CISC106 – Lab 07

This lab is due 07/28/2013 at 11:55 p.m. (submission via Sakai)

- **Now you will probably need to submit multiple files to Sakai, depending on the problem's specifications.**
- **For now on we will be using the Class Design recipe to write classes and methods. You can still use the assertEqual function to run test cases. You need to provide at least 3 test cases for each new method you write, when applicable. Please keep in mind that those elements are worth at least half of the question, so you may not want to forget them!**
- **Make sure the file cisc106.py (attached) is in the same directory of your files while running it. Also make sure you include it in program.**
- **The problems are worth 90 points + 10 points for attending the lab session.**

Create a file named account.py on which you will solve problems 1 and 2

Problem 1: (33 points)

a) Design a class named Account that contains a customer's first name, a customer's last name, balance of the account, interest rate in one period (in decimal form, not in percent form; therefore, 10% interest rate is represented here as 0.1). All the attributes in the class should be private. The methods your class should implement are:
b) A constructor that initializes all 4 data fields;
c) A method to get the first name of the account named **getFirstName**;
d) A method to get the last name of the account named **getLastName**;
e) A method to get the balance of the account named **getBalance**;
f) A method to get the interest rate of the account named **getInterestRate**;
g) A method that makes a deposit of a given amount into the balance named **deposit**;
h) A method to withdraw a given amount from the balance called **withdraw** - this method should return false (without making a withdrawal) if the amount requested is more than the balance, otherwise it should return true indicating the withdrawal was successful;
i) A method to transfer a given amount from one account to another named **transfer** – this method should receive 3 parameters: the amount to transfer, the origin account and the destination account. You also need to make sure there is enough balance in the origin account to be transferred successfully; The method returns True if the transfer was successful and False otherwise;
j) A method that will accrue interest in the balance using the interest rate property to compute the updated balance named **accrueInterest**. The interest will be accrued for one interest period only.
k) Write the __str__ method that should return a string that contains all the information about an account.

Problem 2: (27 points)

Outside the Account class

1) Write a function called **getAccountInformation** that receives a list of accounts and asks the user for a first name and a last name (you can consider the user will type a valid name) and, after finding the account that belongs to the person the user typed, calls the __str__ method from Account passing the account object.

2) Write a function called **totalBalance** that receives a list of accounts and recursively calculates the total balance of all accounts.

3) Write a function called **executeTransfer**, which receives a list of accounts and asks the user which accounts should be involved in a transfer. The user should provide the first and last names of both accounts (origin and destination accounts – you can consider that the user will type valid names) and the amount that needs to be transferred. This function should first find the accounts in the list and then call the method transfer from Account class to execute the transfer. It returns True if the transfer was done successfully, False otherwise.

4) Write a function called **executeAccrue**, which receives a list of accounts and asks the user which account should accrue its interest. The user should provide the first and last names on the account – you can consider that the user will type a valid. This function should first find the account in the list and then call the method **accrueInterest** from Account class.

In your main function

You should create a list with 6 accounts (they can either be entered by the user or defined in your program as a constant. If you choose to ask the user to enter it, I would recommend that you define a constant in your program for testing purposes). After that, your program should interact with the user showing her/him a menu option, such as:

Choose one of the options below:

1) Get an account balance
2) Get the total balance of all accounts
3) Accrue interest in an account
4) Realize a transfer
5) Exit

Depending upon the option the user chooses, your program should the appropriate function you developed.

Now create a new Python file named sudoku.py on which you will answer problem 3

Problem 3: (20 points)

You will implement a class Sudoku, which will represent a Sudoku game. In this class, you will write methods to perform a series of validity tests related to the Sudoku game rules.

1) Description of the Game for this Homework

A Sudoku game is represented by a board composed of 9 x 9 squares. In each square, a number between 1 and 9 is placed. The goal of the game is to place numbers from 1 to 9 in each square such that:

- Each row has one occurrence of each number from 1 to 9;
- Each column has one occurrence of each number from 1 to 9;
- Each 3 x 3 block or box in the board has all the numbers from 1 to 9 without repetition. There are 9 boxes numbered 0 to 8. Refer to Figure 1 for box position.

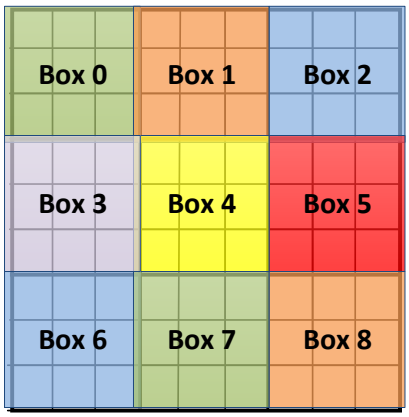

Figure 1: Inner boxes in Sudoku game.

2) Sudoku **class** and **Constructor**

Write a Sudoku class to represent the Sudoku game.

This class must have a private instance field (or instance variable) to hold the 9 x 9 Sudoku board, i.e., a 9 x 9 multidimensional list.

3) **rowCheck** Method

Write a rowCheck method that receives a row number as parameter and returns true (or false) if the row has all the numbers within 1 to 9 without repetition.

4) **columnCheck** Method

Write a columnCheck method that receives a column number as parameter and returns true (or false) if the column has all the numbers within 1 to 9 without repetition.

5) **boxCheck** Method

Write a boxCheck method that receives a box number as parameter and returns true (or false) if the 3 x 3 box in the Sudoku board has all the numbers within 1 to 9 without repetition.

Problem 5: (10 points)

From your project, submit the function that calculates your system's accuracy. Please do not forget to provide comments describing the decisions you made in order to calculate the accuracy of the system.