OOP Programming Assignment
January 2024

1. General Instructions

The general guidelines for this assignment are as follows:

1. Evidence of plagiarism or collusion will be taken seriously, and University regulations will be applied fully to such cases, in addition to ZERO marks being awarded to all parties involved.
2. The total mark for the assignment is 100 marks and contributes 20% to the total grade.
3. This **group assignment is allowed for 4 persons**.
4. The deadline for assignment submission is **28 April 2024, Thursday** before **11:55 pm**.
5. Each group must submit ONE (1) **complete source code (src) with UML diagram (word or pdf)** and ONE (1) **recording video (10 mins max)** to explain each classes function and the output of your application. Please **compress all the files** and rename with your name, student id and program. Marks will be allocated based on your contribution. (Refer to section 3.4)
6. Your file must be submitted to the following platform before the **due time/date**.
   Send your answer script to Email Account:

   i. CN students please send your assignment to:
   UCCD2044FACN@gmail.com

   ii. CS students please send your assignment to:
   UCCD2044FACS@gmail.com

   iii. IA students please send your assignment to:
   UCCD2044FAIA@gmail.com

   iv. IB students please send your assignment to:
   UCCD2044FAIB@gmail.com

Note: Use your "1UTAR" email account to submit your assignment. For the title of your email, please use the file name of your compress file name_ Assignment. That is,

GroupXX_CS_Assignment

2. Background

In this assignment, your team is asked to develop a stock management system (SMS) in Java using Eclipse IDE and object-oriented programming concepts. The main objective of the SMS is to keep track and manage stock on a continual basis for a shop that sells two products (refrigerators and TVs).

3. Requirements

3.1 Program Functions (25 marks)

Your program must fulfill the following functional specifications:

- Display the Welcome to the SMS, the current date and time and names of group member (in alphabetical order).
- Allow user to input their full name.
- Check if a user would like to add any products.
    - If user wish to add products, the program should prompt and require user to enter a number to add the product(s). Then, the program should display a menu option contains all the product selections (Refrigerator and TV). Provide intuitive input method that allows users to select a product, and then prompt the user to enter the values of the variables and use them to create an instance of the Product class so that user can add new entry.
    - If user do not wish to add any products, the program should prompt a request and require user to enter a zero value to exit the program.
- Display the menu option whether to view products, add stock, deduct stock, discontinue a product, or do nothing (Exit) after all the products were added to the system. Check if a user is making valid selection. Execute the appropriate methods that you have defined in the `StockManagement` class (whether to view products, add stock, deduct stock, or discontinue product) based on user selection.
- Display the generated user ID and their name when user exits the system.

3.2 Object-Oriented Design (40 marks)

You must use object-oriented programming concepts in your implementation. Your design should follow good object-oriented design principles such as:

- Single responsibility – a class should have responsibility for a single functionality of a program and that responsibility should be encapsulated by the class.
- Open/closed principle – classes or modules should be open for extension but closed for modification.
- Efficient and no redundancy – keep it simple.

The followings are the basic classes that **MUST** exist in your program. The responsibilities of each of the classes, as well as some of their properties and methods are suggested. You are **free**

to select the **data types** and **method signatures** for the classes, and to **add** additional classes if necessary.

Note: Follow the naming conventions and use descriptive identifiers to name classes, methods, variables, and constants to make your programs easy to read and avoid errors.

Product:

`Product` is an abstract class. The class contains data fields (instance variables) to store the name of a product, product price, quantity available in stock, item number for identifying the product in the store, and status of the product (the default value is true, it means that the product is active).

Include a default constructor and a parameterized constructor that takes in four parameters that are the item number, name of a product, quantity available in stock and product price.

Include the getter/accessor and setter/mutator methods for each of the data fields.

The class should also have the following **instance methods** to:

- Get the total inventory value that is product price multiplied by the quantity available in stock.
- Allow user to add and deduct the number of quantity available in stock. Both methods take quantity as parameter and have no return value. **Note:** Do not allow the user to add stock to a discontinued product line.

Override the `toString()` method to return the information of the `Product` object as follows:

```
Item number        :
Product name       :
Quantity available:
Price (RM)         :
Inventory value (RM):
Product status     :
```

Refrigerator:

`Refrigerator` is a subclass of the `Product` class. It contains additional instance variables to store the door design, color and capacity.

Create a parameterized constructor that accepts values for each instance variables for the `Refrigerator` and `Product` classes.

Add the getter/accessor and setter/mutator methods for the three instance variables.

A method to calculate the value of the stock of a refrigerator.

Override the `toString()` method to return the information of the `Refrigerator` object as follows:

```
Item number        :
Product name       :
Door design        :
Color              :
Capacity (in Litres):
Quantity available:
Price (RM)         :
Inventory value (RM):
Product status     :
```

TV:

`TV` is a subclass of the `Product` class. It contains additional instance variables to store the store screen type, resolution, and display size.

Create a parameterized constructor that accepts values for each instance variables for the `TV` and `Product` classes.

Add the getter/accessor and setter/mutator methods for the three instance variables.

A method to calculate the value of the stock of a TV.

Override the `toString()` method to return the information of the TV object as follows:

```
Item number        :
Product name       :
Screen type        :
Resolution         :
Display size:
Quantity available:
Price (RM)         :
Inventory value (RM):
Product status     :
```

UserInfo:

The `UserInfo` class is used to represent a user so that it records which user managed the stock. A `UserInfo` object has two attributes (name and user ID) and should have methods to:

- Get name of the user. Prompts the user to enter first name and surname.
- Check if the name entered contains space(s).
- Generate a user ID for user if a valid name is entered (contains space(s)). Take the first initial from the first name and add it to the full surname to generate the code. If there is no space, then set the user ID to the default value (“guest”).

For example,
If user's full name is Ah Peng Wong, his user ID will be AWong.
If user's full name is AhPengWong, his user ID will be guest.

StockManagement:

The `StockManagement` class is the application class which controls the flow of the SMS, displays messages and gets inputs from the users. It should at least have an array/list of **Product** objects to store the values for each of the instance variables of the `Product` class. The number of elements will be specified by user.

The `StockManagement` class should also have the following **static methods** to:

- Get the maximum number of products the user wishes to store in the system. Prompt the user for how many products they wish to add, and only accept positive values of 0 and above. This method takes the Scanner object as a parameter and return the maximum number of products to store in the system.
- Display the contents of the products array (index value of the products array and name of each product) allowing the user to select the product that they want to update. This method takes the products array and a Scanner object as a parameter and will return the product choice selected by user.
- Display the menu of the system. This method takes a Scanner object as a parameter and will return the menu choice entered by user. The menu should look like the following:

```
1. View products
2. Add stock
3. Deduct stock
4. Discontinue product
0. Exit
Please enter a menu option:
```

  Note: Only accept numbers between 0 and 4. If the user enters invalid inputs, re-prompt to the user for input until they give a valid response.

- Add stock values to each identified product. Prompt the user for how many products to add, and only accept positive values of 0 and above. Once a valid value has been entered, the quantity of selected product in stock should be updated. This method takes the products array and a Scanner object as parameters and have no return value.
- Deduct stock values to each identified product. Prompt the user for how many products to deduct. The restrictions on the input are that the value must be 0 or greater and cannot be greater than the current quantity of stock for that product. This method takes the products array and a Scanner object as parameters and have no return value.
- Allow user to set the status of a product (active or discontinued). Set the value of the product status to **false** for the chosen product. This method takes the products array and a Scanner object as parameters and have no return value.

- Execute the appropriate methods (i.e., to view products, add stock, deduct stock, or discontinue stock) that you have created in this class using switch-case statement. This method takes the menu choice, products array and a Scanner object as parameters and have no return value.
- Allow user to add a refrigerator or TV product. Prompt the user for an input by allowing them to choose between refrigerator or TV product. If the user enters an integer value less than 1 or greater than 2 then an error message (Only number 1 or 2 allowed!) should be displayed. And the user should be re-prompted until a valid input is entered.
  Include a selection statement that will allow the user to add a refrigerator if they enter 1 or a TV if they enter 2. Invoke the appropriate add product methods. This method takes the products array and a Scanner object as parameters and have no return value.
- Add product for refrigerator. Prompts to ask the user for some information in the order: name > door design > color > capacity > quantity available in stock > price > item number. Note: You have to clear the input buffer before you ask for any values.

  In the last line in this method, use the values that were entered by the user to create the refrigerator object and store it into the array. Note: Use polymorphism to create a refrigerator object instead of a generic product object.

- Add product for TV. Ask the user to input values for each of the attributes of the TV class. The prompts should be displayed in the following order: name > screen type > resolution > display size > quantity available in stock > price > item number. Note: You have to clear the input buffer before you ask for any values.

  In the last line in this method, use the values that were entered by the user to create the TV object and store it into the array. Note: Use polymorphism to create a TV object instead of a generic product object.

- Display the contents of the products in the array/list. This method takes the products array as a parameter but will not return any values.
- A **`main()`** method that starts the program that fulfils the requirements stated in Section 3.1.

3.3 Extension (10 marks)

- Application extension – Extend the application by adding new types of product class. A good design should allow extending the application with minimum changes to the main components of the program.
- Add some exception handling to handle runtime errors (for example, incorrect data type of the input value entered by user)
- Graphical user interface (GUI) – Enhance the application by providing easy to use and nice user interface with the use of JavaFX technology.

3.4 Recording Video - Every member MUST take part in the presentation (25 marks)

- Introduction (2 marks):
  Introduce yourself (name and student ID) and state the assignment title with the task distribution of group members as shown in the table below:

Student 1	Student 2	Student 3

- Program description (20 marks):
  Should include the program output and explain/demonstrate on how the program works to get the output.
- UML Class diagrams (3 marks):
  Should include the classes and their attributes, constructors and methods and the relationship between classes.