Date: 9/27/2020

Assignment 1

CSCI 45000 - Principles of Software Engineering Due Date : 10/12/2020

(1) [20 Points]
Draw a use case diagram for a ticketing system for the New York City subway system using the following guidelines. The subway system must include two actors: a Passenger who buys various types of tickets, and a Central Facility such as a Computer System that stores pricing information for various types of tickets in a reference database.

Your use case diagram should include the following use cases:
Purchase-1-Way-Ticket,
Purchase-Daily-Pass
Purchase-Weekly-Pass,
Purchase-Monthly-Pass, and
Update-Ticket-Price.

Also include the following exceptional cases: Time-Out (i.e., a passenger took too long to insert the right amount), Transaction-Aborted (i.e., passenger pressed the cancel button without completing the transaction), Ticketing-Machine-OutOfChange, and Ticketing-Machine-OutOfPaper. You must use extensions, generalizations and inclusions whenever applicable.

---

(2) [20 Points]
Write the possible sequence of steps (2-column format) to show the flow of events for the use case Update-Ticket-Price that you drew in Question (1) above. Use your imagination, but be realistic. Do not forget to specify any relationships, pre- and post-conditions. Can you think of any quality requirements ?

---

(3) [20 Points]
Draw a UML class diagram representing hypothetical entities or things **A**, **B,C**, **D**, defined by the following statements: **A** is composed of a number of **B**'s, which in turn are composed of a number of **C**'s. Each **C** is again composed of **D**'s." Focus only on classes and relationships. [15 points]

Add multiplicity to the class diagram you produced. [5 points]

---

4) [40 points] Use java RMI to program a simple remote bank account for a customer "John Raymond". The account object should exist remotely and the client should use Java RMI to lookup the remote account object and perform operations on it.

The remote bank account should be implemented in a class named **AccountImpl**. It should store two attributes **name** and **balance**. Attribute balance should of type **Money**, a simple serializable wrapper

around a **double**. Class **AccountImpl** should have get/set methods for **name** and **balance**, it should also be able to have the following two functions to operate on **balance**: **withdraw** and **deposit**. Your code for creating an **AccountImpl** object and registering it with rmiregistry should be in class **RemoteAccount**. The client should have the code to lookup the remote object from rmiregistry and use that object to deposit/withdraw money to the account. You are free to use any other additional classes/methods needed to complete the requirements for deposit, withdrawal and display current balance remotely in the client.

Java code package structure: You should create three directories : bank, common, and client.

Code for the remote server classes should be in "bank".
Code for the common interfaces should be in "common"
Code for client should be in "client".

Open two terminals. In terminal one, start the rmiregistry in the background and then start the service for the remote account as : **java bank.RemoteAccount**. Assuming your client is defined in class **AccountClient**, In the other terminal, you should run the client **java client.AccountClient** to perform the operations of deposit and withdrawal. It should print simple messages in the terminal showing the effect of the operations.

You can use the following machines for RMI :

in-csci-rrpc01.cs.iupui.edu
in-csci-rrpc02.cs.iupui.edu
in-csci-rrpc03.cs.iupui.edu
in-csci-rrpc04.cs.iupui.edu
in-csci-rrpc05.cs.iupui.edu
in-csci-rrpc06.cs.iupui.edu

First do : ssh -X your_user_name@tesla.cs.iupui.edu

Then you can ssh to the other machines above.
You are free to run your server **bank.RemoteAccount** and client **client.AccountClient** in any of the machines above.

For example:

You are running the server in "in-csci-rrpc01.cs.iupui.edu" and client in "in-csci-rrpc02.cs.iupui.edu". Your code in **bank.RemoteAccount** should have the correct machine name ("in-csci-rrpc01.cs.iupui.edu") to bind your remote object. Similarly your code in **client.AccountClient** should have the correct machine name ("in-csci-rrpc01.cs.iupui.edu") to lookup the remote server object.

Also, before running rmiregistry, make sure no older instance of rmiregistry is running in your account, if it is running, you may have to kill it before starting the new rmiregistry.

Notes: Any diagrams or handwritten text should be clear and legible, otherwise points will be deducted.