CSE 142 Sample Final Exam #1

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

1. Expressions

For each expression at left, indicate its value in the right column. List a value of appropriate type and capitalization. e.g., `7` for an `int`, `7.0` for a `double`, `"hello"` for a `String`, `true` or `false` for a `boolean`.

Expression	Value
`8.0 / 4 / 4 + 3 / 4 + 0.5 * 3`	_____
`88 % 10 % 5/2 + 7 % 8`	_____
`2 * 3 + 4 + " = " + 5 * 6 + 7`	_____
`10 + 7 > 20 && 16 % 4 > 2`	_____
`2 * (0.75 + 1.25)/5 * 10 - 7/4`	_____

2. Array Mystery

Consider the following method:

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

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 = {7};` `arrayMystery(a1);`	_____
`int[] a2 = {4, 3, 6};` `arrayMystery(a2);`	_____
`int[] a3 = {7, 4, 8, 6, 2};` `arrayMystery(a3);`	_____
`int[] a4 = {10, 2, 5, 10};` `arrayMystery(a4);`	_____
`int[] a5 = {2, 4, -1, 6, -2, 8};` `arrayMystery(a5);`	_____

3. Reference Semantics Mystery

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

```
public class BasicPoint {
    int x;
    int y;

    public BasicPoint() {
        x = 2;
        y = 2;
    }
}

public class ReferenceMystery {
    public static void main(String[] args) {
        int a = 7;
        int b = 9;
        BasicPoint p1 = new BasicPoint();
        BasicPoint p2 = new BasicPoint();

        addToXTwice(a, p1);
        System.out.println(a + " " + b + " " + p1.x + " " + p2.x);

        addToXTwice(b, p2);
        System.out.println(a + " " + b + " " + p1.x + " " + p2.x);
    }

    public static void addToXTwice(int a, BasicPoint p1) {
        a = a + a;
        p1.x = a;
        System.out.println(a + " " + p1.x);
    }
}
```

4. Inheritance Mystery

Assume that the following classes have been defined:

```
public class A extends B {
    public void method2() {
        System.out.print("a 2  ");
        method1();
    }
}

public class B extends C {
    public String toString() {
        return "b";
    }

    public void method2() {
        System.out.print("b 2  ");
        super.method2();
    }
}
```

```
public class C {
    public String toString() {
        return "c";
    }

    public void method1() {
        System.out.print("c 1  ");
    }

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

public class D extends B {
    public void method1() {
        System.out.print("d 1  ");
        method2();
    }
}
```

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

```
C[] elements = {new A(), new B(), new C(), new D()};
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();
}
```

5. File Processing

Write a static method named `printStrings` that takes as a parameter a `Scanner` holding a sequence of integer/string pairs and that prints to `System.out` one line of output for each pair with the given `String` repeated the given number of times. For example if the `Scanner` contains the following data:

```
6 fun. 3 hello   10    <> 2      25    4  wow!
```

your method should produce the following output:

```
fun.fun.fun.fun.fun.fun.
hellohellohello
<><><><><><><><><><>
2525
wow!wow!wow!wow!
```

Notice that there is one line of output for each integer/string pair. The first line has 6 occurrences of `"fun."`, the second line has 3 occurrences of `"hello"`, the third line has 10 occurrences of `"<>"`, the fourth line has 2 occurrences of `"25"` the fifth line has 4 occurrences of `"wow!"`. Notice that there are no extra spaces included in the output. You are to exactly reproduce the format of this sample output. You may assume that the input values always come in pairs with an integer followed by a `String` (which itself could be numeric, such as `"25"` above). If the `Scanner` is empty (no integer/string pairs), your method should produce no output.

6. File Processing

Write a static method named `reverseLines` that accepts a `Scanner` containing an input file as a parameter and that echoes the input file to `System.out` with each line of text reversed. For example, given the following input file:

```
If this method works properly,
the lines of text in this file
will be reversed.

Remember that some lines might be blank.
```

Your method should print the following output:

```
,ylreporp skrow dohtem siht fI
elif siht ni txet fo senil eht
.desrever eb lliw

.knalb eb thgim senil emos taht rebmemeR
```

Notice that some of the input lines can be blank lines.

7. Array Programming

Write a static method `isAllEven` that takes an array of integers as a parameter and that returns a boolean value indicating whether or not all of the values are even numbers (true for yes, false for no). For example, if a variable called `list` stores the following values:

```
int[] list = {18, 0, 4, 204, 8, 4, 2, 18, 206, 1492, 42};
```

Then the call of `isAllEven(list)` should return `true` because each of these integers is an even number. If instead the list had stored these values:

```
int[] list = {2, 4, 6, 8, 10, 208, 16, 7, 92, 14};
```

Then the call should return `false` because, although most of these values are even, the value 7 is an odd number.

8. Array Programming

Write a static method named `isUnique` that takes an array of integers as a parameter and that returns a boolean value indicating whether or not the values in the array are unique (`true` for yes, `false` for no). The values in the list are considered unique if there is no pair of values that are equal. For example, if a variable called `list` stores the following values:

```
int[] list = {3, 8, 12, 2, 9, 17, 43, -8, 46, 203, 14, 97, 10, 4};
```

Then the call of `isUnique(list)` should return `true` because there are no duplicated values in this list. If instead the list stored these values:

```
int[] list = {4, 7, 2, 3, 9, 12, -47, -19, 308, 3, 74};
```

Then the call should return `false` because the value 3 appears twice in this list. Notice that given this definition, a list of 0 or 1 elements would be considered unique.

9. Critters

Write a class `Ostrich` that extends the `Critter` class from the Critters assignment, including its `getMove` and `getColor` methods. An `Ostrich` object first stays in the same place for 10 moves, then moves 10 steps to either the WEST or the EAST, then repeats. In other words, after sitting still for 10 moves, the ostrich randomly picks to go west or east, then walks 10 steps in that same direction. Then it stops and sits still for 10 moves and repeats. Whenever an `Ostrich` is moving (that is, whenever its last call to `getMove` returned a direction other than `Direction.CENTER`), its color should be white (`Color.WHITE`). As soon as it stops moving, and initially when it first appears in the critter world, its color should be cyan (`Color.CYAN`). When randomly choosing west vs. east, the two directions should be equally likely.

You may add anything needed (fields, other methods) to implement the above behavior appropriately. All other critter behavior not discussed here uses the default values.

Solutions

1.
```
2.0
8
"10 = 307"
false
7.0
```

2.

Call	Final Contents of Array
`int[] a1 = {7};` `arrayMystery(a1);`	`{7}`
`int[] a2 = {4, 3, 6};` `arrayMystery(a2);`	`{4, 2, -2}`
`int[] a3 = {7, 4, 8, 6, 2};` `arrayMystery(a3);`	`{7, 4, -2, -5, -3}`
`int[] a4 = {10, 2, 5, 10};` `arrayMystery(a4);`	`{10, 9, 6, -1}`
`int[] a5 = {2, 4, -1, 6, -2, 8};` `arrayMystery(a5);`	`{2, -1, 2, -1, 5, 2}`

3.
```
14 14
7 9 14 2
18 18
7 9 14 18
```

4.
```
b
c 1
a 2  c 1

b
c 1
b 2  c 2

c
c 1
c 2

b
d 1  b 2  c 2
b 2  c 2
```

5.
```
public static void printStrings(Scanner input) {
    while (input.hasNextInt()) {
        int times = input.nextInt();
        String word = input.next();
        for (int i = 0; i < times; i++) {
            System.out.print(word);
        }
        System.out.println();
    }
}
```

6.

```
public static void reverseLines(Scanner input) {
    while (input.hasNextLine()) {
        String text = input.nextLine();
        for (int i = text.length() - 1; i >= 0; i--) {
            System.out.print(text.charAt(i));
        }
        System.out.println();
    }
}
```

7.

```
public static boolean isAllEven(int[] list) {
    for (int i = 0; i < list.length; i++) {
        if (list[i] % 2 != 0) {
            return false;
        }
    }
    return true;
}
```

8.

```
public static boolean isUnique(int[] list) {
    for (int i = 0; i < list.length; i++) {
        for (int j = i + 1; j < list.length; j++) {
            if (list[i] == list[j]) {
                return false;
            }
        }
    }
    return true;
}
```

9.

```
import java.awt.*;    // for Color
import java.util.*;   // for Random

public class Ostrich extends Critter {
    private Random rand;
    private int steps;
    private boolean west;     // true if going west; false if east
    private boolean hiding;

    public Ostrich() {
        rand = new Random();
        hiding = true;
        steps = 0;
        west = rand.nextBoolean();   // or call nextInt(2) and map 0=false, 1=true
    }

    public Color getColor() {
        if (hiding) {
            return Color.CYAN;
        } else {
            return Color.WHITE;
        }
    }

    public Direction getMove() {
        if (steps == 10) {
            steps = 0;   // Pick a new direction and re-set the steps counter
            hiding = !hiding;
            west = rand.nextBoolean();
        }

        steps++;
        if (hiding) {
            return Direction.CENTER;
        } else if (west) {
            return Direction.WEST;
        } else {
            return Direction.EAST;
        }
    }
}
```