Working with Component Life Cycle Methods

- Each component in React has several "**life-cycle methods**" which are useful to run different pieces of code at particular times in the component processing.

Eg:

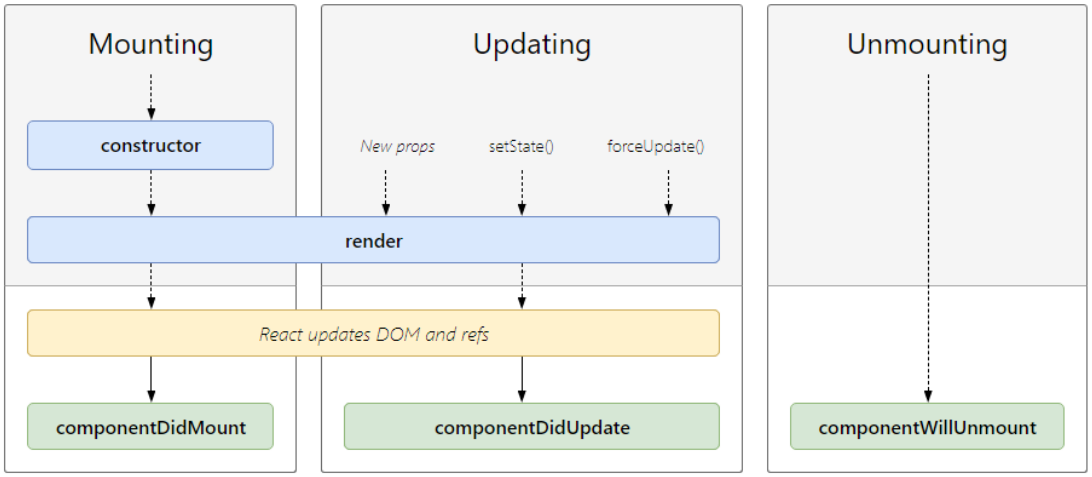
```
import React from 'react'
import ReactDOM from 'react-dom'

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

    render() {
        return (
            <div>
                <h2>Employee Component</h2>
            </div>
        );
    }
}
const element = <EmployeeComponent></EmployeeComponent>
ReactDOM.render(element, document.getElementById("root"));
```

Component Life Cycle Phases:

1. Mounting
2. Updating
3. Unmounting



Mounting Phase:

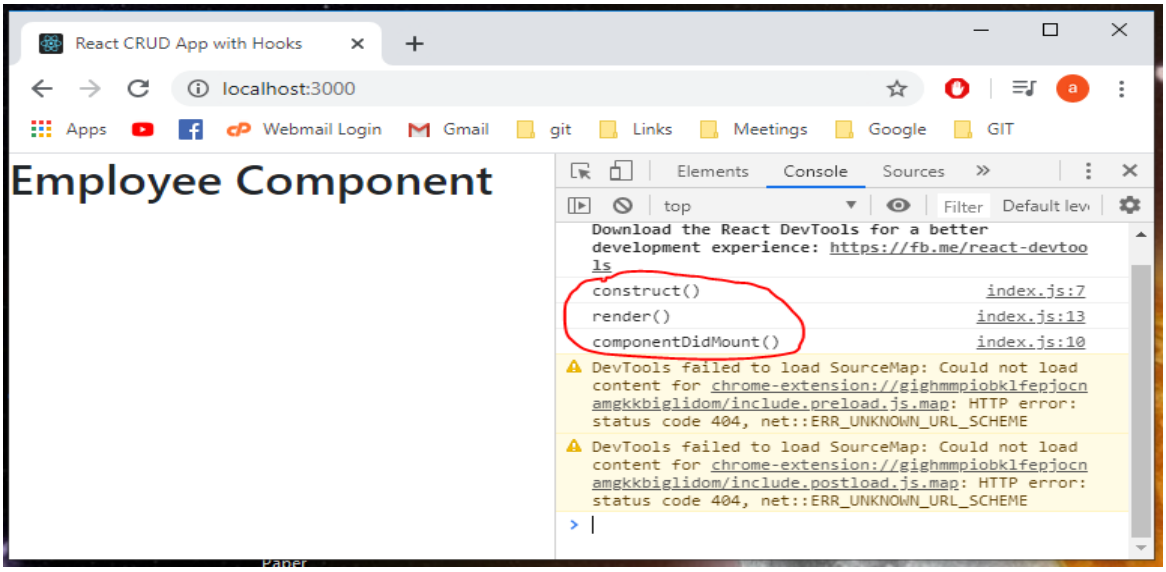
- The process of inserting a component (EmployeeComponent) into the DOM Tree is called as **Mounting**.

- These methods are called in the following order when an instance of a component is being created and inserted into the DOM:
  1. **constructor()**
  2. **render()**
  3. **componentDidMount()**

Eg:

```
import React from 'react'
import ReactDOM from 'react-dom'

class EmployeeComponent extends React.Component {
    constructor(props) {
        super(props);
        console.log("construct()")
    }
    componentDidMount(){
        console.log("componentDidMount()")
    }
    render() {
        console.log("render()");
        return (
            <div>
                <h2>Employee Component</h2>
            </div>
        );
    }
}
const element = <EmployeeComponent></EmployeeComponent>
ReactDOM.render(element, document.getElementById("root"));
```



constructor():

- The constructor of a React component is called before it is mounted.
- When implementing the constructor for a React.Component subclass, we should call **super(props)** before any other statement. Otherwise, this.props will be undefined in the constructor, which can lead to bugs.

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

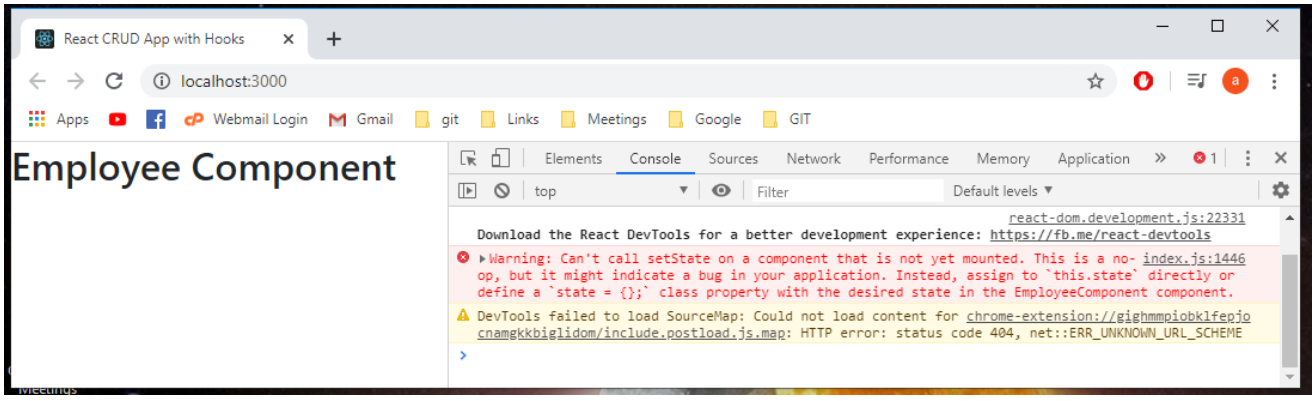
- Typically, in React constructors are only used for two purposes:
  1. Initializing local state by assigning an object to **this.state**.
  2. Binding event handler methods.

Note: Remember that we should not call **setState()** in the constructor.

```
import React from 'react'
import ReactDOM from 'react-dom'

class EmployeeComponent extends React.Component {
    constructor(props) {
        super(props);
        this.setState({name:'Sai'});
    }

    render() {
        return (
            <div>
                <h2>Employee Component</h2>
            </div>
        );
    }
}
const element = <EmployeeComponent></EmployeeComponent>
ReactDOM.render(element, document.getElementById("root"));
```



Conclusion:

1. Constructor is the only place where we should assign **this.state** directly.
2. In all other methods, we need to use **this.setState()** instead.

render():

- The render() method is the only required method in a class component.
- Output of a Component is dependent on what we return from this method.
- When called, it should examine **this.props** and **this.state** and return one of the following types:
    1. React elements
    2. Arrays and fragments.
    3. Portals
    4. String and numbers.
    5. Booleans or null.

Note:

The render() function should be pure, meaning that it does not modify component state, it returns the same result each time it's invoked, and it does not directly interact with the browser.

componentDidMount():

- This is invoked immediately after a component is mounted (inserted into the tree).
- It is best place if we need to load data from a remote endpoint (REST call) and to instantiate the network request.
  Eg: We have used it in our previous sessions.

Updating Phase:

- An update can be caused by changes to **props or state**.
- These methods are called in the following order when a component is being re-rendered:
    1. render()
    2. componentDidUpdate()

componentDidUpdate():

- This is invoked immediately after updating occurs.
- This method is not called for the initial render.
- We can use this method to operate on the DOM when the component has been updated.

Unmounting Phase:

- When we develop one application, we develop multiple components as part of it.
- User will be navigating from one component to another component. Just like going from Home to AboutUs and then navigating to Contact Us tabs.
- When user navigate from one Component to the Other Component, the previous component will be removed from the DOM and the new Component contents will be displayed in the UI.
- For this purpose, we should go for **Unmounting**.
- **Unmouting** is nothing but removing the component from DOM tree.
- If we want to perform any necessary cleanup activities, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in **componentDidMount()** then it should performed in **componentWillUnmount()** method.
- **componentWillUnmount()** is invoked immediately before a component is unmounted and destroyed.

Eg:

```
import React from 'react'
import ReactDOM from 'react-dom'

const employeeList = [
    { id: 1, name: 'Scott', salary: 4400 },
    { id: 2, name: 'Allen', salary: 7500 },
    { id: 3, name: 'Jones', salary: 9200 },
    { id: 4, name: 'James', salary: 9200 },
    { id: 5, name: 'James', salary: 8400 },
];

class EmployeeComponent extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            employees: []
        };
    }
    componentDidMount(){
        this.getEmployees();
    }
    getEmployees(){
        this.setState({
            employees: employeeList
        });
    }
    render() {
        return (
            <div>
                <h2>Employees Details</h2>
                <table>
                    <thead>
                        <tr>
                            <th>Id</th>
                            <th>Name</th>
                            <th>Salary</th>
                        </tr>
                    </thead>
                    <tbody>
                        {this.state.employees.map(emp => (
                            <tr key={emp.id}>
                                <td>{emp.id}</td>
                                <td>{emp.name}</td>
                                <td>{emp.salary}</td>
                            </tr>
                        ))}
                    </tbody>
                </table>
            </div>
        );
```

```
    }
}

const element = < EmployeeComponent  />
ReactDOM.render(element, document.getElementById("root"));
```