Sorting algorithms

In this lab you will gain practice sorting `Strings` as well as a user created class called `Person`.

Part 1: Selection Sort

As a reminder, the pseudocode for the selection for algorithm is as follows

```
for i = 0..n
    min = findSmallestElementIndex(arr, i)
    swap(arr, i, min)
end
```

Where `findSmallestElementIndex(arr, i)` is a function which will find the smallest element in the unsorted portion of our array (beginning at index `i`), and `swap(arr, i, min)`, which is a function that swaps indicies `i` and `min` in the array `arr`.

For your first task, implement Selection sort for an array of `Strings`. When we want to compare Strings, we should use the `compareTo()` method.

From the Java documentation

`public in compareTo(String anotherString)`

> Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this String object is compared lexicographically to the character sequence represented by the argument String. The result is a negative integer if this String object lexicographically precedes the argument string. The result is positive if this String object lexicographically follows the argument string. The result is zero if the strings are equal; compreTo returns 0 exactly when the equals(Object) method would return true.

Part 2: Arrays.sort()

When programming, you will not need to reinvent the wheel whenever we need to sort data. Majority of the time you need to sort something, you will use the `Arrays.sort()` method. This method takes in two arguments 1. The array we want sorted 2. How we want to order the array

The *how* we want the array ordered by is achieve through the implementation of the `Comparator` interface. You have been given an example of one such implementation which orders strings alphabetically when the String is reversed. Write two other implementations of the `Comparator` interface which will allow us to order the Strings by length of the word, and by non-case sensitive alphabetical.

For Extra Credit

Implement a third `Comparator` with a custom ordering. For instance, you could order the Strings alphabetically by the last letter, or order the strings alphabetically by the middle letter.

Part 3: Sorting Classes

More often than not you will be sorting Obejcts rather than primitive data types. To make this possible, we can use our class to implement the `Comparable` interface, which also has one method, `compareTo`.

Override the `compareTo` method in the provided `Person` class so it will order Person elements by their last name. If there are people with the same last names, then they should be ordered by their first name, if any Person objects share a last and first name, then their order should be determined by the year they were born.

The `compareTo` method works with classes, not primitive data types, so if we want to compare primitive data types, then we first need to cast them into their proper class.

```java
int age1 = 29;
int age2 = 35;
Integer new_age1 = (Integer)age1;
Integer new_age2 = (Integer)age2;

new_age1.compareTo(new_age2);
```