Practice Midterm Quiz

Answer all questions in the space provided. You need to work quickly and carefully. Do not "idiot-proof" your code unless explicitly instructed to do so. Don't panic if you have no time to look over your work at the end.

You may use any notes, books, or other references at your disposal. However, do not waste too much time leafing through your materials! Notice that the point value of each problem should give you a rough indication of how to budget your time ... you can figure on slightly more than around one minute per exam point (50 points of problems to complete in approximately 60 minutes). Don't leave the hardest questions for the last few minutes of this quiz!

Show all of your scrap work (in the space provided) for partial credit on various questions. If any problem appears ambiguous or seems to be worded unclearly, ask one of the CSCI E-10b teaching staff for help. It is silly to misinterpret a question you are capable of answering.

Write your name and the name of your teaching fellow (you do know them, don't you?) in the spaces below! **Best of luck!**

Your Name	
Your TF's Name	

[1]	[2]	[3]	[4]	TOTAL

[1] True/False (2 points each; 14 points total)

For each of the following, if the statement is false, explain why or fix it so it becomes true. If a statement is already true, just indicate so.

(a) The *superclass* constructor always gets executed before the *subclass* constructor.

(b) Suppose the **Foobar** class has been defined with a zero-argument constructor. The statement

```
System.out.println (new Foobar());
```

will cause an error at execution time unless a **toString()** method has been defined inside the **Foobar** class.

(c) You cannot have more than one **catch** clause per **try** statement.

(d) A constructor can use the keyword **super**, as if it were a method name, to invoke a different constructor in the same class.

(e) If two **`String`** objects are compared using the operator **==** a runtime error message will occur and the program will abort.

(f) In the base case, a *recursive* method calls itself with a smaller version of the original problem.

(g) If variable **a** is an array, variable **b** is an **ArrayList**, and variable **c** is a **String** object, then the statement

```
System.out.println ( a.length() + "  " +
                     b.size() + "  " + c.length );
```

should compile and execute without producing an error message.

[2] File Input Problem: 8 points total

The following program is supposed to report how many "blank lines" appear inside a file whose name is supplied on the command line. Four different Java expressions have been purposely omitted (each indicated by **???**); you will need to change each **???** directly on the code to make this program work as intended.

```
import java.io.*;
import java.util.*;

class Blank
{
  public static void main (String [] args) throws ???1
  {
     File f = new File ( ???2 );
     Scanner s = new Scanner (f);
     int count = 0;
     while (  ???3 )
     {
        String str = s.nextLine();
        if ( ???4 ) count++;
     }
    System.out.println("# of blank lines = " + count);
  }
}
```

[3] Inheritance Problem: 16 points total

Freight trains, like the one shown above, consist of 3 different types of railroad cars: rectangular *box cars* (which have a height, length and width), cylindrical *tank cars* (which have a diameter and a length), and *engines*. All railroad cars, regardless of their type, have two properties: a **serial number** and a **year** in which it was **built**.

Shown below is a Java program that constructs a freight train by constructing individual **BoxCar**s, **TankCar**s and **Engine**s and storing them in an *array*. The program then prints some basic information about these railroad cars.

```
class TrainDemo
{
   public static void main (String [] args)
   {
      RailroadCar [] myFreightTrain =
        {
           new Engine ("OhBama", 1961),
           new BoxCar ("bar34", 1953, 10, 10.0, 8.9),
           new TankCar("TBrady", 1977, 10, 5.0),
           new BoxCar ("blah17", 1966, 10, 11, 5)
        };

      for (RailroadCar r : myFreightTrain)
      {
         System.out.println( r + " and has a volume of " +
                             r.volume() );
      }
    }
}
```

The output from this program appears below:

```
Engine #OhBama was built in 1961 and has a volume of 0.0
Box Car #bar34 was built in 1953 and has a volume of 890.0
Tank Car #TBrady was built in 1977 and has a volume of 785.3981
Box Car #blah17 was built in 1966 and has a volume of 550.0
```

The **Engine** class is pretty trivial to define:

```
class Engine extends RailroadCar
{
   public Engine (String serialNumber, int yearBuilt)
   {
      super( serialNumber, yearBuilt );
   }

   public double volume ()  {return 0.0; }

   public String toString()
   {
      return "Engine " + super.toString();
   }
}
```

Part (a) **6 points**

Fill in the two missing code fragments (one in the **RailroadCar** constructor, the other in the **toString** method) to complete the definition of the following **RailroadCar** class.

```
public abstract class RailroadCar
{
   protected String serialNumber;
   protected int yearBuilt;

   public RailroadCar (String s, int y)
   {     // complete this




   }
```

```
    public abstract double volume();
    public String toString ()
    {       // complete this









    }
}
```

Part (b) **2 points**

What sort of error message would the **`javac`** compiler produce when trying to compile the main program if the statement

```
public abstract double volume();
```

were removed from the definition of **`RailroadCar`**?

Part (c) **6 points**

Now define the complete **`BoxCar`** class. Note that the volume of a **`BoxCar`** is easily computed as the product of its length, width and height.

Part (d) **2 points**

Suppose we want an easy way to keep track of the total number of **RailroadCar**s that get constructed (whether they get stored in arrays or elsewhere). In one or two simple, unambiguous English sentences, how can this be accomplished by making changes to the **`RailroadCar`** class only?

[4] Recursive Programming Problem: 12 points total

(3 points for each part)

Recall the *binary search algorithm,* which searches through a *sorted* array for a particular value, and returns the index of where the value was found in the array; it returns with **-1** if the value cannot be located.

An incomplete *recursive* solution for this algorithm appears below:

```
static int binary (int [] a, int fromIndex,
                   int toIndex, int key)
{
    if (fromIndex > toIndex) return ???1     ;
    else
    {
       int middle = (fromIndex + toIndex)/2;
       if (key == a[middle]) return ???2     ;
       else if (key > a[middle]) return binary( ???3 ) ;
       else return binary (   ???4   )   ;
    }
}
```

Figure out precisely what code is missing from each of the four **???**'s and fill it in above.

The first parameter **a**, is an array of sorted integers that gets searched. The fourth parameter (*key*) is the integer value that is being search for inside of **a**. The second and third parameters (*fromIndex* and *toIndex*) stipulate the array indices of where to begin and end the search. If we wanted to search through an entire sorted integer array named **foobar** for the value -17, we might call on the above method as follows:

```
binary (foobar, 0, foobar.length-1, -17)
```