    </section>);
}

StatelessFunctionPropValidation.propTypes =
{
   myValueSecond: PropTypes.boolean.isRequired
};
```


PropTypes was originally part of React itself, but was separated into its own project to make it possible to use in non-React applications.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Default Property Values

- You can define a component property named **defaultProps** that specifies default values if parent component fails to supply a value

```
1    function StatelessFunctionPropValidation (props)
2    {
3       return(
4         <section>
5           <div>
6             My third value (should be a string, default: abc):
7                {props.myValueThird}
8           </div>
9        </section>);
10
11   }
12
13   StatelessFunctionPropValidation.defaultProps =
14   {
15      myValueThird: 'abc'
16   };
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Component Lifecycle Callbacks

- React manages the creation and destruction of components, and as it does, it can notify class-based component of lifecycle events, including:
  - **componentWillMount()** - React is about to mount the component into the page DOM
  - **componentDidMount()** - React successfully mounted the component
  - **componentWillReceiveProps(nextProps)** - A component's **props** changed after initial mounting
  - **shouldComponentUpdate(nextProps, nextState)** - React asking if a component should update, return **false** to prevent
  - **componentWillUpdate(nextProps, nextState)** - Component is about to render after initial mounting
  - **componentDidUpdate(prevProps, prevState)** - Component was updated after initial mounting
  - **componentWillUnmount()** - React is about to remove the component from the page DOM

Stateless functional components do NOT receive these lifecycle callbacks.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Component Lifecycle Callback Guidelines

- The **componentDidMount()** callback is useful for components that need to initialize component state via a network request (we will cover component state shortly)
- The **componentWillReceiveProps()** callback is useful for components that copy *props* into **state**
- The React docs recommend not using **shouldComponentUpdate()** for most components

As we will see, the React docs recommend that components do NOT copy props into state, but instead "lift state" into a parent component. Nonetheless, componentWillRecieveProps() is useful if you don't follow that guideline.

The React docs warn that a future React release may treat shouldComponentUpdate() as a "hint" - in other words, even if you return false, React may actually re-render the component.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Component State

- Some components need to maintain mutable state (recall that **props** should be read-only)
- React defines a programming model for such stateful components
- Note that you obviously do not define stateful components using the stateless function approach, so the recommend approach is to define state in ES6 classes using **this.state.xxx**

Examples of components that need to maintain state include forms and paged data tables that need to keep track of what's showing on the "current page".

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Initializing State

- The common approach to initializing state is from within the class **constructor()**
- The constructor is the only place where you should directly modify state - in all other cases use the **setState()** method

```
1    class StatefulClassComponentLifecycle extends Component
2    {
3       constructor(props)
4       {
5          super(props);
6
7          this.state =
8          {
9             myState: ""
10         };
11      }
12   . . .
13   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSX Event Handlers

- JSX lets you attach handler functions to events in a similar way as standard HTML
- Note that JSX requires camel-cased event names

```
1    class StatefulClassComponentLifecycle extends Component
2    {
3    . . .
4       render()
5       {
6          return (
7            <section>
8                <input
9                   name="myState"
10                  type="text"
11                  value={this.state.myState}
12                  onChange={this.handleInputChange} />
13           </section>);
14      }
15   }
```

In JSX, the common practice is to use onXxxx instead of using something like addEventListener(). In plain JavaScript (no React) the trend was to remove all JavaScript from HTML to keep it pure and thus use addEventListener() instead of onXxxx, but since JSX, by definition, mingles a component's markup (HTML) and behavior (JavaScript), it's not considered bad form to use onXxxx.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Updating State with setState()

- The **setState()** method accepts an object whose property names match up with state variables
- It's common to call setState() from a component's event-handler functions (e.g. *onChange* of a text box)

```
1    class StatefulClassComponentLifecycle extends Component
2    {
3    . . .
4       handleInputChange(event)
5       {
6          this.setState(
7          {
8            myState: event.target.value
9          });
10      }
11   }
```

Calling setState() will trigger the component's render() method as well as the lifecycle callbacks shouldComponentUpdate() and componentWillUpdate(). Note that any child components' render() methods will be triggered too.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Binding of this in Event Handlers

- By default, when React invokes a component's event-handling functions, **this** does NOT refer to the component
- One solution is to **bind** the component's *this* to the event-handler function during the component's constructor

```
1    class StatefulClassComponentLifecycle extends Component
2    {
3       constructor(props)
4       {
5    . . .
6            this.handleInputChange =
7              this.handleInputChange.bind(this);
8       }
9    . . .
10   }
```

The bind() method was introduced in ECMAScript version 5 - it creates and returns a new function that uses the specified "this".

Another approach is to code the event-handler function using ES6 "arrow syntax" - by definition, arrow functions use the "this" of the enclosing context:

```
handleInputChange = (event) =>
{
   this.setState(
   {
     myState: event.target.value
   });
}
```

Note that you don't need to worry about "this" in lifecycle methods such as componentDidMount() - React automatically binds those standard methods for you.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Component Example

App.js

StatelessFunction.js

StatelessArrow.js

StatelessArrowDestructured.js

StatelessFunctionPropValidation.js

StatefulClassComponentLifecycle.js

StatefulClassComponent.js

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    import React, { Component } from 'react';
2
3    import './App.css';
4
5    import StatelessFunction
6      from './StatelessFunction.js';
7    import StatelessArrow
8      from './StatelessArrow.js';
9    import StatelessArrowDestructured
10     from './StatelessArrowDestructured.js';
11   import StatefulClassComponent
12     from './StatefulClassComponent.js';
13   import StatelessFunctionPropValidation
14     from './StatelessFunctionPropValidation.js';
15   import StatefulClassComponentLifecycle
16     from './StatefulClassComponentLifecycle.js';
17
18   class App extends Component
19   {
20     render()
21     {
22       return (
23         <section>
24            <h1>Components</h1>
25            <StatelessFunction myValue="Good stuff"/>
26            <StatelessArrow myValue="Good stuff"/>
27            <StatelessArrowDestructured myValue="Good stuff"/>
28            <StatefulClassComponent myValue="Good stuff" />
29            <StatelessFunctionPropValidation
30               myValueFirst={12} myValueSecond="true"/>
31            <StatefulClassComponentLifecycle myValue="Good stuff"/>
32         </section>
33       );
34     }
35   }
36
37   export default App;
```

```
1    import React from 'react';
2
3    function StatelessFunction (props)
4    {
5       return(
6         <section>
7           <h2>
8              Stateless Function Component!
9           </h2>
10          <div>
11            My props: {props.myValue}
12          </div>
13       </section>);
14
15   }
16
17   export default StatelessFunction;
```

```
1    import React from 'react';
2
3    const StatelessArrow = (props) =>
4    {
5       return(
6         <section>
7           <h2>
8              Stateless Arrow Function Component!
9           </h2>
10          <div>
11            My props: {props.myValue}
12          </div>
13       </section>);
14   }
15
16   export default StatelessArrow;
```

```
1    import React from 'react';
2
3    const StatelessArrowDestructured = ({myValue}) =>
4    {
5       return(
6         <section>
7           <h2>
8              Stateless Arrow Function with
9              ES6 Destructured Object Syntax!
10          </h2>
11          <div>
12            My props: {myValue}
13          </div>
14       </section>);
15   }
16
17   export default StatelessArrowDestructured;
```

```
1    import React from 'react';
2    import PropTypes from 'prop-types';
3
4
5    function StatelessFunctionPropValidation (props)
6    {
7       return(
8         <section>
9           <h2>
10             Stateless Function Component with
11             Property Validation!
12          </h2>
13          <div>
14            My first value (should be a number):
15               {props.myValueFirst}
16          </div>
17          <div>
18            My second value (should be a boolean):
19               {props.myValueSecond}
20          </div>
21          <div>
22            My third value (should be a string):
23               {props.myValueThird}
24          </div>
25       </section>);
26
27   }
28
29   StatelessFunctionPropValidation.propTypes =
30   {
31      myValueFirst: PropTypes.number.isRequired,
32      myValueSecond: PropTypes.bool.isRequired,
33      myValueThird: PropTypes.string
34   };
35
36   StatelessFunctionPropValidation.defaultProps =
37   {
38      myValueThird: 'abc'
39   };
40
41   export default StatelessFunctionPropValidation;
```

```
1    import React, { Component } from 'react';
2    import ChildComponent from './ChildComponent.js';
3
4    class StatefulClassComponentLifecycle extends Component
5    {
6       constructor(props)
7       {
8          super(props);
9
10         this.state =
11         {
12            myState: ""
13         };
14
15          this.handleInputChange =
16            this.handleInputChange.bind(this);
17      }
18
19      handleSubmit = (event) =>
20      {
21          console.log("in handleSubmit()");
22
23          this.forceUpdate();
24
25         for(var aProp in this)
26           console.log(aProp + ":" + this[aProp]);
27
28          event.preventDefault();
29      }
30
31      handleInputChange(event)
32      {
33         const target = event.target;
34         const value = target.value;
35         const name = target.name;
36
37         this.setState(
38         {
39           [name]: value
40         });
41      }
42
43      render()
44      {
45         console.log("in render()");
```

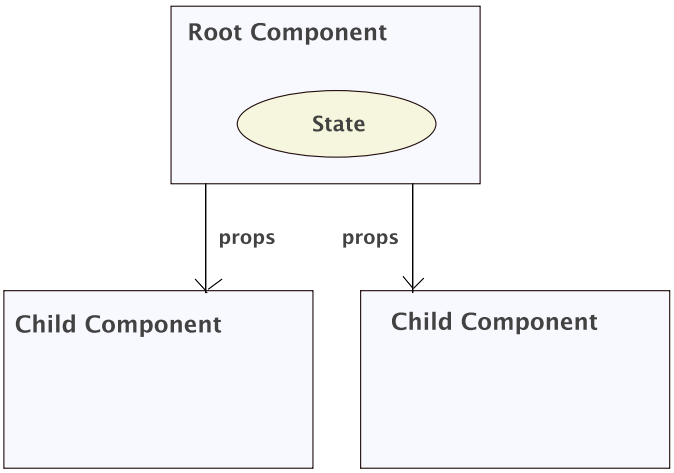
```
46
47          return (
48            <section>
49              <h2>Stateful Class Component Lifecycle!</h2>
50              <form>
51                <input
52                   name="myState"
53                   type="text"
54                   value={this.state.myState}
55                   onChange={this.handleInputChange} />
56                <input
57                   type="submit"
58                   value="Update"
59                   onClick={this.handleSubmit}/>
60              </form>
61              <ChildComponent myValue={this.state.myState} />
62            </section>);
63       }
64
65       componentWillMount()
66       {
67          console.log("in componentWillMount()");
68       }
69
70       componentDidMount()
71       {
72          console.log("in componentDidMount()");
73
74          for(var aProp in this)
75            console.log(aProp + ":" + this[aProp]);
76
77       }
78
79
80       componentWillReceiveProps(nextProps)
81       {
82          console.log("in componentWillReceiveProps()");
83       }
84
85       shouldComponentUpdate(nextProps, nextState)
86       {
87          console.log("in shouldComponentUpdate()");
88          return true;
89       }
90
```

```
91      componentWillUpdate()
92      {
93         console.log("in componentWillUpdate()");
94      }
95
96      componentDidUpdate()
97      {
98         console.log("in componentDidUpdate()");
99      }
100
101     componentWillUnmount()
102     {
103        console.log("in componentWillUnmount()");
104     }
105  }
106
107  export default StatefulClassComponentLifecycle;
```

```
1    import React, { Component } from 'react';
2
3    class StatefulClassComponent extends Component
4    {
5       render()
6       {
7          return (
8            <section>
9              <h2>Stateful Class Component!</h2>
10             <div>
11               My props: {this.props.myValue}
12             </div>
13           </section>);
14      }
15   }
16
17   export default StatefulClassComponent;
```

The Component Root as the Source of State

- A common architecture in React is to compose the UI into a component tree, where the bulk of the app's state resides in root component
- This lets you create mostly stateless UI components that are easier to reuse, debug and understand



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Pass State as Properties Down the Tree

- Since the root component stores the app's state, it can make it available to descendent components as **props**
- When the root component calls **this.setState()**, descendents will re-render with the updated data

ParentComponent.js

```
handleSomeEvent()
{
  var newValue = .......
  this.setState({myState: newValue});
}

render()
{
   return (
     . . .
    <ChildComponent
        myValue={this.state.myState} />
   );
}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Handle Events Where the State Lives

- Though the root component stores the app's state, descendent components often present a UI so the user can manipulate the data
- To make this work, the root component can pass references to callback functions to descendents
- The root's callback functions can then call **this.setState()** causing re-rendering and changes propagating to descendants via updated **props**

ParentComponent.js

```
handleSomeEvent()
{
  var newValue = .....
  this.setState({myState: newValue});
}

render()
{
   return (
     . . .
    <MyForm onSomeEvent=
        {this.handleSomeEvent} />
   );
}
```

MyForm.js

```
handleSubmit()
{
    this.props.onSomeEvent();
}

render()
{
   return (
     . . .
    <input type="submit"
       onClick={this.handleSubmit} />
   );
}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Component Tree Complete Example

ComponentTreeApp.js

FindStudentForm.js

DisplayStudent.js

DisplayTranscript.js

students.json

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    import React, { Component } from 'react';
2    import axios from 'axios';
3    import './App.css';
4
5    import FindStudentForm from './FindStudentForm.js';
6    import DisplayStudent from './DisplayStudent.js';
7
8    class ComponentTreeApp extends Component
9    {
10     constructor(props)
11     {
12        super(props);
13
14        this.state =
15        {
16           studentList: [],
17           currentStudent:
18           {
19              studentId: -1,
20              name: "No Name",
21              gpa:  -1,
22              transcript:
23              [
24              ]
25           },
26           findStudentMessage: ""
27        };
28
29        this.studentSelected = this.studentSelected.bind(this);
30     }
31
32     studentSelected(studentId)
33     {
34       const students = this.state.studentList;
35
36       for(var i = 0; i < students.length; i++)
37       {
38          var found = false;
39          if(studentId == students[i].id)
40          {
41              found = true;
42              this.setState(
43              {
44                 currentStudent: students[i],
45                 findStudentMessage: ""
```

```
46              });
47              break;
48          }
49       }
50
51       if(!found)
52       {
53          this.setState({findStudentMessage: "No student with that ID"});
54       }
55     }
56
57     componentDidMount()
58     {
59       axios.get('http://localhost:3000/students.json')
60         .then(response =>
61         {
62           const data = response.data;
63
64           this.setState(
65             {
66               studentList: data,
67               currentStudent: data[0]
68             });
69         })
70
71     }
72
73     render()
74     {
75       return (
76         <section>
77           <h1>Student Information</h1>
78           <div>
79             <FindStudentForm
80                message={this.state.findStudentMessage}
81                onStudentSelected={this.studentSelected}/>
82           </div>
83           <div>
84             <DisplayStudent currentStudent={this.state.currentStudent}/>
85           </div>
86         </section>
87       );
88     }
89   }
90
```

```
91   export default ComponentTreeApp;
```

```
1    import React, { Component } from 'react';
2
3    class FindStudentForm extends Component
4    {
5       constructor(props)
6       {
7          super(props);
8
9         this.state =
10        {
11          studentId: -1
12        }
13
14        this.handleInputChange = this.handleInputChange.bind(this);
15        this.handleSubmit = this.handleSubmit.bind(this);
16      }
17
18      handleInputChange(event)
19      {
20         const target = event.target;
21         const value = target.value;
22         const name = target.name;
23
24         this.setState(
25         {
26           [name]: value
27         });
28      }
29
30      handleSubmit(event)
31      {
32        console.log("in FindStudentForm.handleSubmit()");
33
34        // invoke the callback function from this component's parent
35        if(this.props.onStudentSelected)
36          this.props.onStudentSelected(this.state.studentId);
37
38        event.preventDefault();
39      }
40
41      render()
42      {
43        return(
44            <section>
45              <h2>Find Student</h2>
```

```
46              <form>
47               <label>
48                  Student ID: 
49                  <input type="number"
50                    name="studentId"
51                    value={this.state.studentId}
52                    onChange={this.handleInputChange} />
53               </label>
54               <input
55                  type="submit"
56                  value="Find"
57                  onClick={this.handleSubmit}/>
58               <span className="errorMsg">
59                   {this.props.message}
60               </span>
61             </form>
62            </section>
63        );
64      }
65   }
66
67   export default FindStudentForm;
```

```
1    import React from 'react';
2
3    import DisplayTranscript from './DisplayTranscript.js';
4
5    const DisplayStudent = ({currentStudent}) =>
6    {
7       return(
8         <section>
9           <h2>Current Student</h2>
10          <div>Name: {currentStudent.name}</div>
11          <div>ID: {currentStudent.id}</div>
12          <div>GPA: {currentStudent.gpa}</div>
13          <DisplayTranscript transcript={currentStudent.transcript} />
14        </section>
15      );
16   }
17
18   export default DisplayStudent;
```

```
1    import React from 'react';
2
3    const DisplayTranscript = ({transcript}) =>
4    <section>
5      <h3>Transcript</h3>
6      <table>
7        <thead>
8          <tr>
9            <th>Course ID</th>
10           <th>Name</th>
11           <th>Credit Hours</th>
12           <th>Grade</th>
13         </tr>
14       </thead>
15       <tbody>
16       {
17         transcript.map((course) =>
18           <tr key={course.courseId}>
19             <td>{course.courseId}</td>
20             <td>{course.name}</td>
21             <td>{course.credits}</td>
22             <td>{course.grade}</td>
23           </tr>
24       )}
25       </tbody>
26     </table>
27   </section>;
28
29   export default DisplayTranscript;
```

```
[
    {
       "id":12, "name":"Sue Jones", "gpa":3.45,
       "transcript":

         [
           {
              "courseId": 583,
              "name": "Introduction to Algorithms",
              "credits": 4,
              "grade": "B"
           },
           {
              "courseId": 1078,
              "name": "Java 101",
              "credits": 3,
              "grade": "C"
           }
         ]

    },
    {
       "id":44, "name":"Sam Smith", "gpa":2.0,
       "transcript":

         [
           {
              "courseId": 1078,
              "name": "Java 101",
              "credits": 3,
              "grade": "C"
           }
         ]

    },
    {
       "id":76, "name":"Paul Westerberg", "gpa":3.00,
       "transcript":

         [
           {
              "courseId": 583,
              "name": "Introduction to Algorithms",
              "credits": 4,
              "grade": "B"
```

```
46                }
47             ]
48
49         }
50    ]
```

Chapter Summary

In this chapter, you learned:

- How to pass input data to components via **props**
- How to store and update component state, especially in a root component
- About component lifecycle callback methods

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum