CSE 142 Review – Monday Oct. 3, 2016

0. **sameDashes (Strings)**

Write a method `sameDashes` that takes two `String`s as parameters and that returns whether or not they have dashes in the same places (returning `true` if they do and returning `false` otherwise). For example, below are three pairs of `String`s of equal length that have the same pattern of dashes. Notice that the last pair has no dashes at all.

```
String 1: "hi—there-you."     "criminal-plan"     "abc"
String 2: "12--(134)-7539"    "(206)555-1384"     "9.8"
```

To be considered a match, the `String`s must have exactly the same number of dashes in exactly the same positions, but the `String`s might be of different length. For example, the following calls should each return `true`:

```
sameDashes("1st-has-more characters", "2nd-has-less")
sameDashes("1st-has-less", "2nd-has-more characters")
```

because the `String`s each have two dashes and they are in the same positions. But the following calls should each return `false` because the longer `String` has a third dash where the shorter `String` does not:

```
sameDashes("1st-has-more-characters", "2nd-has-less")
sameDashes("1st-has-less", "2nd-has-more-characters")
```

1. **`flipLines` (File Processing with Scanners)**

Write a method named `flipLines` that accepts as its parameter a Scanner for an input file and that writes to the console the same file's contents with successive pairs of lines reversed in order. For example, if the input file contains the following text:

```
Twas brillig and the slithy toves
did gyre and gimble in the wabe.
All mimsey were the borogroves,
and the mome raths outgrabe.

"Beware the Jabberwock, my son,
the jaws that bite, the claws that catch,
Beware the JubJub bird and shun
the frumious bandersnatch."
```

The program should print the first pair of lines in reverse order, then the second pair in reverse order, then the third pair in reverse order, and so on. Therefore your method should produce the following output to the console:

```
did gyre and gimble in the wabe.
Twas brillig and the slithy toves
and the mome raths outgrabe.
All mimsey were the borogroves,
"Beware the Jabberwock, my son,

Beware the JubJub bird and shun
the jaws that bite, the claws that catch,
the frumious bandersnatch."
```

Notice that a line can be blank, as in the third pair. Also notice that an input file can have an odd number of lines, as in the one above, in which case the last line is printed in its original position. You may not make any assumptions about how many lines are in the Scanner.

2. **`repeatedSequence`** (Arrays)

Write a method named `repeatedSequence` that accepts two arrays of integers a1 and a2 as parameters and returns `true` if a2 is composed entirely of repetitions of a1 and `false` otherwise. For example, if a1 stores the elements {2, 1, 3} and a2 stores the elements {2, 1, 3, 2, 1, 3, 2, 1, 3}, the method would return `true`.

If the length of a2 is not a multiple of the length of a1, your method should return `false`. You may assume that both arrays passed to your method will have a length of at least 1.

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

Arrays	Returned Value
`int[] a1 = {2, 1};` `int[] a2 = {2, 1, 2, 1, 2, 1};`	`repeatedSequence(a1, a2)` returns `true`
`int[] a3 = {2, 1, 3};` `int[] a4 = {2, 1, 3, 2, 1, 3, 2};`	`repeatedSequence(a3, a4)` returns `false`
`int[] a5 = {23};` `int[] a6 = {23, 23, 23, 23};`	`repeatedSequence(a5, a6)` returns `true`
`int[] a7 = {5, 6, 7, 8};` `int[] a8 = {5, 6, 7, 8};`	`repeatedSequence(a7, a8)` returns `true`
`int[] a9 = {5, 6};` `int[] a0 = {5, 6, 7, 5, 6, 5};`	`repeatedSequence(a9, a0)` returns `false`

3. Implementing the `Point` class (Objects)
Write a class named Point that represents a two-dimensional Cartesian Point with x and y coordinates.

Your class should have the following methods:

- A constructor that accepts another Point as a parameter and initializes the new Point to have the same (*x*, *y*) values.
- A `toString()` method that returns a String representation of a Point object. For example, if a Point object stores the x-y pair (5, -17) the toString method should return "`Point[x=5,y=-17]`"
- `getX()` and `getY()` methods to return a Point object's x and y coordinate values, respectively
- `setX()` and `setY()` methods to set a Point object's x and y coordinate values, respectively
- a `distance(Point other)` method which returns the distance as a double between the current Point object and the given other Point object. The distance between two points is equal to the square root of the sum of the squares of the differences of their x- and y-coordinates. In other words, the distance between two points (x_1, y_1) and (x_2, y_2) can be expressed as the square root of $(x_2 - x_1)^2 + (y_2 - y_1)^2$. Two points with the same (x, y) coordinates should return a distance of 0.0.

4. `interleave` (ArrayLists)

Write a method called `interleave` that accepts two `ArrayLists` of integers *a1* and *a2* as parameters and inserts the elements of *a2* into *a1* at alternating indices. If the lists are of unequal length, the remaining elements of the longer list are left at the end of *a1*.

For example, if *a1* stores [10, 20, 30] and *a2* stores [4, 5, 6, 7, 8], the call of `interleave(a1, a2);` should change *a1* to store [10, 4, 20, 5, 30, 6, 7, 8]. If *a1* had stored [10, 20, 30, 40, 50] and *a2* had stored [6, 7, 8], the call of `interleave(a1, a2);` would change *a1* to store [10, 6, 20, 7, 30, 8, 40, 50].