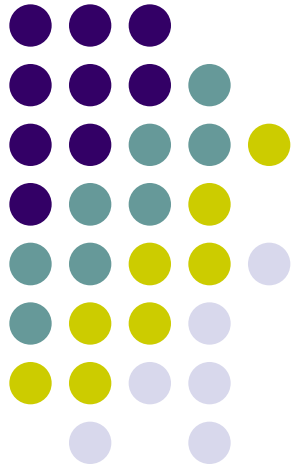
Мова програмування Java та технології J2EE
Лекція 1

Особливості мови програмування Java

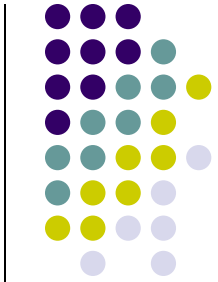


Сирота О.П.

Скорочення

- JDK = Java Development Kit
- JRE = Java Runtime Environment
- J2SE = Java 2 Standard Edition
- JavaSE = Java Standard Edition
- J2EE = Java 2 Enterprise Edition
- JavaEE = Java Enterprise Edition
- JavaME = Java Micro Edition
- JVM = Java Virtual Machine

Історія мови Java



Все почалося в 1990-1991 році з побутових пристроїв…

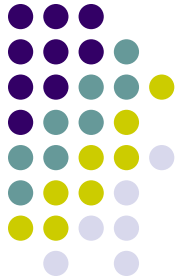
1991	Розпочався проект Green по розробці нової технології програмування
1992	Створено мову **Oak**
1994	Випущено Веб-броузер **WebRunner**
1995	Народилася мова **Java** (шляхом переіменування **Oak**)
1996	JDK 1.0
1997	JDK 1.1 (прототип Standard Edition)
1998	Personal Java (прототип Java MicroEdition)
1998	Java Professional Edition (прототип Enterprise Edition)
1998	Засновано процес стандартизації Java - Java Community Process (JCP)

- Сучасні сфери застосування
  - Розподілені інформаційні системи
  - Desktop-застосування
  - Мобільні телефони
  - Смартфони
  - Побутові пристрої
  - «Розумний дім»
- Предмет нашого курсу – Java SE та Java EE – розподілені інформаційні системи
- Процес JCP – специфікації JSR-XXX

Версії JavaSE

Версії	Функціональність у складі JDK	Рік
JDK 1.0		1996
JDK 1.1	- Внутрішні класи - AWT - JDBC - RMI - reflection	1997
J2SE 1.2	- Swing - Collections - Java IDL (CORBA IDL) - JIT-компілятор	1998
J2SE 1.3	- JNDI - RMI over IIOP - CORBA ORB	2000
J2SE 1.4	- XML, XSLT - регулярні вирази - logging API - Security, cryptography (JCE, JSSE, JAAS) - Java Web Start	2002
J2SE 5	- Новий синтаксис (для циклів, generic, Enum, функції із змінною кількістю параметрів) - Анотації	2004
Java SE 6	- Підтримка скриптових мов (javax.script) - Покращений моніторинг JVM - Java Smart Card I/O API - JAXB (XML->Java object, Java object->XML)	2006
Java SE 6 update 01-21	- Інші	2006-2010
Java SE 7	- VM support for non-Java languages - Інші	2011
Java SE 8	- lambda	2014
Java SE 9	- modularization	

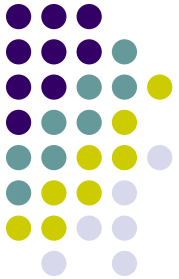
Стандартизація



OMG	CORBA, UML, BPMN
W3C	HTTP, HTML&CSS, Ajax XML, XSLT, XPath, XSL-FO WSDL (SOAP) та багато-багато інших
OASIS	ODF (Open Document Format for Office Applications) SAML SPML UDDI та багато-багато інших

- **Розробка специфікацій Java здійснюється суспільством JCP (Java Community Process) – з 1998 р.**

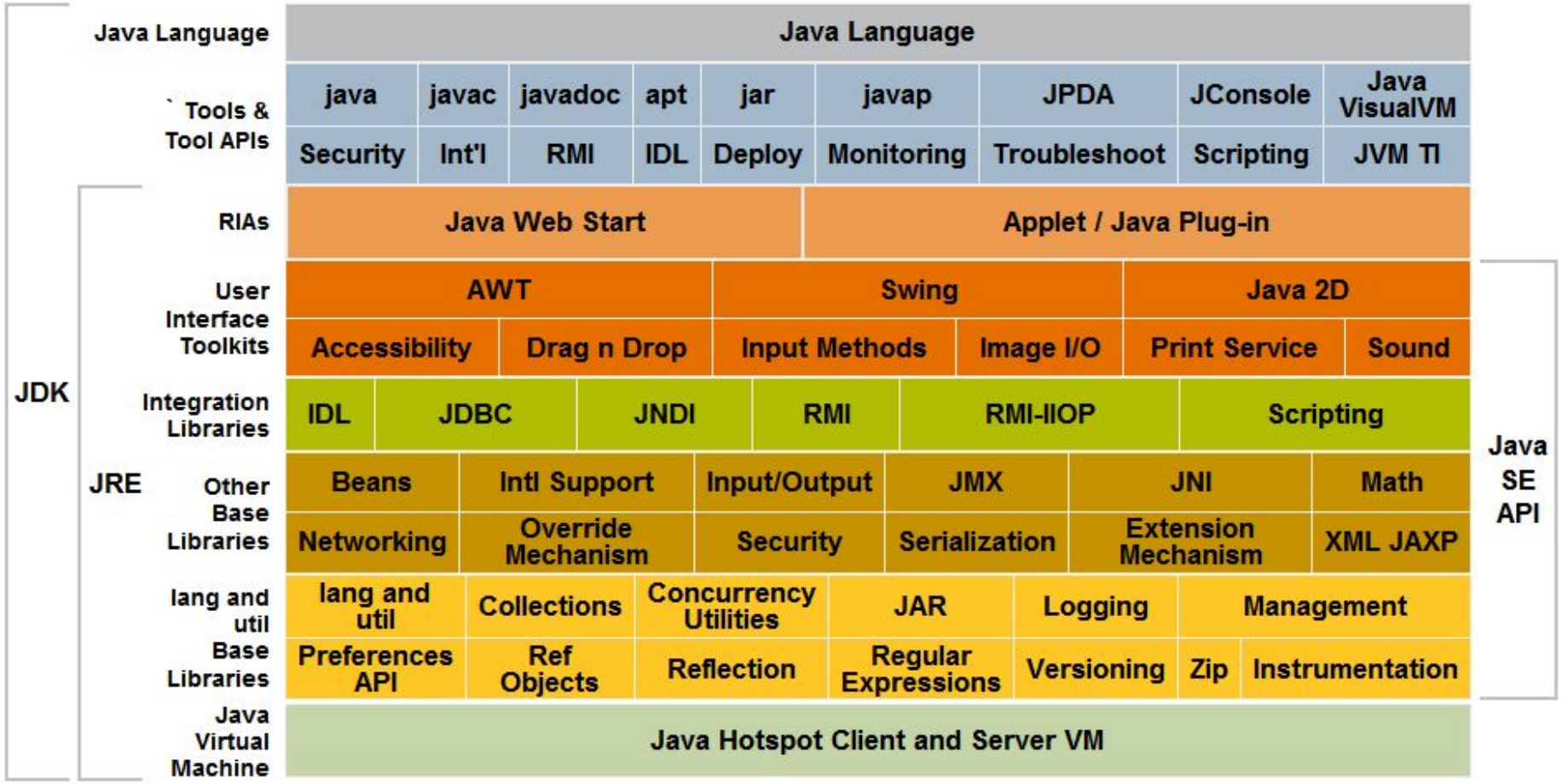
Виробники JavaSE



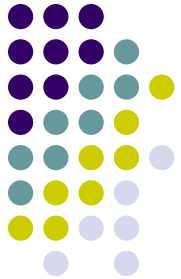
- Sun JDK (->Oracle JDK)
- IBM Java SDK
- Open JDK

JDK JRE JavaSE

- JDK – платформа для розробки
- JRE – платформа для виконання Java-програм
- JavaSE - API

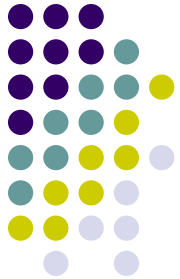


Відмінності від C++

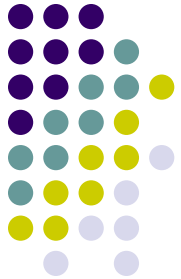


- **Ваші пропозиції**

Відмінності від C++



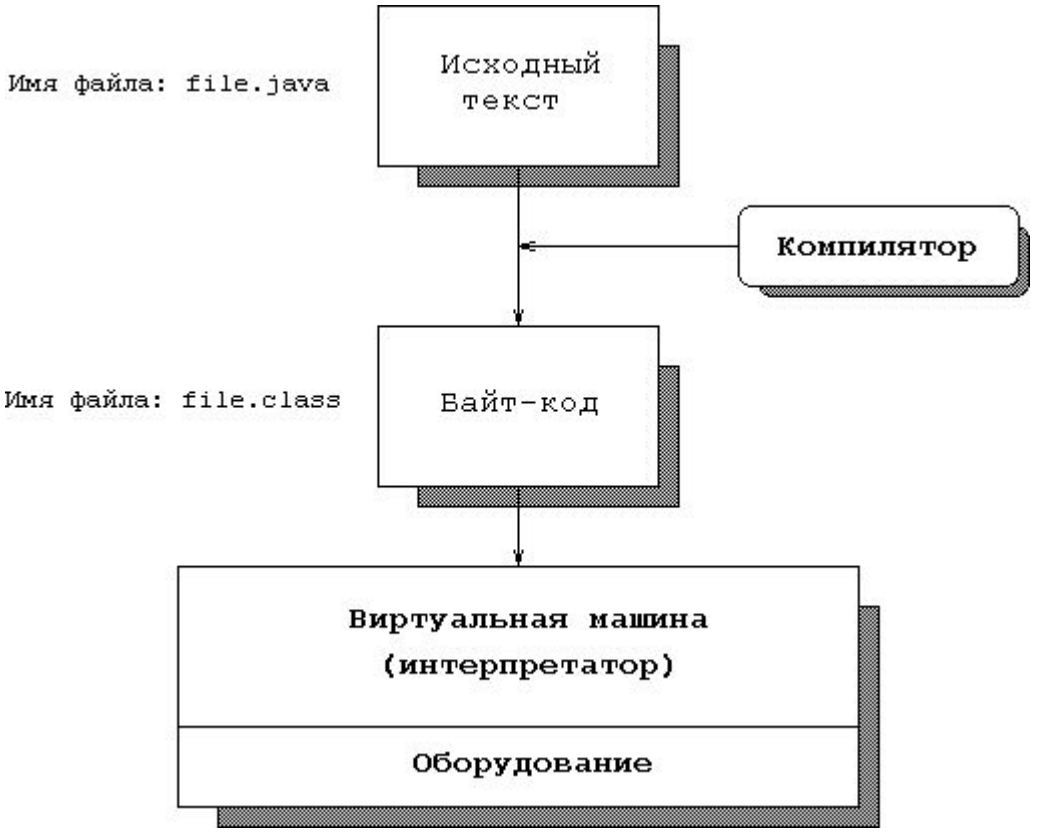
- Чиста об’єктно-орієнтована мова
- Кореневий об’єкт «java.lang.Object»
- Відсутні вказівники, тільки “посилання”
- Виділення пам’яті для об’єктів – тільки в області «heap» («куча»)
  - Java: o = new myobject() - heap
  - C++ : o= new myobject() – heap, o = myobject() – stack.
- Відсутня адресна арифметика
- Збирання сміття
- Відсутнє множинне успадкування класів (можливе множинне успадкування тільки інтерфейсів)
- Відсутні пре-процесор та макроси
- Відсутнє перевантаження операторів
- Пакети замість просторів імен («namespace»)
  - Обробка виключних ситуацій в Java є обов’язковою та контролюється компілятором (Checked/Unchecked Exceptions)
- Інтроспекція, рефлексія



JAVA-ПРОГРАМА

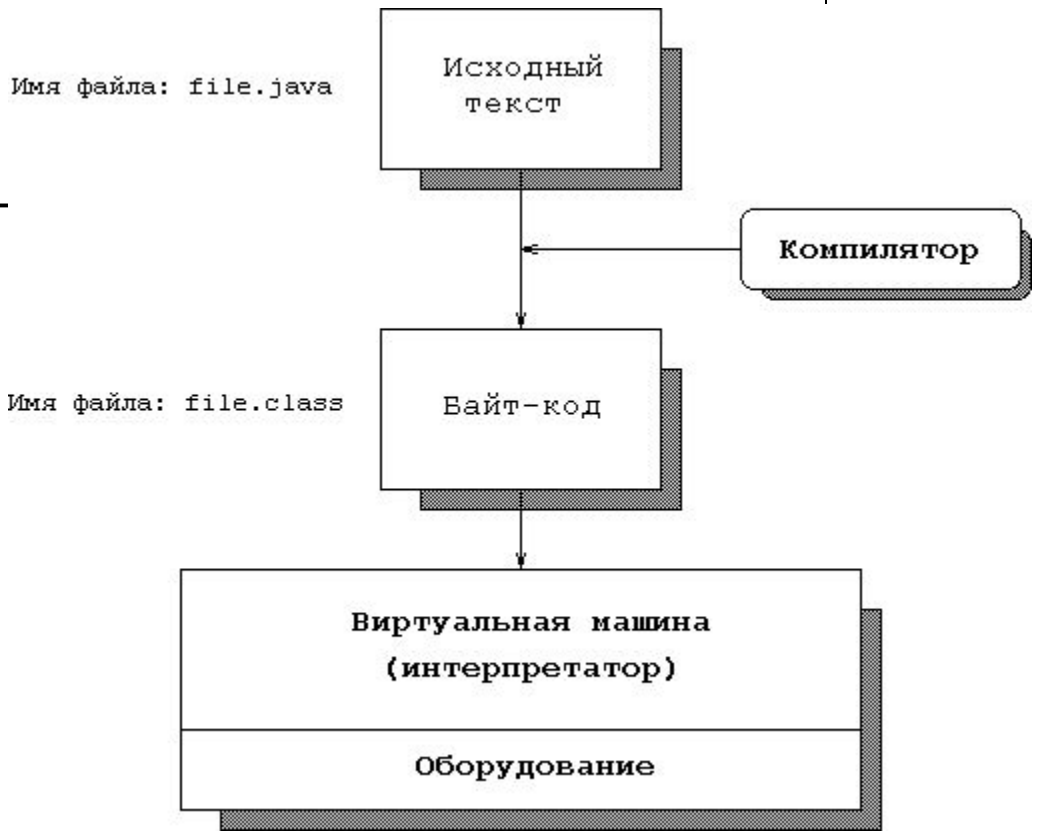
Java-програма

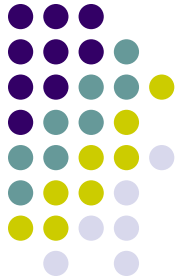
- Код програми пишеться на мові програмування Java у файлі *.java
- Код програми компілюється компілятором `javac`. В результаті отримаємо байткод (файл *.class)
- Байткод виконується за допомогою програми `java`. Ця програма запускає віртуальну машину JVM (Java Virtual Machine)



Характеристики Java-програми

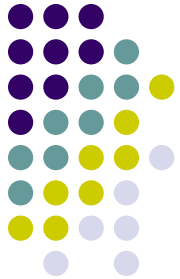
- Характеристики
  - Незалежність від апаратного забезпечення
  - Незалежність від операційних систем
- Незалежність завдяки JVM





JAVA VIRTUAL MACHINE (JVM)

Старт JVM

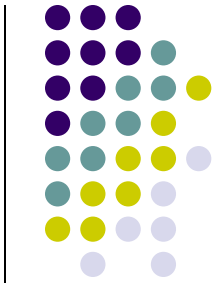


- JVM подається на вхід початковий клас
- Виконується метод `main` початкового класу

```
> java HelloWorld
```

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println(“Hello World!”);
    }
}
```

JVM (Java Virtual Machine)



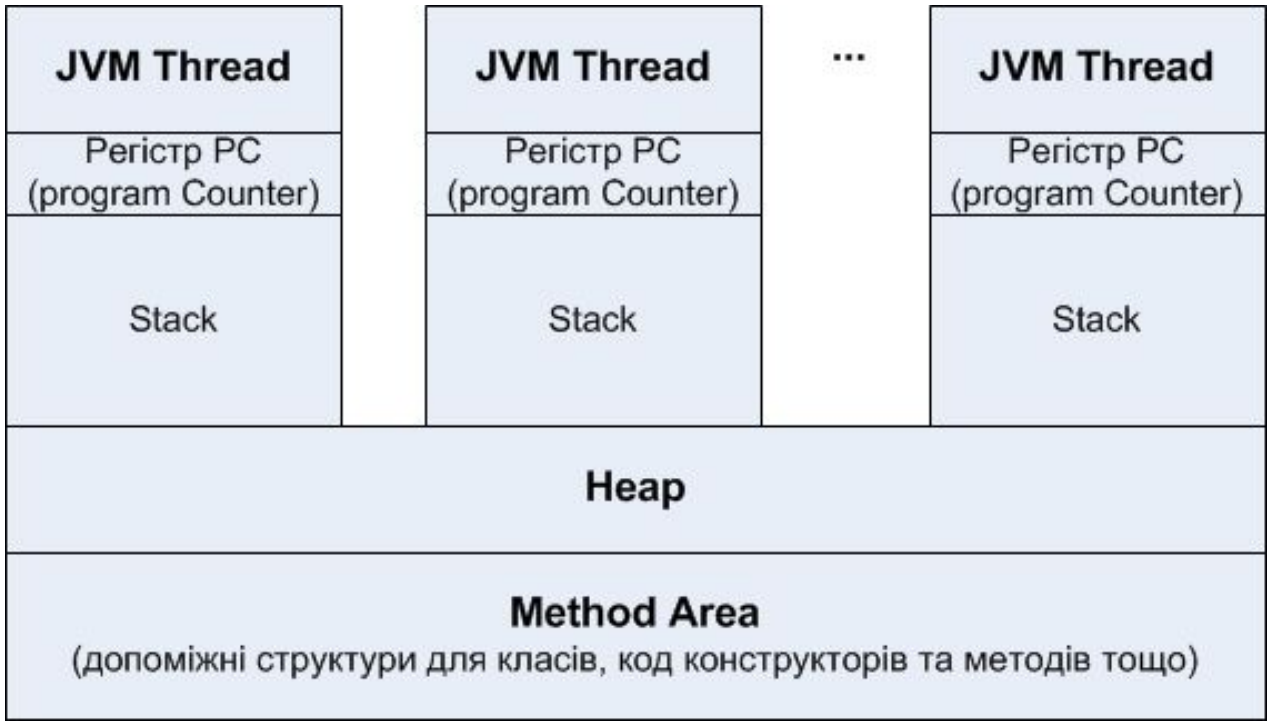
- JVM – абстрактна обчислювальна машина
  - Має власний набір інструкцій
  - Нічого не знає про мову Java
  - Виконує байт-код (bytecode) – набір інструкцій JVM
- Class-файли
  - Містять байткод та допоміжну інформацію
  - Java-програми компілюються у байт-код та розповсюджуються у вигляді “class”-файлів або їх архівів (“jar”-файлів)
- Non-Java JVM?
  - Мови із статичною типизацією: Ada, C, Pascal
  - Мови із динамічною типізацією (скриптові мови) – починаючи з JDK 7

Типи JVM

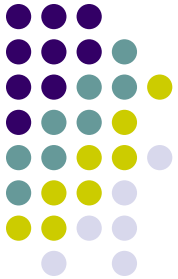
- Software JVM
  - Sun JDK, IBM Java SDK, Open JDK
- Hardware JVM
  - PicoJava, ARM Jazelle
- Embedded JVM
  - Портативні пристрої, побутові пристрої - Java 2 Micro Edition – Scelmer CEE-J, Jeode
  - Веб-броузери - аплети

Структура JVM

- JVM – це абстрактна **стекова** обчислювальна машина
- Має власні потоки виконання (JVM Thread)
- Містить загальні для всіх потоків виконання області Heap, Method Area



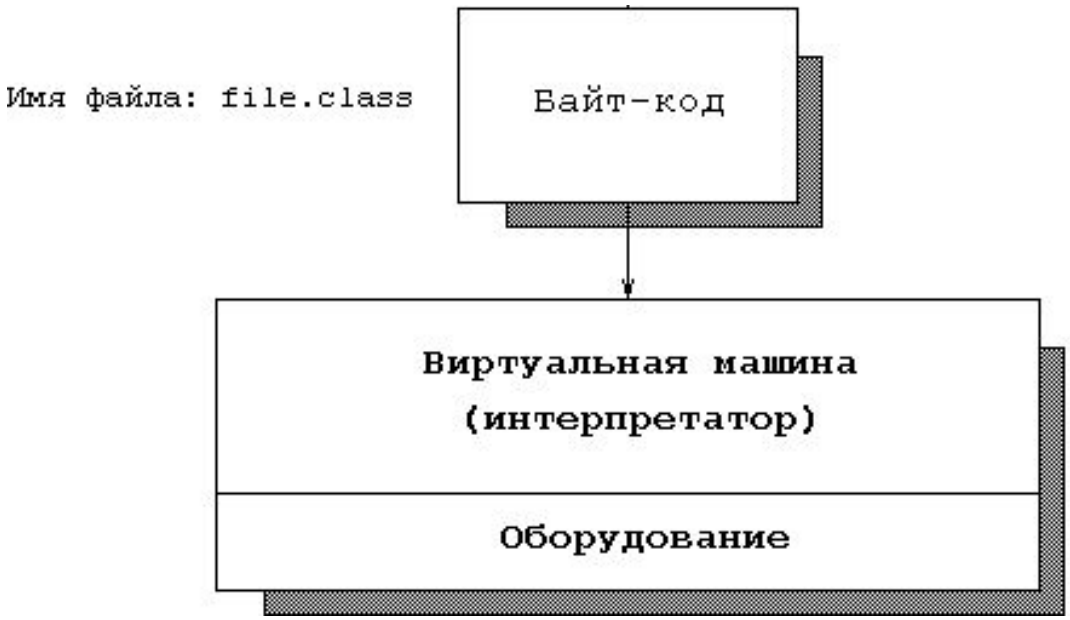
Функції JVM



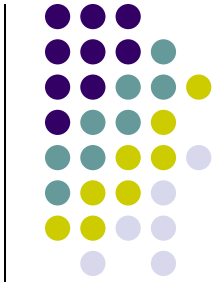
- Інтерпретація
  - Для підвищення продуктивності може бути залучений JITC (Just In Time Compiler).
- Class Loader – завантаження класів
  - Завантаження, зв’язування, ініціалізація класів
- Garbage Collector (GC) - збирання сміття
  - Здійснюється неявний виклик, якщо в області “heap” неможливо виділити пам’ять
  - Для явного виклику GC застосовуються System.gc() – але такого краще не робити
  - Розмір “heap” встановлюється параметром -Xmx
- Паралельне виконання
  - Можливість одночасної роботи декількох потоків виконання

Інтерпретація

- JVM виконує байт код
  - Виконання відбувається шляхом трансляції кожної команди байт-коду в машинний код
  - Така трансляція відбувається постійно під час виконання кожної команди з байт-коду



JITC

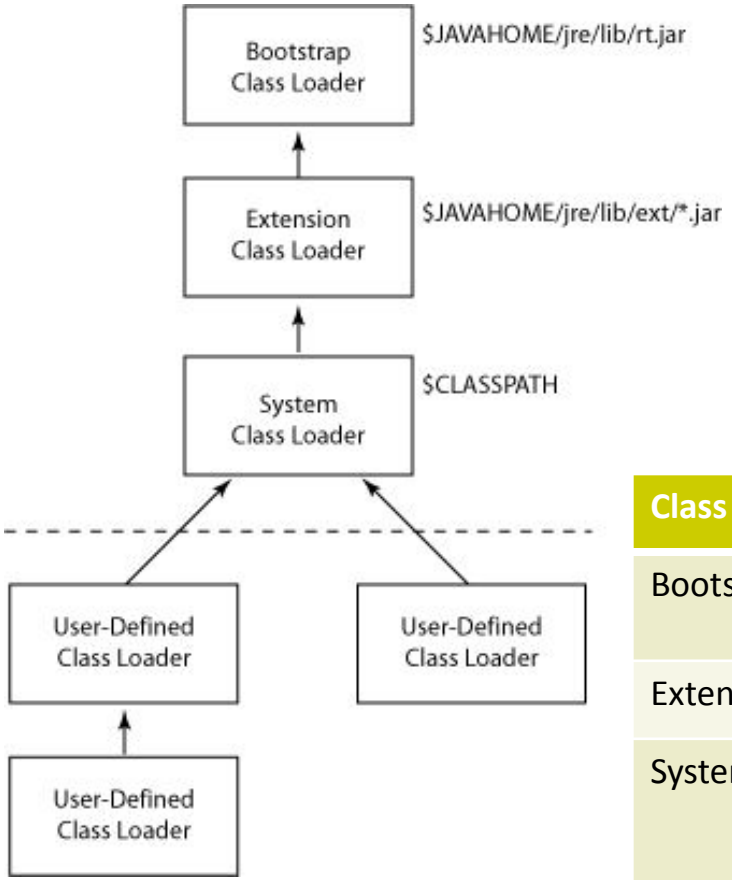


- JITC (just in time compilation) –
  - Це оптимізація виконання байт-коду
  - Кешування машинного коду для раніше трансльованого байткоду

```
>java -version
java version "1.6.0_21"
Java(TM) SE Runtime Environment (build
  1.6.0_21-b07)
Java HotSpot(TM) Client VM (build 17.0-b17, mixed
  mode, sharing)
```

Java HotSpot – це вбудований Just In Time Compiler

Завантаження класів



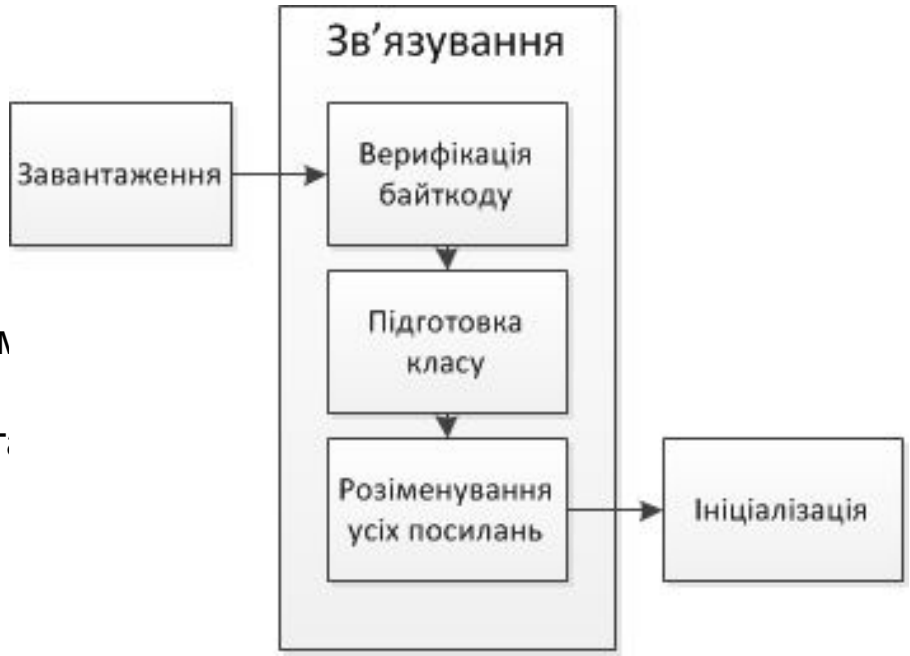
- Завантаження Java-класів здійснюють завантажувачі класів (class loader)
- Завантажувачі класів організовані у дерево
- Кожний наступний завантажувач класів бачить тільки ті класи, які завантажені попереднім завантажувачем

Class loader	Пояснення
Bootstrap	Завантажує внутрішні класи JDK, класи з пакетів java.*
Extensions	Завантажує jar-файли з каталогу lib/ext
System	Завантажує jar-файли з classpath (визначається або системною змінною $CLASSPATH або опцією java -cp
User-Defined	Власні завантажувачі класів. Наприклад, для завантаження класів, які зберігаються в БД

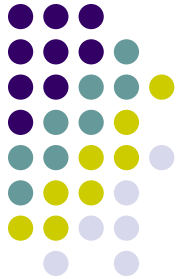
Процес завантаження класу

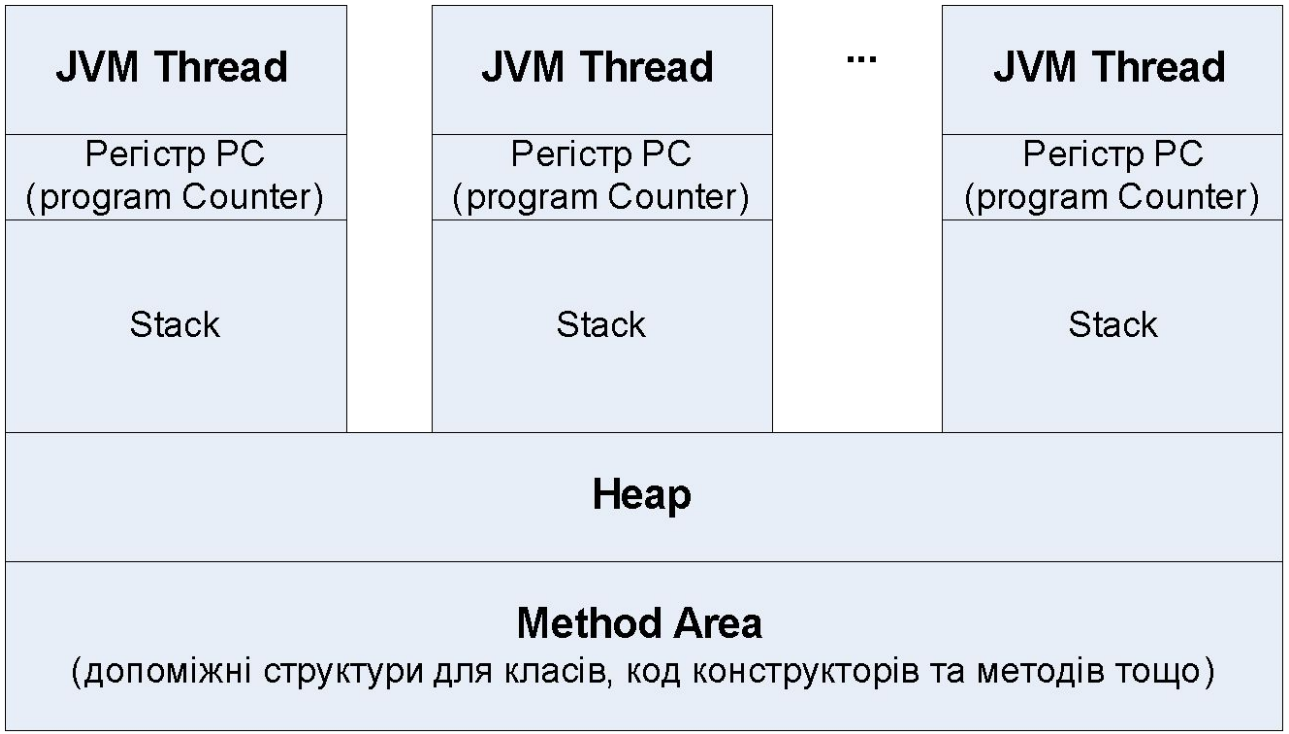
- В результаті клас завантажений та готовий до використання
  - Не плутати з інстаціюванням класу

- **Крок 1 – завантаження**
  - Пошук class-файлу
  - Завантаження байткоду
- **Крок 2 – зв'язування**
  - Верифікація байткоду
  - Підготовка класу (відтепер за допом Reflection API можна динамічно звернутися до класу, його методів т полів).
  - Розіменування усіх класів, на які посилається даний клас
- **Крок 3 – ініціалізація**
  - Виконання блоків статичної ініціалізації
  - Ініціалізація статичних полів



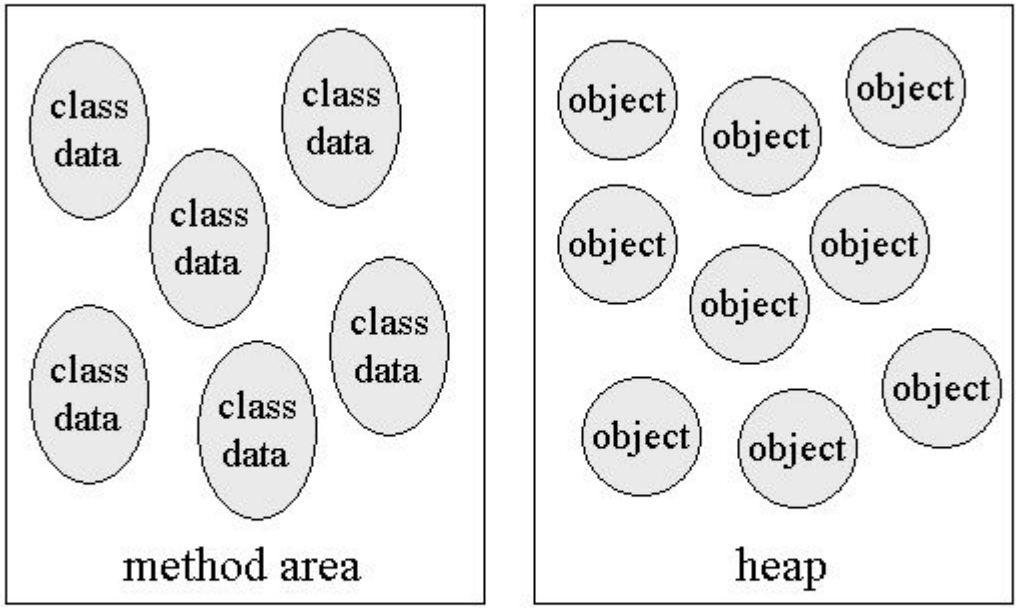
Паралельне виконання у JVM





- JVM управляє власними потоками виконання
  - Саме тому Java містить конструкції для роботи з багатопоточністю на рівні мови (наприклад, ключове слово synchronized)
- Кожний потік виконання містить свій регістр PC та стек
- Кожний потік виконання виконує байткод

Heap, Method area

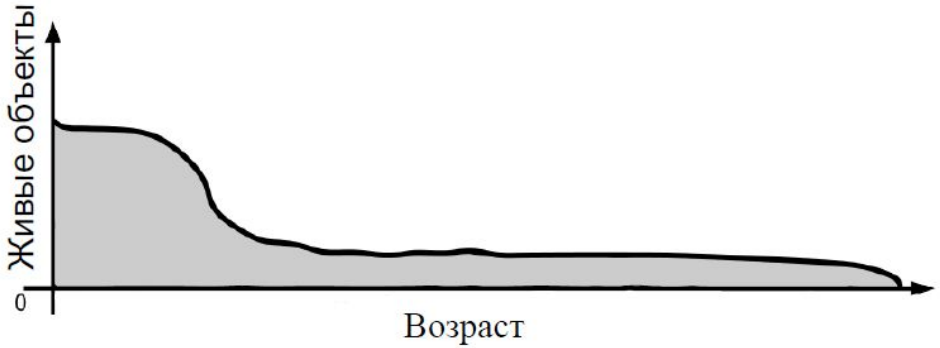


- *Heap memory* is the runtime data area from which memory for all class instances and arrays is allocated.
- *Non-heap memory*
  - **method area**. It stores per-class structures such as a runtime constant pool, field and method data, and the code for methods and constructors
  - memory required for the internal processing or optimization of the JVM.

- GC очищує Heap Memory

Garbage Collector (1/2)

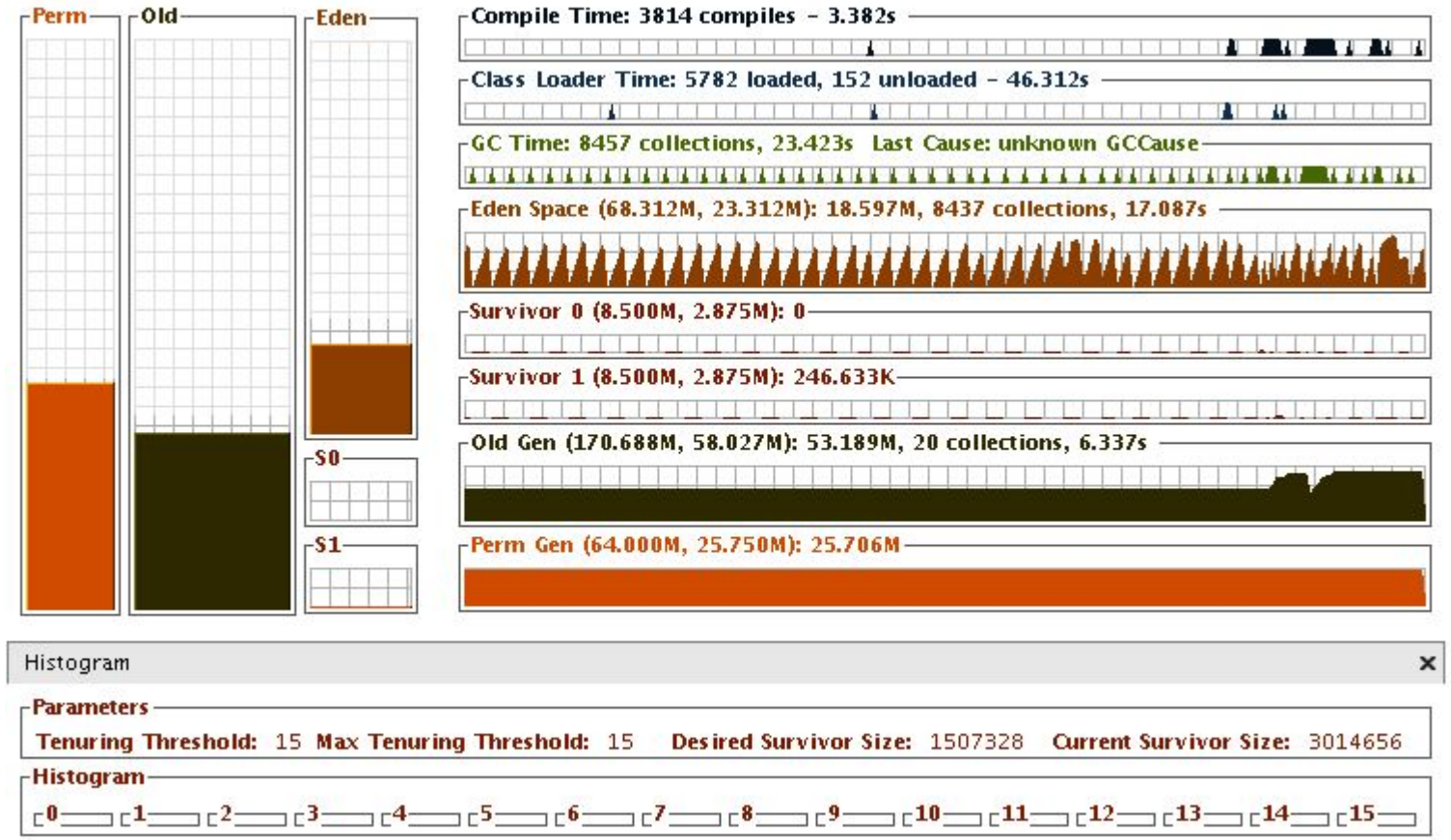
- Знаходить та звільнює місце, яке зайняте непотрібними об’єктами
- Алгоритм GC заснований на ідеї поколінь (most objects die young)



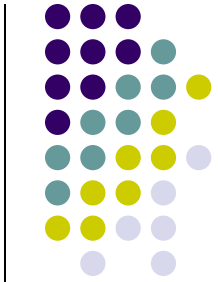
- Heap поділяється на покоління (розміри «поколінь» та ньюанси алгоритму GC є предметом тонкої настройки)

Garbage Collector (2/2)

- За допомогою утиліт JVisualVM / VisualGC можна слідкувати за роботою GC

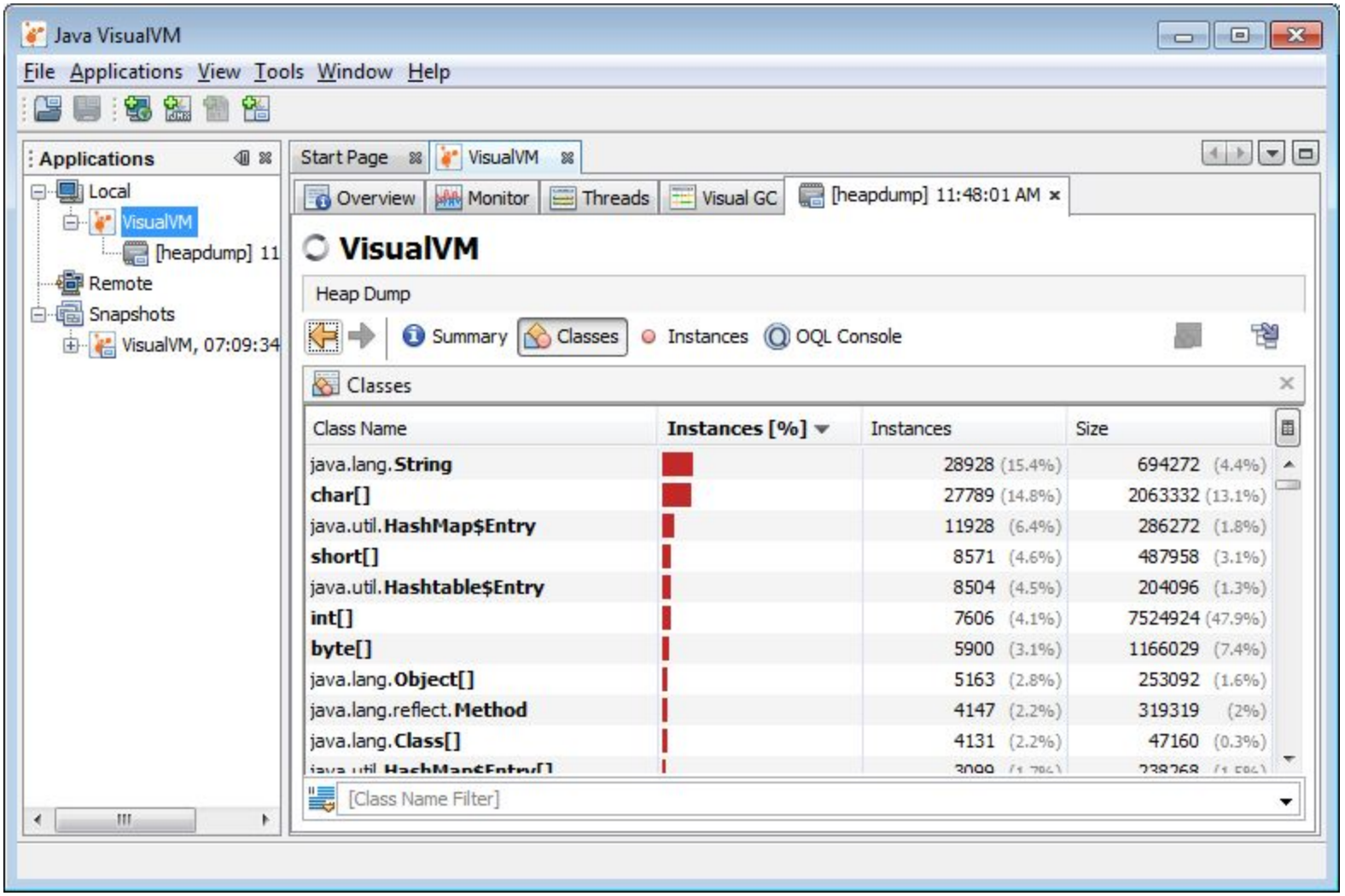


Моніторинг JVM

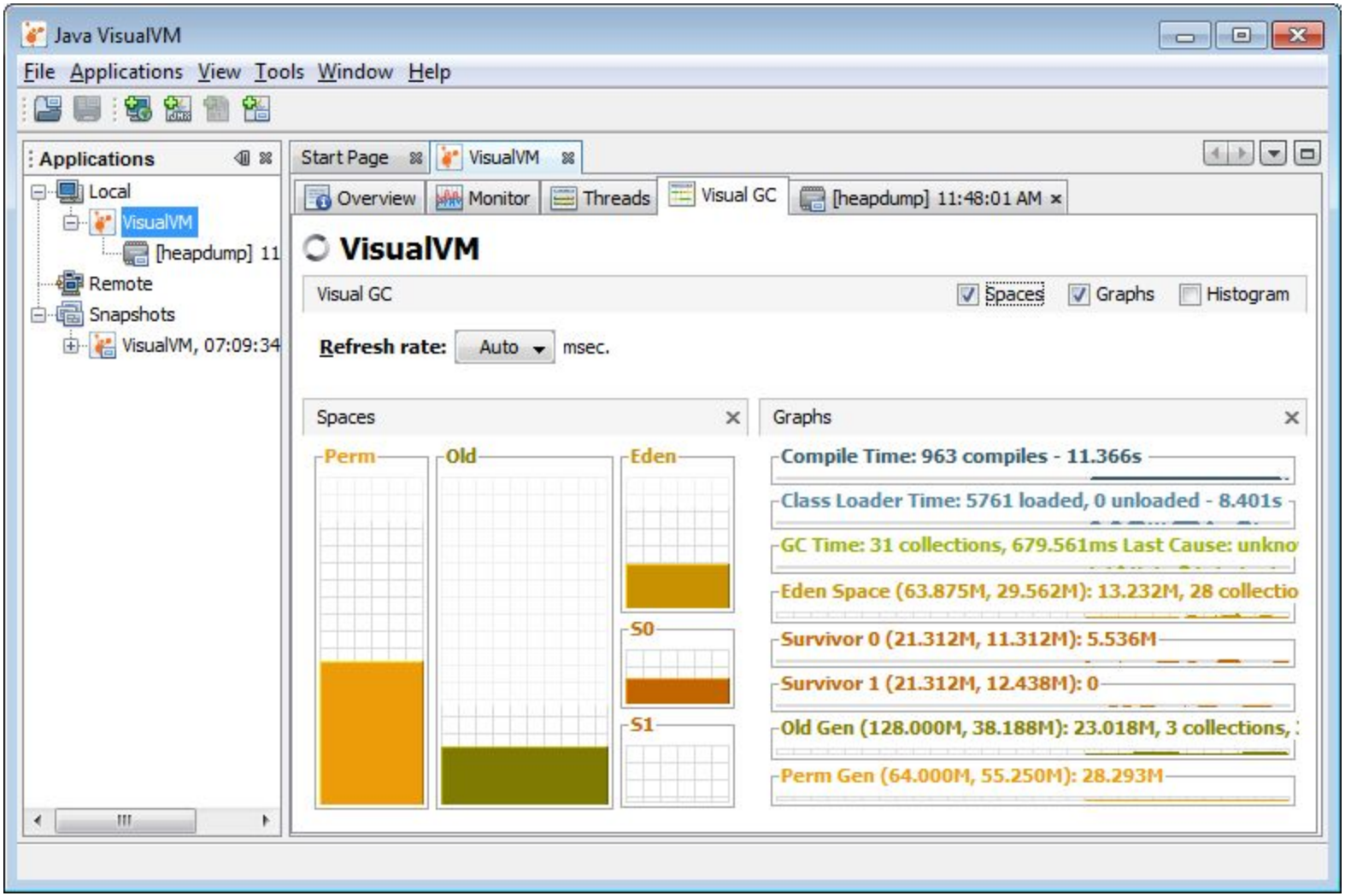


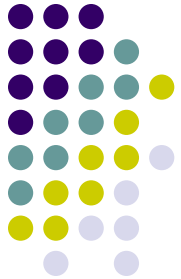
- JVisualVM
- VisualGC
  - може бути встановлений як plugin до JVisualVM
- JConsole

JVisualVM


Java VisualVM
File Applications View Tools Window Help
Applications
Local
VisualVM
[heapdump] 11
Remote
Snapshots
VisualVM, 07:09:34
Start Page
VisualVM
Overview
Monitor
Threads
Visual GC
[heapdump] 11:48:01 AM
VisualVM
Heap Dump
Summary
Classes
Instances
OQL Console
Classes
Class Name
Instances [%]
Instances
Size
java.lang.String 28928 (15.4%) 694272 (4.4%)
char[] 27789 (14.8%) 2063332 (13.1%)
java.util.HashMap$Entry 11928 (6.4%) 286272 (1.8%)
short[] 8571 (4.6%) 487958 (3.1%)
java.util.Hashtable$Entry 8504 (4.5%) 204096 (1.3%)
int[] 7606 (4.1%) 7524924 (47.9%)
byte[] 5900 (3.1%) 1166029 (7.4%)
java.lang.Object[] 5163 (2.8%) 253092 (1.6%)
java.lang.reflect.Method 4147 (2.2%) 319319 (2%)
java.lang.Class[] 4131 (2.2%) 47160 (0.3%)
[Class Name Filter]

JVisualVM


Java VisualVM
File Applications View Tools Window Help
Applications
Local
VisualVM
[heapdump] 11
Remote
Snapshots
VisualVM, 07:09:34
Start Page
VisualVM
Overview
Monitor
Threads
Visual GC
[heapdump] 11:48:01 AM
VisualVM
Visual GC
Spaces
Graphs
Histogram
Refresh rate:
Auto
msec.
Spaces
Graphs
Perm
Old
Eden
S0
S1
Compile Time: 963 compiles - 11.366s
Class Loader Time: 5761 loaded, 0 unloaded - 8.401s
GC Time: 31 collections, 679.561ms Last Cause: unkno
Eden Space (63.875M, 29.562M): 13.232M, 28 collectio
Survivor 0 (21.312M, 11.312M): 5.536M
Survivor 1 (21.312M, 12.438M): 0
Old Gen (128.000M, 38.188M): 23.018M, 3 collections,
Perm Gen (64.000M, 55.250M): 28.293M



БАЙТ-КОД

Байт-код

- Набір інструкцій:
  - http://en.wikipedia.org/wiki/Java_bytecode_instruction_listings

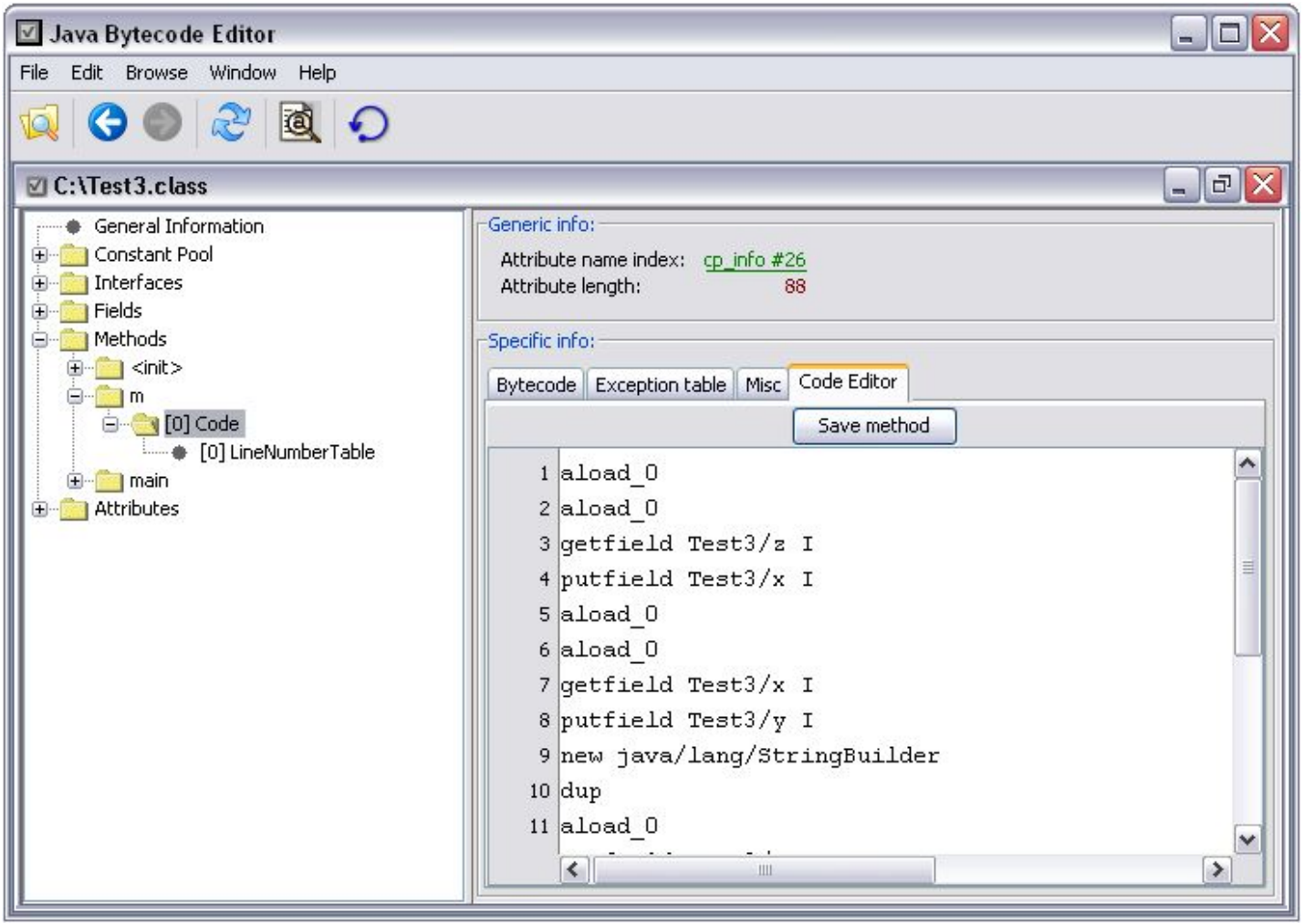
```
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

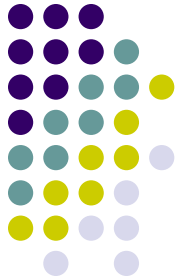
javac Hello.java
javap -c Hello

```
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
  Code:
   0:   aload_0
   1:   invokespecial    #1; //Method java/lang/Object."<init>":()V
   4:   return

public static void main(java.lang.String[]);
  Code:
   0:   getstatic    #2; //Field java/lang/System.out:Ljava/io/PrintStream;
   3:   ldc    #3; //String Hello World
   5:   invokevirtual      #4; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
   8:   return
}
```

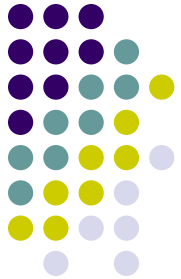
JBE - Java Bytecode Editor



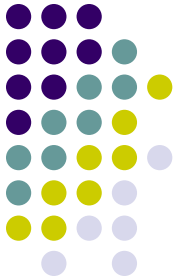


ВЛАСТИВОСТІ МОВИ JAVA

Властивості Java

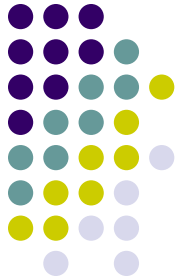


- Завдяки JVM:
  - Кросплатформеність
  - Простота управління пам’яттю
  - Підтримка багатопоточності
- Інші властивості:
  - Чиста об’єктно-орієнтована мова



HELLO WORLD

Hello World!

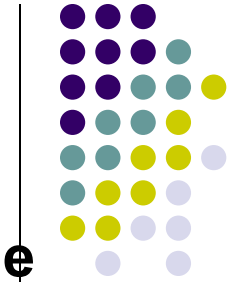


- HelloWorld.java

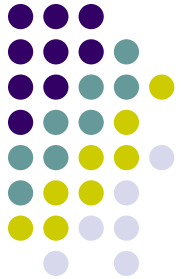
```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

- javac HelloWorld.java -> HelloWorld.class
- java HelloWorld

Ресурси



- James Gosling, Bill Joy, Guy Steele. **The Java Language Specification**. - Addison Wesley. - 3 edition. - 2005. - 688p. -http://java.sun.com/docs/books/jls/
- Tim Lindholm, Frank Yellin. The **Java Virtual Machine Specification**, Second Edition. - Prentice Hall. - 1999. – 496p. - http://java.sun.com/docs/books/jvms/
- **Java SE 6 Documentation** http://download.oracle.com/javase/6/docs/
- **Java Tutorials** http://download.oracle.com/javase/tutorial/index.html
- Bill Venners. **The Java Virtual Machine. Chapter 5 of Inside the Java Virtual Machine.** http://www.artima.com/insidejvm/ed2/jvm.html



- Запитання?