Cornell net id ________________________ Name ___________________________

Section day ________________________ Section time _______________________

CS 100J Prelim 3 15 April 2008

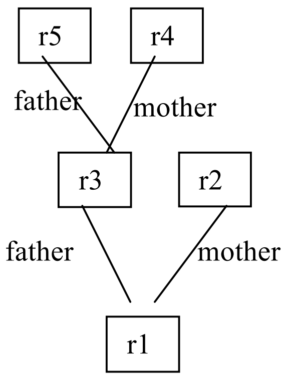
This 90-minute exam has 6 questions (numbered 0..5) worth a total of 100 points. Spend a few minutes looking at all questions before answering any. Budget your time wisely. Use the back of the pages, if you need more space. We have a stapler at the front of the room, so you can tear the pages apart.

Question 0 (2 points). Write your netid and your name, legibly, at the top of each page (Hint: do it now).

Question 1 (18 points) Recursion. Write function `hasName`, defined below in class `Rhino` —we put only the fields and methods needed for this question. Do not use loops; you must use recursion. Do not write any other methods.

To the right is a possible tree for a rhino, showing the rhino (`r1`) and its known ancestors.

Note that `hasName` is not a static function; it appears in every object of class `Rhino`.



```
public class Rhino {

   private Rhino father;  // this Rhino's father (null if unknown)
   private Rhino mother; // this Rhino's mother (null if unknown)
   private String name;  // this Rhino's name
   /** = "this rhino or one of its ancestors has name n. */
   public boolean hasName(String n) {




   }
```

Cornell net id ________________________ Name ___________________________

Section day ________________________ Section time ________________________

Question 2 (20 points). Exceptions.

(a) What is the output of the call `P3.foo()`, given the following definition for class `P3`?

```
class P3 {
    private static int calls = 0;

    public static void foo() {
      try {
         bar();
         System.out.println("first bar done");
         bar();
      }
      catch (Exception e) {
         System.out.println("Exception!!");
      }
    }

    private static void bar() throws Exception {
      if (calls > 0)
         System.out.println("Call number " + calls);
      else
              throw new Exception();
    }
}
```

(b) On the back of the previous page, write a subclass `BadNumberException` of class `Exception` (which is a subclass of `Throwable`). It needs the usual two constructors.

(c) The specification of the following function has a precondition. (The function calculates the greatest common divisor `gcd(b,c)` of `b` and `c`.) To make it easier to write robust code that calls this function, change the function to throw a `BadNumberException` (with a suitable detail message) if the precondition is false — first change the specification; then change the body. Use the back of the previous page for your answer. In writing the body, you don't have to copy the cde shown below, but you *do* have to show where it goes.

You don't have to understand completely how the body works, but if you are interested, note that it rests on two properties: (1) `gcd(b, c)` = `gcd(c, b)`, and, for `c` > `b`, `gcd(b, c)` = `gcd(b, c-b)` because anything that divides `b` and `c` also divides `b` and `c-b`, and vice versa.

```
/** = the greatest common divisor of x and y.
      Precondition: x > 0 and y > 0. */
public static int GCD(int x, int y) {
      int b= x; int c= y;
      // invariant: gcd(x, y) = gcd(b, c) and b > 0 and c > 0
      while (b != c) {
         if (b < c) c= c - b;
         else b= b - c;
      }
      return b;
}
```

Cornell net id ________________________ Name ___________________________

Section day ________________________ Section time ______________________

Question 3 (25 points). For-loops and arrays.

A magic square is a square where each row and column adds up to the same number (sometimes, one also includes the diagonals, but for this problem, we won't). For example, in the following 5-by-5 square, the elements in each row and column add up to 70:

```
18 25  2  9 16
24  6  8 15 17
 5  7 14 21 23
11 13 20 22  4
12 19 26  3 10
```

(a.) Write the body of the following function —so that it returns **true** if all the rows sum to `sum` and returns **false** otherwise. The body should use the given invariant.

```
/** = "all the rows of square array sq sum to sum".
      Precondition: sq is not null and indeed is a square array. */
public static boolean areMagicRows(int[][] sq, int sum) {

      // invariant: each row 0..k-1 of  sq sums to sum.
      for (int          ;                    ;                   ) {
            // Return false if row k does not sum to sum.









      }
      // postcondition: each row  0..sq.length-1 sums to sum
      return              ;
}
```

Cornell net id ________________________ Name ___________________________

Section day ________________________ Section time _______________________

(b.) Now complete function `isMagicSquare`, whose specification is given below. Here are some ground rules.

1. There should be one set of nested for-loops —do not write more for-loops or while-loops.

2. Do not use recursion.

3. We do not give you a postcondition or invariant for the loops. You do not have to write them, but it may help you to do so.

```
/** = "sq is a magic square with sum sum".
        Precondition: sq is not null and indeed is a square array. */
public static boolean isMagicSquare(int[][] sq, int sum){































}
```

Cornell net id ________________________ Name ___________________________

Section day ________________________ Section time _______________________

Question 4 (12 points). Classes.
Answer the follow questions in a few sentences:

(a) Explain the "Inside out rule".

(b) Explain the two uses of **this** and **super** in Java.

(c) What is meant by "overriding a method"? How does one call an overridden method from within the class that does the overriding?

Cornell net id ________________________ Name ___________________________

Section day ________________________ Section time _______________________

0	___________	out of 02
1	___________	out of 18
2	___________	out of 20
3	___________	out of 25
4	___________	out of 12
5	___________	out of 23
Total	________	out of 100

Question 5 (23 points). Algorithms.

(a) On the back of the previous page, write down the four loopy questions used in developing or understanding a while loop.

(b) Consider an array segment `d[p..q-1]` of **`char`**s, where each **`char`** is either '`r`', '`w`', or '`b`'. Write an algorithm (in Java) to swap the elements of `d[p..q-1]` so that all the 'r's are first, then the 'w's, and finally the 'b's. We expect you to:

(1) Draw a precondition for the algorithm; (2 points)

(2) Draw a postcondition for the algorithm; (4 points)

(3) Draw an invariant for a while loop (with initialization) for the problem; this invariant should lead to an algorithm that makes at most one swap per iteration of its while loop. (6 points)

(4) Write the while loop with initialization, using the four loopy questions. The loop must be consistent with the invariant that you drew. To swap to array elements, write simply "swap … and …". (11 points).

Precondition:

Postcondition:

Invariant:

Code: