Java Performance Tuning, Tools - JMH

Read README.md ;) as well

https://github.com/azatsatklichov/Java-Features

Azat Satklichov
azats@seznam.cz,
http://sahet.net/htm/java.html

Agenda

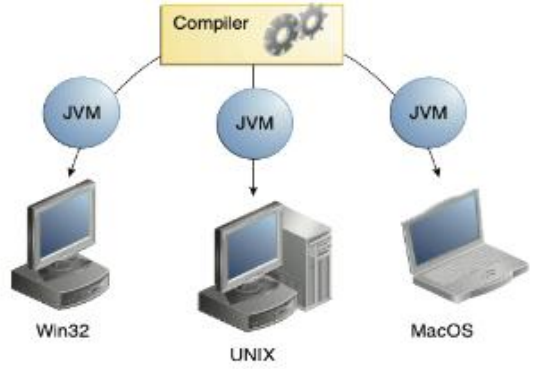
See also, demos:
"JVM Memory Management - Garbage Collection, GC Tools, ..
"JVM Architecture, Classloading, etc. "
"Java Graal VM

- ❑ Java Platform
- ❑ Java Nostalgia - Sun Microsystems and Oracle
- ❑ JVM Architecture
- ❑ Types of Memory areas Allocated by the JVM
- ❑ ClassLoading
- ❑ ..

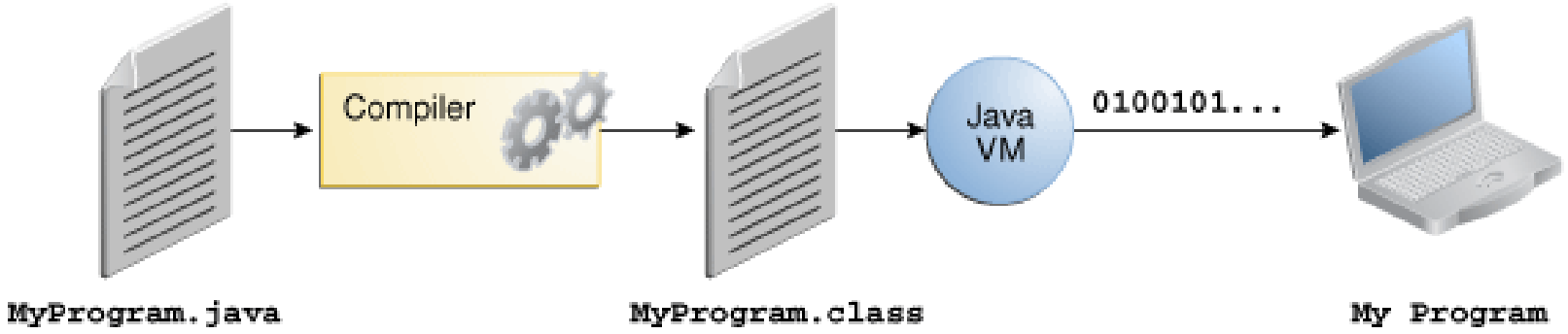
Java Platform

Five main goals which Java language intended to bring

- Simple, object-oriented, distributed
- Robust and secure.
- Architecture-neutral (agnostic) and portable.
- Execute with high performance.
- Interpreted, multi threaded, and dynamic.



Through the Java VM, the same application is capable of running on multiple platforms.



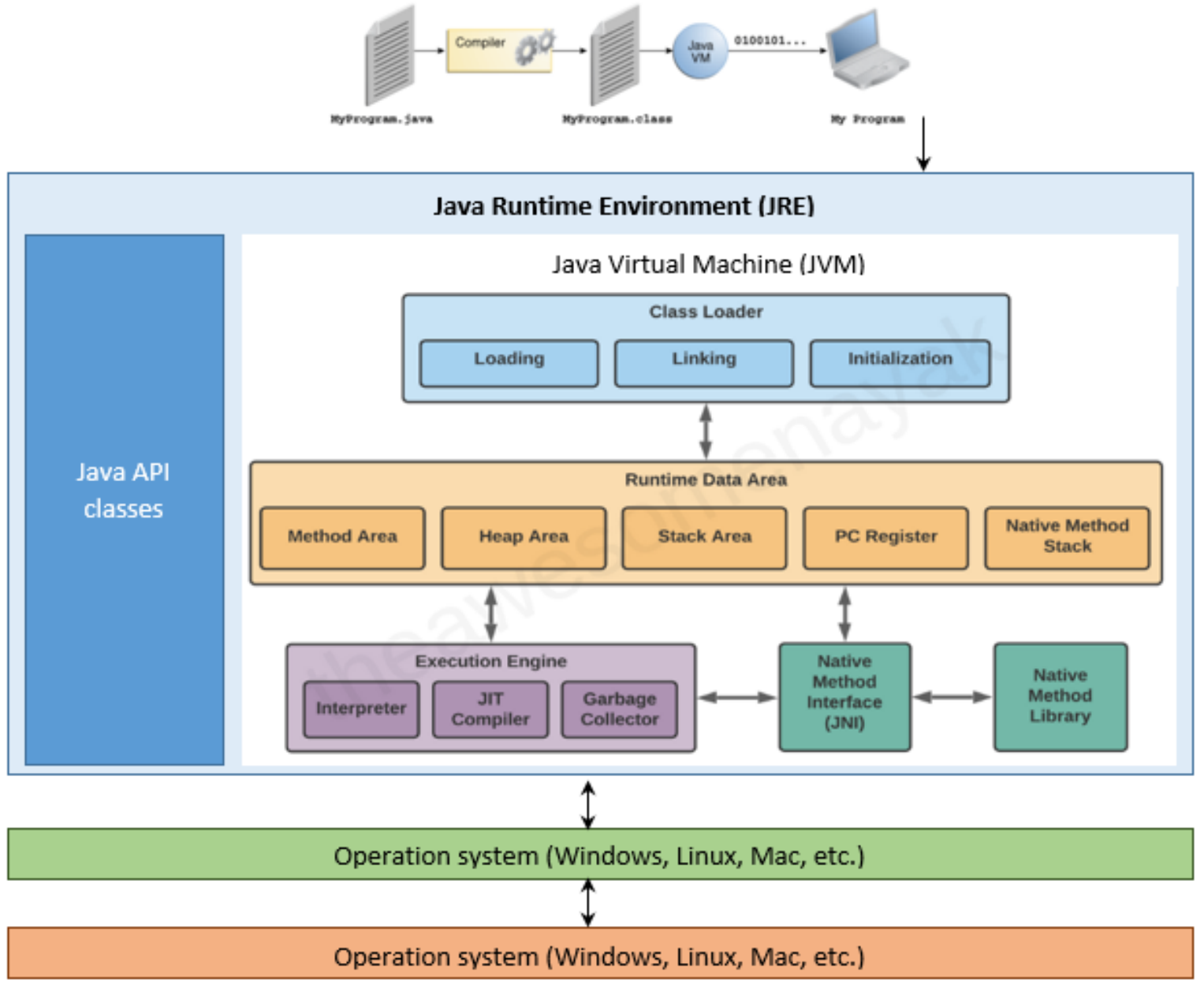
An overview of the software development process.

JVM Architecture

A virtual machine is a *virtual representation of a physical computer*. A **Java virtual machine** (**JVM**) is a virtual machine that enables a computer to run Java programs as well as programs written in other languages that are also compiled to Java bytecode.

The five major components inside JVM are

- Class Loader (loads class files to RAM)
- Memory Area (contains runtime data)
- Execution Engine (executes byte-code using interpreter)
- Native Method Interface
- Native Method Library



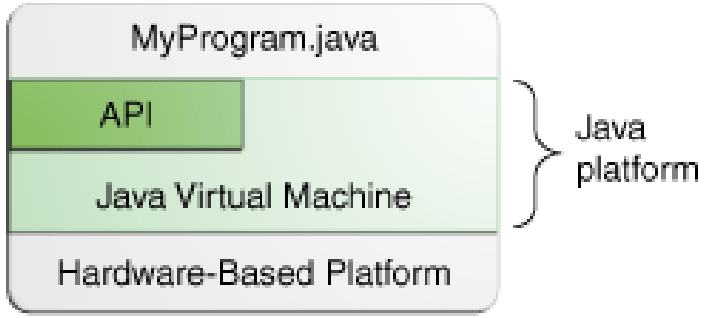
JVM Architecture

Java Platform

Java Platform

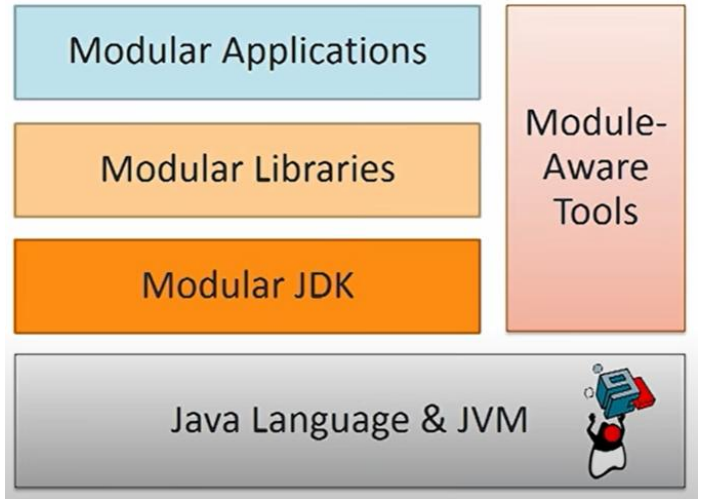
The Java platform is a software-only platform that runs on top of other hardware-based platforms (Win / Linux / MacOS).
The Java platform has two components:
- The Java Virtual Machine - The Java Application Programming Interface (API)

As a platform-independent environment, the Java platform can be a bit slower than native code. However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability. All Java Platforms consist of a JVM, an API, and other platfrom specific components.



The API and Java Virtual Machine insulate(isolated) the program from the underlying hardware.

Illustrates Java 8 and before.
Since Java 9, JRE is consumed in JDK. See sahet.net for Java Platform in detail



Java Platform since Java 9 - It is modular

Micro Benchmarking with JMH – JEP 230 (JDK 12 uses JMH)

Throughput – Operations per time unit / number of trx per second (**HIGHER is better**). Measure **ops/sec -** Mode.Throughput

Latency [Average Time] – maximum duration of a transaction (**LOWER is better**). Measure **secs/op** - Mode..Average

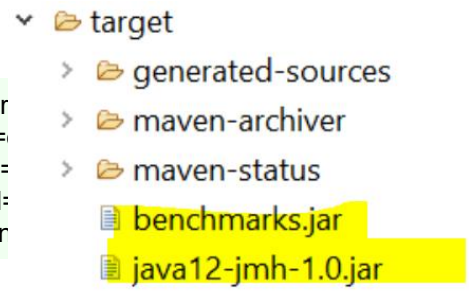
- JMH is all about monitoring and measuring performance of piece of code. Measures in **nano**/micro/milli/…,
- Measure exec-time, compare alternatives algorithms, (choose algs., obj.immutability or pooling. Condition-first to avoid catching Exc)
), prevent performance regressions.
- JMH – Reproducability [repeating calls], JVM **warm-up** handling, runs benchmark multiple times, consistent reporting, multithreading support.
- @Benchmark - by default Mode.Throughput – operations per time unit, or Mode.AverageTime shows average running(exec) time or [latency] . Mode.SimpleTime (verbose Average), .. Mode.ALL
- Default: Forks 5 JVM, 5 diff. runs. Use @Fork(1) **to tune**
- TimeUnit.***NANOSECONDS , ..***
- Optional to use: @Setup, @Teardown (like in Junit)

- **Pitfalls**: Dead code eliminations (better to **return value, or Blackhole if more value to return**), compiler optimizations (combine statements, loops, constant folding), assumptions (just numbers …)
- You can integrate JMH output into **CI/CD pipeline**, to publish or visualize on different runs. Helps comparing Performance regression
(another measure like regression testing)

```
//Don't Do This - via System.currentTimeMillis(), just benchmark once
//e.g. no consider optimizations, etc. .. Many runs, ..
long start = System.currentTimeMillis();
//code under benchmark
long end = System.currentTimeMillis();
long elapsed = end - start;
System.out.println("Elapsed time: " + elapsed);
```

```
<dependency>
    <groupId>org.openjdk.jmh</groupId>
    <artifactId>jmh-core</artifactId>
    <version>1.21</version>
</dependency>
<dependency>
    <groupId>org.openjdk.jmh</groupId>
    <artifactId>jmh-generator-annprocess</artifactId>
    <version>1.21</version>
</dependency>
```

```
mvn archetype:gener
DarchetypeGroupId=
DarchetypeArtifactId=
archetype -DgroupId=
benchmark -Dversion
```



> java -jar target/benchmarks.jar

THANK YOU

References

https://medium.com/platform-engineer/understanding-jvm-architecture-22c0ddf09722