Exercise 11 – Functional Programming in Java

Objective

By the end of this exercise you should be more comfortable with what Lambda expressions are and how to use them in Java. Part Two will look at how to use virtual extension methods and how to provide default implementations of methods in interfaces.

With all these exercises it will be possible to find a non-functional way of achieving the same result. If you are struggling with the functional method then try writing the standard object oriented approach first and then change it around. The aim of this exercise is to use functional programming concepts, so while we can do achieve the exercises in other ways, it would be somewhat missing the point.

Overview

Lambda Expressions

1. Use a range method to create a list of the years from 1960 to present
2. Filter this list based on whether each year is a leap year or not

   The rules for determining if something is a leap year are:

   i. The year is divisible exactly by four AND

   ii. the year is not exactly divisible by 100 OR

   iii. the year is exactly divisible by 400

   Hint: Use the modulo operation %
3. Print out the contents of the list. Remember you can chain these operations together
4. Create a simple person object
   a. A person should have a String name, an int age and a double salary
   b. Implement the constructor, getter, setter and toString methods for this class
   c. Implement the Comparison interface and write a compareTo method for the class which compares people by their name
5. Create an ArrayList of several person objects

6. Chain together the following operations on the person list:
    a. Filter the person list for people over the age of 30 who have a salary of less than 20,000.
    b. Use map to increase the filtered people's salary. This will create a new collection of person objects, rather than change the current collection so you will need to create a new Person and return that object.
    c. Sort the results using sorted()
    d. Print out the results of these operations.
    e. What happens if you remove the sorted method()? What happens if you try and change the collection after the print statement has run?
    f. Comment out your print statement and use collect to create a new list of Person objects
7. Print out the average salary of all the people in your original list that are under 30 and print out the average for all the people over 30.

    Hint: Filter and collect will be useful here!

8. Write a method that takes an ArrayList of Integers and two funtions that applies those functions one after another to that list. The method should be called with the following code:

```
ArrayList<Integer> intList = new ArrayList<Integer>(
                     IntStream.range(0, 10)
                     .boxed()
                     .collect(Collectors.toList())
                     );
ArrayList<Integer> newIntList = applyFunctions(intList,
                     (Integer i) -> i = i + 1,
                     (Integer i) -> i = i * 2
                     );
newIntList.forEach(System.out::println);
```

The output of this code should be:

```
2
4
6
8
10
12
14
16
18
20
```

Virtual Extension Methods

In this exercise we're going to create some log writer interfaces. The basic interface has no default implementation. The console and time writer interfaces extend the basic interface and provide a Log(String msg) method which will output the message to System.out.println() on its own, or with the current time at the start.

9. Create a Logger interface, it should have one method, log which takes a String message.
10. Create a ConsoleLogger interface which extends Logger and provides a default implementation of the Log method and writes to the output window.
11. Create a TimeWriter interface which extends Logger and provides a different default implementation of the Log method, printing out the current time before the message (you will need to look up how to get the current date and time in Java. Java 8 provides a new library for doing this!)
12. Create an object which implements the Console Logger and print out a message.
13. Create a second object which implements the TimeWriter interface and print out a message.
14. Now implement both the Console and TimeWriter interfaces, what happens?
15. Force your class to use the TimeWriter implementation of the interface.

If you have time add a new "File Writer" which writes the output of the message to a file. You will need to handle opening and closing the file in the interface.

Glossary of key terms

Lambda Expressions

Lambda expressions are the same concept as anonymous functions

ArrayList

It is one of the implementation classes of **List** interface and it is not synchronized

Streams

An abstract representation of data flow, functional methods are available to streams which are not available to the standard collection classes