CSE 142 Sample Final Exam #4

(based on Autumn 2008's final)

1. Array Mystery

Consider the following method:

```
public static void arrayMystery(int[] a) {
    for (int i = a.length - 2; i > 0; i--) {
        if (a[i - 1] < a[i + 1]) {
            a[i] += a[i - 1];
        } else {
            a[i] += a[i + 1];
        }
    }
}
```

Indicate in the right-hand column what values would be stored in the array after the method `arrayMystery` executes if the integer array in the left-hand column is passed as a parameter to it.

Original Contents of Array	Final Contents of Array
`int[] a1 = {1, 2, 3};` `arrayMystery(a1);`	_____
`int[] a2 = {8, 2, 3, 1, 6};` `arrayMystery(a2);`	_____
`int[] a3 = {1, 1, 1, 1, 1, 1};` `arrayMystery(a3);`	_____
`int[] a4 = {40, 10, 25, 5, 10, 30};` `arrayMystery(a4);`	_____
`int[] a5 = {15, 6, -1, 4, 8, -2, 7, 4};` `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.

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

        x++;
        a[x - 1] = 3;
        mystery(x, a);
        System.out.println(x + " " + Arrays.toString(a));

        x++;
        a[x - 1] = 2;
        mystery(x, a);
        System.out.println(x + " " + Arrays.toString(a));
    }

    public static void mystery(int x, int[] a) {
        a[x]++;
        x--;
        a[x - 1] = a[x + 1];
        System.out.println(x + " " + Arrays.toString(a));
    }
}
```

3. Inheritance Mystery

Assume that the following classes have been defined:

```
public class McCain extends Biden {
    public void republican() {
        System.out.print("mccain-R   ");
    }
}

public class Palin {
    public void republican() {
        System.out.print("palin-R   ");
    }

    public void democrat() {
        republican();
        System.out.print("palin-D   ");
    }

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

```
public class Obama extends Palin {
    public void republican() {
        super.republican();
        System.out.print("obama-R   ");
    }
}

public class Biden extends Palin {
    public String toString() {
        return "biden";
    }

    public void democrat() {
        System.out.print("biden-D   ");
        super.democrat();
    }
}
```

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

```
Palin[] politicians = {new Biden(), new Palin(), new McCain(), new Obama()};
for (int i = 0; i < politicians.length; i++) {
    System.out.println(politicians[i]);
    politicians[i].republican();
    System.out.println();
    politicians[i].democrat();
    System.out.println();
    System.out.println();
}
```

4. File Processing

Write a static method named `reportBlankLines` that accepts a `Scanner` containing an input file as a parameter and that outputs the line numbers of any blank lines and that reports the total number of blank lines in the file. For example, given the following input file:

```
Remember that a file
can have blank lines
like the one below:

A blank line:

is read as a String
of length 0

by Scanner
```

Your method should print the following output:

```
line 4 is blank
line 6 is blank
line 9 is blank
total blank lines = 3
```

Notice that each blank line produces a line of output and that there is a final line of output reporting the total number of blank lines. Also notice that lines are numbered starting with 1 (first line is line 1, second line is line 2, and so on). If the input has no lines or no blank lines, output the following:

```
total blank lines = 0
```

5. File Processing

Write a static method named `printDuplicates` that accepts as its parameter a `Scanner` for an input file containing a series of lines. Your method should examine each line looking for consecutive occurrences of the same token on the same line, and print each duplicated token along how many times it appears consecutively. Non-repeated tokens are not printed. Repetition across multiple lines (such as if a line ends with a given token and the next line starts with the same token) is not considered in this problem.

For example, if the input file contains the following text (sequences of duplicated tokens are underlined for emphasis):

```
hello how how are you you you you
I I I am Jack's Jack's smirking smirking smirking smirking smirking revenge
   bow  wow wow yippee yippee   yo yippee   yippee yay  yay yay
one fish two fish red fish blue fish
It's the Muppet Show, wakka wakka wakka
```

Your method would produce the following output for the preceding input file:

```
how*2 you*4
I*3 Jack's*2 smirking*5
wow*2 yippee*2 yippee*2 yay*3

wakka*3
```

Your code prints only the repeated tokens; the ones that appear only once in a row are not shown. Your code should place a single space between each reported duplicate token and should respect the line breaks in the original file. This is why a blank line appears in the expected output, corresponding to the fourth line of the file that did not contain any consecutively duplicated tokens. You may assume that each line of the file contains at least 1 token of input.

6. Array Programming

Write a static method named `arraySum` that accepts two arrays of real numbers *a1* and *a2* as parameters and returns a new array *a3* such that each element of *a3* at each index *i* is the sum of the elements at that same index *i* in *a1* and *a2*. For example, if *a1* stores `{4.5, 5.0, 6.6}` and *a2* stores `{1.1, 3.4, 0.5}`, your method should return `{5.6, 8.4, 7.1}`, which is obtained by adding 4.5 + 1.1, 5.0 + 3.4, and 6.6 + 0.5.

If the arrays *a1* and *a2* are not the same length, the result returned by your method should have as many elements as the larger of the two arrays. If a given index *i* is in bounds of *a1* but not *a2* (or vice versa), your result array's element at index *i* should be equal to the value of the element at index *i* in the longer of *a1* or *a2*. For example, if *a1* stores `{1.8, 2.9, 9.4, 5.5}` and *a2* stores `{2.4, 5.0}`, your method should return `{4.2, 7.9, 9.4, 5.5}`.

The table below shows some additional calls to your method and the expected values returned:

Arrays	Call and Value Returned
`double[] a1 = {4.5, 2.8, 3.4, 0.8};` `double[] a2 = {1.4, 8.9, -1.0, 2.3};`	`arraySum(a1, a2)` returns `{5.9, 11.7, 2.4, 3.1}`
`double[] ax = {2.4, 3.8};` `double[] ay = {0.2, 9.2, 4.3, 2.8, 1.4};`	`arraySum(ax, ay)` returns `{2.6, 13.0, 4.3, 2.8, 1.4}`
`double[] aa = {1.0, 2.0, 3.0};` `double[] ab = {4.0, 5.0};`	`arraySum(aa, ab)` returns `{5.0, 7.0, 3.0}`
`double[] ai = {};` `double[] aj = {42.0};`	`arraySum(ai, aj)` returns `{42.0}`

For full credit, you should not modify the elements of *a1* or *a2*. You may not use a `String` to solve this problem.

7. Array Programming

Write a static method named `partition` that accepts an array of integers *a* and an integer element value *v* as its parameters, and rearranges ("partitions") the array's elements so that all its elements of *a* that are less than *v* occur before all elements that are greater than *v*. The exact order of the elements is unimportant so long as all elements less than v appear before all elements greater than *v*. For example, if your method were passed the following array:

```
int[] a = {15, 1, 6, 12, -3, 4, 8, -7, 21, 2, 30, -1, 9};
partition(a, 5);
```

One acceptable ordering of the elements after the call would be: (elements < 5 and > 5 are underlined for emphasis)

{-1, 1, 2, -7, -3, 4, 8, 12, 21, 6, 30, 15, 9}

Hint: Your method will need to rearrange the elements of the array, which will involve swapping various elements from less desirable indexes to more desirable ones.

You may assume that the array contains no duplicates and does not contain the element value *v* itself. You may not use `Arrays.sort`, `Collections.sort`, or any other pre-defined sorting algorithm from the Java Class Libraries or textbook to solve this problem. You also may not use a `String` to solve this problem.

8. Classes and Objects

Suppose that you are provided with a pre-written class `Date` 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 **`absoluteDay`** that will be placed inside the `Date` class to become a part of each `Date` object's behavior. The `absoluteDay` method should return the "absolute day of the year" between 1 and 365 represented by the `Date` object. January 1 is absolute day #1, January 2 is day #2, ..., and December 31st is absolute day #365. For example, calling this method on a date representing February 13th should return 44, and calling it on a `Date` representing September 19th should return 262.

Suppose the following dates have been declared:

```
Date jan1 = new Date(1, 1);
Date jan4 = new Date(1, 4);
Date feb1 = new Date(2, 1);
Date mar10 = new Date(3, 10);
Date sep19 = new Date(9, 19);
Date dec31 = new Date(12, 31);
```

The results of calling your method on the above objects are:

Call	Returns
jan1.absoluteDay()	1
jan4.absoluteDay()	4
feb1.absoluteDay()	32
mar10.absoluteDay()	69
sep19.absoluteDay()	262
dec31.absoluteDay()	365

You should not solve this problem by writing 12 `if` statements, one for each month; this is redundant and poor style. If you solve the problem that way, you will receive at most half credit. Also, your method should not have the side effect of modifying the `Date` object on which it was called. In other words, when your method is done executing, the values of the fields of the `Date` object on which it was called should be unchanged. A solution that does so will receive at most half credit. (It's okay for you to modify the object's state temporarily if it helps you solve this problem, but when your method is done executing, the state should be the same as when you started.)

Recall that your method is allowed to call other methods on `Date` objects or construct other objects if you like.

```
// Each Date object stores a single
// month/day such as September 19.
// This class ignores leap years.

public class Date {
    private int month;
    private int day;

    // Constructs a date with
    // the given month and day.
    public Date(int m, int d)

    // Returns the date's day.
    public int getDay()

    // Returns the date's month.
    public int getMonth()

    // Returns the number of days
    // in this date's month.
    public int daysInMonth()

    // Modifies this date's state
    // so that it has moved forward
    // in time by 1 day, wrapping
    // around into the next month
    // or year if necessary.
    // example:  9/19 -> 9/20
    // example:  9/30 -> 10/1
    // example: 12/31 -> 1/1
    public void nextDay()


    // your method would go here

}
```

Solutions

1. Array Mystery

Expression	Final Contents of Array
`int[] a1 = {1, 2, 3};` `arrayMystery(a1);`	`{1, 3, 3}`
`int[] a2 = {8, 2, 3, 1, 6};` `arrayMystery(a2);`	`{8, 7, 5, 4, 6}`
`int[] a3 = {1, 1, 1, 1, 1, 1};` `arrayMystery(a3);`	`{1, 2, 2, 2, 2, 1}`
`int[] a4 = {40, 10, 25, 5, 10, 30};` `arrayMystery(a4);`	`{40, 45, 35, 20, 15, 30}`
`int[] a5 = {15, 6, -1, 4, 8, -2, 7, 4};` `arrayMystery(a5);`	`{15, 8, 2, 3, 11, 3, 5, 4}`

2. Reference Semantics Mystery

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

3. Inheritance Mystery

```
biden
palin-R
biden-D   palin-R   palin-D

palin
palin-R
palin-R   palin-D

biden
mccain-R
biden-D   mccain-R   palin-D

palin
palin-R   obama-R
palin-R   obama-R   palin-D
```

4. File Processing

```
public static void reportBlankLines(Scanner input) {
    int line = 0;
    int count = 0;
    while (input.hasNextLine()) {
        String text = input.nextLine();
        line++;
        if (text.length() == 0) {
            System.out.println("line " + line + " is blank");
            count++;
        }
    }
    System.out.println("total blank lines = " + count);
}
```

5. File Processing (two solutions shown)

```
public static void printDuplicates(Scanner input) {
    while (input.hasNextLine()) {
        String line = input.nextLine();
        Scanner lineScan = new Scanner(line);
        String token = lineScan.next();
        int count = 1;
        while (lineScan.hasNext()) {
            String token2 = lineScan.next();
            if (token2.equals(token)) {
                count++;
            } else {
                if (count > 1) {
                    System.out.print(token + "*" + count + " ");
                }
                token = token2;
                count = 1;
            }
        }

        if (count > 1) {
            System.out.print(token + "*" + count);
        }
        System.out.println();
    }
}
```

```
public static void printDuplicates(Scanner input) {
    while (input.hasNextLine()) {
        String line = input.nextLine();
        Scanner lineScan = new Scanner(line);
        String token = lineScan.next();
        int count = 1;
        while (lineScan.hasNext()) {
            String token2 = lineScan.next();
            if (token2.equals(token)) {
                count++;
            }

            if (count > 1 && (!lineScan.hasNext() || !token2.equals(token))) {
                System.out.print(token + "*" + count + " ");
                count = 1;
            }
            token = token2;
        }

        System.out.println();
    }
}
```

6. Array Programming (five solutions shown)

```
public static double[] arraySum(double[] a1, double[] a2) {
    double[] a3 = new double[Math.max(a1.length, a2.length)];
    for (int i = 0; i < a3.length; i++) {
        if (i >= a1.length) {           // done with a1; take from a2
            a3[i] = a2[i];
        } else if (i >= a2.length) {    // done with a2; take from a1
            a3[i] = a1[i];
        } else {
            a3[i] = a1[i] + a2[i];      // take sum of a1 and a2
        }
    }
    return a3;
}

public static double[] arraySum(double[] a1, double[] a2) {
    double[] a3 = new double[Math.max(a1.length, a2.length)];
    for (int i = 0; i < a1.length; i++) {    // add a1 into result
        a3[i] += a1[i];
    }
    for (int i = 0; i < a2.length; i++) {    // add a2 into result
        a3[i] += a2[i];
    }
    return a3;
}

public static double[] arraySum(double[] a1, double[] a2) {
    double[] a3;                             // create result array
    if (a1.length > a2.length) {
        a3 = new double[a1.length];
    } else {
        a3 = new double[a2.length];
    }
    for (int i = 0; i < a1.length; i++) {      // add a1 into result
        a3[i] += a1[i];
    }
    for (int i = 0; i < a2.length; i++) {      // add a2 into result
        a3[i] += a2[i];
    }
    return a3;
}

public static double[] arraySum(double[] a1, double[] a2) {
    int minLength = Math.min(a1.length, a2.length);
    int maxLength = Math.max(a1.length, a2.length);
    double[] a3 = new double[maxLength];            // create result array

    for (int i = 0; i < minLength; i++) {
        a3[i] = a1[i] + a2[i];
    }
    for (int i = minLength; i < maxLength; i++) {   // add a1,a2 into result
        if (a1.length > a2.length) {
            a3[i] = a1[i];
        } else {
            a3[i] = a2[i];
        }
    }
    return a3;
}

public static double[] arraySum(double[] a1, double[] a2) {
    double[] shorter = a1;
    double[] longer = a2;
    if (a1.length > a2.length) {
        shorter = a2;
        longer = a1;
    }
    double[] a3 = new double[longer.length];
    for (int i = 0; i < shorter.length; i++) {
        a3[i] = shorter[i] + longer[i];
    }
    for (int i = shorter.length; i < longer.length; i++) {
        a3[i] += longer[i];
    }

    return a3;
}
```

7. Array Programming (six solutions shown)

```
public static void partition(int[] a, int v) {
    int i2 = a.length - 1;
    for (int i1 = 0; i1 < i2; i1++) {
        while (i2 > i1 && a[i2] >= v) {
            i2--;
        }
        int temp = a[i1];
        a[i1] = a[i2];
        a[i2] = temp;
    }
}

public static void partition(int[] a, int v) {
    int i1 = 0;
    int i2 = a.length - 1;
    while (true) {
        while (i2 > i1 && a[i2] >= v) {
            i2--;
        }
        while (i2 > i1 && a[i1] <= v) {
            i1++;
        }
        if (i1 >= i2) {
            break;
        }

        int temp = a[i1];
        a[i1] = a[i2];
        a[i2] = temp;
    }
}

public static void partition(int[] a, int v) {
    int[] copy = new int[a.length];
    int target = 0;

    for (int i = 0; i < a.length; i++) {
        if (a[i] < v) {
            copy[target] = a[i];
            target++;
        }
    }

    for (int i = 0; i < a.length; i++) {
        if (a[i] > v) {
            copy[target] = a[i];
            target++;
        }
    }

    for (int i = 0; i < a.length; i++) {
        a[i] = copy[i];
    }
}

public static void partition(int[] a, int v) {
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a.length - 1; j++) {
            if (a[j] > a[j + 1]) {
                int temp = a[j];
                a[j + 1] = a[j];
                a[j] = temp;
            }
        }
    }
}
```

```
public static void partition(int[] a, int v) {
    for (int i = 0; i < a.length; i++) {
        int smallest = i;
        for (int j = i + 1; j < a.length; j++) {
            if (a[j] < a[smallest]) {
                smallest = j;
            }
        }
        int temp = a[i];
        a[i] = a[smallest];
        a[smallest] = temp;
    }
}

public static void partition(int[] a, int v) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] > a[i + 1]) {
            int temp = a[i];
            a[i + 1] = a[i];
            a[i] = temp;
            partition(a, v);
        }
    }
}
```

8. Classes and Objects (nine solutions shown)

```
public int absoluteDay() {
    int myMonth = month;
    int myDay = day;
    month = 1;
    day = 1;
    int count = 1;
    while (month != myMonth || day != myDay) {
        count++;
        nextDay();
    }
    return count;
}

public int absoluteDay() {
    Date temp = new Date(1, 1);
    int count = 1;
    while (day != temp.day || month != temp.month) {
        count++;
        temp.nextDay();
    }
    return count;
}

public int absoluteDay() {
    int count = day;
    Date temp = new Date(1, 1);
    for (int i = 1; i < month; i++) {
        count += temp.daysInMonth();
        temp.month++;
    }
    return count;
}

public int absoluteDay() {
    int count = 0;
    for (int i = 1; i <= month - 1; i++) {
        if (i == 4 || i == 6 || i == 9 || i == 11) {
            count += 30;
        } else if (i == 2) {
            count += 28;
        } else {
            count += 31;
        }
    }
    count += day;
    return count;
}

public int absoluteDay() {
   int[] dayCount = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
   int count = 0;
   for (int i = 1; i <= month - 1; i++) {
      count += dayCount[i - 1];
   }
   return count + day;
}

public int absoluteDay() {
    Date copy = new Date(month, day);
    int count = 365;
    while (copy.getMonth() != 12 || copy.getDay() != 31) {
        copy.nextDay();
        count--;
    }
    return count;
}
```

```
public int absoluteDay() {
    int oldMonth = month;
    int oldDay = day;
    int count = 0;
    while (month != 12 || day != 31) {
        nextDay();
        count++;
    }
    month = oldMonth;
    day = oldDay;
    return 365 - count;
}

public int absoluteDay() {
    int oldMonth = month;
    month = 0;
    int count = 0;
    for (int i = 1; i < oldMonth; i++) {
        month = i;
        count += daysInMonth();
    }
    count += day;
    return count;
}

public int absoluteDay() {
    int oldMonth = month;
    int count = 0;
    for (int i = month; i > 1; i--) {
        month--;
        count += daysInMonth();
    }
    count += day;
    month = oldMonth;
    return count;
}
```