Ministry of SMEs and Startups

KOSME Korea SMEs and Startups Agency

LIKELION

JOBKOREA

LESSON 05
Event Handling

WEEK 01

Introduction to Event Handling

- **What is Event Handling?**
  Managing user interactions (clicks, inputs, etc.) in React components.
- **Why Important?**
  Enables dynamic, interactive UIs.
- **React’s Approach**
  Synthetic events wrap native browser events for consistency.

Synthetic Events in React

❖ **Definition**
React's cross-browser wrapper for native events.

❖ **Benefits**
Consistent behavior across browsers, better performance.

❖ **Key Properties**
event.target, event.type, event.preventDefault().

```javascript
const handleSubmit = (e) => {
   e.preventDefault();
   console.log('Form submitted!');
 };
```

Declaring Event Handlers

❖ **Inline Functions**
Defined directly in JSX (less common).

❖ **Named Functions**
Defined in component, reusable, cleaner.

❖ **Arrow vs. Regular Functions**
Arrow functions for lexical this, regular for performance.

```
export default function DeclaringEventHandlers() {
  const handleToggle = () ⇒ {
    console.log('Toggled!');
  };
  return <button onClick={handleToggle}>Toggle</button>;
}
```

Event Handler Props

- **Common Props**
  onClick, onChange, onSubmit, onMouseOver, etc.
- **Naming Convention**
  CamelCase (e.g., onClick vs. onclick).
- **Passing Arguments**
  - Use arrow functions or bind for custom params.
  - Pass a function reference (not function call)

Handling Events in Functional Components

❖ **Using Functions with Hooks**

- ➢ Use useState to manage changes
- ➢ Arrow functions simplify inline handlers
- ➢ Avoid defining handler inside JSX to improve performance

```
function Counter() {
  const [count, setCount] = useState(0);
  return <button onClick={() => setCount(count + 1)}>Add</button>;
}
```

Passing Arguments to Event Handlers

❖ **How to Pass Parameters**

- ➢ Use arrow functions or bind to pass arguments
- ➢ Avoid calling handler directly in JSX

```
export default function PassingArguments() {
  const handleClick = (arg1, arg2) => {
    console.log(arg1, arg2);
  };

  return (
    <div>
      <button onClick={() => handleClick('Hello', 'World')}>Click Me</button>
    </div>
  );
}
```

Handling Input Changes

- **Controlled Components**
  Input value managed by state.
- **onChange Event**
  Updates state on user input.

```
export default function HandlingInputChanges() {
  const handleChange = (event) => {
    console.log(event.target.value);
  };
  return (
    <div>
      <input type='text' onChange={handleChange} />
    </div>
  );
}
```

Preventing Default Behavior

- **What is it?**
  Stops browser's default action (e.g., form submission).
- **How?**
  Use event.preventDefault().

```jsx
export default function PreventingDefaultBehavior() {
  const handleSubmit = (event) => {
    event.preventDefault();
    console.log('Form submitted');
  };

  return (
    <form onSubmit={handleSubmit}>
      <button type='submit'>Submit</button>
    </form>
  );
}
```

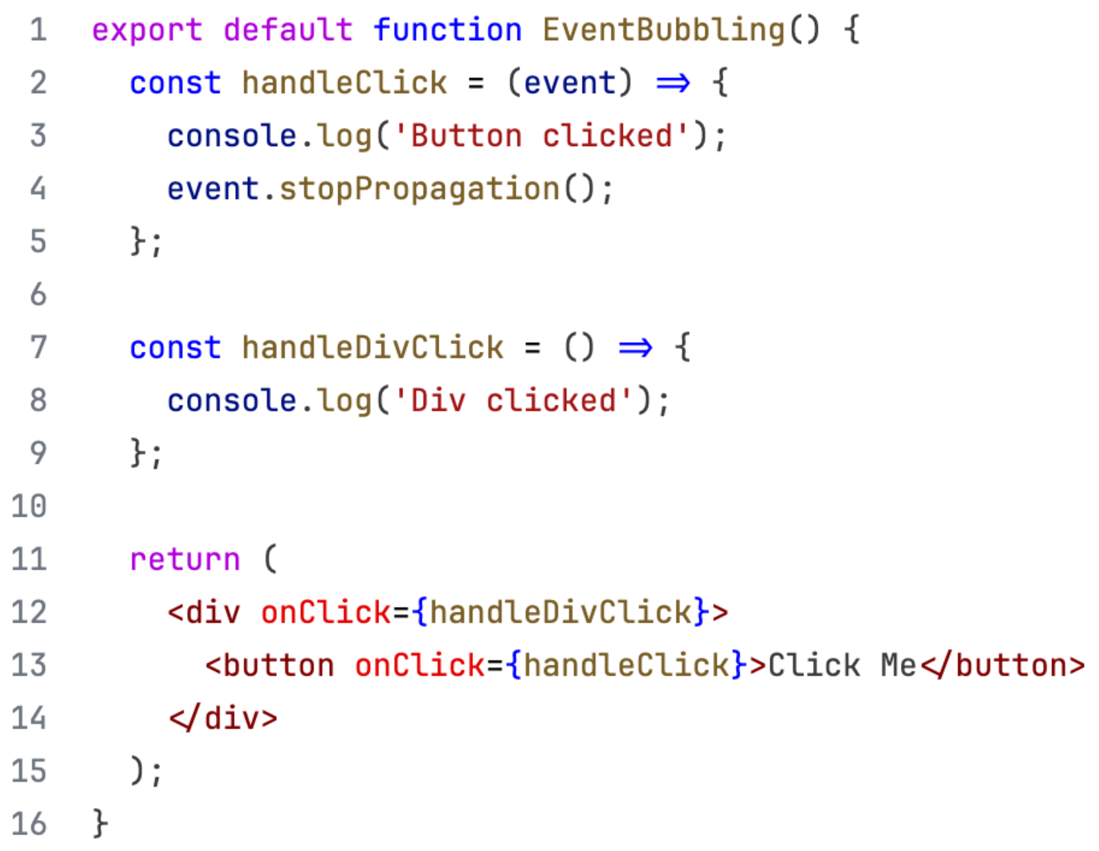
Event Bubbling

❖ **Definition**

Events propagate from child to parent elements.

❖ **Stopping Bubbling**

Use event.stopPropagation().



```
export default function EventBubbling() {
  const handleClick = (event) => {
    console.log('Button clicked');
    event.stopPropagation();
  };

  const handleDivClick = () => {
    console.log('Div clicked');
  };

  return (
    <div onClick={handleDivClick}>
      <button onClick={handleClick}>Click Me</button>
    </div>
  );
}
```

Capturing Events

- **Capture Phase**
  Events handled from parent to child before bubbling.
- **Syntax**
  Append Capture to event prop (e.g., onClickCapture).

Handling Form Submission

❖ **Form Events**
onSubmit for form submission.

❖ **Controlled Forms**
Manage form data with state.

```
export default function HandlingFormSubmission() {
  const [inputValue, setInputValue] = useState('');

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log('Form submitted:', inputValue);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type='text' value={inputValue}
        onChange={(e) => setInputValue(e.target.value)} />
      <button type='submit'>Submit</button>
    </form>
  );
}
```

Handling Mouse Events

- **Common Events**
  - onClick
  - onDoubleClick
  - onMouseEnter
  - onMouseLeave
  - onMouseDown
  - onMouseUp
  - onMouseMove
- **Use Cases**
  - Often used for hover effects or drag interactions

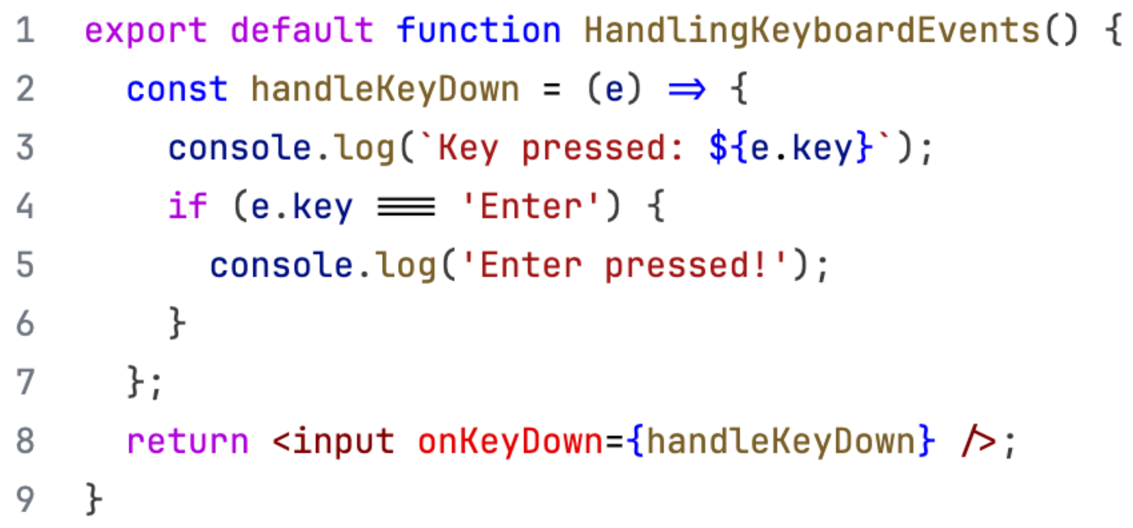
```
export default function HandlingMouseEvents() {
  const handleMouseEnter = () => {
    console.log('Mouse entered');
  };

  const handleMouseLeave = () => {
    console.log('Mouse left');
  };

  return (
    <div
      onMouseEnter={handleMouseEnter}
      onMouseLeave={handleMouseLeave}>
      Hover over this div
    </div>
  );
}
```

Handling Keyboard Events

- **Common Events**
  - onKeyDown
  - onKeyUp
  - onKeyPress
- **Key Properties**
  - Works on input, textarea, and some divs (with tabIndex)
  - Use event.key or event.code



```
export default function HandlingKeyboardEvents() {
  const handleKeyDown = (e) => {
    console.log(`Key pressed: ${e.key}`);
    if (e.key === 'Enter') {
      console.log('Enter pressed!');
    }
  };
  return <input onKeyDown={handleKeyDown} />;
}
```

Ministry of SMEs and Startups KOSME Korea SMEs and Startups Agency LIKELION JOBKOREA

Handling Touch Events

❖ **Common Events**
onTouchStart, onTouchMove, onTouchEnd.

❖ **Use Cases**
Mobile gestures, swipes.

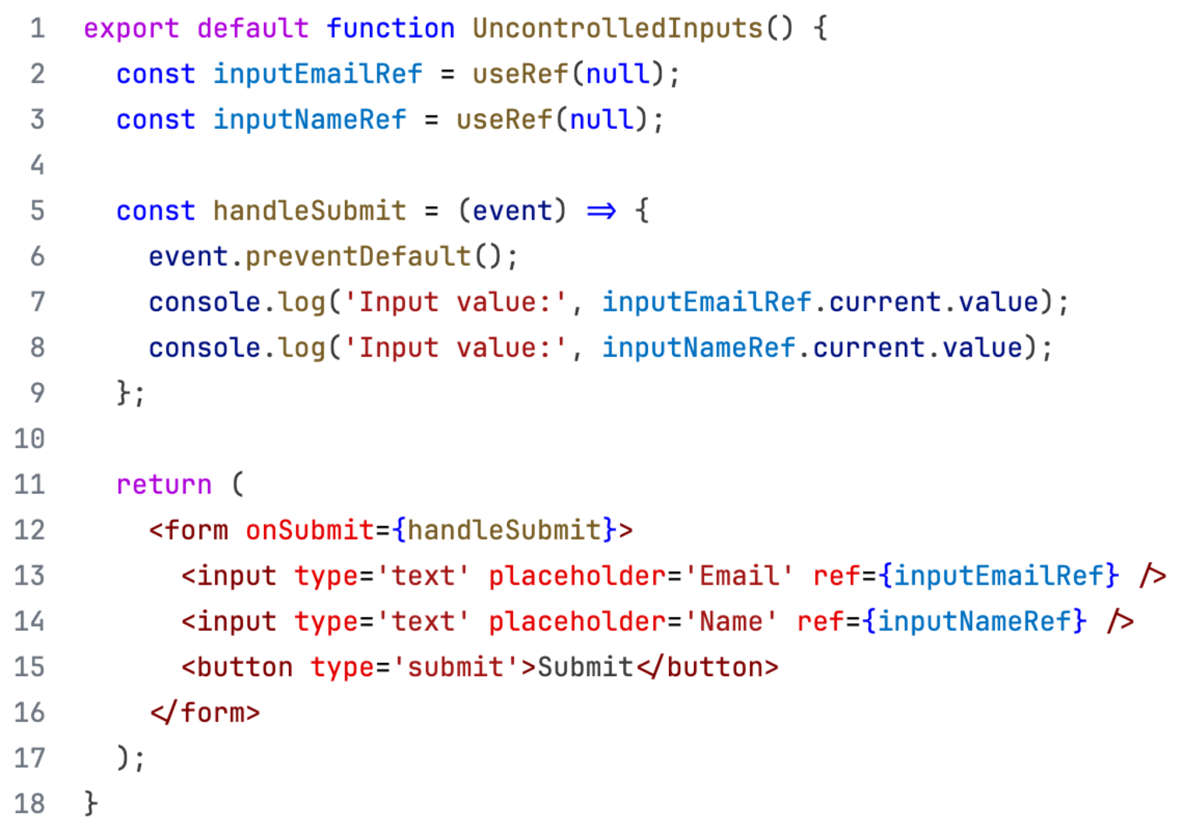
```
export default function UncontrolledInputs() {
  const inputEmailRef = useRef(null);
  const inputNameRef = useRef(null);

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log('Input value:', inputEmailRef.current.value);
    console.log('Input value:', inputNameRef.current.value);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type='text' placeholder='Email' ref={inputEmailRef} />
      <input type='text' placeholder='Name' ref={inputNameRef} />
      <button type='submit'>Submit</button>
    </form>
  );
}
```

Uncontrolled Inputs

- **Controlled**: Input state is in React (useState)
- **Uncontrolled**: Use ref to access input value
- Controlled forms offer better validation and control



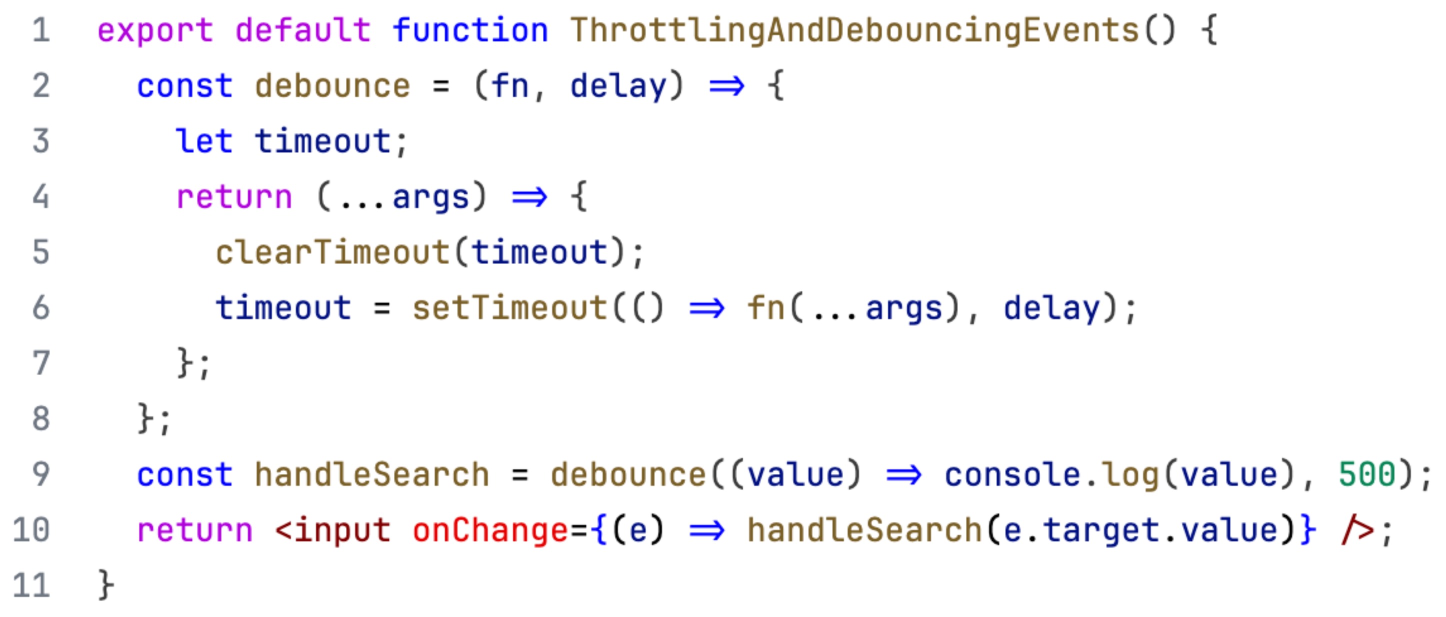
```
export default function UncontrolledInputs() {
  const inputEmailRef = useRef(null);
  const inputNameRef = useRef(null);

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log('Input value:', inputEmailRef.current.value);
    console.log('Input value:', inputNameRef.current.value);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type='text' placeholder='Email' ref={inputEmailRef} />
      <input type='text' placeholder='Name' ref={inputNameRef} />
      <button type='submit'>Submit</button>
    </form>
  );
}
```

Throttling and Debouncing Events

- **Definitions**
  Throttle: Limit event frequency. Debounce: Delay until event stops.
- **Use Cases**
  Scroll, resize, search input.

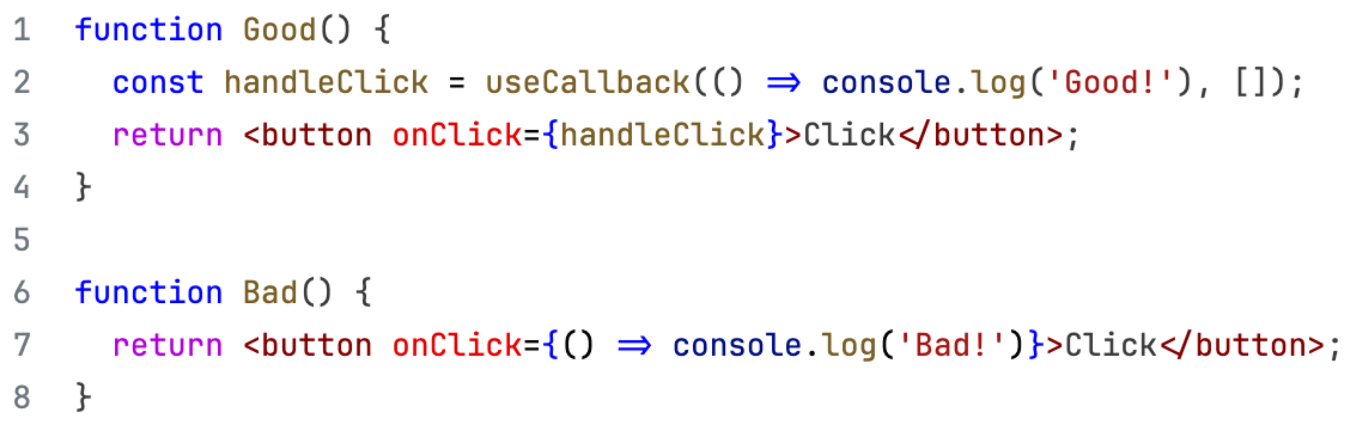


```
export default function ThrottlingAndDebouncingEvents() {
  const debounce = (fn, delay) => {
    let timeout;
    return (...args) => {
      clearTimeout(timeout);
      timeout = setTimeout(() => fn(...args), delay);
    };
  };
  const handleSearch = debounce((value) => console.log(value), 500);
  return <input onChange={(e) => handleSearch(e.target.value)} />;
}
```

Common Pitfalls

❖ **Inline Functions**
Cause re-renders, avoid in loops.

❖ **Missing Dependencies**
useCallback needs correct dependency array.



```jsx
function Good() {
  const handleClick = useCallback(() => console.log('Good!'), []);
  return <button onClick={handleClick}>Click</button>;
}

function Bad() {
  return <button onClick={() => console.log('Bad!')}>Click</button>;
}
```