Quiz #5: Classes and Interfaces (and reference casting) INF-42, Winter 2011

Name _____

Lab _____

Recall the Academic Integrity statement that you signed. Write all answers clearly on these pages, ensuring your final answers are easily recognizable. The number of points for each problem is clearly marked, for a total of 25 points. I will post my solutions on the web on Monday, off the **Solutions** link, after class.

Test your code in Eclipse, but submit the code written on this document (**not** via Checkmate).

1. (15 pts) In the Eclipse folder, examine the `LengthLess` and `DriverForCatenateAndLengthLess` classes, which are fully written. Write and test a class named **Catenate** via the driver (which uses **Catenate** and **LengthLess**). Each object constructed from this class stores three instance variables:

(1) a reference to some object constructed from a class that implements the **Decision** interface shown below (this interface is so general/important, it is in `introlib`).

```
public interface Decision {public boolean isOK(Object o);}      //Compressed onto one line
```

(2) a **String** that contains the catenation of all the "OK" values that it has examined (via the `tryIt` method, using the 1st instance variable) separated by a space. **Important**: there are spaces only **between** the catenated values.

(3) a count of the number of times that the `tryIt` method has been called (whether or not it was OK).

The constructor should initialize these fields appropriately; the accessor **getCatenation** should return the catenation of all the "OK" values; the accessor **getCount** should return the number of times the **tryIt** method was called; the mutator **tryIt** is passed an **Object** parameter whose `.toString()` value (the `Object` class defines a `toString` method) is conditionally catenated at the end of the "OK" values.

If we declare **Catenate c = new Catenate(new LengthLess(5));** (the **LengthLess** class implements the **Decision** interface and has an **isOK** method that returns **true** when its parameter is a `String` whose length is strictly less than the value passed to its constructor: i.e., **"abcd"** is legal; **"abcde"** is not legal) and statements: **c.tryIt("abc"); c.tryIt ("abcxyz"); c.tryIt ("xyz"); c.tryIt ("b12"); c.tryIt ("1234567"); System.out.println("c.getCount()+":+c.getCatenation());** Java prints **5:abc xyz b12**.

Finally, write a parameterless mutator named **reset** that resets the instance variables to their initial state (right after the object was constructed: no "OK" Strings, **0** for the count, same `Decision`).

Write the entire class: put the fields, constructor and all accessors on the left, all mutators on the right.

2. (5 pts) In the Eclipse folder, examine the `DriverForCatenateAndPrefix` class, which is fully written, and your `Catenate` class, from Problem 1 (solve that problem first and don't change your solution!). Write a class named **Prefix** that implements the **Decision** interface. When objects of this class are constructed, they are provided with a **String** argument. The **isOK** method of this class should return **true** when its parameter is a `String` that has this argument (the one passed to the constructor) as a prefix. For example, if we declare **Decision d = new Prefix("ab");** then **d.isOK("abstract")** returns **true** but **d.isOK("allocate")** returns **false**. Note that the **String** class defines a non-**static** method named **startsWith** one of whose overloaded prototypes is **boolean startsWith(String prefix)**: calling **"abnormal".startsWith("ab")** returns **true** and calling **"allocate".startsWith("ab")** returns **false**.

3. (5 pts) In the Eclipse folder, examine the `DriverForArrayWithPrefix` class, and your `Catenate` and `Prefix` classes from Problems 1 and 2. In this folder, write a **public static** method named **arrayWithPrefix**, which takes as parameters a **String[]** and a **String**. This method processes each `String` in the array and returns a **String** containing all the values from this array that are prefixed by the second (**String**) parameter.

Note: You **must** write this method, using the **Catenate** and **Prefix** classes (not writing much other code). Even if you did not fully write the solutions to the previous problems, you can use theses classes in this method as if you had written them correctly (although you cannot test your code).

If the second parameter were the **String "un"** and the array contained the strings **"until"**, **"umpteen"**, **"unknown"**, **"unkempt"**, **"use"**, and **"under"** in that order, then this method would return the **String "until unknown unkempt under"**.