CMSC131
Summer 2004
Midterm #2

Name: _____

Student ID: _____

Section: _____

Grader Use Only:

#1		(15)
#2		(26)
#3		(35)
#4		(24)
Total		(100)

Problem 1 (15 pts)

For the following program, draw a memory diagram that represents the state of the variables max, winter, fall, good, bad, k, and temp at the point where execution reaches the point marked as "END POINT". Notice that this point is right before the method returns. Draw all the objects that have been created.

```
public class Message {
      private String text;
      private int priority;

      public Message(String theText, int thePriority) {
            text = theText;
            priority = thePriority;
      }

      public void setMessage(String newText, int newPriority) {
            text = newText;
            priority = newPriority;
      }

      public static void passing(Message good, Message bad, double k) {
            Message temp = good;
            temp.setMessage("SUMMER", 100);
            bad = null;
            good.setMessage("LAST", 200);
            k = 200;
            // END POINT (Before returning from method)
      }

      public static void main(String[] args) {
            double max = 79.0;
            Message winter = new Message("WINTER", 30);
            Message fall = new Message("FALL", 40);
            passing(winter, fall, max);
      }
}
```

SPACE FOR MEMORY DIAGRAM

Problem 2 (26 points)

The City class is defined as follows:

```
public class City {
      public String name;
      private int population;
      public static int totalCities = 100;
      public static final int maxBudget = 20;

      public City(String theName, int thePopulation) {
            name = theName;
            population = thePopulation;
      }

      public void display() {
            System.out.println("Name: " + name);
            System.out.println("Population: " + population);
      }
}
```

Which of the following are considered valid statements that can appear in the **main** method for the class? Write **Valid** or **Invalid** next to the entry.

```
a. City.name = null;

b. City.population = 10;

c. City.totalCities = 100;

d. City c1 = City();

e. City c2 = new City();

f. City c3 = new City("Washington", 3);

g. City.display();

h. City c4 = new City(new City());

i. name = "Washington";

j. population = 10;

k. City.maxBudget = 30;

l. totalCities = 20;

m. new City().display();
```

Problem 3 (35 points)

For this problem you will implement a complete class called Rectangle which represents a rectangle. At the end, we have provided a driver (and associated output) to help you understand what is expected from each method you need to implement.

Fields (instance variables)

The class has two fields:

- **len**- a double representing the length of the rectangle.
- **wid** – a double representing the width of the rectangle.

Methods you must implement

- **Constructor** – Define a constructor that takes two parameters (the length and width of a rectangle) and initializes the fields accordingly.
- **area** - Define a method called area that returns the area of the rectangle represented by the current object.
- **merge** - This method takes a Rectangle reference. The method will create a new Rectangle object whose length is the sum of the lengths of the current object and the length of the parameter. Similarly, the width of the new Rectangle object is the sum of the width of the current object and the width of the parameter. A reference to the Rectangle object created must be returned. The current object and the parameter object may not be modified.
- **toString** - Based on the provided output write the toString method associated with this class.
- **totalArea –** This is an **static** method that takes an array of Rectangles as a parameter. The method returns the sum of the areas of the rectangles in the array.

Driver

```
public static void main(String[] args) {
      Rectangle r1 = new Rectangle(2,3);
      Rectangle r2 = new Rectangle(7,5);
      System.out.println(r1);
      System.out.println("Area: " + r1.area());
      System.out.println(r1.merge(r2));

      Rectangle[] allRects = new Rectangle[2];
      allRects[0] = r1;
      allRects[1] = r2;
      System.out.println("Total Area: " + Rectangle.totalArea(allRects));
}
```

Driver Output

```
Length: 2.0 Width: 3.0
Area: 6.0
Length: 9.0 Width: 8.0
Total Area: 41.0
```

Problem 4 (24 points)

Define a class called **Utilities** which has the following public methods:

a. Define a method called **count** that takes two parameters named **data** and **cutoff**. The **data** parameter represents an array of integers. The **cutoff** parameter is an integer value. The method will return the number of entries in the array with a value less or equal to the **cutoff** parameter.

b. Define a method called **filter** that takes two parameters named **data** and **cutoff**. The data parameter represents an array of integers. The **cutoff** parameter is an integer value. The method will return a new array of integers where elements of the **data** array with a value greater than the **cutoff** value are not present. Feel free to use the **count** method we described earlier, even if you have not implemented the method.

The following driver provides an example dealing with the methods you are expected to define.

Driver

```
public static void main(String[] args) {
      Utilities util = new Utilities();
      int[] values = {10, 1, 8, 13, 2, 15, 18};

      System.out.println("Count: " + util.count(values, 8));
      int[] result = util.filter(values, 8);
      for (int i=0; i<result.length; i++) {
            System.out.print(result[i] + " ");
      }
}
```

Output

```
Count: 3
1 8 2
```