



Wprowadzenie do technologii JVM

Szymon Szepietowski



Senior Software Engineer & Team Leader @ Cosmose

Branża marketingu online

- 7 lat w branży IT.
- Śmiało odzywajcie się na slacku.

Agenda



1. Dlaczego Java?
2. Czym jest JVM?
3. Podstawowe narzędzia linii komend.
4. Model pamięci w Javie.
5. Monitorowanie aplikacji.

Repo: <https://github.com/sszepiet/jvm-intro>



Dlaczego Java?

- Write Once Compile Anywhere (WORA) – aplikacja napisana w Javie może zostać uruchomiona na dowolnym systemie operacyjnym z takim samym rezultatem.
 - W innych językach takich jak C++ mogą być zauważalne różnice pomiędzy uruchomieniami na różnych środowiskach, np. nie jest gwarantowana wielkość w pamięci zmiennej typu integer.
- Automatyczne czyszczenie pamięci (Garbage Collector).
- Wsparcie dla paradygmatu programowania obiektowego.

Czym jest JVM?



- Program odpowiedzialny za wykonywanie kodu.
 - Ale jakiego kodu?
- Dostarcza warstwę abstrakcji nad systemem operacyjnym oraz sprzętem (dzięki temu możemy cieszyć się z WORA ;))
- Zawarty w Java Runtime Environment oraz Java Development Kit.



- Co to jest abstrakcja? Warstwa abstrakcji?





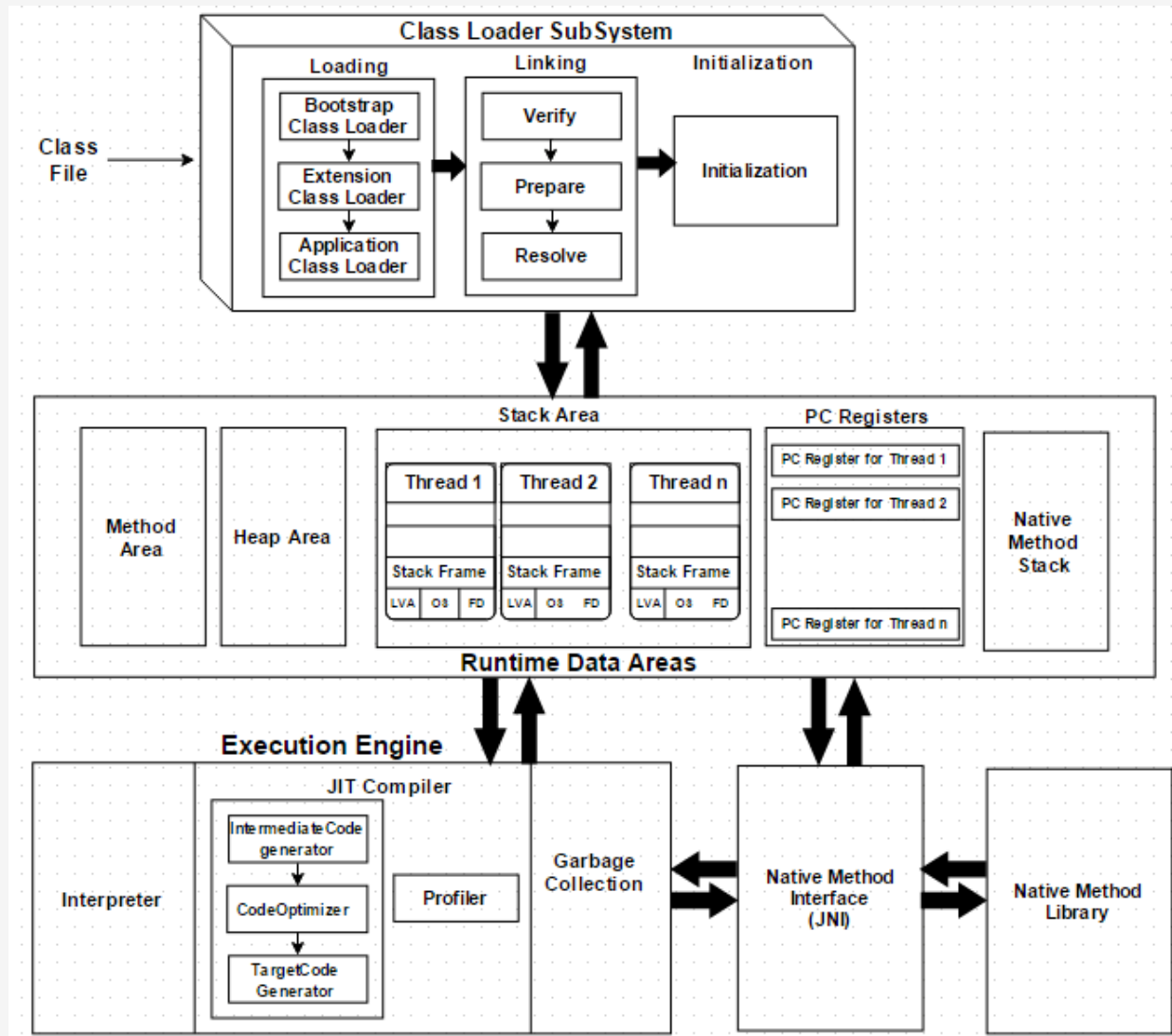
- Co musi mieć aplikacja, żeby być przydatną?



Kilka definicji

- **Byte-code** - rezultat kompilacji kodu napisanego w Javie, zestaw instrukcji dla JVM. Po samej kompilacji może być dodatkowo modyfikowany i optymalizowany.
- **Just in time compiler** - kompilator stosujący dodatkowe optymalizacje na poziomie byte code'u, działający już po uruchomieniu aplikacji na JVM.
- **Garbage collection** - proces usuwania z pamięci obiektów, które nie są już dłużej używane.

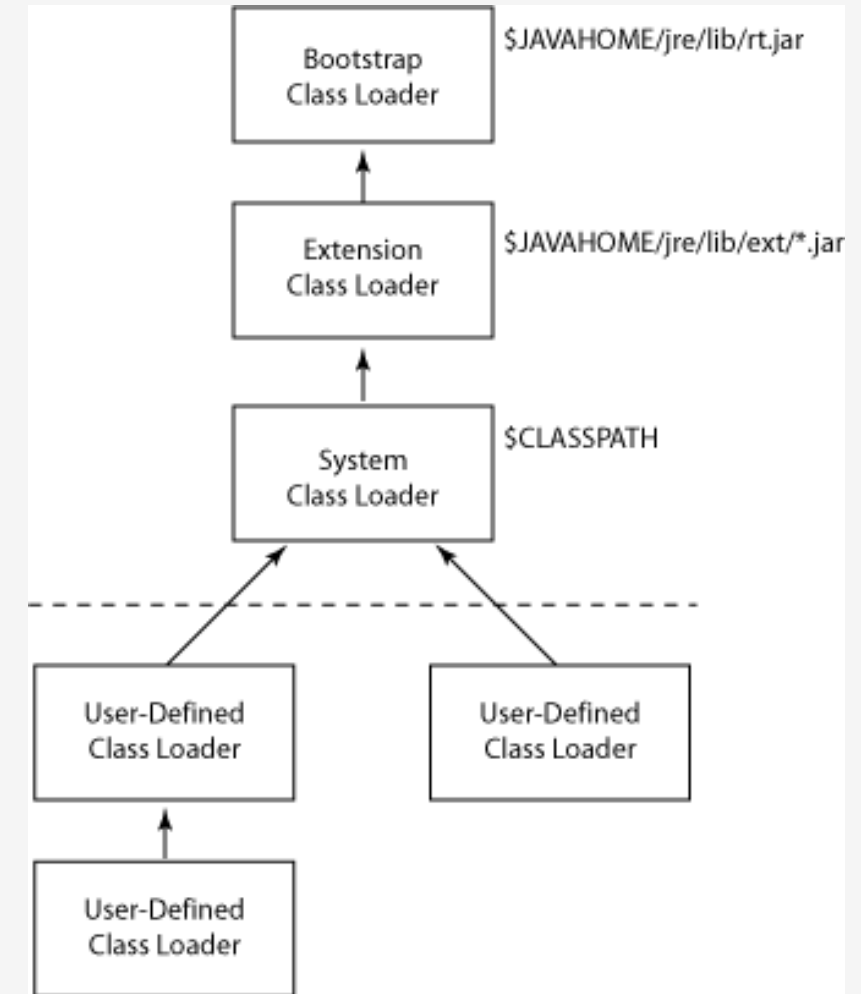
JVM - architektura



JVM - class loader



- ładuje obiekty pochodzące z byte-code'u
- uruchamiane są hierarchicznie (bootstrap, extension, system)
- istnieje możliwość definiowania własnego class loadera



JVM - nie tylko Java



- Kotlin
- Scala
- Groovy



Postawowe narzędzia linii komend

- Kompilacja: `javac -cp <class_path oddzielony ';'> -d <directory> <class to compile>`
- Show bytecode: `javap -v <compiled_class>`
- Uruchomienie: `java -cp <class_path oddzielony ';'> <class to run>`
- Budowanie jar: `jar -cf <jar_name>.jar -C <directory> .`
 - Dodatkowe parametry `e` – podanie EntryPoint, `m` – podanie manifest.



Model pamięci w Javie

- dwa główne obszary pamięci: **stos i sterta**
- **stos** przechowuje ramki programu dla każdego wątku oraz referencje do obiektów na sterckie, a także zmienne lokalne
- **sterta** przechowuje instancje obiektów, ma strukturę generacyjną – jest podzielona na części gdzie zaalokowane są krótko i długo żyjące obiekty
- **method area** – przechowuje metadane załadowanych klas i stałe



- wspólna dla wszystkich wątków
- przechowuje instancje obiektów
- tylko Garbage Collector może usunąć obiekty ze sterty
- struktura pokoleniowa
 - Młoda generacja (Eden, Survivor)
 - Stara generacja
- struktura pokoleniowa pomaga zapobiegać fragmentacji pamięci
- ponadto spodziewamy się, że młodsze obiekty będą żyć krócej



- wspólna dla wszystkich wątków
- przechowuje instancje obiektów
- tylko Garbage Collector może usunąć obiekty ze sterty
- struktura pokoleniowa
 - Młoda generacja (Eden, Survivor)
 - Stara generacja
- struktura pokoleniowa pomaga zapobiegać fragmentacji pamięci
- ponadto spodziewamy się, że młodsze obiekty będą żyć krócej

Jak zaimplementować GC?



- Skalarnie:
 - Zliczać referencje.

Nie uda się wtedy wykryć cykli, kiedy kilka obiektów posiada referencje do siebie nawzajem.

Jak zaimplementować GC?



- Grafowo:
 - Znaleźć obiekty będące korzeniami.
 - Oznaczyć obiekty osiągalne zaczynając od korzeni.
 - Usunąć obiekty nieoznaczone.

Wykrywa pętle, ale jest dużo trudniejsze w implementacji, zdarza się stop-the-world, żeby zatrzymać zmiany w pamięci celem jej analizy.

Jak zaimplementować GC?



- Grafowo:
 - Znaleźć obiekty będące korzeniami.
 - Oznaczyć obiekty osiągalne zaczynając od korzeni.
 - Usunąć obiekty nieoznaczone.

Wykrywa pętle, ale jest dużo trudniejsze w implementacji, zdarza się stop-the-world, żeby zatrzymać zmiany w pamięci celem jej analizy.



Parametry pamięci

- `-Xms1024Mb` – startowy rozmiar sterty
- `-Xmx1024Mb` – maksymalny rozmiar sterty
- `-XX:+HeapDumpOnOutOfMemoryError`



- JMX (Java Management Extension) – umożliwia wgląd do parametrów JVM.
- JConsole.
- JVisualVM.

Koniec



- Pytania?