**React Lifecycle Methods**

* Learn about Component Lifecycle methods of React.

 - Mounting

 - Updating

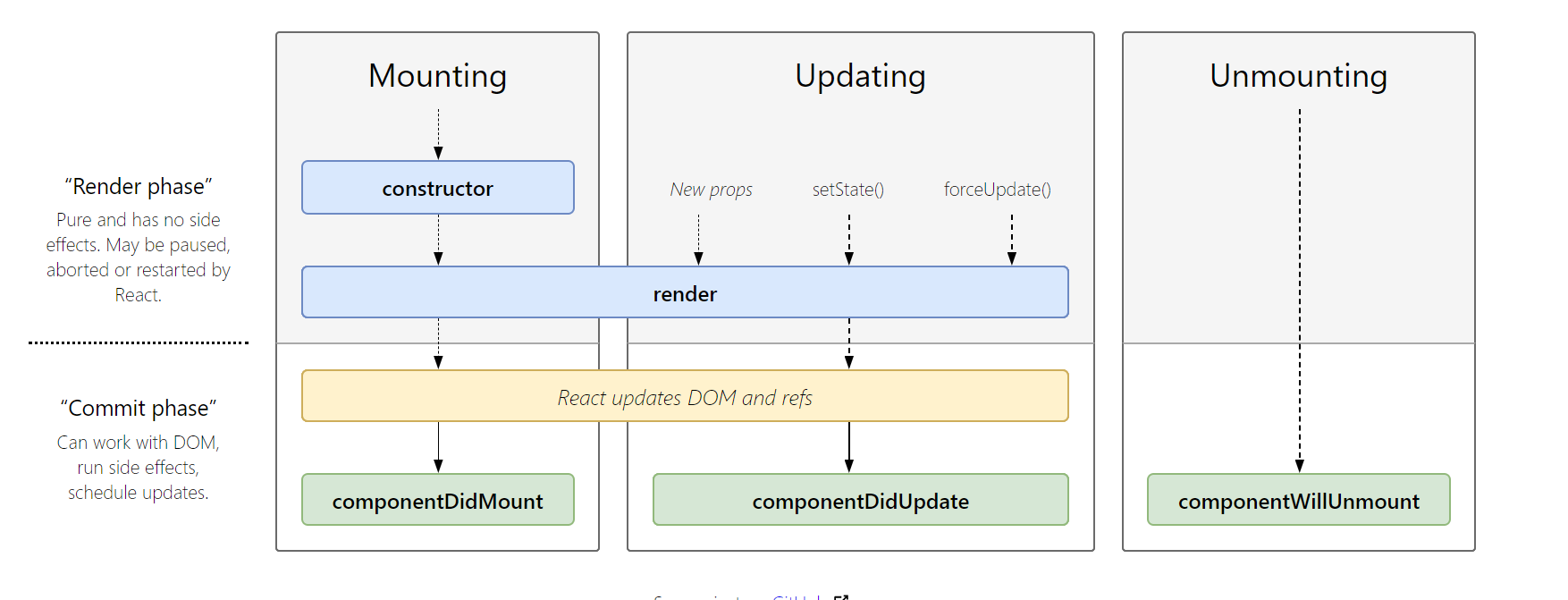
 - Unmounting

* Create an example code for each lifecycle method in the above three phases.
* Create a document in which all the methods should be explained with a code sample written in the document or pasted from your text editor.
* Push the document on github.

React lifecycle components methods are a series of events that from a birth of a react component till it’s death.

Here is a diagram of the three phases-

* Mounting
* Updating
* Unmounting

he

Mounting - The birth of your component:

In the Mounting lifecycle method, the following methods are called in order,

**Constructor()**:

This is the first method and known as ‘brought to life’ and is called before the component is mounted to the DOM. Usually, you’d initialise state and bind event handlers methods within the constructor method.

*const MyComponent extends React.Component {*

*constructor(props) {*

*super(props)*

*this.state = {*

*points: 0*

*}*

*this.handlePoints = this.handlePoints.bind(this)*

*}*

*}*

Also, the constructor is NOT where to introduce any side-effects or subscriptions such as event handlers.

**static getDerivedStateFromProps()**:

This method is called before the component is rendered to the DOM on initial mount.

Let us consider a simple component that renders the number of points scored by a football team. This method allows a component to update its internal state in response to a change in props.

We put it in the **static getDerivedStateFromProps** method:

class App extends Component {

state = {

points: 10

}

static getDerivedStateFromProps(props, state) {

return {

points: 1000

}

}

render() {

return (

<div className="App">

<header className="App-header">

<img src={logo} className="App-logo" alt="logo" />

<p>

You've scored {this.state.points} points.

</p>

</header>

</div>

);

}

}

**Render:**

After the static getDerivedStateFromProps method is called, the next lifecycle method in line is the render method:

class MyComponent extends React.Component {

render() {

return <h1> Hurray! </h1>

}

}

In the event that you don’t want to render anything, you could return a Boolean or null within the render method

class MyComponent extends React.Component {

render() {

return null

}

}

class MyComponent extends React.Component {

// guess what's returned here?

render() {

return (2 + 2 === 5) && <div>Hello World</div>;

}

}

**componentDidMount():**

After render, to mount the component to the DOM, **componentDidMount()** is invoked:

*class Header extends React.Component {*

*constructor(props) {*

*super(props);*

*this.state = {favoritecolor: "red"};*

*}*

*componentDidMount() {*

*setTimeout(() => {*

*this.setState({favoritecolor: "yellow"})*

*}, 1000)*

*}*

*render() {*

*return (*

*<h1>My Favorite Color is {this.state.favoritecolor}</h1>*

*);*

*}*

*}*

*ReactDOM.render(<Header />, document.getElementById('root'));*

**Updating Lifecycle Methods:**

**static getDerivedStateFromProps():**

Firstly, the static getDerivedStateFromProps method is also invoked. That’s the first method to be invoked.

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

static getDerivedStateFromProps(props, state) {

return {favoritecolor: props.favcol };

}

changeColor = () => {

this.setState({favoritecolor: "blue"});

}

render() {

return (

<div>

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

<button type="button" onClick={this.changeColor}>Change color</button>

</div>

);

}

}

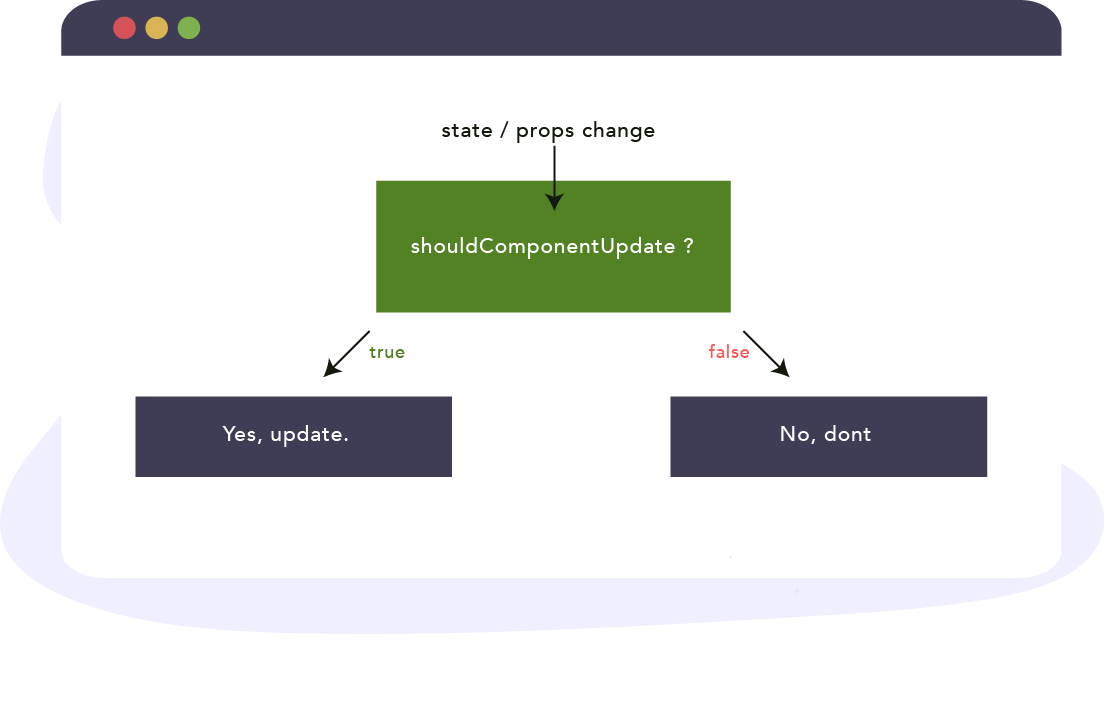
ReactDOM.render(<Header favcol="yellow"/>, document.getElementById('root'));

What’s important to note is that this method is invoked in both the mounting and updating phases. The same method.

**shouldComponentUpdate():**

As soon as the static getDerivedStateFromProps method is called, the shouldComponentUpdate method is called next.

By default, or in most cases, you’ll want a component to re-render when state or props changes. However, you do have control over this behavior.

**z**

*class Header extends React.Component {*

*constructor(props) {*

*super(props);*

*this.state = {favoritecolor: "red"};*

*}*

*shouldComponentUpdate() {*

*return false;*

*}*

*changeColor = () => {*

*this.setState({favoritecolor: "blue"});*

*}*

*render() {*

*return (*

*<div>*

*<h1>My Favorite Color is {this.state.favoritecolor}</h1>*

*<button type="button" onClick={this.changeColor}>Change color</button>*

*</div>*

*);*

*}*

*}*

*ReactDOM.render(<Header />, document.getElementById('root'));*

**render():**

*class Header extends React.Component {*

*constructor(props) {*

*super(props);*

*this.state = {favoritecolor: "red"};*

*}*

*changeColor = () => {*

*this.setState({favoritecolor: "blue"});*

*}*

*render() {*

*return (*

*<div>*

*<h1>My Favorite Color is {this.state.favoritecolor}</h1>*

*<button type="button" onClick={this.changeColor}>Change color</button>*

*</div>*

*);*

*}*

*}*

*ReactDOM.render(<Header />, document.getElementById('root'));*

**getSnapshotBeforeUpdate:**

In the getSnapshotBeforeUpdate() method you have access to the props and state before the update, meaning that even after the update, you can check what the values were before the update.

If the getSnapshotBeforeUpdate() method is present, you should also include the componentDidUpdate() method, otherwise you will get an error.

The example below might seem complicated, but all it does is this:

When the component is mounting it is rendered with the favorite color "red".

When the component has been mounted, a timer changes the state, and after one second, the favorite color becomes "yellow".

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

componentDidMount() {

setTimeout(() => {

this.setState({favoritecolor: "yellow"})

}, 1000)

}

getSnapshotBeforeUpdate(prevProps, prevState) {

document.getElementById("div1").innerHTML =

"Before the update, the favorite was " + prevState.favoritecolor;

}

componentDidUpdate() {

document.getElementById("div2").innerHTML =

"The updated favorite is " + this.state.favoritecolor;

}

render() {

return (

<div>

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

<div id="div1"></div>

<div id="div2"></div>

</div>

);

}

}

ReactDOM.render(<Header />, document.getElementById('root'));

**componentDidUpdate:**

componentDidUpdate() is invoked immediately after updating occurs. This method is not called for the initial render. We may call setState () immediately in componentDidUpdate (). Typically, in componentDidUpdate, we wrap the logical in a conditional so that you do not execute it on every state or props change.

*class Header extends React.Component {*

*constructor(props) {*

*super(props);*

*this.state = {favoritecolor: "red"};*

*}*

*componentDidMount() {*

*setTimeout(() => {*

*this.setState({favoritecolor: "yellow"})*

*}, 1000)*

*}*

*componentDidUpdate() {*

*document.getElementById("mydiv").innerHTML =*

*"The updated favorite is " + this.state.favoritecolor;*

*}*

*render() {*

*return (*

*<div>*

*<h1>My Favorite Color is {this.state.favoritecolor}</h1>*

*<div id="mydiv"></div>*

*</div>*

*);*

*}*

*}*

*ReactDOM.render(<Header />, document.getElementById('root'));*

**Unmounting:**

**componentWillUnmount:**

The componentWillUnmount method is called when the component is about to be removed from the DOM.

class Container extends React.Component {

constructor(props) {

super(props);

this.state = {show: true};

}

delHeader = () => {

this.setState({show: false});

}

render() {

let myheader;

if (this.state.show) {

myheader = <Child />;

};

return (

<div>

{myheader}

<button type="button" onClick={this.delHeader}>Delete Header</button>

</div>

);

}

}

class Child extends React.Component {

componentWillUnmount() {

alert("The component named Header is about to be unmounted.");

}

render() {

return (

<h1>Hello World!</h1>

);

}

}

ReactDOM.render(<Container />, document.getElementById('root'));