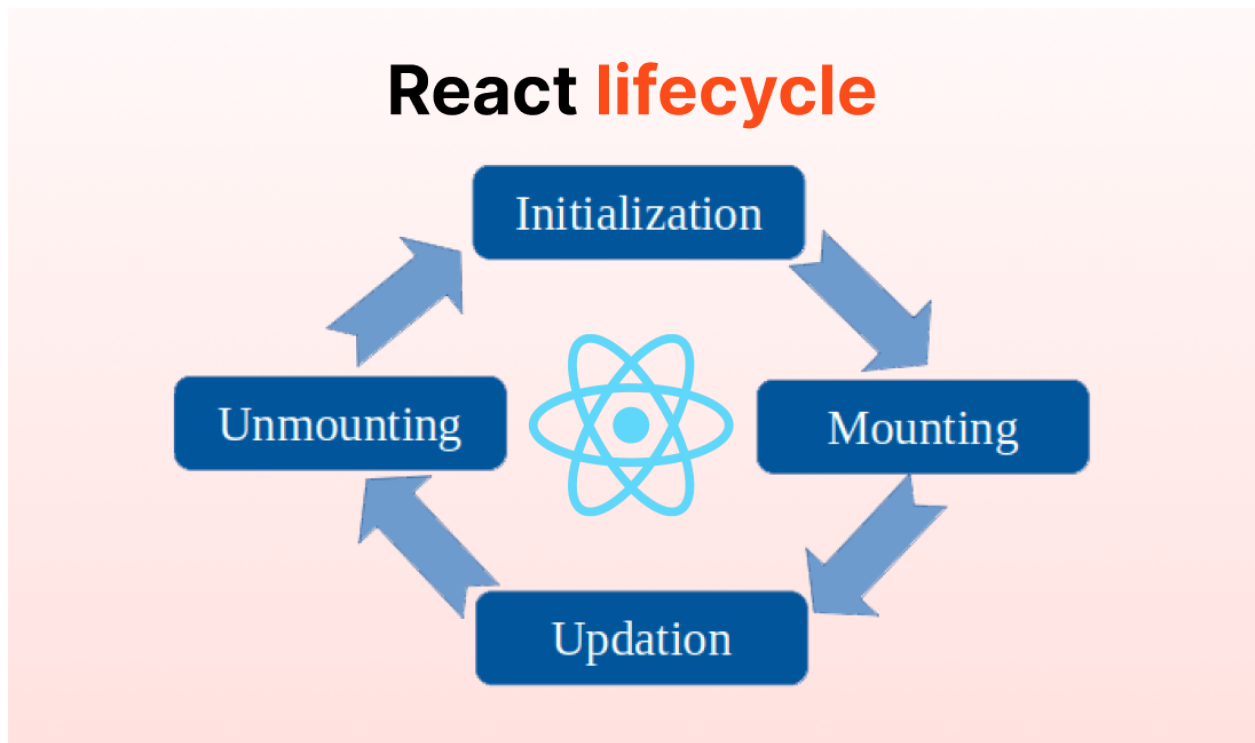


MODULE: 10 List and Hooks

- Explain Life cycle in Class Component and functional component with Hooks.

Answer: The React component lifecycle can be divided into three main phases: Mounting, Updating and Unmounting. Each phase has various methods invoked at specific points in the component's lifecycle.



A table summarizing the different stages of the React component lifecycle and the methods associated with each stage:

| Lifecycle Stage | Methods |
|-----------------|-----------------------------------|
| Mounting | constructor() |
| | static getDerivedStateFromProps() |
| | render() |
| | componentDidMount() |
| Updating | static getDerivedStateFromProps() |
| | shouldComponentUpdate() |
| | render() |
| | getSnapshotBeforeUpdate() |
| | componentDidUpdate() |
| Unmounting | componentWillUnmount() |
| Error Handling | static getDerivedStateFromError() |
| | componentDidCatch() |

Mounting:

- Mounting means putting elements into the DOM.
- React has four built-in methods that gets called, in this order, when mounting a component:
 1. constructor()
 2. getDerivedStateFromProps()
 3. render()
 4. componentDidMount()

- The render() method is required and will always be called, the others are optional and will be called if you define them.

1.constructor:

- The constructor() method is called before anything else, when the component is initiated, and it is the natural place to set up the initial state and other initial values.
- The constructor() method is called with the props, as arguments, and you should always start by calling the super(props) before anything else, this will initiate the parent's constructor method and allow the component to inherit methods from its parent (React.Component).

Example:

The constructor method is called, by React, every time you make a component:

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

```
}
```

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

2.getDerivedStateFromProps:

- The `getDerivedStateFromProps()` method is called right before rendering the element(s) in the DOM.
- This is the natural place to set the state object based on the initial props.
- It takes state as an argument, and returns an object with changes to the state.

Example:

The example below starts with the favorite color being "red", but the `getDerivedStateFromProps()` method updates the favorite color based on the `favcol` attribute:

The `getDerivedStateFromProps` method is called right before the render method:

```
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>  
  
  );  
  
}  
  
}  
  
ReactDOM.render(<Header favcol="yellow"/>, document.getElementById('root'));
```

3.render:

- The render() method is required, and is the method that actually outputs the HTML to the DOM.

Example:

A simple component with a simple render() method:

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

4.componentDidMount:

- The `componentDidMount()` method is called after the component is rendered.
- This is where you run statements that require that the component is already placed in the DOM.

Example:

At first my favorite color is red, but give me a second, and it is yellow instead:

```
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>  
  
    );  
  
  }  
  
}
```

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

Updating:

- The next phase in the life cycle is when a component is updated.
- A component is updated whenever there is a change in the component's state or props.
- React has five built-in methods that gets called, in this order, when a component is updated:
 1. `getDerivedStateFromProps()`
 2. `shouldComponentUpdate()`
 3. `render()`
 4. `getSnapshotBeforeUpdate()`
 5. `componentDidUpdate()`
- The `render()` method is required and will always be called, the others are optional and will be called if you define them.

1.getDerivedStateFromProps:

- Also at updates the `getDerivedStateFromProps` method is called. This is the first method that is called when a component gets updated.
- This is still the natural place to set the state object based on the initial props.

Example:

If the component gets updated, the `getDerivedStateFromProps()` method is called:

```
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: "blue"});
}

render() {
  return (
    <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Change color</button>
    </div>
  );
}
}

ReactDOM.render(<Header favcol="yellow"/>, document.getElementById('root'));

```

2.shouldComponentUpdate:

- In the shouldComponentUpdate() method you can return a Boolean value that specifies whether React should continue with the rendering or not.
- The default value is true.

Example:

The example below shows what happens when the `shouldComponentUpdate()` method returns false:

Stop the component from rendering at any update:

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

```
}  
}
```

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

3.render:

- The render() method is of course called when a component gets updated, it has to re-render the HTML to the DOM, with the new changes.

Example:

The example below has a button that changes the favorite color to blue:

Click the button to make a change in the component's state:

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

```
<button type="button" onClick={this.changeColor}>Change color</button>
</div>

);
}
}

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

4.getSnapshotBeforeUpdate:

- In the getSnapshotBeforeUpdate() method you have access to the props and state before the update, meaning that even after the update, you can check what the values were before the update.
- If the getSnapshotBeforeUpdate() method is present, you should also include the componentDidUpdate() method, otherwise you will get an error.

Example:

The example below might seem complicated, but all it does is this:

When the component is mounted it is rendered with the favorite color "red". When the component has been mounted, a timer changes the state, and after one second, the favorite color becomes "yellow".

This action triggers the update phase, and since this component has a getSnapshotBeforeUpdate() method, this method is executed, and writes a message to the empty DIV1 element.

Then the `componentDidUpdate()` method is executed and writes a message in the empty `DIV2` element:

Use the `getSnapshotBeforeUpdate()` method to find out what the state object looked like before the update:

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

    <div id="div1"></div>

    <div id="div2"></div>

  </div>

);

}

}

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

```

5.componentDidUpdate:

- The componentDidUpdate method is called after the component is updated in the DOM.

Example:

The example below might seem complicated, but all it does is this:

When the component is mounted it is rendered with the favorite color "red".When the component has been mounted, a timer changes the state, and the color becomes "yellow".

This action triggers the update phase, and since this component has a componentDidUpdate method, this method is executed and writes a message in the empty DIV element:

The componentDidUpdate method is called after the update has been rendered in the DOM:

```
class Header extends React.Component {

  constructor(props) {

    super(props);

    this.state = {favoritecolor: "red"};
  }

  componentDidMount() {

    setTimeout(() => {

      this.setState({favoritecolor: "yellow"})

    }, 1000)

  }

  componentDidUpdate() {

    document.getElementById("mydiv").innerHTML =

    "The updated favorite is " + this.state.favoritecolor;

  }

  render() {

    return (

      <div>

        <h1>My Favorite Color is {this.state.favoritecolor}</h1>

        <div id="mydiv"></div>

      </div>

    );

  }

}
```

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

Unmounting:

- The next phase in the life cycle is when a component is removed from the DOM, or unmounting as React likes to call it.
- React has only one built-in method that gets called when a component is unmounted:
 1. `componentWillUnmount()`

1.componentWillUnmount:

- The `componentWillUnmount` method is called when the component is about to be removed from the DOM.

Example:

Click the button to delete the header:

```
class Container extends React.Component {  
  
  constructor(props) {  
  
    super(props);  
  
    this.state = {show: true};  
  
  }  
  
  delHeader = () => {  
  
    this.setState({show: false});  
  
  }  
}
```

```
render() {  
  
  let myheader;  
  
  if (this.state.show) {  
  
    myheader = <Child />;  
  
  };  
  
  return (  
  
    <div>  
  
      {myheader}  
  
      <button type="button" onClick={this.delHeader}>Delete Header</button>  
  
    </div>  
  
  );  
  
}  
  
}  
  
class Child extends React.Component {  
  
  componentWillMount() {  
  
    alert("The component named Header is about to be unmounted.");  
  
  }  
  
  render() {  
  
    return (  
  
      <h1>Hello World!</h1>  
  
    );  
  
  }  
  
}
```



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