Class component interview questions (5)

1. What is a class component in React, and how is it different from a functional component?

A **class component** is a component in React that is created using an ES6 class which **extends `React.Component`**.
It allows developers to use **state, props, lifecycle methods, and error boundaries**.
Class components manage internal data using `this.state` and update UI using `this.setState()`.
Before Hooks were introduced, class components were the **primary way to handle state and side effects** in React applications.

Example

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

class Welcome extends Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}

export default Welcome;
```

2. **What are Lifecycle Methods in a React Class Component?**

Lifecycle methods are predefined methods in React class components that execute automatically at different phases of a component's life cycle — **mounting, updating, and unmounting**.
They allow developers to **control component behavior**, such as fetching data, updating the DOM, optimizing performance, and cleaning up resources.
Lifecycle methods help manage **side effects** in a predictable way.

Example

```
class Data extends React.Component {
 componentDidMount() {
  console.log("Component mounted");
 }

 render() {
  return <h2>Data Loaded</h2>;
 }
}
```

3. **How does setState work in Class Components?**

setState is a built-in method in class components used to **update the component's state and trigger re-rendering** of the UI.
It works **asynchronously** because React batches multiple state updates together to improve performance and avoid unnecessary renders.
Instead of changing state immediately, React schedules the update and re-renders the component efficiently.

Example

```
class Counter extends React.Component {
 state = { count: 0 };


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


 render() {
  return (
   <>
    <h1>{this.state.count}</h1>
    <button onClick={this.increment}>+</button>
   </>
  );
 }
}
```

4. **What is a PureComponent?**

A PureComponent is a special type of class component that **automatically implements shouldComponentUpdate()**.
It prevents unnecessary re-rendering by performing a **shallow comparison** of current and previous props and state.
If there is no change, the component does not re-render, which improves performance in large applications.

Example

```
import React, { PureComponent } from "react";
```

```
class User extends PureComponent {
 render() {
  return <h3>{this.props.name}</h3>;
 }
}
```

5. **What are Error Boundaries in React?**

Error Boundaries are class components that catch **JavaScript errors in their child components**, log those errors, and display a fallback UI instead of crashing the entire application.
They work during rendering, lifecycle methods, and constructors of child components.
Error boundaries must be class components because React provides error-handling lifecycle methods **only in classes**, not in Hooks.

Example

```
class ErrorBoundary extends React.Component {
 state = { hasError: false };


 static getDerivedStateFromError() {
  return { hasError: true };
 }


 render() {
  if (this.state.hasError) {
   return <h2>Something went wrong</h2>;
  }
  return this.props.children;
 }
}
```

Basic React interview questions (5)

1. **What is React, and what problem does it solve compared to traditional DOM manipulation?**

React is a **JavaScript library** used to build fast and interactive user interfaces, especially for **single-page applications**.
In traditional JavaScript, directly manipulating the **DOM** is slow and complex because every

small change updates the entire page.
React solves this problem by using a **Virtual DOM**, updating only the parts of the UI that actually change, which improves **performance, speed, and code maintainability**.

React follows a **component-based architecture**, making applications reusable, scalable, and easier to manage.

Example

```
function App() {

  return <h1>Hello React</h1>;

}
```

2. **What is JSX in React, and how is it different from regular HTML?**

JSX (JavaScript XML) is a syntax extension used in React that allows developers to write **HTML-like code inside JavaScript**.
JSX makes UI code more readable and expressive, but it is **not HTML**—it gets converted into JavaScript by Babel.
Unlike HTML, JSX allows **JavaScript expressions** using {} and uses className instead of class.

Example

```
const name = "Surya";


function App() {

  return <h1>Hello {name}</h1>;

}
```

---

3. What is the difference between props and state in React?

**Props are used to pass data from parent to child components, and they are read-only.
State is used to manage internal data within a component and can change over time.**

Key Differences

Props	**State**
Passed from parent	**Managed inside component**
Read-only	**Can be updated**
Used for communication	**Used for dynamic data**

Example

```
function Child(props) {

  return <h2>{props.name}</h2>;
}


function Parent() {
  return <Child name="React" />;
}
```

4. What is the Virtual DOM, and how does React update the UI efficiently?

The Virtual DOM is a lightweight JavaScript copy of the real DOM maintained by React. When state or props change, React updates the Virtual DOM first, compares it with the previous version using a process called diffing, and updates only the changed elements in the real DOM.
This makes UI updates faster and more efficient compared to direct DOM manipulation.

Example

```
function Counter() {
  const [count, setCount] = React.useState(0);


  return (
    <>
      <h1>{count}</h1>
      <button onClick={() => setCount(count + 1)}>+</button>
    </>
  );
}
```

5. What is a component in React, and what are the main types?

A component is a reusable, independent piece of UI in React that returns JSX. Components help split the UI into smaller parts, making applications modular, reusable, and easy to maintain.

Main Types of Components

1. Functional Components – Simple JavaScript functions
2. Class Components – ES6 classes with lifecycle methods

Example (Functional Component)

```
function Welcome() {
  return <h1>Welcome to React</h1>;
}
```