FPP Standardized Programming Exam
April, 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 be given a list of students, represented as instances of a `Student` class. Each student has a first name, a last name, a gpa, and a class (classes are Freshman, Sophomore, Junior, Senior). In a method called `processStudents`, you will read `Student` objects from an input list and put each in a `HashMap`. Each *value* in the `HashMap` will be a `Student` object; the corresponding *key* for a `Student` object will be a `Key` object whose instance variables are `firstName` and `lastName`, representing the first and last names of this `Student`. For each `Student` object read from the list, you will create the `Key` object and insert the `Key,Student` pair into the `HashMap`.

Your `prob1` package contains a fully implemented `Student` class, and contains a partially implemented `Key` class. This `prob1` package also contains a class `Admin` containing the following unimplemented static method:

```
HashMap<Key,Student> processStudents(List<Student> students).
```

This method will carry out the steps described above: For each `student` object in the input list `students`, it will read `firstName` and `lastName`, and then use these to populate a new `Key` object `key`; it will then insert (`key, student`) as an entry in the `HashMap`. After the `students` list has been processed in this way, the `HashMap` is returned.

There is one additional class in the `prob1` package, called `Test`, which has already been fully implemented. The `Test` class has a `main` method that will provide sample data to test your `processStudents` method. The `main` method will output "pass" to the console if the test passes, but will output "fail" if it does not. In order to get full credit, the `main` method must (correctly) output "pass".

Your tasks for this problem are as follows:

(1) Add any necessary code to the `Key` class to ensure it may be used reliably as a key in a `HashMap`.
(2) Implement the `processStudents` method
(3) Run the `main` method in the `Test` class to verify that your solution works (and correct your code if it does not).

Requirements for this problem.

(A) You may not modify the `Student` class or change the instance variables provided in the `Key` class

(B) You are allowed to modify the `Test` class, but your code *must* pass the test that has been provided for you in the `main` method of this class.
(C) Appropriate changes must be made to the `Key` class (to follow best practices and to ensure that the `main` method will output "pass" to the console).
(D) There must be no compiler or runtime errors in your submitted code.

Problem 2. [Polymorphism] In a local bicycle shop, the owner sells both bicycles and bicycle accessories. A bicycle has an id, a brand, and a price. The four types of accessories sold at the store are bicycle pumps, kickstands, mirrors, and security locks. Each of these accessories has an id, a price, and is labeled as one of these four types. At the end of each day, the owner runs an inventory software package that computes the total value of all items in the store; this is done by adding the list price of each bicycle and all accessories currently in the store.

In the `prob2` package of your workspace, you will find fully implemented classes `Bicycle` and `Accessory`, representing the products in the bicycle shop. `Bicycle` brands have been represented in the `enum Brand` and `Accessory` types have been represented in the `enum Item`.There is also a class `Inventory` which contains three static methods. One of these methods is a `public` method `inventoryValue`, which has already been implemented for you. The other two methods are `private` and *unimplemented*. These are: `prepareList` and `computeCurrentValue`; they are helper methods for the `public` method `inventoryValue`.

Your task in this problem is to implement the two `private` methods `prepareList` and `computeCurrentValue` in the `Inventory` class. Here are their signatures and return types:

```
List prepareList(List<Bicycle>, List<Accessory>)
double computeCurrentValue(List)
```

The method `prepareList` must combine the two input lists of type `List<Bicycle>` and `List<Accessory>` into a single, appropriately typed return list. The type used for the return list should be a common type for `Bicycle` and `Accessory`—the interface `Product` has been provided in your workspace to serve this purpose (you will need to write the code for this interface).

The method `computeCurrentValue` polymorphically determines the total value of all products in the combined list. It does this by calling the method `getTotalValue` on each object. (Notice that in both `Accessory` and `Bicycle`, the `getTotalValue` method returns the price of a product times the number in stock – that is, `price * numInStock`). Your method will polymorphically reads these totals, sum them, and return this sum.

The public method `inventoryValue` transforms the total computed in `computeCurrentValue` into an integer and returns this value to the caller.

Requirements for this problem.

(1) You must implement `computeCurrentValue` *using polymorphism*. (For instance, if you obtain the total by first adding totals from the `Bicycle` list and then adding the totals obtained from the `Accessory` list, you will receive no credit.)
(2) Your implementation of `computeCurrentValue` may not check types (using `instanceof` or `getClass()`) in order to read prices from any of the products in the input list.
(3) You must use parametrized lists, not "raw" lists. (Example: This is a parametrized list: `List<Duck> list`. This is a "raw" list: `List list`.) This means that all `Lists` that

appear in the code (in the `Main` class and in the `Inventory` class) must be given proper type parameters.

(4) You must implement both the methods `prepareList` and `computeCurrentValue` in the `Inventory` class.

(5) Your computation of total value must be correct.

(6) There must not be any compilation errors or runtime errors in the solution that you submit.