Lifecycle methods in React

Topics covered:

- What is the life cycle of a component?
- Mounting
    - constructor()
    - getDrivedStateFromProps()
    - render()
    - componentDidMount()
- Update
    - getDerivedStateFromProps()
    - shouldComponentUpdate()
    - render()
    - getSnapshotBeforeUpdate()
    - componentDidUpdate()
- Unmounting
    - componentWillUnmount()

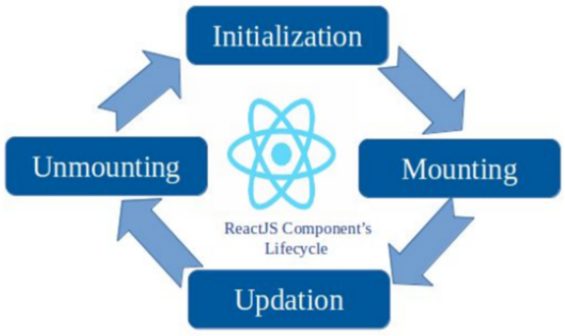
1. **Lifecycle of a component**:

The three major phases of each component's lifecycle in React can be used to monitor and modify each component.

It consists of three stages:

- Mounting (Inserting element in DOM)
- Updating (Modifying element in DOM)
- Unmounting (Removing element from DOM)

These represent the beginning, development, and ending of a component, respectively.



2. Mounting

When a React component is generated and added to the DOM, we can view it on the user interface or in the browser;

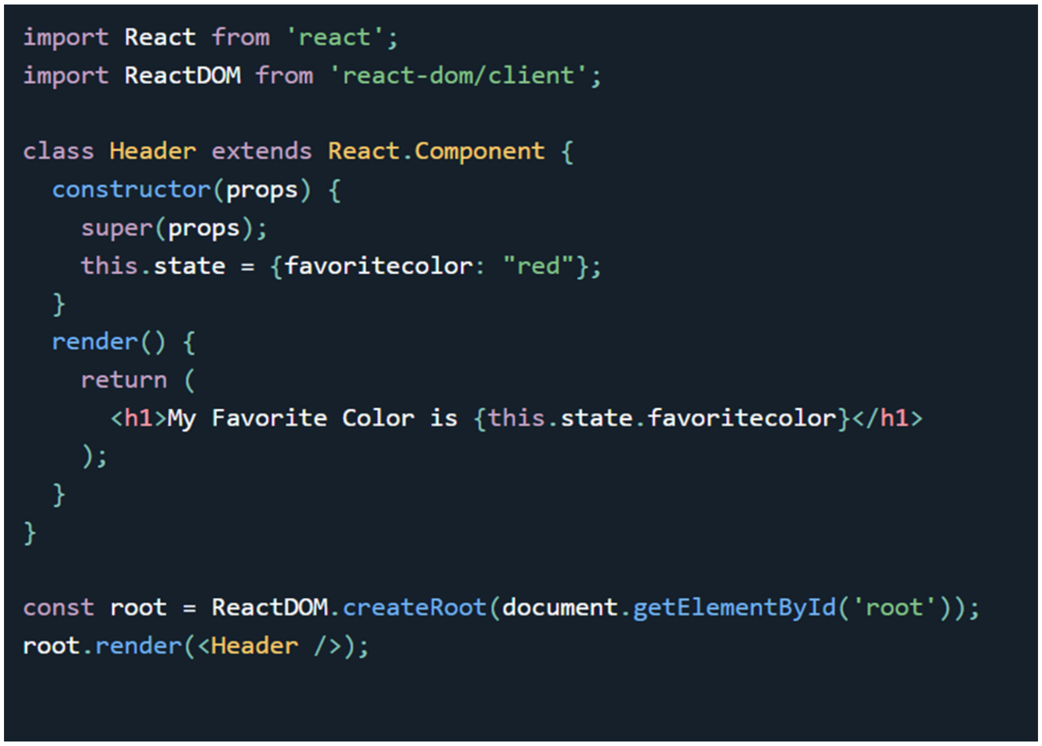
it goes through the following processes, listed here in the order they occur:

- Constructor.
- static getDerivedStateFromProps()
- render()
- componentDidMount()

constructor:

When the component is started, the Constructor() method is the first function that is executed, making it the obvious place to set up the initial state and other basic settings.

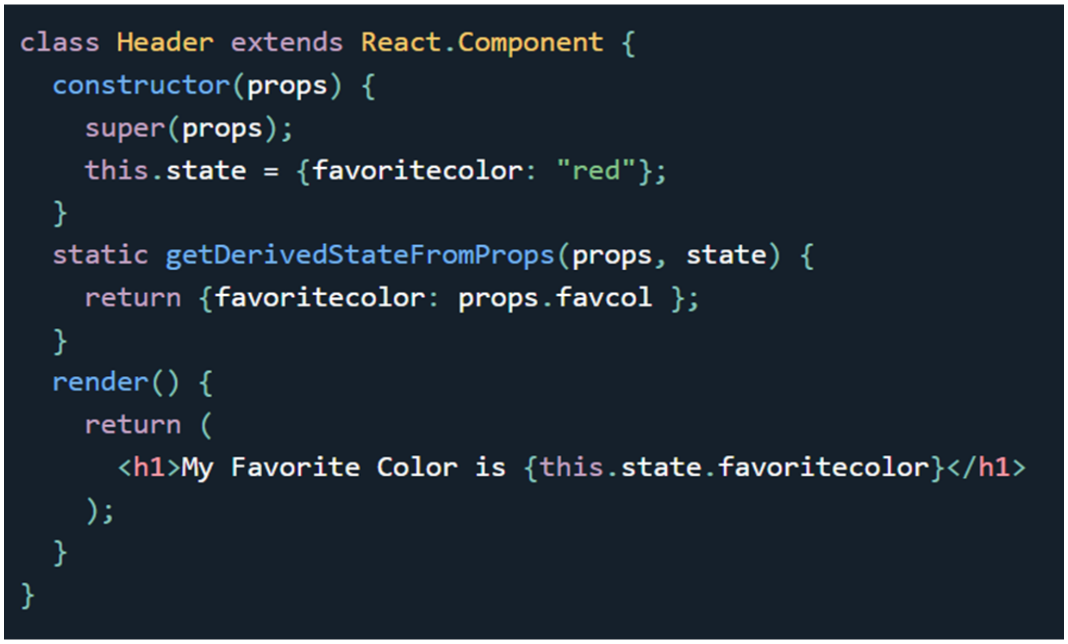
You should always use the super(props) method before any other method in order to start the parent's constructor procedure and allow the component to inherit methods from its parent. The Constructor() method is invoked with the props as arguments (React.Component).


```
import React from 'react';
import ReactDOM from 'react-dom/client';

class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

getDerivedStateFromProps():

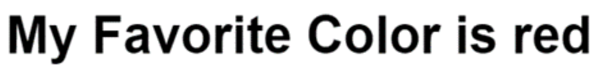
The second method that is called, both on the initial mount and future updates, is getDerivedStateFromProps().

- It is called just before invoking the render method.
- If the state needs to be updated, it must return an object; otherwise, it must return null.
- It is not frequently used and is only used when state derivation is necessary.



```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  static getDerivedStateFromProps(props, state) {
    return {favoritecolor: props.favcol };
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}
```

Output:



```
My Favorite Color is red
```

Render method

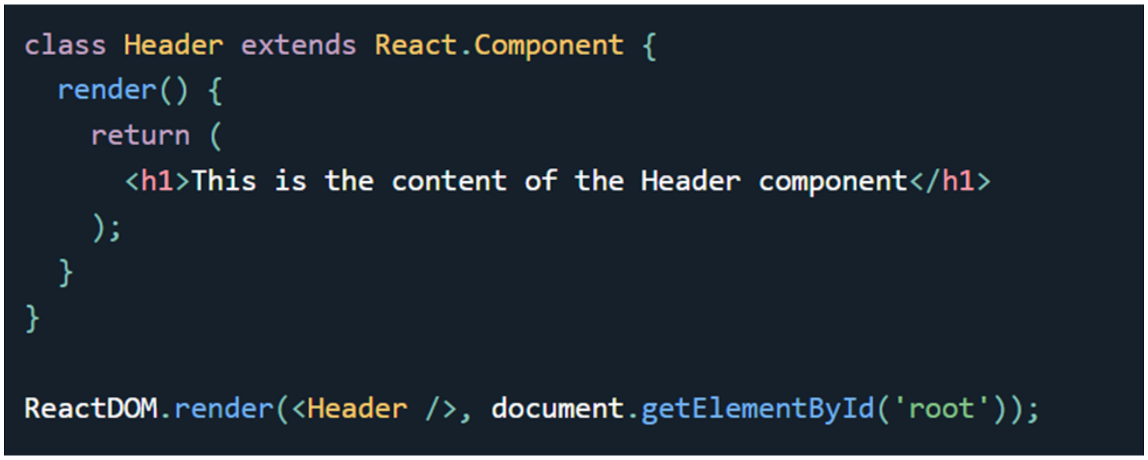
The third method is called render, and it is the only one that is absolutely necessary to render our class component.

It is called anytime there is a change in state or props, and it is largely used for printing the JSX in DOM.

Class Header extends Component{

    render(

```
                    //JSX

                        <React.Fragment>

                                <h1>Hello JavaScript Users </h1>

                        </React.Fragment>

          )
}
```

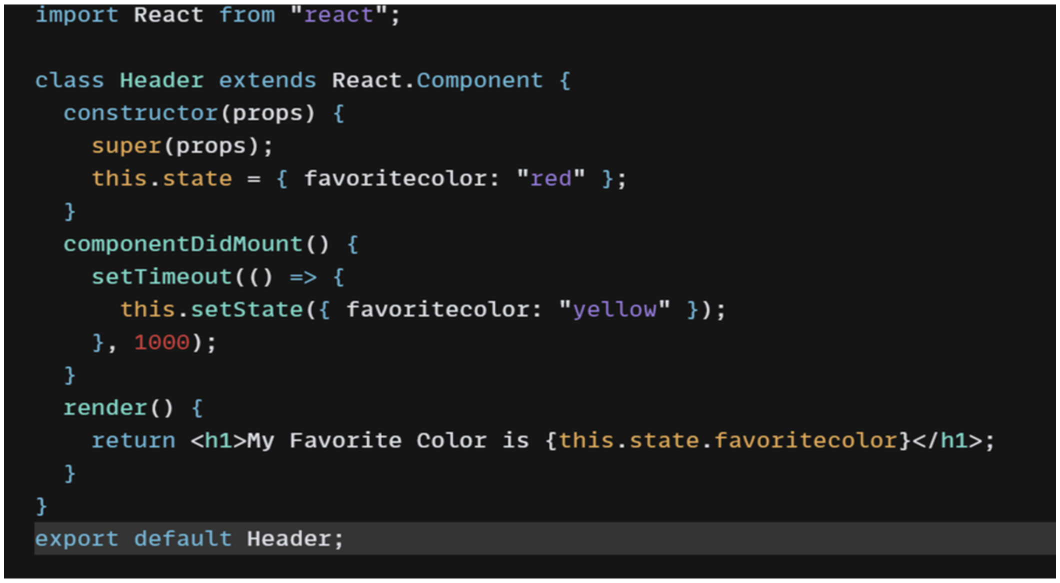


```
class Header extends React.Component {
  render() {
    return (
      <h1>This is the content of the Header component</h1>
    );
  }
}

ReactDOM.render(<Header />, document.getElementById('root'));
```

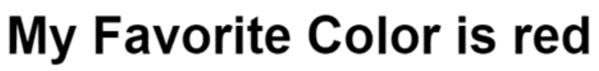
componentDidMount():

The final lifecycle method in the mounting stage, componentDidMount(), is called right away when a component is put into the DOM.

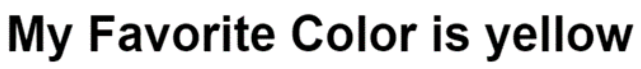
It is mostly used for making API calls and is only called once.


```
import React from "react";

class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = { favoritecolor: "red" };
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({ favoritecolor: "yellow" });
    }, 1000);
  }
  render() {
    return <h1>My Favorite Color is {this.state.favoritecolor}</h1>;
  }
}
export default Header;
```

This will be the output before 1000 milliseconds:


My Favorite Color is red

This will be the output after 1000 millisecond:


My Favorite Color is yellow

3. **Updating:**

The next stage in the lifecycle occurs when a component is updated.
Every time a component's state or props change, the component is updated.
When a component is changed, React's five built-in methods are called in the following order:

- static getDerivedStateFromProps()
- shouldComponentUpdate()

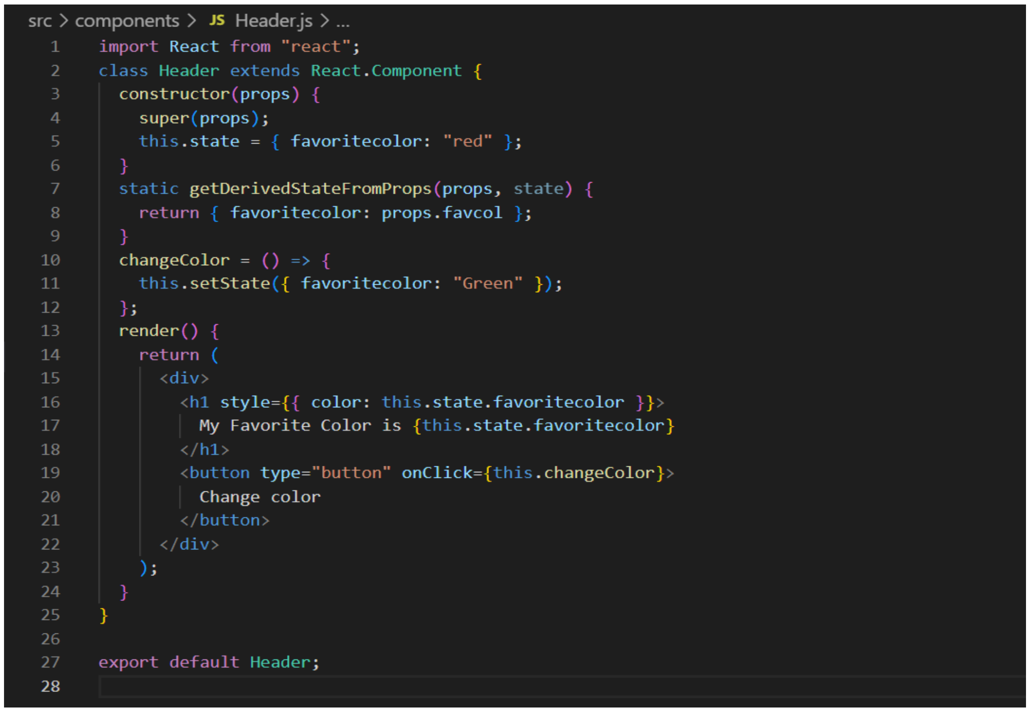
- render()
- getSnapshotBeforeUpdate()
- componentDidUpdate()

The ***render()*** method is essential and is always called; the others are optional and are only called if they are defined.

getDerivedStateFromProps:

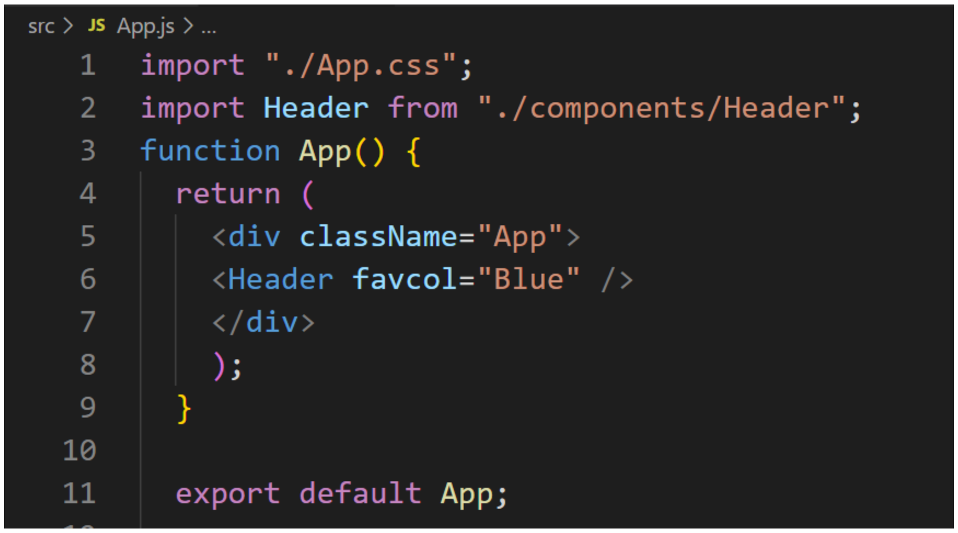
- The getDerivedStateFromProps function is also invoked during updates. When a component is updated, this is the first method that is called.
- This is still the correct location to set the state object based on the initial props.
- Two procedures that are used frequently during the Mounting stage are static getDerivedStateFromProps and *render()*.

The button in the following example changes the favorite color to green, however because the getDerivedStateFromProps() function is invoked, which updates the state with the color from the favcol attribute, the favorite color remains blue:



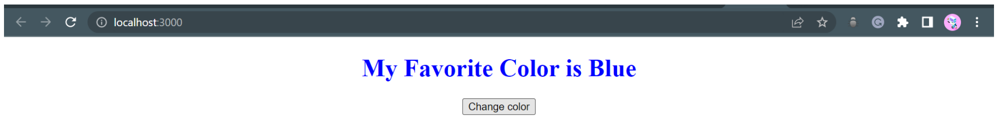
```
src > components > JS Header.js > ...
```

```javascript
import React from "react";
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = { favoritecolor: "red" };
  }
  static getDerivedStateFromProps(props, state) {
    return { favoritecolor: props.favcol };
  }
  changeColor = () => {
    this.setState({ favoritecolor: "Green" });
  };
  render() {
    return (
      <div>
        <h1 style={{ color: this.state.favoritecolor }}>
          My Favorite Color is {this.state.favoritecolor}
        </h1>
        <button type="button" onClick={this.changeColor}>
          Change color
        </button>
      </div>
    );
  }
}

export default Header;

```



```
src > JS App.js > ...
import "./App.css";
import Header from "./components/Header";
function App() {
  return (
    <div className="App">
    <Header favcol="Blue" />
    </div>
    );
  }

  export default App;
```

Output:

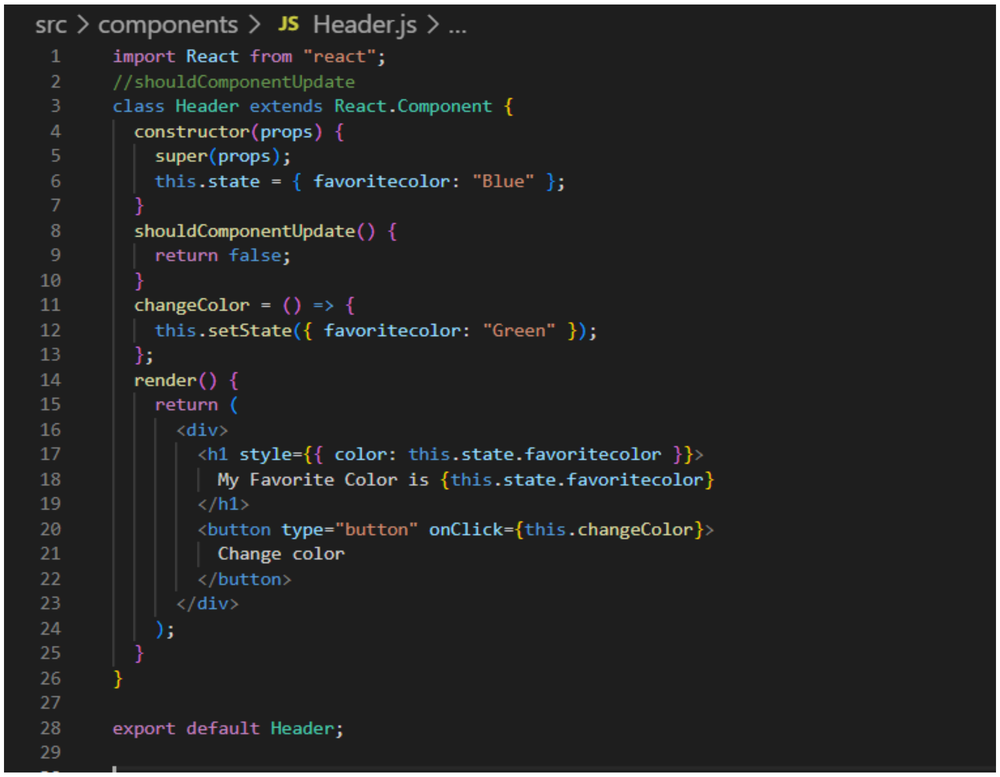


localhost:3000

My Favorite Color is Blue

shouldComponentUpdate:

- The shouldComponentUpdate() method returns a Boolean value indicating whether React should continue with the rendering or not.
- True is the default value.
- *shouldComponentUpdate()* is used to inform React whether the current change in state or props has no impact on a component's output.
- Re-rendering occurs automatically by default whenever a state changes.
- Similar to *componentDidMount()*, *componentDidUpdate()* is the final method to be called during the update phase. It is only called once, and when it is, it indicates that the DOM has been changed.
- After the DOM has been updated, *ComponentDidUpdate()* is typically used to operate on DOM elements.
- The following example illustrates what transpires when the shouldComponentUpdate() function returns false:

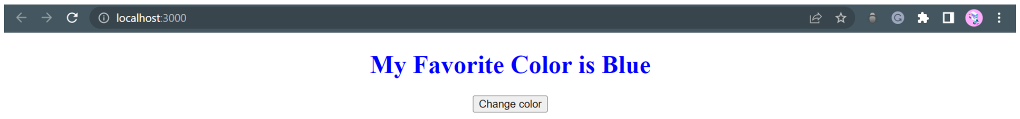


src > components > JS Header.js > ...

```
import React from "react";
//shouldComponentUpdate
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = { favoritecolor: "Blue" };
  }
  shouldComponentUpdate() {
    return false;
  }
  changeColor = () => {
    this.setState({ favoritecolor: "Green" });
  };
  render() {
    return (
      <div>
        <h1 style={{ color: this.state.favoritecolor }}>
          My Favorite Color is {this.state.favoritecolor}
        </h1>
        <button type="button" onClick={this.changeColor}>
          Change color
        </button>
      </div>
    );
  }
}

export default Header;
```

In this case, we send the value to shouldComponentUpdate, which returns false.

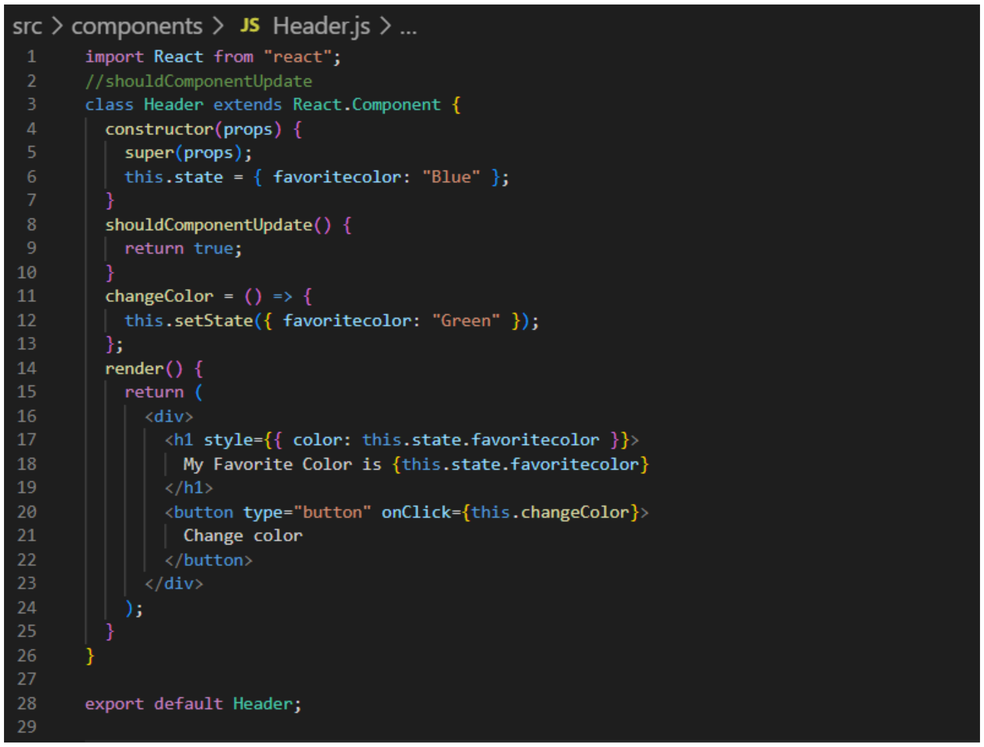
So, if we click the change color button in the output, there will be no change in the component because the value returned by shouldComponentUpdate is false.



localhost:3000

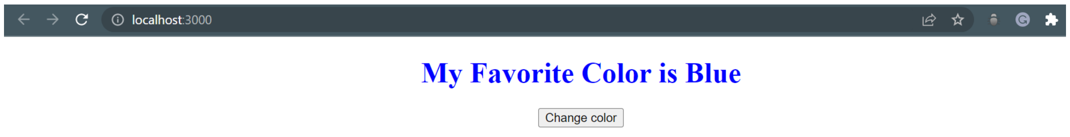
My Favorite Color is Blue

If we return the true value to shouldComponentUpdate, the component will be updated when we click the change color button.

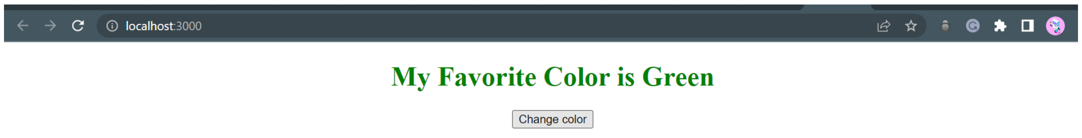

src > components > JS Header.js > ...

```
import React from "react";
//shouldComponentUpdate
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = { favoritecolor: "Blue" };
  }
  shouldComponentUpdate() {
    return true;
  }
  changeColor = () => {
    this.setState({ favoritecolor: "Green" });
  };
  render() {
    return (
      <div>
        <h1 style={{ color: this.state.favoritecolor }}>
          My Favorite Color is {this.state.favoritecolor}
        </h1>
        <button type="button" onClick={this.changeColor}>
          Change color
        </button>
      </div>
    );
  }
}

export default Header;
```

Output:


localhost:3000

My Favorite Color is Blue

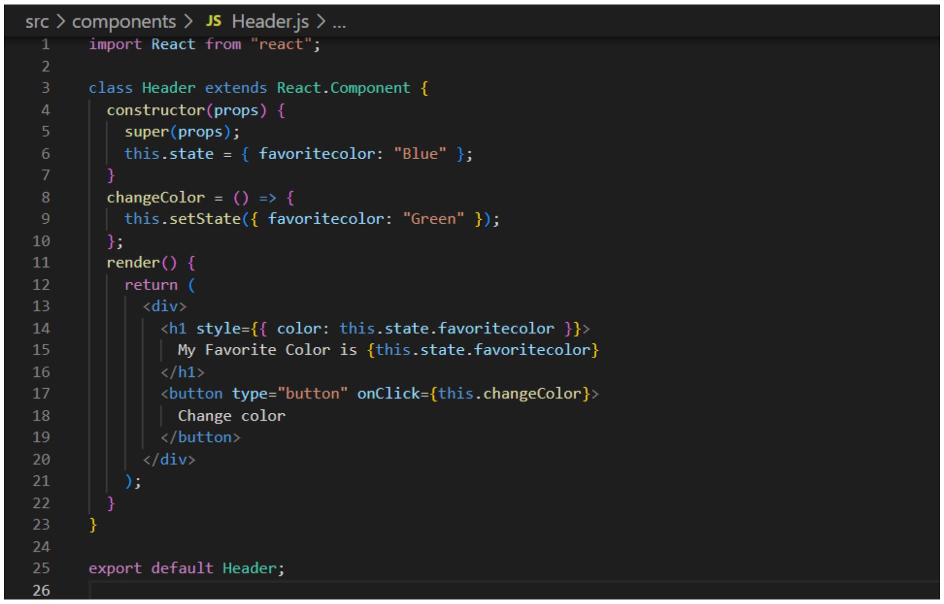

localhost:3000

My Favorite Color is Green

render():

When a component is updated, it must re-render the HTML to the DOM with the new changes, which calls the render() method.

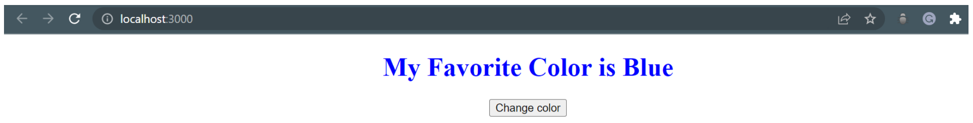
The button in the example below switches the favorite color to Green:



src > components > JS Header.js > ...

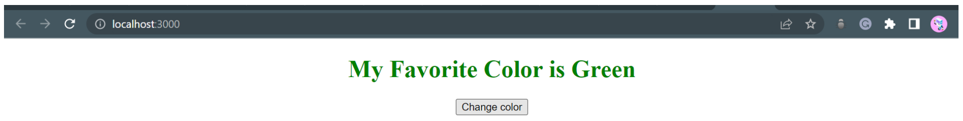
```jsx
import React from "react";

class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = { favoritecolor: "Blue" };
  }
  changeColor = () => {
    this.setState({ favoritecolor: "Green" });
  };
  render() {
    return (
      <div>
        <h1 style={{ color: this.state.favoritecolor }}>
          My Favorite Color is {this.state.favoritecolor}
        </h1>
        <button type="button" onClick={this.changeColor}>
          Change color
        </button>
      </div>
    );
  }
}

export default Header;
```

Output:



localhost:3000

My Favorite Color is Blue



localhost:3000

My Favorite Color is Green

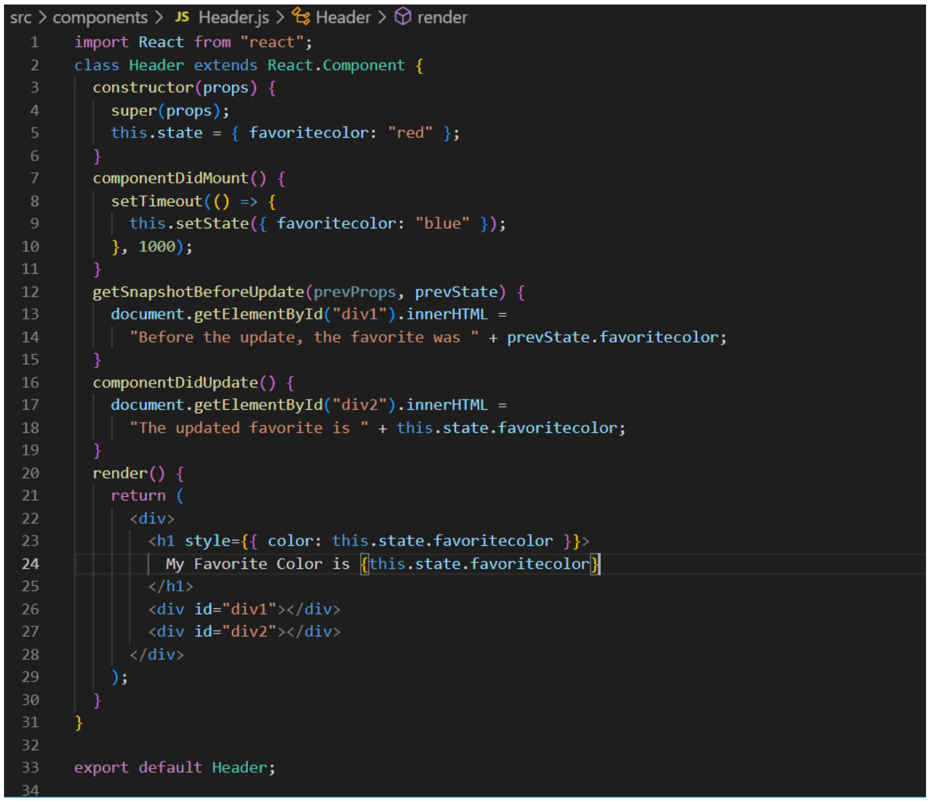
getSnapshotBeforeUpdate():

- The getSnapshotBeforeUpdate() method gives you access to the props and state from before the update, so you can examine the values from before the update even after it has occurred.
- The componentDidUpdate() function must be present in addition to the getSnapshotBeforeUpdate() method in order to avoid an error.

Although the example below appears complex, all it accomplishes is the following:

- The preferred color "red" is displayed when the component is mounting.
- After mounting the component, a timer modifies the state, and after one second, the preferred color switches to "yellow."
- Since this component includes a getSnapshotBeforeUpdate() method, it is called when the update phase is triggered. This method writes a message to the empty DIV1 element.
- The empty DIV2 element is then written with the following message by the componentDidUpdate() method:

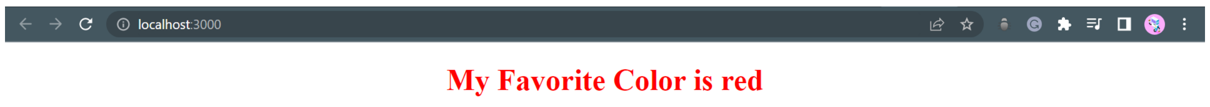
To see what the state object looked like before the change, use the getSnapshotBeforeUpdate() method:



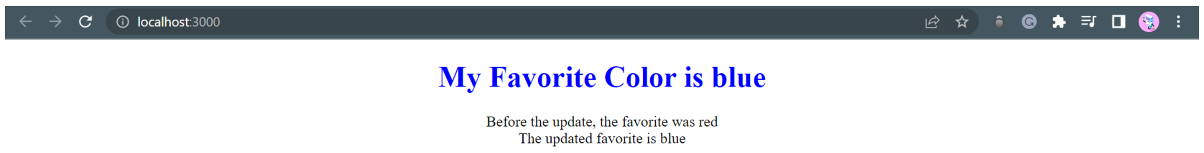
src > components > Header.js > Header > render

```
import React from "react";
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = { favoritecolor: "red" };
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({ favoritecolor: "blue" });
    }, 1000);
  }
  getSnapshotBeforeUpdate(prevProps, prevState) {
    document.getElementById("div1").innerHTML =
      "Before the update, the favorite was " + prevState.favoritecolor;
  }
  componentDidUpdate() {
    document.getElementById("div2").innerHTML =
      "The updated favorite is " + this.state.favoritecolor;
  }
  render() {
    return (
      <div>
        <h1 style={{ color: this.state.favoritecolor }}>
          My Favorite Color is {this.state.favoritecolor}
        </h1>
        <div id="div1"></div>
        <div id="div2"></div>
      </div>
    );
  }
}

export default Header;
```

Output:


localhost:3000

My Favorite Color is red

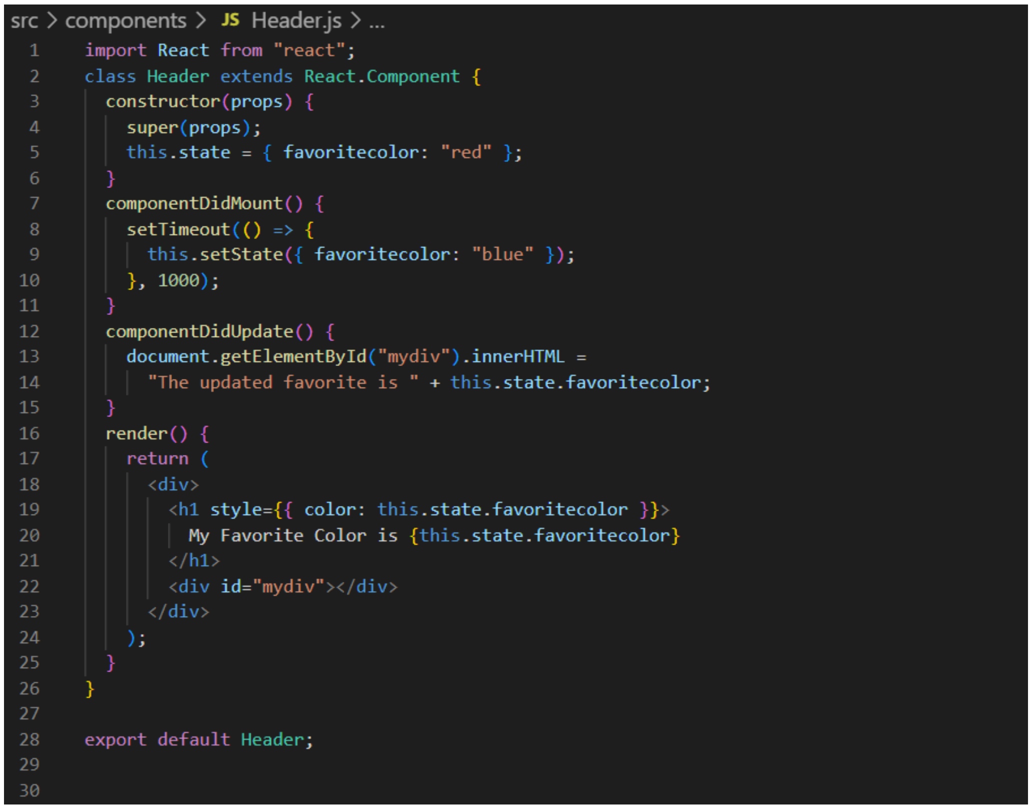

localhost:3000

My Favorite Color is blue

Before the update, the favorite was red
The updated favorite is blue

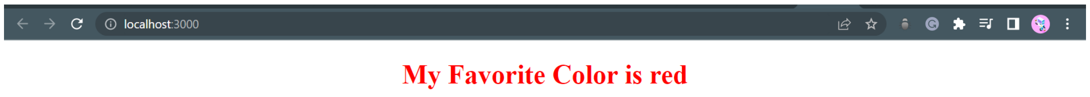
componentDidUpdate():

- After the component has been updated in the DOM, the componentDidUpdate function is called.
- The following example may appear difficult, however it only does the following:
- When the component is mounted, the color "red" is used to represent it.
- A timer changes the state and the color to "blue" after the component is mounted.
- This action initiates the update phase, and because this component includes a componentDidUpdate function, this method is invoked and a message is written to the empty DIV element:

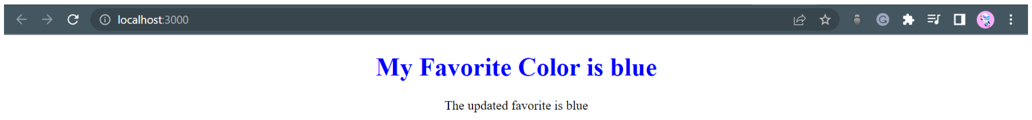


```
src > components > JS Header.js > ...
import React from "react";
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = { favoritecolor: "red" };
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({ favoritecolor: "blue" });
    }, 1000);
  }
  componentDidUpdate() {
    document.getElementById("mydiv").innerHTML =
      "The updated favorite is " + this.state.favoritecolor;
  }
  render() {
    return (
      <div>
        <h1 style={{ color: this.state.favoritecolor }}>
          My Favorite Color is {this.state.favoritecolor}
        </h1>
        <div id="mydiv"></div>
      </div>
    );
  }
}

export default Header;
```

Output:



My Favorite Color is red



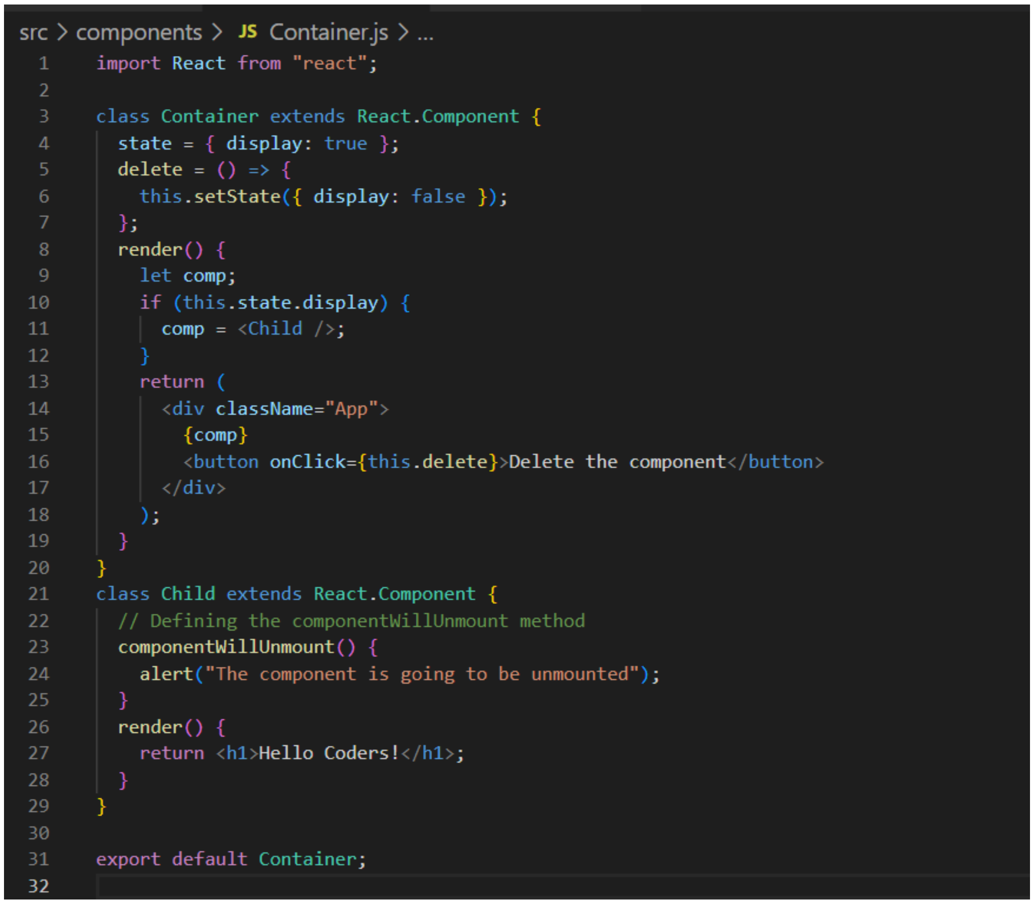
My Favorite Color is blue

The updated favorite is blue

4. Unmounting:

The final phase of a component's lifecycle, unmounting, marks the end of the component's existence in the DOM.

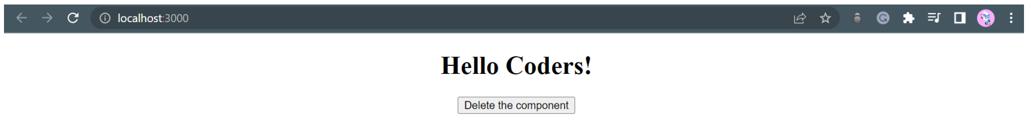
- componentWillUnmount() is the main method it contains.
- Before a component is unmounted and destroyed, componentWillUnmount() is called.
- It is mostly used to clean up our code, such as cancelling API calls or removing any subscriptions generated by componentDidMount ().
- Here is an example of unmounting:



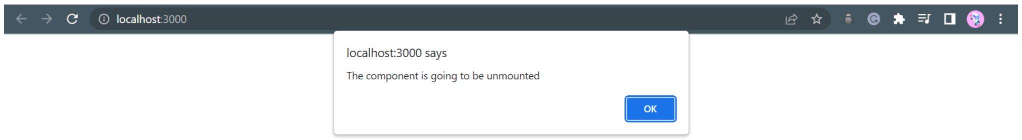
src > components > JS Container.js > ...

```
import React from "react";

class Container extends React.Component {
  state = { display: true };
  delete = () => {
    this.setState({ display: false });
  };
  render() {
    let comp;
    if (this.state.display) {
      comp = <Child />;
    }
    return (
      <div className="App">
        {comp}
        <button onClick={this.delete}>Delete the component</button>
      </div>
    );
  }
}
class Child extends React.Component {
  // Defining the componentWillUnmount method
  componentWillUnmount() {
    alert("The component is going to be unmounted");
  }
  render() {
    return <h1>Hello Coders!</h1>;
  }
}

export default Container;

```

Here is the Output:



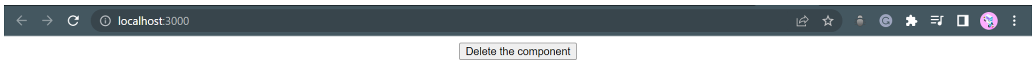
Hello Coders!



localhost:3000

localhost:3000 says

The component is going to be unmounted



localhost:3000