CSCI 136 Written Exam #2 **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	**12**	
3	**10**	
4	**10**	
5	**12**	
Total	**54**	

1. **Arrays, loops, strings** (10 points). Consider the following program:

```
public class Prob1
{
    public static void main(String [] args)
    {
        final int ROWS  = Integer.parseInt(args[0]);
        final int COLS  = Integer.parseInt(args[1]);
        final int SUM   = Integer.parseInt(args[2]);
        String [] lines = new String[ROWS];

        for (int row = 0; row < ROWS; row++)
        {
            lines[row] = "";
            for (int col = 0; col < COLS; col++)
            {
                if   ((row + col) == SUM)  lines[row] += "*";
                else                       lines[row] += ".";
            }
        }
        for (String s : lines)
            System.out.println(s);
    }
}
```

Below are five 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 2 2 1`	
`% java Prob1 2 3 1`	
`% java Prob1 2 3 -1`	
`% java Prob1 3 4`	
`% java Prob1 2 4 3 5.5`	

2. **File I/O, collections** (12 points). You are developing a program that computes the total sales for each product your company sells. The program reads data from the filename specified as the first command-line argument. The file has one sale logged on each line. Lines have three whitespace-separated columns: 1) product name (a string), 2) quantity (a positive integer), and 3) price per unit (a double). For example:

```
banana   12   0.23
apple     1   0.50
orange    5   0.30
apple     4   1.00
```

The program prints to standard output, in no particular order, the name of all unique products and the total dollar amount sold of each product (rounded to two decimal places). The output for the above file is:

```
orange 1.50
banana 2.76
apple 4.50
```

If the filename given as the first command-line argument is not found, it prints "Failed to open file!" and exits. If something else goes wrong parsing data in the file, it prints "Error parsing file!" and exits. Place letters in the underlined spaces to create a working implementation. ***Letters may be used 0 or more times.***

```
public static void main(String[] args)
{
   HashMap____ map = new HashMap____();
   try
   {
      Scanner scanner = new Scanner(____);
      while (scanner.hasNext())
      {
         String name  = scanner.next();
         int    qty   = scanner.nextInt();
         double price = scanner.nextDouble();

         if (map.containsKey(____))

            map.____(name, map.____(name) + qty * price);
         else

            map.____(name, qty * price);
      }
      scanner.close();
   }
   catch (____ e)
   {
      System.out.println("Failed to open file!");
      return;
   }
   catch (____ e)
   {
      System.out.println("Error parsing file!");
      return;
   }
   for (Map.Entry<String, Double> entry : map.entrySet())

      System.out.printf(____, ____, ____);
}
```

```
A. name
B. qty
C. price
D. args[0]
E. args.length
F. new File(args[0])
G. get
H. put
I. Double
J. "%1 %2\n"
K. "%s %.2f\n"
L. "%s %2d\n"
M. "%s %f"
N. InputMismatchException
O. FileNotFoundException
P. Error
Q. entry.getKey()
R. entry.getValue()
S. entry.get(name)
T. entry[name]
U. entry[price]
V. <char[], double>
W. <String, Double>
X. <String, double>
Y. <String, Int>
Z. <double, String>
```

3. **Multiple choice** (10 points, 2 points each). For each question, circle the ***best*** answer.

a) You are developing a program that represents the concentrations (in parts per million) of a particular contaminant at various locations in a cubic meter of water. You want to store 1000 equally spaced independent measurements within the volume of water. Which of the following is a variable declaration that is ***best*** suited for storing the data (best = compiles and only uses as much memory as necessary)?

```
  I. int          ppm  = new int[1000];
 II. int []       ppm  = new Integer[1000];
III. int [][][]   ppm  = new int[10][10][10];
 IV. int [2]      ppm  = new int[10][10];
  V. int [][][]   ppm  = new int[333][333][334];
```

b) You are benchmarking an algorithm that takes an input of size N. For an input of size N=5,000, the algorithm took 2.1 minutes. For an input of size N=10,000 it took 4.1 minutes. For an input of size N=20,000 it took 8.2 minutes. Which of the following best describes the order of growth of the algorithm:

I. $O(1)$
II. $O(N)$
III. $O(N^2)$
IV. $O(N^3)$
V. $O(2^N)$

c) You have an object named `obj` of the class `MyObject`. `MyObject` has an instance method `foo()` declared using the **`static`** keyword. Which of the following is ***TRUE***:

I. Only one thread can be executing the `foo()` method of `obj` at any one point in time.
II. Only one thread can be executing any **`static`** method of `obj` at any one point in time.
III. The `foo()` method can only use instance variables declared as **`private`**.
IV. The `foo()` method can only use instance variables declared as **`static`**.
V. The `foo()` method can only call instance methods declared as **`synchronized`**.

d) You have an object named `obj` of the class `MyObject`. `MyObject` has an instance method `foo2()` declared with the **`synchronized`** keyword. Which of the following is ***TRUE***:

I. Only one thread can be executing any method of `obj` at any one point in time.
II. Only one thread can be executing any **`synchronized`** method of `obj` at any one point in time.
III. The `foo2()` method can only use instance variables declared as **`private`**.
IV. The `foo2()` method can only use instance variables declared as **`static`**.
I. The `foo2()` method can only call instance methods declared as **`synchronized`**.

e) Why do computer scientists love using collection data types based on a *hash* function?

I. Because items can be retrieved from or put into the collection in constant time.
II. Because items can be retrieved from or put into the collection in amortized exponential time.
III. The hash function provide both type-safety and thread-safety.
IV. The hash function puts data into buckets that reduce the overall memory required by the collection.
V. Hash-based collections provide sorting in constant time.

4. **Matching** (10 points). Match each description of a computer science/Java programming concept with the ***best*** corresponding letter. Each letter will be used ***at most once***.

	Refers to the ability of a Java class to work with objects of any reference type.
	When a class defines multiple methods with the same name but different signatures.
	When different classes in a class hierarchy define methods with the same name and the same signature.
	Searches a sorted list in O(log N) time.
	Sorts a list of items in O(N log N) time.
	In Java Android programming, this typically corresponds to a screen in the user interface.
	A compact language for performing string matching and manipulation.
	Handles how different widgets are arranged in a GUI.
	Responds to things such as mouse or button clicks in a GUI.
	In Java socket programming, this side must specify the domain name/IP address and port number of its communication partner.
	In Java socket programming, this side calls the `accept()` method and typically spins up a thread to handle each connection.
	In Java desktop GUI programming, this class is typically instantiated or extended.
	In Java desktop GUI programming, this class can as a place to draw graphics or for as a container for grouping interface elements together.
	A Java construct that may include implemented methods but which itself cannot be instantiated.
	A Java construct that specifies method signatures but is forbidden from having any implemented methods.
	Implementing this Java interface allows a collection of objects of a particular type to be easily sorted.
	Implementing this Java interface allows a collection of objects of a particular type to be looped over via an enhanced for-loop (aka for-each loop).
	Implementing this Java interface allows code to be executed on a separate thread.
	Data types that must be instantiated with the new operator before use.
	Data types that do not need to be instantiated with the new operator before use.

A. Overloading
B. Generics
C. Layout manager
D. Hash search
E. Server
F. Overriding
G. Binary search
H. Widget
I. Merge sort
J. Regular expressions
K. Activity
L. Intent
M. Abstract class
N. Event handler
O. Threaded
P. Iterable
Q. Comparable
R. Interface
S. JFrame
T. Runnable
U. Reference
V. JPanel
W. Collection
X. Primitive
Y. Client
Z. JWindow

5. **Generics and linked structures** (12 points). The following class implements a first-in first-out (FIFO) queue abstract data type (ADT) using Java generics. The ADT uses a linked list data structure with instance variables that track: the beginning of the list, the end of the list, and the size of the list.

a) Fill in the missing code in the underlined sections:

```
public class MyQueue<E>
{
   private class Node
   {
      private E    item;
      private Node next;
   }

   private Node first = null;
   private Node last  = null;
   private int  size  = 0;

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

   // Return the current number of items in the queue
   public int size()
   {
      return size;
   }

   // Add a new item to the queue

   public void enqueue(__________________________________ s)
   {

      Node n = __________________________________;
      n.item = s;

      if (isEmpty())
      {
         first = n;
         last = n;
      }
      else
      {
         last.next = n;
         last = n;
      }
      size++;
   }
```

5. **Generics and linked structures (continued)**

```
   // Remove an item from the queue (the item that has been in the queue the longest)
   public E dequeue()
   {
      if (isEmpty()) throw new RuntimeException("Queue is empty!");

      E result = __________________________________;
      first = first.next;
      size--;

      if (isEmpty())
         last = null;

      return __________________________________;
   }

   // A string representing the queue, e.g. program in part b returns "the cat sat "
   public String toString()
   {
      String result = "";

      Node current = __________________________________;

      while (current != ______________________________)
      {
         result += current.item + " ";

         current = __________________________________;
      }

      return result;
   }
}
```

b) Draw a diagram showing the linked list structure resulting from the following program. Be sure to include in your diagram where the `first` and `last` instance variables point to:

```
MyQueue<String> q = new MyQueue<String>();
q.enqueue("the");
q.enqueue("cat");
q.enqueue("sat");
```