Here are 20 unique **JavaScript code challenges** covering **Asynchronous Programming**, **Promises**, **Async/Await**, **Callbacks**, and **Object-Oriented Programming** (OOP), including topics like **Classes**, **Objects**, **Inheritance**, **Abstraction**, and **Encapsulation**:

Asynchronous Programming & Promises

1. Create a Promise

- Create a simple Promise that resolves with the message `"Success"` after 2 seconds, then log the message.

2. Handle Promise Rejection

- Create a Promise that rejects with an error message `"Something went wrong"` after 1 second, and handle the rejection.

3. Chain Multiple Promises

- Create two promises that resolve with different values. Chain them to log both results.

4. Promise with `finally()`

- Write a promise that resolves successfully, and use `.finally()` to log a message after the promise is settled (resolved or rejected).

5. Promise.all() to Fetch Multiple APIs

- Fetch data from multiple APIs using `Promise.all()`. Log the results when all promises have resolved.

Async/Await

6. Basic Async/Await Example

- Create an `async` function that fetches user data from an API using `fetch()`. Use `await` to wait for the response and log the data.

7. Async Function with Error Handling

- Create an `async` function to fetch data. If the fetch fails, catch the error and log `"Fetch failed"`.

8. Convert Callback-based Function to Async/Await

- Convert a function using a callback into an `async` function using `await`. Use a simple `setTimeout` to simulate asynchronous behavior.

9. Parallel Async Calls with `Promise.all()` and `await`

- Use `Promise.all()` to call multiple asynchronous operations in parallel using `await`. Wait for all operations to finish and then log the results.

10. Async/Await with Error Handling

- Create an `async` function that makes two API calls. Use `try...catch` for error handling, and log an error message if any of the calls fail.

Callbacks

11. Callback Example

- Write a function that simulates reading data from a database and then calls a callback with the data.

12. Nested Callbacks

- Create two callback functions, one for fetching data from an API and another for processing that data. Nest the callbacks appropriately.

13. Callback Hell

- Create a simple example where three asynchronous functions are nested using callbacks. Refactor the code to avoid "callback hell".

14. Using Callbacks with `setTimeout`

- Use `setTimeout` to simulate an asynchronous operation, and pass a callback function to be executed after a delay.

15. Handle Multiple Callbacks

- Write a function that takes multiple callbacks and executes them sequentially, logging the result after each callback is called.

Object-Oriented Programming (OOP)

16. Create a Class and Object

- Write a simple class called `Person` with properties like `name` and `age`, and create an instance of that class.

17. Class with Methods

- Write a `Car` class with properties like `make`, `model`, and a method `start()` that logs `"Car started"`.

18. Inheritance in OOP

- Create a class `Animal` with a method `speak()`. Then, create a subclass `Dog` that inherits from `Animal` and overrides the `speak()` method.

19. Encapsulation Example

- Write a `BankAccount` class that has a private `balance` property and public methods `deposit()` and `withdraw()` to modify the balance.

20. Abstraction in OOP

- Create a class `Shape` with an abstract method `area()` and extend it into classes `Circle` and `Rectangle`, each implementing the `area()` method differently.