Component Life Cycle Method

Each component has lifecycle that can be manipulated at particular time in process. Some phases of lifecycle are used frequently whereas some are used rarely.

3 Types of Component Lifecycle phases:

1. Mounting
2. Updating
3. Unmounting

Each phase has some methods which we can implement as per the requirement:

**Mounting**

1. Constructor()
2. Static getDerivedStateFromProps()
3. render()
4. componentDidMount()

**Updating**

1. Static getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

**Unmounting**

1. componentWillUnmount()

# Mounting

## Constructor()

If you want to initialize the state then you have to use the constructor.

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 allows the component to inherit methods from its parent (React.Component)

Here we initalized state to 4.

constructor(props) {

super(props);

this.state = {count:4};

}

## getDerivedStateFromProps()

This is called before rendering the elements. This is a rare use case. Here you can set the state object based on the initial props. It maybe called multiple times for a single update, so it’s important to avoid any side-effects. It takes state as an argument and returns an object with changes to the state.

static getDerivedStateFromProps(props, state) {

if (props.selected !== state.selected) {

return {

selected: props.selected,

};

}

## render()

This is a required method. It outputs actual HTML.

render() {

return (

<h1>Bootcamp Training</h1>

);

}

}

## componentDidMount()

This is called after component is rendered. This method is a good place to setup any subscriptions This is a method which requires the component is already placed in the DOM.

componentDidMount() {

**this.timerID = setInterval(**

**() => this.tick(),**

**1000**

**);**

**}**

# Updating

## getDerivedStateFromProps()

This is called before rendering the elements. This is a rare use case. Here you can set the state object based on the initial props. It maybe called multiple times for a single update, so it’s important to avoid any side-effects. It takes state as an argument and returns an object with changes to the state.

static getDerivedStateFromProps(props, state) {

if (props.selected !== state.selected) {

return {

selected: props.selected,

};

}

## ShouldComponentUpdate()

This returns a Boolean value that specifies whether React should continue rendering or not. The default value is true

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {name: "Archit"};

}

shouldComponentUpdate() {

return true;

}

changeName = () => {

this.setState({name: "Shah"});

}

render() {

return (

<div>

<h1>My name is {this.state.name}</h1>

<button type="button" onClick={this.changeName}>Change Name</button>

</div>

);

}

}

## render()

This is a required method. It outputs actual HTML.

render() {

return (

<h1>Bootcamp Training</h1>

);

}

}

## getSnapshotBeforeUpdate()

This is to save the value before the update. If we use this we also need to include componentDidUpdate method as well or else we will get an error

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'));

## ComponentDidUpdate()

This method is called after the component is updated in the DOM. An example of when to use componentDidUpdate() is when we need to call an external API on condition that the previous state and the current state have changed.

The call to the API would be conditional to the state being changed. If there is no state change, no API is called.

componentDidUpdate(prevProps, prevState) {

if (prevState.pokemons !== this.state.pokemons) {

console.log('pokemons state has changed.')

}

}

# Unmounting

## componentWillUnmount()

This is called before a component is unmounted. It is used to perform any necessary cleanups. Such as removing timer, cancelling network request or cleaning the subscriptions

componentWillUnmount(){

console.warn('componentWillUnmount call')

alert('User has been deleted');

}