CSE 142 Sample Final Exam #8

(based on Summer 2009's final; thanks to Victoria Kirst)

1. Array Mystery

Consider the following method:

```
public static void arrayMystery(int[] a) {
    for (int i = 1; i < a.length - 1; i++) {
        a[i] = a[i + 1] + 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 = {3, 7};` `arrayMystery(a1);`	_____
`int[] a2 = {4, 7, 4, 2, 10, 9};` `arrayMystery(a2);`	_____
`int[] a3 = {1, 5, 0, 0, 5, 0};` `arrayMystery(a3);`	_____
`int[] a4 = {13, 0, -4, -2, 0, -1};` `arrayMystery(a4);`	_____
`int[] a5 = {2, 4, 6, 8, 16};` `arrayMystery(a5);`	_____

2. Reference Semantics Mystery

(Missing; we didn't give this type of question that quarter.)

3. Inheritance Mystery

Assume that the following classes have been defined:

```
public class Denny extends John {
    public void method1() {
        System.out.print("denny 1  ");
    }

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

public class Cass {
    public void method1() {
        System.out.print("cass 1  ");
    }

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

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

```
public class Michelle extends John {
    public void method1() {
        System.out.print("michelle 1 ");
    }
}

public class John extends Cass {
    public void method2() {
        method1();
        System.out.print("john 2  ");
    }

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

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

```
Cass[] elements = {new Cass(), new Denny(), new John(), new Michelle()};
for (int i = 0; i < elements.length; i++) {
    elements[i].method1();
    System.out.println();
    elements[i].method2();
    System.out.println();
    System.out.println(elements[i]);
    System.out.println();
}
```

4. File Processing

Write a static method called `runningSum` that accepts as a parameter a `Scanner` holding a sequence of real numbers and that outputs the running sum of the numbers followed by the maximum running sum. In other words, the *n*th number that you report should be the sum of the first *n* numbers in the `Scanner` and the maximum that you report should be the largest such value that you report. For example if the `Scanner` contains the following data:

```
3.25 4.5 -8.25 7.25 3.5 4.25 -6.5 5.25
```

your method should produce the following output:

```
running sum = 3.25 7.75 -0.5 6.75 10.25 14.5 8.0 13.25
max sum = 14.5
```

The first number reported is the same as the first number in the `Scanner` (3.25). The second number reported is the sum of the first two numbers in the `Scanner` (3.25 + 4.5). The third number reported is the sum of the first three numbers in the `Scanner` (3.25 + 4.5 + -8.25). And so on. The maximum of these values is 14.5, which is reported on the second line of output. You may assume that there is at least one number to read.

5. File Processing

Write a static method named `plusScores` that accepts as a parameter a `Scanner` containing a series of lines that represent student records. Each student record takes up two lines of input. The first line has the student's name and the second line has a series of plus and minus characters. Below is a sample input:

```
Kane, Erica
--+-+
Chandler, Adam
++-+
Martin, Jake
+++++++
Dillon, Amanda
++-++-+-
```

The number of plus/minus characters will vary, but you may assume that at least one such character appears and that no other characters appear on the second line of each pair. For each student you should produce a line of output with the student's name followed by a colon followed by the percent of plus characters. For example, if the input above is stored in a `Scanner` called `input`, the call of `plusScores(input);` should produce the following output:

```
Kane, Erica: 40.0% plus
Chandler, Adam: 75.0% plus
Martin, Jake: 100.0% plus
Dillon, Amanda: 62.5% plus
```

6. Array Programming

Write a method `priceIsRight` that accepts an array of integers *bids* and an integer *price* as parameters. The method returns the element in the *bids* array that is closest in value to *price* without being larger than *price*. For example, if *bids* stores the elements `{200, 300, 250, 999, 40}`, then `priceIsRight(bids, 280)` should return `250`, since 250 is the bid closest to 280 without going over 280. If all bids are larger than *price*, then your method should return -1.

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

Arrays	Returned Value
`int[] a1 = {900, 885, 989, 1};`	`priceIsRight(a1, 880)` returns `1`
`int[] a2 = {200};`	`priceIsRight(a2, 120)` returns `200`
`int[] a3 = {500, 300, 241, 99, 501};`	`priceIsRight(a3, 50)` returns `-1`

You may assume there is at least 1 element in the array, and you may assume that the *price* and the values in *bids* will all be greater than or equal to 1. Do not modify the contents of the array passed to your method as a parameter.

7. Array Programming

Write a static method named `compress` that accepts an array of integers *a1* as a parameter and returns a new array that contains only the unique values of *a1*. The values in the new array should be ordered in the same order they originally appeared in. For example, if *a1* stores the elements {`10, 10, 9, 4, 10, 4, 9, 17`}, then `compress(a1)` should return a new array with elements {`10, 9, 4, 17`}.

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

Array	Returned Value
`int[] a1 = {5, 2, 5, 3, 2, 5};`	`compress(a1)` returns {`5, 2, 3`}
`int[] a2 = {-2, -12, 8, 8, 2, 12};`	`compress(a2)` returns {`-2, -12, 8, 2, 12`}
`int[] a3 = {4, 17, 0, 32, -3, 0, 0};`	`compress(a3)` returns {`4, 17, 0, 32, -3`}
`int[] a4 = {-2, -5, 0, 5, -92, -2, 0, 43};`	`compress(a4)` returns {`-2, -5, 0, 5, -92, 43`}
`int[] a5 = {1, 2, 3, 4, 5};`	`compress(a5)` returns {`1, 2, 3, 4, 5`}
`int[] a6 = {5, 5, 5, 5, 5, 5};`	`compress(a6)` returns {`5`}
`int[] a7 = {};`	`compress(a7)` returns {}

Do not modify the contents of the array passed to your method as a parameter.

8. Critters

Write a class `Caterpillar` that extends the `Critter` class from our assignment, along with its movement behavior.

`Caterpillar`s move in an increasing NESW square pattern: 1 move north, 1 move east, 1 move west, 1 move south, then 2 moves north, 2 moves east, etc., the square pattern growing larger and larger indefinitely. If a `Caterpillar` runs into a piece of food, the `Caterpillar` eats the food and immediately restarts the NESW pattern. The size of the `Caterpillar`'s movement is also reset back to 1 move in each direction again, and the increasing square pattern continues as before until another piece of food is encountered.

Here is a sample movement pattern of a `Caterpillar`:

- north 1 time, east 1 time, south 1 time, west 1 time
- north 2 times, east 2 times, south 2 times, west 2 times
- north 3 times, east 3 times, south 3 times, west 3 times
- *(runs into food)*
- north 1 time, east 1 time, south 1 time, west 1 time
- north 2 times, east 1 time
- *(runs into food)*
- north 1 time
- *(runs into food)*
- north 1 time, east 1 time, south 1 time, west 1 time
- north 2 times, east 2 times, south 2 times, west 2 times
- *(etc.)*

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

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 an instance method named **`subtractWeeks`** that will be placed inside the `Date` class to become a part of each `Date` object's behavior. The `subtractWeeks` method accepts an integer as a parameter and shifts the date represented by the `Date` object backward by that many weeks. A week is considered to be exactly 7 days. You may assume the value passed is non-negative. Note that subtracting weeks might cause the date to wrap into previous months or years.

For example, if the following `Date` is declared in client code:

```
Date d = new Date(9, 19);
```

The following calls to the `subtractWeeks` method would modify the `Date` object's state as indicated in the comments. Remember that `Date` objects do not store the year. The date before January 1st is December 31st. `Date` objects also ignore leap years.

```
Date d = new Date(9, 19);
d.subtractWeeks(1);    // d is now 9/12
d.subtractWeeks(2);    // d is now 8/29
d.subtractWeeks(5);    // d is now 7/25
d.subtractWeeks(20);   // d is now 3/7
d.subtractWeeks(110);  // d is now 1/26
                       //  (2 years prior)
```

```
// 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 = {3, 7};` `arrayMystery(a1);`	`[3, 7]`
`int[] a2 = {4, 7, 4, 2, 10, 9};` `arrayMystery(a2);`	`[4, 8, 10, 20, 29, 9]`
`int[] a3 = {1, 5, 0, 0, 5, 0};` `arrayMystery(a3);`	`[1, 1, 1, 6, 6, 0]`
`int[] a4 = {13, 0, -4, -2, 0, -1};` `arrayMystery(a4);`	`[13, 9, 7, 7, 6, -1]`
`int[] a5 = {2, 4, 6, 8, 16};` `arrayMystery(a5);`	`[2, 8, 16, 32, 16]`

2. Reference Semantics Mystery

(missing)

3. Inheritance Mystery

```
cass 1
cass 2
cass

denny 1
denny 1  john 2
denny john

cass 1
cass 1  john 2
john

michelle 1
michelle 1 john 2
john
```

4. File Processing (two solutions shown)

```
public static void runningSum(Scanner input) {
    double sum = input.nextDouble();
    double max = sum;
    System.out.print("running sum = " + sum);
    while (input.hasNextDouble()) {
        sum += input.nextDouble();
        System.out.print(" " + sum);
        if (sum > max) {
            max = sum;
        }
    }
    System.out.println();
    System.out.println("max sum = " + max);
}
```

```
public static void runningSum(Scanner input) {
    double sum = 0;
    double max = 0;
    System.out.print("running sum =");
    while (input.hasNext()) {
        sum += input.nextDouble();
        System.out.print(" " + sum);
        max = Math.max(max, sum);
    }
    System.out.println("\nmax sum = " + max);
}
```

5. File Processing (two solutions shown)

```
public static void plusScores(Scanner input) {
    while (input.hasNextLine()) {
        String name = input.nextLine();
        String data = input.nextLine();
        int plus = 0;
        int count = 0;

        for (int i = 0; i < data.length(); i++) {
            count++;
            if (data.charAt(i) == '+') {
                plus++;
            }
        }

        double percent = 100.0 * plus / count;
        System.out.println(name + ": " + percent + "% plus");
    }
}
```

```
public static void plusScores(Scanner s) {
    while (s.hasNextLine()) {
        System.out.print(s.nextLine() + ": ");
        String p = s.nextLine();
        System.out.print(100.0 * p.replace("-", "").length() / p.length() + "% plus");
    }
}
```

6. Array Programming (three solutions shown)

```
public static int priceIsRight(int[] bids, int price) {
    int bestPrice = -1;
    for (int i = 0; i < bids.length; i++) {
        if (bids[i] <= price && bids[i] > bestPrice) {
            bestPrice = bids[i];
        }
    }
    return bestPrice;
}

public static int priceIsRight(int[] bids, int price) {
    int[] difference = new int[bids.length];
    int bestAnswer = Integer.MIN_VALUE;
    for (int i = 0; i < difference.length; i++) {
        difference[i] = bids[i] - price;
    }
    for (int i = 0; i < difference.length; i++) {
        if (difference[i] <= 0) {
            bestAnswer = (int) Math.max(difference[i], bestAnswer);
        }
    }
    if (bestAnswer == Integer.MIN_VALUE) {
        return -1;
    } else {
        return bestAnswer + price;
    }
}

public static int priceIsRight(int[] prices, int price) {
    int bestPrice = prices[0];
    for (int i = 1; i < prices.length; i++) {
        if (prices[i] <= price && prices[i] > bestPrice) {
            bestPrice = prices[i];
        }
    }

    if (bestPrice <= price) {
        return bestPrice;
    } else {
        return -1;
    }
}
```

7. Array Programming (two solutions shown)

```
// faster; makes use of sorted nature of array
public static int[] compress(int[] a1) {
    int[] unique = new int[a1.length];
    int uniqueCount = 1;
    unique[0] = a1[0];
    for (int i = 0; i < a1.length - 1; i++) {
        if (a1[i] != a1[i + 1]) {
            unique[uniqueCount] = a1[i + 1];
            uniqueCount++;
        }
    }

    int[] compressed = new int[uniqueCount];
    for (int i = 0; i < compressed.length; i++) {
        compressed[i] = unique[i];
    }
    return compressed;
}

// slower, but works for any array
public static int[] compress(int[] a1) {
    int[] unique = new int[a1.length];
    int numUnique = 0;
    for (int i = 0; i < a1.length; i++) {
        int count = 0;
        for (int j = 0; j < numUnique; j++) {
            if (a1[i] == unique[j]) {
                count++;
            }
        }
        if (count == 1) {
            unique[numUnique++] = a1[i];
        }
    }
    return Arrays.copyOf(unique, numUnique);
}
```

8. Critters

```
public class Caterpillar extends Critter {
    private int stepLength = 1;
    private int steps = 0;

    public boolean eat() {
        steps = 0;
        stepLength = 1;
        return true;
    }

    public Direction getMove() {
        steps++;
        if (steps > 4 * stepLength) {
            stepLength++;
            steps = 1;
        }

        if (steps <= stepLength) {
            return Direction.NORTH;
        } else if (steps <= 2 * stepLength) {
            return Direction.EAST;
        } else if (steps <= 3 * stepLength) {
            return Direction.SOUTH;
        } else {
            return Direction.WEST;
        }
    }
}
```

9. Objects

```
public void subtractWeeks(int weeks) {
    for (int i = 1; i <= weeks; i++) {
        day -= 7;
        if (day <= 0) {
            month--;
            if (month == 0) {
                month = 12;
            }
            day += daysInMonth();
        }
    }
}

public void subtractWeeks(int weeks) {
    int days = 7 * weeks;
    for (int i = 1; i <= days; i++) {
        day--;
        if (day == 0) {
            month--;
            if (month == 0) {
                month = 12;
            }
            day = daysInMonth();
        }
    }
}

public void subtractWeeks(int weeks) {
    day -= 7 * weeks;
    while (day <= 0) {
        month--;
        if (month == 0) {
            month = 12;
        }
        day += daysInMonth();
    }
}
```