CSCI 136 Written Exam #1 **Name: _____**
Fundamentals of Computer Science II
Spring 2014

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	10	
2	18	
3	16	
4	12	
5	12	
Total	68	

1. **Methods, Strings** (10 points). Consider the following program:

```
import java.util.*;

public class Prob1
{
   public static void foo1(String [] args)
   {
      Arrays.sort(args);
      System.out.print(args[0]);
   }

   public static void foo2(String [] args)
   {
      for (String s : args)
         System.out.print(s.length() + " ");
   }

   public static void main(String [] args)
   {
      if (args.length % 2 == 0)
         foo1(args);
      else
         foo2(args);
   }
}
```

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 "error".

Command line	Output
% java Prob1	error
% java Prob1 bob	3
% java Prob1 bob carol	bob
% java Prob1 carol bob	bob
% java Prob1 bob carol alex	3 5 4

2. **Multiple choice** (18 points, 2 points each). For each question, circle the ***ONE*** best answer.

a) Consider the following two-dimensional array that holds temperature observations for the last week:

```
final int DAYS = 7;
final int HOURS = 24;
double [][] temps = new double[DAYS][HOURS];
```

All of the following lines of code cause either a compile error or a runtime exception ***EXCEPT***:

```
 I.  temps[0, 0] = 42;
II.  temps[0][HOURS - 1] = 42;
III. temps[DAYS][0] = 42;
IV.  temps[][] = 42;
 V.  temps[7][24] = 42.0;
```

b) You have a concrete class `Circle` that has lots of useful circle-related methods. You want to develop a special kind of circle that also has a text label. You would like your new class to inherit functionality from `Circle`. Which of the following is the ***best*** class declaration for your needs:

```
 I.  public class LabeledCircle extends Circle
II.  public class Circle extends LabelCircle
III. public class LabeledCircle implements Circle
IV.  public class Circle implements LabeledCircle
 V.  public class LabeledCircle<Circle>
```

c) Which of the following is the ***best*** reason for defining an abstract base class `Shape` from which many different concrete classes such as `Circle`, `Rectangle`, and `Polygon` can inherit from:

I. The memory required by objects of the concrete child types would be reduced.
II. Allows objects of any of the concrete child types to directly access private instance variables of any object that inherits from `Shape`.
III. Allows any object of the different concrete child types of `Shape` to be stored in the same array.
IV. Ensures that only one instance of any concrete child type is stored on the heap.
V. Ensures multi-threaded use of the concrete child types is thread-safe.

d) With regards to exception handling in Java, which of the following is ***TRUE***:

I. Only one `catch` block is allowed for a given `try` block.
II. Every line of code in a `try` block is eventually executed, regardless of whether an exception occurs somewhere inside the `try` block.
III. The `throws` keyword can be used as part of a method's declaration to avoid requiring code that may cause a checked exception from needing to be enclosed in a `try-catch` block.
IV. Code that has the potential to cause a runtime exception, such as `Integer.parseInt` must always be surrounded by a `try-catch` block.
V. Execution of the program always terminates after the final line of a `catch` block.

e) With regards to generics in Java, which of the following is ***FALSE***:

I. Wrapper classes such as `Double` and `Integer` must be used instead of primitive data type with Java generics.
II. The following is a valid class declaration of a generic data type: `public class Graph<E>`
III. The following is a valid class declaration of a generic data type: `public class Graph<Item>`
IV. Java's built-in collection classes such as `ArrayList` and `HashMap` use generics.
V. An object of a generic data type cannot be instantiated as its sole purpose is to serve as a template for polymorphism.

f) What is the primary advantage of using a data type that stores objects according to a hash function?

I. No matter how many items are stored, lookups only take constant time.
II. No matter how many items are stored, lookups only take time linear in the number of items.
III. No matter how many items are stored, lookups only take time linearithmic in the number of items.
IV. The items can easily be sorted.
V. The items can easily be moved between the stack and the heap.

g) In some languages, a class can inherit functionality from several parents. Java does not support such multiple inheritance but instead supports interfaces declared via the `implements` keyword. All the following are common uses of interfaces in Java ***EXCEPT***:

I. Defining a class that can be run as a separate thread.
II. Defining a class that can easily be sorted if put in a collection such as an `ArrayList`.
III. Defining a class that holds a collection of items and allowing the collection to be iterated over via an enhanced for-loop (also known as a for-each loop).
IV. Defining a class that holds a collection of items of any reference type.

h) What is the ***best*** reason for declaring an instance method using the `synchronized` keyword?

I. The method needs to make use of instance variables in its parent class.
II. The method modifies the state of an instance variable and may be called by multiple threads at the same time.
III. The method is time critical and requests the JVM optimize the method's byte code for performance.
IV. The method is recursive and thus needs additional stack space.
V. The method can be called without instantiating an object of its class data type.

i) You are developing a client-server program using Java sockets. When establishing the connection, the client provides both a domain name and a port number. What is the purpose of the ***port number***?

I. The client uses the port number to translate the domain name of the server to an IP address.
II. Allows the client's connection to work through all firewalls.
III. Allows the client to target a specific host machine somewhere on the network.
IV. Allows the client to target a specific process running on the target host machine.
V. Specifies the version of the IP protocol that the server should use for communication.

3. **Objects and references** (16 points). Consider the following program:

```
00  Duck d = new Duck("donald");
01  System.out.println("quack!");
02  Duck [] a = new Duck[3];
03  System.out.println("quack!");
04  for (int i = 0; i < a.length; i++)
05     a[i] = new Duck("daffy" + i);
06  System.out.println("quack!");
07  a[0] = null;
08  System.out.println("quack!");
09  a[1] = d;
10  System.out.println("quack!");
```

a) If the program is currently executing the given line, give the number of Duck objects currently on the heap and list the name(s) of each duck. The name of a Duck object is the string passed to its constructor.

Line	# of ducks on heap	Names of ducks on the heap
01	1	donald
03	1	donald
06	4	donald, daffy0, daffy1, daffy2
08	3	donald, daffy1, daffy2
10	2	donald, daffy2

b) Here is the current implementation of the Duck class:

```
public class Duck
{
   private String name;
   private static AudioFile sound = new AudioFile("quack.wav");

   public Duck(String name) { this.name = name; }
   public void quack()      { sound.play();     }
}
```

Your program has millions of Duck objects and requires lots of memory mostly due to each Duck having its own audio file. Modify the above program to fix this problem.

c) Write an equals instance method that returns true if and only if two ducks have exactly the same name.

```
public boolean equals(Duck other)
{
   return name.equals(other.name);
}
```

4. **Regular expressions** (12 points). Write a regular expression that matches the specified set of strings (and nothing else). You may use any regular expression operator support by Java. For the purposes of this problem, assume a ***word*** is defined as containing only the letters A through Z (uppercase) plus apostrophe.

a) Words that start with QU (including the word QU).

```
QU[A-Z']*
```

b) Words that are three to six characters long.

```
[A-Z']{3,6}
```

c) Words that have at least one vowel (vowels = AEIOU).

```
[A-Z']*[AEIOU][A-Z']*
```

d) Words that start with the prefix SUPER and end with either the suffix EST or the suffix ING.

```
SUPER[A-Z']*(EST|ING)
```

e) Words that contain no vowels (vowels = AEIOU).

```
[BCDFGHJKLMNPQRSTVWXYZ]+
```

f) Sentences consisting of one or more words. Words in the sentence are separated by a single space. Use a ␣ symbol for any spaces in your expression. Sentences cannot have leading or trailing whitespace.

```
[A-Z']+(␣[A-Z']+)*
```

5. **Socket programming** (12 points). You are writing the client portion of a program that transfers a text file. Your colleague is writing the server program. You have agreed on the following protocol:

1) The client initiates the conversation by sending a line of text with the number of lines in the file.
2) The client sends each line in the file in sequence.
3) The server responds with "OK" if all went well, the client can close the connection and terminate.
4) If the server responds with anything else, repeat the transmission (starting with the number of lines).

The client program takes three command line arguments: the filename, server name, and server port. Place letters in the boxes of the client program to create an implementation per this protocol.
Not all letters will be used and some letters may be used more than once.

```
public static void main(String [] args)
                        throws IOException
{
   String filename = args[0];
   String host = args[1];
   int port = Integer.parseInt(args[2]);

   _K_ lines = new _K_();

   Scanner scanner = new Scanner( _S_ );
   while (scanner.hasNextLine())
      lines.add( _Q_ );
   scanner.close();

   Socket sock = new Socket( _V_ )

   PrintWriter writer = new PrintWriter( _Y_ , true);

   InputStreamReader s = new InputStreamReader( _X_ );

   BufferedReader reader = new BufferReader(s);

   String result = "";

   _E_
   {
      writer.println( _J_ );
      for (String line : lines)
         writer._U_(line);

      result = reader._T_();
   }
   _A_

   reader.close();
   writer.close();
   sock.close();
}
```

A. `while (!result.equals("OK"));`
B. `while (result.equals("OK"));`
C. `while (result != "OK");`
D. `for (String result : results)`
E. `do`
F. `while`
G. `for`
H. `lines.get(i)`
I. `lines.get()`
J. `"" + lines.size()`
K. `ArrayList<String>`
L. `HashSet<String>`
M. `HashMap<String,Integer>`
N. `String[]`
O. `nextLine()`
P. `scanner.nextInt()`
Q. `scanner.nextLine()`
R. `filename`
S. `new File(filename)`
T. `readLine()`
U. `println()`
V. `host, port`
W. `new Connection(host, port)`
X. `sock.getInputStream()`
Y. `sock.getOutputStream()`
Z. `sock.toString()`