Introduction

In this example, we have two React components: a class-based component (ClassBased) and a functional component (FunctionBased). Both components manage state and handle events to update the state and render dynamic content.

Key Concepts Covered

1. **Class-Based Components**
2. **Functional Components**
3. **State Management**
4. **Event Handling**

Class-Based Components:

- A class-based component extends React.Component and includes a render method to return JSX. It uses a constructor to initialize state.

```
import React, { Component } from "react";
import './ClassBased.css';

class ClassBased extends Component {
  constructor() {
    super();
    this.state = {
      tranInfo: "Morning Class Express at 09:00"
    };
  }

  handleClick = () => {
    this.setState({
      tranInfo: "Evening Class Express at 19:00"
    });
  };

  render() {
    return (
      <div className="train-info">
        <h1>Welcome to Class Component</h1>
        <h2>{this.state.tranInfo}</h2>
        <button onClick={this.handleClick}>Train Status</button>
      </div>
    );
  }
}

export default ClassBased;
```

The state is managed using this.state and updated using this.setState.

Functional Components:

- A functional component is a JavaScript function that returns JSX. It uses the useState hook to manage state.

```
import './ClassBased.css';
import { useState } from 'react';

let FunctionBased = () => {
  let [state, setState] = useState("Morning Function Express at 09:00");

  let handleClick = () => {
    setState("Evening Function Express at 19:00");
  };

  return (
    <div className='train-info'>
      <h1>Welcome to Functional Component</h1>
      <h2 id="train">{state}</h2>
      <button onClick={handleClick}>Train Status</button>
    </div>
  );
};

export default FunctionBased;
```

The useState hook initializes the state and returns a state variable and a function to update it.

State Management:

- **Class-Based Component:** State is initialized in the constructor and updated using this.setState

```
constructor() {
    super();
    this.state = {
      tranInfo: "Morning Class Express at 09:00"
    };
  }

  handleClick = () => {
    this.setState({
      tranInfo: "Evening Class Express at 19:00"
    });
  };
```

Functional Component: State is managed using the useState hook

```
let [state, setState] = useState("Morning Function Express at 09:00");

let handleClick = () => {
  setState("Evening Function Express at 19:00");
};
```

Event Handling:

- Both components handle button clicks to update the state.
- In class-based components, event handlers are methods bound to the component instance

```
<button onClick={this.handleClick}>Train Status</button>
```

In functional components, event handlers are defined within the function.

```
<button onClick={handleClick}>Train Status</button>
```

Interview Questions:

What is a class component in React?

- **Answer:** A class component in React is an ES6 class that extends from React.Component and contains a render method. Class components can hold and manage their own state and have access to lifecycle methods.

What is a functional component in React?

- **Answer:** A functional component is a plain JavaScript function that returns JSX. Functional components were initially stateless, but with the introduction of hooks, they can now manage state and side effects.

How do you manage state in a class component?

- **Answer:** In a class component, state is managed using the this.state object, and the state can be updated using the this.setState method.

How do you manage state in a functional component?

- **Answer:** In a functional component, state is managed using the useState hook, which returns an array containing the current state value and a function to update it.

What is the difference between functional and class components?

Functional Components

1. **Definition**: A JavaScript function that returns JSX.
2. **State Management**: Uses the useState hook for managing state.
3. **Lifecycle Methods**: Uses the useEffect hook to handle side effects and mimic lifecycle methods.
4. **Syntax**: Simple and concise syntax.
5. **Hooks**: Can use hooks like useState, useEffect, useContext, etc., to add functionality.
6. **Initialization**: No need for a constructor.
7. **this Keyword**: No need to use this.
8. **Performance Optimization**: Can use React.memo, useMemo, and useCallback for performance optimization.
9. **Binding Event Handlers**: No need for binding, typically uses arrow functions.
10. **Readability and Maintenance**: Generally easier to read and maintain due to their simpler structure.
11. **Side Effects**: Handled using the useEffect hook.

Class Components

1. **Definition**: An ES6 class that extends React.Component.
2. **State Management**: Uses this.state and this.setState for managing state.
3. **Lifecycle Methods**: Has built-in lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount.
4. **Syntax**: More verbose due to class syntax.
5. **Hooks**: Cannot use hooks directly, relies on lifecycle methods and class features.
6. **Initialization**: Often requires a constructor for state initialization.
7. **this Keyword**: Uses this to refer to the component instance.
8. **Performance Optimization**: Can use PureComponent and shouldComponentUpdate for performance optimization.
9. **Binding Event Handlers**: Often requires binding methods in the constructor.
10. **Readability and Maintenance**: Can be more complex and harder to manage due to the verbose syntax.
11. **Side Effects**: Handled using lifecycle methods like componentDidMount.