Workshop

- **As a tech enthusiast**, I want detailed specifications of the device, so what I understand the performance comparisons easily.
- **As a budget-conscious shopper**, I want clear pricing details and discount offers, so what I know if the device fits within my budget.
- **As a sustainability advocate**, I want information about the environmental impact of the device, so what I can decide if the purchase aligns with my eco-friendly values.
- **As a fashion-forward shopper**, I want to see design options, including colors and size dimensions, so what I can evaluate the aesthetic appeal of the device.
- **As a gamer**, I want to see GPU performance and refresh rates, so what I choose a device that can handle gaming smoothly.
  Analysis

1. Abstraction The project effectively abstracts real-world entities (like electronic devices and shopping carts) into simplified class representations. By focusing on essential attributes and behaviors, the code becomes more readable and maintainable. For example, the `Device` class abstracts common properties of all devices, while `ShoppingCart` encapsulates the actions associated with adding, removing, and calculating totals for items in a cart.

2. Encapsulation The project demonstrates strong encapsulation by restricting direct access to class attributes. This protects data integrity and prevents unintended modifications. For instance, the `stock_quantity` attribute of a `Device` is modified only through methods like `update_stock()`, ensuring that inventory levels are always accurate.

3. Inheritance While the current project doesn't explicitly use inheritance, the analysis correctly points out potential use cases. For example, a `Smartphone` class could inherit from a base `Device` class, allowing for specialization and code reuse.

4. Polymorphism The project could benefit from the introduction of polymorphism, especially in scenarios where different classes need to implement a common method in their own unique ways. For example, a `calculate_discount()` method could be defined in a base `Device` class, and subclasses like `Laptop` and `Smartphone` could override this method to provide specific discount calculations.

5. Single Responsibility Principle (SRP) The project adheres well to the SRP, as each class is responsible for a single, well-defined task. For instance, the `DeviceCategory` class is solely concerned with device categories, while the `ShoppingCart` class handles shopping cart operations.

6. Open-Closed Principle (OCP) The design is extensible, allowing for the addition of new features without modifying existing code. For example, new device types can be added by creating subclasses of the `Device` class.