CSE 142 Sample Final Exam #4

(based on Winter 2008's final)

1. Array Mystery

Consider the following method:

```
public static void arrayMystery(int[] a) {
    for (int i = a.length - 1; i >= 0; i--) {
        if (a[i] == a[a.length - i - 1]) {
            a[i]++;
            a[a.length - 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, 8, 3, 8, 7};`
`arrayMystery(a1);`
	______________________________

`int[] a2 = {4, 0, 0, 4, 0, 0, 4, 0};`
`arrayMystery(a2);`
	______________________________

`int[] a3 = {9, 8, 7, 6, 4, 6, 2, 9, 9};`
`arrayMystery(a3);`
	______________________________

`int[] a4 = {42};`
`arrayMystery(a4);`
	______________________________

`int[] a5 = {5, 5, 5, 6, 5, 5, 5};`
`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 x = 0;
        int[] a = new int[2];
        mystery(x, a);
        System.out.println(x + " " + Arrays.toString(a));
        x++;
        a[1] = a.length;
        mystery(x, a);
        System.out.println(x + " " + Arrays.toString(a));
    }

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

3. Inheritance Mystery

Assume that the following classes have been defined:

```
public class Brian extends Lois {
    public void b() {
        a();
        System.out.print("Brian b   ");
    }

    public String toString() {
        return "Brian";
    }
}

public class Lois extends Meg {
    public void a() {
        System.out.print("Lois a   ");
        super.a();
    }

    public void b() {
        System.out.print("Lois b   ");
    }
}
```

```
public class Meg {
    public void a() {
        System.out.print("Meg a   ");
    }

    public void b() {
        System.out.print("Meg b   ");
    }

    public String toString() {
        return "Meg";
    }
}

public class Stewie extends Brian {
    public void a() {
        super.a();
        System.out.print("Stewie a   ");
    }

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

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

```
Meg[] elements = {new Lois(), new Stewie(), new Meg(), new Brian()};
for (int i = 0; i < elements.length; i++) {
    elements[i].a();
    System.out.println();
    elements[i].b();
    System.out.println();
    System.out.println(elements[i]);
    System.out.println();
}
```

4. File Processing

Write a static method named `tokenStats` that accepts as a parameter a `Scanner` containing a series of tokens. It should print out the sum of all the tokens that are legal integers, the sum of all the tokens that are legal doubles but not integers, and the total number of tokens of any kind. For example, if a `Scanner` called `data` contains the following tokens:

```
3 3.14 10 squid 10.x 6.0
```

Then the call of `tokenStats(data);` should print the following output:

```
integers: 13
real numbers: 9.14
total tokens: 6
```

If the `Scanner` has no tokens, the method should print:

```
integers: 0
real numbers: 0.0
total tokens: 0
```

5. File Processing

Write a static method named `pizza` that accepts as its parameter a `Scanner` for an input file. Imagine that a college dorm room contains several boxes of leftover pizza. A complete pizza has 8 slices. The pizza boxes left in the room each contain either an entire pizza, half a pizza (4 slices), or a single slice. Your method's task is to figure out the fewest boxes needed to store all leftover pizza, if all the partial pizzas were consolidated together as much as possible.

Your `pizza` method will read lines from its input `Scanner` where each line of data represents the contents of the pizza boxes in one dorm room. These contents are written as `"whole"`, `"half"`, or `"slice"` in either upper or lower case, separated by at least one space. You should print to the console the number of pizza boxes necessary to store all the slices of pizza out of the total. You must use a whole number of boxes. For example, if there are 10 total slices of pizza in a dorm room, 2 pizza boxes are needed: one for the first whole pizza, and one for the final 2 slices. Note that some lines might be blank, meaning that the dorm room contains no pizzas; output for such a case is shown below.

For example, consider the following input file representing 5 dorm rooms (note that the fourth is blank):

```
slice half slice whole whole half half half
whole HALF   WhoLE half WHOLE  WhOlE Slice half    sLICe halF
WHOLE slice WHOLE  SLICE whole SLICE   whole slice WHOLE  half

slice
```

For the input above, your method should produce the following output:

```
5 / 8 pizza boxes used
7 / 10 pizza boxes used
6 / 10 pizza boxes used
0 / 0 pizza boxes used
1 / 1 pizza boxes used
```

The format of your output must exactly match that shown above.
Assume the `Scanner` contains at least 1 line of input, and that no line will contain tokens other than whole/half/slice.

6. Array Programming

Write a static method named `sweep` that accepts an array of integers as a parameter and performs a single "sweep" over the array from lowest to highest index, comparing adjacent elements. If a pair of adjacent elements are out of order (if the element at the lower index has a greater value than the element at the higher index), your method should swap them. For example, in an array of 6 elements, your method first examines elements at indexes 0-1, then 1-2, then 2-3, 3-4, and 4-5, swapping if they are out of order. One side effect of this method is that the single element with the largest value will be moved into the final index of the array. (Repeated "sweeping" can be used to sort an array.)

If your method ends up swapping any elements, it should return `true` to indicate that the array was changed. Otherwise (if the array was already in ascending order before the sweep), it should return `false`.

The table below shows calls to your method and the array contents after the method is finished executing:

Array / Call	Contents of Array After Call	Value Returned

`int[] a1 = {1, 6, 2, 7, 3, 6, 4};`
`sweep(a1);`
	`{1, 2, 6, 3, 6, 4, 7}`	`true`

`int[] a2 = {9, 4, 2, 1, 3, 12, 14, 6, 8};`
`sweep(a2);`
	`{4, 2, 1, 3, 9, 12, 6, 8, 14}`	`true`

`int[] a3 = {3, 4, 8, 2, 1, 8, 8, 4, 12};`
`sweep(a3);`
	`{3, 4, 2, 1, 8, 8, 4, 8, 12}`	`true`

`int[] a4 = {-1, -4, 17, 4, -1};`
`sweep(a4);`
	`{-4, -1, 4, -1, 17}`	`true`

`int[] a5 = {2, 3, 5, 7, 11, 13};`
`sweep(a5);`
	`{2, 3, 5, 7, 11, 13}`	`false`

`int[] a6 = {42};`
`sweep(a6);`
	`{42}`	`false`

You may assume that the array contains at least one element (its length is at least 1). Do not make assumptions about the values of the elements in the array; they could be very large or small, etc.

7. Array Programming

Write a static method named `hasMirrorTwice` that accepts two arrays of integers *a1* and *a2* as parameters and returns `true` if *a1* contains all the elements of *a2* in reverse order at least twice (and `false` otherwise). For example, if *a2* stores the elements `{1, 2, 3}` and *a1* stores the elements `{6,` **`3, 2, 1`**`, 4, 1,` **`3, 2, 1`**`, 5}`, your method would return `true`.

Assume that both arrays passed to your method will have a length of at least 1. This means that the shortest possible mirror will be of length 1, representing a single element (which is its own mirror). A sequence that is a palindrome (the same forwards as backwards) is considered its own mirror and should be included in your computations. For example, if *a1* is `{6,` **`1, 2, 1`**`, 4,` **`1, 2, 1`**`, 5}` and *a2* is `{1, 2, 1}`, your method should return `true`. The two occurrences of the mirror might overlap, as shown in the fourth sample call below.

The following table shows some calls to your method and their expected results:

Array	Returned Value
`int[] a1 = {6, 1,` **`2, 1`**`, 3, 1, 3,` **`2, 1`**`, 5};`	`hasMirrorTwice(a1, a2)` returns `true`
`int[] a2 = {1, 2};`	
`int[] a3 = {`**`5, 8, 4`**`, 18, 5, 42, 4, 8, 5, 5};`	`hasMirrorTwice(a3, a4)` returns `false`
`int[] a4 = {4, 8, 5};`	
`int[] a5 = {6,` **`3, 42`**`, 18, 12, 5,` **`3, 42`**`,` **`3, 42`**`};`	`hasMirrorTwice(a5, a6)` returns `true`
`int[] a6 = {42, 3};`	
`int[] a7 = {6,` **`1, 2, 4, 2, 1, 2, 4, 2, 1`**`, 5};`	`hasMirrorTwice(a7, a8)` returns `true`
`int[] a8 = {1, 2, 4, 2, 1};`	
`int[] a9 = {`**`0`**`,` **`0`**`};`	`hasMirrorTwice(a9, aa)` returns `true`
`int[] aa = {0};`	
`int[] ab = {8, 9, 2, 1};`	`hasMirrorTwice(ab, ac)` returns `false`
`int[] ac = {5, 7, 1, 2, 9, 8};`	

Do not modify the contents of the arrays passed to your method as parameters.

8. Critters

Write a class `Crab` that extends the `Critter` class from the Critters assignment, along with its movement behavior. `Crab` objects move in the following repeating pattern:

- west 1 time, east 2 times
- west 3 times, east 4 times
- west 5 times, east 6 times
- west 7 times, east 8 times
- west 7 times, east 6 times
- west 5 times, east 4 times
- west 3 times, east 2 times
- *(entire pattern repeats)*

Write your complete `Crab` class below. All other aspects of `Crab` besides movement use the default critter behavior. You may add anything needed to your class (fields, constructors, etc.) to implement this behavior appropriately.

Note: You will receive a maximum of only 5 points if you "hard-code" the entire set of movements (by simply keeping a move counter and using an extremely long chain of `if` or `if/else` statements) rather than recognizing the crab's overall movement pattern.

9. 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 a method named `daysTillXmas` that will be placed inside the `Date` class to become a part of each `Date` object's behavior. The `daysTillXmas` method returns how many days away the `Date` object is from Christmas, December 25, in the same year. For example:

- **Dec. 12** is **13** days away from Christmas.
- **Nov. 22** is **33** days away (the 8 remaining days in November, and 25 more to reach 12/25).
- **Sep. 3** is **113** days away (the 27 remaining days in September, the 61 days in October and November, and 25 more to reach 12/25).

Here is an example call of your method that would print 113:

```
Date d = new Date(9, 3);
System.out.println(d.daysTillXmas());    // 113
```

Here are more dates and the values that the method should return when called on `Date` objects representing them. 12/25 is 0 days away from itself, and days after 12/25 return a negative value.

- **Dec. 25** is **0** days away from Christmas.
- **Dec. 26** is **-1** days away from Christmas.
- **Dec. 31** is **-6** days away from Christmas.
- **Jan. 1** is **358** days away from Christmas.
- **Feb. 14** is **314** days away from Christmas.

Your method should not modify the `Date` object's state, such as by changing its `day` or `month` field values. You will not receive full credit if your solution uses 12 `if/else` statements or explicitly enumerates the days in each month.

```
// 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, 8, 3, 8, 7};` `arrayMystery(a1);`	`[1, 10, 5, 10, 7]`
`int[] a2 = {4, 0, 0, 4, 0, 0, 4, 0};` `arrayMystery(a2);`	`[4, 0, 2, 4, 0, 2, 4, 0]`
`int[] a3 = {9, 8, 7, 6, 4, 6, 2, 9, 9};` `arrayMystery(a3);`	`[11, 8, 7, 8, 6, 8, 2, 9, 11]`
`int[] a4 = {42};` `arrayMystery(a4);`	`[44]`
`int[] a5 = {5, 5, 5, 6, 5, 5, 5};` `arrayMystery(a5);`	`[7, 7, 7, 8, 7, 7, 7]`

2. Reference Semantics Mystery

```
1 [-1, 0]
0 [-1, 0]
2 [-1, 1]
1 [-1, 1]
```

3. Inheritance Mystery

```
Lois a   Meg a
Lois b
Meg

Lois a   Meg a   Stewie a
Lois a   Meg a   Stewie a   Brian b
Brian Stewie

Meg a
Meg b
Meg

Lois a   Meg a
Lois a   Meg a   Brian b
Brian
```

4. File Processing (four solutions shown)

```
public static void tokenStats(Scanner s) {
    int ints = 0;
    double doubles = 0.0;
    int total = 0;

    while (s.hasNext()) {
        if (s.hasNextInt()) {
            ints += s.nextInt();
        } else if (s.hasNextDouble()) {
            doubles += s.nextDouble();
        } else {
            s.next();
        }
        total++;
    }

    System.out.println("integers: " + ints);
    System.out.println("real numbers: " + doubles);
    System.out.println("total tokens: " + total);
}
```

5. File Processing (four solutions shown)

```
public static void pizza(Scanner input) {
    while (input.hasNextLine()) {
        String line = input.nextLine();
        Scanner lineScan = new Scanner(line.toLowerCase());
        double boxes = 0.0;
        int totalBoxes = 0;
        while (lineScan.hasNext()) {
            totalBoxes++;
            String box = lineScan.next();
            if (box.equals("whole")) {
                boxes++;            // 1
            } else if (box.equals("half")) {
                boxes += 0.5;       // 1/2
            } else {  // "slice"
                boxes += 0.125;     // 1/8
            }
        }
        System.out.println((int) Math.ceil(boxes) + " / " + totalBoxes +
                           " pizza boxes used");
    }
}

public static void pizza(Scanner input) {
    while (input.hasNextLine()) {
        Scanner lineScan = new Scanner(input.nextLine().toLowerCase());
        int slices = 0;
        int totalBoxes = 0;
        while (lineScan.hasNext()) {
            totalBoxes++;
            String box = lineScan.next();
            if (box.equals("whole")) {
                slices += 8;
            } else if (box.equals("half")) {
                slices += 4;
            } else {
                slices++;
            }
        }

        int boxes = slices / 8;   // round up to an even box
        if (slices % 8 != 0) {
            boxes++;
        }
        System.out.println(boxes + " / " + totalBoxes + " pizza boxes used");
    }
}
```

```
public static void pizza(Scanner input) {
    while (input.hasNextLine()) {
        Scanner lineScan = new Scanner(input.nextLine().toLowerCase());
        int slices = 0;
        int totalBoxes = 0;
        while (lineScan.hasNext()) {
            totalBoxes++;
            String box = lineScan.next();
            if (box.equals("whole")) {
                slices += 8;
            } else if (box.equals("half")) {
                slices += 4;
            } else {
                slices++;
            }
        }

        if (slices % 8 != 0) {              // round up to an even box
            slices += (8 - slices % 8);
        }
        System.out.println(slices / 8 + " / " + totalBoxes + " pizza boxes used");
    }
}

public static void pizza(Scanner input) {
    while (input.hasNextLine()) {
        Scanner lineScan = new Scanner(input.nextLine().toLowerCase());
        int slices = 0;
        int halves = 0;
        int wholes = 0;
        int totalBoxes = 0;
        while (lineScan.hasNext()) {
            totalBoxes++;
            String box = lineScan.next();
            if (box.equals("whole")) {
                wholes++;
            } else if (box.equals("half")) {
                halves++;
            } else {
                slices++;
            }
        }

        int boxes = (int) Math.ceil((8*wholes + 4*halves + slices) / 8.0);
        System.out.println(boxes + " / " + totalBoxes + " pizza boxes used");
    }
}
```

6. Array Programming (two solutions shown)

```
public static boolean sweep(int[] a) {
    boolean changed = false;
    for (int i = 0; i < a.length - 1; i++) {
        if (a[i] > a[i + 1]) {
            int temp = a[i];
            a[i] = a[i + 1];
            a[i + 1] = temp;
            changed = true;
        }
    }
    return changed;
}

public static boolean sweep(int[] a) {
    int count = 0;
    for (int i = 1; i < a.length; i++) {
        int temp1 = a[i];
        int temp2 = a[i - 1];

        if (a[i - 1] > a[i]) {
            a[i - 1] = temp1;
            a[i] = temp2;
            count++;
        }
    }

    if (count > 0) {
        return true;
    } else {
        return false;
    }
}
```

7. Array Programming (four solutions shown)

```
public static boolean hasMirrorTwice(int[] a1, int[] a2) {
    int mirrorCount = 0;
    for (int i = 0; i <= a1.length - a2.length; i++) {
        int count = 0;
        for (int j = 0; j < a2.length; j++) {
            if (a1[i + j] == a2[a2.length - 1 - j]) {
                count++;
            }
        }
        if (count >= a2.length) {
            mirrorCount++;
        }
    }

    return mirrorCount >= 2;
}

public static boolean hasMirrorTwice(int[] a1, int[] a2) {
    int mirrorCount = 0;
    for (int i = a2.length - 1; i < a1.length; i++) {
        int count = 0;
        for (int j = 0; j < a2.length; j++) {
            if (a1[i - j] == a2[j]) {
                count++;
            }
        }
        if (count >= a2.length) {
            mirrorCount++;
        }
    }

    return mirrorCount >= 2;
}

public static boolean hasMirrorTwice(int[] a1, int[] a2) {
    // copy/reverse a2 into a3
    int[] a3 = new int[a2.length];
    for (int i = 0; i < a2.length; i++) {
        a3[i] = a2[a2.length - 1 - i];
    }

    int mirrorCount = 0;
    for (int i = 0; i <= a1.length - a3.length; i++) {
        int count = 0;
        for (int j = 0; j < a3.length; j++) {
            if (a1[i + j] == a3[j]) {
                count++;
            }
        }
        if (count >= a3.length) {
            mirrorCount++;
        }
    }

    if (mirrorCount >= 2) {
        return true;
    } else {
        return false;
    }
}

public static boolean hasMirrorTwice(int[] a1, int[] a2) {
    int mirrorCount = 0;
    for (int i = a1.length - 1; i >= 0; i--) {
        int count = 0;
        for (int j = 0; j < a2.length; j++) {
            if (i + j < a1.length && a1[i + j] == a2[a2.length - 1 - j]) {
                count++;
            }
        }
        if (count >= a2.length) {
            mirrorCount++;
        }
    }

    return mirrorCount >= 2;
}
```

8. Critters (two solutions shown)

```
public class Crab extends Critter {
    private int max = 1;
    private int moves = 0;
    private boolean wider = true;

    public Direction getMove() {
        if (moves == max) {
            moves = 0;
            if (wider) {
                max++;
            } else {
                max--;
            }
            if (max == 8 || max == 1) {
                wider = !wider;
            }
        }

        moves++;
        if (max % 2 == 0) {
            return Direction.EAST;
        } else {
            return Direction.WEST;
        }
    }
}

public class Crab extends Critter {
    private int moves = 0;

    public Direction getMove() {
        moves++;
        if (moves == 64) {   // 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8  = 36
            moves = 1;       // 36 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 64
        }

        int sum = 0;
        for (int i = 1; i <= 8; i++) {
            if (moves >= sum && moves <= sum + i) {
                if (i % 2 == 0) { return Direction.EAST; }
                else            { return Direction.WEST; }
            }
            sum += i;
        }

        for (int i = 7; i >= 2; i--) {
            if (moves >= sum && moves <= sum + i) {
                if (i % 2 == 0) { return Direction.EAST; }
                else            { return Direction.WEST; }
            }
            sum += i;
        }
        return Direction.CENTER;   // it will never reach here
    }
}
```

9. Objects (two solutions shown)

```
public int daysTillXmas() {
    int days = 0;
    Date copy = new Date(month, day);
    while (copy.month < 12) {
        // get to December
        days += copy.daysInMonth();
        copy.month++;
    }
    return days + 25 - day;
}

public int daysTillXmas() {
    if (month == 12) {
        return 25 - day;
    } else {
        int days = 0;
        Date copy = new Date(month, day);
        while (copy.month != 12 || copy.day != 25) {
            // get to December
            copy.nextDay();
            days++;
        }
        return days;
    }
}
```