

## FPP Standardized Programming Exam March, 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]** In your `prob1` package, you will find a class `DoublyLinkedList`. For this problem, you must implement the method

```
void addLast(String s)
```

This method inserts the input `s` into the linked list so that it is the *last* element in the list. A `toString` method has been provided so you can test your code.

*Example.* Suppose your list has these values:

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

After executing `addLast` with input `String "Carol"`, the list should contain these elements (in this order):

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

*Requirements for Problem 1:*

- (1) Your code must run correctly if the list already contains one or more elements or if it contains no elements.
- (2) No data may be placed in the header node.
- (3) You may not introduce any new instance variables, and you may not modify the other methods in `DoublyLinkedList`.
- (4) Any `Node` in your `DoublyLinkedList` must have correct values for the next and previous `Nodes`.
- (5) You are allowed to add a constructor to the `Node` if you wish.
- (6) There should be no compiler or runtime errors.

**Problem 2. [Polymorphism]** In a particular video game, a player attempts to travel the greatest possible number of miles within a specified timeframe. The player will encounter various terrains as he travels. He may make use of a variety of vehicles in order to travel. If he encounters a lake, he can choose to drive a boat across the lake. If he needs to climb paths through hills, he may choose to use a bicycle. If he has an open road, he may use an automobile. As the game proceeds, in the background a record is kept of the number of miles he travels with each vehicle; this number is stored in an object representing this type of vehicle. For example, if the player uses a bicycle, a new instance of a `Bicycle` class is

created, and the number of miles traveled until the next stop is stored in the `milesUsedToday` field.

In the `prob2` package of your workspace, you will find implementations of three types of vehicle classes: `Bicycle`, `Automobile`, and `Boat`. The game code (not shown) will insert the number of miles traveled in instances of these classes and arrange these instances into a `vehiclesArray`. This `vehiclesArray` may contain any combination of instances of `Bicycle`, `Automobile`, and `Boat`, according to how a given player plays the game.

For instance, a player may first ride a bicycle for a while, then switch to an automobile, then drive a boat, then drive a different automobile, and after that, may use another bicycle.

The objective of this problem is to compute the total number of miles traveled during one play of the game. For this purpose, a class `MilesCounter` has been provided for you in your `prob2` package. `MilesCounter` has two static methods which you need to implement.

```
public static List convertArray(Object[] vehicleArray)
public static int computeMiles(List vehicleList)
```

The `convertArray` method converts the array of vehicles that is passed to it by the game code and converts it to a `List` of the proper type. The `computeMiles` method uses this list to polymorphically compute the total miles traveled with all the vehicles in the list.

A `main` method (in the `Main` class) has been provided, which simulates a play of the game. After several vehicles are initialized, the `main` method makes a call to `convertArray` and then passes the returned `List` to `computeMiles`, and prints the total to the console.

In order to do your polymorphic computation of total miles, in the `convertArray` method you will need to make use of a common type for all the vehicles in the input array; the abstract class `Vehicle` (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 `computeMiles`.

#### *Requirements for this problem.*

- (1) You must compute total miles *using polymorphism*. (For instance, if you obtain the total miles by first computing miles from each of the vehicles and then summing these, you will receive no credit.)
- (2) Your implementation of `computeMiles` may not check types (using `instanceof` or `getClass()`) in order to read mileage from any of the vehicles 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 `MilesCounter` class) must be given proper type parameters.
- (4) You must implement both the methods `convertArray` and `computeMiles` in the `MilesCounter` class.
- (5) Your computation of total number of miles traveled must be correct.
- (6) There must not be any compilation errors or runtime errors in the solution that you submit.