

**OBJECT ORIENTED PRINCIPLES**

ASSIGNMENT3 DS: A JAVA APPLICATION FOR MANAGING A COMPUTER STORE.

**Declaration of Authorship**

I, Daniel Soden, declare that the work presented in this assignment titled ‘A Java Application for Managing a Computer 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 01/12/2024

Signature: Daniel Soden

**Dependencies:**

Although Maven should for the most part ensure this code will work on any machine, with my usage of SQLite I think my video demonstration would be the best way to view the work I have put into this project. I had to do a lot of shuffling of packages to get SQLite to work for me initially so I hope it will still work for you too. In the future if I were to do this project again I would put it in a docker container so that then there would be no trouble at all with building the project and running the code.

* Maven (Used to manage packages, builds and directories rather than intellij’s standard method).
* OpenJDk23 – I.E. Java 23.
* SQLite3 – Database used for program.

**Java Application Description.**

This Java application provides an interactive, text menu-based, session for managing and purchasing computer products. The functionality of the application includes:

* Adding/removing users from the shop.
* Displaying the info of a user/computer.
* For users to purchase products from the Shops;
* For the user/store content to be loaded/stored to a SQLite3 database. – Resource for sqlite (<https://www.sqlitetutorial.net/sqlite-java/>) - Main usage of resource was for the boilerplate before any SQL code is made, after the boilerplate I declare all SQL code is my own. On top of this for demonstration purposes, I have filled in some SQL data manually via the SQLite CLI

**Technical Difficulty: OOP Concepts Demonstrated in the Java Application.**

1. **Primitive and Reference Variables.**
   * The class Order,java has the fields productID and customerID (an int and, therefore, primitive variable) and these fields (a Borrow and, therefore, reference variable).
2. **Classes and Objects.**
   * The class Customer.java models a user of the library, and the method ShopImp.java::addCustomer creates a new Customer object Customer.
3. **Encapsulation.**
   * The class Computer.java has a public field gbRam, and public methods getRam and setRam methods to access/update the field from other classes.
4. **Inheritance.**
   * The classes Product.java inherits from Agent.java
5. **Class Hierarchy.**
   * The class Computer.java inherits from Product.java. Therefore, there is a class hierarchy, where computer is a child class of the parent Product.
6. **Static Polymorphism (overloading).**
   * The class MyMain.java has two versions of the method selectIntOption, each of them with a different signature.
7. **Dynamic Polymorphism (overwriting).**
   * The classes User.java and Item.java overwrite the method toString, specified in the class Object any Java class automatically inherits from.
8. **Interface.**
   * The class Shop.java is an interface, modelling the management of a computer Shop (via methods as purchaseItem, saveReceipt, etc). The interface is implemented in the class ShopImp.java.
9. **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:
     + Let’s assume the class MyMain.java was implemented by Programmer1. She can look at Library.java and then create a variable of type Library to use all its functionality (the methods addUser, removeUser, etc.), without knowing how all this functionality is internally implemented.

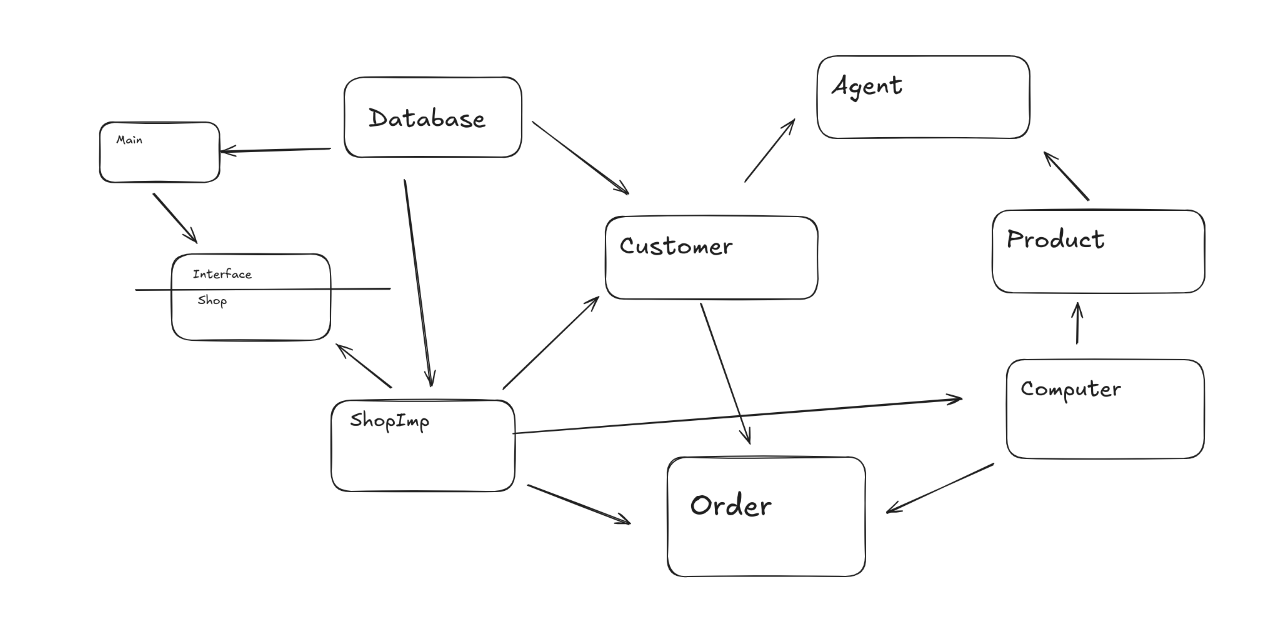
All she needs is to use the object of type Library for her own application, programmed in the methods of the class MyMain.java. In this case, her application is an interactive text menu for using a library.

* + - Let’s assume the class LibraryImp.java was implemented by Programmer2. She knows how to represent internally a library (via a number of fields) and how to implement each of the methods offered. On doing so, she also implements the rest of classes (Agent.java, User.java, etc.)

On programming the library implementation and the rest of classes, she makes sure other programmer can create a variable of type Library 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).

1. **Static Fields and Methods.**
   * The class Database.java has a static field url. Therefore, the field does not belong to a single object of the class, but to all objects of the class.This decision was made so I could access this field universally for database connectivity.
   * The class User.java has a method isUserInUsersList, to compute whether any of the users of a list contains a concrete id. As a public static method, it can be called from any class without the need of a concrete User object, just by using the prefix User.isUserInUsersList.
2. **Final Fields, Methods and Classes.**
   * The class User.java has a final field name, as once it is defined, it cannot be modified.
   * The class Agent.java has a final method getId, so that no other class inheriting from Agent.java (for example, User.java or Item.java) can overwrite the method and compute the id in a different way.
3. **Exception Handling.**
   * The methods relating to the Sqlite connection all contain exception handling in that if it is unable to reach the Structured query language data, it will give back an sql, like exception.
4. **File Reading and Writing.**
   * Although not directly file reading, throughout the course of my application, espeically for listItems and listRecentOrders we are reading from the shop.db database file.
   * The method saveReceipt takes in an order O and from here is called within purchaseItem as an optional method to save the contents of their order to a txt file receipt**.**

**UML Design: Java Application.**



**Testing the Java Application.**

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

My application is completely based off text inputs as seen by the scan(<GivenDatatype>) Methods in the main, ensuring the user gives both the correct length and that they give a variable of some sort.

**Overall I am quite dissapointed in myself as although all the code I have written has some logical backing to it, between the lack of javadoc comments, overall functionality due to simple errors and being overly focused on implementing sqlite (not required) I put myself in a very difficult position to adhere also to the OOP principles.**