Name: Nardine William Boules
Track: Open Source – Alexandria
Intake 44

Day 4 Report

Abstract Classes vs. Interfaces

In JavaScript, there is no direct equivalent of abstract classes or interfaces as found in languages like Java or TypeScript. However, developers often simulate similar concepts using different patterns.

Abstract Classes:

- **Instantiation:**
  Abstract classes cannot be directly instantiated. They exist to be subclassed by other classes.
- **Implementation:**
  1. Abstract classes can include both abstract methods (methods without implementation) and concrete methods (methods with implementation).
  2. Abstract classes can have properties (attributes) in addition to methods.
- **Extending:**
  Subclasses extend abstract classes. The abstract class provides a common structure and can include shared behavior that is inherited by its subclasses.
- **Usage:**
  Useful for code reuse: Abstract classes are useful for sharing common functionality among related classes, providing a template for implementation.

Interfaces:

- **Instantiation:**
  Interfaces cannot be instantiated directly. They define a contract for classes to implement.
- **Implementation:**
  Interfaces only contain method signatures (names and parameter lists) without any implementation. They focus on what methods a class should have without specifying how they should be implemented.
- **Extending:**
  Classes implement interfaces. Multiple interfaces can be implemented by a single class, allowing for flexibility in defining behavior.
- **Usage:**
  Interfaces are used to enforce that a class must implement certain methods. They provide a way to ensure that classes adhere to a specific set of behaviors.

Simulating Abstract Classes and Interfaces in JavaScript:

In JavaScript, developers often use constructor functions, prototypes, and conventions to achieve similar results:

Abstract Classes: Use a constructor function along with prototypes to create a base class that cannot be instantiated on its own and may contain both abstract and concrete methods.

Interfaces: Use documentation, conventions, or explicit checks for method existence to define a contract that classes must adhere to.

Choosing Between Simulation Approaches:

- Use **Abstract Classes** when you want to provide a common structure with the possibility of shared implementations among related classes.

- Use **Interfaces** when you want to enforce that classes implement a specific set of methods without providing any implementation details.