Technical University of Cluj-Napoca
Faculty of Automation and Computer Science
Department of Computer Science

2016/2017

Programming Techniques

Homework 5

Stream Processing using Lambda Expressions

Lecturer: Prof. Ioan Salomie
Teaching Assistant: Claudia Pop

Student: Acu D. Raul-Mihai
Group: 30425

Table of Contents

1. Problem specification

A smart house features a set of sensors that may be used to record the behavior of a person living in the house. The historical log of the person's activity is stored as tuples (startTime, endTime, activityLabel), where startTime and endTime represent the date and time when each activity has started and ended while the activity label represents the type of activity performed by the person: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare_Time/TV, Grooming.

a. System`s requirements

Define a class MonitoredData having startTime, endTime and activityLabel as instance variables and read the input file data into the data structure monitoredData of type List. Using stream processing techniques and lambda expressions introduced by Java 8, write the following set of short programs for processing the monitoredData.

1. Count the distinct days that appear in the monitoring data.
2. Determine a map of type that maps to each distinct action type the number of occurrences in the log. Write the resulting map into a text file.
3. Generates a data structure of type Map> that contains the activity count for each day of the log (task number 2 applied for each day of the log) and writes the result in a text file.
4. Determine a data structure of the form Map that maps for each activity the total duration computed over the monitoring period. Filter the activities with total duration larger than 10 hours. Write the result in a text file.
5. Filter the activities that have 90% of the monitoring samples with duration less than 5 minutes, collect the results in a List containing only the distinct activity names and write the result in a text file.

2. Usages & Design

Getting Started with Streams

Let's start with a bit of theory. What's the definition of a stream? A short definition is "a sequence of elements from a source that supports aggregate operations." Let's break it down:

- **Sequence of elements:** A stream provides an interface to a sequenced set of values of a specific element type. However, streams don't actually store elements; they are computed on demand.
- **Source:** Streams consume from a data-providing source such as collections, arrays, or I/O resources.
- **Aggregate operations:** Streams support SQL-like operations and common operations from functional programing languages, such as filter, map, reduce, find, match, sorted, and so on.

Furthermore, stream operations have two fundamental characteristics that make them very different from collection operations:

- **Pipelining:** Many stream operations return a stream themselves. This allows operations to be chained to form a larger pipeline. This enables certain optimizations, such as *laziness* and *short-circuiting*, which we explore later.
- **Internal iteration:** In contrast to collections, which are iterated explicitly (*external iteration*), stream operations do the iteration behind the scenes for you.

Streams Versus Collections

Both the existing Java notion of collections and the new notion of streams provide interfaces to a sequence of elements. So what's the difference? In a nutshell, collections are about data and streams are about computations.

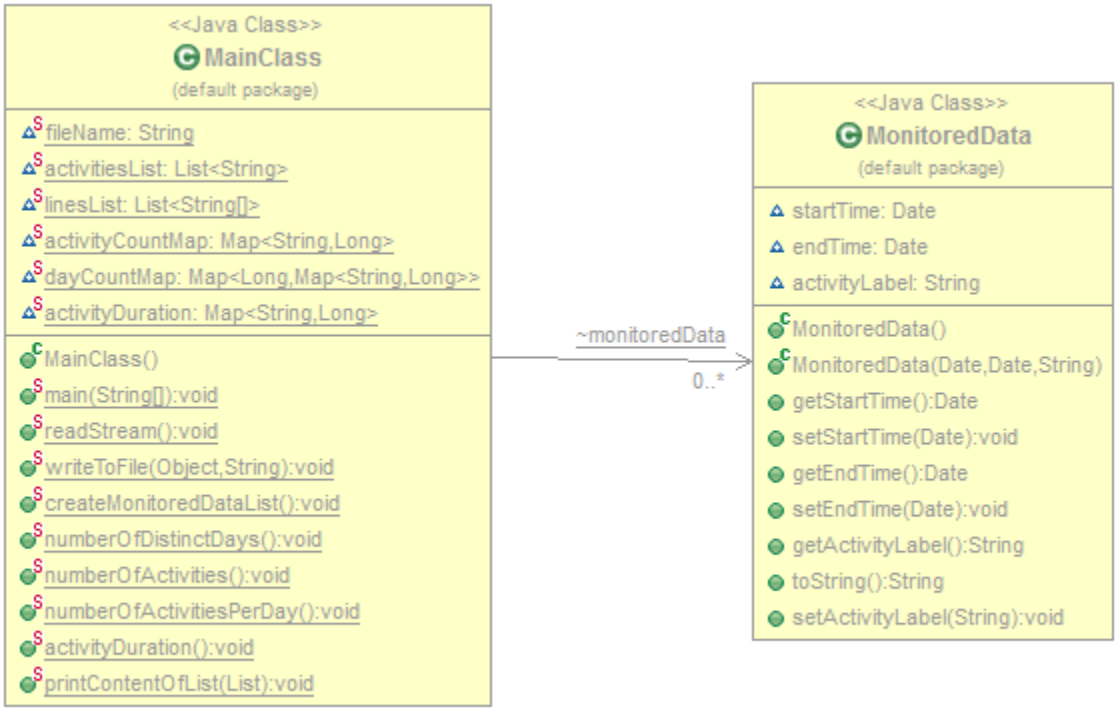
Consider a movie stored on a DVD. This is a collection (perhaps of bytes or perhaps of frames—we don't care which here) because it contains the whole data structure. Now consider watching the same video when it is being streamed over the internet. It is now a stream (of bytes or frames). The streaming video player needs to have downloaded only a few frames in advance of where the user is watching, so you can start displaying values from the beginning of the stream before most of the values in the stream have even been computed (consider streaming a live football game).

In the coarsest terms, the difference between collections and streams has to do with *when* things are computed. A collection is an in-memory data structure, which holds all the values that the data structure currently has—every element in the collection has to be computed before it can be added to the collection. In contrast, a stream is a conceptually fixed data structure in which elements are computed on demand.
Using the Collection interface requires iteration to be done by the user (for example, using the enhanced for loop called foreach); this is called external iteration.

Building Streams

There are several ways to build streams. You've seen how you can get a stream from a collection. Moreover, we played with streams of numbers. You can also create streams from values, an array, or a file. In addition, you can even generate a stream from a function to produce infinite streams!

UML Class Diagram



The Unified Modeling Language (UML) is a general-purpose,developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.

Structure diagrams emphasize the things that must be present in the system being modeled. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems

3. Conclusion

a. Advantages and Disadvantages

- **Advantages**
  - Easy to use;
  - Plain and simple, nothing to be confused by.
- **Disadvantages**
  - One client cannot order more than a product at a time.
  - It might fail if given wrong inputs;

b. Future development: ways of improving the system

- Desigining a better relational database;
- Trying to use as less memory as possible;
- Trying to improve the response time;
- Designing a more user-interactive interface;