CSCI 135 Exam #1 **Name: _____**
Fundamentals of Computer Science I
Fall 2013

This exam consists of 5 problems on the following 7 pages.

You may use your two-sided hand-written 8 ½ x 11 note sheet during the exam. No calculators, computers, or communication devices of any kind are permitted.

If you have a question, raise your hand and I will stop by. Since partial credit is possible, **please write legibly and show your work**.

Problem	Points	Score
1	8	
2	10	
3	19	
4	16	
5	10	
Total	63	

1. Loops, input (8 points). Consider the following program:

```
public class Prob1
{
   public static void main(String [] args)
   {
      int i;
      for (i = 0; i < args.length - 1; i++)
      {
         double val1 = Integer.parseInt(args[i]);
         double val2 = Integer.parseInt(args[i + 1]);

         if (val1 > val2)
            break;
      }
      if (i == args.length - 1)
         System.out.println("yes");
      else
         System.out.println("no");
   }
}
```

Give the output of `Prob1` for each of the following commands. If the program would crash, write "error":

Command	**Program output**
`% java Prob1 3 7 9`	`"yes"`
`% java Prob1 -5 0 4 2 10`	`"no"`
`% java Prob1`	`"no"`
`% java Prob1 3.5 4.5 5.5`	`error`
`% java Prob1 8`	`"yes"`
`% java Prob1 1 1 1 1 1 1 1 1`	`"yes"`

2. Multiple choice (10 points, 2 points each). Circle the ***best*** single answer.

I. Consider the following code fragment:

```
int i = 5 / 10;
System.out.println(i);
```

What is the output of the `System.out.println`?

a) 0
b) 0.5
c) 1
d) none of the above

II. Consider the following code fragment:

```
int i = 2;
i++;
++i;
i = i + 1;
System.out.println(i);
```

What is the output of the `System.out.println`?

a) 2
b) 3
c) 4
d) 5

III. The **`private`** keyword when added to the declaration of an instance variable of a class helps enforce which of the following principles?

a) data encapsulation
b) immutability
c) avoiding repeated code
d) divide-and-conquer

IV. The `Fraction` class we developed in class had a **`private`** helper method named `reduce` that reduced the fraction object to lowest terms. This was done for which of the following reasons?

a) to allow sorting of `Fraction` objects
b) to make the class immutable
c) to avoid repeated code from appearing in multiple methods (e.g. in both `add` and `substract`).
d) to serve as a base case for the recusion

V. The **`this`** keyword can be used in an instance method to achieve which of the following?

a) creating a new instance of the class
b) specifying the use of an instance variable when there is an identically named local or parameter variable
c) avoiding repeated code
d) hiding implementation details from clients of the class

3. Objects, standard input, arrays (19 points). Consider the following class that represents a charged particle:

```
public class Charge
{
   private double rx, ry;   // position
   private double q;        // charge

   public Charge(double x0, double y0, double q0)
   {
      double rx = x0;
      double ry = y0;
      double q = q0;
   }
}
```

a) What is wrong with the above code?

By declaring the type of the rx, ry, and q variables in the Charge constructor, we have created local instances with the same name as the instance variables. Thus the instance variables are not set by the constructor. This can be fixed by removing the double before each variable assignment statement.

b) Assume you fixed the problem in part a. You are now developing a client program that reads via standard input a list of charged particles. Here is an example file `3charge.txt` with a positively charged particle at (0.5, 0.7), a neutral particle at (1.0, 0.1), and a negative particle at (-2.25, -10.5):

```
 3
 0.5    0.7  +1.0
 1.0    0.1   0.0
-2.25 -10.5  -1.0
```

For now, the program just reads and stores all the charged particles in an array. Below is the skeleton of the client program. In the empty boxes, write the letter of the code fragments that combine to create a correct implementation. ***Not all letters will be used and each letter can only be used once.***

```
public class ChargeClient
{
   public static void main(String [] args)
   {
      [ D ] = StdIn.readInt();

      Charge [] particles = [ F ];

      [ Q ] ( [ A ] = 0; i < N; i++)
      {
         double x = StdIn.readDouble();
         double y = StdIn.readDouble();

         double q = [ L ];

         particles[i] = [ I ];
      }
   }
}
```

A. `int i`
B. `final int i`
C. `String N`
D. `final int N`
E. `Charge[N]`
F. `new Charge[N]`
G. `new Charge()`
H. `new Charge(N)`
I. `new Charge(x, y, q)`
J. `Charge(x, y, q)`
K. `StdIn.readInt()`
L. `StdIn.readDouble()`
M. `!StdIn.isEmpty()`
N. `Double.parseDouble(args[0])`
O. `Double.parseDouble(args[1])`
P. `while`
Q. `for`
R. `do while`
S. `null`

(continued on next page)

3. Objects, standard input, arrays (continued)

c) In an effort to test your program from part b, you added the following line of code to the end of the loop:

```
System.out.println(particles[i]);
```

Unfortunately, you are getting strange looking output. Here is an example run using the example input file:

```
% java ChargeClient < 3charge.txt
Charge@128ef465
Charge@674f1c67
Charge@7ad1e32d
```

What is the name of method you need to implement to get the above `System.out.println` to produce more sensible output?

```
toString()
```

What class should this method be added to?

```
Charge
```

d) Match the description on the left with the signature of an instance method of `Charge` that would ***best*** provide the stated functionality. Each letter will be used ***exactly once***.

Return the net force exhibited on a given particle by a list of zero or more other particles. __E__	A. `public boolean foo1()` B. `public boolean foo2(double a)` C. `public boolean foo3(Charge a)` D. `public double foo4(Charge a)` E. `public double foo5(Charge [] a)` F. `public Charge()` G. `public Charge(Charge a)` H. `public Charge(Charge a, Charge b)`
Return a new particle that has the same location and charge of an existing particle. __G__	
Return a new particle that has a location and charge that is the average of two existing particles. __H__	
Return a new particle with some default location and charge. __F__	
Return the Euclidean distance between two particles. __D__	
Determine whether two particles have the same charge. __C__	
Determine whether a particle is negatively charged. __A__	
Determine whether a particle's position is within a given radius of the origin. __B__	

4. Static methods (16 points). Consider the following library containing a number of static methods:

```
public class NumHelpers
{
   public static int foo(int a, int b)
   {
      a = a + b;
      return a;
   }

   public static boolean foo(int n)
   {
      return (n % 2 == 0);
   }

   public static int foo(int a, int b, int c)
   {
      return Math.max(a, Math.max(b, c));
   }

   public static int foo(int target, int [] vals)
   {
      int result = 0;
      for (int i = 0; i < vals.length; i++)
      {
         if (vals[i] == target)
            result++;
         System.out.printf("TRACE %d %d %d %d\n", i, vals[i], target, result);
      }
      return result;
    }
}
```

a) Assume you are developing code a program `Calculator.java` that makes use of the `NumHelpers` library. Mark whether each of the following lines is valid (i.e. compiles) or invalid (i.e. causes a compile error).

Code	Valid	Invalid
`boolean r1 = NumHelpers.foo(20);`	X	
`boolean r2 = foo(20);`		X
`int r3 = NumHelpers.foo(10.0, 20.0);`		X
`boolean r4 = (NumHelpers.foo(-5, 23, 928) > 5.0);`	X	
`int r5 = NumHelpers.foo(Integer.parseInt("2"), Integer.parseInt("3"));`	X	
`int r6 = NumHelpers.foo(1, 2, 3, 4);`		X

(continued on next page)

4. Static methods (continued)

b) For each of methods in `NumHelpers` implemented on the previous page, briefly describe what it does (i.e. given its input what does the output of the method represent).

`public static int foo(int a, int b)`
 	
`Adds a and b and returns the sum`

`public static boolean foo(int n)`
 	
`Determines if n is an even number`

`public static int foo(int a, int b, int c)`
 	
`Returns the maximum of the three numbers a, b and c`

`public static int foo(int target, int [] vals)`
 	
`Counts the number of items in the vals array that are equal to the target value`

c) The following code fragment makes use of the first method in `NumHelpers`. What does it output?

```
int a = 5;
int b = NumHelpers.foo(1, 2);
System.out.println("a = " + a + ", b = " + b);
```

`"a = 5, b = 3"`

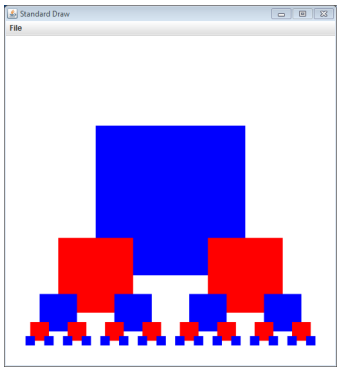
d) The following code fragment makes use of the last method in `NumHelpers`. Fill in the table giving the values output by the TRACE `System.out.printf` statement in the method (you may not need all rows). As the `printf` outputs 4 values, fill out the table cells showing these values for each loop iteration on the given input.

```
int [] data = {1, 4, 8, 4};
NumHelpers.foo(4, data);
```

TRACE	**0**	**1**	**4**	**0**
TRACE	**1**	**4**	**4**	**1**
TRACE	**2**	**8**	**4**	**1**
TRACE	**3**	**4**	**4**	**2**
TRACE				
TRACE				

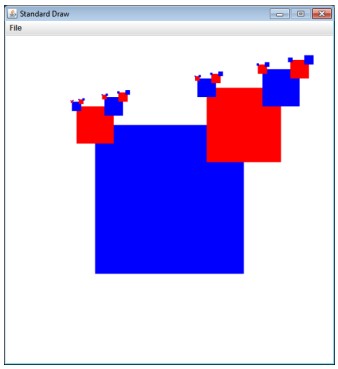
5. Recursive drawing (10 points). Label each `StdDraw` output with the letter of the recursive method that generated it, each image was generated by ***exactly one of the methods***. Each method was initially called with the same parameters: *drawX*`(n, 0.5, 0.5, 0.25);` One of the recursive methods generates a stack overflow due to unbounded recursion, ***circle this method***.

_A_



```
static void drawA(int n, double x, double y, double s)
{
   if (n <= 0) return;
   if (n % 2 == 0)
      StdDraw.setPenColor(StdDraw.RED);
   else
      StdDraw.setPenColor(StdDraw.BLUE);
   StdDraw.filledSquare(x, y, s);
   drawA(n - 1, x - s, y - s, s / 2.0);
   drawA(n - 1, x + s, y - s, s / 2.0);
}
```

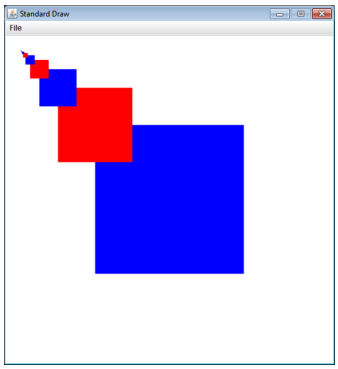
_D_



```
static void drawB(int n, double x, double y, double s)
{
   if (n % 2 == 0)
      StdDraw.setPenColor(StdDraw.RED);
   else
      StdDraw.setPenColor(StdDraw.BLUE);
   StdDraw.filledSquare(x, y, s);
   drawB(n - 1, x - s, y + s, s / 2.0);
   drawB(n - 1, x + s, y + s, s / 2.0);

}
```

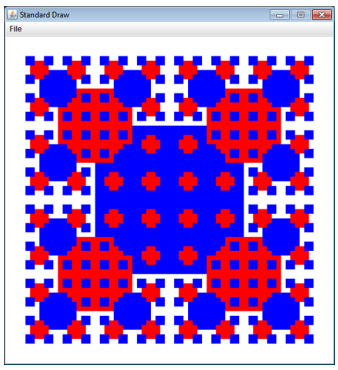
_B_



```
static void drawC(int n, double x, double y, double s)
{
   if (n <= 0) return;
   if (n % 2 == 0)
      StdDraw.setPenColor(StdDraw.RED);
   else
      StdDraw.setPenColor(StdDraw.BLUE);
   StdDraw.filledSquare(x, y, s);
   drawC(n - 1, x - s, y + s, s / 2.0);
   drawC(n - 1, x + s, y + s, s / 2.0);
   drawC(n - 1, x - s, y - s, s / 2.0);
   drawC(n - 1, x + s, y - s, s / 2.0);

}
```

_C_



```
static void drawD(int n, double x, double y, double s)
{
   if (n <= 0) return;
   if (n % 2 == 0)
      StdDraw.setPenColor(StdDraw.RED);
   else
      StdDraw.setPenColor(StdDraw.BLUE);
   StdDraw.filledSquare(x, y, s);
   drawD(n - 1, x - s, y + s, s / 4.0);
   drawD(n - 1, x + s, y + s, s / 2.0);
}
```