Programming Assignment 6-4

There is a Java library class called java.util.Arrays; it has a static method sort, which will arrange in sorted order any array of objects that have a natural ordering, like Integers and Strings.

Example: Obtain `Integer[] arr = new Integer[6]`. Place values in `arr`:
`arr[0] = 1, arr[1] = -2, arr[2] = 4, arr[3] = 3, arr[4] = 11, arr[5] = -8`

Now you can sort arr with the call

```
Arrays.sort(arr);
```

The original array arr is then sorted.

If, however, the objects in an array are not naturally ordered, or if you want a different ordering from the usual one, then `Arrays.sort` expects you to pass in a `Comparator`. The signature of this version of sort when applied, for example, to `String`s is:

```
void sort(String[] arr, Comparator<String> comp)
```

For this exercise, create a class `StringSort` with a constructor

```
StringSort(Comparator<String> myComparator)
```

which sets the value of `myComparator` as an intstance variable in `StringSort`. `StringSort` also has a method

```
public String[] stringSort(String[] arr)
```

which makes use of the `Comparator` stored as an instance variable to sort the given input array using `Arrays.sort`; it then returns the array in its sorted order.

For this exercise, create a `Comparator` that is a static nested class of `StringSort` called `LengthCompare`, which uses the following logic to compare to `String`s: Suppose `s1` and `s2` are `String`s. Then `s1` is "less then" `s2` if length of `s1` is less then length of `s2`. With this logic, two `String`s will be considered "equal" if they have the same length. (Better ways of doing this kind of thing will be discussed in Lesson 8.)

Create a `main` method in a separate class `Main` which creates an array of test `String`s (any `String`s you like), instantiates `StringSort`, passing in an instance of the static nested class from `StringSort`, invokes the `stringSort` method using the test Strings as input, and then prints the result to console.

Finally, create a class `MainLambda` whose `main` method also creates a list of test `String`s, and then instantiates `StringSort` by passing in a lambda expression that realizes the `Comparator` interface. Define your lambda expression so that it enforces the following ordering on `String`s: Declare that `s1` is "less than" `s2` if `s1.compareTo(s2)` is a *positive* integer (this is the usual alphabetical ordering in reverse). Using this lambda-defined `Comparator` to instantiate `StringSort`, then, as before, invokes the `stringSort` method on the test `String`s defined earlier in `main`, and prints the output to console.