Component Mounting and Unmounting Lab (CodeGrade)

 (https://github.com/learn-co-curriculum/react-hooks-component-mounting-lab)  (https://github.com/learn-co-curriculum/react-hooks-component-mounting-lab/issues/new)

Learning Goals

- Use the `componentDidMount` lifecycle method
- Use the `componentWillUnmount` lifecycle method

Introduction

In this lab, we will be be implementing React lifecycle methods for a simple app: MultiTimer. MultiTimer allows users to start multiple one-second timers. Start as many timers as you need! Each timer is a single component and keeps track of its own time using state.

Go on and run the app with `npm start`. The app is already partially working, but there are some changes we need to make. By clicking 'Add Timer', timers will get added to the page, but stay at `0`.

In order to get our timers working properly, we will need to use component lifecycle methods to handle initiating and clearing an interval.

Deliverables

To pass the tests in this lab, you will need to write `componentDidMount` and `componentWillUnmount` methods where necessary in `App.js` and `Timer.js`.

App - componentDidMount

Since App is the top level component, its `componentDidMount` method will be invoked before any other child components are even constructed.

You can always use the `constructor`, which fires first, to set up your initial state, so while it is possible to set state from `componentDidMount`, it isn't a common pattern. Using `componentDidMount` is instead reserved for taking initial actions within an app. Actions might include getting remote API data, setting cursor focus, or creating an interval or timeout.

The App component is keeping track of timers using an array of random ID numbers. This allows for easy removal and addition of Timer components.

In App, write a `componentDidMount` method that invokes the existing `handleAddTimer` class method.

Note: When writing lifecycle methods, avoid using arrow functions - while they may work in browser, we want these methods to exist on the prototype chain of whatever JavaScript class we've created. Lifecycle methods written using arrow functions will not exist on the prototype chain and will not pass the tests in this lab.

If you've got it working and have the app served up on a browser tab, you'll see that, upon refresh, a timer will be present. The timer is still not working, but that's okay for now.

Timer - componentDidMount

The `componentDidMount` method is often a good place to include **setInterval (https://www.w3schools.com/jsref/met_win_setinterval.asp)** or setTimeout functions, allowing you to delay something from happening on a component or cause some repeating change. Perfect for our timer app.

In the Timer component, there is already a method, `clockTick`, that handles updating the state. The state value, `time`, is then included in the render. We just need to set up an interval to call `clockTick`.

To create a **setInterval (https://www.w3schools.com/jsref/met_win_setinterval.asp)**, the best practice is to assign it to a variable within the scope of our class:

```
this.interval = setInterval(...)
```

Write a `componentDidMount` that initializes an interval. Pass `clockTick` as the callback function and set it to `1000` to update every second.

Once this is working, in our application, when a new timer is added, we should see the displayed number increase every second!

Timer - componentWillUnmount

It is important to make sure that we clean up after ourselves when it comes to intervals. Not cleaning up can cause memory leaks (meaning that system memory is allocated to something that is no longer necessary and won't free up), as intervals can keep firing after a component unmounts.

To clear an interval, we use the built in `clearInterval` method, passing in the local variable:

```
clearInterval(this.interval);
```

Write a `componentWillUnmount` method in Timer that cleans up the interval you've created.

Run `npm test` to confirm you've passed the tests for adding `componentDidMount` and `componentWillUnmount` to both App and Timer.

Conclusion

To quickly recap, the `componentDidMount` method is useful for initiating one-time or ongoing actions within the logic of a component. It can also be used for DOM manipulation, fetching data or opening a web socket connection.

The `componentWillUnmount` method is most often used for cleaning up ongoing processes such as intervals, and it can also be used to halt ongoing activities involved in 3rd party libraries.

Resources

- **React: Component Specs and Lifecycle (https://reactjs.org/docs/react-component.html)**

This tool needs to be loaded in a new browser window

Load Component Mounting and Unmounting Lab (CodeGrade) in a new window