Software Architectures

Assignment 1: Akka Streams

Camilo Velazquez, Ahmed Zerouali
Emails: {cavelazq, azeroual}@vub.be
Office: {10F725}

For this assignment, you need to implement a Pipe and Filter architecture using Akka Streams.

Assignment

You will have to implement and report on your original solution to a problem using Akka Streams. You will find the input file needed for this assignment in *Assignments > Assignment 1.*

Deadline: January 10^{th} 2021 by 23:59. The deadline is fixed and will not be extended for any reason.

Deliverables A report and source code. The report (in English) explains your solution to the problem and the main components used in your implementation. When possible, use simple diagrams to illustrate.

The report should be handed in as a single PDF file of no more than 3 pages (excluding diagrams/screenshots/code). The file should follow the naming schema `FirstName-LastName-SA1.pdf`, for example: `Camilo-Velazquez-SA1.pdf`.

The source code is your implementation of the project zipped or exported from IntelliJ.
Submit the *report* and *source code* as a single zip file on the Software Architectures course page in Canvas, by clicking on *Assignments > Assignment 1.*

Plagiarism Note that copying – whether from previous years, from other students, or from the internet – will not be tolerated, and will be reported to the dean as plagiarism, who will decide appropriate disciplinary action (e.g., expulsion or exclusion from the examination session). If you use any other resources besides those provided in the lectures and in this document, remember to cite them in your report.

Grading Your solution will be graded and can become subject of an additional defense upon your or the assistants' request.

Problem Description

In this assignment you will implement a Pipe and Filter architecture for a system that processes library dependencies of the Maven software ecosystem[1], i.e., given a dataset of Maven software library dependencies, we want to know how many dependencies each library has.

You are provided with a **zipped** file named *maven_dependencies.zip* containing the data you need in this Assignment. Within the aforementioned **zip** you can find a text file containing 200,000 records of information related to the dependencies of libraries in Maven. Each of the records contains three attributes separated by a comma (`library, dependency, type`): 1) `library` refers to the name of the library, 2) `dependency` refers to the dependency that the library is depending on, and 3) `type` refers to the type of the dependency.

More specifically, a library name consists of three components separated by a colon (:) `GroupID:ArtifactID:Version`. The second attribute `dependency` refers to the dependencies of the corresponding library and it also consists of three components separated by a colon (:) `GroupID:ArtifactID:Version`. The dependency `type`s (i.e., the third attribute) are distributed among four different kinds: `Compile`, `Provided`, `Runtime` and `Test`. Each row in the file contains exactly one input referring to a library dependency and its type. A library may have a set of dependencies, thus each library may have several rows of dependencies in the file. You should create the necessary abstractions (i.e., classes) in order to map each row in the file to an object in Scala.

Figure 1 provides a graphical representation of the application you will have to implement for this work. Once the file is extracted and the rows have been converted to objects of a user-defined class, you should create a **Source** from it. In a follow up step the Source streaming should be grouped by library name in order to be able to process a set of dependencies of the same library. The maximum number of sub streams from the grouping should be the largest integer value (i.e., `Int.MAX`) or the total amount of records. This will allow you to have enough sub streams to stream all groups of dependencies.

A custom **Flow** shape (i.e., the Flow Dependencies box in Figure 1) should be created in order to count the number of dependencies in a parallel way. The shape should have a **balanced** approach to send groups of dependencies of the same library to the two pipelines. The pipelines should work in the same way by counting the number of dependencies by type (i.e., filters in Figure 1). You should create necessary classes and objects to be able to compute and store the number of dependencies. The two pipelines should return the name (common to all elements in a group) of the library and the number of dependencies per type. Both of these pipelines will be **merged** in a single outlet as the output of the custom shape.

Finally, the **output** of the processing should be saved into an external file for future analysis. The file should contain one line per library adhering to the following format: *NameOfLibrary –> Compile: # Provided: # Runtime: # Test: #*. Where the hashtags

[1] https://mvnrepository.com

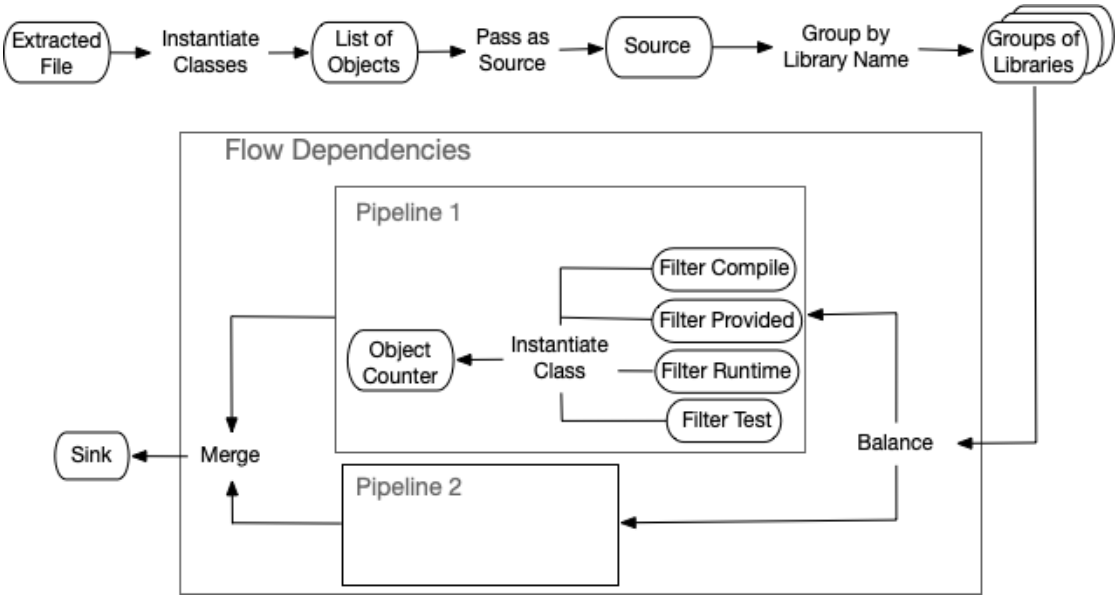

Figure 1: Application diagram

(#) are replaced by the actual number of dependencies per type.

The screenshot in Figure 2 provides a real example of the expected output for the described process.

```
br.com.objectos:assertion:0.1.0 --> Compile: 9 Provided: 0 Runtime: 0 Test: 4
br.com.objectos:assertion:0.1.1 --> Compile: 9 Provided: 0 Runtime: 0 Test: 0
ai.grakn:grakn-test-profiles:1.1.0 --> Compile: 1 Provided: 0 Runtime: 0 Test: 0
br.com.objectos:assertion:0.1.2 --> Compile: 9 Provided: 0 Runtime: 0 Test: 0
```

Figure 2: Screenshot of a fragment from the expected output.

A line consists of the name of a library and the number of dependencies per type for that library.

The above requirements should be implemented using Akka Streams. You are also required to make use of the `GraphDSL`. **Motivate your design decisions** in the report as per the problem description and illustrate your approach using concepts from Akka Streams.

More information on Akka Streams

- `https://doc.akka.io/docs/akka/2.5.32/stream/index.html`
- `https://doc.akka.io/docs/akka/2.5.32/stream/stream-graphs.html`