CSE 142 Sample Final Exam #2

(based on Winter 2005's final; thanks to Stuart Reges)

1. Array Mystery

Consider the following method:

```
public static int arrayMystery(int[] array) {
    int x = 0;
    for (int i = 0; i < array.length - 1; i++) {
        if (array[i] > array[i + 1]) {
            x++;
        }
    }
    return x;
}
```

In the left-hand column below are specific arrays of integers. Indicate in the right-hand column what value would be returned by method `arrayMystery` if the integer array in the left-hand column is passed as its parameter.

Original Contents of Array	Value Returned
`int[] a1 = {8};` `int result1 = arrayMystery(a1);`	_____
`int[] a2 = {14, 7};` `int result2 = arrayMystery(a2);`	_____
`int[] a3 = {7, 1, 3, 2, 0, 4};` `int result3 = arrayMystery(a3);`	_____
`int[] a4 = {10, 8, 9, 5, 6};` `int result4 = arrayMystery(a4);`	_____
`int[] a5 = {8, 10, 8, 6, 4, 2};` `int result5 = arrayMystery(a5);`	_____

2. Reference Semantics Mystery

The following program produces 4 lines of output. Write the output below, as it would appear on the console.

```
import java.util.*;   // for Arrays class

public class ReferenceMystery {
    public static void main(String[] args) {
        int y = 1;
        int x = 3;
        int[] a = new int[4];

        mystery(a, y, x);
        System.out.println(x + " " + y + " " + Arrays.toString(a));

        x = y - 1;
        mystery(a, y, x);
        System.out.println(x + " " + y + " " + Arrays.toString(a));
    }

    public static void mystery(int[] a, int x, int y) {
        if (x < y) {
            x++;
            a[x] = 17;
        } else {
            a[y] = 17;
        }
        System.out.println(x + " " + y + " " + Arrays.toString(a));
    }
}
```

3. Inheritance Mystery

Assume that the following classes have been defined:

```
public class Pen extends Sock {
    public void method1() {
        System.out.print("pen 1  ");
    }
}

public class Lamp {
    public void method1() {
        System.out.print("lamp 1  ");
    }

    public void method2() {
        System.out.print("lamp 2  ");
    }

    public String toString() {
        return "lamp";
    }
}
```

```
public class Book extends Sock {
    public void method2() {
        System.out.print("book 2  ");
        super.method2();
    }
}

public class Sock extends Lamp {
    public void method1() {
        System.out.print("sock 1  ");
        method2();
    }

    public String toString() {
        return "sock";
    }
}
```

Given the classes above, what output is produced by the following code?

```
Lamp[] elements = {new Book(), new Pen(), new Lamp(), new Sock()};
for (int i = 0; i < elements.length; i++) {
    System.out.println(elements[i]);
    elements[i].method1();
    System.out.println();
    elements[i].method2();
    System.out.println();
    System.out.println();
}
```

4. File Processing

Write a static method named `wordStats` that accepts as its parameter a `Scanner` holding a sequence of words and that reports the total number of words (as an integer) and the average word length (as an un-rounded real number). For example, suppose the `Scanner` is scanning an input source that contains the following words:

```
To be or not to be, that is the question.
```

For the purposes of this problem, we will use whitespace to separate words. That means that some words include punctuation, as in `"be,"`. (This is the same definition that the `Scanner` uses for tokens.) For the input above, your method should produce exactly the following output:

```
Total words    = 10
Average length = 3.2
```

5. File Processing

Write a static method named `flipLines` that accepts as its parameter a `Scanner` for an input file and that writes to the console the same file's contents with successive pairs of lines reversed in order. For example, if the input file contains the following text:

```
Twas brillig and the slithy toves
did gyre and gimble in the wabe.
All mimsey were the borogroves,
and the mome raths outgrabe.

"Beware the Jabberwock, my son,
the jaws that bite, the claws that catch,
Beware the JubJub bird and shun
the frumious bandersnatch."
```

The program should print the first pair of lines in reverse order, then the second pair in reverse order, then the third pair in reverse order, and so on. Therefore your method should produce the following output to the console:

```
did gyre and gimble in the wabe.
Twas brillig and the slithy toves
and the mome raths outgrabe.
All mimsey were the borogroves,
"Beware the Jabberwock, my son,

Beware the JubJub bird and shun
the jaws that bite, the claws that catch,
the frumious bandersnatch."
```

Notice that a line can be blank, as in the third pair. Also notice that an input file can have an odd number of lines, as in the one above, in which case the last line is printed in its original position. You may not make any assumptions about how many lines are in the Scanner.

6. Array Programming

Write a static method named `minGap` that accepts an integer array as a parameter and returns the minimum 'gap' between adjacent values in the array. The gap between two adjacent values in a array is defined as the second value minus the first value. For example, suppose a variable called `array` is an array of integers that stores the following sequence of values.

```
int[] array = {1, 3, 6, 7, 12};
```

The first gap is 2 (3 - 1), the second gap is 3 (6 - 3), the third gap is 1 (7 - 6) and the fourth gap is 5 (12 - 7). Thus, the call of `minGap(array)` should return `1` because that is the smallest gap in the array. Notice that the minimum gap could be a negative number. For example, if array stores the following sequence of values:

```
(3, 5, 11, 4, 8)
```

The gaps would be computed as 2 (5 - 3), 6 (11 - 5), -7 (4 - 11), and 4 (8 - 4). Of these values, -7 is the smallest, so it would be returned.

This gap information can be helpful for determining other properties of the array. For example, if the minimum gap is greater than or equal to 0, then you know the array is in sorted (nondecreasing) order. If the gap is greater than 0, then you know the array is both sorted and unique (strictly increasing).

If you are passed an array with fewer than 2 elements, you should return 0.

7. Array Programming

Write a static method named `longestSortedSequence` that accepts an array of integers as a parameter and that returns the length of the longest sorted (nondecreasing) sequence of integers in the array. For example, if a variable named `array` stores the following values:

```
int[] array = {3, 8, 10, 1, 9, 14, -3, 0, 14, 207, 56, 98, 12};
```

then the call of `longestSortedSequence(array)` should return `4` because the longest sorted sequence in the array has four values in it (the sequence -3, 0, 14, 207). Notice that sorted means nondecreasing, which means that the sequence could contain duplicates. For example, if the array stores the following values:

```
int[] array2 = {17, 42, 3, 5, 5, 5, 8, 2, 4, 6, 1, 19}
```

Then the method would return `5` for the length of the longest sequence (the sequence 3, 5, 5, 5, 8). Your method should return 0 if passed an empty array. Your method should return 1 if passed an array that is entirely in decreasing order or contains only one element.

8. Critters

Write a class `Bumblebee` that extends the `Critter` class from the Critters assignment.
A `Bumblebee` object should move in a "spiral" pattern from W to S to E to N, lengthening each time:

- one step west
- two steps south
- three steps east
- four steps north
- five steps west
- six steps south
- seven steps east
- eight steps north
- nine steps west

  ...

All other `Bumblebee` behavior uses the `Critter` defaults. You may add anything needed (fields, other methods) to implement this behavior appropriately.

9. Classes and Objects

Suppose that you are provided with a pre-written class `Rectangle` as described at right. (The headings are shown, but not the method bodies, to save space.) Assume that the fields, constructor, and methods shown are already implemented. You may refer to them or use them in solving this problem if necessary.

Write an instance method named **`union`** that will be placed inside the `Rectangle` class to become a part of each `Rectangle` object's behavior. The `union` method accepts another rectangle *r* as a parameter and turns this rectangle into the union of itself and *r* ; that is, modifies the fields so that they represent the smallest rectangular region that completely contains both this rectangle and *r* .

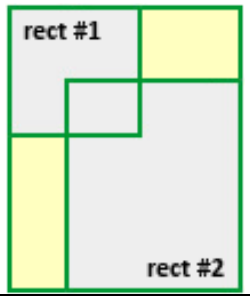
For example, if the following `Rectangle`s are declared in client code:

```
Rectangle rect1 = new Rectangle(5, 12, 4, 6);
Rectangle rect2 = new Rectangle(6,  8, 5, 7);

Rectangle rect3 = new Rectangle(14, 9, 3, 3);
Rectangle rect4 = new Rectangle(10, 3, 5, 8);
```

The following calls to the `union` method would modify the objects' state as indicated in the comments.

```
rect1.union(rect2);   // {(5, 8), 6x10}
rect4.union(rect3);   // {(10, 3), 7x9}
```

```
// A Rectangle stores an (x, y)
// coordinate of its top/left
// corner, a width and height.

public class Rectangle {
    private int x;
    private int y;
    private int width;
    private int height;

    // Constructs a new Rectangle
    // with the given x,y,w,h.
    public Rectangle(int x, int y,
                     int w, int h)

    // returns the field values
    public int getX()
    public int getY()
    public int getWidth()
    public int getHeight()

    // example: {(5, 12), 4x8}
    public String toString()


    // your method would go here

}
```



Solutions

1.

Call	Value Returned
`int[] a1 = {8};` `int result1 = arrayMystery(a1);`	0
`int[] a2 = {14, 7};` `int result2 = arrayMystery(a2);`	1
`int[] a3 = {7, 1, 3, 2, 0, 4};` `int result3 = arrayMystery(a3);`	3
`int[] a4 = {10, 8, 9, 5, 6};` `int result4 = arrayMystery(a4);`	2
`int[] a5 = {8, 10, 8, 6, 4, 2};` `int result5 = arrayMystery(a5);`	4

2.

```
2 3 [0, 0, 17, 0]
3 1 [0, 0, 17, 0]
1 0 [17, 0, 17, 0]
0 1 [17, 0, 17, 0]
```

3.

```
sock
sock 1  book 2  lamp 2
book 2  lamp 2

sock
pen 1
lamp 2

lamp
lamp 1
lamp 2

sock
sock 1  lamp 2
lamp 2
```

4.

```
public static void wordStats(Scanner input) {
    int count = 0;
    int sumLength = 0;

    while (input.hasNext()) {
        String next = input.next();
        count++;
        sumLength += next.length();
    }

    double average = (double) sumLength / count;
    System.out.println("Total words    = " + count);
    System.out.println("Average length = " + average);
}
```

5.

```
public static void flipLines(Scanner input) {
    while (input.hasNextLine()) {
        String first = input.nextLine();
        if (input.hasNextLine()) {
            String second = input.nextLine();
            System.out.println(second);
        }
        System.out.println(first);
    }
}
```

6.

```
public static int minGap(int[] list) {
    if (list.length < 2) {
        return 0;
    } else {
        int min = list[1] - list[0];
        for (int i = 2; i < list.length; i++) {
            int gap = list[i] - list[i - 1];
            if (gap < min) {
                min = gap;
            }
        }
        return min;
    }
}
```

7.

```
public static int longestSortedSequence(int[] list) {
    if (list.length == 0) {
        return 0;
    }

    int max = 1;
    int count = 1;
    for (int i = 1; i < list.length; i++) {
        if (list[i] >= list[i - 1]) {
            count++;
        } else {
            count = 1;
        }

        if (count > max) {
            max = count;
        }
    }
    return max;
}
```

8.

```
public class Bumblebee extends Critter {
    private int steps;
    private int max;
    private int direction;    // 0=west, 1=south, 2=east, 3=north

    public Bumblebee() {
        steps = 0;
        max = 1;
        direction = 0;
    }

    public Direction getMove() {
        steps++;
        if (steps > max) {
            // Pick a new direction and re-set the steps counter
            steps = 1;
            max++;
            direction = (direction + 1) % 4;
        }

        if (direction == 0) {
            return Direction.WEST;
        } else if (direction == 1) {
            return Direction.SOUTH;
        } else if (direction == 2) {
            return Direction.EAST;
        } else {  // direction == 3
            return Direction.NORTH;
        }
    }
}
```

9.

```
public void union(Rectangle r) {
    // find the union's bounds
    int left = Math.min(x, r.getX());
    int top = Math.min(y, r.getY());
    int right = Math.max(x + width, r.getX() + r.getWidth());
    int bottom = Math.max(y + height, r.getY() + r.getHeight());

    x = left;
    y = top;
    width = right - left;
    height = bottom - top;
}
```