LAB-9-HOMEWORK

First Half : 4,5,6,10

Second Half : 1,2,3,7,8,9

1. You have a `Stream` of `Strings` called `stringStream` consisting of the values “Bill”, “Thomas”, and “Mary”. Write the one line of code necessary to print this stream to the console so that the output looks like this:

   Bill, Thomas, Mary

2. You have a `Stream` of `Integers` called `myIntStream` and you need to output both the maximum and minimum values. Write compact code that efficiently accomplishes this.

3. Implement a method with the following signature and return type: -

```
public int countWords(List<String> words, char c, char d, int len)
```

which counts the number of words in the input list `words` that have length equal to `len`, that contain the character `c`, and that do not contain the character `d`. Create a Good and Better solution, as described in the slides (see package lesson8.lecture.filter) – a Good solution creates a lambda expression each time values are passed into countWords, whereas a Better solution has parametrized lambda expressions pre-made, and so a call to countWords simply substitutes values into these expressions. Try also creating a Best solution in which there is just one lambda expression.

4. Implement a method

```
public static void printSquares(int num)
```

which creates an `IntStream` using the `iterate` method. The method prints to the console the first `num` squares. For instance, if `num` = 4, then your method would output 1, 4, 9, 16. Note: You will need to come up with a function to be used in the second argument of `iterate`.

5. Create a method

`Stream<String> streamSection(Stream<String> stream, int m, int n)` which extracts a substream from the input stream `stream` consisting of all elements from position `m` to position `n`, inclusive; you must use only `Stream` operations to do this. You can assume `0 <= m <= n`.

```
public class Section {
    public static Stream<String> streamSection(Stream<String> stream, int m, int n) {

// Implement the code
    }
    public static void main(String[] args) {
   // Make three calls for the streamSection() method with different inputs
   // use nextStream() method to supply the Stream input as a first argument in streamSection() method
    }
```

```
        //support method for the main method -- for testing
        private static Stream<String> nextStream() {
                return Arrays.asList("aaa", "bbb", "ccc", "ddd", "eee", "fff", "ggg", "hhh",
"iii").stream();
        }
    }
```

6. Implement a method

```
public Set<String> union(List<Set<String>> sets)
```

by creating a stream pipeline that transforms a list of sets (of type String) into the union of those sets. Make use of the `reduce` method for streams.

Example: The union method should transform the list [{“A”, “B”}, {“D”}, {“1”, “3”, “5”}] to the set {“A”, “B”, “D”, “1”, “3”, “5”}.

7. In prob7a folder, there is an `Employee` class and a `Main` class, which has a `main` method that loads up a Stream of `Employee` instances.
   a. In the final line of the `main` method, write a stream pipeline (using filters and maps) which prints, *in sorted order (comma-separated, on a single line),* the full names (first name + “ “ + last name) of all `Employees` in the list whose salary is greater than $100,000 and whose last name begins with any of the letters in the alphabet *past* the letter ‘M’ (so, any letters in the range ‘N’--‘Z’). Filter the lastname which is after M in the alphabets, that is filtered names should be in the range of N—Z.

      For the main method provided in your lab folder, expected output is:

      ```
      Alice Richards, Joe Stevens, John Sims, Steven Walters
      ```

   b. Turn your lambda/stream pipeline from part (a) into a Lambda Library element, following the steps in the slides. First, create a class LambdaLibrary; this class will contain only public static final lambda expressions. Then, identify the parameters that need to be passed in so that your lambda/stream pipeline can operate properly. Finally, think of a function-style interface (Function, BiFunction, TriFunction, etc) that can be used to accommodate your parameters and then name your pipeline, with the function-type interface as its type (as in the slide example). Call your Library element in the main method instead of creating the pipeline there, as you did in part (a).

8. In prob8 folder, you have three classes such as Trader, Transaction and PuttingIntoPractice. In PuttingIntoPractice class have 7 unimplemented query parts. Do that by using the concepts you learned from this lesson.

9. In prob9 folder you have a one class called Dish and it has a List menu. In that class implement some static methods to decide the following with help of Optional, anyMatch(), allMatch(), noneMatch(),findAny(), findFirst() operations from stream.

a. Is there any Vegetarian meal available (return type boolean)

b. Is there any healthy menu have calories less than 1000 (return type boolean)

c. Is there any unhealthy menu have calories greater than 1000 (return type boolean)

d. find and return the first item for the type of MEAT(return type Optional<Dish>)

e. calculateTotalCalories() in the menu using reduce. (return int)

f. calculateTotalCaloriesMethodReference()in the menu using MethodReferences. (return int)

e. Implement a main method to test.

10. In prob10 folder there is ConstructorReference.java file. Three queries are available in the main method. Implement those.