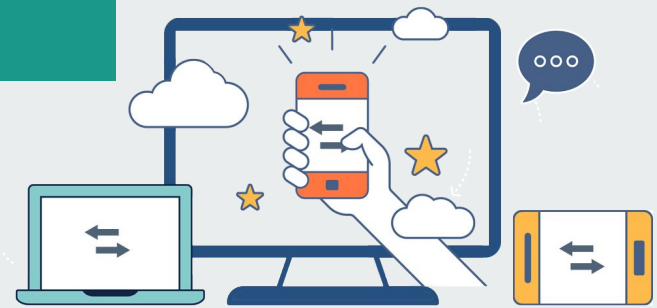
-Lecture 8-

Chapter 5– **React lifecycle methods**



Adil **CHEKATI**, PhD

adil.chekati@univ-constantine2.dz

Prerequisites

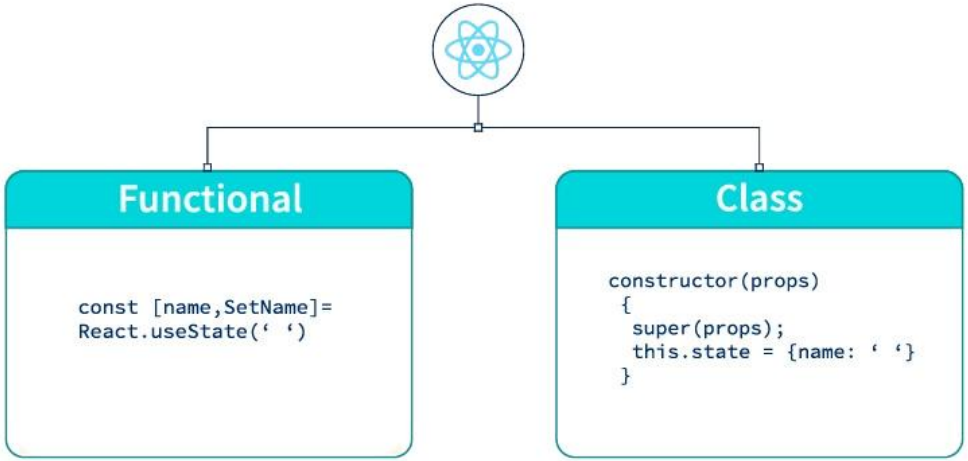
- ❏ Foundational React Knowledge (Knowledge and Application)
- ❏ Experience with Component Rendering (Application)

React lifecycle methods

Objectives

- ➔ Describe the lifecycle of a React component, including mounting, updating, and unmounting phases.
- ➔ Implement lifecycle methods.
- ➔ Understand when to use functional components and React hooks for state management and side effects.

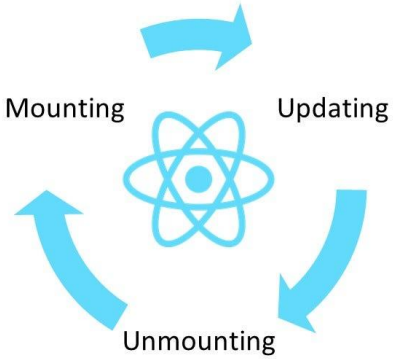
Introduction



*Before version 16.8.0, React components were **classes**.*
*From version 16.8.0, components became **functions** (the latest is version 18.2.0 of June 2022).*

Introduction

- One of the key features of React is its lifecycle methods.
- Lifecycle methods allow developers to control the behavior of components at various stages of their lifecycle.
- Understanding and utilizing these lifecycle methods is crucial for writing efficient and functional React components.



Introduction

The following section about lifecycle methods concerns only **Class Component .**

1. What are React lifecycle methods?

- ❏ React components go through different phases during their lifecycle, starting from initialization, rendering, updating, and finally unmounting.
- ❏ React provides a set of lifecycle methods that allow developers to execute code at specific points in a component's lifecycle.
- ❏ These methods are predefined functions that are invoked automatically by React, giving developers the opportunity to perform certain actions at the right time.

*Overriding these methods allows you to run your code at **particular times** during the process. So React offers and supports methods for each phase in the component lifecycle.*

1. What are React lifecycle methods?

1.1. Types of lifecycle methods

a) Mounting methods

These methods are executed when an instance of a component is being created and inserted into the DOM. They include **`constructor()`**, **`render()`**, **`componentDidMount()`**, and **`getDerivedStateFromProps()`**.

b) Updating methods

These methods are called when a component is being re-rendered due to changes in props or state. They include **`render()`**, **`componentDidUpdate()`**, **`shouldComponentUpdate()`**, **`getSnapshotBeforeUpdate()`**, and **`componentDidUpdate()`**.

c) Unmounting methods

These methods are invoked when a component is being removed from the DOM. They include **`componentWillUnmount()`**.

1. What are React lifecycle methods?

1.1. Types of lifecycle methods

a) Mounting methods

b) Updating methods

c) Unmounting methods

*Each of these methods serves a **specific purpose** and can be used to perform different tasks in different lifecycle stages.*

1. What are React lifecycle methods?

1.2. Common use cases for React lifecycle methods

➔ React lifecycle methods offer several use cases for developers to enhance the functionality and optimize the performance of their components.

Some common scenarios where these methods are widely used include:

- Initializing and setting up component state or variables in the **`constructor()`**.
- Fetching data from an API or performing asynchronous operations in the **`componentDidMount()`**.
- Implementing logic to determine whether a component should update or not using **`shouldComponentUpdate()`**.
- Saving or restoring the scroll position of a component before an update using **`getSnapshotBeforeUpdate()`**.
- Cleaning up resources, subscriptions, or event listeners in the **`componentWillUnmount()`**.

➔ By leveraging these lifecycle methods, developers can control and manipulate the behavior of their React components, ensuring optimal performance and user experience.

1. What are React lifecycle methods?

1.2. New lifecycle methods in React 16.3 and beyond

React 16.3 introduced several new lifecycle methods to provide more granular control over the component lifecycle. These methods include:

- `static getDerivedStateFromProps():` This method allows a component to update its internal state based on changes in props before rendering it.
  It replaces the legacy `componentWillReceiveProps()` method.

- `getSnapshotBeforeUpdate():` This method is used to capture information about the component's state before it gets updated.
  It returns a value that will be passed to `componentDidUpdate()`. It is often used when dealing with scrolling or animations

1. What are React lifecycle methods?

1.2. Choosing the Right Lifecycle Method

➔ When working with React, it's important to select the appropriate lifecycle method for the specific needs of your application.

➔ While the basic lifecycle methods cover the most common scenarios, the advanced ones provide additional control and customization possibilities:

- By properly implementing **`shouldComponentUpdate()`**, you can optimize rendering performance and prevent unnecessary re-renders.
- Use **`getSnapshotBeforeUpdate()`** when you need to capture and store the current state before making any changes.
- Lastly, **`componentDidCatch()`** allows you to gracefully handle errors and prevent a component tree from crashing due to unhandled exceptions.

*Understanding and properly utilizing these advanced lifecycle methods will contribute to creating **more efficient** and **error-tolerant** React applications.*

2. Phase#1: Component Mounting

- Mounting in React means or represents the process of converting the virtual components into actual DOM elements.

- React supports 4 mounting lifecycle methods for component classes: **`constructor()`**, which is called **first**.
  then **`static getDerivedStateFromProps()`** which is rarely used
  followed by **`render()`**,
  and finally **`componentDidMount()`.**

2. Phase#1: Component Mounting

1.1. constructor()

- Called when a component is being initialized. It is used to initialize state, bind event handlers, and perform other one-time setup tasks.
- It is the first method that gets called before the other mounting methods and takes props as an argument.

2. Phase#1: Component Mounting

1.1. constructor()

```
01  // constructor() method example
02
03  class ComponentExample extends React.Component {
04
05  constructor(props) {
06    super(props);
07    this.state = { showForm : false };
08    this.handleClick = this.handleClick.bind(this);
09  }
10
11   handleClick=()=>{
12      this.setState({showForm : true});
13  }
14
15  }
```

2. Phase#1: Component Mounting

1.2. static getDerivedStateFromProps()

- Rarely used.
- The React team introduced this method as an alternative to the `componentWillReceiveProps()`.
- Invoked right before rendering and is used to update component state based on changes in props. It should return an object to update the state or null to indicate no changes are necessary.
- This lifecycle method is used when the component state depends on changes in its props, such as in transition components.

2. Phase#1: Component Mounting

1.2. static getDerivedStateFromProps()

```
01  // static getDerivedStateFromProps() method example
02
03  class ComponentExample extends React.Component {
04
05  constructor(props) {
06    super(props);
07    this.state = { showForm : false };
08
09  }
10
11
12
13  static getDerivedStateFromProps(props, state) {
14      if (props.showForm === true) {
15  //props.showForm was defined by by the component caller of this component.
16        return { showForm : props.ShowForm};
17      } else {
18      return null;
19    }
20  }
21
22  }
```

2. Phase#1: Component Mounting

1.3. render()

- This method is responsible for returning the JSX that represents the component's UI.
- It should be a pure function that does not modify component state or interact with the DOM (i.e we cannot change the component state using setState in it.)

2. Phase#1: Component Mounting

1.3. render()

```
// render() method example

class ComponentExample extends React.Component {

constructor(props) {
  super(props);
  this.state = { showForm : false };
  this.handleClick = this.handleClick.bind(this);
}

 handleClick=()=>{
    this.setState({showForm : true});
}



render(){
      return (
        <div>
             <div >
              <h3> Click here to see the Form </h3>
              <Button onClick = {this.handleClick}>ADD NEW</Button>
            </div>
            <div >
              {this.state.showForm && <FormComponent />}
            </div>
      </div>
    );
   }
```

2. Phase#1: Component Mounting

1.4. componentDidMount()

- Most used lifecycle method.
- Called after the component has been mounted and inserted into the DOM.
- It is commonly used to perform side effects like fetching data from an API or setting up event listeners.

2. Phase#1: Component Mounting

1.4. componentDidMount()

```
01  // componentDidMount() method example
02
03  class ComponentExample extends React.Component {
04
05  constructor(props) {
06    super(props);
07    this.state = {
08         showForm : false,
09         elements : []
10   };
11
12  }
13
14  componentDidMount(){
15
16        Axios.get('/api/elements')
17       .then(res => {
18           if(res.data.elements){
19               this.setState({elements: res.data.elements});
20
21           } else{
22           alert(res.data.message);
23           }
24       });
25
26     setTimeout(() => {
27        this.setState({showForm : true})
28      }, 2000);
29
30      }
31
32  }
```

3. Phase#2: Component Updating

- Component updating occurs when a component's props or state changes.
- React automatically re-renders the component and applies the necessary updates.
- The following methods are invoked during the updating phase:
  - **`render()`:** Same as in the mounting phase, this method is responsible for returning the JSX representation of the component.
  - **`static getDerivedStateFromProps(props, state)`:** Just like in the mounting phase, this method is called before rendering when props have changed. It allows you to update the state based on the new props.
  - **`componentDidUpdate()`:** more details in the following slide.

3. Phase#2: Component Updating

3.1. componentDidUpdate()

- Called after the component has been updated.
- It is typically used for side effects that depend on the updated state or props, such as updating the DOM or making additional API calls.

3. Phase#2: Component Updating

3.1. componentDidUpdate()

```
01  // componentDidUpdate() method example
02
03  class ComponentExample extends React.Component {
04
05  constructor(props) {
06    super(props);
07    this.state = {
08         userInfo : {}
09   };
10
11  }
12
13  componentDidUpdate(prevState){
14    // In case user informations have not changed, we won't make an api call.
15    if (this.state.userInfo !== prevState.userInfo) {
16        Axios.get('/api/users/${id}')
17       .then(res => {
18           if(res.data.user){
19               this.setState({userInfo: res.data.user});
20
21           } else{
22           alert(res.data.message);
23           }
24       });
25
26      }
27  }
```

4. Phase#3: Component Unmounting

- Called right before the component is removed from the DOM.
- Component unmounting refers to the process of removing a component from the DOM.
- This happens when a component is no longer needed or when the parent component gets updated.
- The following method is called during the unmounting phase:

- **`componentWillUnmount()`:** This method is invoked immediately before a component is unmounted and destroyed.
  It is commonly used for cleanup tasks like removing event listeners or cancelling timers created in **`componentDidMount()`**.

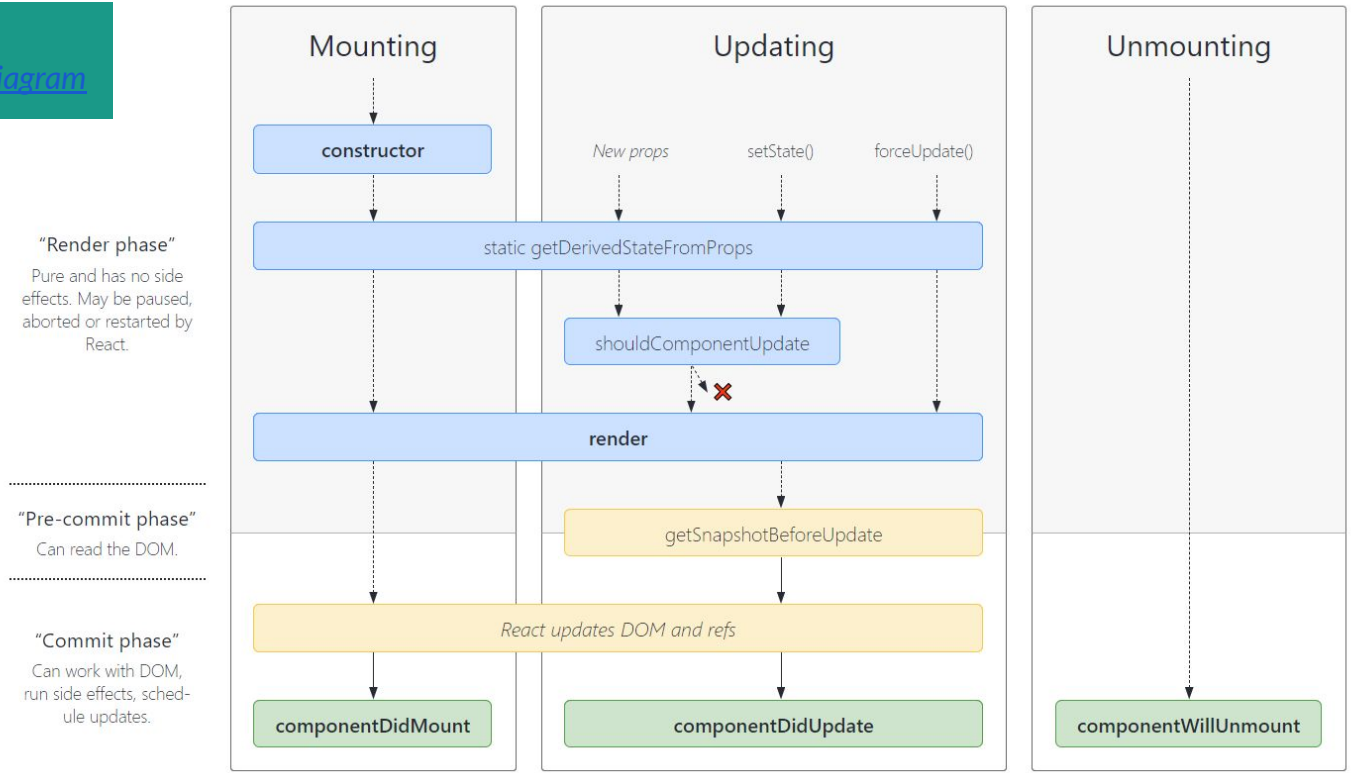
4. Phase#3: Component Unmounting

4.1. componentWillUnmount()

```
01  // WelcomeComponent is the child component.
02
03  class WelcomeComponent extends React.Component {
04    componentWillUnmount() {
05      alert("The welcome message is about to be removed.");
06      // here we can cancel all the api calls made by this component
07
08    }
09    render() {
10      return (
11        <h1>WELCOME !!</h1>
12      );
13    }
14  }
```

5. React Lifecycle Methods Diagram

More details:
lifecycle-methods-diagram



6. Functional Component Lifecycle

- Functional Components, introduced with React Hooks, also have their lifecycle management using **Hooks**.

- How these hooks correspond to the lifecycle phases.:

  - **`useState`** *-> Constructor( )*
    Used to initialize and manage state in Functional Components.

  - **`useEffect`** *-> componentDidMount( )*
    Triggered automatically after a component is successfully mounted and rendered for the first time. It handles component lifecycle events, including rendering (updating) and cleanup (unmounting).

6. Functional Component Lifecycle

```
1  import React, {useState, useEffect} from 'react'
2  import axios from 'axios'
3
4  function List() {
5    const [data, loadData] = useState(null)
6    useEffect(async () => {
7      let listItems = await axios(
8        'http://imaginaryurl.com/api/listItems'
9      )
10     loadData(listItems.data)
11   }, [])
12
13   let listItems
14   if (data) {
15     listItems = data.map((item, i) => {
16       <p key={i}>{item.name}</p>
17     }
18   } else {
19    listItems = <p>Loading...</p>
20   }
21   return {listItems}
22 }
```

Recap

- ➔ *React component lifecycle has three categories – Mounting, Updating and Unmounting.*
- ➔ *The **render()** is the most used lifecycle method.*
  - ◆ *It is a pure function.*
  - ◆ *You cannot set state in render()*
- ➔ *The **componentDidMount()** happens as soon as your component is mounted.*
  - ◆ *You can set state here but with caution.*
- ➔ *The **componentDidUpdate()** happens as soon as the updating happens.*
  - ◆ *You can set state here but with caution.*

Recap

- ➔ *The **componentWillUnmount()** happens just before the component unmounts and is destroyed.*
  - ◆ *This is a good place to cleanup all the data.*
  - ◆ *You cannot set state here.*
- ➔ *The shouldComponentUpdate() can be used rarely.*
  - ◆ *It can be called if you need to tell React not to re-render for a certain state or prop change.*
  - ◆ *This needs to be used with caution only for certain performance optimizations.*
- ➔ *The two new lifecycle methods are **getDerivedStateFromProps()** and **getSnapshotBeforeUpdate()**.*
  - ◆ *They need to be used only occasionally.*
  - ◆ *Not many examples are out there for these two methods and they are still being discussed and will have more references in the future.*

Textbook

➔ All academic materials will be available on:

Google Drive.
E-learning platform of Constantine 2 University.
Google Classroom.

aoa5lne

Google Classroom

References

➔ **Book:**
Stoyan Stefanov -*React Up and Running: Building Web Applications - 2nd édition (2021).*

➔ **Online Resource:**
React Lifecycle Methods Diagram
(https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/)

Questions,
& comments...

adil.chekati@univ-constatine2.dz