Date.....

Chapter 08 Let's get classy

Earlier react used to just have class based components, there's was nothing like functional components there was no concept of hooks (no concept useState, no concept of useEffect). And it used to be very painful to write code bt it was very magical at that time because a lot developers who are coming from the era JQuery and you move to react it used to be super amazing thing:, there was reconcilliation process which was not very effective like it is today but it was still very effective.

Reasons for not using class-based components.
1) when we write class-based component, the code is very much messy, it's not clear.
2) class-based components used to be very big as compared to func. comp.

→ from previous lecture
(How do we created nested route [children inside children route])

We can simply write it inside our children component.

eg.

```
const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    errorElement: <ERROR/>,
    children: [
      {
        path: "/about",
        element: <About/>
        children: [
          { path: "profile",
            element: <profile/>
          },
        ],
      },
    ],
  },
]);
```

if you will write "/profile" ye interprete karega "http://localhost:1234/profile"

Spiral,

Teacher's Sign .....

```
{ path: "/"
  element: <Body />,
},
{ path: "/contact",
  element: <Contact/>
},
{ path: "restaurant/:resId"
  element: <RestaurantMenu />,
},
],
},
```

also we have ~~to~~ add <profile/> component in <About/> so in About.js we have to use <Outlet/>
// in About.js.

```
import { Outlet } from "react-router-dom";

const About = () => {
  return (
    <div>
      <h1> About us page</h1>
      <p>
        This is the Namaste React Live Course
      </p>
      <Outlet/>  ~~~> we can directly import & render our profile component as well
    </div>
  );
};

export default About;
```

Class base component :- A class based component end of the day is a class.

So if we want to write our class based component for "profile".

```
// our function based component for profile.

const profile = () => {
    return (
        <div>
            <h2> Profile component </h2>
        </div>
    );
};
export default profile;

// our class based component

import React from "react";

class Profile extends React.Component {
    render() {
        return <h1> Profile component </h1>;
    }
}
export default Profile;
```

you are telling JS that this is not normal class of JavaScript

name of class based component

only mandatory method in class-based component is the render method.

what ever you return from this will be injected to the dom.

In functional component, it is a function that returns some JSX, class base component has a render method and now this render method returns some JSX.

Date.....

passing props in class-based component

```
import React from "react";

class Profile extends React.Component {

    render() {
        return (
            <div>
            <h1>profile class component </h1>
            <h2> Name : {this.props.name} </h2>
            </div>
        );
    }
}
```

for

```
<profile name={"Akshay"} xyz={"pqr"} />
```

create state variable in class-based component

```
import React from "react";
class Profile extends React.Component {
    constructor (props) {
        super(props);
        //create state
        this.state = {
            count: 0
            count1: 0,
        };
    }
    render() {
        <div>
            <h1> Profile class component </h1>
            <h2> Count : {this.state.count} </h2>
        </div>
```

isko pehle object destructure use kar sakte hai

```
{count} = this.state;
```

Spiral

Teacher's Sign.....

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

    // We do not mutate STATE Directly
    // Never do this.state = something
    this.setState( {

        count : 1,
        count1 : 2,
    } );

    }
    },
}> setCount </button>.
```

ek hi baar mein jitne chahe state set kar skte hai

So Summary of above.

CLASS BASED COMPONENTS

→ It was less maintainable, have more code and a little messy.
→ They are no longer used.
→ Developers can do almost everything using functional components now.
→ Functional components, at the end is just normal js func.

Similarly ↓

→ class based components are just normal js class.

Date.....

CLASS BASED COMPONENT	FUNCTIONAL COMPONENT

import React from "react";
class Profile extends React.component {
render() {
return   Profile
}
}
export default Profile;
	const Profile = () => {
	return (
	  Profile
	);
	}
	export default Profile;

React.component come from here

You've to tell react that this is a class component not javascript class.

name of the component

```
import React from "react";
class Profile extends React.component {
```

Keyword class

render() { => return some JSX

you can't create a class component without render method

```
return <h1> profile </h1>;
}
```

→ whatever you return from this will be injected to the DOM

```
export default Profile;
```

render() { } is the only mandatory method for class-based components.

Date.....

PROPS IN CLASS COMPONENTS

In About.js, suppose we called class component 'profile.js' as :- and if we pass props inside it :-

```
<Profile class name = {"piyush K"} />
```

So, in class component, we have 'this' Keyword.
So, in 'profile.js"

```
<h2> Name : {this.props.name} </h2>
```

```
<Profile class name = {"Ashnaya"} xyz = {"abc"} />
```

Even if there are many props, react will collect all these props and attach with keyword 'this'. And I can use the props like :-

```
<h2> Name : {this.props.name} </h2>
<h2> xyz : {this.props.xyz} </h2>
```

STATE IN CLASS COMPONENT

Constructor function is called only once, when the component is first created & initialized, making it an ideal place to set the initial state of the component using

```
this.state = {...}
```

Date.....

class profile extends React.component {

constructor (props); {
super (props);
}

→ creates and initialize the component's state

this state = {
count : 0,
count2 : 1,
};

This is necessary because it passes the received props to the constructor of the base component class, allowing the component access it's properties.

↳ yani this.props mein props dalega.

- whenever we load a class, a constructor is called.
- To use the state we created :

```
render () {
  return (
    <h2> Count : {this.state.count} </h2>
    <h2> Count2 : {this.state.count2} </h2>
  );
}
```

- setCount fn

we do not mutate state directly

```
render () {
  return (
    <h2> Count : {this.state.count} </h2>
    <button
      onClick = {() => {
        this.setState ({
          count : 1, count2 : 2,
        })
      }}> setCount </button>
```

Spiral

Teacher's Sign .....

Date.....

React - Lifecycle

→ first constructor is called.
→ Then, component is rendered.

React component lifecycle refers to the series of methods that get executed at different stages of a component's existence in React application.
The lifecycle methods can be divided into 3 phases :→

1. Mounting Phase
2. Updating phase
3. Unmounting Phase

Mounting Phase

→ These methods are called when an instance of a component is being created & inserted into the DOM:

- constructor()
- render()
- ComponentDidMount() → This method is called immediately after the first render of a component. It is executed only once during the lifecycle of a component.

Updating phase :

→ These methods are called when a component is updated in response to changes in its props or state :-

- **shouldComponentUpdate()** –
  This method is called before a component is updated. It returns a boolean value indicating whether the component should be updated or not.

- **componentWillUpdate()** –
  This method is called just before a component is updated. It is only executed if shouldComponentUpdate() is true.

- **render()** –
  Used to render the component after it has been updated.

- **componentDidUpdate()** –
  This method is called immediately after a component is updated. It is only executed if shouldCompleteUpdate() is true.

Unmounting phase :

→ The phase where component is being removed from the DOM.

- **componentWillUnmounting()** – F
  This method is called just before a component is removed from the DOM.

Date.....

Best Place to make API call in class Components

→ componentDidMount() {...}

This is because during mounting phase.

1) constructor() → is called
then, 2) render() → is called
after 3) componentDidMount() → is called.

Eg: class ApidataFetchexample extends React.Component {

```
constructor(props) {
    super(props);
    // create state
    this.state = {
        userInfo: {
            name: "Dummy Name",
            location: "Dummy location",
        },
    };
    console.log("child
};

async componentDidMount() {
    //API calls
    const data = await fetch("https://api.github.com/...");
    const json = await data.json();
    this.setState({
        userInfo: json,
    });
}
```


Teacher's Sign.....

Date.....

```
render() {
  const return (
    <div>
      <h1> Profile class Comp. </h1>
      <img src={this.state.userInfo.url}/>
      <h2>{this.state.userInfo.name}</h2>
    </div>
  );
```

CORE BASIC OF CLASS COMPONENT

About.js (Parent Component)

```
class About extends React.Component {
  constructor (props) {
    super(props);
    console.log("Parent Const");
  }

  componentDidMount() {
    console.log("Parent - Component
                 Did Mount");
  }

  render() {
    console.log("Parent - render");
    return (

    ); }

}
```

Profile.js (child component)

```
class Profile extends React.Component {
  constructor (props) {
    super(props);
    console.log("child - constructor");
  }

  componentDidMount() {
    console.log("child - componentDid
                 Mount");
  }

  render() {
    console.log("child - render");
    return (
      <h1> Profile class </h1>
    ); }

}
```

① improvisation

→ In above eg,

<About/> is the parent component

<profile/> is the child component of <About/>

→ In what order the above code will execute?

① Parent - Constructor

② Parent - render

③ child - constructor

④ child - render

⑤ child - componentDidMount

⑥ Parent - componentDidMount

(In render() of About it sees <profile/> and it will trigger the lifecycle method of this children component)

Date.....

Another case

If <About/> component have 2 children :→
first child & second child

Let's see how it will be executed :-

About.js

```
class About extends React.Component {

    constructor(props) {

        super(props);
        console.log("Parent - constructor");
    }
    componentDidMount() {

        console.log("Parent - componentDidMount");
    }
    render() {

        console.log("Parent - render");
        return (

            <>

            <Profile name={"first child"} />

            <Profile name={"second child"} />

            </>
        );
    }

}
```

Date.....

Profile Class.js

```
class Profile extends React.Component {

    constructor(props) {
        super(props);
        this.state = {
            count: 0,
        };
        console.log("child-constructor" + this.props.name);
    }

    componentDidMount() {
        console.log("child-componentDidMount" +
                    this.props.name);
    }

    render() {
        console.log("child-render" + this.props.name);
        return (
            <h1> Profile class </h1>
        );
    }

}
```

Date.....

Order of execution :-

1. Parent - Constructor
2. Parent - Render
3. first child - constructor
4. first child - Render
5. Second child - constructor
6. Second child - Render

COMMIT PHASE STARTS ↓

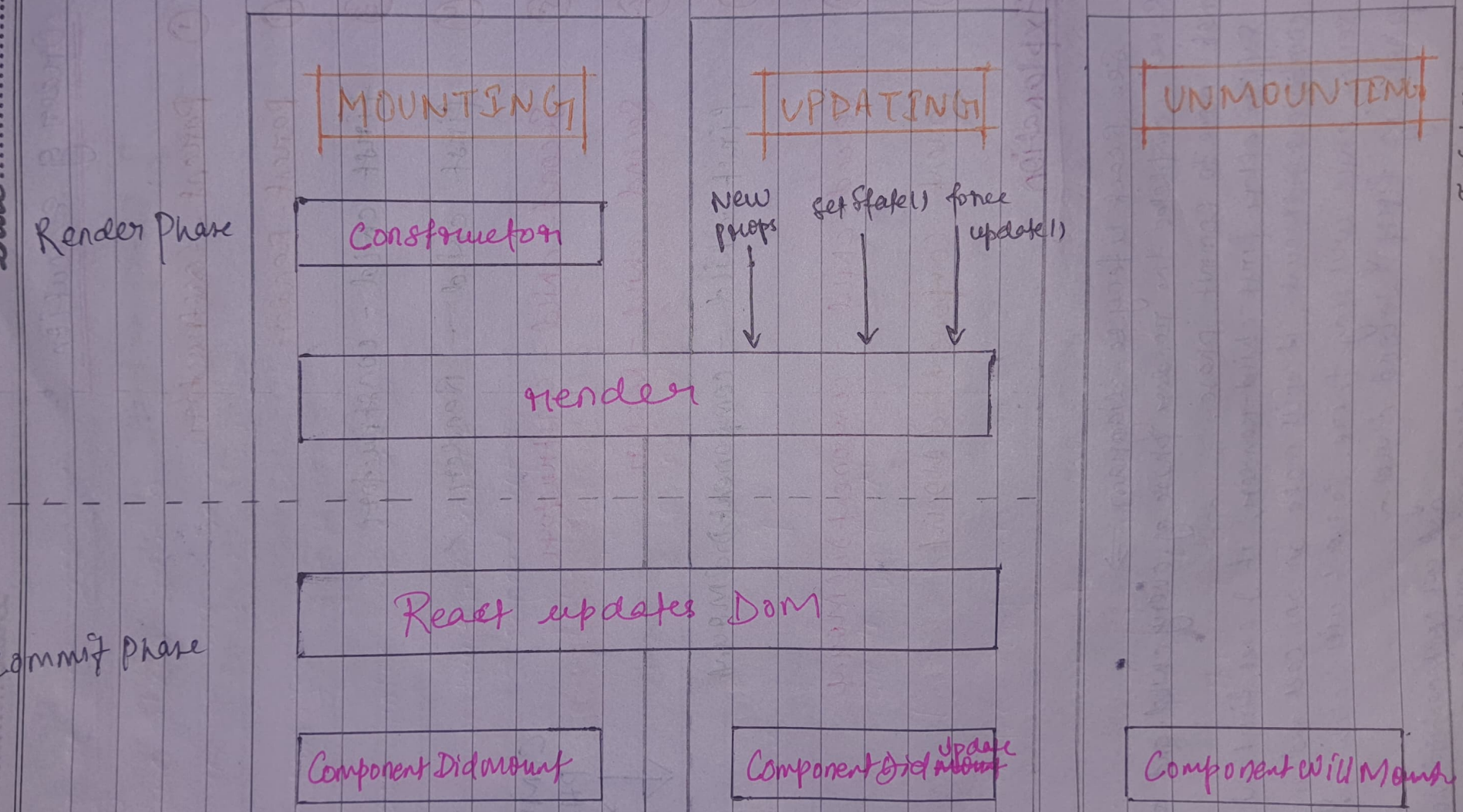
7. first child - componentDidMount
8. Second child - componentDidMount
9. Parent - componentDidMount

Explanation

See React lifecycle diagram ⟹
React completes the render phase of every child and after that move to commit phase.
Suppose after first-child-render, if I see firstchild-componentDidMount, it will make an api call and my second child will stay there. So react will batch the render phase of first & second child.)
↓ and call componentDidMount for children based mein.

Teacher's Sign .....

LIFE CYCLE



Date.....

Teacher's Sign.....

Spiral

Date.....

React do rendering in 2 phases :-

1. Render phase
2. Commit phase

First of all, react finishes the RENDER PHASE.

Render phase: → is fast

→ includes constructor and render method

Commit phase:

→ Phase where react modifies the DOM.
→ componentDidMount is called after the initial render has finished.
→ Commit phase is slow.

Date.....

Making an Api call

- Let's use github user api
- make an api call in the child component

profileclass.js :-

```
class Profile extends React.Component {

    constructor(props) {
        super(props);
        this.state = {
            userInfo : {
                name : "PKJ",
                location : "Delhi",
            }
        }
        console.log("child-constructor");
    }

    async componentDidMount() {
        const data = await fetch("https://api.github");
        const json = await data.json();
        console.log(json);
        this.setState({
            userInfo : json,
        });
        console.log("child-component did mount");
    }
```


Teacher's Sign .....

Date.....

```
render() {

    console.log("child-render");
    return (
        <>
        <h1> Name : {this.state.userInfo.name} </h1>
        <img src={this.state.userInfo.avatar_url} />
        <h2> Location : {this.state.userInfo.location} </h2>
        </>
    )
}
```

Sequence of method called in above code

I have parent 'About.js' inside our child 'profile.js'.

1. Parent - Constructor
2. Parent - Render
3. child - constructor
4. child render
5. ~~API call~~
5. Parent - componentDidMount } is called before mapping api call

This is because, React finishes render cycle first and then it goes to commit cycle. As child - componentDidMount, will take some time for the data to load, parent - componentDidMount is called before.

So hence, this sequence → next page.

Date.....

1. Parent - constructor
2. Parent - render
3. child - ~~render~~ constructor
4. child - render
5. DOM is updated
6. json is logged in console
7. Parent - componentDidMount
8. child - componentDidMount

It is called before but it is been put into the wait cycle. Because we are using async.

9. child render (state change hone k baad phirse render hoga

* setState trigger next render. It will trigger reconciliation process. So, the child will be rendered once again when we have the data.

This re-render cycle is known as 'UPDATING'.

Date.....

- ComponentDidMount is called after first render.
- ComponentDidUpdate is called after every next render.
- Before the component is unmounted from the DOM, ComponentWillUnmount will be called.

> NB: Never compare React lifecycle method with functional components. In modern React code, they removed the concepts of lifecycle method.

ComponentDidUpdate

→ is called after every subsequent render.
→ In functional component, we use dependency array in useEffect which indicates when should useEffect should be called.

Eg:-
```
useEffect ( () => {
    //API call
}, [count, count2] );
```

This means that whenever the count and count2 gets updated, useEffect gets executed.

Earlier, in class-based components, this is done like ⟶ (next page)

Which is Hectic! 6

Date.....

componentDidUpdate(prevProps, prevState){

if(

this.state.count !== prevState.count ||

this.state.count2 !== prevState.count2)

{

//code

}

}

alag-alag useEffect k liye alag alag if statement.

(Senior developer Banna hai ?)

→ Single Page application mein bhi cons hote hai, agr tune ek page (jaise About/contact/home) kisi pe bhi koi process start kar rakha hai (eg setInterval(()=>{},1000)) toh tu page change bhi kar le ye chalta rahega phirse is page pe aayega toh naya process phir start ho jaayega. So it's imp k tumne jo mess create kiya hai usse saaf karo.
So in class based component we use componentWillUnmount() as ye last mein chalega jab tu page leave kar dega. so yahan pe tune jo mess create kiya (jo process shuroo kiya upar) usse saaf kar do.
In react functional based component we use return() in useEffect().

Spiral

Teacher's Sign .....

Date.....

In functional based component

```
useEffect (() => {

    //API call
const timer = setInterval (() => {
    console.log ("NAMASTE REACT OP");
}, 1000);

return () => {
    clearInterval (timer);
    console.log ("useEffect Return");
};

}, []);
```

ye const me chalega Jab tu page leave kar rha hoga

In class Based component

```
componentDidMount () {
  this.timer = setInterval (() => {
      console.log ("Namaste React O P");
  }, 1000);
}

componentWillUnmount () {
    clearInterval (this.timer);
}
```