Assignment 08 - Let's get Classy

Owner	Pankaj Kumar

1: How do you create `Nested Routes react-router-dom` configuration?

```
We can create a `Nested Routes` inside a react router configuration as follows:
```

first call createBrowserRouter for routing different pages

```
const router = createBrowserRouter([
   {
      path: "/", // show path for routing
      element: <Parent />, // show component for particular path
      errorElement: <Error />, // show error component for path is different
      children: [ // show children component for routing
         {
            path: "/path",
            element: <Child />
         }
      ],
   }
])
```

Now we can create a nested routing for `/path` using `children` again as follows:

```
const router = createBrowserRouter([
   {
      path: "/",
      element: <Parent />,
      errorElement: <Error />,
      children: [
         {
        path: "/about",
        element: <About />,
        children: [
          {
            path: "profile", // // nested routing for subchild
            element: <Profile />,
          },
        ],
        }
      ],
   }
])
```

2: Read about `createHashRouter`, `createMemoryRouter` from React Router docs.

`createHashRouter` is useful if you are unable to configure your web server to direct all traffic to your React Router application. Instead of using normal URLs, it will use the `hash (#)` portion of the URL to manage the "application URL". Other than that, it is functionally the same as `createBrowserRouter`.

`createMemoryRouter` Instead of using the browsers history a memory router manages it's own history stack in memory. It's primarily useful for testing and component development tools like Storybook, but can also be used for running React Router in any non-browser environment.

3: What is the `order of life cycle method calls` in `Class Based Components`?

Following is the order of lifecycle methods calls in `Class Based Components`:

1. constructor()
2. render ()
3. componentDidMount()
4. componentDidUpdate()
5. componentWillUnmount()

For more reference: ***React-Lifecycle-methods-Diagram***

▼ **Explanation:-**

Class based components are executed in two phases : Render phase & commit phase.

Render phase is pure and no side effects. It may be paused, restarted or aborted by React (when child component is created for eg). The constructor(), render() and componentDidMount() happens in this phase.

In constructor, the props are passed to its parents.

These methods are called in the following order when an instance of a component is being created and inserted into the DOM:

Mounting :

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

- Initializing local state by assigning an object to this.state
- Binding event handler methods to an instance.

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

1. componentDidMount() - componentDidMount() is invoked immediately after a component is mounted (inserted into the tree). You may call setState() immediately in componentDidMount() so that it triggers re-render before the browser updates the screen.

Updating : 3. componentDidUpdate() - componentDidUpdate() is invoked immediately after updating occurs. This method is not called for the initial render.

Unmounting : 4. componentWillUnmount() -componentWillUnmount() is invoked immediately before a component is unmounted and destroyed. Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in componentDidMount().

4: Why do we use `componentDidMount`?

`componentDidMount` - In CBC it is the best place to make an API call.Like we make an api call inside useEffect in functional component .Bcz intially react first finishes the render() Phase and it updates the DOM ,Then it make an API call(it need to load some data).So it takes some time to load and also we use `componentDidMount` as a `async function` so it delays the component to print.It is called after intial or every render. Example:

```
    async componentDidMount() {
    //This is the best place we make an API call
    const data = await fetch("https://api.github.com/users/sam-0905");
    const json = await data.json();s
    this.setState({
      userInfo: json,
    });
    console.log("userInfo", json);
    console.log("componentDidMount");
  }
```

5: Why do we use `componentWillUnmount`? Show with `example`.

`componentWillUnmount` is useful for the cleanup of the application when we switch routes from one place to another. Since we are working with a SPA the component process always runs in the background even if we switch to another route.So it is required to stop those processes before leaving the page. If we revisit the same page, a new process starts that affects the browser performance.

For example, in Repo class, during componentDidMount() a timer is set with an interval of every one second to print in console. When the component is unmounted (users moves to a different page), the timer will be running in the background, which we might not even realise and causing huge performance issue. To avoid such situations the cleanup function can be done in componentWillUnmount, in this example `clearInterval`(timer) to clear the timer interval before unmounting Repo component.

6: (Research) Why do we use `super(props)` in `constructor`?

`super(props)` is used to inherit the properties and access of variables of the React parent class when we initialize our component.

super() is used inside constructor of a class to derive the parent's all properties inside the class that extended it. If super() is not used, then `Reference Error : Must call super constructor in derived classes before accessing 'this' or returning from derived constructor` is thrown in the console.

A component that extends `React.Component` must call the `super()` constructor in the derived class since it’s required to access this context inside the derived class constructor.

When you try to use props passed on parent to child component in child component using `this.props.name`, it will still work without super(props). Only super() is also enought for accessing props in render method.

The main difference between super() and super(props) is the this.props is undefined in child's constructor in super() but this.props contains the passed props if super(props) is used.

7: (Research) Why `can't we have` the `callback function` of `useEffect async`?

`useEffect` expects it's callback function to return nothing or return a function (cleanup function that is called when the component is unmounted). If we make the callback function as `async`, it will return a `promise` and the promise will affect the clean-up function from being called.

Solution to this is not making the callback function async but created another async function inside callback function of useEffect().