FPP Standardized Programming Exam
June, 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. 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 `DoublyLinkedDeleteFirst`. For this problem, you must implement the following methods

```
String deleteFirst()
boolean isEmpty()
```

This `deleteFirst()` method removes the node in position 1 of the list and returns the string contained in that node. The `isEmpty()` method returns `true` if there are no items in the list, `false` otherwise.

If `deleteFirst`() is called on a list that contains 0 or 1 element, an `IllegalStateException` must be thrown.

A `toString` method has been provided so you can test your code.

Example. Suppose your list has these values:

`["Bob","Bill","Tom"]`

After executing `deleteFirst`, the list should contain these elements (in this order):

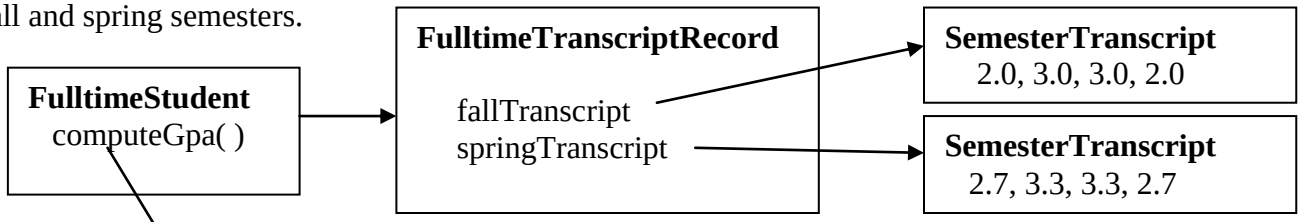
`["Bob", "Tom"]`

Requirements for Problem 1:

(1) Your code must run correctly for lists containing any number of elements, including an empty list.
(2) No data may be placed in the header node.
(3) `IllegalStateException` must be thrown when `deleteFirst` is called on your list when it has only 0 or 1 element.
(4) You must use Java's `IllegalStateException` class (you must not create your own exception class for this).
(5) You may not introduce any new instance variables or instance methods, and you may not modify the other methods in `DoublyLinkedDeleteFirst`.
(6) The `Node` class contained in `DoublyLinkedDeleteFirst` must not be modified (in particular, no constructor other than the default constructor should be used to construct instances of `Node`).
(7) During execution, each `Node` in your `DoublyLinkedDeleteFirst` must have correct values for the `next` and `previous` `Node`s.
(8) There should be no compiler or runtime errors. In particular, no `NullPointerException`s should be thrown during execution.

Problem 2. [Polymorphism] The administrative office at a particular university keeps records of all student grade point averages (GPAs). Normal record formatting is done for full-time students, but a less detailed type of record formatting is done for part-time students.

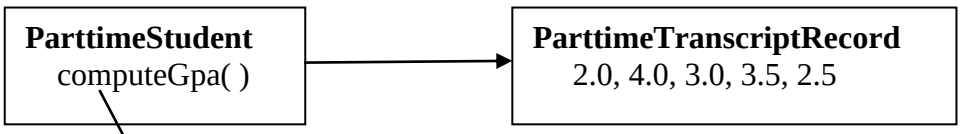
In the `prob2` package, you will find the classes `FulltimeStudent`, `FulltimeTranscriptRecord` and `SemesterTranscript`. These classes work together to provide information about a full-time student's performance during a particular school year. A `SemesterTranscript` just contains a list of grades (numerical values in the range 0.0 – 4.0). A `FulltimeTranscriptRecord` contains a `fallTranscript` and a `springTranscript`, each of which is an instance of `SemesterTranscript`. And a `FulltimeStudent` has a `FulltimeTranscriptRecord`, which provides a complete record of a student's grades in both the fall and spring semesters.



$$\text{returns: } \frac{2.0 + 3.0 + 3.0 + 2.0 + 2.7 + 3.3 + 3.3 + 2.7}{8} = 2.75$$

The `FulltimeStudent` class provides a method `computeGpa` which sums all the grades stored in the student's `FulltimeTranscriptRecord`, computes the average, and returns it.

Also in the `prob2` package, you will find the classes `ParttimeStudent` and `ParttimeTranscriptRecord`. These classes work together to provide information about a part-time student's performance during a school year. A `ParttimeTranscriptRecord` contains a list of grades (numerical values in the range 0.0 – 4.0), and every `ParttimeStudent` has a `ParttimeTranscriptRecord`. Note that grades for a part-time student are tracked for the whole year, and not considered separately for each semester as they are for full-time students.



$$\text{returns: } \frac{2.0 + 4.0 + 3.0 + 3.5 + 2.5}{5} = 3.0$$

A `ParttimeStudent` contains a method `computeGpa` which sums all the grades in the student's `ParttimeTranscriptRecord`, computes the average, and returns it.

The objective of this problem is to compute the average gpa for the current year of all students in a given input array. This is to be accomplished in two steps, by implementing the following two static methods contained in the `Admin` class:

```
public static List convertArray(Object[] studentArray)
public static int computeAverageGpa(List studentList)
```

The `convertArray` method converts the array of students that is passed to it by a calling class (in this problem, the calling class is the `Main` class) and converts it to a `List` of the proper type. The

`computeAverageGpa` method uses this list to polymorphically compute the average gpa of all the students in the list.

In order to do your polymorphic computation of average gpa, in the `convertArray` method you will need to make use of a common type for both types of students (part-time and full-time) that may occur in the input array; the abstract class `Student` (unimplemented) has been provided in your `prob2` package for this purpose. With this common type, you will be able to do the necessary polymorphic computation in `computeAverageGpa`.

Requirements for this problem.

(1) You must compute average gpa *using polymorphism.*
(2) Your implementation of `computeAverageGpa` may not check types (using `instanceof` or `getClass()`) in order to `computeGpa` from any of the students 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 `Admin` class) must be given proper type parameters.
(4) You must implement both the methods `convertArray` and `computeAverageGpa` in the `Admin` class.
(5) Your computation of average gpa must be correct.
(6) You may not remove the `abstract` keyword from `Student` or change `Student` to be an interface.
(7) There must not be any compilation errors or runtime errors in the solution that you submit.