**Advanced Java Programming** **2019**

**Instructors:** Prof. Walter Binder and Dr. Andrea Rosà **TA:** Michael Weiss

# DiSL Project (40 points)

**Due date:** 17 December 2019, 08:00 a.m.

In this project, you are required to implement multiple profilers using DiSL. Please read *carefully* the instructions (including the submission instructions) and the exercises below. If something is unclear, please post a message on the forum on iCorsi as soon as possible.

## Instructions

This project is composed of 8 exercises. Each exercise is composed of an observed application (already provided) and a profiler (that you have to implement). In each exercise, both the observed application and the profiler are independent from the ones of the other exercises.

On iCorsi, you can download a template for this project. The profiler required for solving exercise *i must* be implemented by adding its source code in `src/exercisei` (note that this means that profiler classes should be contained in package `exercisei`). The profiler can be composed of how many classes you like. However, all DiSL snippets *must* be added to the class `Instrumentation.java` (already provided in the template). The source code of the observed applications cannot be changed.

All the profilers *must* be thread-safe. Non thread-safe implementation will receive 0 points.

Please refer to the instructions in the `README.md` file (in the template) for additional information about how to compile, build and run the observed application and the profiler for a given exercise.

## Submission Instructions

You must implement the profilers by modifying/adding classes as needed to the provided template. Your submission must consist of an archive containing the modified project template. You are not required to submit any other document. If needed, you can specify your notes as comments in the source code of the profiler. There is no naming requirement for the submission.

To be evaluated, the project must be submitted to iCorsi before the deadline specified in this document. Moreover, all code *must* compile.

## Note on Grading

To have the project graded, it is *mandatory* to attend the project review session (scheduled on December 19, 2019 during the usual lecture slot) and discuss your submission during such session. Submitting the project without discussing it *will not grant you any point*.

Since the project is part of the final exam, the grade will not be communicated to you on iCorsi. You will receive the final course grade from the Dean's Office after the final exam.

## Exercise 1 (6 points)

Implement a profiler that detects all *successful* object and array allocations occurred in the `run` method of all classes defined in package `exercise1.allocator`. The profiler must print the *total* number of allocations (objects + arrays) at the end of application execution.

**Note:** a correct implementation should *always* detect 5555000 allocations.

## Exercise 2 (6 points)

Implement a profiler that detects the total number of bytecodes executed in any method of class `exercise2.MainThread`. The profiler must print the *total* number of bytecodes executed at the end of application execution.

**Note:** a correct implementation should *always* detect 9000150 bytecodes executed.

## Exercise 3 (6 points)

Implement a profiler that detects all method invocations occurred in the `run` method of class `exercise3.MainThread`. The profiler must maintain a different counter for each of the different bytecode instruction that can be used to invoke a method. The values of such counters must be printed at the end of application execution.

**Note:** a correct implementation should *always* detect a total of 25000 method invocations. Each counter should be multiple of 1000.

## Exercise 4 (7 points)

Implement a profiler that detects all object allocations, all method invocations and all accesses to fields (i.e., either reading or writing them) done by any thread of class `exercise4.MainThread`. When the `run` method of class `exercise4.MainThread` terminates, the profiler should print the following line:

```
Thread: <a> - #invocations: <b>, #allocations: <c>, #fields accesses: <d>.
```

Replace `<a>` with the name of the currently executing thread, `<b>` with the number of method invocations done *by such thread*, `<c>` with the number of object allocations done *by such thread*, `<d>` with the number of field accesses done *by such thread*.

**Remark 1:** unlike the previous exercises, the profiler must maintain *per-thread* counters, and not single, global counters for each event of interest.

**Remark 2:** the profiler must detect method invocations, object allocations and field accesses occurred *in any method of any class*, not only those occurred in class `exercise4.MainThread`.

**Note:** a correct implementation should print lines similar to the ones below (actual numbers printed and line order can vary):

```
Thread: Application Thread 0 - #invocations: 649, #allocations: 33, #fields accesses: 1095
Thread: Application Thread 1 - #invocations: 948, #allocations: 25, #fields accesses: 1732
Thread: Application Thread 2 - #invocations: 1355, #allocations: 34, #fields accesses: 2494
Thread: Application Thread 3 - #invocations: 1784, #allocations: 44, #fields accesses: 3282
Thread: Application Thread 4 - #invocations: 2223, #allocations: 54, #fields accesses: 4076
Thread: Application Thread 5 - #invocations: 2675, #allocations: 64, #fields accesses: 4875
```

```
Thread:  Application Thread 6 - #invocations:  5291, #allocations:  418, #fields accesses:  10923
Thread:  Application Thread 7 - #invocations:  3759, #allocations:  88, #fields accesses:  6759
Thread:  Application Thread 8 - #invocations:  4030, #allocations:  94, #fields accesses:  7274
Thread:  Application Thread 9 - #invocations:  4476, #allocations:  104, #fields accesses:  8055
```

## Exercise 5 (7 points)

Implement a profiler that detects all object constructors *successfully* executed by *any* thread. At the end of each constructor, the profiler should print, in a single line:

- the class whose constructor has just been executed
- the name of the currently executing thread
- the time taken by such thread to execute the constructor
- the number of constructors executed by such thread so far

## Exercise 6 (8 points)

Implement a profiler that detects the execution of each loop iteration. At the end of the application, for each method where at least one loop iteration was executed, the profiler must print the number of loop iterations executed inside that method.

**Note 1:** to clarify, each time the following method `m` is executed

```java
void m() {
  for (int i = 0; i < 100; i++) {
    ... //Implementation omitted
  }
}
```

the profiler should increment the counter associated to `m` by 100.

**Note 2:** remember to print the *fully qualified class name* of each method with a loop.

## Exercise 7 (3 bonus points)

Implement a profiler that detects all invocations of method `targetMethod` of class `exercise7.MainThread`. *After* each *successful* execution of such method, the profiler must print the following line:

```
Arguments:  <a> <b> <c> - Return value:  <d>
```

Replace `<a>` with the value of the first argument passed to `targetMethod`, `<b>` with the value of the second argument passed to `targetMethod`, `<c>` with the *hashcode* of the third argument passed to `targetMethod`, `<d>` with the value returned by `targetMethod`.

**Important:** the profiler **must** instrument only `targetMethod` inside class `exercise7.MainThread`.

**Note:** a correct implementation should print lines similar to the ones below (line order can vary):

```
Arguments:  43 27 1000 - Return value:  2161
Arguments:  44 28 1000 - Return value:  2232
Arguments:  45 29 1000 - Return value:  2305
Arguments:  46 30 1000 - Return value:  2380
```

```
Arguments:  47 31 1000 - Return value:  2457
Arguments:  48 32 1000 - Return value:  2536
Arguments:  49 33 1000 - Return value:  2617
Arguments:  50 34 1000 - Return value:  2700
Arguments:  51 35 1000 - Return value:  2785
Arguments:  52 36 1000 - Return value:  2872
```

## Exercise 8 (7 bonus points)

Implement the same profiler specified in Exercise 7 (i.e., a profiler that detects all invocations of method `targetMethod` of class `exercise8.MainThread`). However, this time, the profiler **must** instrument only method `run` inside class `exercise8.MainThread` (i.e., the only method that calls `targetMethod`).