PROG 24178
Object Oriented Programming 2

Assignment 1

This assignment is to be done individually; you are not allowed to work on this assignment with anyone. See also the Academic Integrity page

Description

Use Eclipse to create a new project that allows a user to manage Inventory items.

Project Specs

Project Name: **A1_LoginName** (where LoginName is your own sheridan login name).

Package Name: **loginname** (your own login name, in lower-case letters). Do not use prog24178.loginname or anything else, just loginname.

Main class Name: **Main**

Add a second class to the project called: **Inventory**

When you create your project, make sure the main class is called **Main**. Add a second class called **Inventory**.

Failure to follow the above instructions could result in penalties. Your package and class names must follow the specifications above.

There are two parts to this assignment:

1. The Inventory Class
    - Defined by the class Inventory.
    - Models an item that can be found in a store's inventory.
2. The Inventory Program
    - This is a main() class that tests and uses the Inventory class.
    - Defined by the class Main.

Both of these classes should be part of the same package.

Class: Inventory
Data Members: - id : String - name : String - qoh : int - rop : int - sellPrice : double
Method Members: + Inventory() + Inventory(id : String, name : String, sellPrice : double) + Inventory(id : String, name : String, qoh : int, rop : int, sellPrice : double) + getId() : String + setId(id : String) : void + getName() : String + setName(name : String) : void + getQoh() : int) + setQoh(qoh : int) : void + getRop() : int + setRop(rop : int) : void + getSellPrice() : double + setSellPrice(price : double) : void + toString() : String

Notes:

- The Inventory ID stored in the id member is a specially-formed String that consists of 3 letters (can be upper-case or lower-case), followed by a dash, followed by 4 numbers. For example, "abc-1234". All ID values must follow this pattern, otherwise they are invalid. The default ID value is "ABC-1234".
- The string **name** can't be a null object (e.g. **null**, not pointing to any object) and can't be a null-String (e.g. "" with trim() applied). The default string value is "New Item".
- "qoh" is short for Quantity On Hand, and represents the quantity of this inventory item currently in stock. For example, if there are 10 items currently in inventory, then the QOH is 10. QOH should always be 0 or more, never negative. The default value is 0.
- "rop" is short for Re-Order Point, and represents the amount that the QOH should remain above before this item needs to be re-ordered. Usually this is used in special reports (and you'll do these later in the course) to make a list of items that need to be ordered: if an item's QOH is more than the ROP, then there is enough of this item in stock; if the item's QOH is less than or equal to the ROP, then you must order more of this item before you run out of it. You won't need to worry right now about performing the QOH/ROP check, you only need to make sure that the ROP is more than 0, otherwise it's invalid. The default ROP is 25.
- The sellPrice represents the selling price per unit of this item. Selling Price must be 0 or more, and the default value is 0.
- The default constructor should initialize the data members to their default values, unless you already did this when you defined those data members.
- The multi-param constructors set the values of the appropriate data members to the param values.

- All mutator methods should only assign the param value if it is valid, otherwise an IllegalArgumentException is thrown with a specific, yet concise and informative error message.
- the toString() returns the Inventory object as a String in the following format:
  Item ID (Item Name), QOH: x Price: $x.xx
  where "Item ID" is the item's ID, "Item Name" is the actual name of the item, x is the value in the QOH member, and $x.xx is the value stored in the sellPrice member (formatted with a $ and 2 decimal places).

Important!
Remember that error messages should be descriptive (what went wrong and how do you do it right?) but also concise, and should not be specific as to where param values came from. E.g. "Invalid data." and "You must enter a value greater than 0" are bad error messages (the first is too ambiguous, the second is implying that the invalid value came from keyboard entry when we have no clue if it did - it could have come from a file or from output of another method, etc). Better messages would be "Error: value must be greater than 0." or "Sorry, the value is not a valid number."

Your Inventory Class must follow all the standards and conventions used in OOP classes in Java.

The Inventory Program

This program must be in the class called Main in your assignment 1 project.

This program prompts the user to enter the inventory item ID, item name, quantity on hand, re-order point, and selling price of a specific inventory item of their choice.

Use the methods of your Inventory class to ensure that the user's input is valid. If any input is not valid, repeatedly prompt the user to enter data until that data is valid.

Note
TIP: Construct a default Inventory object *before* you start getting the user's input. Then, for each user input, use a loop to make sure that particular input doesn't cause an exception when you use the appropriate mutator method:

```
// construct default Inventory object REPEAT:
PRINT "Enter [whatever data member you're asking for]:"
GET userInput
// TRY to assign userInput to the data member using
// the mutator method
// CATCH any exceptions: display an error msg to user
WHILE the userInput is still invalid
```

You can do that same block of code for each user input that requires validation. You could even use methods to eliminate redundant code (I used one for the strings, one for the integers, and one for the price/double).

After the user has entered all 5 input values:

- Display the Inventory object as a String.
- If the item needs to be re-ordered, display the message "You need to order more *item name*." where *item name* is the actual item name.
- Ask the user how much of the item they'd like to buy, then display the total cost (with HST of 13%) formatted to 2 decimal places. If the user enters 0 or less, just display $0 for the price.

All user interaction and output should be displayed on the console/screen. Sample interaction and output below (you can use these values to test your program):

Sample Interaction and Output

Program Prompts are in GREEN BOLD,
User Input is in BLUE ITALIC,
`Program Output is in MONOSPACE BOLD`.
Note that the colours are for demonstration purposes only: you are not required to have colours in your program's interaction/output.

Example 1:

Enter Inventory Item ID: *DEC-7761*
Enter Item Name: *Gotta Go Clay Cat Litter, 15kg*
Qty On Hand: *14*
Re-Order Point: *15*
Selling Price: *15.95*

```
DEC-7761 (Gotta Go Clay Cat Litter, 15kg), QOH: 14 Price: $15.95
You need to order more Gotta Go Clay Cat Litter, 15kg.
```

Enter # of units to buy: *10*
```
Total Cost: $180.24
```

Example 2:

Enter Inventory Item ID: *TEC-3304*
Enter Item Name: *Furry Toy Mouse, Grey*
Qty On Hand: *67*
Re-Order Point: *20*
Selling Price: *1.9*

```
TEC-3304 (Furry Toy Mouse, Grey), QOH: 67 Price: $1.90
```

Enter # of units to buy: *5*
```
Total Cost: $10.74
```

Example 3:

Enter Inventory Item ID: *toy111*
Error: Inventory ID must be in the form ABC-1234
Enter Inventory Item ID: *toy-111*
Error: Inventory ID must be in the form ABC-1234
Enter Inventory Item ID: *toy-1111*
Enter Item Name:
Error: you must enter an item name.
Enter Item Name: *Sisal Ball, Blues*
Qty On Hand: *-2*
Error: QOH must be 0 or more.
Qty On Hand: *32*
Re-Order Point: *-2*
Error: ROP must be greater than 0.
Re-Order Point: *0*
Error: ROP must be greater than 0.
Re-Order Point: *12*
Selling Price: *-7*
Error: Selling price must be greater than 0.
Selling Price: *1.49*

```
toy-1111 (Sisal Ball, Blues), QOH: 32 Price: $1.49
```

Enter # of units to buy: *-5*
```
Total Cost: $0.00
```

Your program interaction must appear as above (except the colours: those are only there to help you see what's what in the examples). If you change the prompts, they must be just as informative and concise (or more so) than the ones in the example, or you will lose marks

Submission

Follow these instructions carefully or you risk penalties up to 100% of your grade!

You are to submit 3 files, separately:

1. The Eclipse project that contains your assignment, in a ZIP/RAR file.
2. A text or word processed document containing all of your source code for both the Inventory class and the Main class.
3. A text file (.txt) of the output using the "Save a File" button from the Console (output window).

1. ZIP/RAR of Eclipse Project:

- Your project file should ONLY contain the files needed for this current assignment. It should NOT contain any other projects or files.
- Your project **must** be a valid Eclipse project. No other project types will be accepted.
- Your submission file must be archived into a valid ZIP or RAR file.
- The name of this file should be **`loginName_a1.zip`** or **`loginName_a1.rar`** where "loginName" is ***your Sheridan user name***.

See Archiving an Eclipse Project if you're not sure how to do zip/rar an Eclipse project.

2. Document of Source:

You must **also** copy and paste all of your source code from both of your classes into a plain text file (e.g. .TXT) or Word document (e.g. .DOC/.DOCX).

You don't have to format this code - it's used by TurnItIn (the originality checker in SLATE, which is a piece of software that checks your submission for plagiarism against other submissions in the college, in other colleges, from the web, and various other sources).

Submit this document **in addition to** your source code zip/rar file. **DO NOT add it inside your zip/rar file** - it must be a separate file. This is used for TurnItIn (it won't accept java programs and won't examine the contents of zip/rar files).

2. Document of Output:

You must submit a text file (.txt) of the output using the "Save to File" button. In the Console/output window, right click and select "Save to File". Filename should be "**log.txt**". You must demonstrate entering at least **FOUR** different entries (similar to above interactions showing the defensive behaviour of the program) to test your program on and include the output for the run.

Upload All Three Files

Submit your assignment to the Assignment 1 drop box in SLATE. Upload the ZIP/RAR file, upload the TXT/DOC file and upload the log.txt file separately to the same drop box.

Failure to follow any of the instructions above will result in penalties or a grade of 0.

Evaluation

Your submission will be evaluated based on the following criteria:

Efficient Code: Program doesn’t use too much repetitive code (e.g. uses methods instead of copy/pasting code). Program uses variables where and only when necessary; program doesn't define variables that are never used, nor does it use too many variables for unnecessary tasks; program logic is written concisely and is not cluttered with unnecessary tasks.

Functionality: Program functions according to specifications.

Programming Style: Proper indentation and spacing; use of comments; coding conventions regarding variable/method/class names followed.

Other: All instructions regarding submissions and program specifications have been followed; submission was completed and submitted as requested in a timely fashion; techniques discussed in class have been used.

Originally adapted from Prof. Wendi Jollymore’s material.