Handling Events in React

Question 1: How are events handled in React compared to vanilla JavaScript? Explain the concept of synthetic events.

Ans:

Event handling in React is similar in concept to vanilla JavaScript, but the implementation and abstraction are different.

1. Event Handling in Vanilla JavaScript

In plain JavaScript, events are attached directly to DOM elements.

Example:

```
<button id="btn">Click me</button>


<script>
 document.getElementById("btn").addEventListener("click", function () {
  console.log("Button clicked");
 });
</script>
```

Key Points:

- Events are bound directly to real DOM nodes
- Different browsers may have slight inconsistencies
- You manually manage event listeners
- Need to remove listeners to avoid memory leaks

2. Event Handling in React

React uses JSX and handles events declaratively.

Example:

```
        function App() {
 function handleClick() {
   console.log("Button clicked");
 }

 return <button onClick={handleClick}>Click me</button>;
}
```

Key Points:

- Event names use camelCase (onClick, not onclick)
- You pass a function reference, not a string
- React manages event binding automatically
- Cleaner and more readable code

3. What Are Synthetic Events in React?

React does not attach event listeners directly to each DOM node.

Instead, it uses a system called Synthetic Events.

🔶 Definition:

A Synthetic Event is a wrapper around the browser's native event, created by React to provide:

- Cross-browser compatibility
- Consistent behavior across all browsers
- Better performance

4. How Synthetic Events Work

- React attaches one event listener at the root level (event delegation)
- When an event occurs, React creates a SyntheticEvent object
- This object behaves the same across browsers

Example:

```
function handleClick(e) {
  console.log(e);         // SyntheticEvent
  console.log(e.target);    // Same as native event
}
```

5. Key Differences Between Native Events & Synthetic Events

Feature	Vanilla JS	React (Synthetic Events)
Event system	Browser native	React abstraction
Browser compatibility	Manual handling	Built-in
Event binding	addEventListener	JSX attributes

Feature	Vanilla JS	React (Synthetic Events)
Performance	Many listeners	Event delegation
Event object	Native Event	SyntheticEvent

6. Benefits of Synthetic Events

Cross-browser consistency
Automatic event pooling & optimization
Cleaner, declarative syntax
Better performance via event delegation

7. Important Note (Event Pooling)

In older versions of React, Synthetic Events were pooled, meaning the event object was reused.

Example (older React):

```
function handleClick(e) {
  setTimeout(() => {
    console.log(e.target); // ❌ null (event reused)
  }, 1000);
}
```

Solution:

```
function handleClick(e) {
  e.persist(); // keeps event
```

}

Question 2: What are some common event handlers in React.js? Provide examples of onClick, onChange, and onSubmit.

Ans:

React.js provides many built-in event handlers that allow you to respond to user actions like clicks, typing, and form submissions.

Below are the most commonly used event handlers, with clear examples.

1. onClick Event

Used for:

- Button clicks
- Icon clicks
- Any clickable element

Example:

```
function ClickExample() {
  const handleClick = () => {
    alert("Button clicked!");
  };

  return <button onClick={handleClick}>Click Me</button>;
}
```

Key Points:

- Uses camelCase → onClick
- Pass a function reference, not handleClick()

2. onChange Event :

Used for:

- Input fields
- Textarea
- Select dropdowns

Example:

```
import { useState } from "react";

function ChangeExample() {
  const [name, setName] = useState("");

  const handleChange = (e) => {
    setName(e.target.value);
  };

  return (
    <div>
      <input
        type="text"
        placeholder="Enter your name"
```

```
      onChange={handleChange}
    />
    <p>Name: {name}</p>
  </div>
 );
}
```

Key Points:

- Fires on every keystroke
- Uses event.target.value to read input value
- Commonly used with useState

3. onSubmit Event :

Used for:

- Form submission
- Login / Signup forms

Example:

```
function SubmitExample() {
 const handleSubmit = (e) => {
  e.preventDefault(); // prevents page reload
  alert("Form submitted!");
 };
```

```
  return (
    <form onSubmit={handleSubmit}>
      <input type="email" placeholder="Email" required />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Key Points:

- Always use e.preventDefault() in React forms
- Attached to the <form> element
- Handles Enter key + submit button automatically

4. Other Common React Event Handlers :

Event Handler	Purpose
onMouseEnter	Mouse hover
onMouseLeave	Mouse out
onKeyDown	Key press
onFocus	Input focus

Event Handler	Purpose
onBlur	Input loses focus
onDoubleClick	Double click

Question 3: Why do you need to bind event handlers in class components?

Ans:

In class components of React.js, event handlers do not automatically bind to the component instance. Because of this, we need to bind event handlers to make sure the this keyword works correctly.

1. The Core Problem: this Loses Context

In JavaScript classes, methods are not bound by default.

Problem Example (Without Binding):

```
class Counter extends React.Component {
 constructor() {
  super();
  this.state = { count: 0 };
 }

 handleClick() {
  this.setState({ count: this.state.count + 1 });
 }
```

```
  render() {
    return <button onClick={this.handleClick}>+</button>;
  }
}
```

Error: this is undefined inside handleClick.

This happens because:

- The method is passed as a callback
- It loses its reference to the class instance

2. Solution 1: Binding in the Constructor (Most Common):

Correct Way:

```
class Counter extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };

    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState({ count: this.state.count + 1 });
```

```
  }

  render() {
    return <button onClick={this.handleClick}>+</button>;
  }
}
```

3. Solution 2: Arrow Function Method (No Binding Needed)

Arrow functions do not have their own this, so they automatically use the surrounding context.

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

  handleClick = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return <button onClick={this.handleClick}>+</button>;
  }
}
```

Cleaner syntax
No constructor required

4. Solution 3: Arrow Function in JSX (Not Recommended)

<button onClick={() => this.handleClick()}>+</button>

Why not recommended?

- Creates a new function on every render
- Can cause performance issues

5. Why Binding Is Necessary (Summary)

Reason	Explanation
this context	Ensures access to this.state and this.props
JavaScript behavior	Class methods are not auto-bound
Event callbacks	Lose reference to the component instance
Error prevention	Avoids undefined errors
