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 one second (use e.g. **`Thread.sleep()`**).

Inspect the program execution with VisualVM.

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`** 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 continuos growth of the heap? Find the code causing the bug and fix it.

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

Exercise 4

During 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. Also, if the hash value coincides with the hash of a string of a **`case`** clause, an additional comparison between the two string is performed.

Using **`javap`**, 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; cheking the Java/JVM specification.

Exercise 5

Run and inspect the program **`GCstrange`** using **`VisualVM`**, in particular check the evolution of the heap, the activity of the GC and the activity of the **`Finalizer`** thread. Using **`GCstrange`** as a template, write a simple class that overrides the method **`finalize`** in order to estimate how many times the garbage collector is invoked. Write a main to test this class.

Goal: Understanding the finalization of objects in Java; exploiting the finalize method to infer simple properties of garbage collection.