1. Expressions (5 points)

For each expression in the left-hand column, indicate its value in the right-hand column.
Be sure to list a constant of appropriate type (e.g., `7.0` rather than `7` for a `double`, `String`s in "quotes"). If the expression is illegal, then write "error".

Expression	Value
`0.25 * 2 + 1 / 2`	`0.5`
`20 % 3 + 3 % 10 * 5`	`17`
`!(1 == 2) && ((3 > 4) || (5 < 6))`	`true`
`"answer = " + 3 - 2`	`error`
`4 / 2.0 - 2 + 4 / 2`	`2.0`

2. Array Mystery (10 points)

Consider the following method:

```
public static void mystery(int[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i] = array[array.length - 1 - i];
        array[array.length - 1 - i] = array[i];
    }
}
```

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

Array	Final Contents of Array
`int[] a1 = {8};` `mystery(a1);`	`{ 8 }`
`int[] a2 = {14, 7};` `mystery(a2);`	`{ 7, 7 }`
`int[] a3 = {7, 1, 3, 2, 0, 4};` `mystery(a3);`	`{ 4, 0, 2, 2, 0, 4 }`
`int[] a4 = {10, 8, 9, 5, 6};` `mystery(a4);`	`{ 6, 5, 9, 5, 6 }`
`int[] a5 = {8, 10, 8, 6, 4, 2};` `mystery(a5);`	`{ 2, 4, 6, 6, 4, 2 }`

3. Inheritance Mystery (10 points)

Assume that the following classes have been defined:

```
public class Queen extends Joker {
    public void method2() {
        System.out.println("Queen 2");
    }
}

public class King extends Joker {
    public void method2() {
        System.out.println("King 2");
    }

    public String toString() {
        return "I am king!";
    }
}
```

```
public class Joker {
    public void method2() {
        System.out.println("Joker 2");
    }

    public String toString() {
        return "Ha! Ha!";
    }

    public void method1() {
        System.out.println("Joker 1");
    }
}

public class Jack extends King {
    public void method2() {
        System.out.println("Jack 2");
    }
}
```

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

```
Joker[] court = { new King(), new Joker(), new Queen(), new Jack() };
for (int i = 0; i < court.length; i++) {
    court[i].method1();
    System.out.println(court[i]);
    court[i].method2();
    System.out.println();
}
```

```
Joker 1
I am king!
King 2

Joker 1
Ha! Ha!
Joker 2

Joker 1
Ha! Ha!
Queen 2

Joker 1
I am king!
Jack 2
```

4. Mystery (5 points)
Suppose b is a Boolean variable. In 20 words or less, explain what the following line does:

```
b = (b == false);
```

It flips the value of `b`. If `b` is `true`, it becomes `false`, and vice versa. It is equivalent to:

```
b = !b;
```

5. Mystery (4 points)
What is the output of the following?

```
System.out.print("[" + 1);
for (int i = 2; i <= 10; i++) {
    System.out.print("," + i);
    i++;
}
System.out.println("]");
```

```
[1, 2, 4, 6, 8, 10]
```

6. Non-Programming Programming (10 points)

You have information about each person's birthday (*e.g.*, June 6) in the class. How would you find out which birthday is shared by the most people? Describe what data you would need to store and how you would process that data.

Though not at all necessary, you may write some parts of your explanation in code if it helps you organize your thoughts. You may also assume that certain methods exist as long as you describe exactly what the methods do and that they perform a fairly small task, *i.e.*, saying "Call the `computeBirthdaySharedByMostPeople()` method" is **not** a valid answer.

Create an array of size 366 (if you forgot February 29, that's okay). For each birthday, convert it to a number between 1 and 366 representing which day of the year it is. For example, January 1 is the first day of the year, so it will be converted to a 1; February 19 is the fiftieth day of the year, so it will be converted to a 50. For each birthday, tally its number in the array (subtract one to get the right index into the array). After all the birthdays have been tallied, go through the array looking for the largest count. Convert that number back to a month and day and that is the birthday shared by most people.

7. Programming (15 points)

Write a static method `maxWord` that accepts a `Scanner` for an input file. Each line of input has a word followed by some positive numbers. The method should return the word with the maximum sum of numbers. You may assume that the input file is properly formatted and has at least one line.

For example, given the following input file:

```
cse142 176 148 561
rocks 937 889
computer 654 27878
science 583
4life 864 747
```

For a `Scanner` variable named `input` referring to the file above, the call of `maxWord(input)` would return `computer`, since it is followed by numbers whose sum is 28,532 (654 + 27878), which is more than any of the other sums.

```
public static String maxWord(Scanner input) {
    int maxCount = 0;
    String maxName = "";

    while (input.hasNextLine()) {
        Scanner lineScan = new Scanner(input.nextLine());
        String name = lineScan.next();

        int count = 0;
        while (lineScan.hasNextInt()) {
            count += lineScan.nextInt();
        }

        if (count > maxCount) {
            maxCount = count;
            maxName = name;
        }
    }

    return maxName;
}
```

8. Programming (10 points)

Write a static method `canGo` that accepts 3 parameters: a `Scanner` holding a sequence of strings representing sequential bus stops, a string representing a start location, and a string representing an end destination. The method should return `true` if the start location appears before the end location in the sequence of bus stops; otherwise, it should return `false`.

For example, suppose a `Scanner` variable named `stops` contained the following bus stops:

```
UVillage  HUB  UW-Medical  MeanyHall  UW
```

Here are some example calls to the method and their expected return results

Call	Value Returned
`canGo(stops, "UVillage", "MeanyHall")`	`true`
`canGo(stops, "UW-Medical", "UW")`	`true`
`canGo(stops, "MeanyHall", "UVillage")`	`false`
`canGo(stops, "Earth", "UW-Medical")`	`false`

```
public static boolean canGo(Scanner cities, String start, String end) {
    while (cities.hasNext()) {
        if (start.equals(cities.next())) {
            while (cities.hasNext()) {
                if (end.equals(cities.next())) {
                    return true;
                }
            }
        }
    }

    return false;
}
```

9. Programming (15 points)

Write a static method `interleave` that accepts two arrays of integers as parameters. The method returns a new array whose elements are the same as the elements of the two array parameters woven together.

If one array is shorter than the other, then after all the elements of the shorter array are woven in, the remaining elements of the longer array are just copied in sequence to the end of the returned array. The following example should make this all clear.

Assuming the following arrays have been declared:

```
int[] a1 = { 1, 2, 3, 4, 5 };
int[] a2 = { 6, 7, 8, 9, 10 };
int[] a3 = { 11, 12, 13 };
```

Here are some example calls to the method and their expected return results.

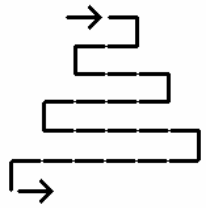
Call	Value Returned
`interleave(a1, a2)`	`{ 1, 6, 2, 7, 3, 8, 4, 9, 5, 10 }`
`interleave(a1, a3)`	`{ 1, 11, 2, 12, 3, 13, 4, 5 }`
`interleave(a2, a3)`	`{ 6, 11, 7, 12, 8, 13, 9, 10 }`
`interleave(a1, a1)`	`{ 1, 1, 2, 2, 3, 3, 4, 4, 5, 5 }`

For this problem, you may assume that the first array will always have a length greater than or equal to that of the second array, *i.e.* the first array will never be shorter than the second array.

```
public static int[] interleave(int[] array1, int[] array2) {
    int[] retVal = new int[array1.length + array2.length];

    for (int i = 0; i < array2.length; i++) {
        retVal[2 * i] = array1[i];
        retVal[2 * i + 1] = array2[i];
    }

    for (int i = array2.length; i < array1.length; i++) {
        retVal[array2.length + i] = array1[i];
    }

    return retVal;
}
```

10. Programming (15 points)

Write the `getMove` method for the class `Snake` that implements the `Critter` interface from Homework 8. Instances of the `Snake` class should move as follows: `EAST` 1 time, `SOUTH` 1 time, `WEST` 2 times, `SOUTH` 1 time, `EAST` 3 times, `SOUTH` 1 time, `WEST` 4 times, `SOUTH` 1 time, `EAST` 5 times, `SOUTH` 1 time, `WEST` 6 times, `SOUTH` 1 time, and so on, each time going `SOUTH` 1 time before flipping directions (`EAST` to `WEST` and vice versa) and moving one step farther than in the previous direction.



Use the constants for directions defined in the `Critter` interface. You may add anything needed (fields, other methods, constructors, etc.) to implement `getMove` appropriately.

```
public class Snake implements Critter {
    // declare any necessary fields and methods here
    private int steps;
    private int max;

    public Snake() {
        steps = 0;
        max = 1;
    }


    // fight, getColor, getChar methods omitted (you do not need to write them)

    public int getMove(CritterInfo info) {
        // complete the getMove method here
        if (steps == max) {
            steps = 0;
            max++;
            return SOUTH;
        }

        steps++;

        if (max % 2 == 1) {
            return EAST;
        } else {
            return WEST;
        }
    }
}
```