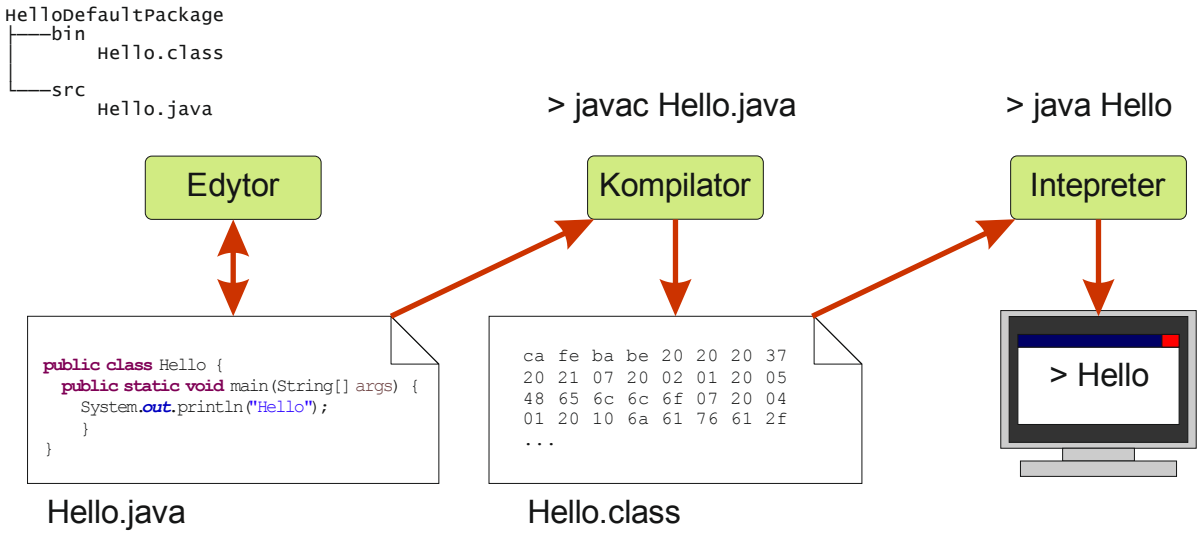
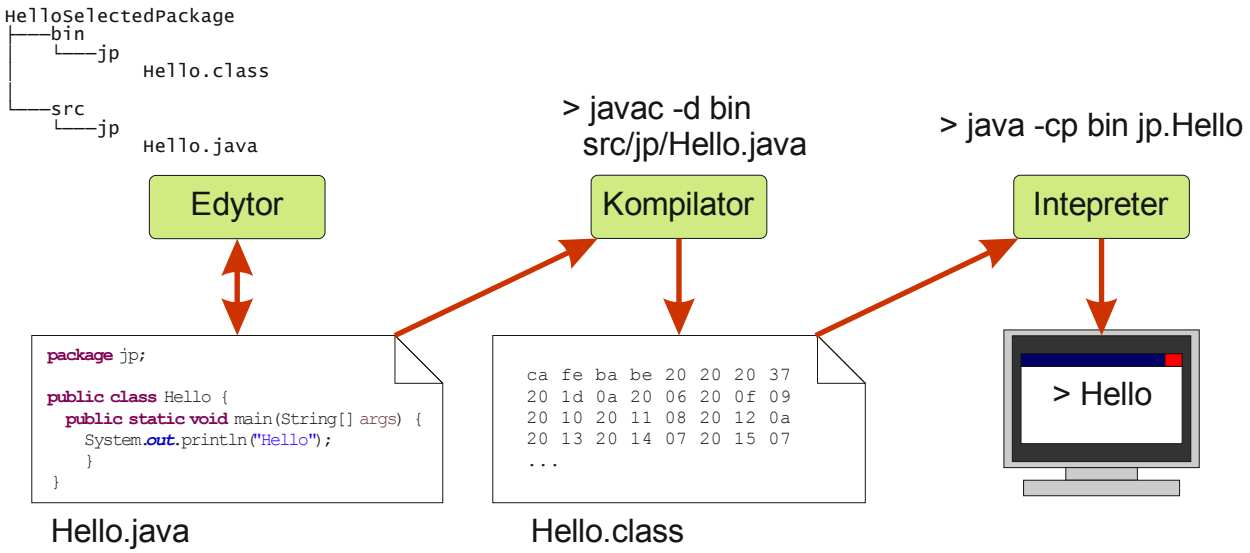
Języki Programowania

dr inż. Tomasz Kubik
tomasz.kubik.staff.iiar.pwr.edu.pl

Wytwarzanie oprogramowania w języku JAVA - podejście standardowe (klasa w pakiecie domyślnym)



Wytwarzanie oprogramowania w języku JAVA
- podejście standardowe (klasa w pakiecie zdefiniowanym)

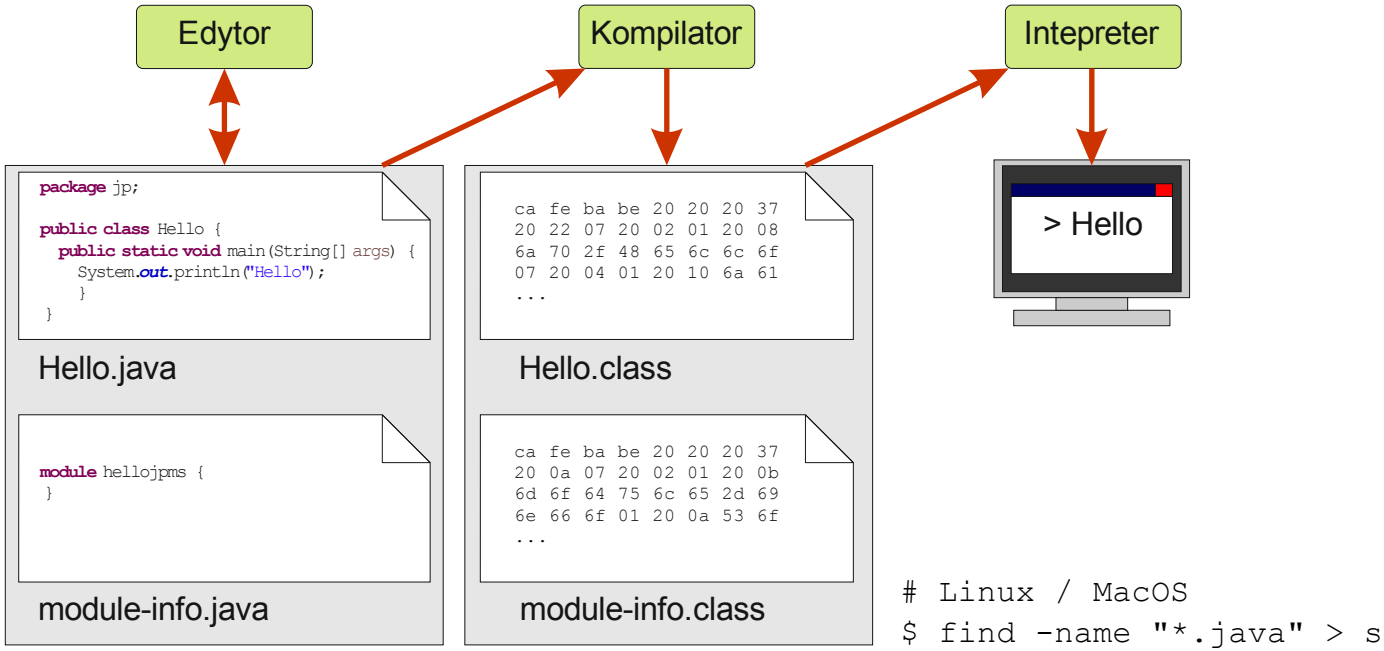


Wytwarzanie oprogramowania w języku JAVA - podejście z jednym, domyślnym modułem

```
HelloSingleJPMS:
├──bin
│   module-info.class
│   └──jp
│       Hello.class
└──src
    module-info.java
    └──jp
        Hello.java
```

> javac -d bin
> src\jp\Hello.java
> src\module-info.java

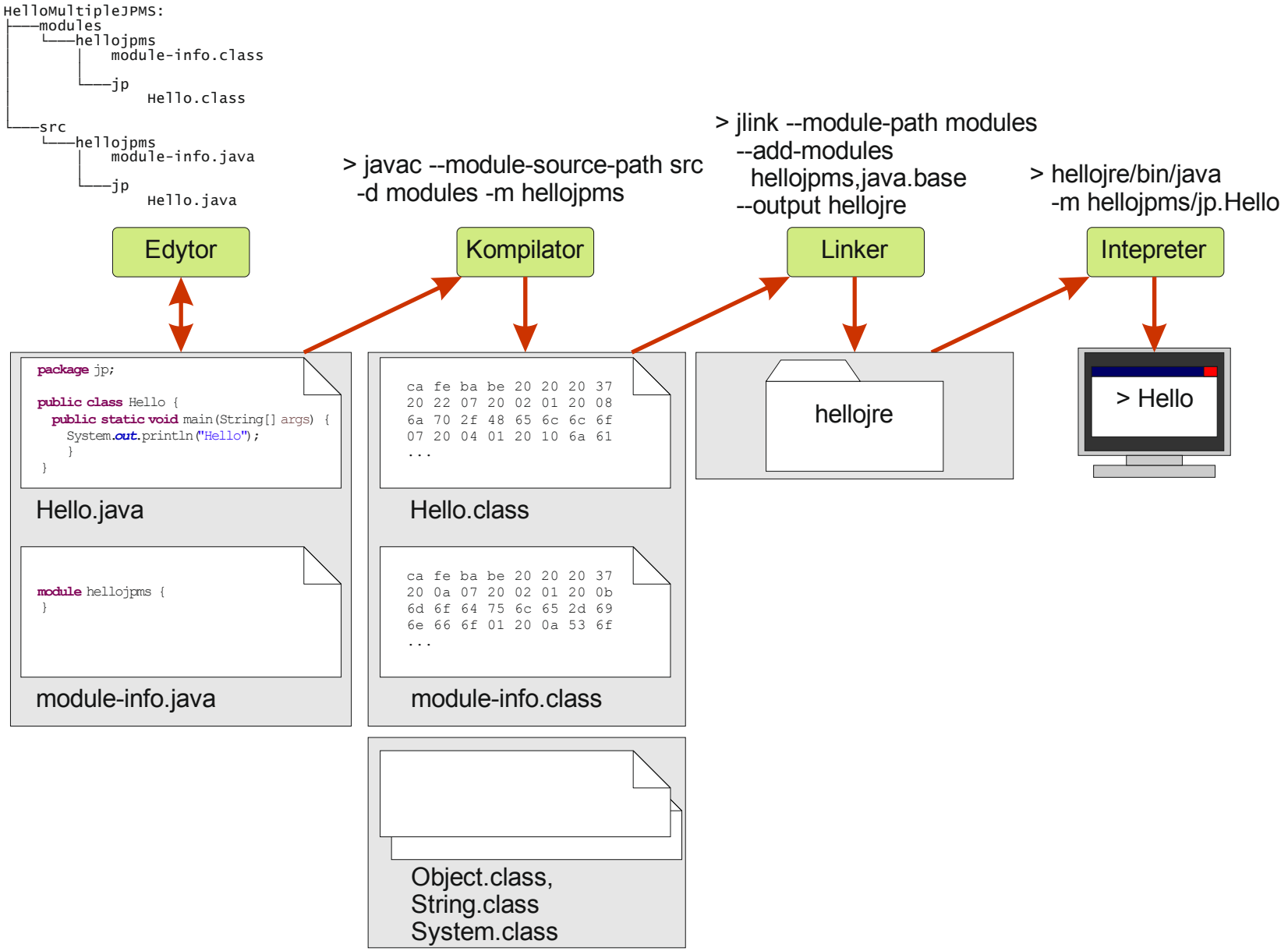
> java --module-path bin
> --module hellodefault/jp.Hello




```
# Linux / MacOS
$ find -name "*.java" > sources.txt
$ javac @sources.txt

:: Windows
> dir /s /B *.java > sources.txt
> javac @sources.txt
```

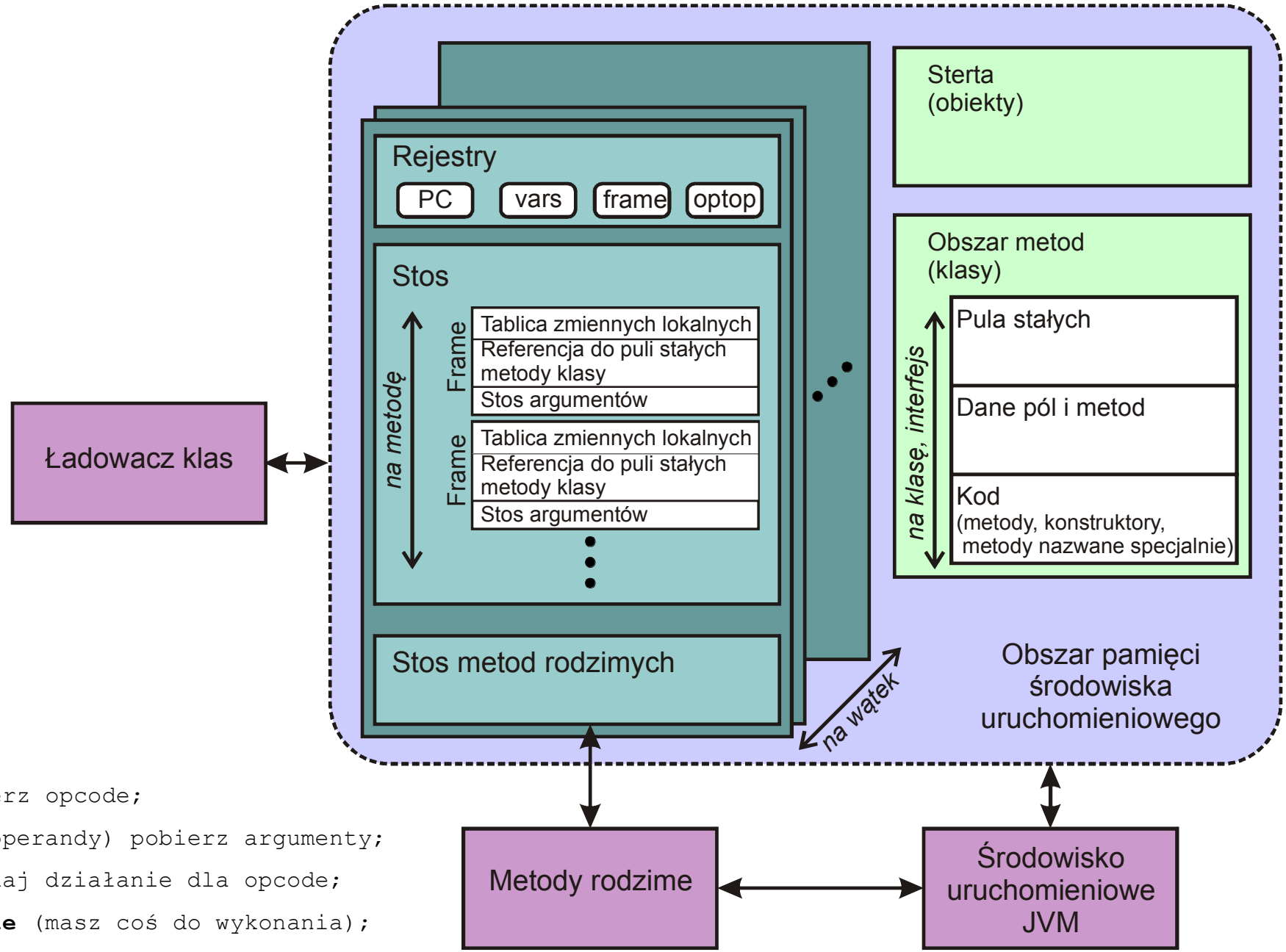
Wytwarzanie oprogramowania w języku JAVA - podejście bazujące na wykorzystaniu modułów



Przydatne linki

- Tworzenie i kompilacja projektów modułowych
  - https://www.logicbig.com/tutorials/core-java-tutorial/modules/modes.html
  - https://www.baeldung.com/java-9-modularity
  - https://www.developer.com/java/ent/using-a-java-9-module-as-a-jar-file.html
  - https://www.geeksforgeeks.org/jlink-java-linker/
- Specyfikacja Java
  - https://docs.oracle.com/javase/specs/
- Specyfikacja linkera
  - https://openjdk.java.net/jeps/282

Wirtualna maszyna JAVA




```
do {
 pobierz opcode;
 if (operandy) pobierz argumenty;
 wykonaj działanie dla opcode;
} while (masz coś do wykonania);
```

Instrukcje kodu bajtowego

- 212 instrukcje
  - opcode (8bit) + 0 lub więcej argumentów
- 44 instrukcje zarezerwowane
  - przyszłe rozszerzenia lub pośrednia optymalizacja JVM
- mnemoniki
  - a… : manipulacja referencjami (Class, Interface, Array)
  - s… : operacje dla typu short
  - i… : operacje dla typu integer (boolean)
  - l… : operacje dla typu long,
  - b… : operacje dla typu byte,
  - c… : operacje dla typu char,
  - f… : operacje dla typu float
  - d… : operacje dla typu double

```
// Bytecode stream:
// 03 3b 84 00 01 1a 05 68 3b a7 ff f9
// Disassembly:
iconst_0      // 03
istore_0      // 3b
iinc 0, 1     // 84 00 01
iload_0       // 1a
iconst_2      // 05
imul          // 68
istore_0      // 3b
goto -7       // a7 ff f9
```

Instrukcje kodu bajtowego

- instrukcje w podziale na manipulowane nimi elementy architektury JVM
  - Stos: iconst, iload, bipush, istore, pop, dup
  - PC: goto, ifeq, ifgt, return, athrow
  - Sterta: new, newarray
  - Pola: getstatic, putstatic, getfield, putfield
  - Metody: invokestatic, invokevirtual
- instrukcje w podziale na realizowane funkcje
  - Przerzucanie: pop, swap, dup, ...
  - Obliczanie: iadd, isub, imul, idiv, ineg,...
  - Konwersja: d2i, i2b, d2f, i2z,...
  - Operacje na pamięci lokalnej: iload, istore,...
  - Operacje na tablicach: arraylength, newarray,...
  - Zarządzanie obiektami: get/putfield, invokevirtual, new
  - Operacje typu push: aconst_null, iconst_m1,....
  - Strumień sterowania: nop, goto, jsr, ret, tableswitch,...
  - Wielowątkowość: monitorenter, monitorexit,...

Czytelny kod bajtowy klasy

```
// Compiled from Hello.java (version 1.6 : 50.0, super bit)
public class Hello {

  // Method descriptor #6 ()V
  // Stack: 1, Locals: 1
  public Hello();
    0  aload_0 [this]
    1  invokespecial java.lang.Object() [8]
    4  return
      Line numbers:
        [pc: 0, line: 2]
      Local variable table:
        [pc: 0, pc: 5] local: this index: 0 type: Hello

  // Method descriptor #15 ([Ljava/lang/String;)V
  // Stack: 2, Locals: 1
  public static void main(java.lang.String[] args);
    0  getstatic java.lang.System.out : java.io.PrintStream [16]
    3  ldc <String "Hello"> [22]
    5  invokevirtual java.io.PrintStream.println(java.lang.String) : void [24]
    8  return
      Line numbers:
        [pc: 0, line: 8]
        [pc: 8, line: 9]
      Local variable table:
        [pc: 0, pc: 9] local: args index: 0 type: java.lang.String[]
}
```

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

javac Hello.java
javap –c Hello

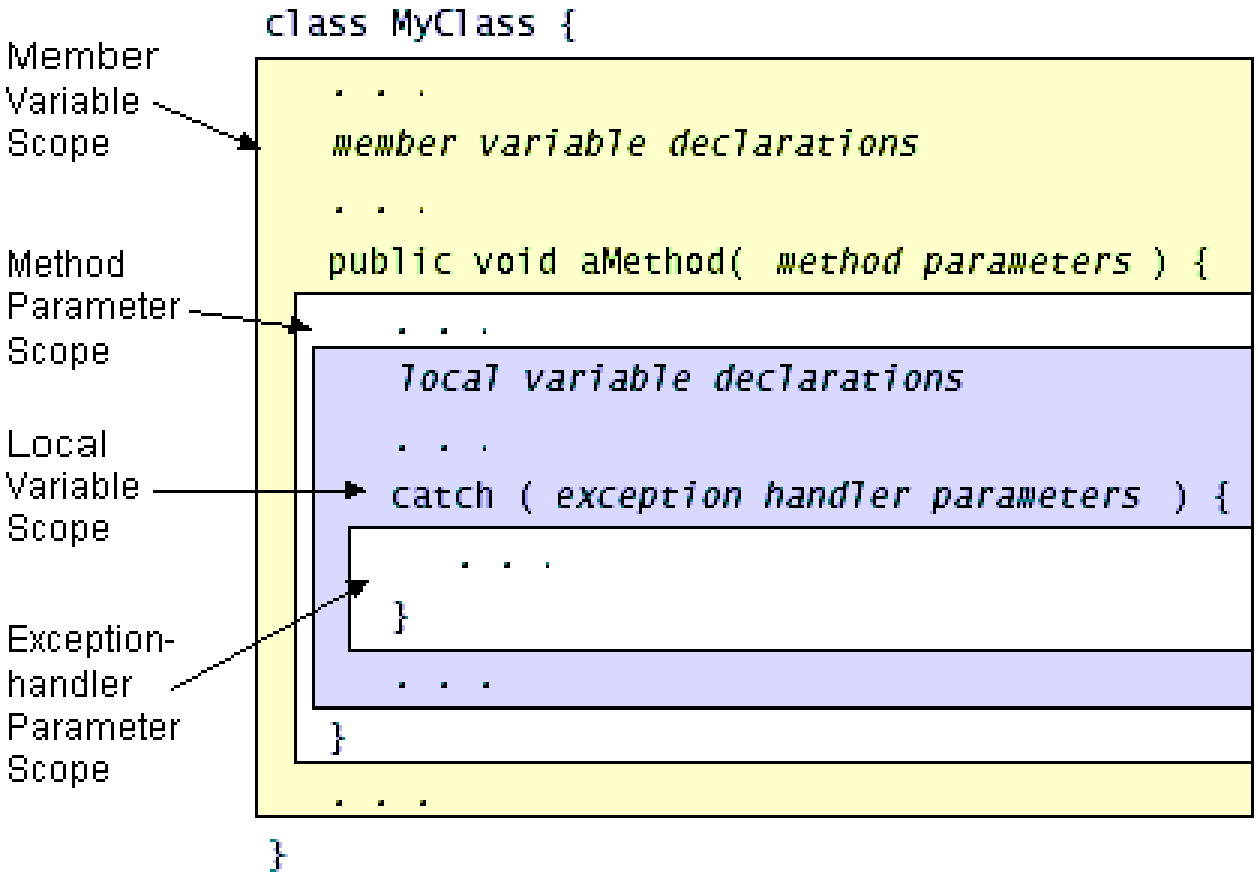
Języki kompilowane do kodu bajtowego JVM

- PHP , z Quercus
- Clojure, dialekt Lisp
- Groovy, język skryptowy
- JavaFX Script, język skryptowy (Rich Internet application)
- JRuby, implementacja Ruby
- Jython, implementacja Python
- Rhino, implementacja JavaScript
- Scala, język programowania obiektowego i funkcjonalnego
- istnieją kompilatory Ada oraz COBOL.

Deklaracja zmiennych

- Deklaracja pojedynczej zmiennej
  - *typ nazwazmiennej,* np. **int i;**
- Deklaracja wielu zmiennych
  - *typ nazwamiennej, nazwazmiennej,* np. **int i, j;**
- Nazwa zmiennej
  - jest ciągiem znaków Unicode o dowolnej długości zaczynającym się literą,
  - nie pokrywa się ze słowami kluczowymi,
  - nie powtarza się w zasięgu obszaru, w którym została zdefiniowana,
  - konwencja: nazwy zmiennych zaczynają się małą literą, nazwy klas – dużą, np. **int toJestZmienna**, **class ToJestKlasa**.

Zasięg zmiennych



- Dostęp:
  - **public**,
  - **private**,
  - **protected**
- Zasięg:
  - parametr klasy,
  - zmienna lokalna,
  - parametr metody,
  - obsługa wyjątku

Typy podstawowe

- Całkowite:
  - [**byte** | **short** | **int** | **long**];
- Zmiennoprzecinkowe:
  - [**float** | **double**]
- Inne:
  - [**char** | **boolean**]

Zmienne finalne

- Modyfikowane tylko raz

```
final int aFinalVar=0;
final int aBlankFinal; // deklaracja
aBlankFinal=0; // modyfikacja
```

Łańcuchy znaków

- String
- StringBuffer.append

Zmienne wyliczeniowe

- Są testowane na zgodność typów w czasie kompilacji
- Istnieją we własnej przestrzeni nazw
- Zbiór stałych musi być stały przez cały czas
- Można dokonywać wyboru ze względu na stałą wyliczeniową (switch)
- Posiadają metodę o wartościach statycznych, która zwraca tablicę zawierającą wszystkie wartości typu enum w porządku ich zadeklarowania. Metoda ta zazwyczaj jest używana w kombinacji z konstrukcją for-each aby iterować poprzez wartości typu wyliczeniowego
- Można dostarczyć metody oraz pola, implementację interfejsu, itd.
- Dostarczają implementacji wszystkich metod klasy Object. Są Comparable oraz Serializable, oraz forma serial jest zaprojektowana tak, aby przeciwstawiać się zmianom w typie enum.
- Uwagi:
  - konstruktor jest prywatny, inny typ konstruktora będzie odrzucany przez kompilator;
  - chociaż typ enum jest klasą, typy enum nie mogą dziedziczyć po sobie
- w java.util są specjalne implementacje klas korzystających z typu enum: EnumSet, EnumMap.

```
enum Day{ SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
    SATURDAY };
for (Day d : EnumSet.range(Day.MONDAY, Day.FRIDAY))
      System.out.println(d);
```

Typ wyliczeniowy

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7),
    PLUTO   (1.27e+22,  1.137e6);
    private final double mass;   //in kilograms
    private final double radius; //in meters
    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }
    public double mass()   { return mass; }
    public double radius() { return radius; }
    public static final double G = 6.67300E-11; //universal gravitational constant  (m3 kg-1 s-2)
    public double surfaceGravity() {
        return G * mass / (radius * radius);
    }
    public double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
   }
}
```