Cornell net id ____________________________ **Name** __________________________________

Section day ______________________________ **Section time** ____________________________

CS 1110 Prelim 2 *Grading and posting of grades done Wednesday night* 21 October 2008

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).
(a) Evaluate the call `Tester.f(1)` (see the class at the bottom of the page. Note that we have labeled each statement to help you identify them). As you evaluate the call, draw frames for all function calls that are evaluated. Stop executing just *before* executing one of the statements labeled `m3` and `m4`, whichever comes first. Do not continue executing. That way, we will be able to see that you drew frames and executed correctly. Please be sure to maintain the instruction counters of the frames.

(b) When is parameter `k` created? When is local variable `p` created? When is local variable `s` created?

```
public class Tester {
    public static int f(int k) {
       int p= k+1;
       f1: if (k == 1) {
              f2: return f(p);
       }
       f3: return m(p);
    }

    public static int m(int n) {
        m1: if (n != 1) {
                int s;
                m2: s= n * n;
                m3: return s;
        }
        m4: return n;
    }
```

Cornell net id ______________________ **Name** ____________________________

Section day _________________________ **Section time** _______________________

Question 2 (18 points). Below is a class `Pres`, which contains information about a president of the US. We have omitted many fields, the constructors, and other methods. We put in only the fields that you need to answer this question.

Complete the bodies of the three functions shown in class `Pres`. In function `equals`, do not simply test whether **`this`** and `p` are equal; implement it as specified.

```
/** An instance maintains info about a president */
public class Pres {
    private int sYear;    // Was president in years
    private int eYear;    // sYear..eYear

    private String name; // name of president

    private Pres prev;    // the previous president (null if none)

    /** = the previous president (or null if none)*/
    public Pres getPrevious() {


    }

    /** = number of presidents before this one */
    public int before() {






    }

    /** = "p is a Pres and has the same starting and ending year and
             the same name as this President" */
    public boolean equals(Object p) {














    }
}
```

Cornell net id ______________________ **Name** ____________________________

Section day ________________________ **Section time** ______________________

Question 3 (20 points) At the bottom of the page are two subclasses of class `Pres` of question 2. We show only the fields and methods that you need to answer this question.

(a) Complete function `beforeInSameParty` in both classes**.**

(b) Suppose variable `p` is as declared below, and suppose `p` contains the name of an object of one of the classes `Pres`, `Dem` , and `Rep`.

`Pres p;` // declaration of variable `p`

Consider the call `p.beforeInSameParty()`. Circle the statement below that you think is correct:

1. The call is syntactically illegal.
2. The call is syntactically legal, but what happens when it is evaluated depends on the real class of `p` —whether it contains a `Pres`, `Dem` , or `Rep` object. It may or may not result in an error.

(c) If you circled 1 above, write down what you would place in class `Pres` to make the call legal.

If you circled 2 above, rewrite the call, perhaps using a conditional expression, so that it produces 0 if the call would produce an error and the proper value if the call can be evaluated.

```
/** An instance has info about a Democratic
      president */
public class Dem extends Pres {
   // previous Democratic pres. (null if none)
   private Pres prevDem;

   /** = number of Democratic presidents
          before this one */
   public int beforeInSameParty() {




   }
```

```
/** An instance has info about a Republican
      president */
public class Rep extends Pres {
   // previous Republican pres. (null if none)
   private Pres prevRep;

   /** = number of Republican presidents
          before this one */
   public int beforeInSameParty() {




   }
```

Cornell net id ___________________________ **Name** _________________________________

Section day _____________________________ **Section time** ____________________________

Question 4. (20 points). In answering this question, it will help to draw the objects that are created and the frames for the function calls that are being evaluated. You don't have to do this, but it should help.

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):

```
Two a;   a= new Two(2);
Two b;   b= new Two(3);
Two c;   c= new One(4);
Two d;   d= b;
d.bV= 5;
```

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

1. `Two.add(a, b)`

2. `Two.add(b, d)`

3. `Two.add(a, c)`

4. `a` **`instanceof`** `Two`

5. `a` **`instanceof`** `One`

6. `c` **`instanceof`** `One`

7. `c` **`instanceof`** `Two`

8. ((One)`c).altMult()`

(b) In the expression `Two.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 Two {
   public int bV;

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

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

   public static int add(Two a, Two b) {
     return a.mult() + b.mult();
   }
}
```

```
public class One extends Two {
   public One(int n) {
     super(n);
   }

   public int mult() {
     return bV;
   }

   public int altMult() {
     return super.mult();
   }
}
```

Cornell net id ________________________ **Name** ______________________________

Section day ________________________ **Section time** ________________________

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

Question 5 (20 points). Write the body of recursive function `merge`, 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 a parameter is as small as possible. Then handle the other, recursive, cases.

Do not use loops. Use only recursion.

```
/** = the characters of s1 and s2, in alphabetical order.
    Precondition: the characters of s1are already in alphabetical
    order and the characters of s2 are already in alphabetical order.

    Examples: merge("ab", "") = "ab".
    merge("abbce", "cdg") = "abbccdeg" */
public static String merge(String s1, String s2) {







}
```