OBJECT ORIENTED PRINCIPLES

ASSIGNMENT 3 REPEAT

A JAVA APPLICATION FOR MANAGING A STORE

Declaration of Authorship
I, Alex, declare that the work presented in this assignment titled 'A Java Application for Managing a Store' is my own. I confirm that:

• This work was done wholly by me as part of my BSc. (Hons) in Software Development, my Msc at Munster Technological University.

• Where I have consulted the published work and source code of others, this is always clearly attributed.

• Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this assignment source code and report is entirely my own work.

On 13/08/2025
Signature: Alex McKeever

Java Application Description.

This Java application is an interactive text-based interface that allows a coffee shop employee to manage the shop by letting the user:

- Add, edit, and remove customers from the shop.
- Add, edit, and remove products from the shop.
- Display all the products the shop has that day, including their details such as coffee strength, snack type, price, and name.
- View products that are stocked for the day
- View all orders for the day
-

Technical Difficulty: OOP Concepts Demonstrated in the Java Application.

1. **Primitive variables**

Primitive types such as int and double are used throughout the project, for example int id in Customer and double price in Product. These provide the basic building blocks for storing data

2. **Reference variables**

   Reference variables are used to link objects together. For example, the Order class contains a List<product> reference, which stores all the items in an order. This shows the difference between storing simple values and references to other objects.

3. **Classes and objects**

   Real-world entities such as Customer, Order, and Product are represented as Java classes. Each class contains relevant attributes and behaviour, and objects of these classes are instantiated in MyMain.

4. **Encapsulation**

   Fields like Name, Price and ID are declared private to restrict direct access. Getter and setter methods are used to manage these fields safely. This ensures better control over the data and how it is modified.

5. **Inheritance**

   Tea, Coffee, and Snack inherit from Product. This prevents repeating code and shows us what is known as an "is-a" relationship. For example, "Coffee is-a Product". Common fields like name, price and id are centralised in Product while the subclasses handle any unique attributes.

6. **Static Polymorphism**

   This is demonstrated through method overloading. For example, multiple methods with the same name can exist, but with different parameters. The compiler decides which version to call based on the arguments provided.

7. **Dynamic Polymorphism**

   This is demonstrated through method overriding. Subclasses such as tea, coffee, and snack override methods inherited from product.

8. **Abstract classes**

   I thought about making the product class abstract, as it would work if the shop sold products that were outside a given category. I decided against it because all products in the system must have a name, price and category.

9. **Final fields**

   A final field - "Final String storeName", was added to show immutability. Once assigned, this value cannot be changed, making sure certain information remains constant throughout the application

10. **Aggregation**

    The order class aggregates multiple product objects. An order has a list of products, but a product can exist independently of an order. This shows the real-world relationship between orders and the products they contain.

11. **Exception handling**

Input operations are wrapped in try-catch blocks to catch invalid entries, such as an email without an @. This improves robustness and prevents runtime crashes from user mistakes

12.Default constructors

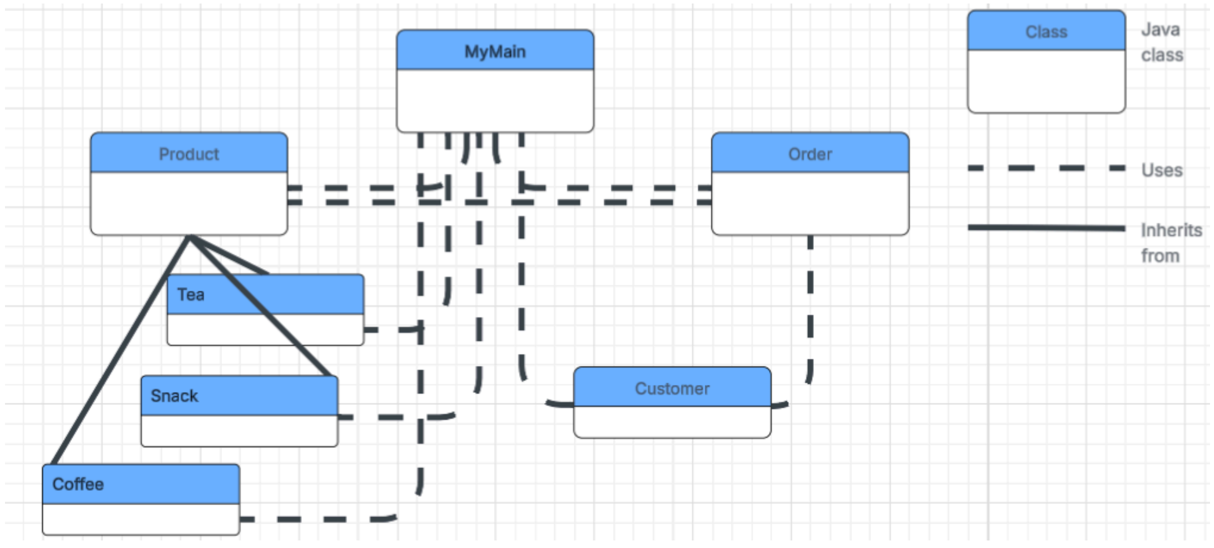
Classes such as customer and order include default constructors. These allow objects to be created when no arguments are passed, providing flexibility in object creation.

Limitations and future improvements

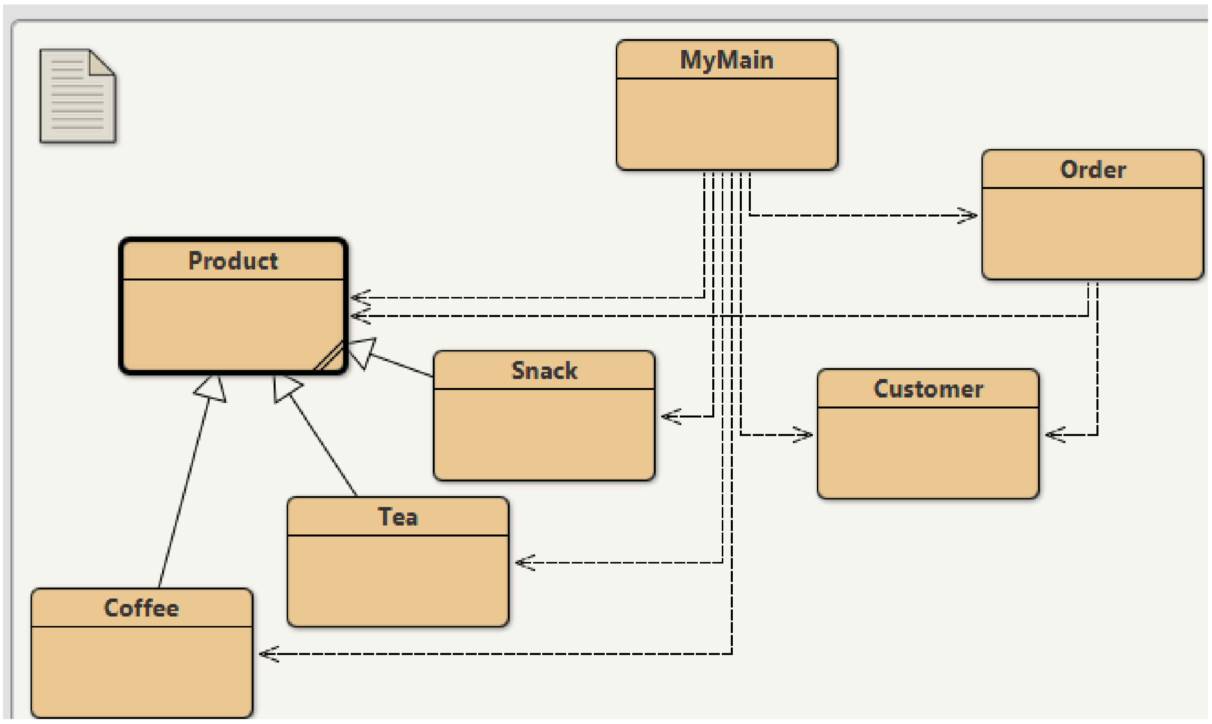
- File storage - Currently, customers are only remembered by the system for one day (One active session). A future improvement would be to save and load this data from a .txt file.
- Product variety - At the moment, the shop only sells three category of products which are tea, coffee and snacks. The system could be improved to add support for cold drinks or sandwiches.
- Scalability - While this is suitable for a small store, it could be expanded with new features such as a loyalty card feature, customer order history and product stock tracker.

UML Design: Coffee shop.

I designed my UML using Lucid chart because it had a variety of tools and designs I could use for free.



I then followed the same layout in my BlueJ application



Testing the Java Application.

The functionality of the application is tested in MyMain.java with an interactive, text-based menu session. On it, the user can select from a list of different commands to test the different functionality of the coffee shop. Making sure to not only follow the 'Happy path' is essential in testing, which is why validators are important, such as a validator to make sure a user's email address has an '@' and if it doesn't, prompt the user to try again until it does.

Assignment walkthrough video link

https://youtu.be/Jf64i32rrGQ