# React Concepts: Events, Conditional Rendering, Forms, Lifecycle & Routing

## Handling Events in React

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

**React vs Vanilla JavaScript Events:**

- In vanilla JavaScript, you attach event listeners directly to DOM elements using methods like `addEventListener` or inline event attributes
- In React, you pass event handlers as props to JSX elements using camelCase naming (onClick, onChange, etc.)

**Synthetic Events:**

- React wraps native DOM events in SyntheticEvent objects
- These provide a consistent interface across different browsers, eliminating cross-browser compatibility issues
- Synthetic events have the same interface as native events but work uniformly across all browsers
- They automatically handle event pooling for performance optimization
- You can still access the original native event using `event.nativeEvent` if needed

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

**Common Event Handlers:**

- **onClick**: Triggered when an element is clicked (buttons, divs, etc.)
- **onChange**: Triggered when form input values change (input fields, select dropdowns, textareas)
- **onSubmit**: Triggered when a form is submitted
- **onFocus/onBlur**: Triggered when elements gain or lose focus
- **onMouseOver/onMouseOut**: Triggered when mouse enters or leaves an element
- **onKeyDown/onKeyUp**: Triggered when keys are pressed or released

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

In class components, `this` context is not automatically bound to event handler methods. Without binding:

- The `this` keyword inside the event handler will be `undefined`
- You cannot access component's state or other methods
- **Solutions**: Bind in constructor, use arrow functions, or use class property syntax with arrow functions
- Function components with hooks don't have this binding issue

## Conditional Rendering

### Question 1: What is conditional rendering in React? How can you conditionally render elements in a React component?

**Conditional Rendering:**

- The ability to render different UI elements or components based on certain conditions
- Allows dynamic content display depending on state, props, or other variables
- Essential for creating interactive and responsive user interfaces

**Methods:**

- If-else statements (outside JSX return)
- Ternary operators (inside JSX)
- Logical AND (&&) operator
- Switch statements
- Conditional assignment to variables

### Question 2: Explain how if-else, ternary operators, and && (logical AND) are used in JSX for conditional rendering.

**If-Else:**

- Cannot be used directly inside JSX
- Use before the return statement to conditionally assign JSX to variables
- Good for complex conditions with multiple branches

**Ternary Operators:**

- Can be used directly inside JSX
- Format: `condition ? trueElement : falseElement`
- Perfect for showing one element or another based on a condition

**Logical AND (&&):**

- Used when you want to render something or nothing
- Format: `condition && elementToRender`
- If condition is true, renders the element; if false, renders nothing
- Be careful with falsy values like 0, which will render as "0"

## Forms in React

### Question 1: How do you handle forms in React? Explain the concept of controlled components.

**Form Handling in React:**

- Forms are handled through component state and event handlers
- Input values are controlled by React state rather than DOM
- Form submission is typically handled by preventing default behavior and processing data through JavaScript

**Controlled Components:**

- Form elements whose values are controlled by React state
- The component's state serves as the "single source of truth"
- Input values are set via the `value` prop and updated via `onChange` handlers
- Provides full control over form data and validation
- Enables real-time validation and formatting

### Question 2: What is the difference between controlled and uncontrolled components in React?

**Controlled Components:**

- Values are managed by React state
- Use `value` prop and `onChange` handlers
- React controls the input value at all times
- Better for validation and formatting
- Recommended approach for most cases

**Uncontrolled Components:**

- Values are managed by the DOM itself
- Use `ref` to access DOM node directly
- React doesn't control the input value
- Similar to traditional HTML forms
- Useful for simple forms or when integrating with non-React libraries
- Use `defaultValue` for initial values instead of `value`

## Lifecycle Methods (Class Components)

### Question 1: What are lifecycle methods in React class components? Describe the phases of a component's lifecycle.

**Lifecycle Methods:**

- Special methods that are automatically called at specific points in a component's existence
- Allow you to hook into different stages of component lifecycle
- Enable setup, cleanup, and optimization tasks

**Three Main Phases:**

1. **Mounting**: Component is being created and inserted into DOM
2. **Updating**: Component is being re-rendered due to changes in props or state
3. **Unmounting**: Component is being removed from DOM

**Key Methods by Phase:**

- **Mounting**: constructor → componentDidMount
- **Updating**: componentDidUpdate
- **Unmounting**: componentWillUnmount

### Question 2: Explain the purpose of componentDidMount(), componentDidUpdate(), and componentWillUnmount().

**componentDidMount():**

- Called once after component is first rendered and mounted to DOM
- Perfect for API calls, setting up subscriptions, or initializing third-party libraries
- DOM elements are available at this point
- Equivalent to useEffect with empty dependency array in hooks

**componentDidUpdate():**

- Called after every re-render (except the initial render)
- Receives previous props and state as parameters
- Used for side effects based on prop or state changes
- Good for conditional API calls or DOM updates
- Must include condition checks to prevent infinite loops

**componentWillUnmount():**

- Called just before component is destroyed and removed from DOM
- Used for cleanup tasks like canceling network requests, removing event listeners, or clearing timers
- Prevents memory leaks and unwanted side effects
- Equivalent to cleanup function returned by useEffect in hooks

## Routing in React (React Router)

### Question 1: What is React Router? How does it handle routing in single-page applications?

**React Router:**

- A declarative routing library for React applications
- Enables navigation between different components/pages without full page refreshes
- Maintains application state while changing views
- Part of the React ecosystem but developed separately

**How it handles SPA routing:**

- Uses browser's History API to manipulate URL without page reloads
- Maps URL paths to specific React components
- Provides programmatic navigation capabilities
- Supports nested routing for complex application structures
- Maintains browser back/forward button functionality
- Enables bookmarkable URLs and deep linking

### Question 2: Explain the difference between BrowserRouter, Route, Link, and Switch components in React Router.

**BrowserRouter:**

- Root routing component that wraps your entire application
- Uses HTML5 history API for clean URLs (without # hash)
- Must be placed at the top level of your component tree
- Provides routing context to all child components

**Route:**

- Defines a mapping between a URL path and a component
- Renders specified component when URL matches the path
- Can be exact match or partial match
- Supports path parameters and query strings

**Link:**

- Replacement for anchor tags (`<a>`) in React Router
- Provides declarative navigation without page refresh
- Generates appropriate href attributes
- Maintains SPA behavior while updating URL

**Switch (now Routes in v6):**

- Renders only the first Route that matches the current location
- Prevents multiple routes from rendering simultaneously
- Provides exclusive routing behavior
- Essential for avoiding overlapping route matches
- In React Router v6, replaced by `Routes` component with improved functionality