CipherSchools

Class 6 ReactJS

Lifecycle Methods and useEffect

What are Lifecycle Methods?

In React, lifecycle methods are functions that get called at different phases of a component's existence:

- When it's created
- When it updates
- When it's about to unmount

Lifecycle in Class Components

1. Mounting Phase (Component is Created)

Method	Purpose
constructor()	Initialize state, bind methods
componentDidMount()	Runs once after the component is added to the DOM

Example:

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

  componentDidMount() {
```

```
    console.log("Mounted!");
  }

  render() {
    return <div>Hello</div>;
  }
}
```

2. Updating Phase (Props or State Change)

Method	Purpose
shouldComponentUpdate()	Control whether re-render should happen
componentDidUpdate()	Called after the component updates

3. Unmounting Phase

Method	Purpose
componentWillUnmount()	Cleanup (event listeners, timers, etc.)

Example

```
componentWillUnmount() {
  clearInterval(this.timer);
}
```

Functional Components and useEffect

In functional components, we don't use lifecycle methods. Instead, we use the useEffect hook, which is more flexible.

What is useEffect in React?

useEffect is a side-effect management hook introduced in React 16.8.

A "side effect" means:

Anything that happens outside the component's rendering logic:

- Fetching data
- Changing the DOM manually
- Setting up event listeners
- Managing timers or intervals
- Subscribing to sockets

Syntax of useEffect

```
useEffect(() => {
  // Code here runs after render

  return () => {
    // Cleanup runs before unmount or before the next effect
  };
}, [dependencies]);
```

How useEffect replaces lifecycle methods

Class Lifecycle Method	useEffect Equivalent
componentDidMount	useEffect(() => { ... }, [])
componentDidUpdate	useEffect(() => { ... }, [dependencies])
componentWillUnmount	return () => { ... } inside useEffect

Examples:

1. Run once on mount:
   ```
   useEffect(() => {
     console.log("Mounted");
   }, []);
   ```

   Why empty []?

Because it has no dependencies, so it runs only once when the component is first rendered.

2. Run when props/state change:
   ```
   useEffect(() ⇒ {
     console.log("Count changed");
   }, [count]);
   ```

   When is this useful?
   - Reacting to prop changes
   - Syncing state with local storage or URL

3. Cleanup before unmount:
   ```
   useEffect(() ⇒ {
     const interval = setInterval(() ⇒ console.log("tick"), 1000);

     return () ⇒ clearInterval(interval); // Cleanup
   }, []);
   ```

   When is this useful?
   - Clearing timers
   - Removing event listeners
   - Unsubscribing from APIs or sockets

Practice Questions

1. Github Profile Search
   a. Default Profile on Mount
   b. Search field for searching Profile

   Code:

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

function UserSearch() {
  const [searchText, setSearchText] = useState("hiteshchoudhary");
  const [result, setResult] = useState(null);
```

```
  useEffect(() => {
    handleSearch();
  }, []);

  const handleSearch = async () => {
    const res = await fetch(`https://api.github.com/users/${searchText}`, {
      method: "GET"
    });
    const jsonData = await res.json();
    const formattedData = {
      name: jsonData.name,
      bio: jsonData.bio,
      profile: jsonData.avatar_url,
      username: jsonData.login
    };
    setResult(formattedData);
  };

  return (
    <main>
      <h1>Github User Search</h1>
      <input
        type="text"
        value={searchText}
        placeholder="Github Username"
        onChange={(e) => setSearchText(e.target.value)}
      />
      <button onClick={handleSearch}>Search</button>
      {result ? (
        <section>
          <img src={result.profile} width={200} height={200} alt="Github_profile" />
          <h3>{result.name}</h3>
          <p>
            <i>{result.username}</i>
          </p>
          <p>{result.bio}</p>
        </section>
      ) : null}
    </main>
  );
}
```

```
export default UserSearch;
```

2. Timer
    a. Start, Stop and Reset functionality
    b. With useEffect Hook
    c. Using SetInterval and clearInterval

Code:

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

function TimerWithEffect() {
  const [status, setStatus] = useState("stop");
  const [interval, setInter] = useState(null);
  const [timer, setTimer] = useState(0);

  useEffect(() => {
    if (status === "start") {
      let interval = setInterval(() => {
        setTimer((prev) => prev + 1);
      }, 1000);

      setInter(interval);
    }

    if (status === "stop") {
      clearInterval(interval);
      setInter(null);
    }

    if (status === "reset") {
      setTimer(0);
    }

    return () => {
        clearInterval(interval);
    }
  }, [status]);

  return (
    <main>
      <h1>Stop Watch</h1>
```

```
      <h1>{timer}</h1>
      <div>
        <button onClick={() => setStatus("start")}>Start</button>
        <button onClick={() => setStatus("stop")}>Stop</button>
        <button onClick={() => setStatus("reset")}>Reset</button>
      </div>
    </main>
  );
}

export default TimerWithEffect;
```

3. AutoStart Timer
    a. Start Timer while Component Mounting
    b. Stop Functionality
    c. With useEffect Hook
    d. Using SetInterval and clearInterval

Code:

```
function TimerWithEffect() {
  const [interval, setInter] = useState(null);
  const [timer, setTimer] = useState(0);

  useEffect(() => {
    let inter = setInterval(() => {
      setTimer((prev) => prev + 1);
    }, 1000);

    console.log("hello");
    setInter(inter);

    return () => {
      clearInterval(inter);
    };
  }, []);

  const handleStop = (interval) => {
    clearInterval(interval);
    setInter(null);
  };
```

```
  return (
    <main>
      <h1>Stop Watch</h1>
      <h1>{timer}</h1>
      <button onClick={() => handleStop(interval)}>Stop</button>
    </main>
  );
}

export default TimerWithEffect;
```

4. StopWatch
    a. Show Time in 00:00:00 format on Display
    b. Use useEffect hook and clearInterval
    c. Timer Start on Mounting

Code:

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

function Stopwatch() {
  const [timer, setTimer] = useState(0);

  useEffect(() => {
    const id = setInterval(() => {
      setTimer((prev) => prev + 1);
    }, 1000);

    return () => {
      clearInterval(id);
    };
  }, []);

  const handleFormat = (timer) => {
    const hours = parseInt(timer / 3600);
    const mins = parseInt((timer - hours * 3600) / 60);
    const secs = timer % 60;

    return `${hours <= 9 ? `0${hours}` : hours}:${mins <= 9 ? `0${mins}` : mins}:${
      secs <= 9 ? `0${secs}` : secs
    }`;
```

```
  };

  return (
    <main>
      <h1>Stopwatch</h1>
      <h3>{handleFormat(timer)}</h3>
    </main>
  );
}

export default Stopwatch;
```

5. Cheating Detector When Resize
    a. Show Cheating Detected on Resizing Window
    b. Add Event Listener and Clear it on Unmounting

Code:

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

function Cheating() {
  const [status, setStatus] = useState(false);

  const handleResize = () => {
    setStatus(true);
  };

  const handleNormal = () => {
    setStatus(false);
  };

  useEffect(() => {
    window.addEventListener("resize", handleResize);

    return () => {
      window.removeEventListener("resize", handleNormal);
    };
  }, []);

  return (
    <main>
      <h1>{!status ? "No Cheating Detected" : "Cheating Detected"}</h1>
    </main>
```

```
  );
}

export default Cheating;
```

6. Save to draft functionality
    a. Auto Save Functionality
    b. Debouncing: Update localStorage after every 300ms

Code:

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

function Draft() {
  const [text, setText] = useState("");
  const [cleared, setCleared] = useState(false);
  // Single useEffect
  //   useEffect(() => {
  //     const getDraftMessage = localStorage.getItem("draftMessage");
  //     if (text === "") setCleared(true);
  //     if (getDraftMessage !== "" && text === "" && !cleared) return;
  //     localStorage.setItem("draftMessage", text);
  //   }, [text]);

  useEffect(() => {
    const getDraftMessage = localStorage.getItem("draftMessage");
    if (getDraftMessage !== "" && text === "") setText(getDraftMessage);
  }, []);

  useEffect(() => {
    const id = setTimeout(() => {
      if (text !== "") {
        console.log("LOcalstorage change");
        localStorage.setItem("draftMessage", text);
      } else {
        setCleared(true);
      }
    }, 500);

    return () => {
      clearTimeout(id);
    };
  }, [text]);
```

```
  if (cleared) localStorage.setItem("draftMessage", "");

  return (
    <main>
      <h1>Save To Draft</h1>

      <input
        type="text"
        placeholder="Drafted  text"
        value={text}
        onChange={(e) => setText(e.target.value)}
      />
    </main>
  );
}

export default Draft;
```

7. Check IsOnline or Offline
    a. Show User is Online or Offline without any action

Code:

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

function InternetConnection() {
  const [isOnline, setIsOnline] = useState(navigator.onLine);

  const handleOnline = () => {
    setIsOnline(true);
  };

  const handleOffline = () => {
    setIsOnline(false);
  };

  useEffect(() => {
    window.addEventListener("online", handleOnline);
    window.addEventListener("offline", handleOffline);

    return () => {
```

```
      window.removeEventListener("online", handleOnline);
      window.removeEventListener("offline", handleOffline);
    };
  }, []);

  return (
    <main>
      <h1>Internet Connection</h1>

      <h3>{isOnline ? "User has good internet connection" : "No Internet
Connection"}</h3>
    </main>
  );
}

export default InternetConnection;
```