Concordia University
Comp 249 - Winter 2016
Object Oriented Programming II
Assignment 1

Deadline:	By 11:59pm, Wed January 27, 2016
Evaluation:	5% of your final grade
Late Submission:	No late submission.
Teams:	The assignment can be done individually or in teams of 2 (from the same lecture section). Submit only one assignment per team.
Purpose:	The purpose of this assignment is to review objects, arrays of objects and other concepts seen in COMP 248 as well as to generate Javadoc, a new concept in COMP249.
CEAB/CIPS Attributes:	Design - Problem analysis

Note & suggestions about this assignment:

- This assignment is designed to allow you to review as many concepts as possible covered in COMP248.
- It is advisable that you code and test parts A and B before writing up part C.
- Try writing the assignment without referring to your notes from COMP248. If you find that you are consulting your notes be sure to review the concepts you are looking up.
- If there are concepts from Comp248 that you still have difficulties with, be sure to review those concepts and to ask for help from your instructor and/or tutor if necessary.
- Plan your solution and avoid solving it by "trial and error". Your goal for this assignment should not be to just get it to work, but to hand in an efficient program and one that you are proud off. Also be sure to understanding how/why your final version works as well as why previous versions did not work.
- You will notice that you are given very little information on how to implement the driver. You are expected to use your knowledge from COMP248 to design an efficient solution. Be mindful of repetitive code and complicated logic.
-

General Guidelines When Writing Programs:

- Include the following comments at the top of your source codes

```
// -----------------------------------------------------
// Assignment (include number)
// Written by: (include your name(s) and student id(s))
// For COMP 249 Section (your section) – Winter 2016
// -----------------------------------------------------
```

- In a comment, give a general explanation of what your program does. As the programming questions get more complex, the explanations will get lengthier.
- Include comments in your program describing the main steps in your program.
- Display a welcome message.
- Display clear prompts for users when you are expecting the user to enter data from the keyboard.
- All output should be displayed with clear messages and in an easy to read format.
- End your program with a closing message so that the user knows that the program has terminated.

Part A: User defined classes and methods – Old Swedish Currency

In 1777, the Swedes used a monetary system based on *riksdaler*, *skilling*, and *runstycken*. This system was rather complicated because it was not decimal: 1 riksdaler was divided into 48 *skillings* and 1 *skilling* was divided into 16 *runstyckens*.
So if you bought something for 4 *riksdaler*, 16 *skilling*, and 6 *runstyckens* and you had an 8% tax, the tax would be 0 *riksdaler*, 16 *skillings*, and 11 *runstyckens*, and the total bill would be 4 *riksdalers*, 33 *skillings*, and 1 *runstycken*!

In this assignment, you will write a class to manipulate this currency called `OldSwedishCurrency`. All objects created must be normalized, in otherwords each denomination has the minimum number in it. This means that that the number of *skilling* is smaller than 48, and the number of *runstycken* is smaller than 16.
Following is an example: if the object created contains 5 *riksdalers*, 100 *skillings* and 37 *runstyckens,* it should be normalized to contains 7 *riksdalers*, 6 *skillings* and 5 *runstyckens*.

i. 37 *runstyckens* = 2 *skillings* and 5 *runstyckens* so 5 *riksdalers*, 100 *skillings* and 37 *runstyckens* is normalized to 5 *riksdalers*, 102 *skillings* and 5 *runstyckens*
ii. 102 *skillings* = 2 *riksdalers*, 6 *skillings* so 5 *riksdalers*, 102 *skillings* and 5 *runstyckens* is normalized to 7 *riksdalers*, 6 *skillings* and 5 *runstyckens*.

It might be useful to include a static method in the class to normalize objects of type `OldSwedishCurrency`.

Your class must contain at least:

- 3 instance variables that represent the *riksdaler*, *skilling* and *runstycken*.
- 3 contructors:
  - A default constructor which initializes all instance variables to 0.
  - A second constructor which takes 3 integer arguments and initializes the instance variables to these given values. If any of these values are negative, initialize all instance variables to zero.
  - Copy constructor.
- 3 accessor methods (one for each instance variable).
- 3 mutator methods (one for each instance variable). Be sure that the object is normalized after any changes.

- an *equals* method that returns true if the value of the instance variables of the calling and passed *OldSwedishCurrency* objects are equal and false otherwise.
- a *compareTo* method that compares two *OldSwedishCurrency* objects and returns -1, 0, or 1 depending upon whether the invoking object is less than, equal to, or greater than the passed *OldSwedishCurrency* object.
- a *toString* method that returns a *String* with the content of the calling *OldSwedishCurrency* object. Here is a recommended format:
  `x riksdaler, y skilling, z runstycken`
- An *add* method that creates a new *OldSwedishCurrency* object containing the sum of the invoking and passed *OldSwedishCurrency* objects. Make sure that the content of the new *OldSwedishCurrency* object is normalized.
- A *substract* method that creates a new OldSwedishCurrency object containing the substraction of the invoking and passed *OldSwedishCurrency* objects. Make sure that the content of the new *OldSwedishCurrency* object is normalized. If the vlue of any of the instance variables are negative, set all of the instance variables to 0.
- A *convertToRunstycken* method which converts the content of the calling object to its equivalent *runstyckens* value and return it as an integer.
- A static *convertFromRunstycken* method which takes an integer *runstyckens* value and returns the corresponding normalized *OldSwedishCurrency* object.

Before moving on to Part B of this assignment it is recommended that you write a driver program to test your *OldSwedishCurrency* class.

Part B: User defined classes and methods – Household Goods

Write a class to manipulate a class called *HouseholdGoods*, which contains a String to describe the type of household good (Electronics, Applicance, Furniture), a description of the household good (String) and its price (object of type *OldSwedishCurrency*).

Your class must contain at least:

- One static instance variable which will keep track of the number of objects of type *HouseholdGoods* instantiated.
- 3 instance variables to represent the type, description and price.
- 2 contructors:
  - A default constructor which initializes the descricption to the null string, the price to 0 riksdaler, 0 skilling, 0 runstycken, and increment the instantiated object counter by 1 to reflect the creation of a new object.
  - A second constructor which takes 3 arguments (two Strings and an object of type *OldSwedishCurrency*), initializes the instance variables to the given values and increment the instantiated object counter by 1 to reflect the creation of a new object.
- 4 accessor methods to return the value of each instance variable.
- 3 mutator methods to set the value of the 3 non-static instance variables.
- an *equals* method that will test if two HouseholdGoods objects are equal.

- a *toString* method that returns a String with the content of a *HouseholdGoods* object with appropriate labels.

Part C: Driver

You are the owner of a household goods store and need help in keeping track of your inventory. Write a driver program that will contain the following methods at least.

1. **a main method**, that will:
   a. Display a welcome message (feel free to change the name of the shop).
   b. Create an empty array that will have the potential to keep records for upto 100 objects of type *HouseholdGoods*.
   c. Display a main menu (Fig 1) with the following choices and keep prompting the user until they enter a number between 1 and 6 inclusive:

```
-----------------------------------------------------
        Welcome to Nancy's Household Goods Store
-----------------------------------------------------

What would you like to do?
      1. Enter a new item in inventory (password required)
      2. Change information of an item in inventory (password required)
      3. Display all items of a specific type
      4. Display all items under a certain price
      5. Statistics on your inventory
      6. Quit
Please enter your choice -->
```

Fig 1. Main menu

   d. When option 1 is entered:
      i. Prompt the store owner for his/her password. (Make sure you have a constant variable containing the password "comp249" - don't use any other password as it will be easier for the marker to check your assignments). The store owner has 3 tries to enter the correct password. After the 3rd illegal entry, the main menu in figure 1 is displayed again.
      ii. If the correct password is entered make sure that there is enough space in the array to add another item. If yes prompt the owner for the necessary information and add the item.
   e. When option 2 is entered :
      i. Prompt the store owner for his/her password. (Make sure you have a constant containing the password "comp249" as a constant- don't use any other password as it will be easier for the marker to check your assignments). Again the owner has 3 tries to enter the correct password. After the 3rd illegal entry, the main menu in figure 1 is displayed again.
      Ask the user which item number s/he wishes to update. The item number is the index of the object in the array. If there is no object at the specified

index location display a message on the screen ask the user if they wish to enter another index or quit this operation and return to the main menu. If the index contains an object, display on the screen all of the current information for that object in the following format:

```
     Item # n
Type: xxx
Description: xxx
Price: x riksdaler, x skilling, x runstycken
```

Then ask the user which attribute they wish to change by displaying the following menu (see Fig. 2).

```
What would you like to change?
  1. Type
  2. Description
  3. Price
  4. Quit
Enter choice >
```

Fig 2. Update menu

Once the user has entered a correct choice, make the changes to the requested attribute display all of the attributes to show that the attribute has been changed. Keep prompting the user for additional changes until the user enters 4.

f. When **option 3** is entered prompt the user for the type and display all objects of that. Make sure that a valid type is entered.

g. When **option 4** is entered:

Promtp the user for a price and display all objects of type *HouseholdGoods* whose price is less than the entered price.

h. When **option 5** is entered display the following menu:

```
What information would you like?
  1. Cost and details of cheapest item
  2. Cost and details of most costly item
  3. Number of items of a each type
  4. Average cost of items in inventory
  5. Quit
Enter your choice >
```

Fig. 3 Statistic menu

Promtp the user for his/her choice (making sure it is a valid choice). Perform the necessary action using the static methods listed below when

possible, otherwise write the necessary code using the methods of class *HouseholdGoods*. Keep prompting the user for choices until s/he decides to quit, at which point you should display the main menu (Fig.1).

i. When **option 6** is entered, display a closing message and end the driver.

2. A **static method** in the driver called *lowestPrice* which will find and return the index of the household item with the lowest price in the array.

3. A **static method** in the driver called *highestPrice* which will find and return the index of the of the household item with the highest price in the array.

4. Any other static method you may find useful.

Note 1: Both classes *OldSwedishCurrency* and *HouseholdGoods* should be documented using Javadoc.
Note 2: It is possible that you don't use all of the methods in classes *OldSwedishCurrency* and *HouseholdGoods*.

Submitting Assignment 1

- Naming convention for zip file: Create one zip file containing all source files and Javadoc files for your assignment using the following naming convention:
  - If the assignment is done by 1 student:
    The zip file should be called *a#_studentID*, where # is the number of the assignment and *studentID* is your student ID number.
    For example, for the first assignment, student 123456 would submit a zip file named a1_123456.zip.
  - If the assignment is done by 2 students:
    The zip file should be called *a#_studentID1_studentID2*, where *#* is the number of the assignment, and *studentID1* and *studentID2* are the student ID numbers of each student.
    For example, for the first assignment, student 123456 and 9876543 would submit a zip file named a1_123456_9876543.zip
- Check your section's course webpage for instructions on where and how to submit your assignment.

Evaluation Criteria for Assignment 1 (20 points)

Javadocs for classes *OldSwedishCurrency* and *HouseholdGoods*	4 pts
Implementation of classes *OldSwedishCurrency* and *HouseholdGoods* based on provided specifications. (3 pts each)	6 pts
Accuracy of driver	4 pts
Efficiency of driver/application – use of methods in classes, use of static method, non-repetitive code.	4 pts
Format of output/ readability	2 pt