Cornell net id _____ **Name** _____

CS 1110 Prelim 2 *Grades are expected to be posted by 1AM tonight* 12 March 2009

This 90-minute exam has 6 questions (numbered 0..5) worth a total of 100 points. Spend a few minutes looking at all the questions before beginning. Use the back of the pages if you need more space.

Question 0 (2 points). Fill in the information, legibly, at the top of *each* page. (Hint: do it now.)

Question 1 (20 points). Write the body of recursive function `firstC`, whose specification and header appear below. In writing it, think of the base cases first —parameter values where the answer is easy to calculate because the value is small. Then handle the other, recursive, case(s).

Do not use loops. Use only recursion.

One additional restriction, which, incidentally, should help you home in on a simple solution: The only String functions you should use are `charAt`, `length`, and `substring`.

```
/** = number of times the first char of s appears at the beginning of s.
      Example: firstC("bbbcb$b") = 3.
      Example: firstC("bcb$bbb") = 1.
      Precondition: s contains at least one character. */
public static int firstC(String s) {




}
```

Cornell net id _____ **Name** _____

Question 2 (20 points). Class `PhD`, below, can be used maintain information about the intellectual genealogy of PhDs —about PhD advisors. For example, on the website www.genealogy.ams.org, Gries traces his roots back to Leibniz. Lee's genealogy is not known that far back because the website does not have the PhD advisor of her intellectual great grandfather, one of the famous founders of AI, Seymour Pappert.

We have elided the constructor body to save space. Note that *class `PhD` has no getter methods*, and none should be inserted.

2.(a). Explain why one would want to make a class abstract. Then write on the class definition for PhD whatever is necessary to make it abstract.

2.(b) Write the body of function `equals` to implement its specification.

2.(c). Insert after the declaration of function `equals` a declaration of an *abstract* function `hasJob()`, whose purpose is to return a String that indicates where this PhD works. Don't forget to specify the function. In the space below this sentence, explain the purpose of making the function abstract.

```
/** An instance maintains information about a PhD. */
public class PhD {
    private String name; // Person's name: <first> <last>
    private int year;    // Year PhD received
    private String univ; // University awarding the PhD
    private PhD advisor1;// Advisor (null if unknown)
    private PhD advisor2;// Advisor (null if unknown)
                         // Either advisor or both may be unknown.

    /** Constructor: a PhD with name n, year y, university u and up to two advisors a1and a2.
                    Precondition: Either of a1 and a2, and perhaps both, can be null. */
    public PhD(String n, int y, String u, PhD a1, PhD a2) {

    /** = "p is an object of class PhD, and the person it represents has the same name as
           this PhD". */
    public boolean equals(Object p) {









    }



}
```

Cornell net id _____ **Name** _____

Question 3 (20 points) You can use the back of the previous page to answer some of these questions.

At the bottom of the page is a subclass `Faculty` of `PhD`.

(a) Write the bodies of the constructor and `equals`. Remember, class `PhD` has no getter methods.

(b) For each of the functions `hasJob` and `rank` at the bottom of class `Faculty`, indicate whether or not class `Faculty` will compile if it is removed, and if not, explain why.

(c) Suppose `p` is as declared in the box to the right. Consider the two calls in the box to the right. For each one:

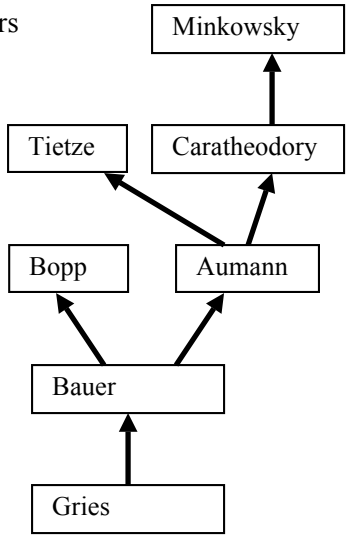
1. Indicate whether the call is syntactically legal or illegal.
2. If it is legal, explain what function is called.

```
PhD p;
…

… p.hasJob()

… p.rank()
```

```
/** An instance is a PhD that is a faculty member */
public class Faculty extends PhD {
    private String univ; // The university at which this person works
    private String rank; // Person's rank --one of "Asst", "Assoc", "Prof"
    /** Constructor: a faculty member with name n, year of PhD y from university u,
        now working at university uw at rank of r The faculty member's advisors were
        a1 and a2 (null if not known or nonexistent).
        Precondition: r is one of "Asst", "Assoc", "Prof". */
    public Faculty(String n, int y, String u,
                   String uw, String r, PhD a1, PhD a2) {




    }
    /** = f is of class Faculty, and f's name, university at which they work,
          and rank are the same as this one.*/
    public boolean equals(Object f) {




    }
    /** = the job this PhD now holds */
    public String hasJob() { … }
    /** = the faculty rank of this Faculty member */
    public String rank() { … }
}
```

Cornell net id _____ **Name** _____

Question 4. (20 points). Part of Gries's intellectual ancestral tree appears to the right. Each arrow goes from a PhD advisee to an advisor. So, Gries's PhD advisor was Bauer, and Bauer had two PhD advisors: Bopp and Aumann. If you take much math, you will run across names like Minkowsky and Caratheodory.



Each of the boxes represents an object of class `PhD`, and an arrow represents the contents of either field `advisor1` or `advisor2` of the object.

Write the function that is specified below, which is to be declared in class `PhD`. It computes (and returns) the total number of intellectual ancestors that a `PhD` has. For example, for the `PhD` object for "Gries" and ancestors as shown to the right, the answer is 6.

Do not use a loop. Use recursion, in that the function will call the same function (but in other objects).

```
/** =  Total number of (known) intellectual ancestral advisors of this PhD.
      That is: the number of advisors plus the number of the advisors' advisors
      plus the number of the advisor's advisor's advisors, etc.
        Precondition: no two people in the ancestral tree share an advisor,
        and no person in the ancestral tree has advisor1== advisor2 unless
        both are null.*/
public int ancestors() {




}
```

Cornell net id _____ **Name** _____

0	_____ out of 02
1	_____ out of 20
2	_____ out of 20
3	_____ out of 20
4	_____ out of 20
5	_____ out of 18
Total	_____ out of 100

Question 5. (18 points) Partial credit for this question cannot be given unless you draw all variables (first) and then actually execute by hand, for example drawing all objects created.

Consider the two classes given at the bottom of this page. Suppose the following assignments have been executed (we also give the declarations of the variables being assigned):

```
A a= new B(2);
A b= new A(5);
A c= b;
b.bV= 7;
b= a;
```

(a) Evaluate the following expressions and write their values to the right of the expressions.

1. `b instanceof A`

2. `b instanceof B`

3. `c instanceof A`

4. `c instanceof B`

5. `A.add(a, b)`

6. `A.add(b, c)`

(b) In the expression `A.add(a, c)`:

1. What are the apparent and real classes of `a`?

2. What are the apparent and real classes of `c`?

```
public class A {
   public int bV;

   public A (int n) {
     bV= n;
   }

   public int m() {
     return 2 * bV;
   }

   public static int add(A x, A y) {
     return x.m() + y.m();
   }
}
```

```
public class B extends A {
   public B(int n) {
     super(n+1);
   }

   public int m() {
     return bV;
   }
}
```