LAST NAME_____________________________first name__________________________ Cornell Net id____________

CS1110 Prelim 1 9 Nov 2010

This 90-minute exam has 5 questions (numbered 0..4) worth a total of 100 points. Scan the whole test before starting. Budget your time wisely. Use the back of these pages if you need more space. You may separate the pages; we have a stapler at the front of the room.

Some potentially useful methods appear at the bottom of the this page.

Question 0 (2 pts). Write your last name, first name, and Cornell NetId, legibly, at the top of each page.

Question 1 (25 pts**) for-loop.** Suppose you have a list of your friends and another list of your enemies. This question asks you to make a list of people who are on both lists.

Complete the body of the method given below, to be placed in a class `Person`, You *must*

(1) write the invariant of a for-loop that processes a range of integers, based on the postcondition R given below,

(2) write the for-loop that processes a range of integers and provides the correct answers to the four loopy questions with respect to your invariant.

Notes:
1. You have to declare a local variable `fr`.
2. Use the methods of class `Vector` given on the bottom of this page.
3. Assume class `Person` does *not* have an `equals` method, so `p1.equals(p2)` calls the `equals` function inherited from class `Object`.

/** = a list of Persons that appear both in friends and in enemies; if there are no such Persons, the list that is returned should be empty (it is not null but is a list containing 0 elements).

Precondition: Vectors friends and enemies are not null, neither contains null, and neither contains duplicates. */
public static Vector<Person> frenemies(Vector<Person> friends, Vector<Person> enemies) {

// *invariant:*

// R: fr contains a list of Persons in friends[0..friends.size()-1] that also appear in enemies.
```
return fr;
```
}

Potentially useful Vector methods, for v a variable with apparent type Vector.		
	`Vector()`	Constructor for an empty Vector —no objects in it
`void`	`v.add(p)`	Append object p to Vector v's list of objects
`int`	`v.size()`	The length of Vector v's list of objects
Object	`v.get(i)`	Return the object at position i in v
`boolean`	`v.contains(ob)`	= "Vector v's list contains ob, according to method ob.equals"

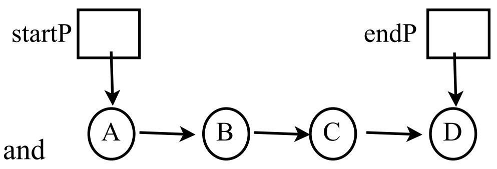
LAST NAME______________________________first name____________________________ Cornell Net id_____________

Question 2 (25 pts**) Recursion.** Suppose class `Person` contains no public fields and *only* the following two public methods (bodies omitted):

```
/** = this Person's best male friend (null if none) */
public Person getMBF() {...}
```

```
/** = this Person's best female friend (null if none) */
public Person getFBF() {...}
```

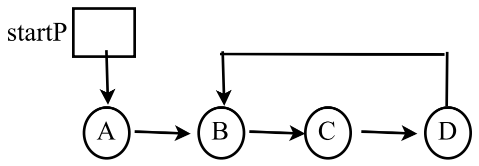
The purpose of the function specified below is to determine whether `Person startP` has a best male friend who has a best male friend who ... who has a best male friend who is `Person endP`. This is illustrated in the diagram to the right, where the objects of class `Person` are written as circles and each arrow denotes the name of the object to which it points.



`startP`, who is `A`, has best male friend `B`, who has best male friend `C`, who has best male friend `D`, who is `endP`. We could write this path as (`startP`, `B`, `C`, `endP`) or (`A`, `B`, `C`, `D`).

A `Person` can be their own best friend (but need not be).

The recursive function must watch out for the situation shown to the right. There is a cycle, and the recursive function won't terminate if it follows this cycle endlessly. The purpose of parameter `ignore` is to contain `Person`'s that should not be looked at to prevent getting trapped in a cycle.



/** = "there is a path of male best friends from startP to endP that does not contain a Person in list ignore". (Note: If startP is the same as endP, that counts as a path.)

Precondition: startP, endP, and ignore are not null; startP and endP are male, and startP and endP are not in ignore.

*/

public static boolean malePathTo(Vector<Person> ignore, Person startP, Person endP) {

/* key recursive insight: if there is a "malePath" (startP, B, ..., endP) where startP and endP are different, then there is a "malePath" (B, ..., endP) that does not include startP. */

}

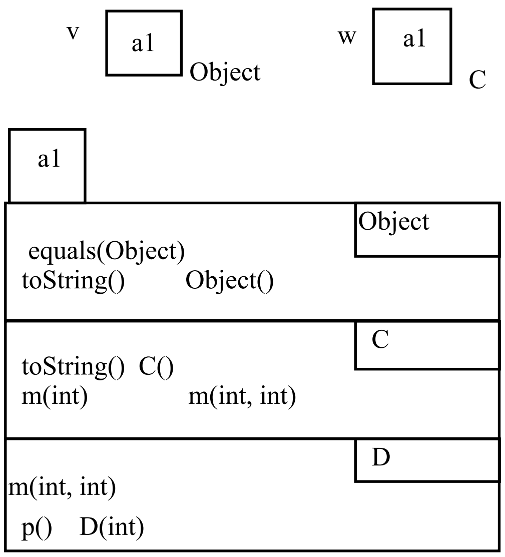
LAST NAME______________________________first name____________________________ Cornell Net id_____________

Question 3 (20 pts) **Methods and OO**

(a) Consider the diagram of two variables and an object, to the right. Each variable is annotated with its apparent class. Below are four method calls; circle those that are legal.

v.m(5) w.m(5) v.p() w.p()

(b) Using variable v, write a legal expression that calls method p of the object. You may have to use casting.



(c) Methods equals. They say that a person is known by the company they keep. Let's take this adage to heart in writing an equals-like method for class `Person`.

Assume class `Person` has exactly two fields, both of type `Person`: `fbf` and `mbf`, holding the person's best female friend and best male friend, respectively.

Write the body of this method, to be placed in `Person`:

/** ="p is a Person, with the same best female friend and same best male friend as this person."

Notes: We count best friends that are null as the same (so two Persons who both have no best friends are viewed as equal). However, this.equalsX(null) is false. */

public boolean equalsX(Object p) {

}

LAST NAME____________________________first name__________________________ Cornell Net id____________

Question 4 (28 pts) **Arrays, classes, subclasses.** Complete the following skeletons for three classes: ***abstract*** class `Mammal` and two subclasses `Dog` and `Platypus` of `Mammal` given on this page and the next. Assume that platypuses (platypi?) are the only species of mammals that lay eggs.

Fill in not just method bodies but also incomplete headers, missing methods (include specifications!), and so forth. Follow any directions given in the comments in the skeletons.

The statements/expressions given in the box on the right should be legal (allowed by the compiler). Also, `m.getBreed()` should be *illegal*, since not all mammals have breeds.

You must place methods optimally; for instance, it is a mistake to place a method in `Dog` or `Platypus` that should be in `Mammal`. Do *not* write any methods not called for in this question (for instance, don't write `toString` functions or a getter for field `noises`).

For `m` of type `Mammal`, `d` of type `Dog`, and `p` of type `Platypus`, these should be legal:

p.getNoise(1) [= "no such noise"]

d.getNoise(1) [= "woof"]

d.laysEggs() [= false]

p.laysEggs() [= true]

d.getBreed()

m= p;

/** An instance is a Mammal */ // don't forget to complete the "header" for this class

________________________________ Mammal ____________________ {

private String[] noises; // list of noises this mammal can make. Cannot be null, can be empty

/** Constructor: a Mammal making the sounds in noises (which can be empty, but not null). */
public Mammal(String[] noises) {

}

/** = the ith kind of noise this Mammal can make, or "no such noise" if this Mammal makes fewer than i noises. Precondition: 1 <= i */
public _________________getNoise(**int** i) {

// recall: the ith element of an array is stored in entry i-1 (if it exists)

}

}

LAST NAME_____________________________first name___________________________ Cornell Net id_____________

0	___________	out of 02
1	___________	out of 25
2	___________	out of 25
3	___________	out of 20
4	___________	out of 28
Total	________	out of 100

```
/** An instance is a purebred dog. */          // Don't forget to fill in the
                                               // "header" for this class.

_________________________ Dog ___________________________  {

    private String breed; // breed of this Dog.  Cannot be null or "".

    // declare a static String array variable  named nArr initialized with
    // an array containing "woof" and "arf".  Use it in the constructor.



    /** Constructor: a new Dog of breed b.
        Precondition: b has length > 0.*/
    public Dog(String b) {


    }
    /** = breed of this dog */

    public ______________________ getBreed() {


    }




}
/** An instance is a Platypus. */                 // Don't forget to fill in the "header" for this class.

__________________________Platypus ___________________  {

    private static String[] nArr= {}; // Noises platypi make (they are silent). Use nArr in the constructor.


    /** Constructor: a new platypus */
    public Platypus() {


    }




}
// Note:  did you remember to put method layEggs in at least one place?
```