Lifecycle Methods:(22)

These lifecycle methods are available in class component. lifecycle methods of class has four phases:

Mounting

When an instance of a component is being created and inserted into the DOM.

📌 Think of it as birth — the moment a component enters the scene.

Key methods/hooks: constructor(), render(), componentDidMount() / useEffect((),static getDerivedStateFromProps(props, state)

Updating

When a component is being re-rendered as a result of changes to either its props or state."

📌 This is growth and adaptation — when your component responds to changes.

Key methods/hooks: shouldComponentUpdate(), render(), componentDidUpdate() / useEffect(() ,static getDerivedStateFromProps(),getSnapshotBeforeUpdate(prevProps, prevState)

Unmounting

When a component is being removed from the DOM.

📌 Death or exit — the component is cleaned up and taken away.

Key method/hook: componentWillUnmount() / useEffect(()

Error Handling

When there is an error during rendering, in a lifecycle method, or in the constructor of any child component.

📌 Recovery — React lets you gracefully handle failure.

Key method: componentDidCatch(error, info) / static getDerivedStateFromError()

Mounting Lifecycle:(23)

Constructor(props) => should never make a HTTP request.

static getDerivedStateFromProps(props, state) =>

- when the state of the component changes in prop overtime.
- Set the state(can not use this.setState,so we have to return)
- Do not cause any side effect.

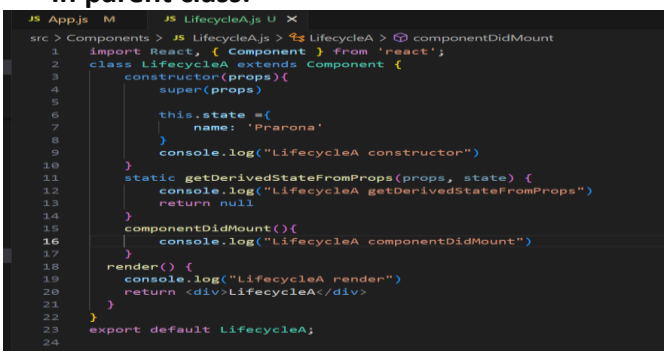
Render() =>

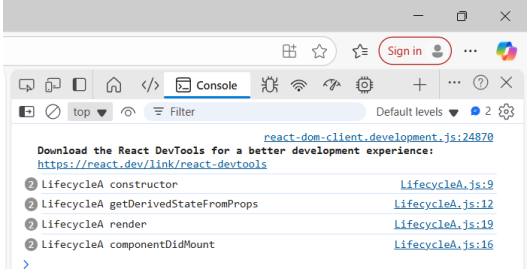
- read props and start and return JSX.
- Do not change state or interact with DOM or make ajax calls.
- Children components lifecycle methods are also executed.

componentDidMount() =>

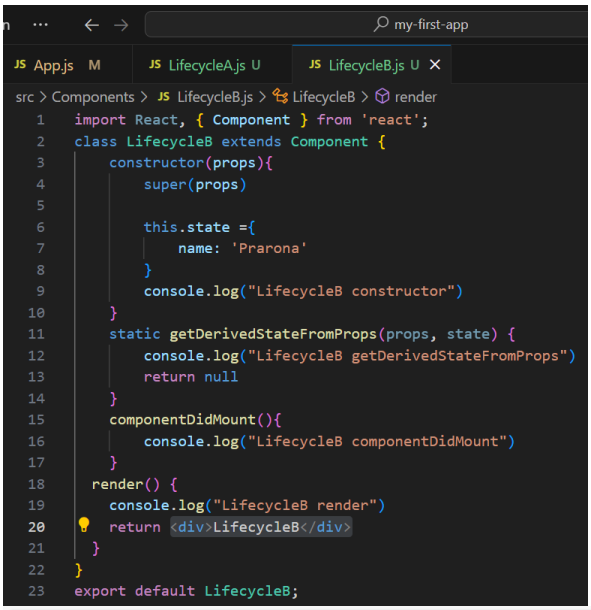
- It calls once in a whole lifecycle.
- Invoked immediately after a component and all its children components have been rendered to the DOM.
- Cause side effects. Ex: Interact with the DOM or perform any ajax calls to load data.

In parent class:



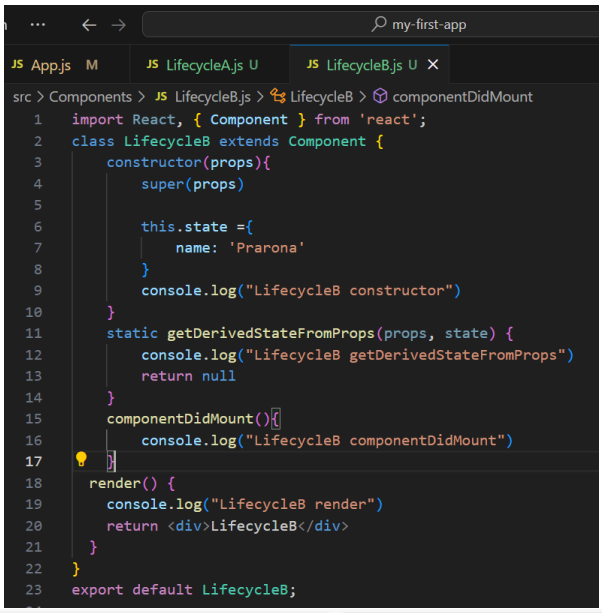


In Child class:



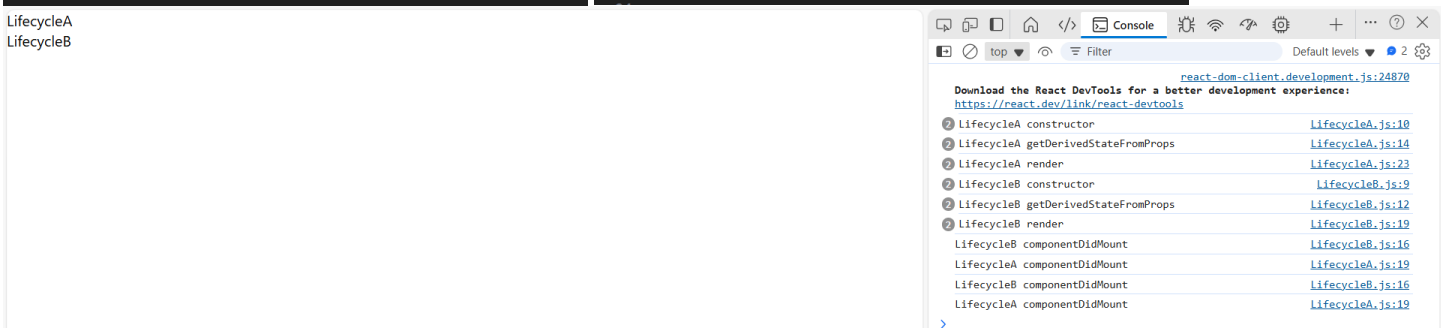
```
import React, { Component } from 'react';
class LifecycleB extends Component {
    constructor(props){
        super(props)

        this.state ={
            name: 'Prarona'
        }
        console.log("LifecycleB constructor")
    }
    static getDerivedStateFromProps(props, state) {
        console.log("LifecycleB getDerivedStateFromProps")
        return null
    }
    componentDidMount(){
        console.log("LifecycleB componentDidMount")
    }
  render() {
    console.log("LifecycleB render")
    return <div>LifecycleB</div>
  }
}
export default LifecycleB;
```



```
import React, { Component } from 'react';
class LifecycleB extends Component {
    constructor(props){
        super(props)

        this.state ={
            name: 'Prarona'
        }
        console.log("LifecycleB constructor")
    }
    static getDerivedStateFromProps(props, state) {
        console.log("LifecycleB getDerivedStateFromProps")
        return null
    }
    componentDidMount(){
        console.log("LifecycleB componentDidMount")
    }
  render() {
    console.log("LifecycleB render")
    return <div>LifecycleB</div>
  }
}
export default LifecycleB;
```



```
LifecycleA
LifecycleB
```

```
react-dom-client.development.js:24870
Download the React DevTools for a better development experience:
https://react.dev/link/react-devtools
LifecycleA constructor                          LifecycleA.js:10
LifecycleA getDerivedStateFromProps             LifecycleA.js:14
LifecycleA render                               LifecycleA.js:23
LifecycleB constructor                          LifecycleB.js:9
LifecycleB getDerivedStateFromProps             LifecycleB.js:12
LifecycleB render                               LifecycleB.js:19
LifecycleB componentDidMount                    LifecycleB.js:16
LifecycleA componentDidMount                    LifecycleA.js:19
LifecycleB componentDidMount                    LifecycleB.js:16
LifecycleA componentDidMount                    LifecycleA.js:19
```

We have lifecycleA constructor then lifecycleA get derived state from props that is followed by the render method of lifecycleA but after that we have the lifecycle methods of the child component which is lifecycleB.

Updating Lifecycle Methods:(24)

static getDerivedStateFromProps(props, state) =>

- **When: Called every time a component is re-rendered.**
- **Set or update state based on props.**
- **Dictates whether rendering needs to happen.**
- **Safe to use for updating state.**
- **No side effects (e.g., HTTP requests, setState).**

shouldComponentUpdate(nextProps, nextState) =>

- **When: After getDerivedStateFromProps, before render.**
- **Optimize performance by deciding if component should re-render.**
- **Good for optimization.**
- **No side effects or setState allowed.**

render() =>

- **When: Every update cycle.**
- **Returns JSX to update the UI.**
- **Reads props and state.**

getSnapshotBeforeUpdate(prevProps, prevState) =>

- **When: Right before the DOM is updated.**
- **Capture scroll position or DOM snapshot before the update.**
- **Value passed as the third parameter to componentDidUpdate.**

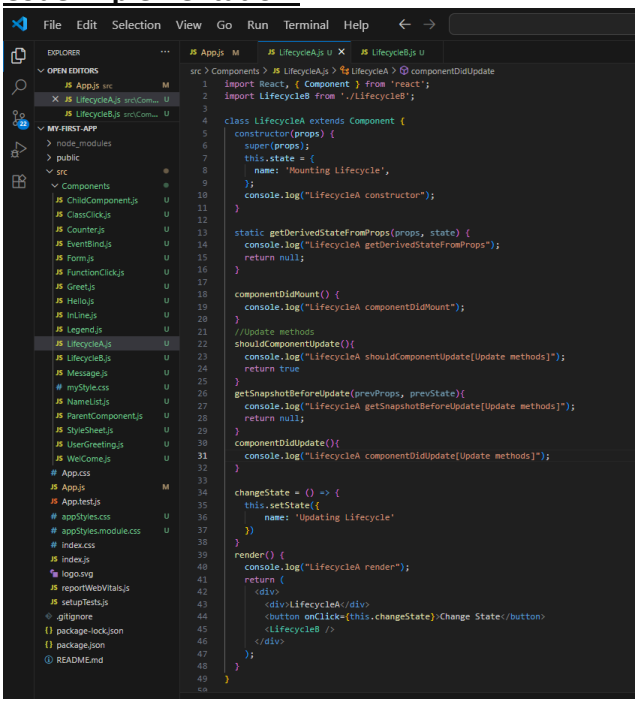
componentDidUpdate(prevProps, prevState, snapshot) =>

- **When: Immediately after updates are flushed to the DOM.**
- **Side effects (e.g., network requests, DOM ops).**
- **Perfect for actions based on the result of the update.**

Typical Flow of Update Phase

getDerivedStateFromProps → shouldComponentUpdate → render → getSnapshotBeforeUpdate → componentDidUpdate.

Code implementation:



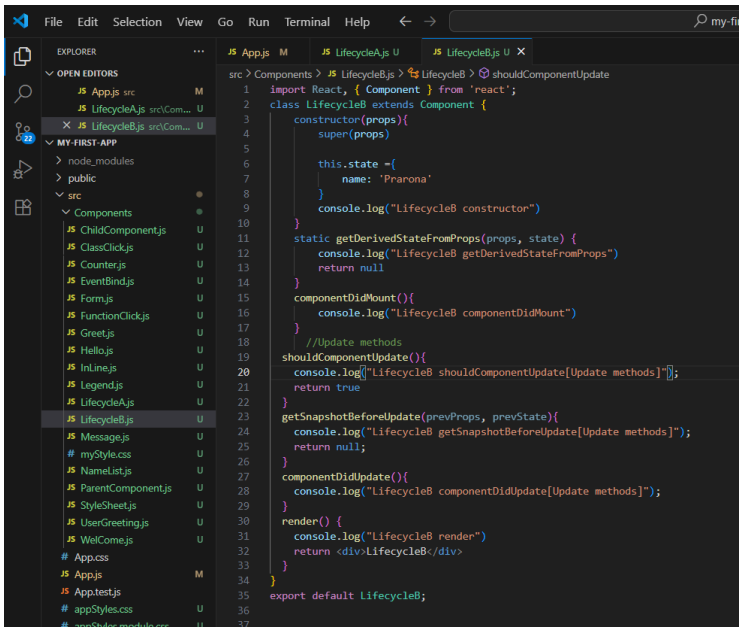
```
import React, { Component } from 'react';
import LifecycleB from './LifecycleB';

class LifecycleA extends Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'Mounting Lifecycle',
    };
    console.log("LifecycleA constructor");
  }

  static getDerivedStateFromProps(props, state) {
    console.log("LifecycleA getDerivedStateFromProps");
    return null;
  }

  componentDidMount() {
    console.log("LifecycleA componentDidMount");
  }
  //Update methods
  shouldComponentUpdate(){
    console.log("LifecycleA shouldComponentUpdate[Update methods]");
    return true
  }
  getSnapshotBeforeUpdate(prevProps, prevState){
    console.log("LifecycleA getSnapshotBeforeUpdate[Update methods]");
    return null;
  }
  componentDidUpdate(){
    console.log("LifecycleA componentDidUpdate[Update methods]");
  }

  changeState = () => {
    this.setState({
      name: 'Updating Lifecycle'
    })
  }
  render() {
    console.log("LifecycleA render");
    return (
      <div>
        <div>LifecycleA</div>
        <button onClick={this.changeState}>Change State</button>
        <LifecycleB />
      </div>
    );
  }
}
```

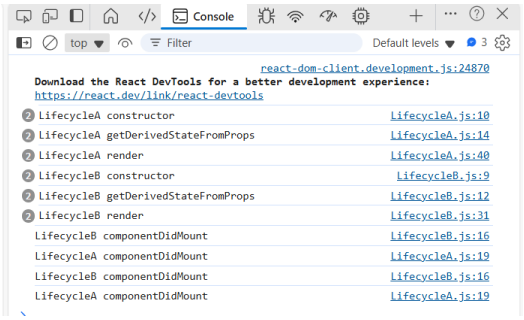


```
import React, { Component } from 'react';
class LifecycleB extends Component {
    constructor(props){
        super(props)

        this.state ={
            name: 'Prarona'
        }
        console.log("LifecycleB constructor")
    }
    static getDerivedStateFromProps(props, state) {
        console.log("LifecycleB getDerivedStateFromProps")
        return null
    }
    componentDidMount(){
        console.log("LifecycleB componentDidMount")
    }
     //Update methods
  shouldComponentUpdate(){
    console.log("LifecycleB shouldComponentUpdate[Update methods]");
    return true
  }
  getSnapshotBeforeUpdate(prevProps, prevState){
    console.log("LifecycleB getSnapshotBeforeUpdate[Update methods]");
    return null;
  }
  componentDidUpdate(){
    console.log("LifecycleB componentDidUpdate[Update methods]");
  }
  render() {
    console.log("LifecycleB render")
    return <div>LifecycleB</div>
  }
}
export default LifecycleB;
```

Before click:

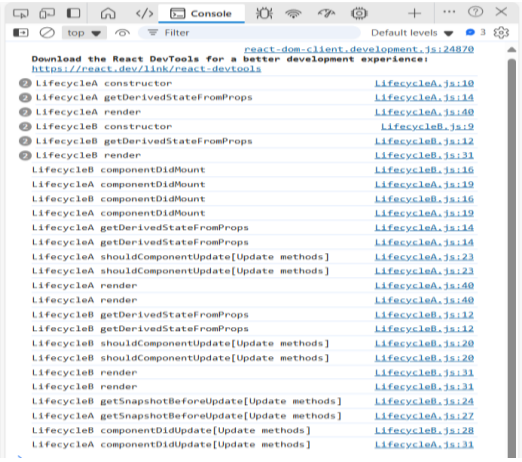
LifecycleA
Change State
LifecycleB



```
react-dom-client.development.js:24870
Download the React DevTools for a better development experience:
https://react.dev/link/react-devtools
LifecycleA constructor                          LifecycleA.js:10
LifecycleA getDerivedStateFromProps             LifecycleA.js:14
LifecycleA render                               LifecycleA.js:40
LifecycleB constructor                          LifecycleB.js:9
LifecycleB getDerivedStateFromProps             LifecycleB.js:12
LifecycleB render                               LifecycleB.js:31
LifecycleB componentDidMount                    LifecycleB.js:16
LifecycleA componentDidMount                    LifecycleA.js:19
LifecycleB componentDidMount                    LifecycleB.js:16
LifecycleA componentDidMount                    LifecycleA.js:19
```

After Click:

LifecycleA
Change State
LifecycleB



```
react-dom-client.development.js:24870
Download the React DevTools for a better development experience:
https://react.dev/link/react-devtools
LifecycleA constructor                          LifecycleA.js:10
LifecycleA getDerivedStateFromProps             LifecycleA.js:14
LifecycleA render                               LifecycleA.js:40
LifecycleB constructor                          LifecycleB.js:9
LifecycleB getDerivedStateFromProps             LifecycleB.js:12
LifecycleB render                               LifecycleB.js:31
LifecycleB componentDidMount                    LifecycleB.js:16
LifecycleA componentDidMount                    LifecycleA.js:19
LifecycleB componentDidMount                    LifecycleB.js:16
LifecycleA componentDidMount                    LifecycleA.js:19
LifecycleA getDerivedStateFromProps             LifecycleA.js:14
LifecycleA getDerivedStateFromProps             LifecycleA.js:14
LifecycleA shouldComponentUpdate[Update methods] LifecycleA.js:23
LifecycleA shouldComponentUpdate[Update methods] LifecycleA.js:23
LifecycleA render                               LifecycleA.js:40
LifecycleA render                               LifecycleA.js:40
LifecycleB getDerivedStateFromProps             LifecycleB.js:12
LifecycleB getDerivedStateFromProps             LifecycleB.js:12
LifecycleB shouldComponentUpdate[Update methods] LifecycleB.js:20
LifecycleB shouldComponentUpdate[Update methods] LifecycleB.js:20
LifecycleB render                               LifecycleB.js:31
LifecycleB render                               LifecycleB.js:31
LifecycleB getSnapshotBeforeUpdate[Update methods] LifecycleB.js:24
LifecycleA getSnapshotBeforeUpdate[Update methods] LifecycleA.js:27
LifecycleB componentDidUpdate[Update methods]   LifecycleB.js:28
LifecycleA componentDidUpdate[Update methods]   LifecycleA.js:31
```

Unmounting Phase — *"Closing time at the cafe"*

Before the cafe (component) closes, the barista (React) must:

- Cancel food orders (cancel async tasks)
- Turn off the music (remove event listeners)
- Clean the counters (clear timers, memory)

componentWillUnmount() {

```
 console.log("Component is unmounting...");
 clearInterval(this.timerID);      // Clear timers
 window.removeEventListener('resize', this.handleResize); // Remove listeners
 if (this.abortController) {
  this.abortController.abort();    // Cancel fetch requests
 }
}
```

⚠️ No setState() here — it's like sending an order to a kitchen that's already closed. 🚫

Error Handling Phase — *"The safety net under your React tightrope"*

🛡️ static getDerivedStateFromError(error)

Sets the stage for graceful UI degradation before disaster hits the DOM.

```
static getDerivedStateFromError(error) {
 return { hasError: true };
}
```

componentDidCatch(error, info)

Perfect for logging or sending issues to tools like Sentry, LogRocket, etc.

```
componentDidCatch(error, info) {
 console.error("Caught an error:", error, info);
 // sendErrorToServer(error, info.componentStack);
}
```