CSCI 136 Written Exam #1 **Name: _____**
Fundamentals of Computer Science II
Spring 2013

This exam consists of 5 problems on the following 6 pages.

You may use your double-sided hand-written 8 ½ x 11 note sheet during the exam. No computers, mobile devices, cell phones, or other 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	14	
3	14	
4	16	
5	14	
Total		

1. **Loops, Methods, Strings** (8 points). Consider the following program:

```
public class Prob1
{
   public static String mystery(String s)
   {
      String r = "";
      for (int i = s.length() - 1; i >= 0; i--)
         r += s.charAt(i);
      return r;
   }
   public static void main(String [] args)
   {
      String s = args[0];
      final int N = Integer.parseInt(args[1]);
      for (int i = 0; i < N; i++)
      {
         System.out.print(s + " ");
         s = mystery(s);
      }
   }
}
```

Below are four example executions of the program. Give the output produced by the program. If the given input would cause a runtime error, write "runtime error". If the given input would cause a stack overflow error, write "stack overflow".

Command line	Output
`% java Prob1 wang 2`	wang gnaw
`% java Prob1 racecar 3`	racecar racecar racecar
`% java Prob1 love -1000`	(no output)
`% java Prob1 love`	Runtime error

2. **Multiple choice** (2 points each). For each question, circle the ***ONE*** best answer.

a) All of the following lines of code result in the variable `i` being increased by one on the line following the given line ***EXCEPT***:

```
 I.  i++;
II.  ++i;
III. foo(i + 1);
IV.  i += 1;
 V.  i = i + 1;
VI.  foo(i++);
```

`foo` is a method with the signature `void foo(int val)`

b) Consider the following recursive method:

```
public static int foo(int n)
{
   if (n <= 1) return 3;
   return foo(n / 2) + foo(n / 2);
}
```

What is the value returned by the method call foo(4)?

```
 I.  0
II.  3
III. 4
IV.  12
```
V. No value, causes a stack overflow.

c) Consider the following code fragment:

```
ArrayList<Color> list = new ArrayList<Color>();
while (true)
   list.add(new Color(0, 0, 0));
```

What would eventually happen if you ran the above code?

I. The JVM would run out of heap memory.
II. The JVM would run out of stack memory.
III. The `ArrayList` will reach capacity and throws an `ArrayListCapcityExceeded` exception.
IV. The `Color` constructor would throw an exception due to repeated calls with the same parameters.
V. Trick question, the above code won't compile due to the infinite `while` loop.

d) If a reference variable is assigned the value `null`, which of the following in ***TRUE***:

I. The assignment results in an *immediate* `NullPointerException` exception.
II. If no other variables reference the same object, results in the Java garbage collector *immediately* freeing the memory associated with the object.
III. If no other variables reference the same object, results in the Java garbage collector freeing the memory associated with the object *at some future point in time.*
IV. Results in the Java garbage collector *immediately* freeing the memory associated with the object.
V. Results in the Java garbage collector freeing the memory associated with the object at *some future point in time.*

e) Consider the following class:

```
public class Cow
{
   private String  name    = "";
   private double  weight  = 0.0;
   private Picture image   = new Picture("cow.jpg");
   // ... implementation ... //
}
```

All Cow objects have different names and weights, but are all drawn with the same image. Which of the following would reduce the memory required by a program having an array of a thousand Cow objects, all having different names and weights?

I. Add the **final** keyword to all the instance variables.
II. Add the **static** keyword to all the instance variables
III. Add the **static** keyword to just the image instance variable
IV. Add the **synchronized** keyword to just the image instance variable
V. Change the access modifiers to **protected** instead of **private**.

f) Consider the following code that declares and creates a multi-dimensional ragged array:

```
double [][] d = new double[3][];
d[0] = new double[1];
d[1] = new double[2];
d[2] = new double[3];
```

Which of the following lists all the valid locations in the array d?

I.
```
d[0][0], d[0][1]
d[1][0], d[1][1], d[1][2]
d[2][0], d[2][1], d[2][2], d[2][3]
```

II.
```
d[0][0]
d[1][0], d[1][1]
d[2][0], d[2][1], d[2][2]
```

III.
```
d[0][0], d[0][1], d[0][2]
d[1][0], d[1][1], d[1][2]
d[2][0], d[2][1], d[2][2]
```

IV.
```
d[0][0], d[0][1], d[0][2]
d[1][0], d[1][1]
d[2][0]
```

g) You are developing a client-server program using Java sockets. The client program establishes a connection to the server and has a multi-step conversation over the course of many seconds. What problem would result if you implement a single-threaded server that utilizes a single Socket object?

I. The client would not be able to obtain the IP address of the server via DNS.
II. Multiple clients could connect at the same time, but the server would be slow to respond since it could not utilize any multiple processor cores present on the server.
III. If a single client is connected to the server, no other clients will be able to receive service until the first client finishes.
IV. The server would quickly exhaust its available pool of socket port numbers.
V. Deadlock would occur due to concurrency issues handling the simultaneous client requests.

3. **Regular expressions** (14 points).

a) Hexadecimal is a way to represent a number in base 16. Hexadecimal numbers consist of the digits: `0123456789ABCDEF`. For each of the following write a regular expression for the following sets of hexadecimal strings. You may use any operations supported by Java regular expressions.

I. All hexadecimal numbers except for the empty string.

```
[0-9A-F]+
```

II. All 4-digit hexadecimal numbers that end in `00` or `FF`.

```
[0-9A-F]{2}(00|FF)
```

III. All hexadecimal numbers that include at least one letter digit.

```
[0-9A-F]*[A-F][0-9A-F]*
```

b) Proteins are described by a sequence of symbols. A C_2H_2 zinc finger consists of an amino acid sequence obeying the following ordering:

1. C
2. Between 2 and 4 amino acids
3. C
4. 3 more amino acids
5. One of the following amino acids: LIVMFYWCX
6. 8 more amino acids
7. H
8. Between 3 and 5 more amino acids
9. H

For example: CAASCGGPYACGGWAGYHAGWH
Write a regular expression that identifies strings that are C_2H_2 zinc fingers.

```
C.{2,4}C...[LIVMFYWC].{8}H.{3,5}H
```

4. **Socket programming** (16 points). A client and server play a number guessing game in which the server chooses a random number between 1-100 (inclusive). The client tries to guess the number efficiently via a binary search style algorithm. The socket communication protocol works as follows:

1) Client sends a line of text containing an integer guess (initial guess = 50).
2) Server responds with a line of text containing an integer: 0 if guess was correct, -1 if guess was too low, +1 if guess was too high.
3) If guess was correct, client terminates and server waits for next client. Otherwise goto 1.

Place letters in the boxes of the client and server programs to create a working implementation. ***Not all letters will be used and some letters may be used more than once.***

```
public class Client
{
   public static void main(String [] a) throws
                                        IOException
   {
      __C__

      __E__
      int msg;
      int low  = 1;
      int high = 100;
      int mid;
      while (true)
      {
         mid  = __P__

         __I__

         msg = __F__

         if      (msg == 0) __M__

         else if (msg < 0)  __N__

         else               __O__

      }
      System.out.println("Number was " + mid);
   }
}
```

```
public class Server
{
   public static void main(String[] a) throws
                                        IOException
   {
      __A__

      while (true)
      {
         __B__

         int num = __J__

         __E__
         while (true)
         {
            int msg = __F__

            if      (msg == num) __M__

            else if (msg < num)  __H__

            else                 __G__

         }
         writer.println(0);
      }
   }
}
```

```
A. ServerSocket serverSock = new ServerSocket(5000);
B. Socket sock = serverSock.accept();
C. Socket sock = new Socket("localhost", 5000);
D. ServerSocket serverSock = new ServerSocket("localhost", 5000);
E. BufferedReader reader = new BufferedReader(new InputStreamReader(sock.getInputStream()));
   PrintWriter    writer = new PrintWriter(sock.getOutputStream(), true);
F. Integer.parseInt(reader.readLine());
G. writer.println(1);
H. writer.println(-1);
I. writer.println(mid);
J. (int) (Math.random() * 100) + 1;
K. (int) Math.random() * 101 + 1;
L. (int) Math.random() * 101;
M. break;
N. low = mid + 1;
O. high = mid - 1;
P. (low + high) / 2;
Q. low + high / 2
R. (high - low) / 2
```

5. **Generics and linked structures** (14 points). The following class implements a stack abstract data type (ADT) using Java generics. Fill in the missing code in the underlined sections.

```
public class MyStack<E>
{
    private class Node
    {
        private E    item;
        private Node next;
    }
    private Node first = null;

    // Check if the stack is empty
    public boolean isEmpty()
    {
        return (first == null);
    }

    // Add a new item to the stack

    public void push(E s)
    {
        Node node   = new Node()
        node.item   = s;
        node.next   = first;

        first       = node;
    }

    // Remove the most recently added item
    public E pop()
    {
        if (isEmpty())
            throw new RuntimeException("Stack is empty!");

        E result = first.item;

        first    = first.next;
        return result;
    }

    // Find out how many items are currently in the stack
    public int size()
    {
        int result = 0;
        Node current = first;

        while (current != null)
        {
            result++;

            current = current.next;
        }
        return result;
    }
}
```