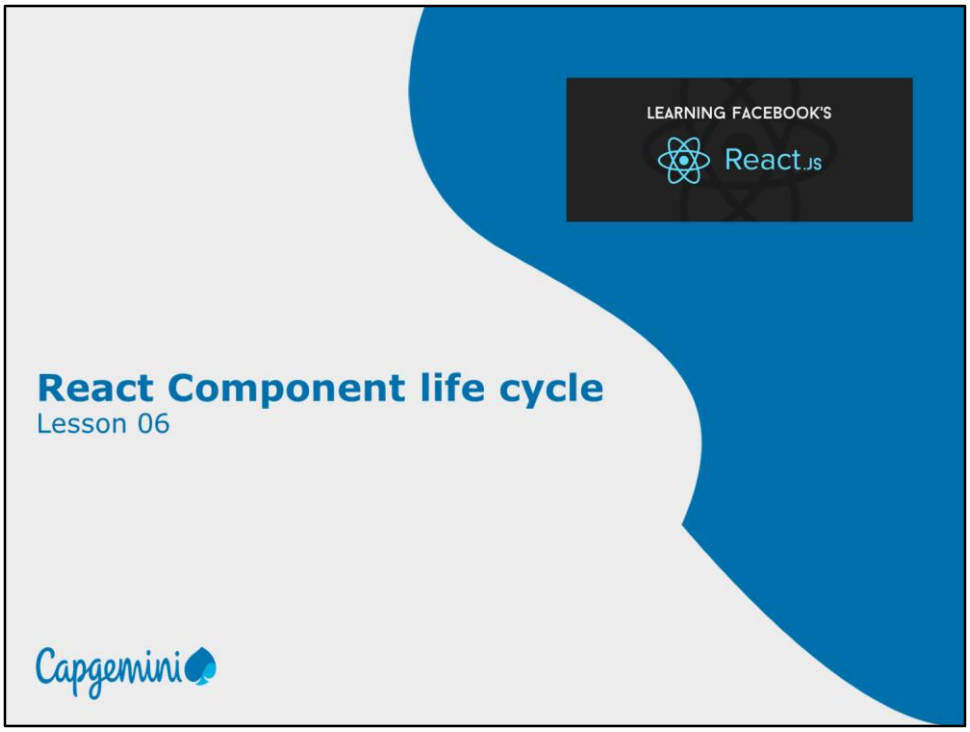
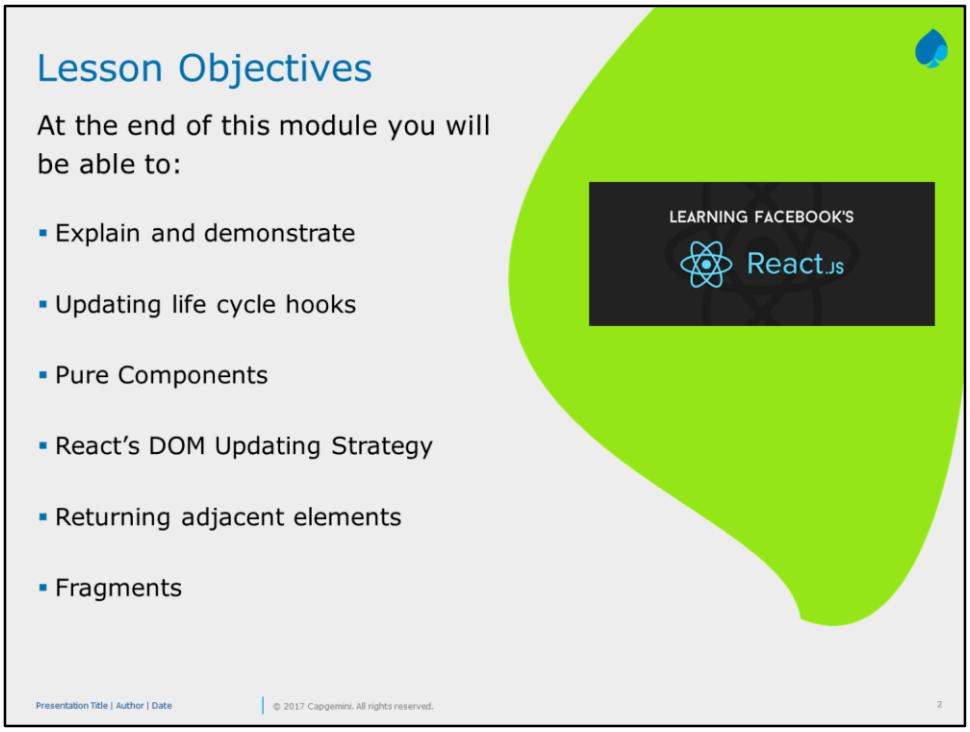
Instructor Notes:

Add instructor notes here.

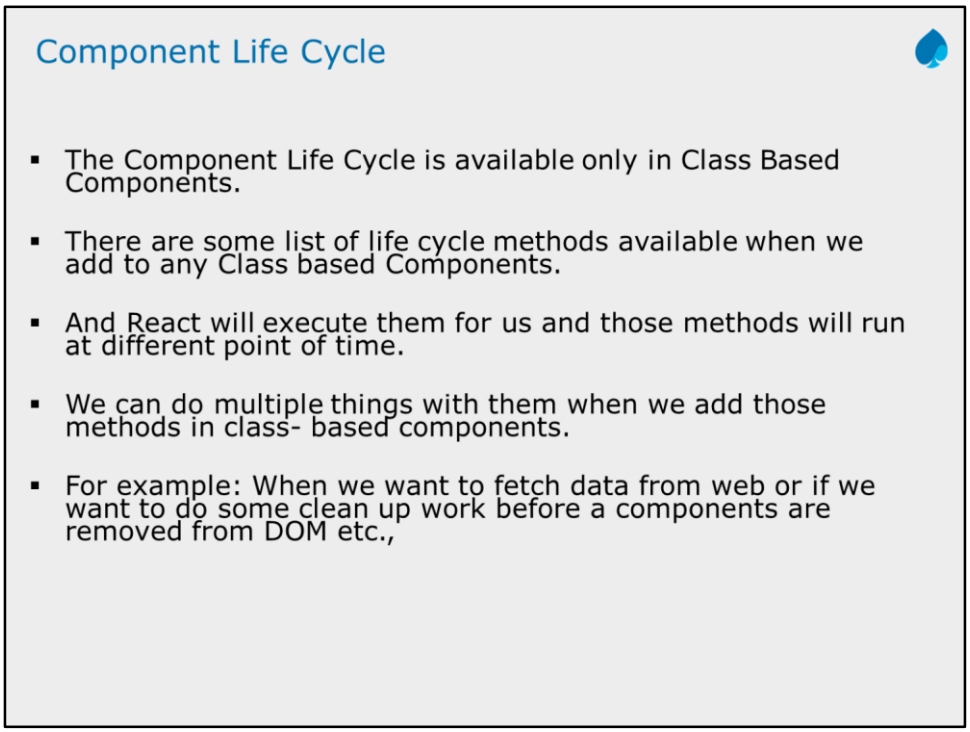


Instructor Notes:

Add instructor notes here.



Instructor Notes:



- The Component Life Cycle is available only in Class Based Components.
- There are some list of life cycle methods available when we add to any Class based Components.
- And React will execute them for us and those methods will run at different point of time.
- We can do multiple things with them when we add those methods in class- based components.
- For example: When we want to fetch data from web or if we want to do some clean up work before a components are removed from DOM etc.,

Instructor Notes:

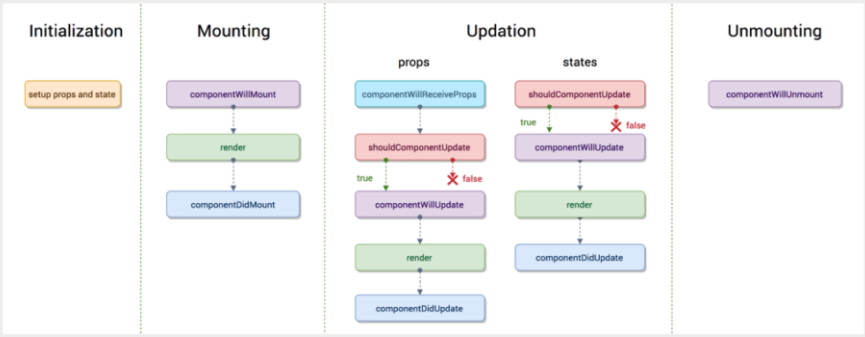
Add instructor notes here.

React Component - Life Cycle Phases

Changes undergone by a component from its creation till its termination.

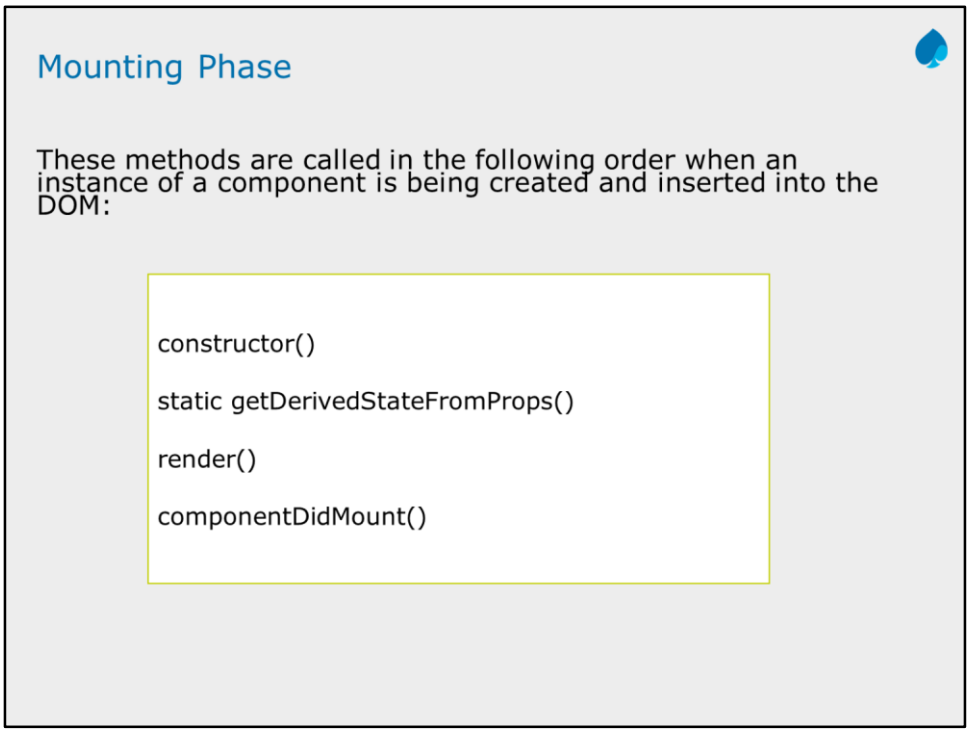
There are main three phases of a React component

- ❑ Mounting
- ❑ Update
- ❑ Unmounting



The invocation of getDefaultProps actually takes place once before any instance of the component is created and the return value is shared among all instances of the component

Instructor Notes:



Mounting is the phase in which our React component mounts on the DOM (i.e., is created and inserted into the DOM).

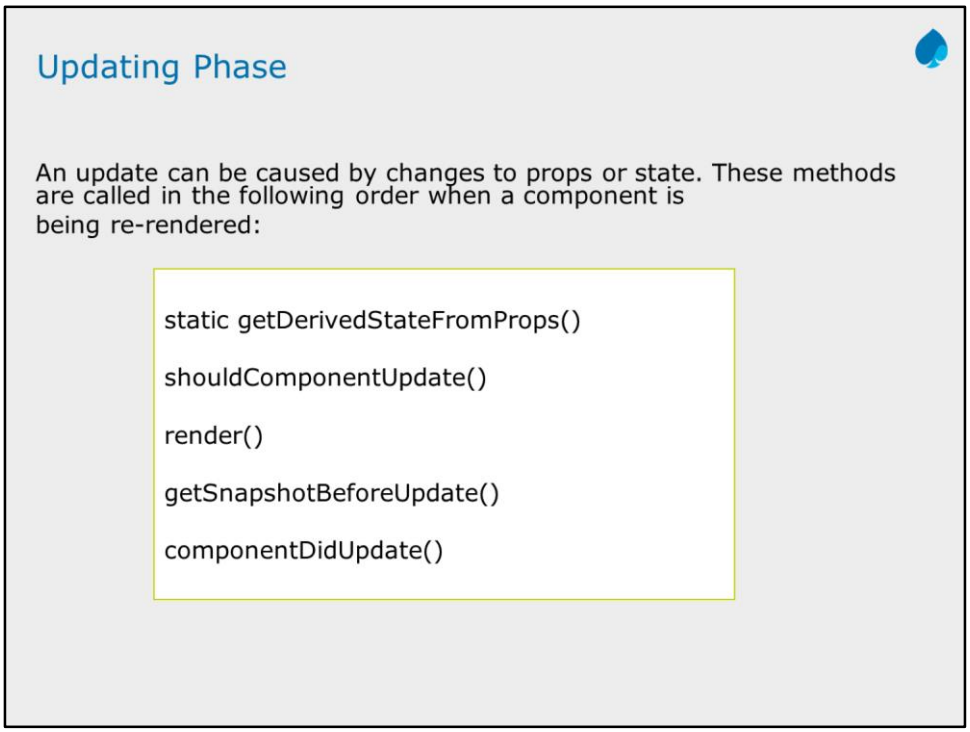
This method is called just before a component mounts on the DOM or the render method is called.

After this method, the component gets mounted.

This phase comes onto the scene after the initialization phase is completed.

In this phase, our component renders the first time

Instructor Notes:



This is the third phase through which our component passes.

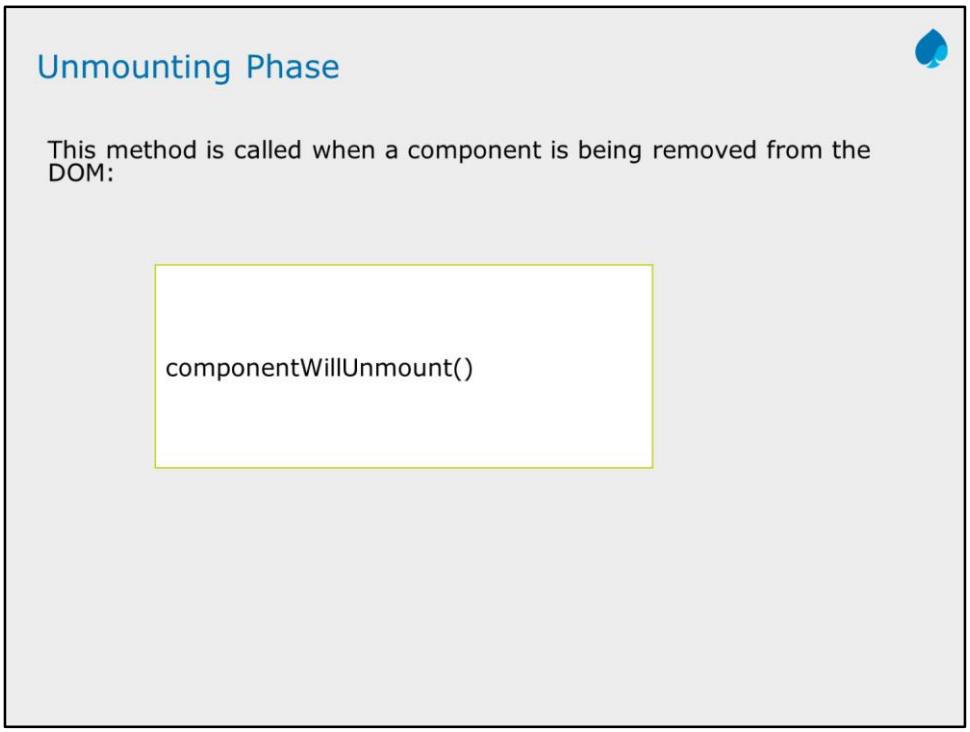
After the mounting phase where the component has been created, the update phase comes into the scene.

This is where component's state changes and hence, re-rendering takes place.

In this phase, the data of the component (state & props) updates in response to user events like clicking, typing and so on.

This results in the re-rendering of the component.

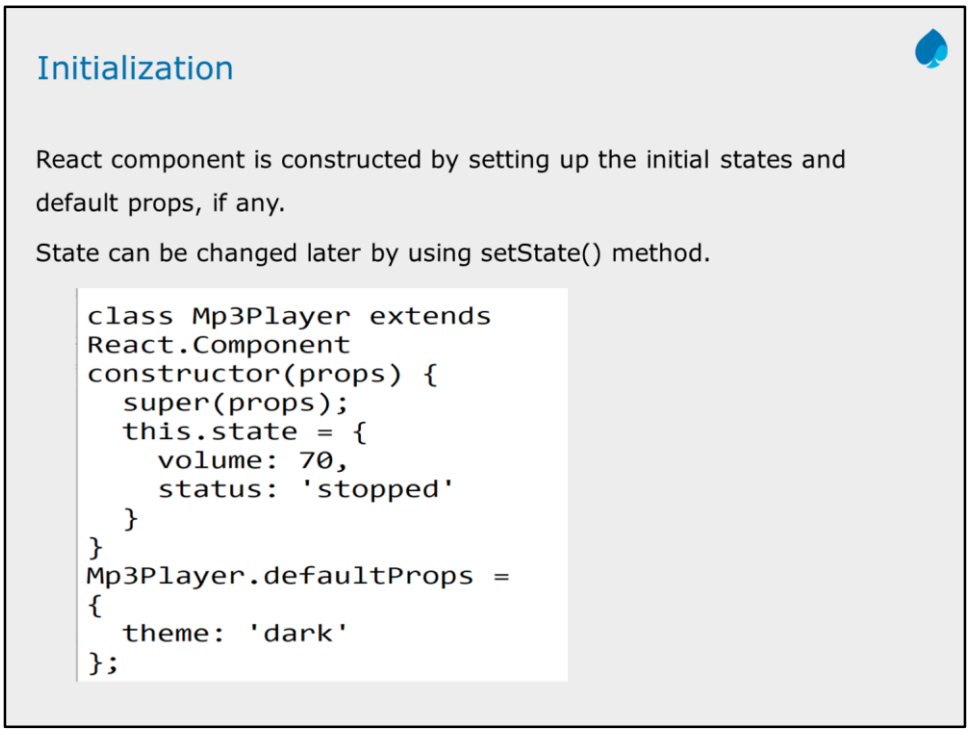
Instructor Notes:



This is the last phase in the component's lifecycle.

As the name clearly suggests, the component gets unmounted from the DOM in this phase.

Instructor Notes:



This is the phase in which the component is going to start its journey by setting up the state and the props.

This is usually done inside the constructor method

Till now we have seen is “what to render” and uses the *render()* method for this purpose.

However this method *render()* method alone is not enough for our requirements.

Take in case if we want to do something before or after the component has rendered or mounted ?

What if we want to avoid a re-render?

Basically all the React component’s lifecyle methods can be split in four phases: initialization, mounting, updating and unmounting.

Below is used for creating a class by React’s createClass.

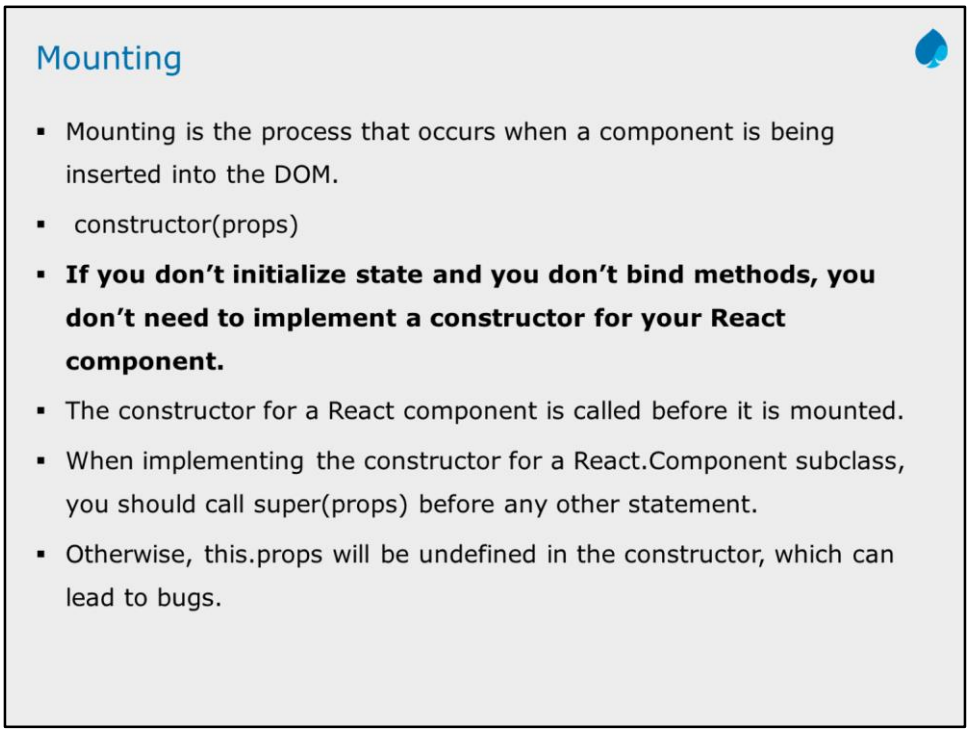
Initialization
The initialization phase is where we define defaults and initial values for *this.props* and *this.state* by implementing *getDefaultProps()* and *getInitialState()* respectively.

The *getDefaultProps()* method is called once and cached—shared across instances.

The *getInitialState()* method is also invoked once, right before the mounting phase. The return value of this method will be used as initial value of *this.state* and should be an object.

```
var Component = React.createClass({
    getDefaultProps: function() {
        console.log('getDefaultProps');
        return {
            title: "Basic counter!!!",
            step: 1
        }
    },
    getInitialState: function() {
        console.log('getInitialState');
        return {
            count: (this.props.initialCount || 0)
        };
    },
}
```

Instructor Notes:



ComponentWillMount:

This method is used for initializing the states or props

Note : It's very important to note that **calling *this.setState()* within this method(componentWillMount) will not trigger a re-render**.

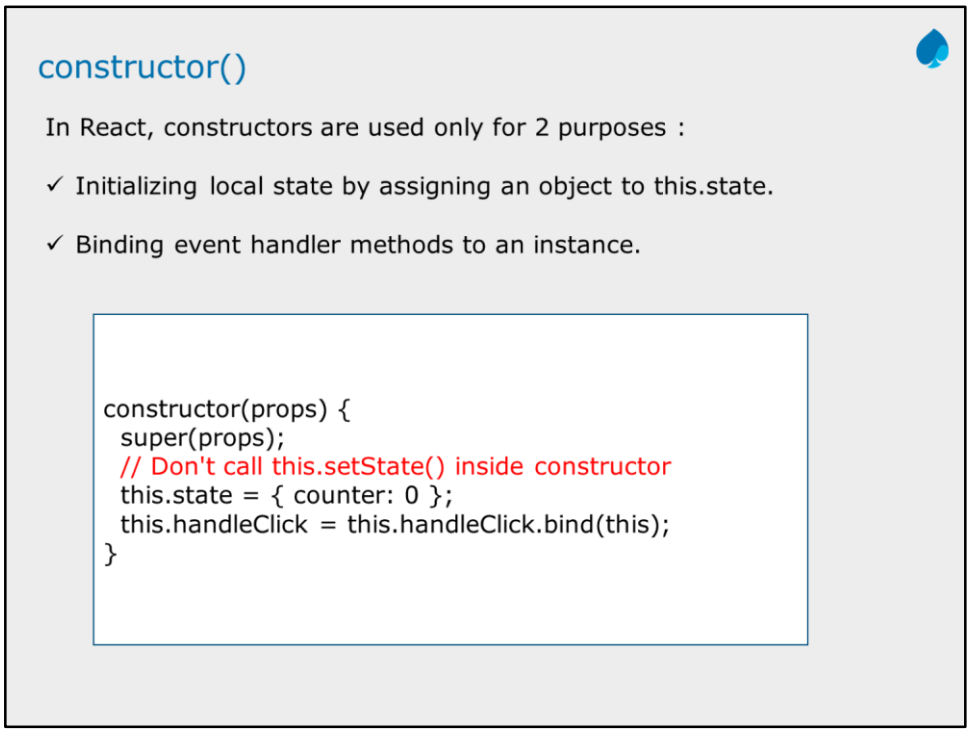
Using React's createClass:

```
getInitialState: function() {...},componentWillMount: function() {
console.log('componentWillMount');
},
```

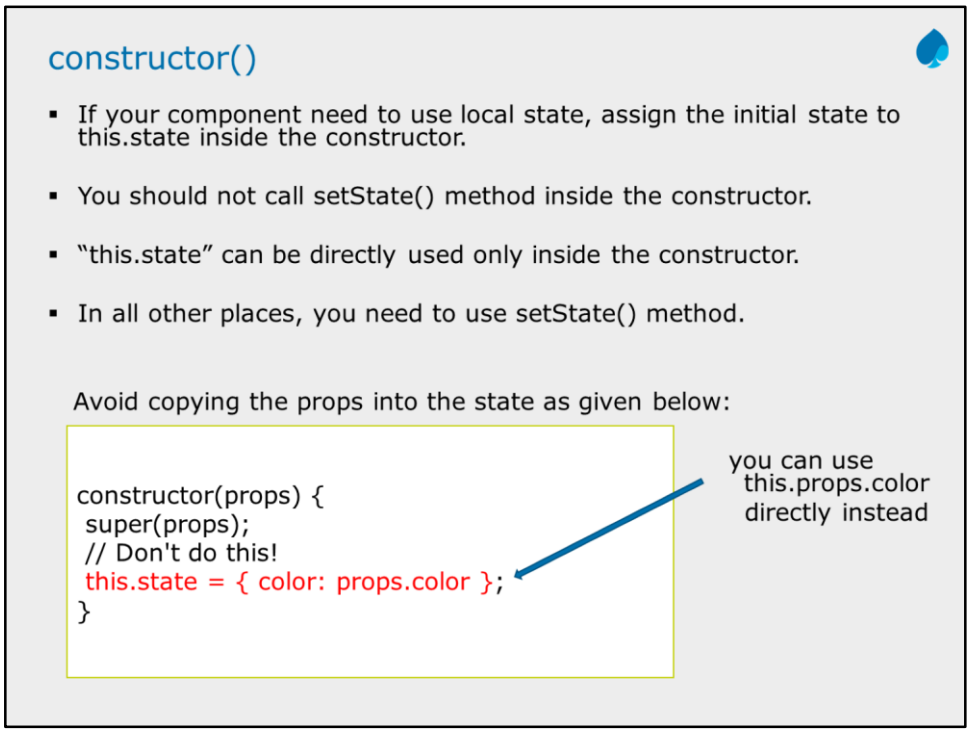
componentDidMount:

The API calls should be made in componentDidMount method always.

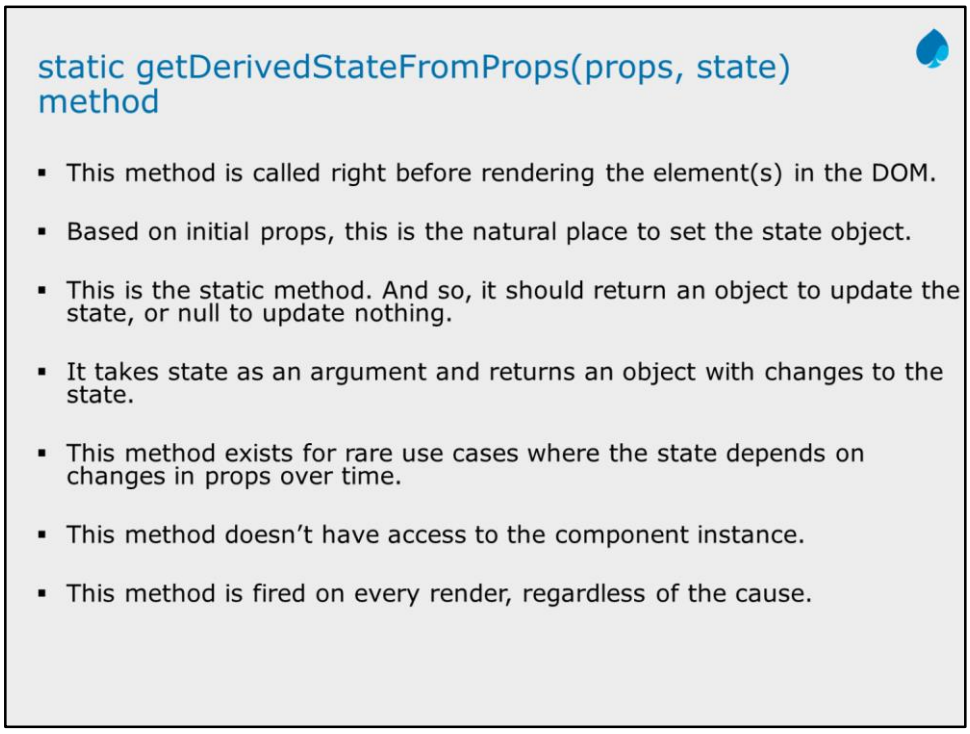
Instructor Notes:



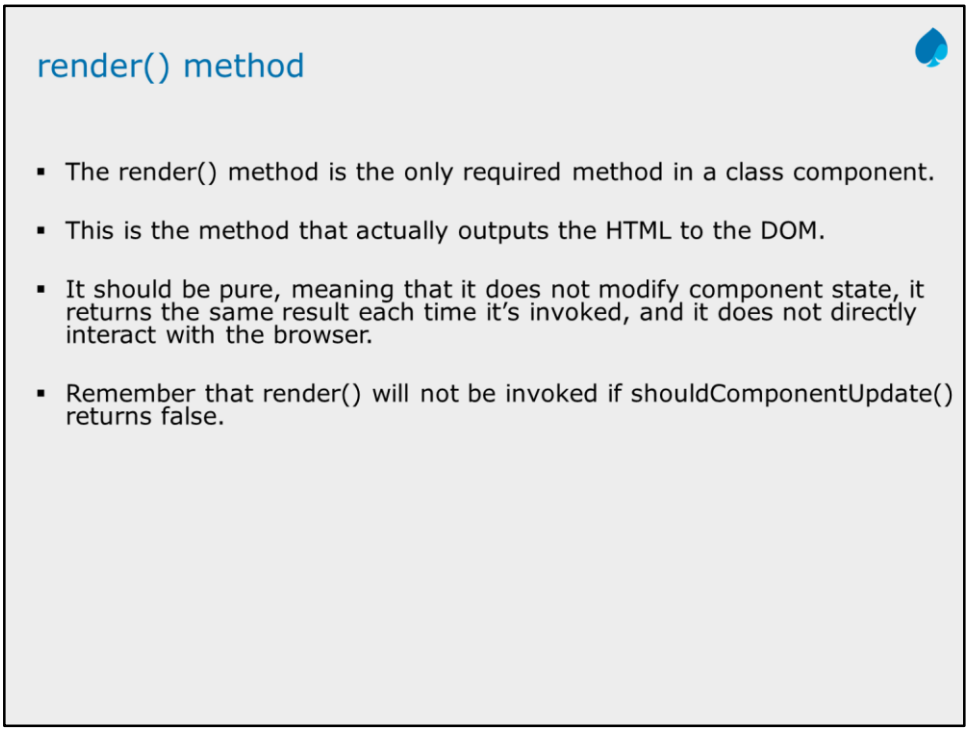
Instructor Notes:



Instructor Notes:



Instructor Notes:



Instructor Notes:

Example for render ()

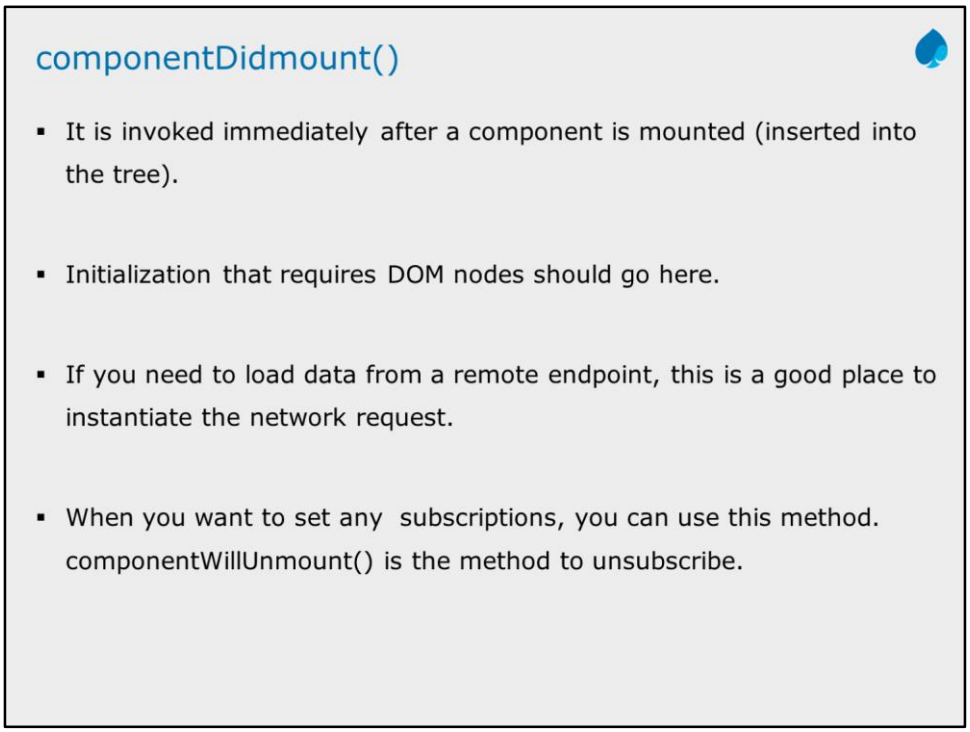
Index.js :

```
class Header extends React.Component
{
  render( ) {
    return (
      <h1>This is the content of the
          Header component</h1>
    );  }
}
ReactDOM.render(<Header />,
document.getElementById('root'));
```

index.html :

```
<html>
  <head>
    <title>React
App</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Instructor Notes:



ComponentWillMount:

This method is used for initializing the states or props

Note : It's very important to note that **calling *this.setState()* within this method(componentWillMount) will not trigger a re-render**.

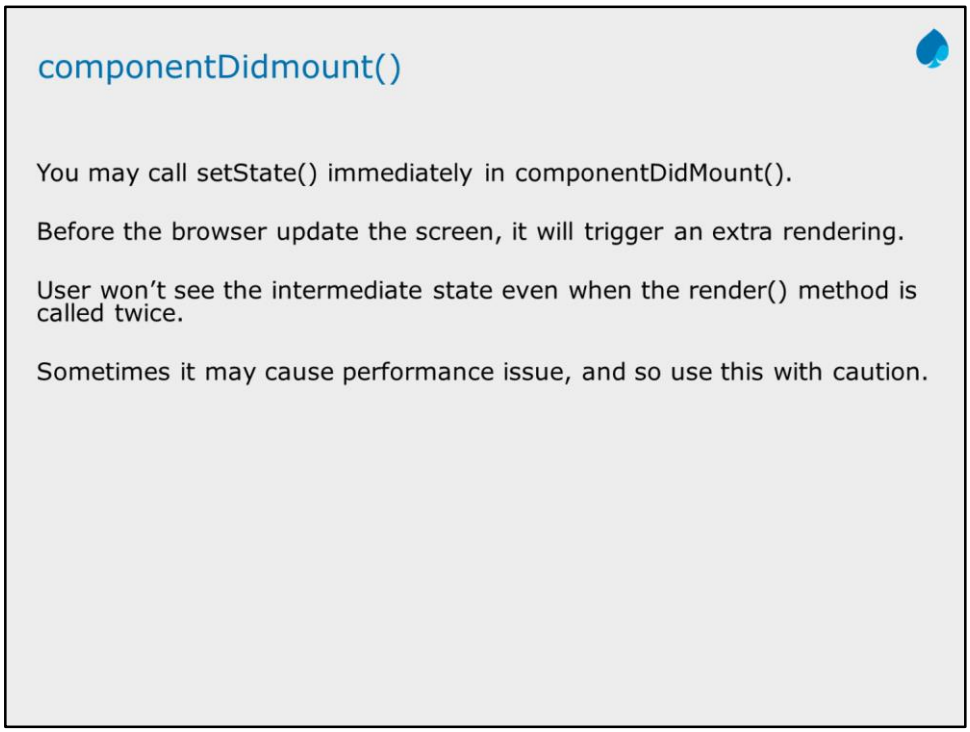
Using React's createClass:

```
getInitialState: function() {...},componentWillMount: function() {
console.log('componentWillMount');
},
```

componentDidMount:

The API calls should be made in componentDidMount method always.

Instructor Notes:



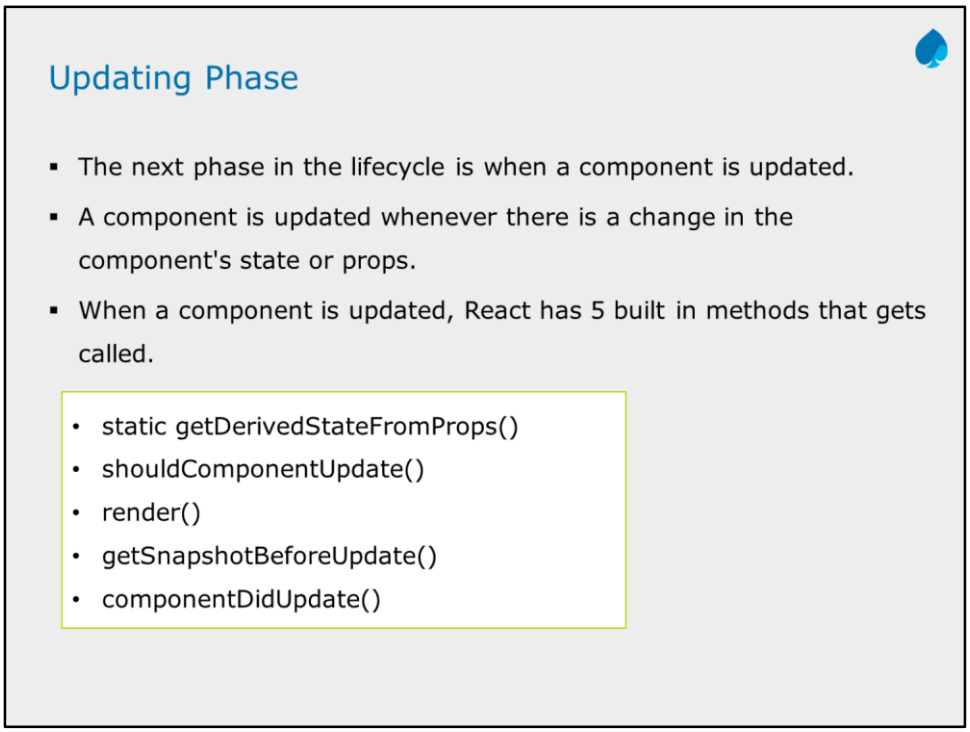
You may call setState() immediately in componentDidMount().

Before the browser update the screen, it will trigger an extra rendering.

User won't see the intermediate state even when the render() method is called twice.

Sometimes it may cause performance issue, and so use this with caution.

Instructor Notes:



The list of methods that will get called when the parent sends new props are as follows:

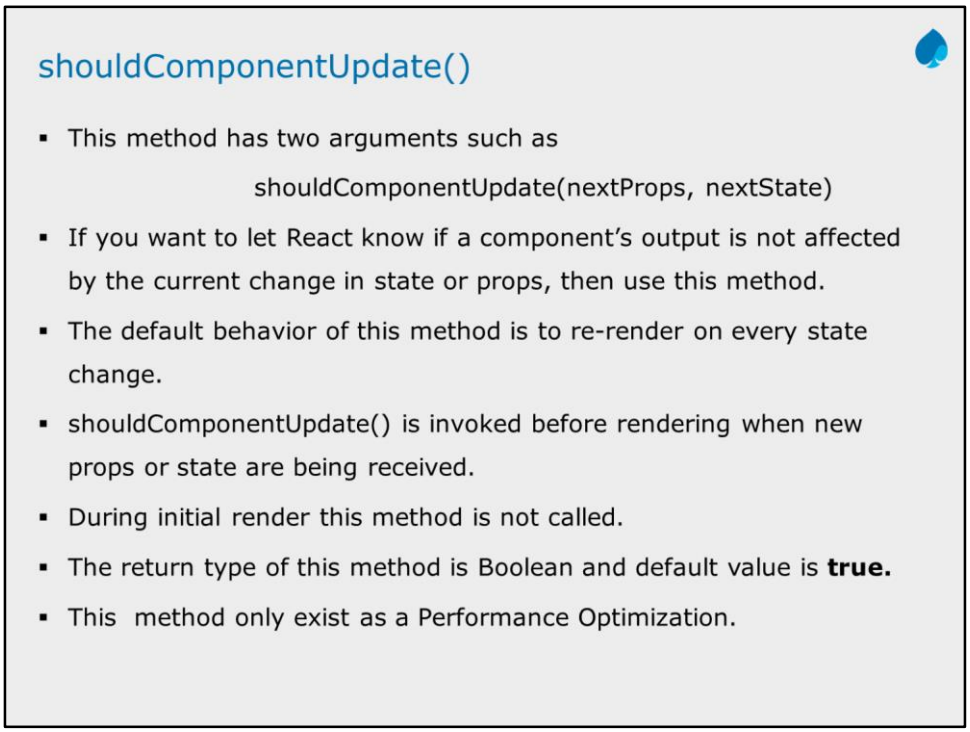
componentWillReceiveProps gets executed when the props have changed and is not first render.

Sometimes state depends on the props, hence whenever props changes the state should also be synced.

This is the method where it should be done.

Usage: This is how the state can be kept synced with the new props.

Instructor Notes:



The list of methods that will get called when the parent sends new props are as follows:

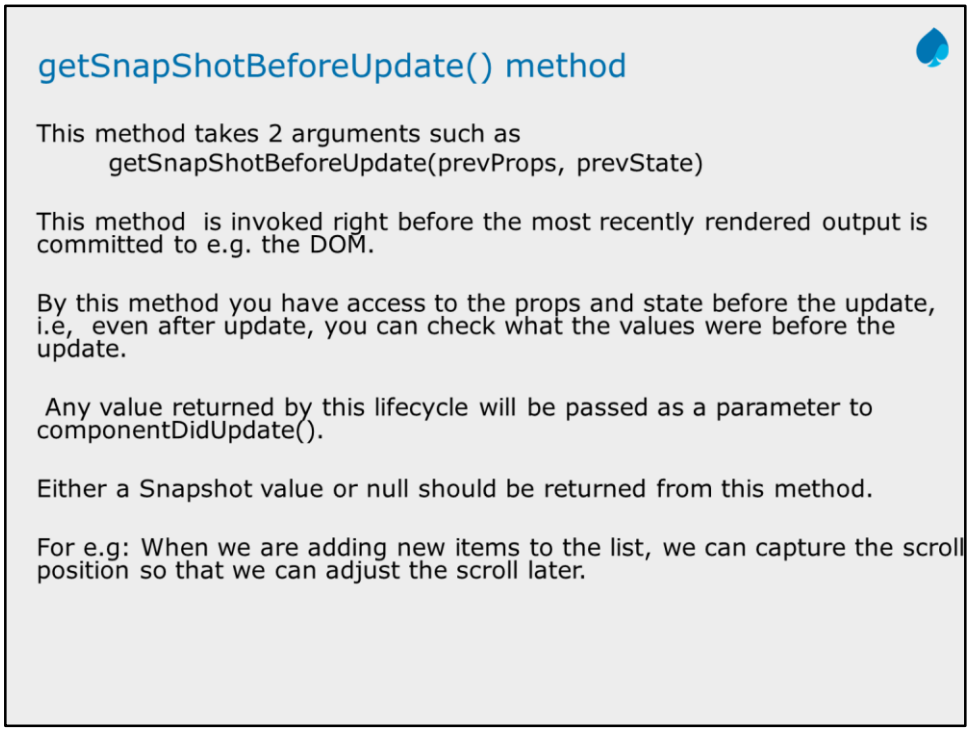
componentWillReceiveProps gets executed when the props have changed and is not first render.

Sometimes state depends on the props, hence whenever props changes the state should also be synced.

This is the method where it should be done.

Usage: This is how the state can be kept synced with the new props.

Instructor Notes:



Instructor Notes:

componentDidUpdate()

This method takes 3 arguments as given below:
componentDidUpdate(prevProps, prevState, snapshot)

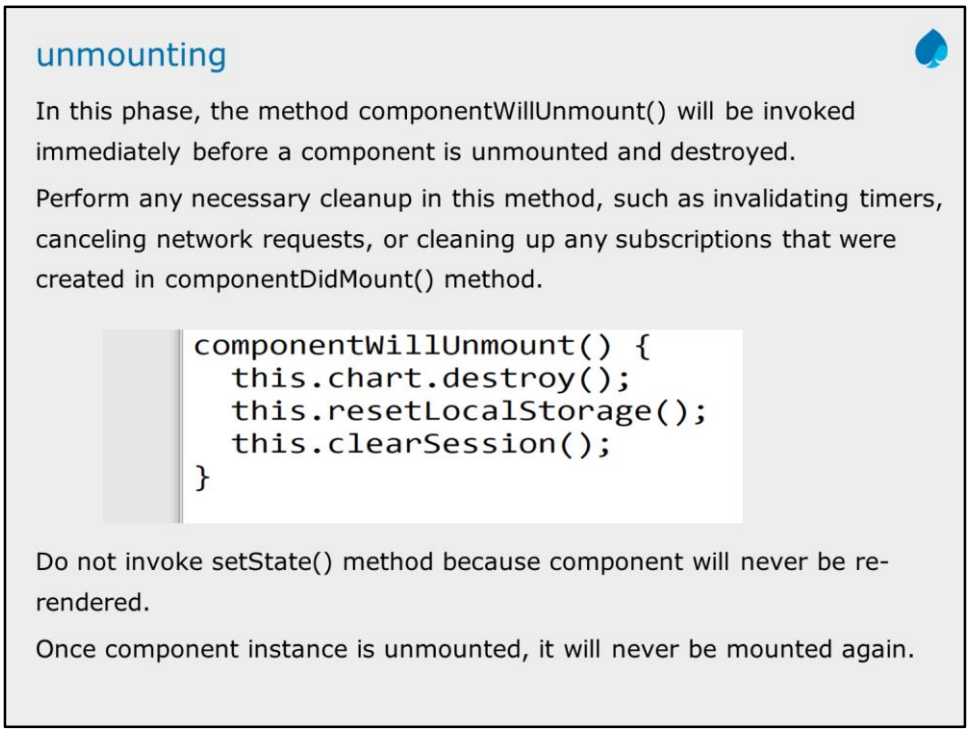
This method will not be called during initial render, it will be invoked immediately after updating occurs.

```
componentDidUpdate(prevProps) {
  // Typical usage (don't forget to compare props):
  if (this.props.userID !== prevProps.userID) {
    this.fetchData(this.props.userID);
} }
```

If your component implements the getSnapshotBeforeUpdate() lifecycle, the value it returns will be passed as a third "snapshot" parameter to componentDidUpdate().

Otherwise this parameter will be undefined. componentDidUpdate() will not be invoked if shouldComponentUpdate() returns false.

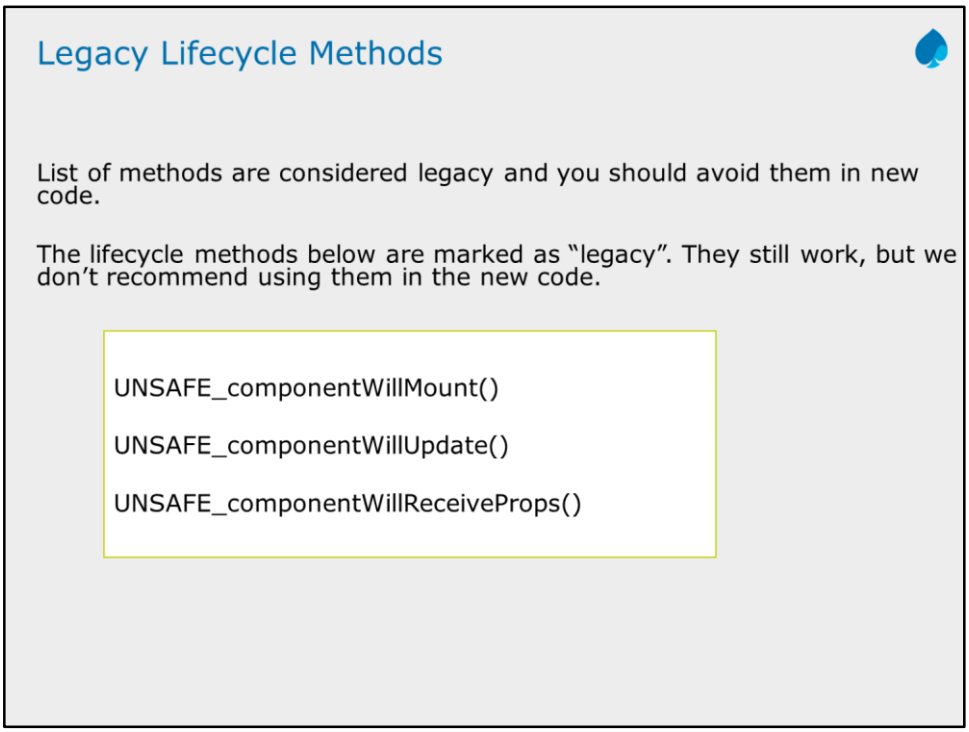
Instructor Notes:



Updating:

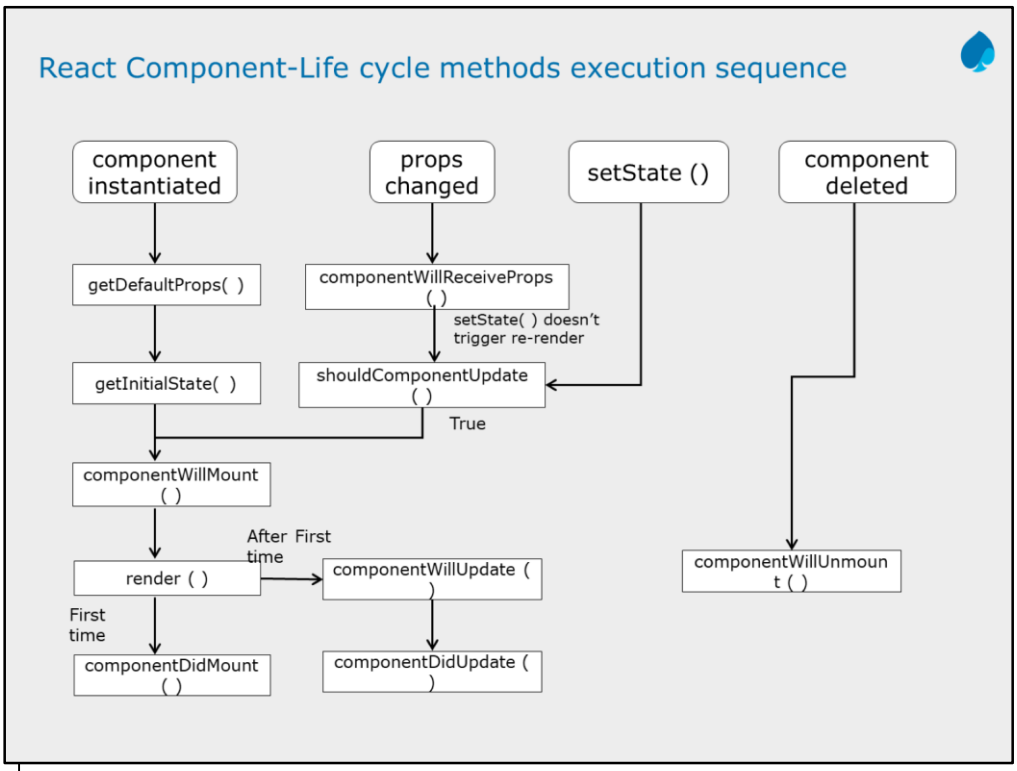
componentWillUnmount This method is the last method in the lifecycle. This is executed just before the component gets removed from the DOM.
Usage: In this method, we do all the cleanups related to the component. For example, on logout, the user details and all the auth tokens can be cleared before unmounting the main component.

Instructor Notes:



Instructor Notes:

Add instructor notes here.



getInitialState : The object returned by this method sets the initial value of this.state

getDefaultProps: The object returned by this method sets the initial value of this.props. If a complex object is returned, it is shared among all component instances.

componentWillMount : Invoked once immediately before the initial rendering occurs. Calling setState here does not cause a re-render.

render: Returns the jsx markup for a component. Inspects this.state and this.props create the markup. Should never update this.state or this.props

componentDidMount : Invoked once immediately after the initial rendering occurs. We have access the access to get the DOM node using **ReactDOM.findDOMNode**().

componentWillReceiveProps : Invoked whenever there is a prop change called before render. Not called for the initial render. Previous props can be accessed by this.props. calling setState here does not trigger an additional re-render.

shouldComponentUpdate : Determines if the render method should run in the subsequent step . Called before a render. Not called for the initial render. If the render method to execute in the next step return true, else return false.

componentWillUpdate : Called immediately before a render. this.setState() cannot be used in this method.

componentDidUpdate : Called immediately after a render.

componentWillUnmount : Called immediately before a component is unmounted.

Instructor Notes:

Pure Components

React.Component life cycle hook has shouldComponentUpdate method which is set to always return true. This may be good but React might trigger unnecessary re-renders.

One way to deal with these extra re-renders is to change the shouldComponentUpdate function to check when your component needs to update.

Another way to stop extra re-renders is to use a PureComponent. React.PureComponent is similar to React.Component.

The difference between them is that React.Component doesn't implement shouldComponentUpdate(), but React.PureComponent implements it with a shallow prop and state comparison.

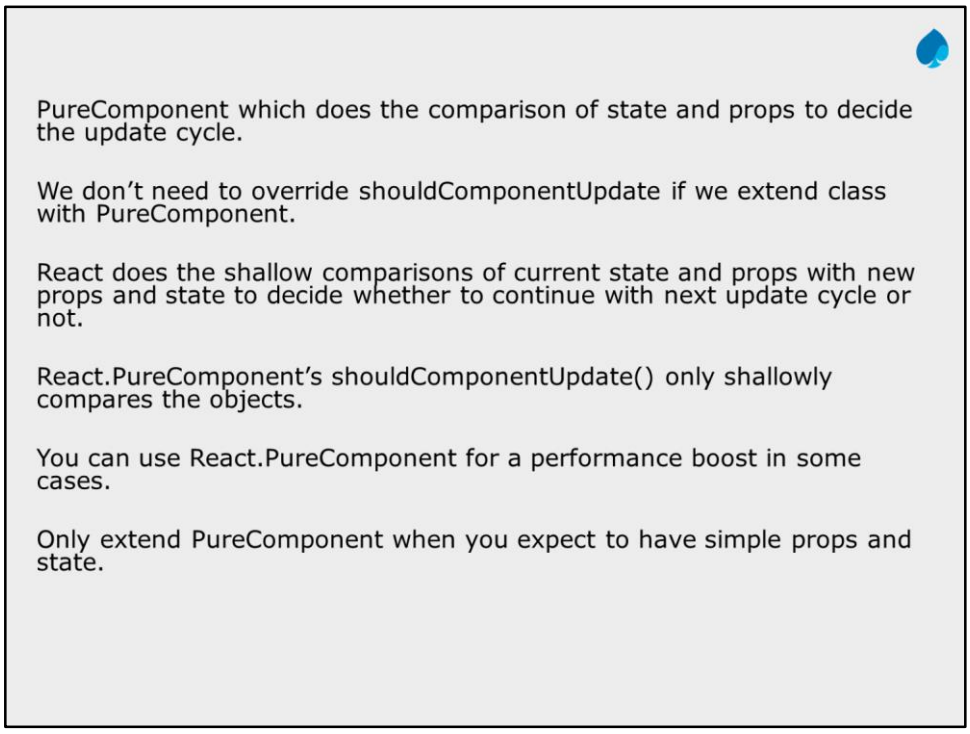
```
import React, {PureComponent} from 'react';
class Demo extends PureComponent{
  render( ){
    return '';
  } }
```

When to use Pure Components?
Suppose you creating a dictionary page in which you display the meaning of all the English words starting with A. Now you can write a component which takes a word and its meaning as props and return a proper view. And suppose you using pagination to display only 10 words at a time and on scroll asking for another 10 words and updating the state of the parent component. Pure Components should be used in this case as it will avoid rendering of all the words which rendered in previous API request.
Also in cases where you want to use lifecycle methods of Component then we have to use Pure Components as stateless components don't have lifecycle methods.

NOTE : above is from medium.com website

Note
React.PureComponent's shouldComponentUpdate() only shallowly compares the objects. If these contain complex data structures, it may produce false-negatives for deeper differences. Only extend PureComponent when you expect to have simple props and state, or use forceUpdate() when you know deep data structures have changed. Or, consider using immutable objects to facilitate fast comparisons of nested data.
Furthermore, React.PureComponent's shouldComponentUpdate() skips prop updates for the whole component subtree. Make sure all the children components are also "pure".

Instructor Notes:



Instructor Notes:

Add instructor notes here.

Demo

React-Component-Life-Cycle-Methods-Execution

react-create-componentlifecycle
React-pure_component

Instructor Notes:

REACT's DOM Updating Strategy

When render() method is invoked, it doesn't immediately render to real DOM.

In React every UI piece is a component, and each component has a state.

Whenever there is a change in state, react determines it.

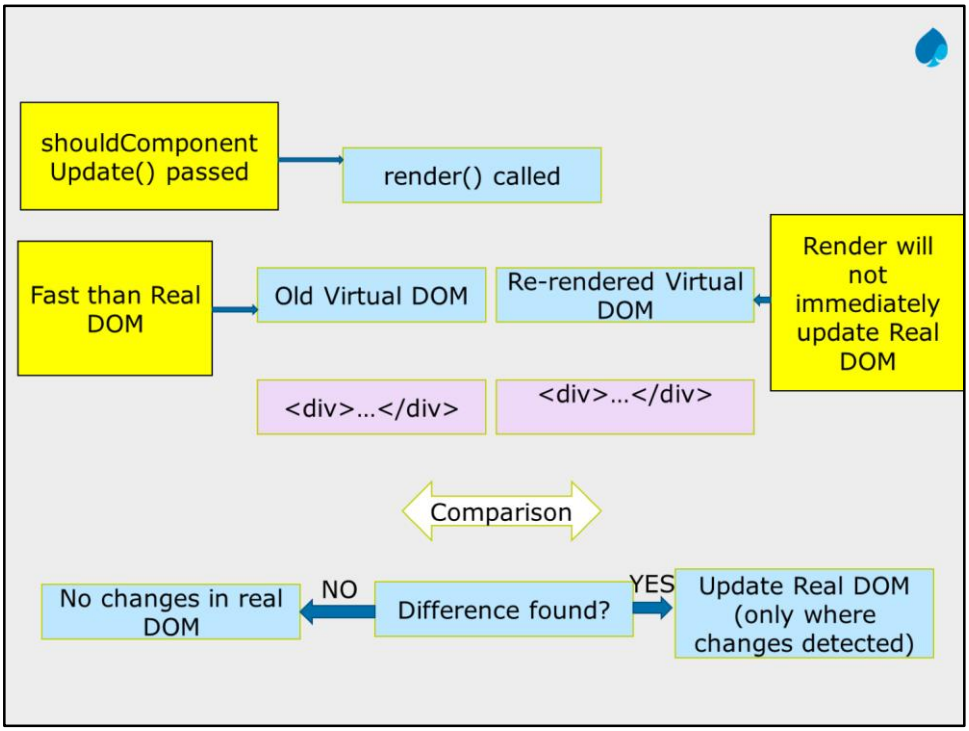
When component's state changes, React will update the virtual DOM Tree.

So, now React will compare the current version of Virtual DOM with previous virtual DOM. This is known as "Diffing".

So, React will update only those updated Objects in real DOM, not the whole real DOM.

This behavior of making change only in respective changed Objects will cause high performance.

Instructor Notes:



Instructor Notes:

Returning adjacent elements

Until React 16 it was required that a render method returns a single React element. With React 16 it is possible now to return an Array of elements.

Below(refer code 1 in note section) you can see an app rendering a list of Employee Assets. In React 16, it's now possible to extend this list with a component containing multiple new entries.

Generally when we render elements , we cannot render them as separate elements, we will get an error stating *'Parsing error: Adjacent JSX elements must be wrapped in an enclosing tag'* .We have to wrap them

Using a div or li tag or we have to make use of React.Fragments. [refer code 2 in note section]

Code 1:

```
class App extends Component{
   render(){
     return (
     <div>
       <li key="1">arun</li>
       <li key="2">ragul</li>
     </div>
                    );

}
}
```

CODE 2:

```
return (
<div>
<h1>hi</h1>
</div>
<div>
<li key="1">arun</li>
<li key="2">ragul</li>
</div>
```

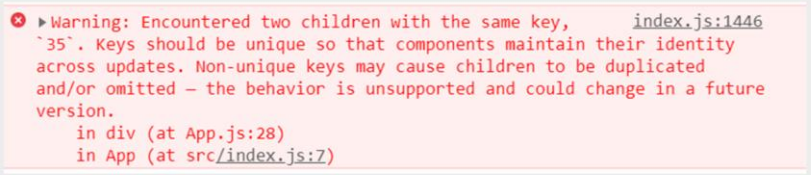
Both div should have been kept in another div,

Instructor Notes:

Returning adjacent elements contd.

Another important thing is we cannot render array of elements also,

If we have an array of data and on rendering them, we have to ensure that they have unique keys to separate them from other elements else we will get the below error.[refer code 3 in note section]



You will find in note section that, key is added which must be unique for each elements to get rendered as array of elements.

Code 3:

In the below code 2 id has 35 as its value which will create duplicate keys warning while rendering

```
let shoppingCart = [
{id: 35, item: 'jumper', color: 'red', size: 'medium', price: 20},
{id: 35, item: 'shirt', color: 'blue', size: 'medium', price: 15},
{id: 71, item: 'socks', color: 'black', size: 'all', price: 5},
]



class App extends Component{
render(){
const items = shoppingCart.map((item, key) =>
<li key={item.id}>{item.name}</li>
);

return ( <div><h1>hi</h1></div>);
}
}
```

Instructor Notes:

Add instructor notes here.

React Fragments

- React@16.2 introduces the Fragment component. Until React@16 a component could only return a single element.
- If returning more than one and then we can see a message like "*Adjacent JSX elements must be wrapped in an enclosing tag*".
- If in case need more than one, then it should be wrapped with another container tag.(say a ul or span or div etc,. or array or Fragments).

```
import React, { Component } from 'react'
class BlogPostExcerpt extends Component
 {
render() {
return (
<React.Fragment>
         <h1>{this.props.title}</h1>
         <p>{this.props.description}</p>
</React.Fragment>
         )
         }
}
```

React@16.2 introduces the Fragment component.
we can either import the Fragment component by using belwo statement,
import React, { Component, Fragment } from 'react'
Or
use it through the React object directly with React.Fragment

Instructor Notes:

Add instructor notes here.

React Fragments

Shorthand syntax + supported attributes : <></> -> A set of empty tags. Be aware though, the Fragment syntax is only supported as of Babel v7.0.0-beta.31.

```
import React, { Component } from 'react'
class Blogs extends Component
 {
render() {
return (
<>
          <h1>{this.props.title}</h1>
          <p>{this.props.description}</p>
</>
          )
          }
}
```

React@16.2 introduces the Fragment component.
we can either import the Fragment component by using belwo statement,
import React, { Component, Fragment } from 'react'
Or
use it through the React object directly with React.Fragment

Instructor Notes:

Add instructor notes here.

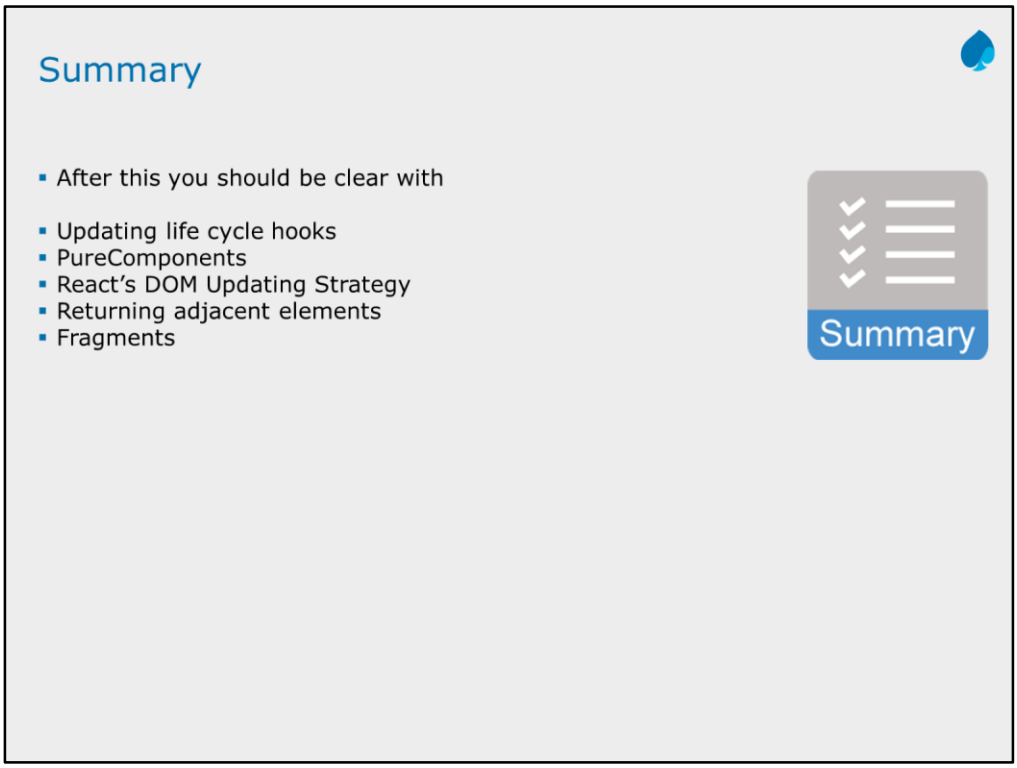
Demo

Fragments

react-fragments-2019

Instructor Notes:

Add instructor notes here.



Add the notes here.

Instructor Notes:

Add instructor notes here.

Review

What are the 3 React component's Life Cycle?

1. Mount,insert,demount,
2. Mounting, merge, unmounting
3. Mounting, merge, demounting
4. Mounting, update, unmounting

In React , Strong Type-checking is Provided by PropTypes? True or False.

_____ are used to share the code between multiple components.Which is an array of objects.

1. Prop
2. State
3. Mixins
4. components