# React - Class

**Topics :** React JS
**Written on** January 02, 2024

Class components in React are JavaScript classes that extend `React.Component`. They were the primary way of creating components with state and lifecycle methods before the introduction of React Hooks. While functional components are now widely used with hooks, class components are still relevant and can be found in many existing React codebases.

Here's an example of a simple class component:

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

class MyComponent extends Component {
constructor(props) {
super(props);
this.state = {
message: 'Hello, React!',
};
}

componentDidMount() {
console.log('Component is mounted!');
}

componentDidUpdate(prevProps, prevState) {
console.log('Component updated!');
}

componentWillUnmount() {
console.log('Component will unmount!');
}

handleClick = () => {
this.setState({ message: 'Button Clicked!' });
};

render() {
return (
<div>
<p>{this.state.message}</p>
<button onClick={this.handleClick}>Click me</button>
</div>
);
```

```
}
}

export default MyComponent;
```

In this example:

- The `constructor` is called when the component is created. It initializes the component's state.
- `componentDidMount` is a lifecycle method called after the component is inserted into the DOM.
- `componentDidUpdate` is a lifecycle method called after the component updates.
- `componentWillUnmount` is a lifecycle method called just before the component is removed from the DOM.
- The `handleClick` method updates the component's state when a button is clicked.
- The `render` method returns the JSX structure that represents the component's UI.

When you need to manage local state, use lifecycle methods, or integrate with existing code that relies on class components, class components are still a valid choice. However, functional components with hooks are now the preferred approach for many developers due to their simplicity and flexibility.

If you were to rewrite the previous example using functional components and hooks, it might look like this:

```
import React, { useState, useEffect } from 'react';

const MyFunctionalComponent = () => {
const [message, setMessage] = useState('Hello, React!');

useEffect(() => {
console.log('Component is mounted!');
return () => {
console.log('Component will unmount!');
};
}, []); // Empty dependency array means this effect runs once after the initial render

useEffect(() => {
console.log('Component updated!');
}, [message]); // This effect runs whenever the 'message' state changes

const handleClick = () => {
setMessage('Button Clicked!');
};

return (
<div>
<p>{message}</p>
<button onClick={handleClick}>Click me</button>
</div>
);
};
```

```
export default MyFunctionalComponent;
```

In this functional component example:

- `useState` is used to manage local state.
- `useEffect` is used for side effects, and it can mimic the behavior of `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` by providing different dependency arrays.