**A picture containing text

Description automatically generated**

**OBJECT ORIENTED PRINCIPLES**

ASSIGNMENT 3: A JAVA APPLICATION FOR MANAGING A STORE.

**Declaration of Authorship**

I, \_\_\_LAIBA ASIF\_\_\_, 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 \_\_\_06\12\2022\_\_

Signature: laiba asif

1. **Java Application Description.**

This Java application provides an interactive, text menu-based, session for managing a drugstore. The functionality of the application includes:

• Adding customer and items to the store (in the case of the items, both hair electronics and epilators are supported);

• Displaying the info of a customer/item;

• For customers to buy hair electronics and hair removing electronics

• For a displaying all data in database a product ID is required;

• For the store products to be loaded/stored to a text-based database.

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

* Primitive Variables.

The class HairElectronic.java has the field power (an int and, therefore, primitive variable).

* Classes and Objects.

The class Customer.java models a customer of the drugstore, and the method MyMain.java ::createCustomer creates a new customer object newcustomer in line 77.

* Encapsulation.

The class Epilator.java has a private field waterproof, and public methods getwaterproof and setwaterproof methods to access/update the field from other classes

* Inheritance.

The classes Epilator.java and HairElectronic.java inherit from Product.java.

* Class Hierarchy.

The classes Epilator.java and HairElectronic.java inherit from Product.java. Therefore, there is a class hierarchy, where Customer.java and Order.java are siblings, and Epilator.java is a grandchildren of Product.java.

* Static Polymorphism (overloading).

The class MyMain.java has two versions of the method selectIntOption, each of them with a different signature.

* Dynamic Polymorphism (overwriting).

The classes Customer.java and Product.java overwrite the method toString, specified in the class Object any Java class automatically inherits from.

* Abstract Class.

The class Product.java is declared abstract, as it contains an abstract method price. The method must, therefore, be overwritten by any class inheriting from Product.java (as is the case in the classes HairElectronicjava and Epilator.java)

* 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/he can look at Epilator.java and then create a variable of type createNewMachine to use all its functionality (the methods make, model, etc.), without knowing how all this functionality is internally implemented. All she/he needs is to use the object of type createNewMachine for her own application, programmed in the methods of the class MyMain.java. In this case, her/his application is an interactive text menu for using a CreateNewMachine.

Let’s assume the class HairElectronic.java was implemented by Programmer2. She /heknows how to represent internally a createnewmachine (via a number of fields) and how to implement each of the methods offered. On doing so, she/he also implements the rest of classes (Customer.java, etc.)

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

* Upcasting.

The method product;; crate product creates the variable Item new product in lines. The variable is there assigned to a newly created HairElectronic object or to a newly created Epilator object.

* Static Fields and Methods.

The class Product.java has a static field generateId. Therefore, the field does not belong to a single object of the class, but to all objects of the class. Indeed, if I were to program the Java application again, I would put this field in the class Order.java, where it definitely fits better.

* Final Fields, Methods and Classes.

The class Customer.java has a final field name, as once it is defined, it cannot be modified.

The class Product.java has a final method getId, so that no other class inheriting from Product.java (for example, Customer.java or Order.java) can overwrite the method and compute the id in a different way.

* Exception Handling.

The method MyMain::selectIntOption handles the exception of a user inputting by keyboard a value that is not an integer (as expected by sc.nextInt ). The instruction is placed in a try block; if something goes wrong, for example if the user enters instea a String value, then the block catch is executed, instead of making the whole application crash.

* Default Constructor and Copy Constructor.

At the time of writing this appendix, I realise I have not included a copy constructor or default constructor in the Java application, but this is something that could have been added also :(

1. **UML Design: Java Application.**

**Diagram

Description automatically generated**

**Diagram

Description automatically generated**

**Diagram

Description automatically generated**

1. **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 commands to test the different functionality of the drugstore. Of course, an alternative way of testing the application would be that is, to create a number of test methods in MyMain.java, and make 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.