Software Architectures

Assignment 1: Akka Streams

TA: Camilo Velázquez-Rodríguez
camilo.ernesto.velazquez.rodriguez@vub.be
F.10.725

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 7th 2024 by 23:59. The deadline is fixed and will not be extended for any reason. Submissions after the deadline will not be considered for grading and the grade will automatically be **ABS**.

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

The report should be handed in as a single PDF file.
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 in a single file. Do not include files or folders resulting from the compilation process of your project, e.g., target/ or *.class. 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 Any suspicion of plagiarism will be reported to the Dean of Faculty without delay. Both user and provider of such code will be reported and will be dealt with according to the plagiarism rules of the examination regulations (cf. OER, Article 118). The dean may decide on (a combination of) the disciplinary sanctions, ranging from 0/20 on the paper of the given programme unit to a prohibition from (re-)enrolling for one or multiple academic years (cf. OER, Article 118$5).

Grading Your solution will be graded and can become the subject of an additional defence upon the lecturer's request.

Problem Description

In this assignment you will implement a Pipe and Filter architecture for a system that processes library dependencies of the Maven repository[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 **CSV** file named *maven_dependencies.csv* containing the data you need to process in this Assignment. Within the mentioned file are 8,092 records of information related to the dependencies of libraries in Maven. Each record contains three attributes separated by a comma (`library, dependency, type`): 1) `library` refers to the name of a library, 2) `dependency` refers to the dependency of the former library, and 3) `type` refers to the type of the dependency.

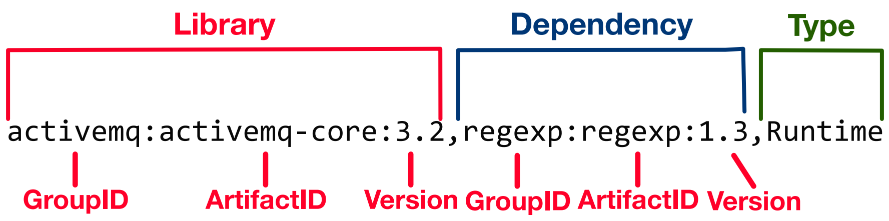

Figure 1: Description of a record in the CSV file.

Figure 1 shows all attributes in one CSV record. A `library` name consists of three components separated by a colon (:) `GroupID`, `ArtifactID` and `Version`. The second attribute `dependency` refers to the dependencies of the corresponding library. A dependency also has three components separated as previously described for the `library`. The dependency `types` (i.e., the third and final attribute) are distributed among two kinds: `Compile` and `Runtime`. Each row contains exactly one input referring to a library dependency and its type. A library may have a group of dependencies, thus, several rows will cover each of the dependencies for the library. You should create the necessary abstractions (i.e., classes, case classes, traits, etc.) to map each row in the file to an object in Scala.

Figure 2 provides a graphical representation of the application you will have to implement for this work. First, you can create the **Source** from a hard-coded file path. In a follow-up step the Source with the **CSV** file should be parsed and subsequently processed to obtain a stream of `Maps` with the file information. Each map element should be transformed into an object instantiation of a user-defined class.

The stream of objects should be grouped by library name to process the set of dependencies belonging to the same library. The maximum number of substreams from the grouping should be 185 as this is the number of unique libraries (i.e., considering `GroupID`, `ArtifactID` and `Version`) in the file. This will allow you to group all libraries with their respective set of dependencies. A **throttle** element will restrict the processing

[1]https://mvnrepository.com

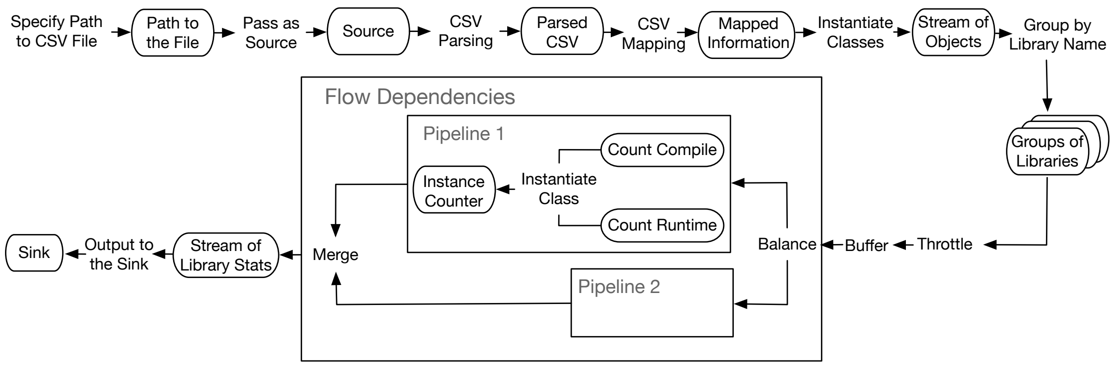

Figure 2: Application diagram.

of the groups at a rate of 10 groups per second. Later, a buffer will also restrict the stream with a capacity of 5 elements and a *backpressure* strategy in case of overflow.

A custom **Flow** shape (i.e., the Flow Dependencies box in Figure 2) should be created in order to count the number of dependency types in parallel. The shape should have a **balanced** approach to send groups of dependencies of the same library to two pipelines. The pipelines should work in the same way by counting the number of dependencies by type (e.g., `Compile` and `Runtime` types). You should create the necessary classes and objects to compute and store the number of dependencies. The two pipelines should output an instantiated object from which you can access the name of the library and the number of dependencies per type. Both pipelines will be **merged** in a single outlet as the output of the custom shape.

Finally, the libraries and the number of dependencies should be printed on the terminal. The screenshot in Figure 3 provides a real example of the expected output for the described process.

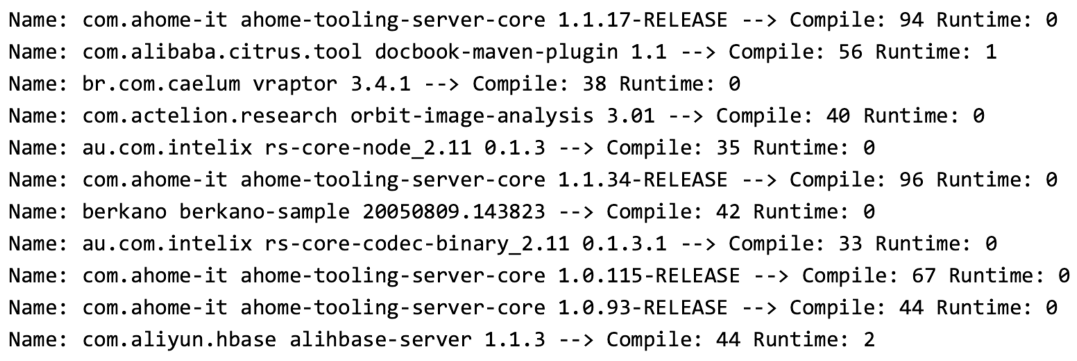
```
Name: com.ahome-it ahome-tooling-server-core 1.1.17-RELEASE --> Compile: 94 Runtime: 0
Name: com.alibaba.citrus.tool docbook-maven-plugin 1.1 --> Compile: 56 Runtime: 1
Name: br.com.caelum vraptor 3.4.1 --> Compile: 38 Runtime: 0
Name: com.actelion.research orbit-image-analysis 3.01 --> Compile: 40 Runtime: 0
Name: au.com.intelix rs-core-node_2.11 0.1.3 --> Compile: 35 Runtime: 0
Name: com.ahome-it ahome-tooling-server-core 1.1.34-RELEASE --> Compile: 96 Runtime: 0
Name: berkano berkano-sample 20050809.143823 --> Compile: 42 Runtime: 0
Name: au.com.intelix rs-core-codec-binary_2.11 0.1.3.1 --> Compile: 33 Runtime: 0
Name: com.ahome-it ahome-tooling-server-core 1.0.115-RELEASE --> Compile: 67 Runtime: 0
Name: com.ahome-it ahome-tooling-server-core 1.0.93-RELEASE --> Compile: 44 Runtime: 0
Name: com.aliyun.hbase alihbase-server 1.1.3 --> Compile: 44 Runtime: 2
```

Figure 3: 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 and Scala 3. 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 and external libraries

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