Advanced Programming
Season 2024-III
Report of Workshop No. 2

Santiago Andrés Benavides Coral
20232020036
Systems engineer
Universidad Distrital Francisco José de Caldas

User stories:

1. **CONSULTATION OF PRODUCTS AVAILABLE FOR PURCHASE.**

As any student, I want to consult the list of products available to purchase so that I can make informed decisions about what items I want to buy.

2. **CONSULTATION OF PURCHASE CATEGORIES.**

As any student, I want to consult the category of products available to purchase so that I can easily find and select products relevant to me.

3. **CONSULTATION OF PRODUCT FOR SYSTEMS ENGINEER STUDENTS.**

As systems engineer, I want to consult the products corresponding to the systems engineering category so that I can easily find products for my career.

4. **CONSULTATION OF PRODUCT FOR ELECTRONIC ENGINEER STUDENTS.**

As electronic engineer, I want to consult the products corresponding to the electronic engineering category so that I can easily find products for my career.

5. **CONSULTATION OF PRODUCT FOR ARCHITECTURE STUDENTS.**

As architect, I want to consult the products corresponding to the architect category so that I can easily find products for my career.

6. **PURCHASE OF PRODUCTS.**

As a buyer, I want to select the products I want to buy and add them to my shopping cart so that I can proceed with checkout.

7. **PURCHASE CHECKOUT.**

As a buyer, I want to check out my shopping cart and proceed to fill out the information for payment and delivery so that I can successfully complete my purchase.

(NEW USER STORIES)

8. **MANAGE PRODUCT.**

As a manager, I can add new products to the inventory so that I can expand the available selection for buyers.

9. **EDIT PRODUCT DETAILS.**

As a manager, I can edit the details of existing products so that I can ensure accurate information is displayed.

10. **REMOVE PRODUCTS.**

As a manager, I can remove products from the inventory so that I can eliminate items there are not available for purchase.

Technical Sheet: Version 2 Improvements

This document outlines the enhancements made in the second version of the product management program. The primary upgrades focus on the separation of classes into distinct files and the introduction of an abstract class alongside the Manager and Client classes. These changes aim to improve the program's organization, readability, and maintainability.

Key improvements:

1. **Separation of Classes into Different Files.**

Organizing classes into separate files enhances code readability and allows developers to locate specific components easily. This structure promotes modularity, making it easier to maintain and update individual parts of the program without affecting the entire codebase.

2. **Implementation of Abstract Class.**

The introduction of the AbstractProductManager class serves as a blueprint for managing products, defining common methods that must be implemented by derived classes. This abstraction promotes code reuse and enforces a consistent interface for all product management functionalities.

3. **Manager and Client Class**

- Manager class**:**
  This class inherits from AbstractProductManager and provides enhanced permissions for product management. Managers can add, delete, and edit products, reflecting their role as users with administrative capabilities.
- Client class:
  Also inheriting from AbstractProductManager, this class restricts clients to viewing products only. Clients cannot modify the product list, ensuring that the integrity of the data is maintained while allowing users to browse available products.

Conclusion:

The improvements introduced in the second version of the program not only optimize its structure, but also increase its functionality and maintainability. Separating classes into individual files, along with implementing an abstract class, offers a clearer and more organized approach to product management. These changes help create a more robust application that can be easily adapted and expanded in the future.

Object-oriented principles analysis:

This program showcases the application of Object-Oriented Programming (OOP) principles in Python. Below is an analysis of the key OOP principles as applied to this program:

1. **ENCAPSULATION.**

Encapsulation involves keeping the fields within a class private and providing access through public methods. This helps bundle data with the methods that operate on it and control access.

Product: Encapsulates product details (product_id, name, category, price) and includes a method to represent the product as a string.

ProductRepository: Manages the list of products and hides the implementation details of loading products from a CSV file. It provides methods to interact with the product list.

ShoppingCart: Maintains a list of cart items and offers methods to add products, calculate totals, and list items.

Checkout: Manages the checkout process and interacts with the ShoppingCart object, encapsulating the checkout logic and user input handling.

AbstractProductManager: This abstract class serves as a base for managing product-related functionalities, encapsulating common methods and properties required for Client and Manager. It promotes code reuse and ensures that both roles implement their own specific behaviors while maintaining a consistent interface.

2. **ABSTRACTION.**

Abstraction hides complex implementation details and shows only the necessary features of an object, helping manage complexity by breaking down the system into more manageable parts.

ProductRepository: Abstracts the details of loading products from a CSV file. Users interact with the repository through methods like list_all_products and list_products_by_category without needing to know how the data is loaded or parsed.

ShoppingCart: Abstracts the complexity of managing the cart and calculating totals. Users interact with a simplified interface provided by methods like add_product, calculate_total, and list_cart_items.

AbstractProductManager: By providing a common interface for Client and Manager, it abstracts the specific implementation details of product management, allowing for easier modifications and enhancements in the future.

3. **INHERITANCE.**

Inheritance allows a class to inherit attributes and methods from another class, promoting code reusability and establishing a relationship between classes.

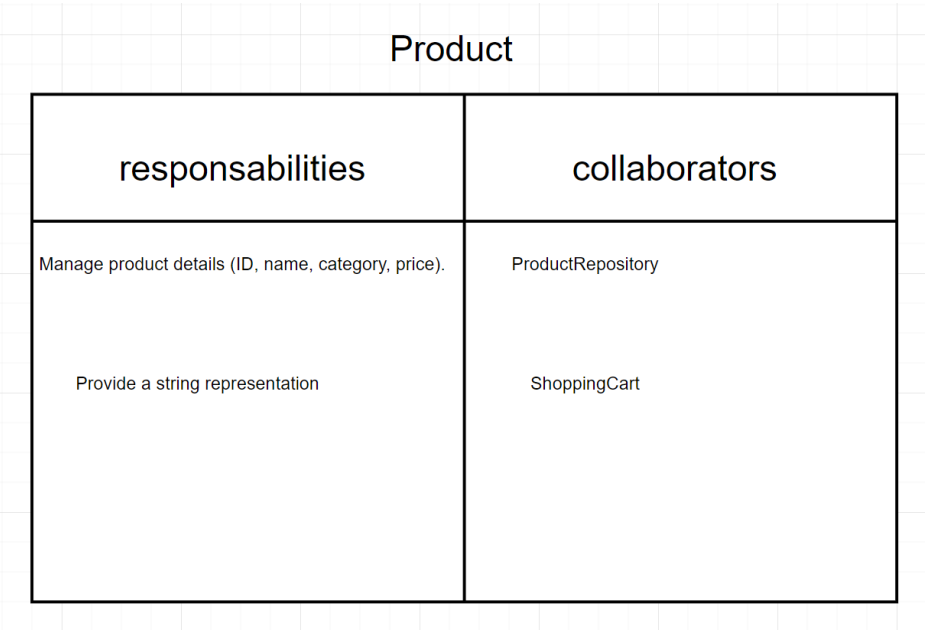
Client and Manager: Both classes inherit from AbstractProductManager. This inheritance allows them to share common functionalities while also implementing their specific behaviors for product management (e.g., Client can only list products, while Manager can add, remove, and edit them).

4. **POLYMORPHISM**

Polymorphism allows objects to be treated as instances of their parent class, even if they belong to different subclasses. This is useful for implementing methods that can operate on objects of different classes.

AbstractProductManager: The abstract methods defined in this class can be implemented differently in Client and Manager, allowing the same interface to interact with different product management behaviors seamlessly.

CRC Cards:



ProductRepository

responsabilities	collaborators
Load products from CSV List and filter products	Product ShoppingCart

ShoppingCart

responsabilities	collaborators
Manage cart items Calculate total price	Product Checkout

Checkout

responsabilities	collaborators
Process checkout Display cart summary and collect user details	ShoppingCart

AbstractProductManager

responsabilities	collaborators
Define abstract methods for adding, removing and editing Implement subclasses with that methods Serve as base classs to enforce a consistent interface management across different user roles	Client Manager

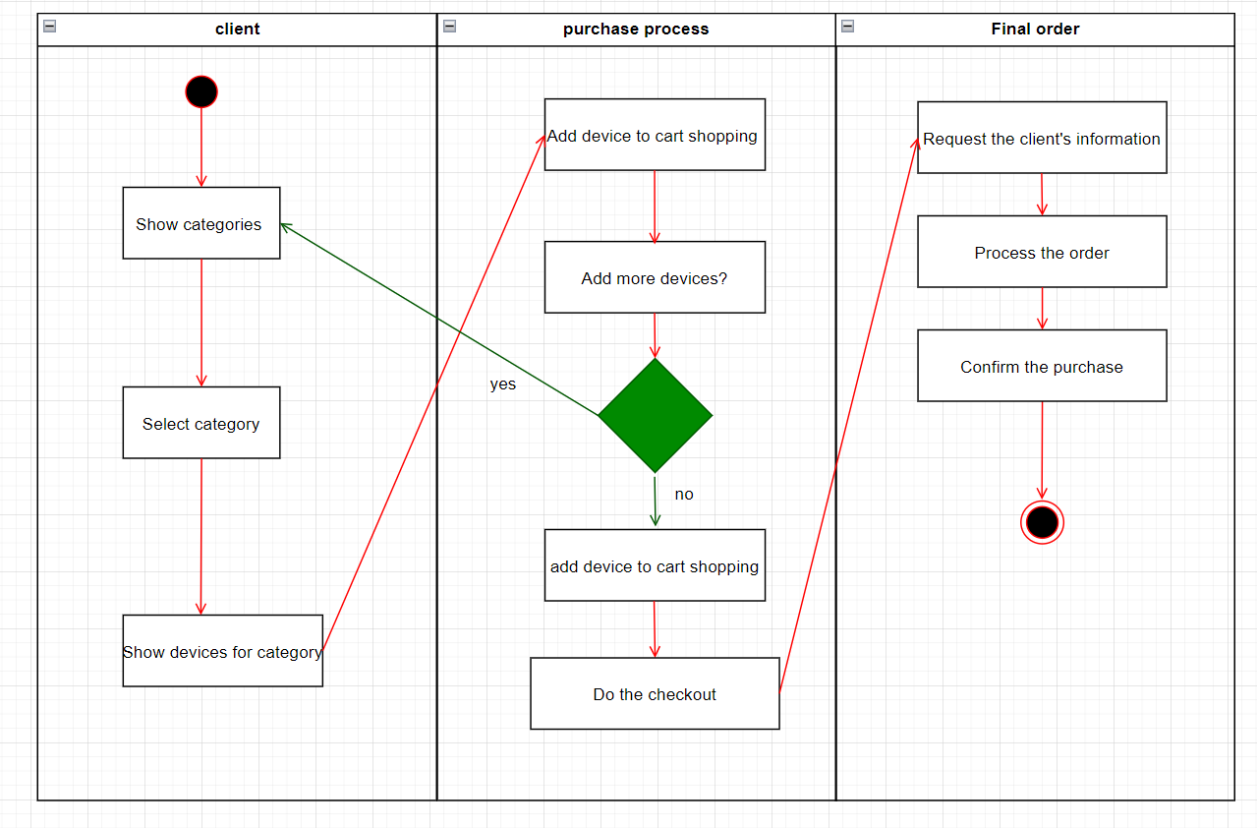
Client

responsabilities	collaborators
View products from the repository List all available products Filter products by category Deny permissions for adding, removing or editing products	ProductRepository Product AbstractProductManager

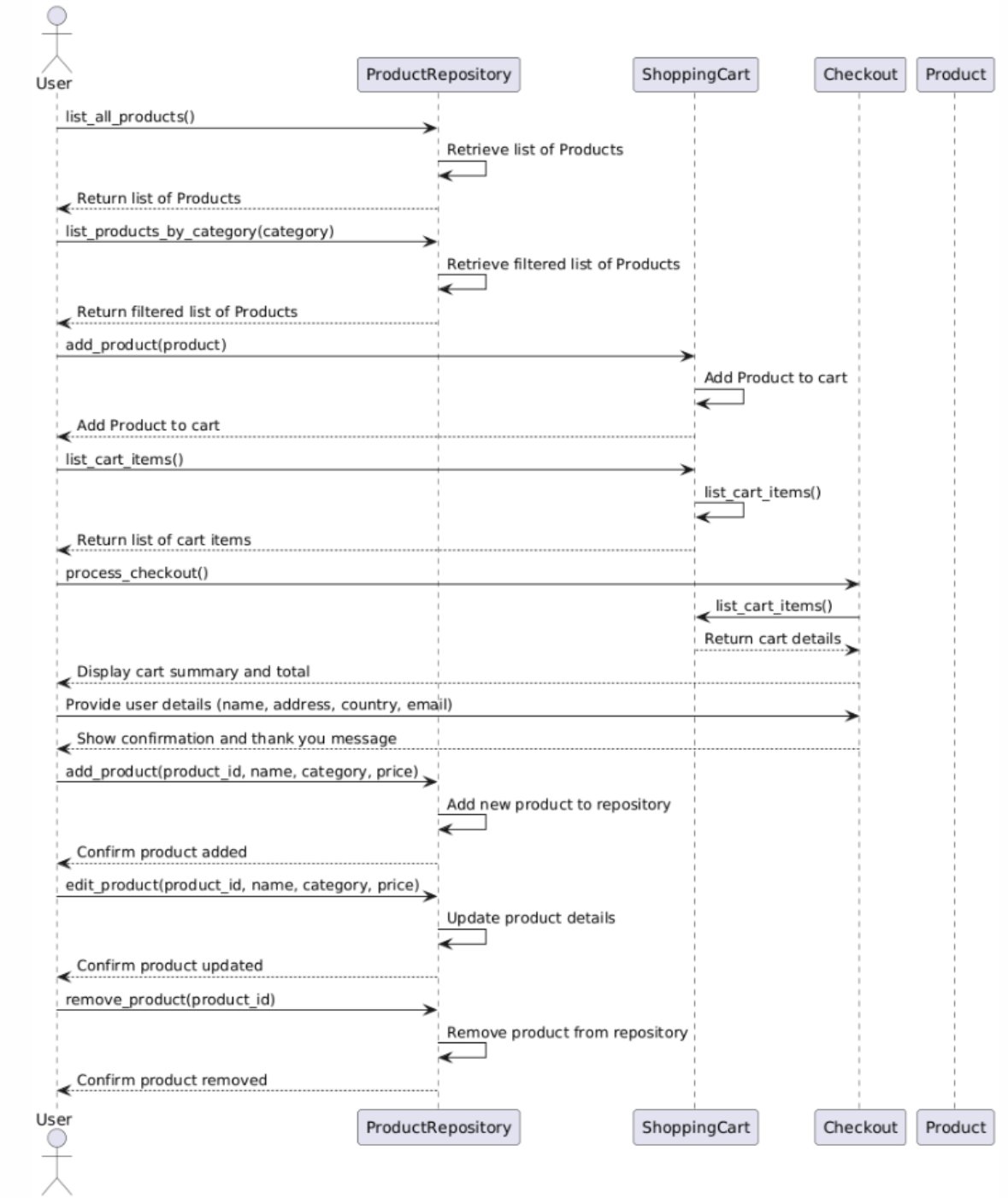
Manager

responsabilities	collaborators
Manage product data whitin the app. Add new products to the product repository. Remove existing products from the repository Edit details of existing products. Save any changes made to the modified product	ProductRepository Product AbstractProductManager

Activity Diagram:



Sequence Diagram:



Class Diagram: