**4. ReactJS – HOL**

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

The **component lifecycle** in React refers to the series of **phases a component goes through** from creation to removal in the DOM. These include:

* **Mounting** (insertion into DOM)
* **Updating** (when props/state change)
* **Unmounting** (removal from DOM)
* **Error Handling**

**Need for Lifecycle Methods**

* To **control behavior** at specific stages (e.g., data fetching, cleanup)
* To **optimize performance** (e.g., avoid unnecessary re-renders)
* To **handle side effects** like API calls, event listeners, and timers
* To **manage resources** (open/close sockets, clear intervals)

**Benefits**

* Improves code **organization** and **reusability**
* Makes **asynchronous operations** predictable (e.g., API fetch in componentDidMount)
* Allows **clean-up** operations in componentWillUnmount
* Provides better **error handling** (componentDidCatch)

**2. Identify Various Lifecycle Hook Methods**

**Mounting (when component is added to DOM):**

| **Method** | **Purpose** |
| --- | --- |
| constructor() | Initialize state, bind functions |
| static getDerivedStateFromProps() | Sync state with props if needed |
| render() | Return JSX to be rendered |
| componentDidMount() | Invoked once after the component is mounted |

**Updating (when props/state change):**

| **Method** | **Purpose** |
| --- | --- |
| static getDerivedStateFromProps() | Called again before re-rendering |
| shouldComponentUpdate() | Control re-rendering for optimization |
| render() | Re-renders the JSX |
| getSnapshotBeforeUpdate() | Capture data (e.g., scroll position) before DOM update |
| componentDidUpdate() | Called after update is done |

**Unmounting (when component is removed):**

| **Method** | **Purpose** |
| --- | --- |
| componentWillUnmount() | Cleanup (timers, listeners, etc.) |

**Error Handling (if any error occurs in rendering):**

| **Method** | **Purpose** |
| --- | --- |
| componentDidCatch() | Catch and handle errors in components |

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

**Sequence during Component Lifecycle**

* **Mounting Phase**

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

* **Updating Phase**

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

* **Unmounting Phase**

1. componentWillUnmount()

* **Error Handling (if error occurs in any phase)**

1. componentDidCatch()

**----------------------------------------------------------------------------------------------------------------------------------------------**

**4. ReactJS-HOL - “blogapp” with Lifecycle Hooks**

**Step 1:** Open Terminal and create a new react project

***npx create-react-app blogap***

***cd blogapp***

**Step 2:** Create Posts.js and edit App.js

**Posts.js**

import React, { Component } from 'react';

class Posts extends Component {

  constructor(props) {

    super(props);

    this.state = {

      posts: [],

      hasError: false

    };

  }

  componentDidMount() {

    this.loadPosts();

  }

  loadPosts() {

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

      .then(res => res.json())

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

      .catch(err => this.setState({ hasError: true }));

  }

  componentDidCatch(error, info) {

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

  }

  render() {

    if (this.state.hasError) {

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

    }

    return (

      <div>

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

          <div key={post.id}>

            <h3>{post.title}</h3>

            <p>{post.body}</p>

          </div>

        ))}

      </div>

    );

  }

}

export default Posts;

**App.js**

import React from 'react';

import Posts from './Posts';

function App() {

  return (

    <div>

      <h1>Blog Posts</h1>

      <Posts />

    </div>

  );

}

export default App;

**Step 4:** In terminal, type ***npm start*** and this will run your code.

**OUTPUT:**

