**Class Based Components Vs Functional Based Component in React JS 🔥**

Before React 16.8, Class components were the only way to track state and lifecycle on a React component. Function components were considered “state-less”.

With the addition of Hooks, Function components are now almost equivalent to Class components.

**A) How to create a Class Based Component Vs Functional Based Component?**

The most important part of class based component is the render method.

**Sample of Class Component:**

Render method returns JSX

The component has to include the extends React.Component statement, this statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

import React from "react";  
  
class ProfileClass extends React.Component {  
 render() {  
 return <h1> Profile Class Component</h1>  
 }  
}  
export default ProfileClass;

**Sample of Functional Component:**

Function that returns JSX

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

**B) Props in Class Based Components Vs Functional Based Component?**

<Profile name={"Shamin"}/> //functional component  
 <ProfileClass name={"Shamin Class"}/> //class component

**Sample of Class Component:**

For class, use “this”

class ProfileClass extends React.Component { //class component  
 render() {  
 return(  
 <>  
 <h1> Profile Class Component</h1>  
 <h2>Name: {this.props.name}</h2>  
 </>  
 )  
 }  
}  
  
export default ProfileClass;

**Sample of Functional Component:**

const Profile = (props) => { //functional component  
 return (  
 <div>  
 <h2>Profile component </h2>  
 <h3>Name: {props.name}</h3>  
 </div>  
 )  
}  
  
export default Profile

**C) State in Class Based Components Vs Functional Based Component?**

super(props) : to inherit parent class i.e React.Component props

in normal classes super were used to get the properties from parent class

**Sample of Class Component:**

State is an object

import React from "react"; //class component  
  
class ProfileClass extends React.Component {  
  
 constructor(props){  
 super(props);  
 //create state  
 this.state = {  
 count : 0  
 }  
 }  
  
 render() {  
 const {count} = this.state; //destructure  
 return(  
 <>  
 <h1> Profile Class Component</h1>  
 <h2>Name: {this.props.name}</h2>  
 <h3>Count: {this.state.count}</h3>  
 {/\* <h3>Count: {count}</h3> //destructure \*/}  
 </>  
 )  
 }  
}  
  
export default ProfileClass;

**Sample of Functional Component:**

useState hook is used

import { useState } from "react" //functional component  
  
const Profile = (props) => {  
 const [count] = useState (0)  
 return (  
 <div>  
 <h2>Profile component </h2>  
 <h3>Name: {props.name}</h3>  
 <h3>Count: {count}</h3>  
 </div>  
 )  
}  
  
export default Profile

**D) Multiple State in Class Based Components Vs Functional Based Component?**

In Class Comp, multiple state is handled in one single state object

**Sample of Class Component:**

import React from "react"; //class component  
  
class ProfileClass extends React.Component {  
  
 constructor(props){  
 super(props);  
 //create state  
 this.state = {  
 count : 0,  
 count1 : 1  
 }  
 }  
  
 render() {  
 const {count} = this.state; //destructure  
 return(  
 <>  
 <h1> Profile Class Component</h1>  
 <h2>Name: {this.props.name}</h2>  
 <h3>Count: {this.state.count}</h3>  
 {/\* <h3>Count: {count}</h3> //destructure \*/}  
 </>  
 )  
 }  
}  
  
export default ProfileClass;

**Sample of Functional Component:**

import { useState } from "react" //functional component  
  
const Profile = (props) => {  
 const [count] = useState (0)  
 const [count1] = useState (1)  
 return (  
 <div>  
 <h2>Profile component </h2>  
 <h3>Name: {props.name}</h3>  
 <h3>Count: {count}</h3>  
 <h3>Count1: {count1}</h3>  
 </div>  
 )  
}  
  
export default Profile

**E) SetState in Class Component Vs Functional Based Component?**

**Sample of Class Component:**

import React from "react";  
  
class ProfileClass extends React.Component {  
  
 constructor(props){  
 super(props);  
 //create state  
 this.state = {  
 count : 0,  
 count1 : 1  
 }  
 }  
  
 render() {  
 const {count} = this.state; //destructure  
 return(  
 <>  
 <h1> Profile Class Component</h1>  
 <h2>Name: {this.props.name}</h2>  
 <h3>Count: {this.state.count}</h3>  
 {/\* // / WE DO NOT MUTATE STATE DIRECTLY \*/}  
{/\* // I NEver do this. state = something \*/}  
 <button onClick={() => {  
 this.setState({  
 count: 1,  
 })  
 }}  
 > Set COunt</button>  
 </>  
 )  
 }  
}  
  
export default ProfileClass;

**Sample of Functional Component:**

import { useState } from "react"  
  
const Profile = (props) => {  
 const [count, setCount] = useState (0)  
 const [count1] = useState (1)  
 return (  
 <div>  
 <h2>Profile component </h2>  
 <h3>Name: {props.name}</h3>  
 <h3>Count: {count}</h3>  
 <button onClick={() => setCount(1)}>Set Count</button>  
 </div>  
 )  
}  
  
export default Profile

**F) Lifecycle in React Class based Component**

**Sample of Class Component:**

**Class based Component** -> constructor -> render -> component did mount

state in constructor as it is called first and it can set default state

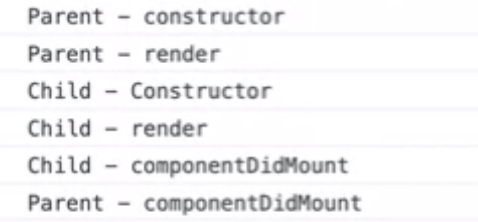
import React from "react";  
  
class ProfileClass extends React.Component {  
  
 constructor(props){  
 super(props);  
 //create state  
 this.state = {  
 count : 0,  
 count1 : 1  
 }  
 console.log('constructor')  
 }  
  
 componentDidMount(){  
 console.log('profile-class')//api call  
 }  
  
  
 render() {  
 console.log('render-class')  
 const {count} = this.state; //destructure  
 return(  
 <>  
 <h1> Profile Class Component</h1>  
 <h2>Name: {this.props.name}</h2>  
 <h3>Count: {this.state.count}</h3>  
 {/\* // / WE DO NOT MUTATE STATE DIRECTLY \*/}  
{/\* // I NEver do this. state = something \*/}  
 <button onClick={() => {  
 this.setState({  
 count: 1,  
 })  
 }}  
 > Set COunt</button>  
 </>  
 )  
 }  
}  
  
export default ProfileClass;

**Sample of Functional Component:**

**Functional Component** -> render -> useEffect

import { useState } from "react"  
import { useEffect } from "react"  
  
const Profile = (props) => {  
 const [count, setCount] = useState (0)  
 const [count1] = useState (1)  
 useEffect(()=>{ // calls after render whatever we can with default state  
 // re renders with use Effect api  
 //calls render then useEffect  
 console.log('Profile')  
 })  
 console.log('render')  
 return (  
 <div>  
 <h2>Profile component </h2>  
 <h3>Name: {props.name}</h3>  
 <h3>Count: {count}</h3>  
 <button onClick={() => setCount(1)}>Set Count</button>  
 </div>  
 )  
}  
  
export default Profile

**G) Parent Child Life Cycle Flow in Class based Component**



**H) Parent with 2 Child Life Cycle Flow in Class based Component**

Complicated lifecycle in class based component

Mounting has two phases:

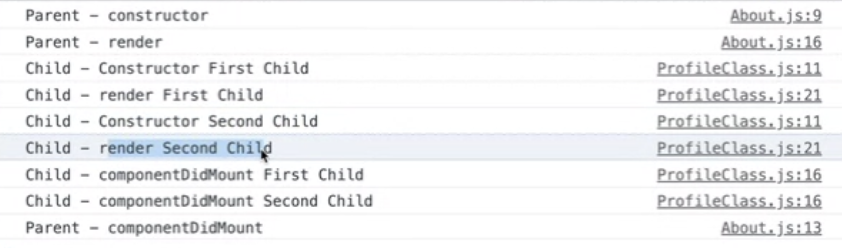
**1.Render Phase** -> constructor and render is called (fast -> forms html)

Pure and has no side effects. May be paused, aborted or restarted by React.

**2.Commit Phase** -> componentDidMount is called; react actually modifies the DOM (slow -> calls api)

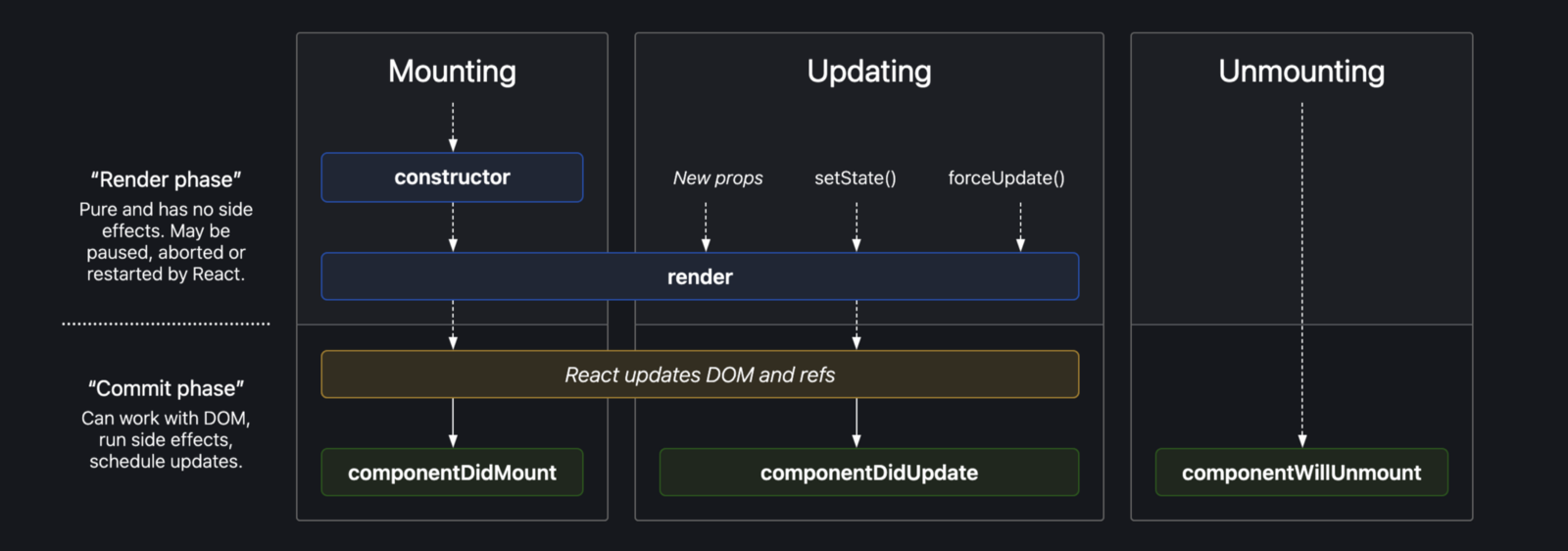
Can work with DOM, run side effects, schedule updates.

Parent Constructor  
Parent Render  
First Child Constructor  
First Child Render  
Second Child Constructor  
Second Child Render  
First Child componentDidMount  
Second Child componentDidMount  
Parent componentDidMount



**I) Life Cycle Hooks Of Class Based Component**

Mounting → Updating -> Unmounting



**ComponentDidMount Vs ComponentDidUpdate Vs ComponentWillUnmount**

ComponentDidMount — calls after first render

ComponentDidUpdate- calls after every other render; No state and props -> it does not go for updating stage

ComponentWillUnmount — calls when you go to another page

**J)** **ComponentDidUpdate Vs useEffect**

Calls once after initial render + every time count changes after every render

**Sample of Class Component:**

ComponentDidUpdate(prevProps,prevState) { //class component  
if(this.state.count != prevState.count){  
//code  
}  
if(this.state.count2 != prevState.count){  
//code  
}  
}

**Sample of Functional Component:**

useEffect{() => { //functional component  
//api call  
},[count]);  
  
useEffect{() => { //functional component  
//api call  
},[count2]);

**K) ComponentWillUnmount Vs return**

React -> SPA

Cons of SPA -> While using setInterval the calls keep on calling until ComponentWillUnmount is called.

ComponentWillUnmount — called when we route to another page. Stops the call.

Use — Clean up, moving away from component, destroy call

**Sample of Class Component:**

componentDidMount(){   
this.timer.setInterval(() = > {  
console.log("number of calls")  
},1000  
}  
  
componentWillUnMount() { //class component -- calls when you leave a page  
clearInterval(this.timer)  
}

**Sample of Functional Component:**

useEffect{() => { //functional component  
//api call  
timer.setInterval(() = > {  
console.log("number of calls")  
},1000)  
  
return () => { //similar to componentWillUnMount -- calls when you leave a page  
console.log("stops")  
clearInterval(this.timer)  
}  
  
},[]);