FPP Standardized Programming Exam
January, 2017

This 90-minute programming test measures the success of your FPP course by testing your new skill level in two core areas of the FPP curriculum: OO programming (specifically, polymorphism) and data structures. You will need to demonstrate a basic level of competency in these areas in order to move on to 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 FPP course, taking into account your work on exams and assignments. A "Fail" means you will need to repeat FPP, 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.

Problem 1. **[Data Structures]** For this problem, you will implement a queue of `ints`, using an array in the background. To start, initialize an array and two pointers (`front` and `rear`) in your queue class as follows:

```
private int[] arr = new int[10];
private int front = -1;
private int rear = 0;
```

When adding an element to the queue, you add the element to the position pointed to by the variable `rear`. When removing an element from the queue, you remove the element in the position pointed to by the variable `front`.

Implement all of the methods declared below so that your class behaves as a queue. The methods to be implemented are: `isEmpty, size, enqueue, dequeue`, and `peek`. A start-up class `ArrayQueueImpl`, listing all of these methods, has been provided for you in the `prob1` package in your workspace. You will need to implement the methods in that class. A `Main` class with an implemented `main` method has been provided for you to test your own code (you do not need to use this class – it is provided only for convenience).

Your code must meet the following requirements:

1. The `enqueue` method should never cause an exception to be thrown. In particular, your queue must support unlimited `enqueue` operations. This means that you will need to incorporate a procedure for resizing the background array periodically.
2. The only situation in which `dequeue` or `peek` should cause an exception to be thrown is if either of these operations is called when the queue is empty. In that case, a `QueueException` should be thrown. (`QueueException` class is provided in the `prob1` package.)
3. You may not use Java library collection classes to create your queue. In particular, you may not use any of Java's list classes, hashtable classes, `Stack` interface, or `Queue` class.
4. There must not be any compilation errors or runtime errors in the solution that you submit.

<u>Problem 2</u>. [Polymorphism] In the `prob2` package of your workspace, there are two subpackages: `prob2.incorrect` and `prob2.solution`. Both packages contain an `Employee` class, as well as classes for three different types of bank accounts (`RetirementAccount, SavingsAccount, CheckingAccount`). An `Employee` has instance variables `id` and `accounts` (which is a list of accounts). Each employee account will be one of the three account types mentioned above. There is also an `AccountManager` class containing a static method `computeAccountBalanceSum`, which takes as input a list of `Employee` objects; for each such `Employee` object, it extracts the balance from each of the accounts in the list of accounts contained in that `Employee`, and adds them to a running `sum` variable; finally, `computeAccountBalanceSum` returns the final value of `sum`.

In the package `prob2.incorrect`, all the code is correct and the total sum of balances that is computed by the `AccountManager` is correct; however, the implementation in this package is of very low quality because polymorphism has not been used.

The objective of this problem is to rewrite this code so that polymorphism is used. All the classes in `prob2.incorrect` have been copied into the package `prob2.solution`. You must make the necessary modifications to the classes you find in `prob2.solution` so that computation of total sum of balances is still correct, but computation is done polymorphically. An abstract class `Account` (unimplemented) has been provided for you to assist in your polymorphic implementation.

In each of the packages `prob2.incorrect` and `prob2.solution`, a `Main` class is provided that can be used to test the `AccountManager.computeAccountBalanceSum` method.

Requirements for this problem.

1. All Lists in your solution package must have an appropriate type (for instance, `List<Employee>` rather than just `List`).
2. Your implementation of `computeAccountBalanceSum` in `AccountManager` must correctly output the sum of the balances of all accounts in all the `Employee` objects passed in as an argument.
3. Your implementation of `computeAccountBalanceSum` must make correct use of polymorphism .
4. *None of the code in the prob2.incorrect package may be modified!*
5. You are allowed to modify declarations of the different bank account classes, but the *final* keyword used in these classes may not be removed.
6. There must not be any compilation errors or runtime errors in the solution that you submit.