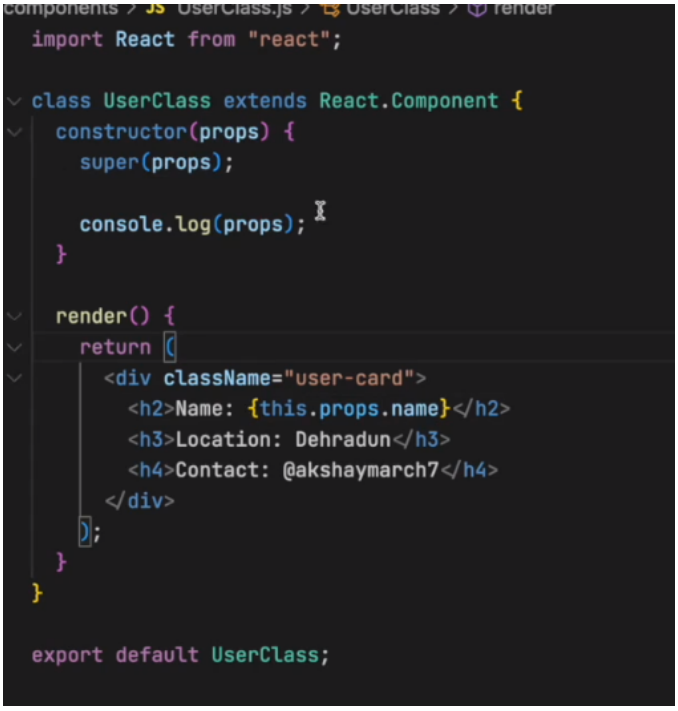
Chapter 08: Lets Get Classy

▼ Class Component

- Class based component is normal Javascript class
- Extending React.component. This will make react know this is class based react component
- `render()` method returns piece of JSX
- constructor will receive the props
    - Add super(props) too



▼ State in class component

- Creating state variable inside class component
    - Created in constructor
    - this.state (state is reserved keyword)

```jsx
import React from "react";

class UserClass extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      count: 0,
    };
  }

  render() {
    const { name, location } = this.props;

    return (
      <div className="user-card">
        <h1>Count : {this.state.count}</h1>
        <h2>Name: {name}</h2>
        <h3>Location: {location}</h3>
        <h4>Contact: @akshaymarch7</h4>
      </div>
    );
  }
}
```

- Creating multiple state variables in class

```jsx
this.state = {
  count: 0,
  count2: 2,
};
```

- update the state variables
  - This is wrong. We should never modify state variable directly. This will create inconsistency. Remember like in functional compoenent, React will hold state value internally and persists it through rerenders. Even though new instance of class is created on rerenders when state changes, class state value which will be uninitialiased will be assigned value by react using its internal variable.

    ❌

```
return (
  <div className="user-card">
    <h1>Count : {count}</h1>
    <button
      onClick={() => {
        this.state.count = this.state.count + 1;
      }}
    >
```

- React gives us `this.setState()` ✅

```
<button
  onClick={() => {
    // NEVER UPDATE STATE VARIABLES DIRECTLY
    this.setState({
      count: this.state.count + 1,
    });
  }}
>
```

▼ componentDidMount

- get called once after first time render() , after component is completely mounted in web page

```
import React from "react";

class UserClass extends React.Component {
  constructor(props) {…
  }

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

  render() {…
  }
}

export default UserClass;
```

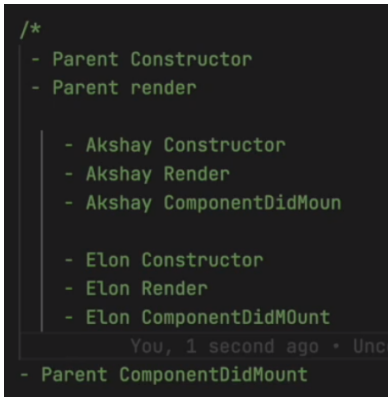
- Used to make api calls. why?
  - like `useEffect(),` dont want wait for api to render component.
  - So render compoenent → Make api call → Fill data to component using rerender
  - React does it fast using reconcialliation , vdom and diff algoritm.

▼ Order of loading when Parent and child are both class components

- Parent constructor()
- Parent render()
- Child constructor()
- Child render()
- Child componentDidMount()
- Parent componentDidMount()

What happens when multiple childs?

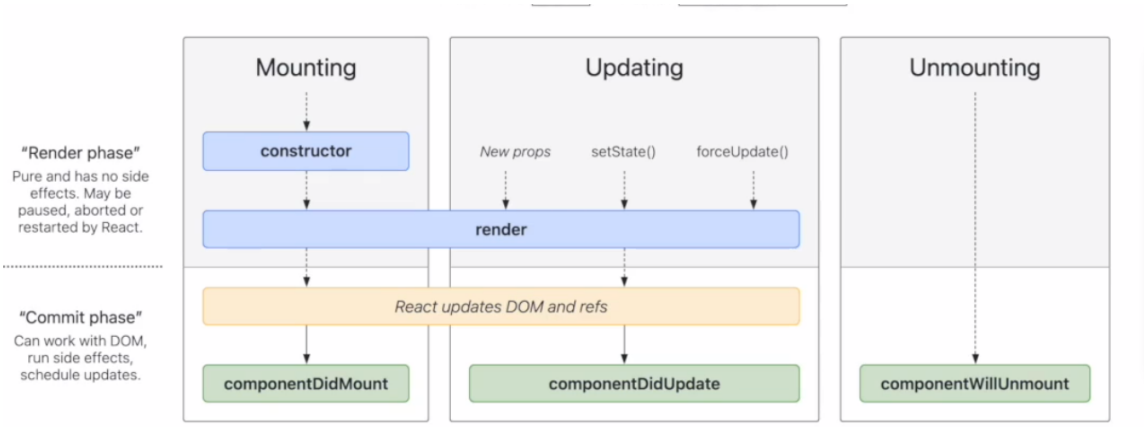
- This is wrong ❌



- This is Right ✅

```
Parent Constructor
Parent Render
Akshay Saini (classs)Child Constructor
Akshay Saini (classs)Child Render
Elon MuskChild Constructor
Elon MuskChild Render
Akshay Saini (classs)Child Component Did Mount
Elon MuskChild Component Did Mount
Parent Component Did Mount
```

- Why so? → React does some optimisation
- Refer react-lifecycle- methods diagram



- See Mounting column.
  Render Phase of two childs are batched together. But Why?
  - DOM manipulation is too much expensive process. React don't want to do commit phase in loop for every child.

▼ componentDidUpdate()

- This runs after every render

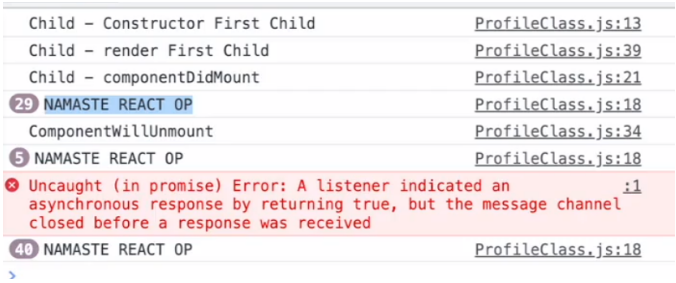
▼ This is how whole lifecycle methods work

```
* --- MOUNTING ----
*
* Constructor (dummy)
* Render (dummy)
*      <HTML Dummy >
* Component Did MOunt
*      <API Call>
*      <this.setState> → State variable is updated
*
* ---- UPDATE
*
*      render(APi data)
*      <HTML (new API data>)
*      ccomponentDid Update
```

▼ componentWillUnmount()

- gets called when component is removed from the page
- like when we go to new page
- When to use this?
    - There are lot of things we need to clear when we change the page within SPA.
      Note: In SPA actually there is only one page, just components are getting changed (mounted and unmounted). But for user we are showing like page is changed (as url change).
    - There is one disdvantage with SPA. Take this example. Even when we change page inside SPA, this will keep getting called, as page is not reloaded and window variable is not reset.

```
componentDidMount() {
  setInterval(() => {
    console.log("NAMASTE REACT OP ");
  }, 1000);
```



- This might get called twice too if we renrender compoent where this setInverval is set
- This will blow up our app after a while. For a scalable app, we need to take care of every line of code we write. Senior developer think like this
- clear interval on unmount

```
componentDidMount() {
  this.timer = setInterval(() => {
    console.log("NAMASTE REACT OP ");
  }, 1000);

  console.log("Child - componentDidMount");
}

componentDidUpdate(prevProps, prevState) {…
}
componentWillUnmount() {
  clearInterval(this.timer);
  console.log("ComponentWillUnmount");
}
```

- What happens if we do this in functional component

```
useEffect(() => {
  // API Call
  setInterval(() => {
    console.log("NAMASTE REACT OP ");
  }, 1000);
}, []);
```

What happens if we change page in SPA, will it stop?

- No, it will not stop . We need to clean up for functional compoent too.
- We can return a function in useEffect() which will be called on unmount

```
useEffect(() => {
  // API Call
  // setInterval(() => {
  //   console.log("NAMASTE REACT OP ");
  // }, 1000);
  console.log("useEffect");

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

- When we change page, “UseEffect Return” will be printed

▼ Some other knowledge

- How to do this in class based component where useEffect run only when particular state change

```
useEffect(() => {
  // API Call
  //console.log("useEffect");
}, [count]);
```

- Like this using prevProps, prevState.
- Mostly internally when updating state, react stores old state in these variables. So that they are persisted too on re renders

```
componentDidUpdate(prevProps, prevState) {
  if(this.state.count !== prevState.count) {

  }
  console.log("Component Did Update");
}
```

- What if I want conditions for multiple states like this

```
useEffect(() => {
  // API Call
  //console.log("useEffect");
}, [count]);

useEffect(() => {
  // API Call
  //console.log("useEffect");
}, [count2]);
```

```
componentDidUpdate(prevProps, prevState) {
  if (this.state.count !== prevState.count) {
    // code
  }
  if (this.state.count2 !== prevState.count2) {
    // code
  }
  console.log("Component Did Update");
}
```

- Never compare react lifecycle methods to functional component- like componentDidMount is same as useEffect() etc. This is wrong and they are not same

▼ Homework:

why cant we make callback is useEffect a async, while we can make componentDidUpdate async

why super(props)