Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

OBJECT ORIENTED PRINCIPLES

ASSIGNMENT 3: A JAVA APPLICATION FOR MANAGING A STORE.

Declaration of Authorship

I, Alberto Ruiz, 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 19/11/2022

Signature: Alberto Ruiz

Java Application Description.

This grocery online shop is a Java application in which the user interacts with a text menu-based. The functionality of the application includes:

- Adding/removing customers and food items to the shop menu.
- Displaying customer order history and customers list.
- Creating and completing an order of a customer.
- Showing shop menu.
- For a manager to review all customers spending.
- For the shop menu to be loaded/stored to a text-based database.

Technical Difficulty: OOP Concepts Demonstrated in the Java Application.

1. **Primitive and Reference Variables.**
    - The class Customer.java has the field *currentOrder* (a Order and, therefore, reference variable). The class Food.java has the field *quantity* (an int and, therefore, primitive variable).
2. **Classes and Objects.**
    - The class Customer.java models a customer of the shop, and the case 1 of the class MyMain.java creates a new Customer object in line 156.
3. **Encapsulation.**
    - The class Menu.java has a private field menu, and public methos getMenu and setMenu methods to access/update the field from other classes.
4. **Aggregation.**
    - The class Shop.java has a private field customerList, an array of objects of type Customer.
5. **Inheritance.**
    - The class Shop.java inherit from Menu.java.
6. **Class Hierarchy.**
    - The class Shop.java inherit from Menu.java. Therefore, there is a basic class hierarchy, where Shop.java is the son, Menu.java is the parent, and there is no grandparent relationship.
7. **Static Polymorphism (overloading).**
    - The class MyMain.java has two versions of the method selectIntOption, each of them with a different signature.
8. **Dynamic Polymorphism (overwriting).**
    - The class Customer.java overwrites the method toString, specified in the class Object any Java class automatically inherits from.
9. **Abstract Class.**
    - The class Menu.java is declared abstract, as it contains an abstract method displayMenu. The method must, therefore, be overwritten by any class inheriting from Menu.java (as is the case in the class Shop.java).
10. **Interface.**

- The class ShopInt.java is an interface, modelling the management of a grocery online shop (via methods as addCustomer, removeCustomer, etc). The interface is implemented in the class Shop.java.

11. User and Developer Isolation.

- Abstract Datatypes isolate the *what* (what represents this data and what operations can we do with it) from the *how* (how is this data internally represented and how is each operation internally implemented).
- See the UML diagram on this appendix:
  - **i.** Let's assume the class MyMain.java was implemented by Programmer1. He can look at ShopInt.java and then create a variable of type ShopInt to use all its functionality (the methods addCustomer, removeCustomer, etc.), without knowing all this functionality is internally implemented. All he needs is to use the object of type Library for his own application, programmed in the methods of the class MyMain.java. In this case, his application is an interactive text menu for using a grocery online shop.
  - **ii.** So now Programmer2 implements Shop.java. He knows how to represent internally a shop (via a number of fields) and how to implement each of the methods offered. On doing so, he also implements the rest of classes (Food.java, Customer.java, etc).
    On programming the shop implementation and the rest of classes, he makes sure other programmer can create a variable of type ShopInt to use all its functionality. But Programmer2 does not know the type of application programmer1 is creating (maybe an interactive text menu, a graphic-based app, a web-based one, etc).

12. Upcasting.

- I have not included this OOP concept.

13. Static Fields.

- The class Order.java has a static field DELIVERY_FEE. Therefore, the field does not belong to a single object of the class, but to all objects of the class.

14. Final Fields, Methods and Classes.

- Following the static fields, DELIVERY_FEE is final. Therefore, as once it is defined, it cannot be modified.
- The class Menu.java has a final method getMenu, so that no other class inheriting from Menu.java (for example, Shop.java) can overwrite the method and compute the menu in a different way.

15. Data Structures.

- The class Shop.java has a field customersList, representing the list of customers of the shop. The list is represented as an ArrayList of Customer objects.

16. Java Generics.

- The class Shop.java has a field currentOrders as an ArrayList<Order> and a field customersList as an ArrayList<Customer>. The use of Java Generics allows to have lists of different types.

17. Downcasting.

- I have not included this OOP concept.

18. Exception Handling.

- The method MyMain::writeFoodMenu handles the exception of a file that cannot be found or an error trying to open the file. The instruction is placed in a try block; if something goes wrong, for example if the file is not found, then the first block catch is executed, instead of making the whole application crash.

19. File Reading and Writing.

- The method MyMain::writeFoodMenu read the content from a text file and loads it into the ArrayList<Food> menu of the shop.

20. Default Constructor and Copy Constructor.

- I have not included this OOP concept.

UML Design: Java Application.

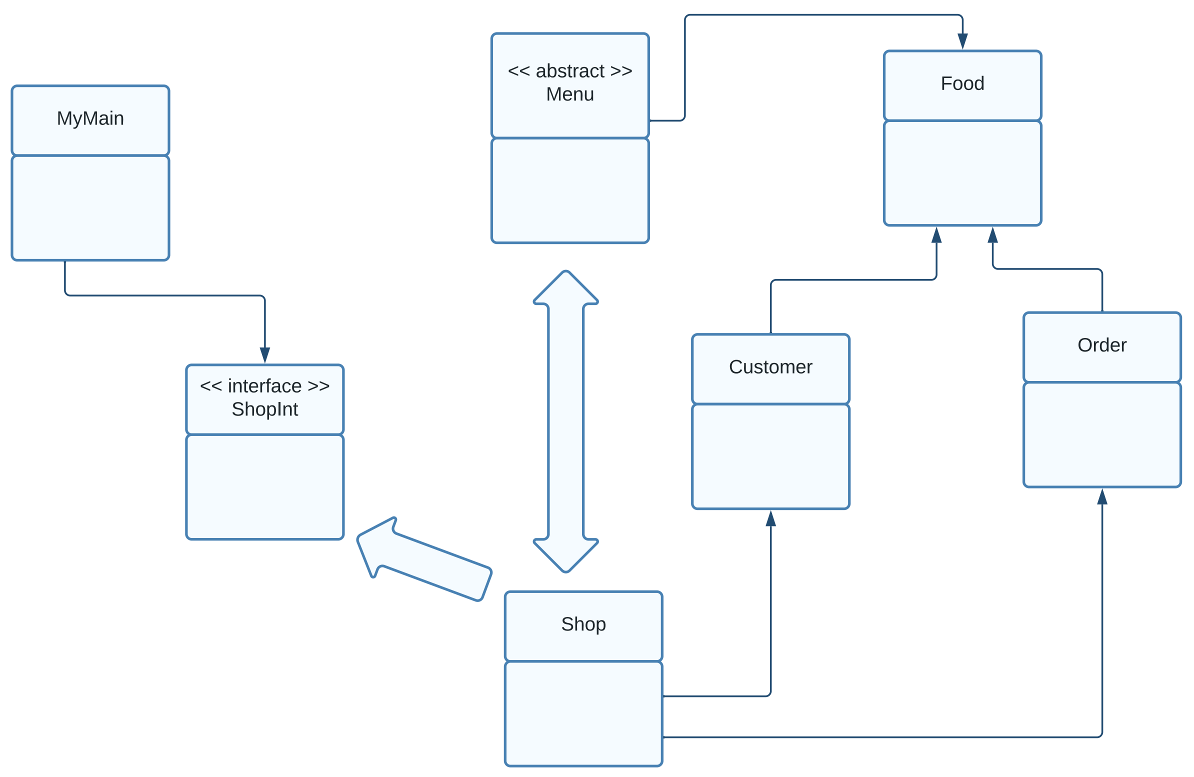
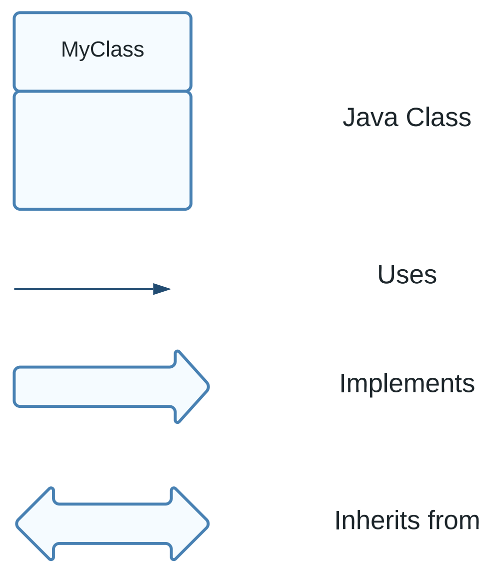


Diagram Legend



Testing the Java Application.

The functionality of the application is testes in MyMain.java via an interactive, text menu-based, session. On it, we can select among a range of different commands to test the different functionality of the library.

We could also create a number of test methods in MyMain.java like we have seen in code examples, making the main method to use an option integer variable and a switch clause to select which test method to try on each run of the Java application.