Seneca College of Applied Arts & Technology **October 29, 2019**

SCHOOL OF INFORMATION & COMMUNICATION TECHNOLOGY

JAC444 **Due dates: Labs 9 and 10**

Workshops 7 & 8

Description:
This assignment lets you practice functional programming as the new paradigm of programming added to Java 8. It includes concepts such as Generics, Functional Interfaces, Lambda Expressions, Method References, Streams, and Collections.

Functional Programming allows you to write programs more concise, and in many cases (like working with collections), faster and with fewer bugs. Mostly, in Functional Programming, you specify what you want to do (not how you want it to be done, as it's the case in non-functional programming paradigms.) Therefore, in this assignment, you should do all the tasks without using the traditional control statements (`for`, `while`, `do/while`, `if/else`, `switch/case` and even `recursion`) and just by using Functional Programming facilities provided in Java 8. (*the only exception is inside the equals, constructor, and setter methods of class Student (discussed below); which you could use the* `if/else` *control structure!*)

Two basic elements of Functional Programming are Lambda Expressions (which are anonymous methods) and Method References, that have had them in the course (Week 7). Java 8 also introduces Streams (hint: don't mix them up with IO Streams! They are new beasts!) Lambda Expressions and Streams let you do variety of tasks on the elements of a collection with great ease.

In this assignment, you should first define a class **Student** which has four fields in this order: `firstName` as a String, `lastName` as a String, `grade` as a double, and `department` as a String. Provide one constructor for the class (which takes all the fields), setter and getter methods for all fields, a `getName` method which returns the full name of the student (*ex. "John White"*), `toString`, and `equals` methods.

We assume that there has been a contest among students of different departments and the results have been gathered as grades. Therefore, in a second class, **StudentProcess**, you are supposed to use functional programming to do various tasks on a collection of **Students**.

You might want to have a look at the following classes/interfaces, as you will need them while doing the assignment:

- `java.util.Arrays;`
- `java.util.Comparator;`
- `java.util.List;`
- `java.util.Map;`
- `java.util.TreeMap;`
- `java.util.function.Consumer;`
- `java.util.function.BiConsumer;`
- `java.util.function.Function;`
- `java.util.function.Predicate;`
- `java.util.stream.Stream;`

- `java.util.stream.Collectors;`
- `java.util.Optional;`

Task 1: Create an array of Students in the beginning of your implementations, populate it with some Students, make a list out of your array, and print all its elements. You could create your own Student objects, hard-coded into your program and I will test your code run, against my input (use arbitrary values for first names, last names, grades – between 0.0 and 100.0 – and departments.) There is no need to read data from the file in this assignment (*hint: have a look at List<E> class and don't forget to use method references to do this task and this assignment.*)

Task 2: Display Students with grades in the range 50.0-100.0, sorted into ascending order by grade. (*hint: you need to return a Stream<Student> out of your List<Student> first, and then use Stream and Comparator classes' methods classes' methods.*)

Task 3: Display the first student in the collection with grade in the range 50.0-100.0 (*hint: you need to return a Stream<Student> out of your List<Student> first, and then use Stream and Optional classes' methods.*)

Task 4: Sort the Students (a)by their last names, and then their first names in ascending and (b)by their last names, and then their first names in descending orders and display the students after each of these two processes. (*hint: you need to return a Stream<Student> out of your List<Student> first, and then use Stream and Comparator classes' methods.*)

Task 5: Display unique Student last names, sorted. (*hint: you need to return a Stream<Student> out of your List<Student> first, and map it to a Stream<String>, and use its methods.*)

Task 6: Display Student full names, sorted in order by last name then first name. (*hint: you need to return a Stream<Student> out of your List<Student> first, use Stream class's methods, and map it to a Stream<String> somewhere along the way.*)

Task 7: Display Students, grouped by their departments. (*hint: you need to have an object of Map<String, List<Student>> and first populate it using Stream class's methods, and second, display the desired output using Map class's methods. You should also use Collectors for this task.*)

Task 8: Count and display the number of Students in each department. (*hint: you need to have an object of Map<String, Long> and first populate it using Stream class's methods, and second, display the desired output using Map class's methods. You should also use Collectors for this task.*)

Task 9: Calculate and display the sum of all Students' grades. (*hint: you need to return a Stream<Student> out of your List<Student> first, and then use Stream class's methods to map it to a DoubleStream, and then, use DoubleStream methods to do the task.)*

Task 10: Calculate and display the average of all Students' grades. (*hint: you need to return a Stream<Student> out of your List<Student> first, and then use Stream class's methods to map it to a DoubleStream, and then, use DoubleStream methods to do the task.)*

Typical Output:
For a typical input such as:

```
 Student[] students = {
         new Student("Jack", "Smith", 50.0, "IT"),
         new Student("Aaron", "Johnson", 76.0, "IT"),
         new Student("Maaria", "White", 35.8, "Business"),
         new Student("John", "White", 47.0, "Media"),
         new Student("Laney", "White", 62.0, "IT"),
         new Student("Jack", "Jones", 32.9, "Business"),
         new Student("Wesley", "Jones", 42.89, "Media")};
```

The output could be:

```
Task 1:

Complete Student list:
Jack     Smith       50.00   IT
Aaron    Johnson     76.00   IT
Maaria   White       35.80   Business
John     White       47.00   Media
Laney    White       62.00   IT
Jack     Jones       32.90   Business
Wesley   Jones       42.89   Media


Task 2:

Students who got 50.0-100.0 sorted by grade:
Jack     Smith       50.00   IT
Laney    White       62.00   IT
Aaron    Johnson     76.00   IT


Task 3:

First Student who got 50.0-100.0:
Jack     Smith       50.00   IT


Task 4:

Students in ascending order by last name then first:
Aaron    Johnson     76.00   IT
Jack     Jones       32.90   Business
Wesley   Jones       42.89   Media
Jack     Smith       50.00   IT
John     White       47.00   Media
Laney    White       62.00   IT
Maaria   White       35.80   Business
```

```
Students in descending order by last name then first:
Maaria   White       35.80   Business
Laney    White       62.00   IT
John     White       47.00   Media
Jack     Smith       50.00   IT
Wesley   Jones       42.89   Media
Jack     Jones       32.90   Business
Aaron    Johnson     76.00   IT


Task 5:

Unique Student last names:
Johnson
Jones
Smith
White


Task 6:

Student names in order by last name then first name:
Aaron Johnson
Jack Jones
Wesley Jones
Jack Smith
John White
Laney White
Maaria White


Task 7:

Students by department:
Media
   John     White       47.00   Media
   Wesley   Jones       42.89   Media
IT
   Jack     Smith       50.00   IT
   Aaron    Johnson     76.00   IT
   Laney    White       62.00   IT
Business
   Maaria   White       35.80   Business
   Jack     Jones       32.90   Business


Task 8:

Count of Students by department:
Business has 2 Student(s)
IT has 3 Student(s)
Media has 2 Student(s)


Task 9:

Sum of Students' grades: 346.59
```

```
Task 10:
Average of Students' grades: 49.51
```

Marking criteria:
Please note that you should:

a- have appropriate indentation.
b- have proper file structures and modularization.
c- follow Java naming conventions.
d- document all the classes properly.
e- not have debug/useless code and/or file(s) left in assignment.
f- have good intra and/or inter class designs.

in your code!

1. Each of the tasks 1-10: **1 mark**.

Deliverables and Important Notes:
You are supposed to show up AND hand in your solution in person (run the solution and/or answer related Qs) in labs 9 (the first 5 tasks) & 10 (the rest).

In case you don't show up OR hand in/run the required task in the lab, you could submit your final solution (described below) on the same due date but note that there would be a 50% penalty! Late submissions would result in additional 10% penalties for each day or part of it.

In this case, you should zip *only the Java files* to a file named after your Last Name followed by the first 3 digits of your student ID. For example, if your last name is **`Savage`** and your ID is **`354874345`** then the file should be named **`Savage354.zip`**. Finally email your zip file to me at reza.khojasteh@senecacollege.ca

Remember that you are encouraged to talk to each other, to the instructor, or to anyone else about any of the assignments, but the final solution may not be copied from any source.