Audit Access Expires Apr 20, 2020
You lose all access to this course, including your progress, on Apr 20, 2020.

Lifting State Up

Lifting State Up

The setState() method only allows components to update their own state. However, there are times when an event occurs on a component and the event handler needs to update the state of another sibling or parent component. In this situation, we need to lift the state up to a parent component that encapuslates all of the components that need updating. The parent component will then pass down binded event handlers to the child components. When the child components call the binded event handlers, the parent component will update its state and may pass the updated state down to child components that may need it.

To demonstrate this concept we will build the following application:

Link to Codepen Solution

The application has four buttons and a description section below the buttons. The buttons initially all have blue text and are all inactive. The last button that is pressed is considered the active button and its text becomes red. Lastly, the description section shows the name of the active button.

We can create a Button, Details, and App Class Component to model the application:

```
class Details extends React.Component{
  render(){
    return <h1>{this.props.details}</h1>
  }
}
class Button extends React.Component{
    render(){
        return (
          <button>
            {this.props.name}
          </button>
        )
    }
}

class App extends React.Component{
    render(){
        return (
            <div>
                <Button name="One"/>
                <Button name="Two"/>
                <Button name="Three"/>
                <Button name="Four"/>
                <Details/>
            </div>


        )
    }
}
```

This application demonstrates the need for lifting the state up because when a Button component is pressed, it needs to tell its sibling Button components to become inactive and it needs to tell the Details section to change its text. Thus, all of the state should be held in the App Class Component and binded event handlers should be passed down to the child components.

Let's add some event handlers and state to the App Class Component: To accomplish this we will add a constructor() method to the App component so we can initialize its state. We will then declare an event handler named clickHandler and bind it to the App component. We will also pass down the event handlers down to the Button components. We will add an id property to each of the Button components so we can identify which Button we are pressing later on. The event handler will take in the id and name of the Button component that is clicked and will update the active Array and details section accordingly.

```
    class App extends React.Component{
        constructor(props){
            super(props)
            this.state = {activeArray:[0,0,0,0], details:""}
            this.clickHandler = this.clickHandler.bind(this)
        }

        clickHandler(id,details){
            var arr = [0,0,0,0]
            arr[id] = 1
            this.setState({activeArray:arr,details:details})
            console.log(id,details)
        }
        render(){
            return (
                <div>
                    <Button id = {0} active =
{this.state.activeArray[0]} clickHandler = {this.clickHandler}
name="bob"/>
                    <Button id = {1} active =
{this.state.activeArray[1]} clickHandler = {this.clickHandler}
name="joe"/>
                    <Button id = {2} active =
{this.state.activeArray[2]} clickHandler = {this.clickHandler}
name="tree"/>
                    <Button id = {3} active =
{this.state.activeArray[3]} clickHandler = {this.clickHandler}
name="four"/>
                    <Details/>
                </div>


            )
        }
    }
```

Next, we will edit the Button component. We will change its text color based on whether or not the button is active and we will define its onClick attribute based on the event handler that was passed down.

```
    class Button extends React.Component{
        render(){
            return (
              <button style = {{color: this.props.active? 'red':
'blue'}} onClick={() =>
{this.props.clickHandler(this.props.id,this.props.name)}}>
                {this.props.name}
              </button>
            )
        }
    }
```

In the end, it should look like this:

```
    class Details extends React.Component{
      render(){
        return <h1>{this.props.details}</h1>
      }
    }
    class Button extends React.Component{
        render(){
            return (
              <button style = {{color: this.props.active? 'red':
'blue'}} onClick={() =>
{this.props.clickHandler(this.props.id,this.props.name)}}>
                {this.props.name}
              </button>
            )
        }
    }
    class App extends React.Component{
        constructor(props){
            super(props)
            this.state = {activeArray:[0,0,0,0], details:""}
            this.clickHandler = this.clickHandler.bind(this)
        }
        clickHandler(id,details){
            var arr = [0,0,0,0]
            arr[id] = 1
            this.setState({activeArray:arr,details:details})
            console.log(id,details)
        }
        render(){
            return (
                <div>
                    <Button id = {0} active =
{this.state.activeArray[0]} clickHandler = {this.clickHandler}
name="bob"/>
                    <Button id = {1} active =
{this.state.activeArray[1]} clickHandler = {this.clickHandler}
name="joe"/>
                    <Button id = {2} active =
{this.state.activeArray[2]} clickHandler = {this.clickHandler}
```

```
name="tree"/>
                    <Button id = {3} active =
{this.state.activeArray[3]} clickHandler = {this.clickHandler}
name="four"/>
                    <Details details = {this.state.details}/>
                </div>



            )
        }
    }



ReactDOM.render(
  <App/>,
  document.getElementById("root")
)
```