# Appendix B

# Java instrumentation agent (ClassMexer)

SQuO implementation is in charge to monitor the computational resources consumed during the execution of a query plan. For the validation of this project, was necessary to monitor specifically the execution time and memory consumption. ClassMexer is a very useful tool since it allows to take detailed measures about the memory that is used by each object.

ClassMexer is a simple Java instrumentation agent that provides some convenience calls for measuring the memory usage of Java objects from within an application. It relies on the Java instrumentation framework [Co081].

## Instrumentation: querying the memory usage of a Java object

The most reliable, but not necessarily the easiest, way to estimate the usage of a Java object is to ask the JVM. Querying the JVM for the memory usage of an object requires the Instrumentation framework. Hotspot supports the instrumentation framework; if you use a VM from a different vendor, it will need to provide similar support, and there may be some variation in the procedures described here [Co081] [Co082].

The general idea is to create a special agent class with a special pre-main method, generally compiling it against the rest of our code so that it can see the definitions of the classes whose memory usage we're interested in measuring;

- The JVM passes a special Instrumentation object to pre-main method, which can be used to query the size of an object;
- Then, is required to package the compiled agent class into a jar with a special manifest file;
- Finally, it is possible to run the program, passing in the agent jar to the VM command line arguments.

## Creating the *instrumentation agent* class

An instrumentation agent is a class with a special method, with a predefined signature, that the JVM will invoke before the rest of the application for you to set up any instrumentation code. Generally, instrumentation is the act of transforming classes for profiling purposes: for example, manipulate the definition of the String class to increment a counter every time a string is created, and thus measure e.g. how many strings per second our application creates. But an interesting additional feature provided by the instrumentation framework is the ability to measure the memory usage of an object. The JVM will pass to a method an implementation of the Instrumentation interface, defined in java.lang.instrument. In turn, this interface defines the method getObjectSize()[Co082].

## Package the agent into a jar

Once an agent class is completed, is needed to package it into a jar. For this, a manifest file must be created. The latter is simple a text file containing a single line that specifies the agent class. Then, to create the jar, the following command jar -cmf manifest.txt agent.jar mypackage/ .class must be exacuted.

## ClassMexer agent

The simplest methods provided by the MemoryUtil class returns the number of bytes occupied by an object, not included any objects it refers to. This method is essentially a wrapper around the JDK method Instrumentation.getObjectSize(). As such, it doesn't always give a very relevant result [Co081] [Co082].

Usually it is more interesting to query the "deep" memory usage of an object, which includes sub-objects (objects *referred* to by a given object). For example, if we try to query the memory usage of a string using memoryUsageOf(), we won't actually be including the characters of the string, because they're stored in a char array (a separate object) referenced by the String object itself . The deepMemoryUsageOf() calls are designed to get round this problem. They *recursively* include "subobjects" or objects referred to by the "main" object(s) passed in [Co081].

This tool also offer takes the measure not just from one object, by contrary, multiple objects. The deepMemoryUsageOfAll() methods take a *collection* of objects and add up the total memory usage of *all* objects in the supplied collection.