MPP Standardized Programming Exam
August, 2017

This two-hour programming test measures the success of your MPP course by testing your new skill level in two core areas of the MPP curriculum: (1) Lambdas and streams, and (2) Implementation of UML in Java.You will need to demonstrate a basic level of competency in these areas in order to move past MPP.

Your test will be evaluated with marks "Pass" or "Fail." A "Pass" means that you have completed this portion of evaluation only; your professor will evaluate your work over the past month to determine your final grade in your MPP course, taking into account your work on exams and assignments. A "Fail" means you will need to repeat MPP, with your professor's approval.

There are two programming problems to solve on this test. You will use the Java classes that have been provided for you in an Eclipse workspace. You will complete the necessary coding in these classes, following the instructions provided below.

In order to pass this programming test, *you must get a score of 70 or higher on both of the problems given here.* Getting a high score on one problem and a low score (below 70) on another will result in a "Fail."

Problem 1. **[Lambdas/Streams]** In your `prob1` package, you will find a class `Customer` and another class `Problem1` that contains two static methods:

```
static List<String> elementsInBoth(List<String> list1, List<String> list2)
static List<String> getZipsOfSpecialCustomers(List<Customer> list)
```

The method `elementsInBoth` returns a list of all `Strings` that occur in both of the two input lists. Below is an example of how this method should behave:

Example: If `list1 = {"A", "B", "D"}` and `list2 = {"B", "C", "D"},` then the return list should be `{"B", "D"}` since both "B" and "D" occur in both lists.

The method `getZipsOfSpecialCustomers` returns a list of the zipcodes, in sorted order, of those Customers who live in a city for which the name of the city contains 6 or more characters, but which does not contain the letter 'e'. Your output list must not contain duplicate elements.

Example: Below are 5 customers.
Customer 1: ["Bob", "11 Adams", "Fairfield", "52556"]
Customer 2: ["Andy", "1000 Channing Ave", "Oskaloosa", "54672"]
Customer 3: ["Zeke", "212 Wilkshire Blvd", "Chicago", "57532"]
Customer 4: ["Tom", "211 Blake Ave", "Oskaloosa", "54672"]
Customer 5: ["Bill", "10 Wolfsen Blvd", "Orkin", "84447"]
When run on this customer list, the method should return the following list of zip codes:

["54672","57532"]

The Fairfield customer was ignored (because 'e' occurs in Fairfield) and the Orkin customer was ignored (because the length of "Orkin" is less than 6). Also, the multiple occurrences of an Oskaloosa zipcode

were reduced to just one so that there were no duplicates in the final list. Note that the final zipcode list is in sorted order.

A `main` method has been provided that will help you test your implementations of both of these methods.

Requirements for Problem 1.

1. Your code may not contain any loops (while loops, for loops).
2. The body of each of the methods `elementsInBoth`, `getZipsOfSpecialCustomers` must be a single `Stream` pipeline. You must not make use of instance variables or local variables declared in the body of either method. (Example of a local variable:
```
int myMethod() {
       int x = //computation
       return x;
 }
```
Here, x is a local variable. Not allowed in this problem.)
3. You may not create auxiliary methods for use in your pipeline.
4. There must not be any compilation errors or runtime errors in the solution that you submit.

Problem 2. [UML → Code] In a company, employees may have multiple bank accounts: zero or more savings accounts and zero or more checking accounts. Each checking account has an account id, a balance, and a monthly fee. Each savings account has an account id, a balance, and an interest rate associated with the particular type of savings account. It is possible to read the current balance in any of these accounts, but it is also possible to determine the balance after interest or monthly fee is applied by calling the `computeUpdatedBalance` method on the account.

An administrator has access to all employee records and from time to time computes the total balance across all employee-owned accounts; for each account, the balance that is needed in this computation is the *updated balance*. This computation is performed in the static method

`computeUpdatedBalanceSum`

in the `Admin` class.

Below is a class diagram showing the classes involved and relationships between them. A sequence diagram for the operation `computeUpdatedBalanceSum` is also provided. Your task in this problem is to write Java code that implements the classes and relationships shown in the diagram. Shells for the `Admin` and the `Employee` classes have been provided in your workspace. Also, a `Main` class (with a `main` method) has been provided for you to test your code (in the `launch` package) – the code in the `main` method has been commented out; when you are ready to test your code, you can uncomment it.

The method `computeUpdatedBalance` in `CheckingAccount` does the following computation to obtain the return value:

`balance – monthlyFee.`

The method `computeUpdatedBalance` in `SavingsAccount` performs the following computation to obtain the return value:

`balance + (interestRate * balance).`

Points to notice about the class diagram:

1. `CheckingAccount` and `SavingsAccount` are subclasses of `Account`. Also, `Account` has several abstract methods which must be implemented in its subclasses.
2. There is a one-way association from `Employee` to `Account`. It is important that your code reflects and maintains this association.
3. The diagram has a mix of dependencies and associations; make sure your code distinguishes between these properly.

You may not modify the signatures or qualifiers of the methods contained in the `Admin` or `Employee` classes that have been provided. You will need to create all the other classes mentioned in the diagrams.

Note: Your submitted code must accurately implement the UML models provided and must have no compiler or runtimte errors.