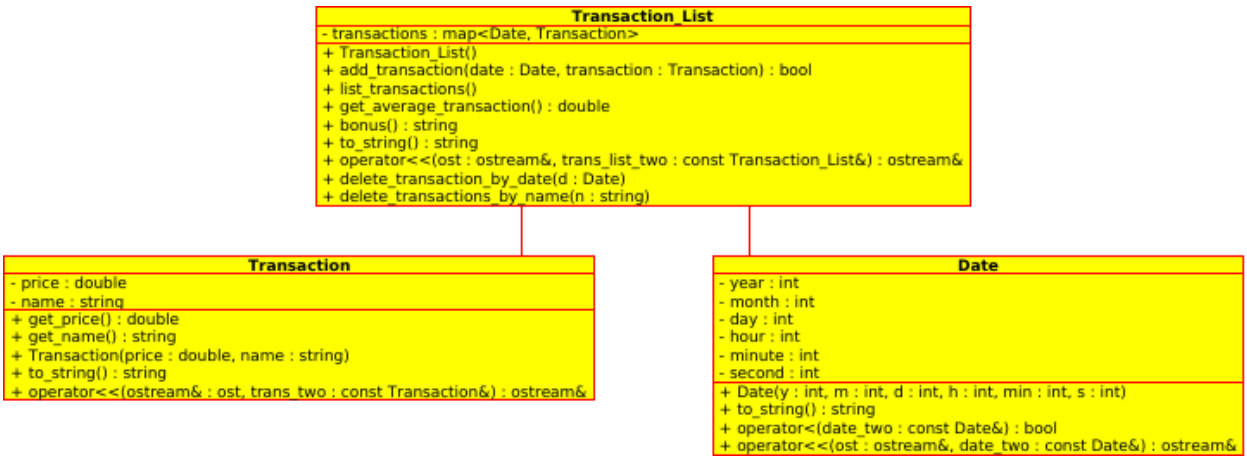
CSE 1325-001 Homework #4 Multiple Classes and Maps

In this homework, you will be making a different version of your Homework #3. You will be using a map to store Transaction objects (values) sorted by Date objects (keys). You will also be making a menu to allow a user to select different options and input data.

Part 1: Classes



You are to implement 3 classes: Date, Transaction, and Transaction List.

Date represents a date, down to the second. It has 6 private variables: year representing the year, month representing the month, day representing day, hour representing the hour, minute representing the minute, and second representing second. There are also 4 public functions. The constructor takes in values for the private variables and assigns them. to_string() converts the data of the class to a string. operator<() overloads the < operation for Date objects when doing comparisons. operator<<() overrides the << operation for Date objects when sending the Date object to and output stream.

Transaction represents a transaction that was ran by a company. Each transaction has a price, the value that was processed by the company, and a name, the person who processed the transaction. Each Transaction object has 5 public functions. get_price() returns the value processed in the transaction. get_name() returns the name of the person who processed the transaction. Transaction() is the constructor that assigns the values of price and name. to_string() converts the data of the class to a string. operator<<() overrides the << operation for Transaction objects when sending the Transaction object to and output stream. CORRECTION TO UML: in the constructor, the variable names are p and n.

Transaction_List represents a list of transactions stored in a map. This map has keys of Dates and values of Transactions. This list is able to add transactions, list all transactions, get the average transaction, delete a transaction when given a date, and delete all transactions by a certain employee. When deleting a transaction, a useful error message must be displayed to the user if unsuccessful. This list should also determine who gets a bonus. Bonuses are determined differently than in Homework #3. In this homework, it is whoever rang up the highest total value of transactions, not just the number. Also there are to_string and an operator<< functions that function similarly to the previous two classes.

Part 2: main.cpp

In the main cpp, you must create a menu where the user can select different options. These options must include printing out the list of transactions, adding a new transaction, deleting a current transaction by date, deleting all transaction by a certain employee, getting the average transaction, and getting the name of the person who earned the bonus. Please note that menus may have a submenu or require additional input, such as asking for the date, price, and name.

Here is an example menu:

```
Welcome to the Transaction List Management Solution.
Please make a selection from the following menu:
1: Print all transactions
2: Add a transaction
3: Delete a transaction(s)
4: Average transaction value
5: Bonus winner
? |
```

All functionally must be able to be tested by the GTA when grading.

Part 3: Bonus 1 (5pts)

This will be the exact same as the full credit, but will require you to get the system time for the date when a new transaction is added instead of asking the user for a date.

Part 4: Bonus 2 (10pts)

You will modify the Transaction class to only have the price private variable and to have the get_name() method removed. This will require the constructor and to_string() method to be changed as well. You will then create a new class called Employee that has private variables name and id. This class should have a constructor, accessor methods for these fields, as well as a to_string() and an operator<< overload.

The map in Transcation_List will now become a multimap with three fields. The order of these fields will be <Date, Transaction, Employee>. This will require several other methods in this class to be changed as well.

You will also show these changes in a new UML Diagram. I have provided the base UML for this project, which is the one pictured on the first page. Be sure to show associativity.

Part 5: Deliverables

You will submit your code and screenshots via Blackboard. You will upload a zip file, named "abc1234_HW4.zip", which contains 1 folder (3 if you did all of the bonus)

- full_credit
    - abc1234_Date.h and abc1234_Date.cpp
    - abc1234_Transaction.h and abc1234_Transaction.cpp
    - abc1234_Transaction_List.h and abc1234_Transaction_List.cpp
    - abc1234_main.cpp
    - makefile
    - abc1234_main.png (or multiple if multiple screenshots were taken). These screenshots will be picture of your code running in terminal.
    - Instructions for compiling and running your code (either in comments in blackboard or in a README file)
- bonus_1
    - abc1234_Date.h and abc1234_Date.cpp
    - abc1234_Transaction.h and abc1234_Transaction.cpp
    - abc1234_Transaction_List.h and abc1234_Transaction_List.cpp
    - abc1234_main.cpp
    - makefile
    - abc1234_main.png (or multiple if multiple screenshots were taken). These screenshots will be picture of your code running in terminal.
    - Instructions for compiling and running your code (either in comments in blackboard or in a README file)
- bonus_2
    - abc1234_HW4_Class_Diagram.xmi
    - abc1234_Date.h and abc1234_Date.cpp
    - abc1234_Transaction.h and abc1234_Transaction.cpp
    - abc1234_Employee.h and abc1234_Employee.cpp
    - abc1234_Transaction_List.h and abc1234_Transaction_List.cpp
    - abc1234_main.cpp
    - makefile
    - abc1234_main.png (or multiple if multiple screenshots were taken). These screenshots will be picture of your code running in terminal.
    - Instructions for compiling and running your code (either in comments in blackboard or in a README file)

Full credit files named incorrectly result in a loss of 5 points each.