**MODULE: 4 (List and Hooks)**

**1) Explain Life cycle in Class Component and functional component with Hooks.**

We use React lifecycle methods to handle different stages of a component's life, such as initialization, rendering, updating, and destruction, and to perform specific actions or add specific behaviour during each stage.

React, a popular JavaScript library for building user interfaces, revolves around the concept of components. These components have a lifecycle that can be divided into three primary phases: Mounting, Updating, and Unmounting. In this blog, we’ll explore these phases and the lifecycle methods associated with them using examples of both Class and Functional Components in React.

**There are 3 Phases of a Component Lifecycle**

**1) Mounting**

**2) Updating**

**3) Unmounting**

**Mounting**

The Mounting phase signifies the time when a component is first created and inserted into the Document Object Model (DOM). It involves setting up the initial state, rendering the component, and performing any side-effects related to the component’s mounting.

**Updating**

The Updating phase occurs when a component receives new props or its state changes. This phase is responsible for rendering updates, re-evaluating the component’s UI, and handling any side-effects associated with these updates.

**Unmounting**

The Unmounting phase happens when a component is removed from the DOM. This is where you can clean up any resources or subscriptions to prevent memory leaks.

**Class Component Lifecycle**

Let’s explore the Class Component lifecycle methods and their role in these phases.

* **`constructor` Method**
* `**componentDidMount` Method**
* **componentDidUpdate` Method**
* **`componentWillUnmount` Method**
* **`render` Method**

**`constructor` Method**

constructor method is rarely used in modern React. It is used to initialize the component's state and bind methods. State initialization is now typically done outside the constructor using class properties.

`**componentDidMount` Method**

componentDidMount is called after the component is inserted into the DOM. It's a suitable place to perform tasks like data fetching, setting up event listeners, or initializing third-party libraries.

**`componentDidUpdate` Method**

componentDidUpdate is called after a render update. It's useful for performing actions based on changes to props or state, such as updating the UI or making additional data requests.

**`componentWillUnmount` Method**

componentWillUnmount is called before the component is removed from the DOM. Here, you should clean up any resources like event listeners or subscriptions to avoid memory leaks.

**`render` Method**

render method defines the component's UI and is called every time the component mounts or updates. It returns the JSX that will be rendered to the DOM.

Example: Simple Counter using Class Component

import React from 'react';

class CounterClass extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

componentDidMount() {

console.log('Component mounted');

}

componentDidUpdate() {

console.log('Component updated');

}

componentWillUnmount() {

console.log('Component unmounted');

}

increment = () => {

this.setState((prevState) => ({ count: prevState.count + 1 }));

};

render() {

return (

<div>

<p>Count: {this.state.count}</p>

<button onClick={this.increment}>Increment</button>

</div>

);

}

}

export default CounterClass;

**Functional Component Lifecycle**

Functional Components, introduced with React Hooks, also have their lifecycle management using hooks. Let’s look at how these hooks correspond to the lifecycle phases.

**`useState` Hook**

The useState hook is used to initialize and manage state in Functional Components. It's equivalent to the constructor in Class Components, where you set the initial state.

**`useEffect` Hook**

useEffect is the swiss army knife of Functional Component lifecycle management. It handles component lifecycle events, including rendering (updating) and cleanup (unmounting).

Example: Simple Counter using Functional Component

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

function CounterFunctional() {

const [count, setCount] = useState(0);

useEffect(() => {

console.log('Component rendered');

return () => {

console.log('Component unmounted');

};

}, []);

const increment = () => {

setCount(count + 1);

};

return (

<div>

<p>Count: {count}</p>

<button onClick={increment}>Increment</button>

</div>

);

}

export default CounterFunctional;