FPP Standardized Programming Exam
September, 2017

This 2 hour 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. In order to pass, you must get a score of at least 70% *on each of the two problems*.

Problem 1. **[Data Structures]** In your `prob1` package, you will find a class `MyIntegerArrayList` that has two implemented methods: `size()` and `toString()`. A `MyIntegerArrayList` is populated by passing an array of `Integers` to the constructor; data validation methods ensure that an input array is non-null and that the array list itself contains no null elements. The constructor stores an input array in the variable `theArray` in `MyIntegerArrayList`, and sets the size of the list at that time to be the length of the input array. Elements of `MyIntegerArrayList` are stored in the background in the `Integer` array `arr`. Note that in this implementation, the list elements are the elements of `theArray` located in positions `0..size-1`.

For this problem implement the method

```
Integer remove(int i)
```

which should remove and return the `Integer` at position i in the list, if there is an element at that position. If no element is at that position, the method throws an `IndexOutOfBoundsException`.

Example: Suppose an instance of `MyIntegerArrayList` has the following as its background array: [3, 1, 4]. Then

i. A call to `remove(2)` should return the integer 4 and the elements remaining in the list will be 3, 1
ii. A call to `remove(-1)` and also a call to `remove(10)` should cause the remove method to throw an `IndexOutOfBoundsException`.

A main method has been provided in `MyIntegerArrayList` that will allow you to test your code. Expected outputs are shown in the comments.

Requirements for Problem 1: Your implementation will be tested by inserting different arrays into an instance of `MyIntegerArrayList`, calling your `remove` method, and examining the string that is returned by the `toString` method. (Note that since the `toString()` is guaranteed to work properly, if there are problems with the output of this method, it is only because of a mistake in your code.)

(1) The output of the `toString` method must never contain the `String` "null".
(2) The output of the `toString` method must contain precisely the elements of the array list in the exact order in which they occur in the list.

(3) In this problem, you may not use any of Java's data structures other than arrays (including `ArrayList`, `LinkedList`, any implementation of the `List` interface, or any other type from Java's collections library).
(4) If `remove(n)` is called on your list when n is negative or greater than or equal to the size of your list, your code must throw an `IndexOutOfBoundsException`. For this purpose, you must use Java's `IndexOutOfBoundsException` class (you must not create your own exception class for this).
(5) The `size()` method must always return the correct size of the list.
(6) There must be no compiler or runtime errors. In particular, no `NullPointerException`s should be thrown during execution.

Problem 2. [Polymorphism] In the `prob2` package of your workspace, you willl see 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 an *unimplemented* static method `computeAccountBalanceSum`, which takes as input a list of `Employee` objects; for each such `Employee` object, `computeAccountBalanceSum` should extract the balance from each of the accounts in the list of accounts contained in that `Employee`, and add them to a running `sum` variable; finally, `computeAccountBalanceSum` should return the final value of `sum`.

The objective of this problem is to implement `computeAccountBalanceSum` so that computation makes use of polymorphism. To do this, you will need to add a new class or interface to the `prob2` package that will generalize the different account types; you will need to adjust the type of the list (of accounts) stored in `Employee` objects accordingly.

NOTE: A method `getBalance` has already been implemented in each of the account classes – this method should be accessed in your polymorphic computation that is done in `computeAccountBalanceSum` .

A `Main` class has been 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. You are allowed to modify declarations of the different bank account classes, but the *final* keyword used in these classes may not be removed.
5. There must not be any compilation errors or runtime errors in the solution that you submit.