

27/02/24 episode - 8 let's get clarity 😊

→ We will learn about class based component.

normal JS class.

for interviews

understanding of react will become very strong.

Syntax: our class comp. is using some properties of react component. → How lifecycle works.

class Name extends React.Component {

Keyword

}; component name → This will tell react that, this class is a react component.

React.Component is a class which is given to us by react.

So import it. import React from 'react';

our class based component is using some properties of React.Component class

class MyComponent extends React.Component {

render () { basically our class compo. is inheriting essential functions & properties from React.Component, which are essential for a class based component to work. }

return (

Hello class based component

)

This will be rendered on to UI

; };

class based components have a render method which returns some

piece of JSX.

→ What is a class based component?

Ans A class based component is a normal JS class, which has a render method which returns a piece of JS.

When you are using class based component, don't forget to import React from 'react';

→ Just a small change in syntax, else everything is same as functional component.

* How to receive props in class based components.

→ import React from 'react';

<class User extends React.Component {

 constructor(props) { → A constructor of a class is called at every class instance

 super(props); → Do 2 things:-

 } };

You can also do destructuring const {name} = this.props received from render () {

<div> {this.props.name}</div>; this.props properties.

→ We can access our props using this.props only in class component.

→ Using super(props) is because without it we will not be able to use this.props.

* How to use / create state variables inside class based component

→ Remember one thing! constructor is the best place to initialize props as well as state variables.

constructor(props) {

 super(props);

 this.state = {

 state variable1: count: 0,

 state variable2: count2: 1,

 };

This is the place where we will declare & initialize all our state variables.

initial value.

→ Remember: If you want to do destructuring do it inside render() in class based component

```
render() {
```

const [count, count2] = this.state;

```
return (
```

[Behind the scenes react is using a big single object to keep all our state variables.]

→ How we do it in func component

```
import {useState} from 'react';
```

```
const useX = () => {
```

```
return const [count, setCount] = useState(0);
```

```
const [count2, setCount2] = useState(1);
```

Above return.

```
return (
```

```
<div> {count} {count2} </div>
```

→ In func. component to create state variable we use **useState HOOK**.

→ How → to update state variables inside a class based component.

```
<button onClick={() => {
```

```
this.setState({
```

you cannot write
count: count + 1

count: this.state.count + 1,
count2: this.state.count2 + 1,

```
} );
```

- For updating State Variable in class-based components we use `this.setState()` method and it takes an object as an argument.
 - Inside that object you can update your state variable as `state-variable-name: updated value`.
 - Inside that single object you can update more than one state variable, if you want as per your need.
 - When react will encounter `this.setState`, it will go into argument which we have passed, it will look for the name/s of the state variable and it will only update that state variables.
 - After updating it will re-render the whole component, and it will only update that state variable on DOM.
 - You can also destructure your React component like
- ```
import {Component} from 'react';
class App extends Component {
 render() {
 return (
 <h1>Hello World!</h1>
);
 }
}
```
- Better definition is written in next page ①
- Mount: Mount means component has been loaded on the UI. Loading is same as mounting of a component.
  - As we say a component has been loaded onto the screen. is equivalent to saying a component is mounted onto the screen.

\* ~~class~~ based component

Lifecycle methods

- Very firstly whenever a component is initiated → Then very firstly the class's ~~constructor~~ <sup>constructor</sup> method is called / executed.
- Instance / Instantiation of a class basically means calling a class based component.
- In JSR as soon as we create an object of a class, it's constructor is called.
- Similarly here whenever we call a C.B.C, its constructor is executed or called.

For example:

```
const comp = () => {
 return (
 < class Comp />
);
};
```

Here we are calling this class based component. As soon as react will encounter this line, then an instance of that class `< class Comp />` based component will be created.

- After executing the constructor `render()` method gets executed.
- After executing `render()` method, `componentDidMount()` method gets executed.



→ class Parent extends React.Component {

```
 render() {
 return (
 < childComp />
);
 };
};
```

- flow or life cycle of above class based component will be
- ① constructor of parent class will be called.
  - ② Render of parent class will be called.
  - ③ constructor of child class will be called.
  - ④ Render of child class will be called.
  - ⑤ componentDidMount of child will be called.
  - ⑥ componentDidMount of child Parent class will be called.

- When React will execute Parent (class based component) then lifecycle methods of parent class will start executing.
- As soon as it will encounter child (class based component) then lifecycle methods of parent class will be paused, and lifecycle methods of the child (class based component) will start getting executed.
- After completing the lifecycle methods of child (C.B.C), lifecycle methods of parent (C.B.C) will resume.

Note: ~~constructor will be executed only once at the time of instantiation of the class based component.~~

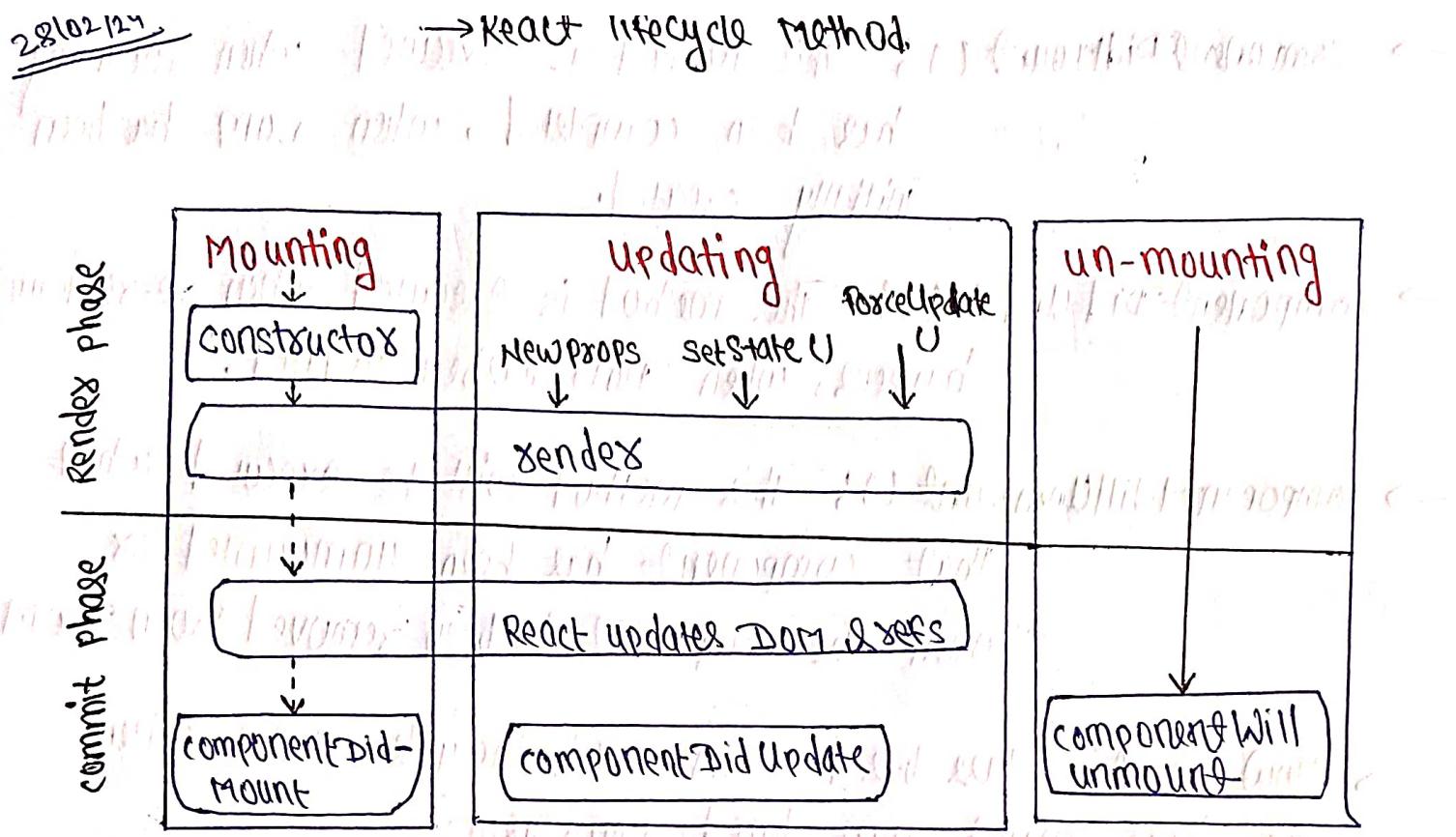
Similarly componentDidMount will only be executed once after completing the ~~initial~~ Mounting (Initial Rendering)

↓  
Mounting means putting element into DOM.

→ Loading is also same, and putting elements into the DOM is done just once & then React simply updates the DOM.

- (a)
- Mounting is when React renders the component for the first time and actually builds the initial DOM for from those instruction. (This DOM did not exist already)
  - A "re-render" is when React calls the function component for the first time again to get a new set of instructions on already mounted component.
  - In a class based component, if a state changes then React calls the `render()` method of that component. Only `render` method of that class based component is executed, `constructor` or `componentDidMount` does not get executed again.
  - In class based component we use `componentDidMount` for making API call, as we use `useEffect()` in functional component.
  - The reason why we use `componentDidMount` is that it will be executed after the mounting of the component.  
Initial rendering / 1st rendering only.  
component has been placed onto the DOM for the first time.





- Disclaimer :- Do not compare the react lifecycle method of class based components to functional components.
- React class based component's life cycle is divided into 3 parts
  - 1) Mounting : Putting the component onto DOM for 1st time
  - 2) Updating : Making changes into DOM or simply re-rendering with updated state variables.
  - 3) Un-Mounting : Removing component from DOM, or when that component has been removed from the screen.
- There are 2 phase in mounting and updating :-
- 1) Render phase : In this phase constructor and render method of the component is executed.
- 2) Commit phase : In this phase DOM is updated and componentDidMount or componentDidUpdate or componentWillUnmount method is called/executed.

- componentDidMount(): This method is executed when mounting has been completed, when DOM has been initially created.
- componentDidUpdate(): This method is executed when re-rendering happens, when DOM has been updated.
- componentWillUnmount(): This method will be executed when that component has been unmounted or when the component has been removed from DOM.
- How React's class based component's lifecycle works when we have nested class based component.

```
import React from 'react';

class Parent extends React.Component {
 constructor(props) {
 super(props);
 this.componentDidMount() {
 // ...
 }
 this.componentDidUpdate() {
 // ...
 }
 this.componentWillUnmount() {
 // ...
 }
 }
 render() {
 // ...
 return (
 <div>
 <Child />
 </div>
);
 }
}
```

- When react have more than one child components made inside any parent components, then it will their combine or batch their rendering and commit phases.
- Rendering phase of both child components will take place 1<sup>st</sup> and then commit <sup>phase</sup> of both child components will take place together.
- And then commit phase of parent component will take place. See long in code for more better understanding.
- But why react batches or combines the rendering phase of all the components together & commit phase of all the components together?
  - Because DOM manipulation is a very expensive operation in react. So to fast our DOM manipulation process, react basically checks what all component it want to add into DOM and then it do DOM updation only once. which increase our DOM updation/manipulation process.