🧠 React `useEffect()` vs. Class Lifecycle Methods

📌 1. `useEffect(() => { ... }, [deps])` = Similar but not same as `componentDidUpdate()`

They **behave similarly**, but they are **not the same**! Let's break it down 🔍

⬛ `useEffect` (Functional Component)

```
import React, { useEffect, useState } from "react";

const Counter = () => {
  const [count1, setCount1] = useState(0);
  const [count2, setCount2] = useState(0);

  // ☑ useEffect triggers only when count1 or count2 changes
  useEffect(() => {
    console.log("🔁 Either count1 or count2 changed!");
  }, [count1, count2]);

  return (
    <div>
      <h2>Count1: {count1}</h2>
      <h2>Count2: {count2}</h2>
      <button onClick={() => setCount1(count1 + 1)}>➕ Count1</button>
      <button onClick={() => setCount2(count2 + 1)}>➕ Count2</button>
    </div>
  );
};
```

⬛ `componentDidUpdate()` (Class Component)

```
import React from "react";

class CounterClass extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count1: 0,
      count2: 0,
    };
  }

  // ☑ Runs after every render — we must manually check what changed
  componentDidUpdate(prevProps, prevState) {
```

```
    if (
      prevState.count1 !== this.state.count1 ||
      prevState.count2 !== this.state.count2
    ) {
      console.log("🔁 Either count1 or count2 changed (class version)!");
    }
  }

  render() {
    const { count1, count2 } = this.state;
    return (
      <div>
        <h2>Count1: {count1}</h2>
        <h2>Count2: {count2}</h2>
        <button onClick={() => this.setState({ count1: count1 + 1 })}>➕
Count1</button>
        <button onClick={() => this.setState({ count2: count2 + 1 })}>➕
Count2</button>
      </div>
    );
  }
}
```

🔍 Behavior Comparison Chart

Feature	`useEffect` (Functional)	`componentDidUpdate` (Class)
Triggered after render?	☑ Yes	☑ Yes
Automatically checks changes?	☑ Yes (via `[deps]`)	❌ No (you check manually using `prevState`)
Initial render runs?	❌ No (if deps exist)	❌ No (not on first render)
Manual check needed?	❌ No	☑ Yes
Cleanup function supported?	☑ Yes (`return () => {}`)	☑ Use `componentWillUnmount()`
Simpler to read?	☑ Yes	❌ Slightly more verbose

❗ Important: Not the Same!

> `useEffect(() => {}, [count1, count2])` and `componentDidUpdate()` are similar **in purpose** but not **identical in behavior**.

For example:

◇ `useEffect()` **won't run on first render** if you provide dependencies. ◇ `componentDidUpdate()` also doesn't run on first render — but it requires **manual checks**.

```
useEffect(() => {
  // logic
}, [count1, count2]);
```

is replicated in **Class-based components**.

☑ Hook Behavior Recap:

```
useEffect(() => {
  // runs when either count1 or count2 changes
}, [count1, count2]);
```

This means: "*Run this effect whenever `count1` or `count2` changes.*"

🔁 Equivalent in Class Components:

In class components, **you use `componentDidUpdate(prevProps, prevState)`** to detect changes in specific state values:

☑ Example (Class-Based)

```
import React from 'react';

class CounterComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count1: 0,
      count2: 0,
    };
  }

  componentDidUpdate(prevProps, prevState) {
    // Check if count1 or count2 has changed
    if (
      prevState.count1 !== this.state.count1 ||
      prevState.count2 !== this.state.count2
    ) {
      console.log("🔄 count1 or count2 changed!");
      // Perform your logic here...
    }
  }

  render() {
    const { count1, count2 } = this.state;
```

```
    return (
      <div>
        <h2>Count1: {count1}</h2>
        <h2>Count2: {count2}</h2>
        <button onClick={() => this.setState({ count1: count1 + 1 })}>
          ➕ Increase Count1
        </button>
        <button onClick={() => this.setState({ count2: count2 + 1 })}>
          ➕ Increase Count2
        </button>
      </div>
    );
  }
}

export default CounterComponent;
```

🧠 Summary

- ☑ Use `useEffect(() => {...}, [deps])` to track specific state/prop changes in functional components
- 🧠 In class components, use `componentDidUpdate(prevProps, prevState)` and compare values manually
- ⚠ Don't assume they are 1:1 interchangeable — their **timing and cleanup behavior differ**

🔍 Summary Table

Hook Version	Class-Based Equivalent
`useEffect(() => {}, [])`	`componentDidMount()`
`useEffect(() => {}, [var1, var2])`	`componentDidUpdate(prevProps, prevState)` with condition
`useEffect(() => { return () => {} })`	`componentWillUnmount()`
