Objectives

- Text Processing Exercise with Java Streams and Lambdas

Stream of String


```
public class App {

    public static void main(String[] args) {
        Stream<String> s = getTestLinesStream();

        s.forEach(System.out::println);
        s.close();
    }

    public static Stream<String> getTestLinesStream() {
        File file = new File("lambtest.txt");

        try {
            return Files.lines(file.toPath());
        } catch (Exception e) {
            System.out.println("Error reading from file");
            return null;
        }
    }
}
```

- The class App shows how to create a stream from a text file
  - Notice the *method references* and the *for each* construction

Lambda Expressions

- One of the major functional ideas introduced in Java 8 are lambda expressions or simply lambdas
- Lambdas are indicated by the syntax -> (minus sign followed by a greater than sign) which is meant to express the idea of a function or mapping
- Lambdas are like anonymous functions, they don't have a name but they have parameters and a return type

```
(String s) -> System.out.println(s)
```

Streams Example

- Suppose you want to chain together the two operations of applying a conditional to the objects in a collection and then returning a new collection with only the `true` results
- Notice here that `filter` is part of the streams API and

```
List<String> users = Arrays.asList("alice","bob","Charlie");
filteredUsers = users.stream()
                     .filter((String s) -> s.length() > 3)
                     .collect(Collectors.toList());
```

Map and Reduce


```
int n = (int) toks.stream().map((String x) -> x.toUpperCase())
                  .filter((String x) -> x.contains("O"))
                  .mapToInt((String x) -> x.length())
                  .reduce(0,(a,b) -> a+b);
```

- We can also perform a reduction using a binary function

Grouping and Flat Map

- The `.collect()` method and `java.util.stream.Collectors` allow us to convert a stream into a "regular" collection type (a `List` for example)
  - `Collectors.groupingBy(<lambda expression>)` is a type of collector that gathers elements based on a *key* function received as a lambda
- `Stream.of(<array>)` takes a regular Java array and converts it into a `Stream`
- `.flatMap(<lambda>)` is useful for creating a single stream when you map a function that can result in a stream (converts a stream of streams to a single "flat" stream)

Cloning the Sample Code

- We have set up a public repository containing a basic Java web application[1]
- You can clone this project and import to Eclipse as you have done previously
- Make sure you are in the Java SE perspective (upper right corner)

[1] `https://github.com/marks1024/exercise-java-streams-361`

Task

- Clone the sample application from Github
- Operate on the stream returned by `getTestLinesStream()` method
- Do the following using *only* stream operations and lambda expressions
  - Perform a word count on the entire stream
  - Print only those lines that contain a word of length greater than 7 characters (us filter)
  - Create a map data structure that contains how many words in the stream begin with each possible letter (use groupingBy)
- Submit an archive of your code to the moodle

Task 1

- Clone the sample application from Github
- Call the method `getTestLinesStream()` to obtain a `Stream<String>`
- Complete all tasks using *only* stream operations and methods, lambda expressions, and method references
- The first task is to perform a word count on the entire file and print the result
- *Hint*: This can be accomplished with a combination of `flatMap` and the terminal operation `count`
  - Also recall that `Stream.of` creates a stream from a list of values or something array-like

Task 2

- The second task is to print only the lines that contain a word of length greater than 7 characters
- For this task you should use a *filter*
- *Hint*: An approach would be have nested chains of stream operations: one to find the longest word in a line and another to actually filter the lines
- *Hint*: Think about how you could create a new stream using `map` whose objects are "pairs" of the orginal text of the line and a number equal to the longest word on the corresponding line
  - For example, a map that sends "The cat meowed" to ("The cat meowed", 6)

Task 3

- The third task is to create a map data structure that contains how many words in the stream begin with each possible letter
- To complete this task you need a combination of `flatMap`, `collect`, and `groupingBy`