LAST NAME GRIES first name David Cornell Net id djg17

Question 1. /** See prelim for the spec. */
```
public static Vector<Person> frenemies(
   Vector<Person> friends, Vector<Person> enemies) {
   Vector<Person> fr= new Vector<Person>();
   /* inv: fr is a list of Persons in friends[0..i-1]
           that also appear in enemies. */
   for (int i= 0; i < friends.size(); i= i+1) {
         Person f= friends.get(i);
         if (enemies.contains(f)) {
            fr.add(f);
         }
   }
   /* R: fr is a list of Persons in
         friends[0..friends.size()-1] that
         also appear in enemies. */
  return fr;
}
```

Question 2. /** See prelim for the spec. */
```
public static boolean malePathTo(
               Vector<Person> ignore,
               Person startP, Person endP) {
    // base case: length-0 path
    if (startP == endP)
        { return true; }

   Person mbf= startP.getMBF();

   // base case: path ends too soon to contain endP
   if (mbf == null) {
         return false;
   }

   // recursive case: check path lengths >=1
   ignore.add(startP);
   if (ignore.contains(mbf)) {
          return false;
   }
   // mbf not in ignore, so spec is satisfied
   return malePathTo(ignore, mbf, endP);
}
```

Question 3. (**a**) (the legal ones are in boxes):

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

(b) ((D) v).p()

(c)
```
/** ="p is a Person, with same best female
      friend and best male friend as this person."
      Best friends that are "null" are the same
       (so two Persons with no best friends are
       equal). But this.equalsX(null) is false. */
public boolean equalsX(Object p) {
    if (!(p instanceof Person))
        { return false; }
    Person newp= (Person)p;
    return newp.mbf == mbf && newp.fbf == fbf;
}
```

Question 4.

Two possible solutions.

The first follows the general object-oriented principle of making method laysEggs() abstract in order to force all subclasses to override it:

```
  /** An instance is a Mammal */
public abstract class Mammal {
   private String[] noises; // list of noises this
                // Mammal can make. Never null
   /** Constructor: a Mammal that makes the
                       sounds listed in noises.
       Precondition: noises is not null
                       (it can be size 0). */
   public Mammal(String[] noises) {
      this.noises= noises;
   }

   /** = the ith kind of noise this Mammal can
       make or the String "no such noise" if the
       Mammal makes fewer than i noises.
     Precondition: 1 <= i. */
   public String getNoise(int i) {
      // recall: the ith element of an array is
      //           stored in entry i-1 (if it exists)
      int e= i-1;
      if (e >= noises.length)
            { return "no such noise"; }
      else { return noises[e]; }
   }

   /** = "this Mammal species lays eggs */
   public abstract boolean laysEggs();
}

/** See prelim for the spec. */
public class Dog extends Mammal {
   private String breed; // Dog's breed. Not null or "".
   private static String[] nArr= {"woof", "arf"};
   /** Constructor: a new Dog of breed b.
       Precondition: b has length >0. */
   public Dog(String b) {
      super(nArr);
      breed= b;
   }

   /** = breed of this dog */
   public String getBreed() {
      return breed;
   }

   // SEE OVER


 /** = "this species lays eggs" */
   public boolean laysEggs()
      { return false; }
}
```

```
/** An instance is a Platypus, a silent mammal that lays
eggs. */
public class Platypus extends Mammal {
   private static String[] nArr= {}; // platypi are silent

   /** Constructor: a new platypus */
   public Platypus()
      { super(nArr); }

   /** = "this species, platypus, lays eggs" */
   public boolean laysEggs()
      { return true; }
}
```

In the second solution, we decide to violate the general object-oriented principle in the particular case of Mammal because almost every subclass should have the exact same implementation of laysEggs(): so, we decide instead to encode the default behavior of laysEggs() in Mammal, making it the responsibility of subclass authors to override the default behavior.

Thus, in this solution, laysEggs would be a non-abstract method in Mammal that always returns false, and Dog would not have a laysEggs method. The Platypus laysEggs method remains as is.