**OBJECTIVES**

**1. Need and Benefits of Component Lifecycle**

React components go through different phases from birth to death in the application.

Lifecycle methods allow developers to:

* **Control component behavior** at specific moments
* **Optimize performance** by managing updates efficiently
* **Handle side effects** (data fetching, subscriptions)
* **Clean up resources** when components are removed
* **Respond to prop/state changes** appropriately

### **Key Benefits**

1. **Performance Optimization**

* Prevent unnecessary re-renders
* Control when updates should occur

1. **Resource Management**

* Properly initialize and clean up resources
* Prevent memory leaks

1. **Data Handling**

* Fetch data at the right time
* Update UI based on data changes

1. **Error Handling**

* Catch and handle errors in component tree

**2. Lifecycle Hook Methods** : **Class Component Lifecycle Methods**

React class components have three main lifecycle phases:

1. **Mounting Phase**
2. **Updating Phase**
3. **Unmounting Phase**
4. **Mounting Phase**

* constructor() - Initializes state and binds methods
* **static getDerivedStateFromProps()** - Updates state based on props
* render() - Returns JSX to be rendered
* componentDidMount() - Runs after first render (ideal for API calls)

1. **Updating Phase**

* static getDerivedStateFromProps() - Updates state before render
* shouldComponentUpdate() - Determines if re-render should occur
* render() - Re-renders component
* getSnapshotBeforeUpdate() - Captures DOM info before update
* componentDidUpdate() - Runs after update completes

1. **Unmounting Phase**

* componentWillUnmount() - Cleanup before component removal
* **Functional Component Hooks**
* Modern React uses these equivalent hooks:
* useState() - Manages component state
* useEffect() - Handles side effects (combines mount/update/unmount)
* useMemo()/useCallback() - Optimizes performance
* useLayoutEffect() - Similar to useEffect but fires synchronously

**3. Sequence of Steps in Rendering a Component**

1. **Initial Render (Mounting)**

* **Constructor** called (state initialization)
* getDerivedStateFromProps() (if used)
* **render()** method executes
* React updates DOM with rendered output
* componentDidMount() executes (side effects here)

1. **Update Cycle (Re-rendering)**

* New props received or state changes
* getDerivedStateFromProps() (if used)
* shouldComponentUpdate() (can cancel render)
* **render()** method executes
* getSnapshotBeforeUpdate() (captures pre-update info)
* React updates DOM
* componentDidUpdate() executes

1. **Unmounting**

* componentWillUnmount() called
* Cleanup operations performed
* Component removed from DOM

**Example**

class UserProfile extends React.Component {

constructor(props) {

super(props);

this.state = { userData: null };

}

componentDidMount() {

// Fetch data after initial render

fetchUserData(this.props.userId)

.then(data => this.setState({ userData: data }));

}

componentDidUpdate(prevProps) {

// Refetch if userId changes

if (this.props.userId !== prevProps.userId) {

fetchUserData(this.props.userId)

.then(data => this.setState({ userData: data }));

}

}

componentWillUnmount() {

// Cancel any pending requests

cancelPendingRequests();

}

render() {

if (!this.state.userData) return <LoadingSpinner />;

return <ProfileCard data={this.state.userData} />;

}

}