JVM & tools

Exercise 1

Write a simple Java program **`JustCreate`** that in a very long loop creates at each iteration one object, discarding immediately any reference to it. Every 1000 iterations the program must print the number of objects created since the beginning and must pause for 50 milliseconds (use e.g. **`Thread.sleep()`**).

Inspect the program execution with VisualVM, and try to tune the parameters so that the heap occupancy profile has the shape of a shark's teeth.

Goal: Monitoring the heap occupation in VisualVM.

Exercise 2

As we have seen recently, the JVM has two instructions that can be used to compile a **`switch`** statement: **`tableswitch`** and **`lookupswitch`**. Taking inspiration from the examples seen at lesson, try to determine when your Java compiler uses **`tableswitch`** and when it uses **`lookupswitch`**.

Does this depend only on the distance between the smallest and the largest constants in the case clauses? Are other criteria considered as well?
Use **`javap -c -v ClassName.class`** or **`godbolt.org`** to disassemble the compiled bytecode.

Goal: Reading disassembled bytecode to infer the compilation scheme for simple but non-trivial high level control structures,

Exercise 3

Run the program **`WrongQueue`** and inspect its behaviour using **`VisualVM`**. Can you explain the continuous growth of the heap? Find the code causing the bug and fix it.

Clearly, there are several ways to fix the bug: for this exercise, try to find one with minimal "edit distance" from the source, i.e. the one obtained with the minimum amount of character insertions or deletions.

Goal: Using **`VisualVM`** to inspect the memory consumed by a Java program; Reviewing Java code to detect non-trivial errors; Fixing bugs.

Exercise 4

In the last lesson, the lecturer claimed that when compiling a **`switch`** based on a **`String`** value, the Java compiler uses a hashing function for strings. Verify that this is true. Also, try to understand the bytecode produced by the compiler: if the hash value coincides with

the hash of a string of a `case` clause, apparently an additional comparison between the two strings is performed. Why is this necessary?

Using `javap` or `godbolt.org`, disassemble the compiled code to see if the lecturer is right. Also, search in the Java or JVM Specification the precise place where this compilation scheme is prescribed.

Goal: Reading disassembled bytecode; checking the Java/JVM specification.

Exercise 5

Exploring threads. Explore the threads of a running application using VisualVM. Write a simple multi-threaded program which spawns a new thread every few second. Monitor the threads, their state and the code they are executing using the thread monitor of VisualVM and the Thread Dump facility. Look for information about the threads dedicated to JIT compilation and to GC,