Systems Programming

Part 1:
Object-oriented programming + input/output (I/O) + testing

1. Introduction

A leading logistics company has asked you to develop a program for managing a warehouse. The development of this program will be done in two phases:

- During the first phase (Module-1) the programming department will work on modelling the different elements involved in the management of the warehouse and on designing the text menus that will serve to interact with the program.
- During the second phase (Module-2) the programming department will work on the logic of the application and on providing functionality to the menus created during the first phase.

2. Class Modelling

The elements involved in the management of a warehouse are described below. Next to each concept we indicate the name in English that you should use as the name of the classes, attributes or methods of your program.

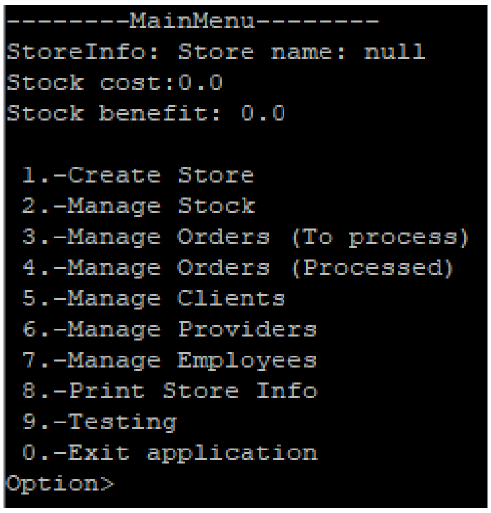
- Product (`Product`). A type of good that can be distributed from the warehouse. It has a name (`name`), brand (`brand`) , category (`category`) which can be identified by one of these four letters: f-FOOD, s-SUPPLIES, e-EQUIPMENTS, m-MISCELLANY, a variable indicating whether the product to be stored is countable or uncountable (`isCountable`), and a string of characters indicating the type of units in which the product is measured (e.g., “kg”, “liters”, “pieces”, etc.) (`measurementUnit`).
- Stockable Product (`StockableProduct`). It is a specialization of the previous concept. It is a product that is already prepared for storage in the stock of the warehouse, so it first contains a numerical identifier assigned by the warehouse automatically and incrementally (`productID`), followed by all the information from any normal product (`name, brand, category, isCountable, measurementUnity`), and finally additional information consisting of: an integer number indicating the number of units of that product available in the warehouse (`numUnits`) , and two decimal numbers to indicate the cost of manufacture of the product in euros (`costPerUnit`) and the selling price also in euros (`pricePerUnit`).
- Product list (`ProductList`). It is a data structure that in addition to a list of stockable products that is declared as `ArrayList<StockableProduct> list`[1], stores additional information on all these products, such as their total cost (`totalCost`), the total selling price (`totalPrice`), or the total benefit obtained from its distribution in the market (total selling price minus total cost) (`totalBenefit`). All these values take into account not only the different products on the list but also the number of units available for each of them.
- Provider (`Provider`). It is the company in charge of supplying a product to the warehouse. It contains the company's tax identification number (`vat`), its name (`name`), the billing address (`taxAddress`) and a contact person (`contactPerson`).

[1] ArrayList<E> is a data structure in Java that is already implemented in the Java libraries (and as it happens with the String class and the Object class includes its own methods). Unlike the “conventional” arrays which have a fixed size, an ArrayList<E> has the advantage that it can grow dynamically (that is to say its size increases automatically as we add elements). Your teacher will give you more information about this data structure when you need to use it.

- Person (`Person`). It refers to any relevant person for the management of a warehouse. It can be a customer of the warehouse, the employee that manages the order, or the contact person for a provider. To identify a person, we need his/her identification number (`id`), first name (`firstName`), last name (`lastName`) and contact email (`email`). At the moment it is not necessary to consider the several possible roles (client/employee/contact person) for the development of the application.
- Order (`Order`). It represents the document (purchase order) that contains each of the orders that are issued in the warehouse. Each purchase is in reality a specialization of a product list (`ProductList`) that also includes an order identifier which is assigned automatically and incrementally every time a new order is created (`orderID`), followed by all information included in any product list (`list`, `totalCost`, `totalPrice`, `totalBenefit`), and a reference to two people, the client who places the order (`client`) and on the other the employee who prepares it (`employee`).
- Store Manager (`StoreManager`). It is the brain of the application and will contain all the logic of the program. At the moment it is only necessary to add the name of the store (`name`), the total cost of the products stored in the store (`stockCost`) and the benefit obtained from the sale (`stockBenefit`), as well as a list (`ProductList`) called `stock` that contains all the products of the warehouse.

3. Menu

When the main method of the `RunApp` class is executed, a menu will be displayed. The following Figure describes the main screen of your program (Figure-1) with this menu. This screen contains general information about the warehouse (its name, the total cost of the Stock of stored products and the benefit that would be obtained by the sale of all of them) and the main menu of the application.

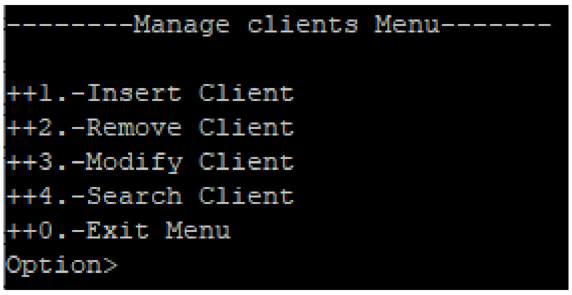


```
--------MainMenu--------
StoreInfo: Store name: null
Stock cost:0.0
Stock benefit: 0.0

 1.-Create Store
 2.-Manage Stock
 3.-Manage Orders (To process)
 4.-Manage Orders (Processed)
 5.-Manage Clients
 6.-Manage Providers
 7.-Manage Employees
 8.-Print Store Info
 9.-Testing
 0.-Exit application
Option>
```

Figure-1: Main menu of the program.

For each of the options that allows managing an element (Options 2 to 7) you must include a new menu that allows you to insert, remove, modify or search for an element within the data structure. You can see an example of a second level menu for option “5 Manage Clients” in Figure-2.



```
--------Manage clients Menu-------

++1.-Insert Client
++2.-Remove Client
++3.-Modify Client
++4.-Search Client
++0.-Exit Menu
Option>
```

Figure-2: Example of Second Level Menu. The ++ symbols in front of each option indicate that it is a second level menu.

4. Work to be done

Class Modelling

The work to be done for module-1 consists of modelling each of the elements of a warehouse, as described in the previous section. To do so, you will have to create a class for each element together with its attributes, taking into account the possible inheritance or aggregation relationships between them. To add functionality to these elements all the classes must also include the following methods:

- Two constructors
    - An empty constructor.
    - A constructor that receives as parameters all those attributes of the class (that are not automatically calculated) in the order indicated in the description above.
- Access/Modification methods `(get/set)` for each of the attributes to be accessed/modified from other classes.
    - NOTE: For those attributes that have some restriction on the values they can take, the modification method (`set`) must check that the value received as parameter meets these restrictions, before storing the value in the corresponding attribute. For example, in the case of the class Product, the product category must be one of the four possible letters (`'f', 's', 'e', 'm'`).
- An additional modification method (`set`) method `public void set(String[] data)` per element. This method receives as a parameter an array of strings with each of the object attributes, converts these strings to the appropriate data type and calls the corresponding set method to store the values in the attributes. This method will be very useful when reading information about the object from a file.
- A `toString` method which overrides the `public String toString()` method from class Object. This method must return a String with all the attributes in the indicated order separated by the character "|" (in Java you must indicate it as `"\\|"`). For example, the method toString of the class Person would return the string `"dni|firstName|LastName|email|"`
- Two methods to write that allow writing the content of the objects in the format indicated by the method `toString.` NOTE: It is not necessary to add these methods in the `StoreManager` class.
    - To write in the standard output (i.e. on the screen): `public void print()`
    - To write in a file: `public void writeToFile()`
- Two method to read that return an object of the type indicated by the class name (e.g., the methods to read for the `Product` class would return a `Product` type object, the methods to read for the Person class would return a `Person` type object, etc.). NOTE: It is not necessary to add these methods in the `StoreManager` class.
    - To read the information about the attributes from the standard input (i.e. keyboard): `public Person readFromStdio()`
    - To read the information of the attributes from a file: `public Person readFromFile(String File)` (the file will contain one line for each object and the lines will have the format indicated in the toString() method of said object).
- NOTE: Your teacher will provide you with additional information on how to read from files and write on files in Java (Input/Output)
- The product list class (`ProductList`) must also contain the following additional methods:
    - `public double calculateCost()` to calculate the total cost of purchasing all the products stored on the list.
    - `public double calculatePrice()` to calculate the total selling price of all products stored on the list.
    - `public double calculateBenefit()` to calculate the benefit (price-cost) from the sale of all products stored on the list.
    - `public StockableProduct mostExpensiveProduct()` to return the most expensive product stored on the list.
    - `public StockableProduct cheapestProduct()` to return the cheapest product stored on the list.

Menu

In this first phase the programming department must implement:

- The code that shows the main menu when starting the application.
- The code that shows the additional menus when options 2 to 7 are selected.
- The code that prints the general information of the warehouse in the format shown in Figure-1 above the main menu. The corresponding method/s will be executed when starting the program and every time the option "8" is selected from the main menu and will show the value of three of the main attributes of the `StoreManager` class: `name, stockCost and stockBenefit`.
- The code that allows exiting the menu (Option-0)
- For the rest of the unimplemented options of each of the menus and submenus you do not need to program the code yet, just create a method for each option that prints on screen what that option would do. For example, selecting option 1 in the "Manage Clients Menu" (see figure-2) would call the following method:

```
public static void insertClientMenu(){
  System.out.println ("Inserting new client");
}
```

- During the second phase of the project (Module-2) we will program some of these functionalities.
- Always verify the correct behavior of each class by performing the corresponding unit tests. Your teacher will provide you some tests as an example during Testing Lecture.