****

**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 Java Application provides an interactive, text menu-based, session for managing a grocery online shop. 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:
      1. Let’s assume the class MyMain.java was implemented by Programmer1. She 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 she needs is to use the object of type Library for her own application, programmed in the methods of the class MyMain.java. In this case, her application is an interactive text menu for using a grocery online shop.
      2. sdfsd
12. **Upcasting.**
    * Sdfsdfsdf
13. **Static Fields and Methods.**
    * Afsldjfaksjdf
14. **Final Fields, Methods and Classes.**
    * Sdfnasdfajd
15. **Data Structures.**
    * Sdkjflasdkfa
16. **Java Generics.**
    * Asdjfalksdjfas
17. **Downcasting.**
    * Asdfjkldsjfl
18. **Exception Handling.**
    * Sdfsdjlfsjdklf
19. **File Reading and Writing.**
    * Sdfsdfasdf
20. **Default Constructor and Copy Constructor.**
    * sfasdfasdfajlsdkfa

**UML Design: Java Application.**

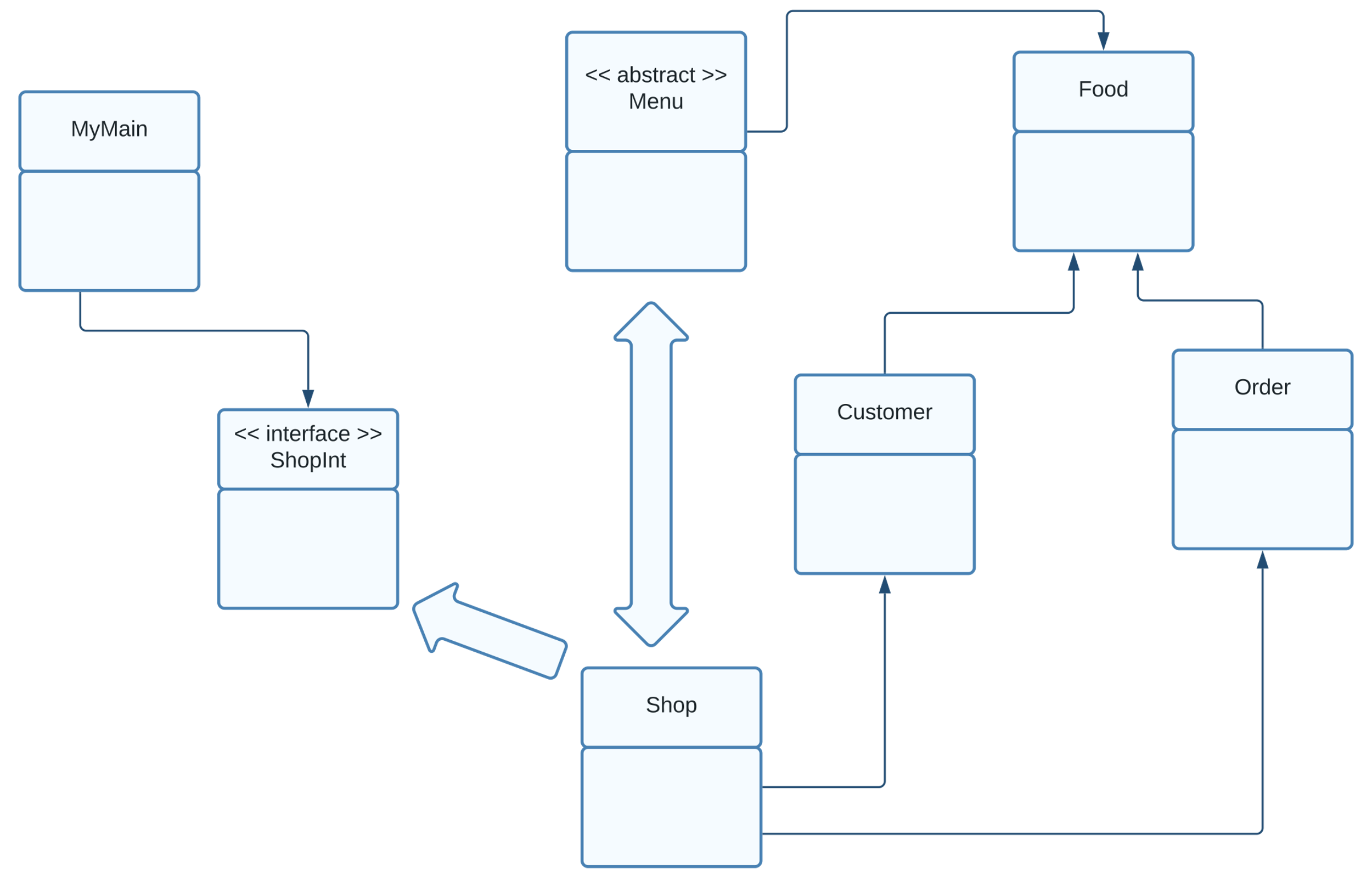
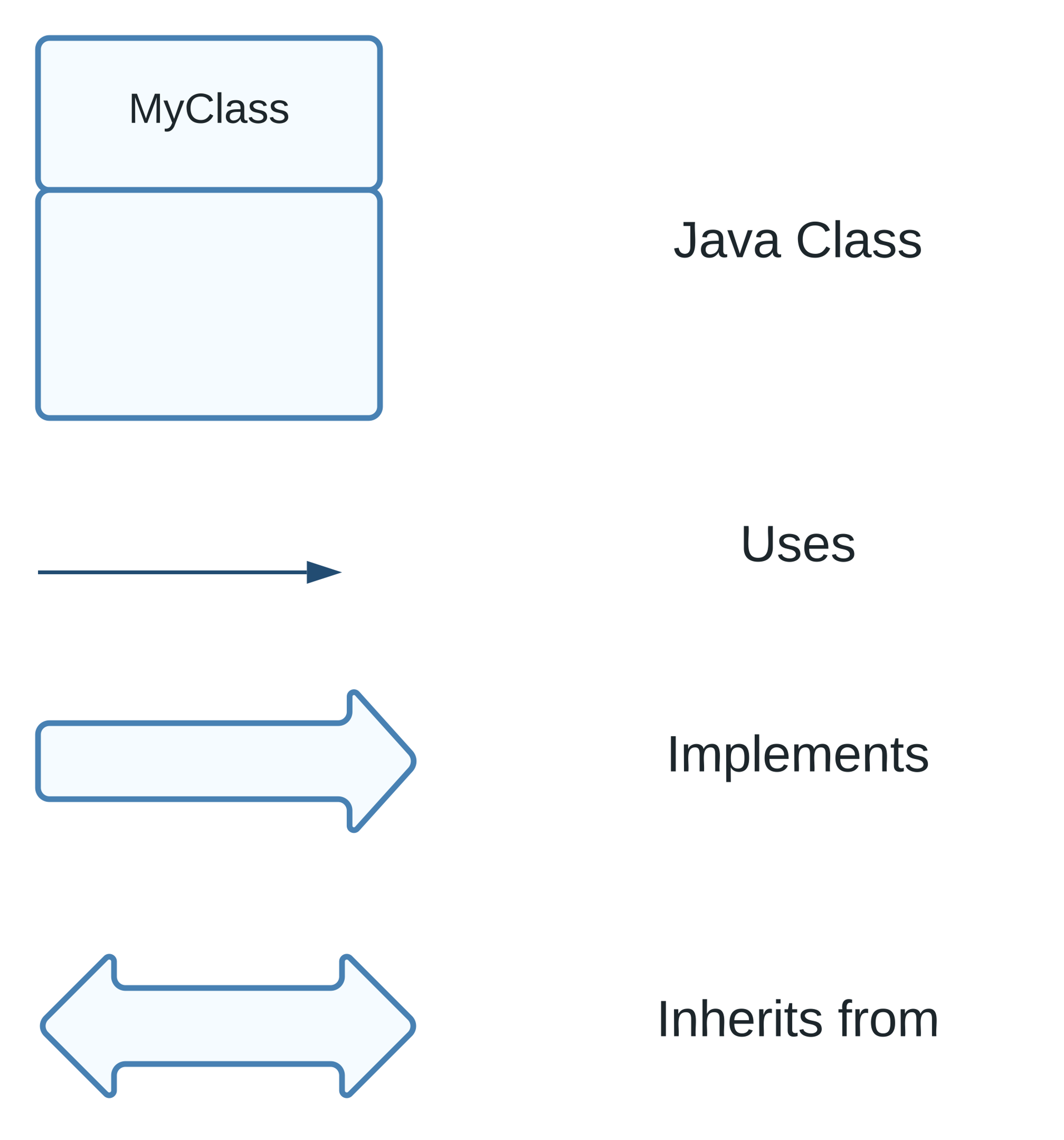
****

Diagram Legend

****

**Testing the Java Application.**

To be completed.