CHAPTER-3
CLASS COMPONENT

3.1 CLASS COMPONENT OVERVIEW IN REACT

Class syntax is one of the most common ways to define a React component. While more verbose than the functional syntax, it offers more control in the form of lifecycle hooks.

3.2 CREATING A CLASS COMPONENT:

Creating a class component is pretty simple; just define a class that extends `Component` and has a `render` function.

Create dir : -> /src/Components/MyComponent.js

```
import React, { Component } from 'react';

class MyComponent extends Component {
  render() {
    return (
      <div>This is my component.</div>
    );
  }
}

export default MyComponent;
```

From there, you can use it in any other component.

Create dir : -> /src/Components/MyOtherComponent.js

```
import React, { Component } from 'react';
import MyComponent from './MyComponent';

class MyOtherComponent extends Component {
  render() {
    return (
      <div>
        <div>This is my other component.</div>
        <MyComponent />
      </div>
    );
  }
}

export default MyOtherComponent;
```

3.3 USING PROPS:

As is, MyComponent isn't terribly useful; it will always render the same thing. Luckily, React allows props to be passed to components with a syntax similar to HTML attributes.

```
<MyComponent myProp="This is passed as a prop." />
```

Props can then be accessed with this.props.

```
class MyComponent extends Component {
  render() {
    const {myProp} = this.props;
    return (
      <div>{myProp}</div>
    );
  }
}
```

3.4 USING STATE

One of the benefits class components have over functional components is access to component state.

```
class MyComponent extends Component {


  render() {
    const {myState} = this.state || {};
    const message = `The current state is ${myState}.`;
    return (
      <div>{message}</div>
    );
  }
}
```

3.5 USING LIFECYCLE HOOKS

Class components can define functions that will execute during the component's lifecycle. There are a total of seven lifecycle methods:

- ➔ **componentWillMount,**
- ➔ **componentDidMount,**
- ➔ **componentWillReceiveProps,**
- ➔ **shouldComponentUpdate,**
- ➔ **componentWillUpdate,**
- ➔ **componentDidUpdate,**
- ➔ **componentWillUnmount**.

For the sake of brevity, only one will be demonstrated.

```
class MyComponent extends Component {


  // Executes after the component is rendered for the first time
  componentDidMount() {
    this.setState({myState: 'Florida'});
  }

  render() {
    const {myState} = this.state || {};
    const message = `The current state is ${myState}.`;
    return (
      <div>{message}</div>
    );
  }
}
```

this.state should not be assigned directly. Use this.setState, instead.

this.setState cannot be used in render.