4. ReactJS-HOL

**Objectives**

**Explain the need and Benefits of component life cycle**

Need:  
React components go through several phases during their existence — creation, updating, and destruction. The lifecycle methods help you control what happens at each stage, such as:

* Fetching data when the component mounts.
* Responding to prop or state changes.
* Cleaning up resources before the component unmounts.

Benefits:

* Automate tasks like API calls, DOM manipulation, and state updates.
* Improve performance by updating only when needed.
* Avoid memory leaks using cleanup code.
* Better debugging and error handling.

**Identify various life cycle hook methods**

React (especially in class components) provides several lifecycle methods, grouped by phase:

Mounting Phase (Component is being created):

* constructor(): Initializes state and binds methods.
* static getDerivedStateFromProps(): Syncs state from props before rendering.
* render(): Returns the JSX to render UI.
* componentDidMount(): Runs after the component is added to the DOM — perfect for API calls.

Updating Phase (Component is re-rendering due to prop/state changes):

* static getDerivedStateFromProps(): Again called before re-rendering.
* shouldComponentUpdate(): Determines whether the component should re-render.
* render(): Re-renders the component.
* getSnapshotBeforeUpdate(): Captures info from the DOM before changes are committed.
* componentDidUpdate(): Called after the update is committed — good for updating DOM or making network requests.

Unmounting Phase (Component is removed):

* componentWillUnmount(): Cleanup work like clearing timers or cancelling API calls.

Error Handling Phase:

* componentDidCatch(error, info): Catches rendering errors in child components.

**List the sequence of steps in rendering a component**

Mounting (When component is added to the DOM):

1. constructor()
2. static getDerivedStateFromProps()
3. render()
4. componentDidMount()

Updating (When props or state change):

1. static getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

Unmounting (When component is removed):

1. componentWillUnmount()

**Step 1: Create a new react application using create-react-app tool with the name as “blogapp”**

npx create-react-app blogapp

cd blogapp

**Step 2: Create Post.js inside src/**

class Post {

constructor(id, title, body) {

this.id = id;

this.title = title;

this.body = body;

}

}

export default Post;

**Step 3: Create Posts.js class component**

import React from "react";

import Post from "./Post";

class Posts extends React.Component {

constructor(props) {

super(props);

this.state = {

posts: [],

hasError: false,

};

}

loadPosts() {

fetch("https://jsonplaceholder.typicode.com/posts")

.then((response) => response.json())

.then((data) => {

// Convert raw data to Post instances

const posts = data.map((post) => new Post(post.id, post.title, post.body));

this.setState({ posts });

})

.catch((error) => {

console.error("Error fetching posts:", error);

this.setState({ hasError: true });

});

}

componentDidMount() {

this.loadPosts();

}

componentDidCatch(error, info) {

alert("An error occurred: " + error.toString());

this.setState({ hasError: true });

}

render() {

if (this.state.hasError) {

return <h2>Something went wrong.</h2>;

}

return (

<div>

<h1>Blog Posts</h1>

{this.state.posts.map((post) => (

<div key={post.id}>

<h2>{post.title}</h2>

<p>{post.body}</p>

<hr />

</div>

))}

</div>

);

}

}

export default Posts;

**Step 4: src/App.js**

import React from "react";

import Posts from "./Posts";

function App() {

return (

<div className="App">

<Posts />

</div>

);

}

export default App;

**Step 5: Run the App**

Npm start

<http://localhost:3000>

