INTRODUCTION

In your role as a software developer, you have been contracted by QUB's Retail Department to create a Java application capable of managing a range of products sourced from a warehouse. Specifically, you are required to build and test a prototype system that meets the deliverables outlined in the sections below.

GENERAL INSTRUCTIONS - CODE IMPLEMENTATION

1. Create a new Java project named 'Assessment2'.
2. Add three packages to 'Assessment2' named part01, part02 and part03.
3. Implement your code as described below:
    - 'part01' should contain all code associated with the core requirements.
    - 'part03' should contain any code relating to testing.
    - 'part03' should contain the code related to implementation of additional features only, the only exception being the console application which you should copy from part01 to part02. Use import statements as required to access the definitions from part01.

SUBMISSION DATE/TIME – FRIDAY 25TH MARCH 2022 BY 5 PM.

PART 1: CORE FUNCTIONALITY (50 MARKS AVAILABLE)

To demonstrate that the proposed Java application is feasible, the following features must be implemented.

1.1 Implementation of the object classes, enumerator and interface shown in Figure 1 below.

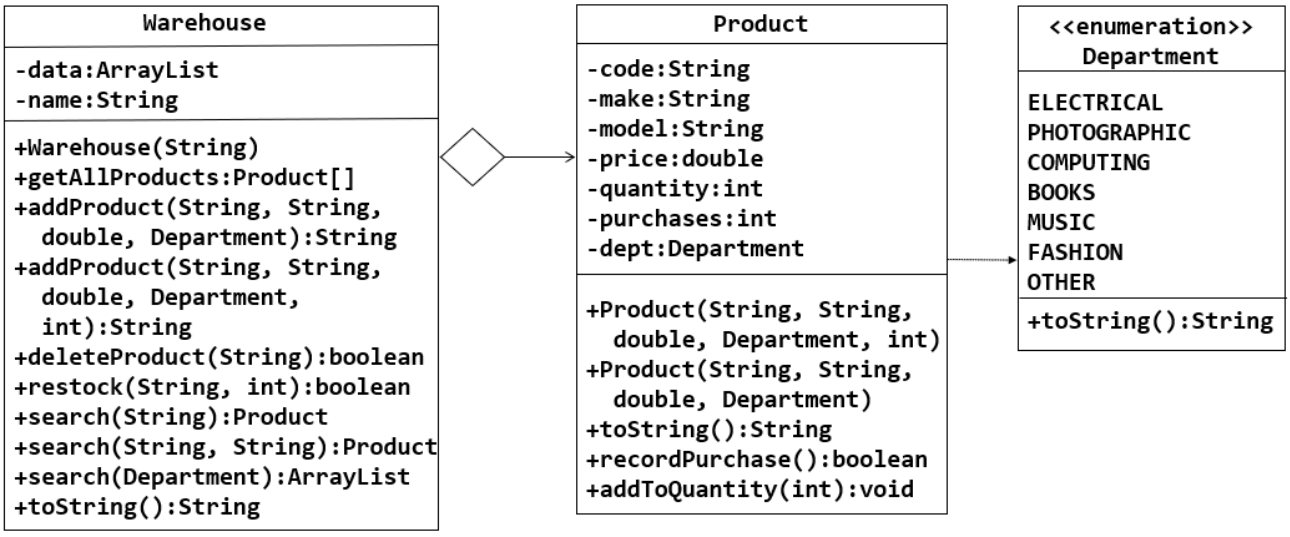


Figure 1: A (partial) UML Class Diagram for QUB Retail Software.

Further details on the role and behaviour of the above classes follows – please read the entire document before starting. Canvas will also contain a list of frequently asked questions (FAQ) – a useful resource for clarification purposes.

Interfaces for ***Warehouse*** and ***Product*** are available on Canvas to help you structure these classes. These will be discussed in class and must be used.

1.2 A standard (Eclipse console) menu-based application (called ***WarehouseApp***) to manage (through a ***Warehouse*** instance) the interaction with ***Product*** instances within the system. The following menu options should be provided:

- ***Add New Product***: to enable addition of a new product to the Warehouse
- ***Delete a Product***: to delete a specific product from the warehouse
- ***List Products for Department***: displays product details for a specific department
- ***List All Products***: displays details for products under each department
- ***Add Stock for a Product***: to update stock quantities for a specific product
- ***Sales Report***: to display sales information for all products, in order of number of purchases
- ***Record Purchase***: to record purchases (quantity of 1 or more) for a specific product
- ***Exit***: the user should be able to exit the system.

Notes:

A. The console application should ensure that products are added (by default) to the ***Warehouse*** instance for testing purposes. You should include at least 2 per department.

B. For enumeration ***Department***, the ***toString***() method should return a String corresponding to the current value:

Value	***String***
ELECTRICAL	"Electrical"
PHOTOGRAPHIC	"Photographic"
COMPUTING	"Computing"
BOOKS	"Books"
MUSIC	"Music"
FASHION	"Fashion"
OTHER	"Other"

C. You should make use of the ***Menu*** class defined in Menu.java (from the assessment specification on Canvas).

D. ***No external classes/libraries*** other than ***Scanner*** or ***ArrayList*** may be used. When using ***ArrayList***, you are restricted to the ***get***, ***add***, ***remove*** and ***size*** methods only.

E. Searching and sorting (where required) should be coded as described in the lectures (use of Collections class is not permitted)

F. You ***may not*** add additional ***public*** methods or attributes to ***Warehouse***. You may add ***private*** attributes and methods to any class. You may add ***getters*** to class ***Product***, but only as specified in the interface ***iPro***. You may add ***setters*** to ***Product*** however these must be justified in comments.

PART 2: TESTING THE APPLICATION (20 MARKS)

As part of QUBs quality assurance procedure, you are required to submit evidence that the console application implemented in Part 1 has been thoroughly tested. This should take the form of a completed testing document together with code covering ***unit and integration testing only***. Testing documentation must be submitted using the template provided by QUB, available on Canvas.

PART 3: ADDITIONAL FEATURES (30 MARKS AVAILABLE)

Further to the features outlined above, QUB's Retail Department is interested in evaluating any additional functionality that may be of benefit to the application. Accordingly, within the time and resource constraints of the project, you have been asked to investigate and (where possible) implement additional functionality.

This will involve the use of an alternative Console class to display information and read keyboard input. Details on how to use this class will be covered in forthcoming lectures with supporting material on Canvas.

A – Defining a New Menu Class

1. Download the java archive (console.jar file) which enables the use of a new ***Console*** class
2. Include this archive in your project (to be discussed in lectures)
3. Extend the ***Console*** class to implement a class called ***ConsoleMenu***. This class should offer the same behaviour/functionality provided in ***Menu.java*** (use of Capabilities of class Console should be carefully considered here for Menu presentation)

B – Updating the Warehouse Application

1. Extend the ***Product*** class to implement a class called ***ProductDetail***.
2. This class should include 2 additional attributes:
    (i) ***description (a String)*** – a text description for a product
    (ii) ***image (ImageIcon)*** – an image associated with a product
3. The constructor for this class should initialise ***description*** and ***image*** using supplied parameters (in that order)
4. Add getters for these attributes.
5. Copy ***WarehouseApp*** from part01 to part03.
6. Update ***WarehouseApp*** to use ***ConsoleMenu*** and ***ProductDetail***. You should also make direct use of the ***Console*** class when displaying product information.

NOTE:

- **Each piece of additional functionality in line with those described above will be awarded marks based on the quality/creativity of the functionality provided.**

CLASS INFORMATION

Product Attributes and Methods:

code – must be unique for a Product and formatted as follows:

Letter + Number

Where **Letter** is the first character of the associated department ('E', 'P', 'C', 'B', 'M', 'F', 'O').
and **Number** is an integer, starting at 1 and 4 characters in length, prefilled with 0

The following are examples of valid codes:

C0001 – a product in department Computing
E0002 – a product in department Electrical
P0003 – a product in department Photographic

Note: the ***code*** attribute should be ***automatically generated***

make – the product manufacturer
model – the product model
price – the product price in £
quantity – defines amount of a product available
purchases – defines the number of purchases for a product
dept – associated product department

constructors – overloaded to allow initialisation of products with or without initial quantity value. Constructor parameters are:

+product(String make, String model, double price, Department dept)
+product(String make, String model, double price, Department dept, int quantity)

toString() – should return a String containing details of a product on a ***single line***
recordPurchase() – to record a single purchase of a product (if available) returning a boolean denoting success/failure
addToQuantity() – parameter defines addition to product quantity

Warehouse Attributes and Methods:

data – an array list containing all product references
name – the warehouse owner
constructor – parameter defines the owner name
getAllProducts() – returns an array of all product references, or null if no products in the warehouse
addProduct() – overloaded to allow for addition of products with or without specifying the initial quantity available.
Parameters (in order) are: make, model, price, department and quantity (if applicable)
Returns a String – "OK" for success or an error message (reason for failure)
Must ensure that there are no 2 products with same make and model.
deleteProduct() – parameter specifies code for product to be deleted. Returns boolean denoting success/failure.
restock() – parameters (in order) are product code and quantity. Returns boolean denoting success/failure.
search() – overloaded to allow for search by product code, search by make & model, search by department.
toString() – should return a formatted string details all products within a warehouse and organised (grouped) by department.

Console Methods:

constructor – takes a single boolean parameter: true (console supports user input), false (input not supported)

clear() – will clear the console of text and images

println() – overloaded to allow for output of text, images (ImageIcon) and other objects.

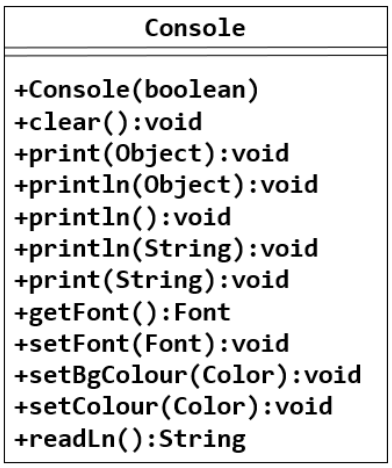
getFont() – returns the current Font in use by the console.

setFont() – parameter defines the Font to be used by the console.

setBgColour() – parameter defines the colour used for the console background.

setColour() – parameter defines the colour used within the console for text.

readLn() – returns a String in response to user tying text within the console (note that the console must be initialised to permit this).



SUBMISSION INSTRUCTIONS

1. Create a new folder to hold all your assignment files for submission. The folder should be suitably named and include your student number.
2. Copy and paste the 'part01', 'part02' and 'part03' folders from your java project into the folder created in step 1. ENSURE that the 'part01', 'part02' and 'part03' folders contain all the corresponding .java files.
3. For part03, copy your Images folder from your java project folder created in step 1.
4. Add your testing spreadsheet to the folder created in step 1. Make sure to include your test code (for Part 2)
5. Verify that you have all the required files included in the folder created in step 1.
6. Compress the folder to a zip file and upload it to the assignment location in the CSC1029 module on Canvas by **5.00pm** on **Friday 25th March 2022.**

NOTE: Please check that you have included the original development Java (.java) files. If you submit .class files these will NOT be marked. These submissions will be date-stamped and in accordance with University regulations, late submissions will be penalised. **Failure to follow the submission instructions may result in a reduced or zero mark for part or all of the assignment!**