Wadood Alam

Antra React Training-Assignment 4

1. What is the difference between instance methods and static methods?
    a. Instance methods are called on the instance of the class while static methods are called on the class itself. In simpler terms, the instance method can access and manipulate the instance properties, unlike static methods. However, static methods are used when the behavior of the function applies to the class as a whole. An example would be:

```
class MyClass {
   constructor(name) {
     this.name = name;
   }
    // Instance method
   greet() {
     console.log(`Hello, ${this.name}!`);
   }

   // Static method
   static greetStatic() {
     console.log('Hello from a static method!');
   }
```

2. How does Javascript handle concurrency?
    a. It uses the event loop(setTimer for example) functionality using call stack, web APIs, callback queue, promises, and microtask queues. The call stack executes synchronous code in a LIFO manner, while async operations are managed by Web APIs that offload tasks like HTTP requests and timers. When these tasks are complete, their callbacks are placed in the callback queue. Promises, once resolved, place their handlers in the microtask queue, which has higher priority than the callback queue. The event loop continuously checks the call stack and processes tasks from the microtask queue before moving on to the callback queue, ensuring efficient execution of both synchronous and asynchronous code.

3. What is async/await? How does it differ from using the promise instance methods?
    a. Async and Await are just syntax sugar to handle promises which use `.then`, and `.catch` can lead to more nested and less readable code. Even though the functionality is very similar, async-await allows for writing more readable code and is similar to writing synchronous code. Error handling in async-await is done by try-catch blocks which is much more straightforward and intuitive to write(due to resemblance with synchronous code). Overall, async-await provides a cleaner and more readable way to write async code.

4. Can you use await outside of an async function?
    a. Await is not allowed outside of an async function and will result in a syntax error. As javascript is a non-blocking language i.e. it holds the async tasks but the synchronous tasks are allowed execution and pause execution of async code until the promises are resolved or rejected. If we define the await outside of the async function, there is no immediate function boundary or module context to manage the asynchronous pause and resume. Hence, disrupting javascript's typical flow.

5. What is callback hell and why is it considered a problem?
    a. Callback hell is when there are nested callbacks to handle async operations. This arises when multiple async tasks are chained together with callbacks. This is a problem as, firstly, it produces a very hard-to-read code leading to difficult maintenance, debugging, and understanding of the code. Secondly, error handling for each nested callback becomes more complex as each callback requires its own error handling leading to repetitive and scattered error management code throughout the application.