Predavanje 4.

# OBJEKTNO-ORIJENTISANO PROGRAMIRANJE

2024/25.

# Agenda
## za danas

Kompatibilnost tipova

Poziv prepisanih metoda i polimorfizam

Poziv prekrivenih metoda

Statičko i dinamičko vezivanje

# Kompatibilnost tipova

AGENDA

Kompatibilnost tipova

Poziv prepisanih metoda i polimorfizam

Poziv prekrivenih metoda

Statičko i dinamičko vezivanje

# Kompatibilnost tipova

- Java je strogo tipiziran jezik

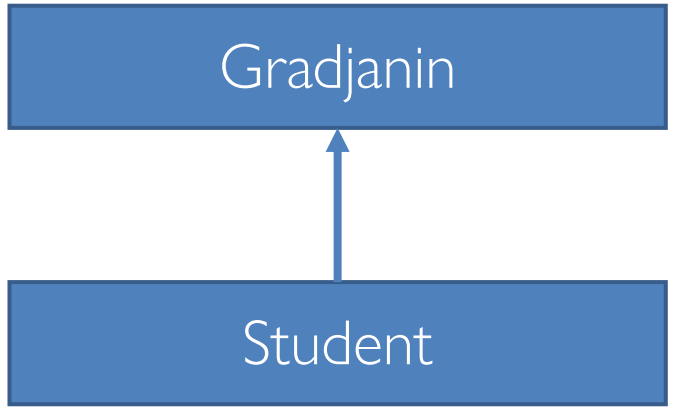
  kompatibilnost tipova se proverava tokom prevodjenja

- Pri operaciji dodele, inicijalizaciji, prenosu parametara i vraćanju rezultata metoda tip izraza mora biti kompatibilan tipu promenljive kojoj se izraz dodeljuje.
- Tip reference koja je rezultat izraza mora biti
  - Tip promenljive kojoj se dodeljuje
  - Njegov podtip
  - ili `null`

# Konverzija

- Nadtipovi su manje posebni, smatraju se širim i nalaze se više u hijerahiji tipova
- Konverzija proširivanja (naviše, nagore)
  - Podtip se konvertuje u nadtip
  - Automatski
- Konverzija sužavanja (naniže, nadole)
  - Nadtip se konvertuje u podtip
  - Eksplicitno (cast)

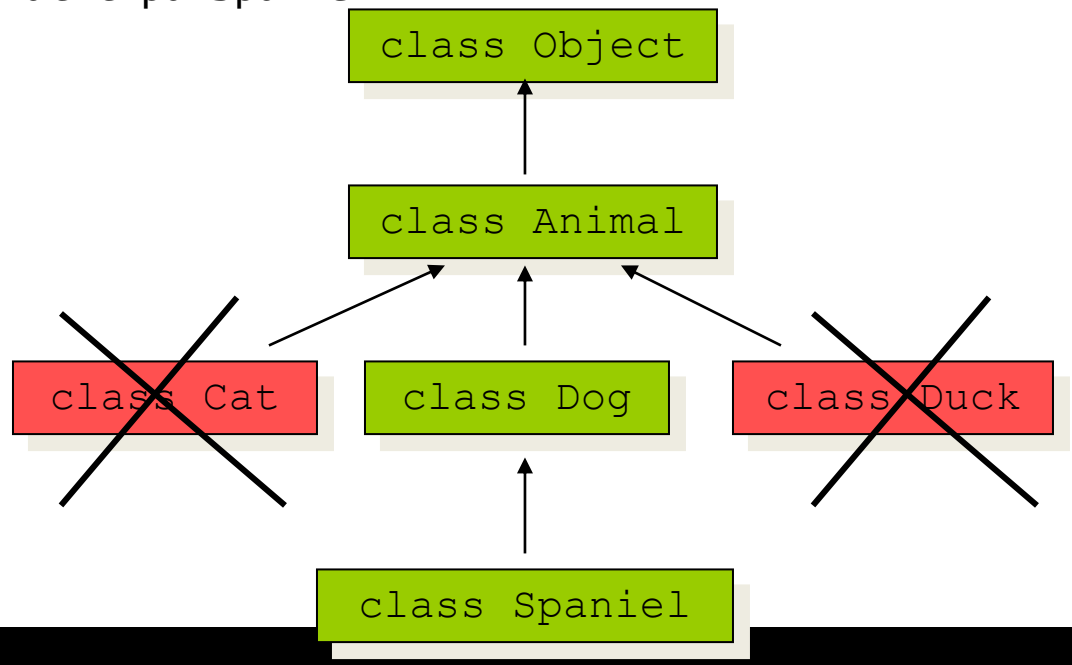
```
class Student extends Gradjanin { … }
…
Sudent s = new Sudent();
Gradjanin g = new Gradjanin();
Gradjanin g2 = s;       // naviše
Sudent s1 = (Sudent) g;  // naniže, obavezan cast
```



# Konverzija sužavanja - CAST

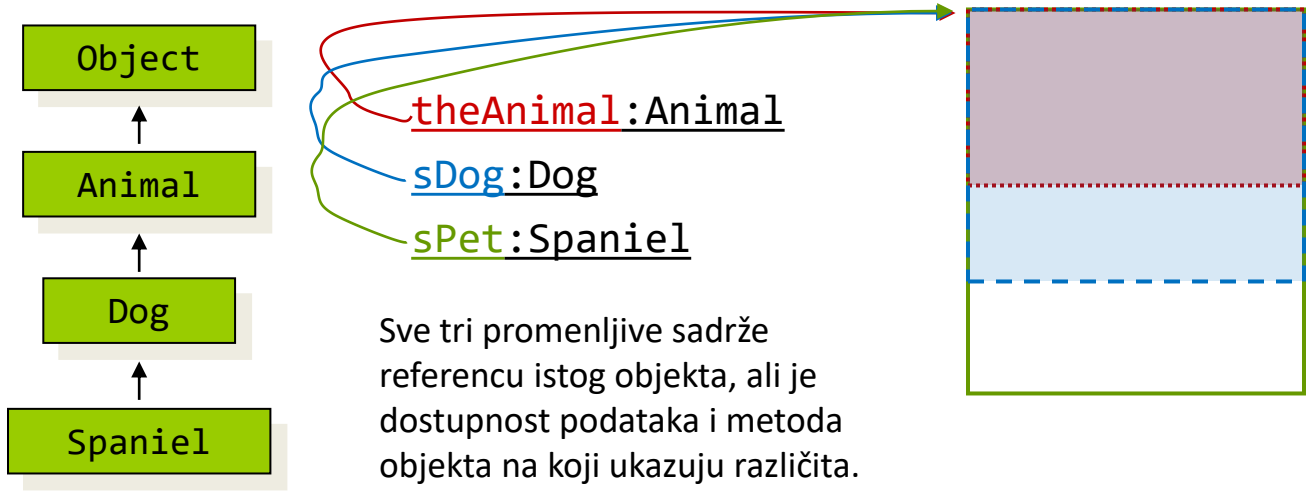
- Cast objekata je moguć u slučaju da radimo cast izmedju klasa koje pripadaju istom lancu nasledjivanja (dakle, jedan tip mora da bude podtip drugog u bilo kom redosledu).
- Java zadržava sve informacije o originalnoj klasi kojoj objekat pripada !!!

```
Spaniel aPet = new Spaniel("Fang"); // Kreiraj objekat tipa Spaniel
Animal theAnimal = (Animal)aPet;
Animal theAnimal = aPet;
   // cast nagore - upward cast
Dog aDog = (Dog)theAnimal;
  // cast nadole
  // downward cast
```



# Konverzija sužavanja - CAST

```
Spaniel aPet = new Spaniel("Fang");
Animal theAnimal = aPet;
Dog aDog = (Dog)theAnimal;
```




- **aDog** se može koristiti SAMO za poziv *overrided* metoda iz klase **Spaniel**. Dakle, ako **Spaniel** ima i dodatne metode, oni neće biti dostupni sa **aDog** refence.
- ako je potrebno pozvati metod specifičan za Spaniel klasu koristeći referencu **aDog,** NEOPHODAN je cast **aDog** na **Spaniel**.

```
((Spaniel)aDog).specmetod();
```

# Poziv prepisanih metoda i polimorfizam

AGENDA

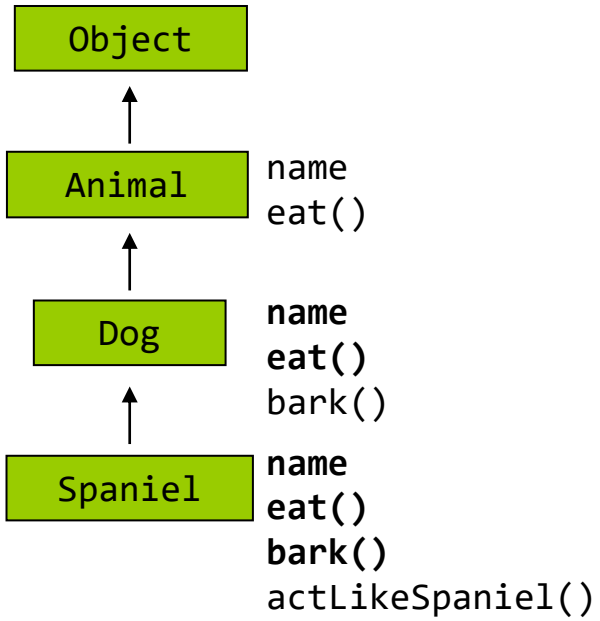
Kompatibilnost tipova

Poziv prepisanih metoda i polimorfizam

Poziv prekrivenih metoda

Statičko i dinamičko vezivanje

# poziv prepisanih metoda

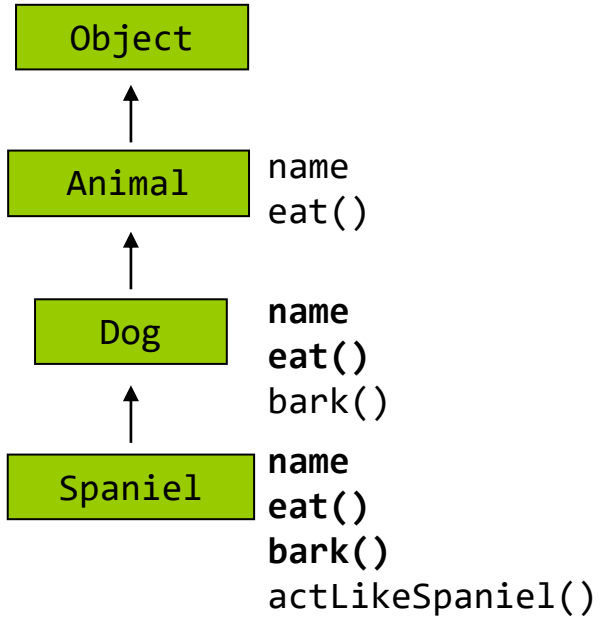



```
Spaniel aPet = new Spaniel("Fang");
Animal theAnimal = aPet;
Dog aDog = (Dog)theAnimal;

// Koji metod će biti pozvan u sledećoj liniji koda?
   Metod definisan u Animal ili Spaniel.
theAnimal.eat();

// Koji metod će biti pozvan u sledećoj liniji koda?
   Metod definisan u Animal, Dog ili Spaniel.
aDog.eat();
```

# POZIV PREPISANIH METODA

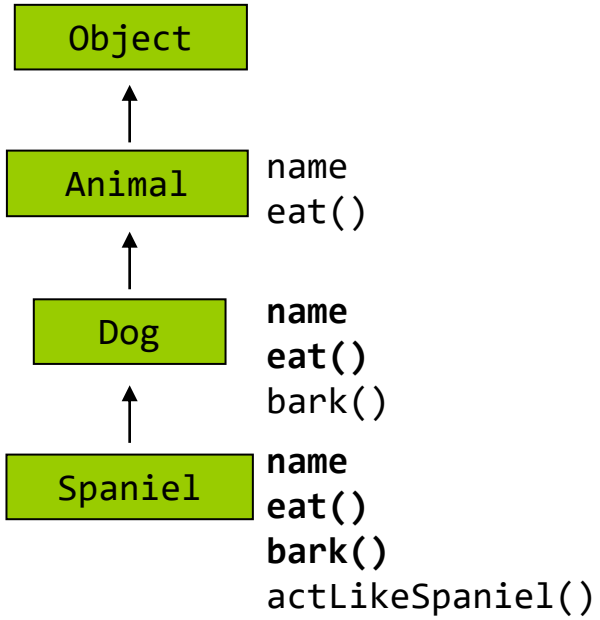



```
Spaniel aPet = new Spaniel("Fang");
Animal theAnimal = aPet;
Dog aDog = (Dog)theAnimal;

// Koji metod će biti pozvan u sledećoj liniji koda?
  Metod definisan u Animal ili Spaniel.
theAnimal.eat(); // Spaniel

// Koji metod će biti pozvan u sledećoj liniji koda?
  Metod definisan u Animal, Dog ili Spaniel.
aDog.eat(); // Spaniel
```

# POZIV PREPISANIH METODA




Slanjem poruke

```
ref.eat();
```

biće pozvan metod objekta na koji ukazuje referenca `ref`.

# POLIMORFIZAM

- mogućnost da varijablom određenog tipa referenciramo objekte različitih tipova i da automatski pozivamo metode koje su specifične za tip objekta na koji varijabla referencira
- uslovi
  - Poziv metoda podklase kroz varijablu bazne klase
  - Pozvana metoda mora biti i član bazne klase
  - Signatura metode i povratni tip moraju biti isti i u baznoj i u izvedenoj klasi
  - Atribut pristupa ne sme biti restriktivniji u izvedenoj klasi nego što je u baznoj klasi
  - Polimorfizam se odnosi samo na metode – reference baznog tipa mogu se koristiti samo za pristup podacima članovima baznog tipa

# Poziv prekrivenih metoda

AGENDA

- Kompatibilnost tipova
- Poziv prepisanih metoda i polimorfizam
- Poziv prekrivenih metoda
- Statičko i dinamičko vezivanje

# POZIV PREKRIVENIH (STATIČKIH) METODA

```
public class Animal {
    public static void testClassMethod() {
                        System.out.println("The class" + " method in Animal.");    }
    public void testInstanceMethod() {
                        System.out.println("The instance " + " method in Animal.");     }
}
public class Cat extends Animal {
    public static void testClassMethod() {
                        System.out.println("The class method" + " in Cat.");     }
    public void testInstanceMethod() {
                        System.out.println("The instance method" + " in Cat.");     }
}
public class Test {
    public static void main(String[] args) {
        Cat myCat = new Cat();
        Animal myAnimal = myCat;
        myCat.testClassMethod();                  // Šta će biti ispisano na izlazu?
        myAnimal.testClassMethod();               // Šta će biti ispisano na izlazu?
        myAnimal.testInstanceMethod();            // stampa: The instance method in Cat.
   }
}
```

# POZIV PREKRIVENIH (STATIČKIH) METODA

```
public class Animal {
    public static void testClassMethod() {
                        System.out.println("The class" + " method in Animal.");    }
    public void testInstanceMethod() {
                        System.out.println("The instance " + " method in Animal.");     }
}
public class Cat extends Animal {
    public static void testClassMethod() {
                        System.out.println("The class method" + " in Cat.");     }
    public void testInstanceMethod() {
                        System.out.println("The instance method" + " in Cat.");     }
}
public class Test {
    public static void main(String[] args) {
        Cat myCat = new Cat();
        Animal myAnimal = myCat;
        myCat.testClassMethod();                  // stampa: The class method in Cat.
        myAnimal.testClassMethod();               // stampa: The class method in Animal.
        myAnimal.testInstanceMethod();            // stampa: The instance method in Cat.
   }
}
```

# POZIV PREKRIVENIH (STATIČKIH) METODA

Dakle, polimorfizma NEMA kod statičkih metoda. Zašto?

| Natklasa<br>Potklasa | Metod objekta | static metod |
|---|---|---|
| Metod objekta | prepisuje | compile-time greška |
| static metod | compile-time greška | sakriva |

# Statičko i dinamičko vezivanje

AGENDA

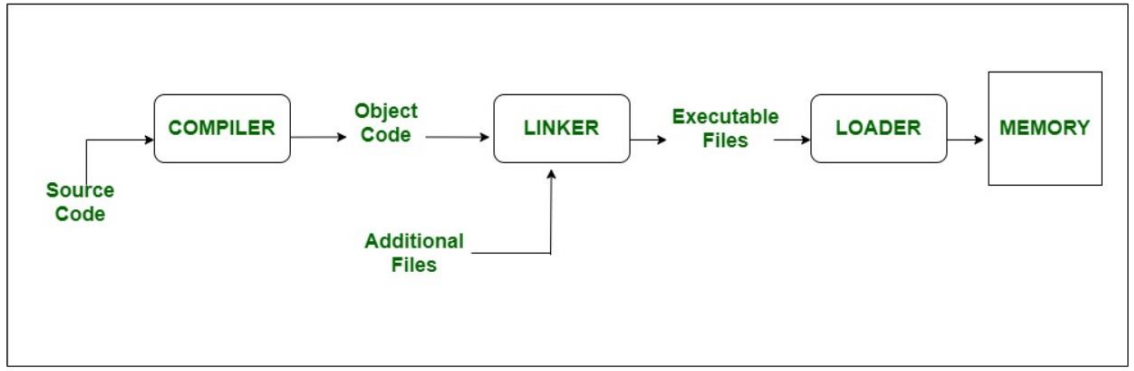
Kompatibilnost tipova

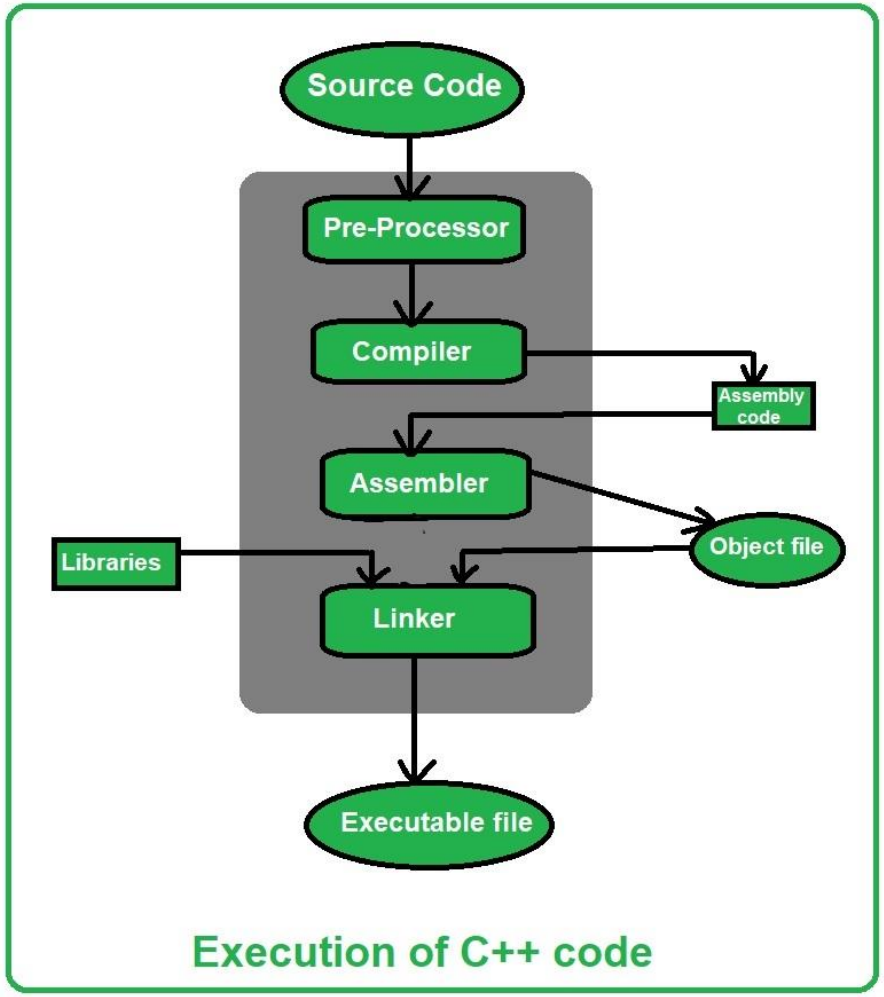
Poziv prepisanih metoda i polimorfizam

Poziv prekrivenih metoda

Statičko i dinamičko vezivanje

# C/C++ put od izvornog koda do procesa

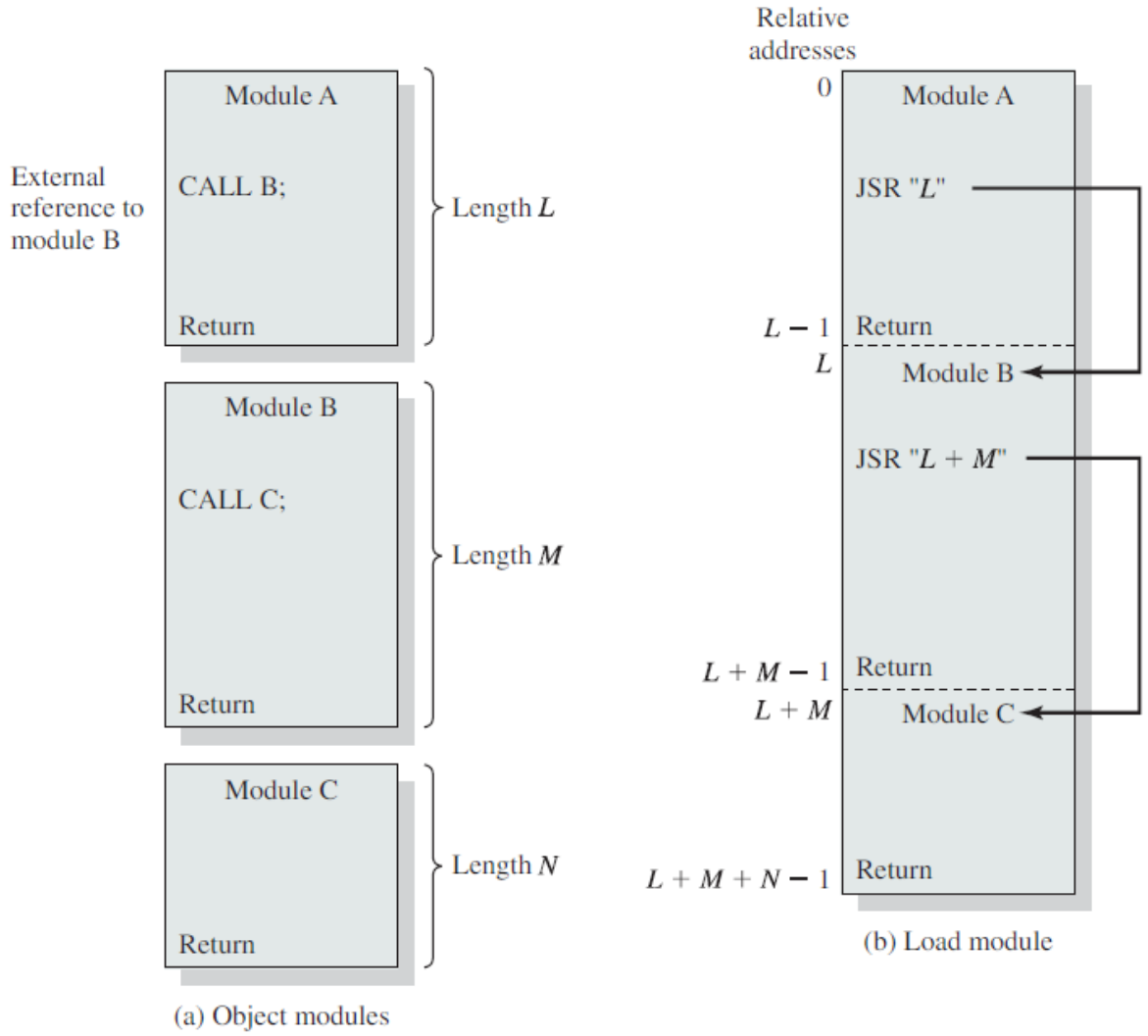


# LINKING - Vezivanje

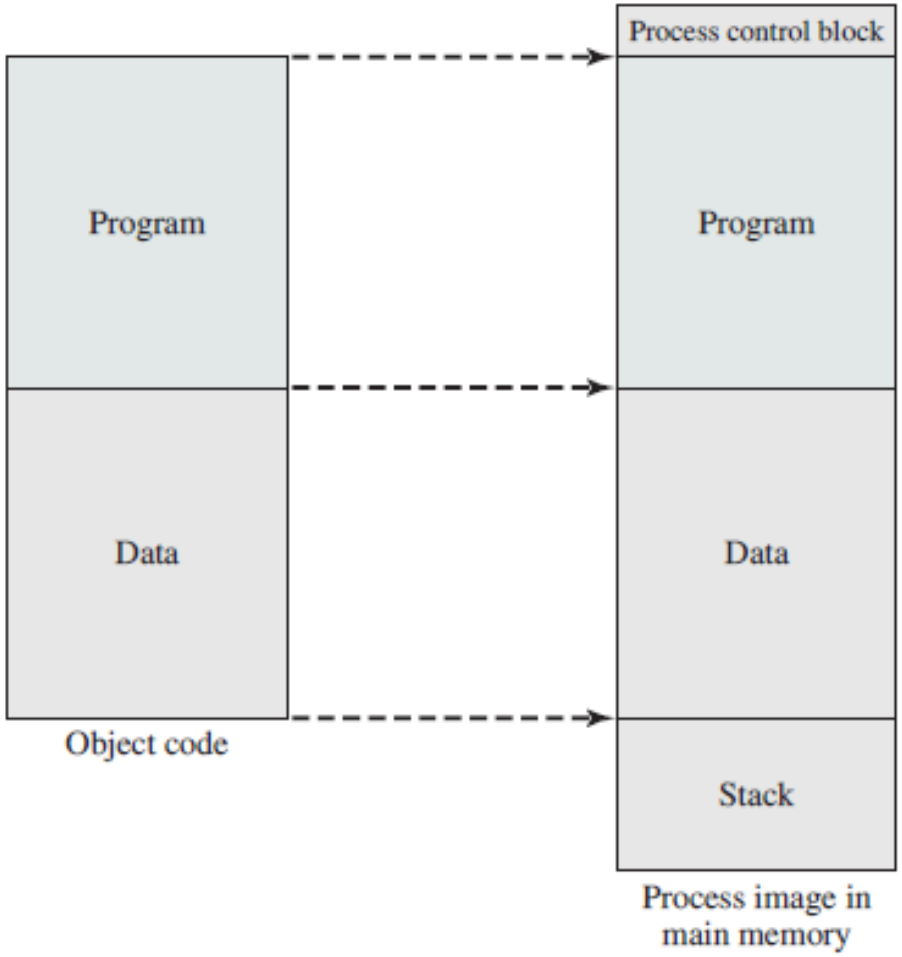
Aplikacija sadrži jedan ili više objektnih modula. Moduli se međusobno referenciraju/pozivaju. Treba razrešiti pozive.

Uloga linkera da je od kolekcije objektnih modula napravi Load modul i preda ga loaderu na učitavanje.

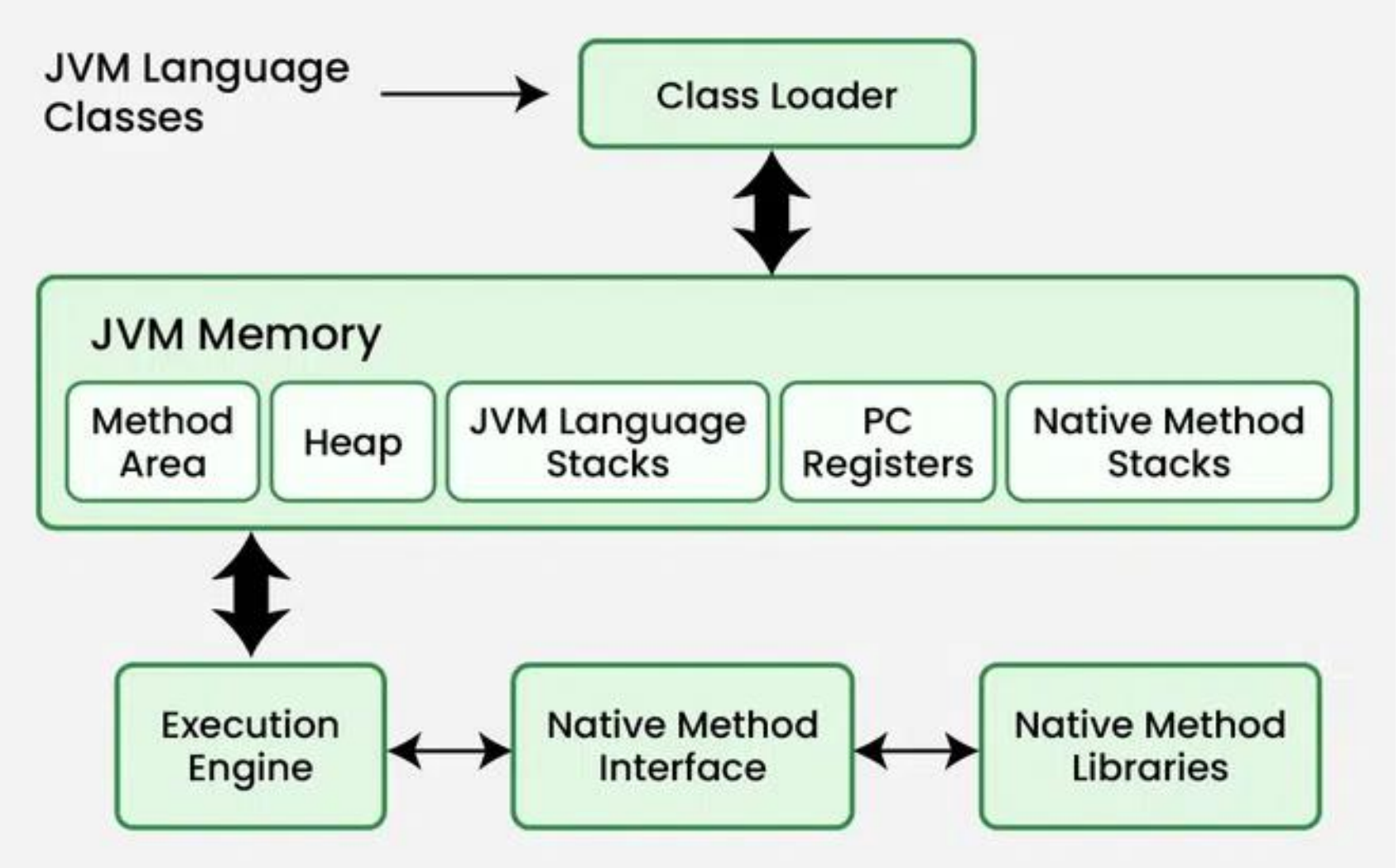


# LOADING - učitavanje

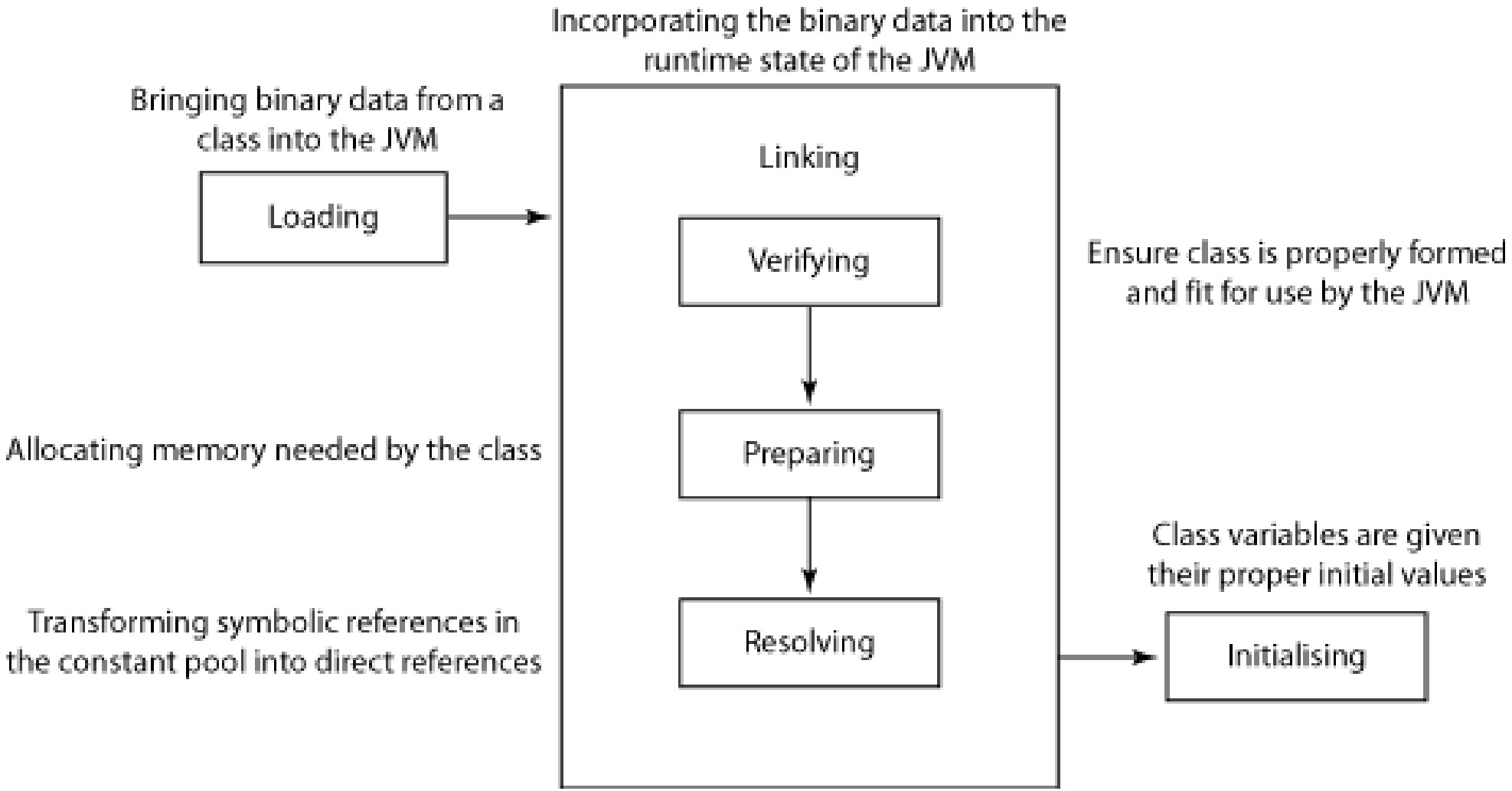
Prvi korak u krereiranju aktivnog procesa jeste njegovo **učitavanje** u glavnu memoriju i kreiranje slike procesa.



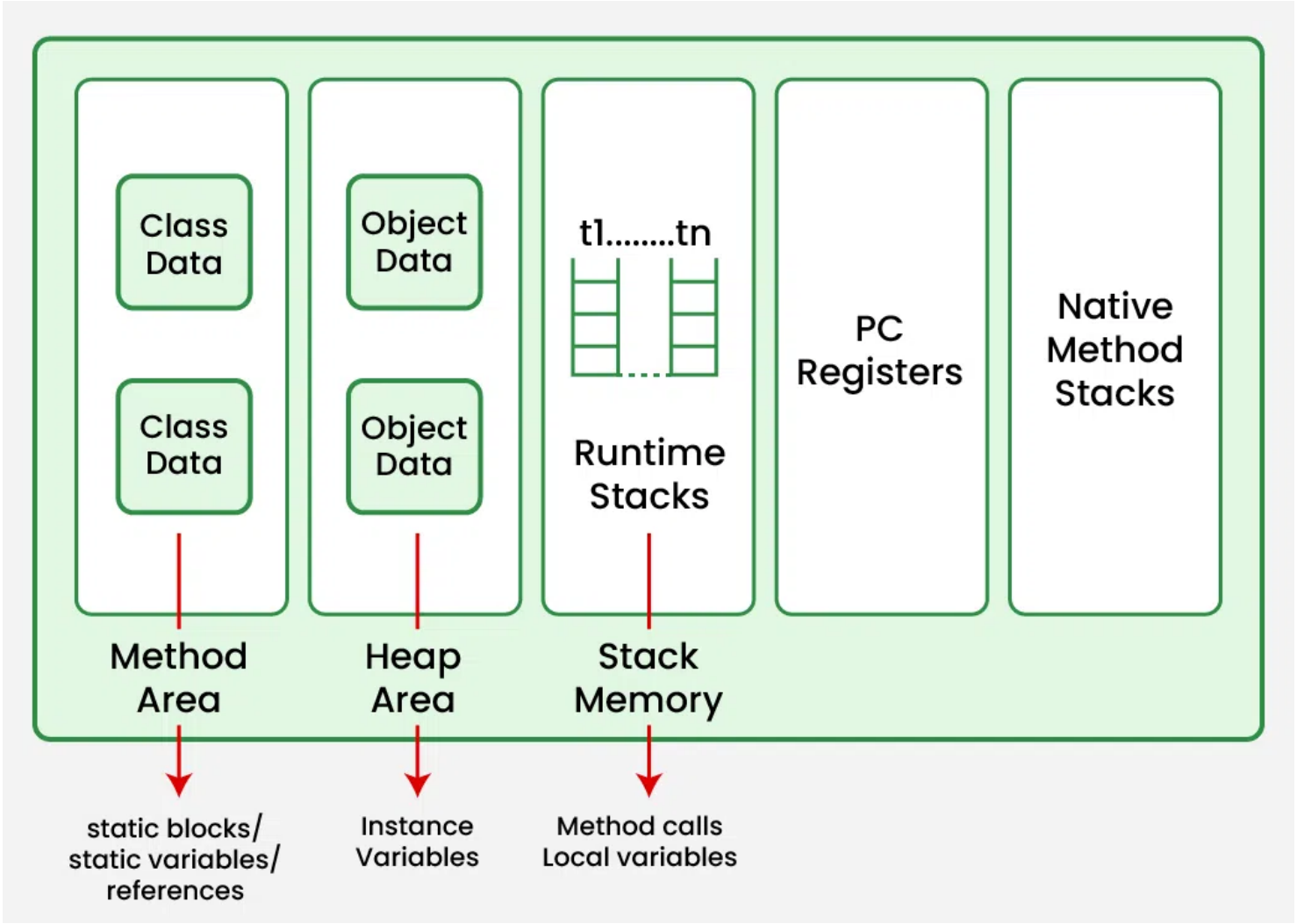
# Java


JVM Language Classes
Class Loader
JVM Memory
Method Area
Heap
JVM Language Stacks
PC Registers
Native Method Stacks
Execution Engine
Native Method Interface
Native Method Libraries

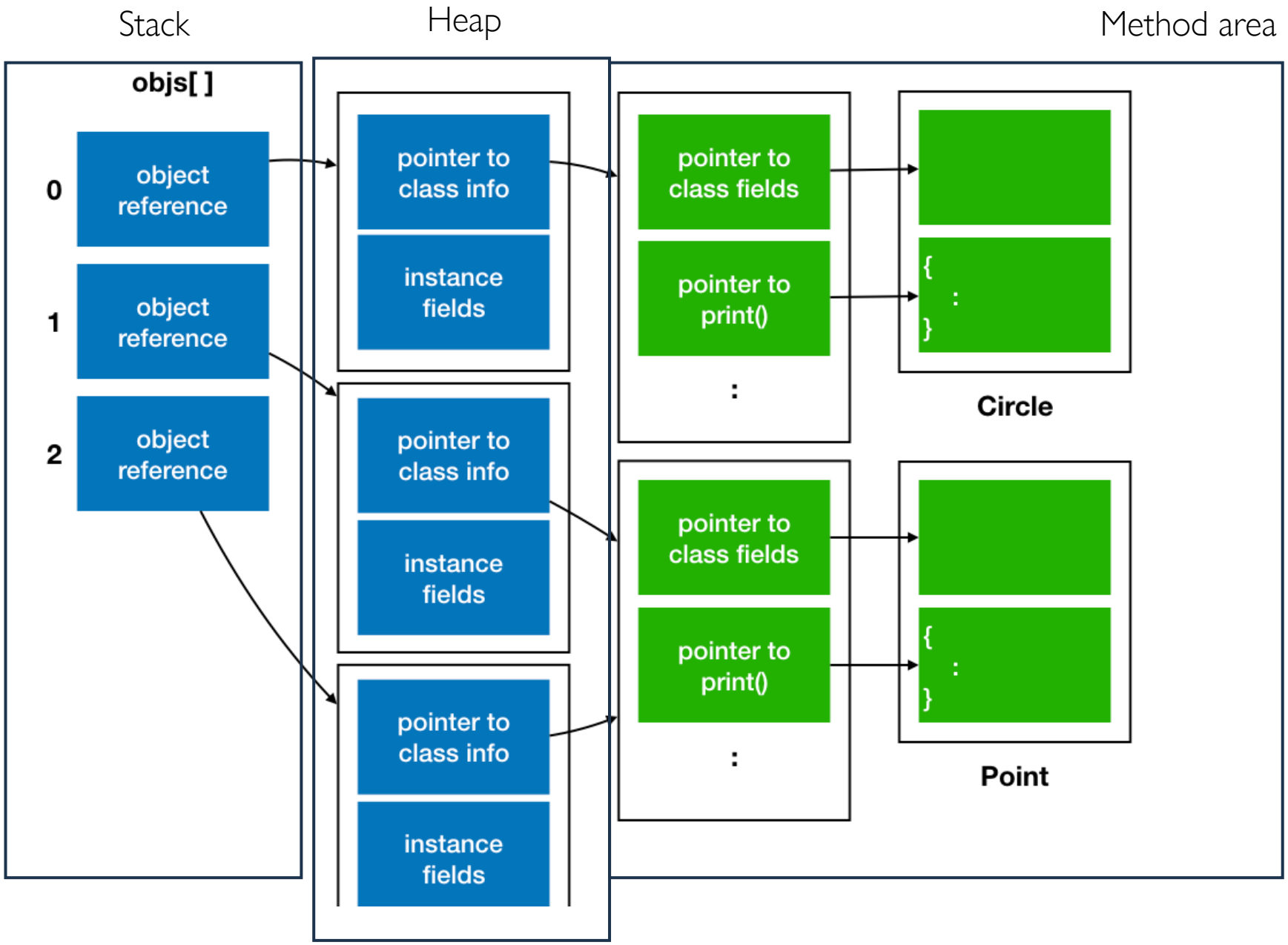
# Java Class Loader uloga



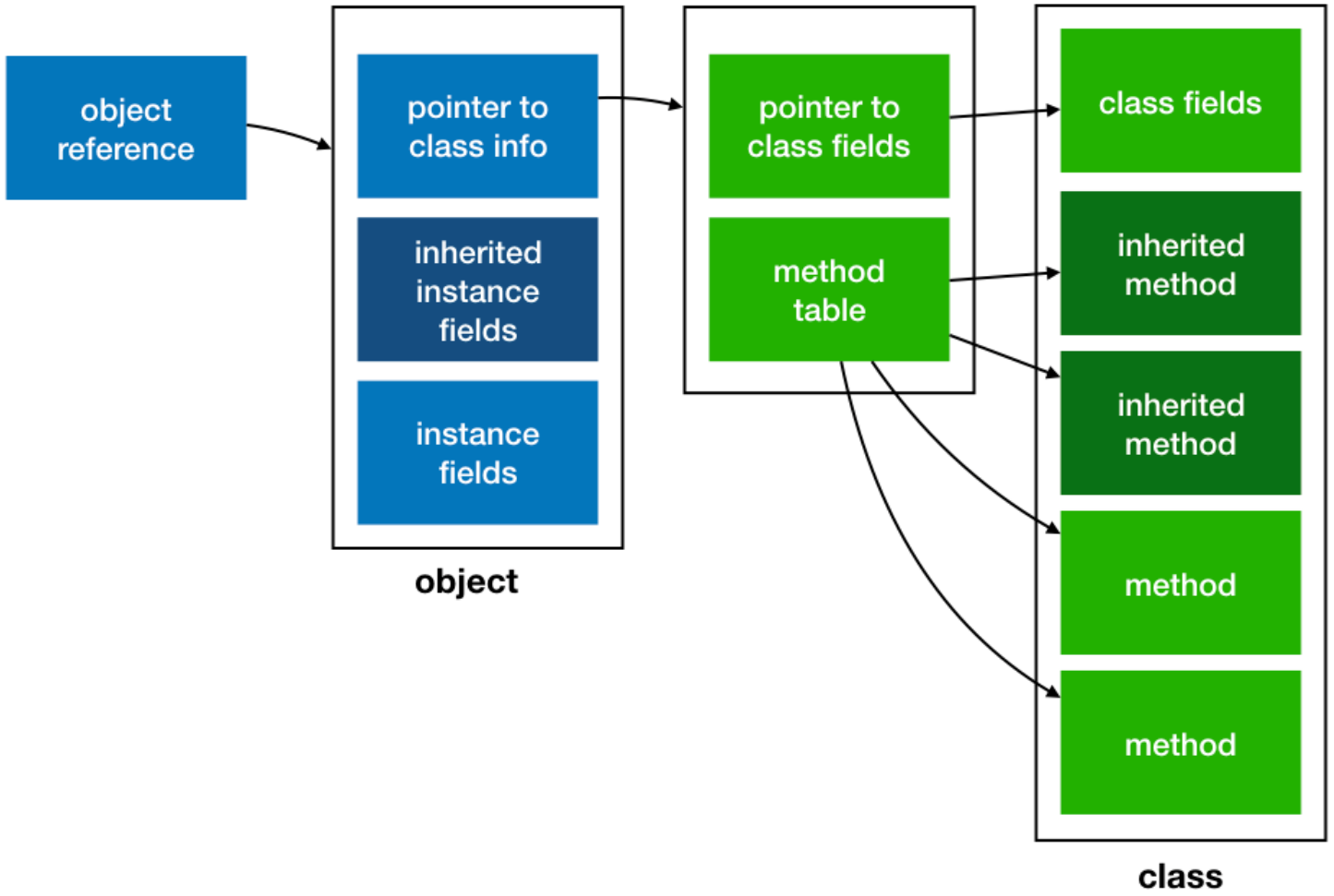
# Java Class Loader uloga



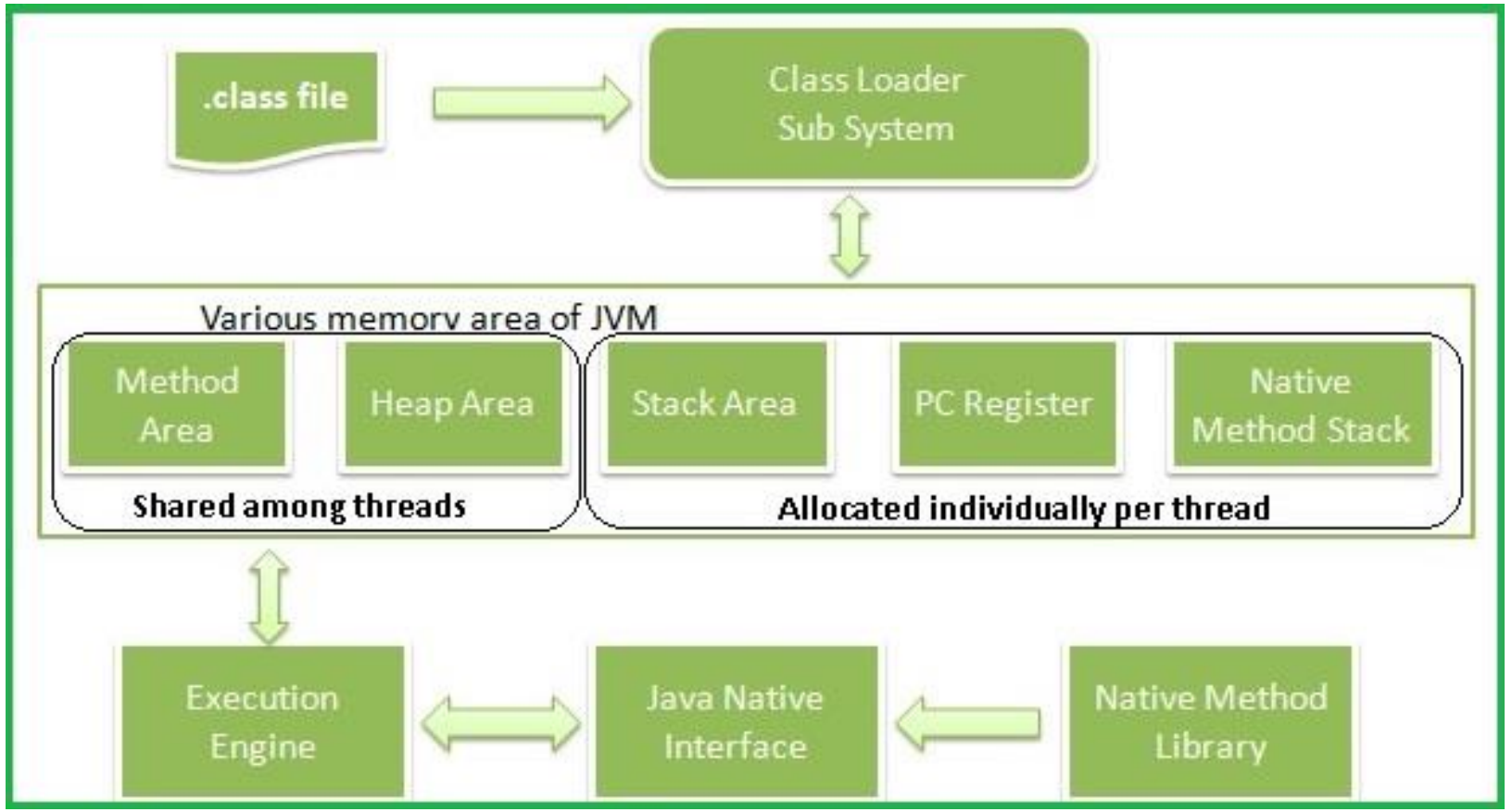
# Java


Stack
Heap
Method area
objs[ ]
0
object reference
1
object reference
2
object reference
pointer to class info
instance fields
pointer to class info
instance fields
pointer to class info
instance fields
pointer to class fields
pointer to print()
:
{
:
}
Circle
pointer to class fields
pointer to print()
:
{
:
}
Point

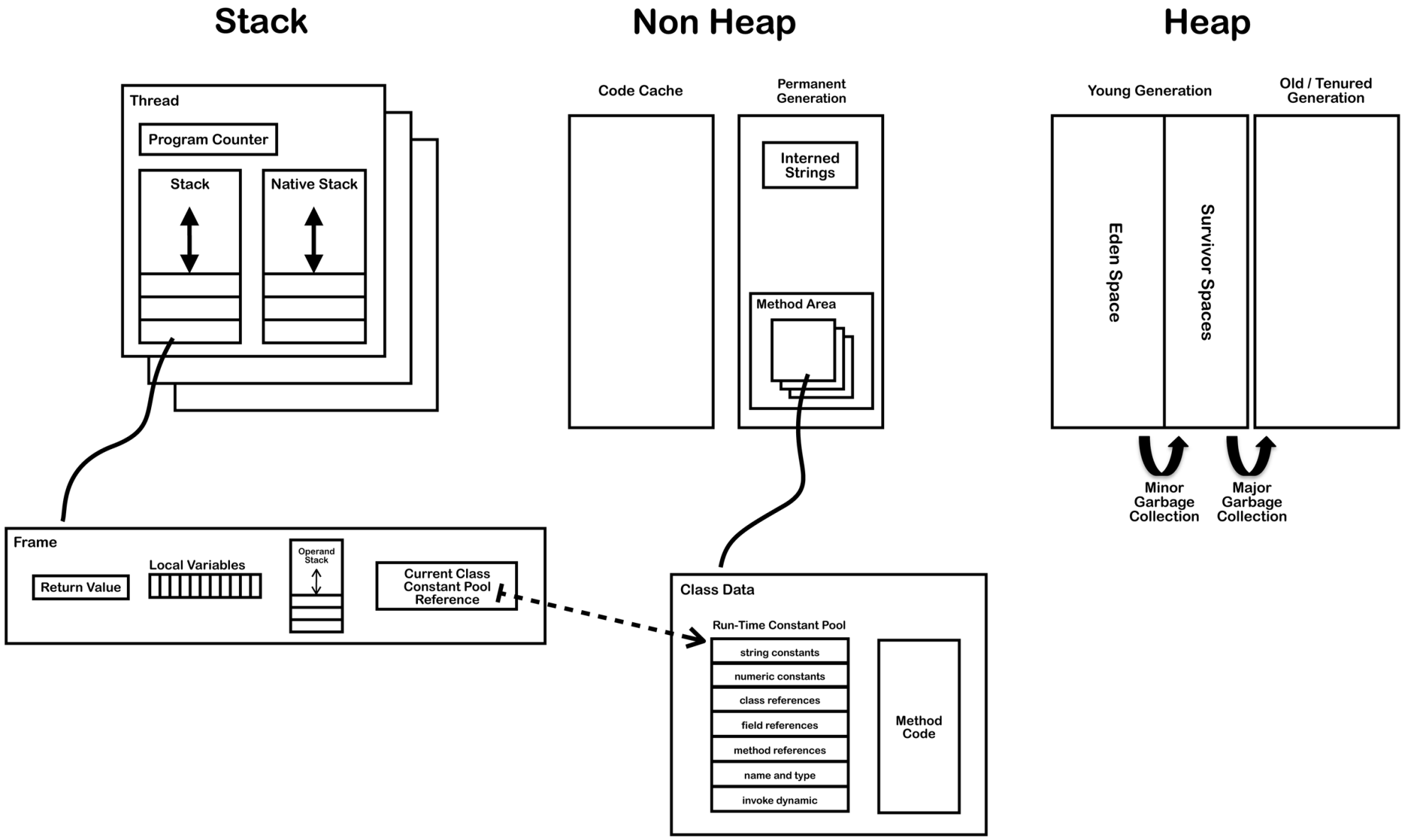
# Java


object reference
pointer to class info
inherited instance fields
instance fields
object
pointer to class fields
method table
class fields
inherited method
inherited method
method
method
class

# Run-time Constant Pool



# Stack Frame, Run-time Constant Pool



# Run-time Constant Pool

```
package org.jvminternals;

public class SimpleClass {

    public void sayHello() {
        System.out.println("Hello");
    }

}
```

```
Constant pool:
   #1 = Methodref          #6.#17         //  java/lang/Object."<init>":()V
   #2 = Fieldref           #18.#19        //  java/lang/System.out:Ljava/io/PrintStream;
   #3 = String             #20            //  "Hello"
   #4 = Methodref          #21.#22        //  java/io/PrintStream.println:(Ljava/lang/String;)V
   #5 = Class              #23            //  org/jvminternals/SimpleClass
   #6 = Class              #24            //  java/lang/Object
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               Lorg/jvminternals/SimpleClass;
  #14 = Utf8               sayHello
  #15 = Utf8               SourceFile
  #16 = Utf8               SimpleClass.java
  #17 = NameAndType        #7:#8          //  "<init>":()V
  #18 = Class              #25            //  java/lang/System
  #19 = NameAndType        #26:#27        //  out:Ljava/io/PrintStream;
  #20 = Utf8               Hello
  #21 = Class              #28            //  java/io/PrintStream
  #22 = NameAndType        #29:#30        //  println:(Ljava/lang/String;)V
  #23 = Utf8               org/jvminternals/SimpleClass
  #24 = Utf8               java/lang/Object
  #25 = Utf8               java/lang/System
  #26 = Utf8               out
  #27 = Utf8               Ljava/io/PrintStream;
  #28 = Utf8               java/io/PrintStream
  #29 = Utf8               println
  #30 = Utf8               (Ljava/lang/String;)V
```

# Run-time Constant Pool

```
package org.jvminternals;

public class SimpleClass {

    public void sayHello() {
        System.out.println("Hello");
    }

}
```

```
{
  public org.jvminternals.SimpleClass();
    Signature: ()V
    flags: ACC_PUBLIC
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0
         1: invokespecial #1    // Method java/lang/Object."<init>":()V
         4: return
      LineNumberTable:
        line 3: 0
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       5     0  this   Lorg/jvminternals/SimpleClass;

  public void sayHello();
    Signature: ()V
    flags: ACC_PUBLIC
    Code:
      stack=2, locals=1, args_size=1
         0: getstatic      #2    // Field java/lang/System.out:Ljava/io/PrintStream;
         3: ldc            #3    // String "Hello"
         5: invokevirtual  #4    // Method java/io/PrintStream.println:(Ljava/lang/String;)V
         8: return
      LineNumberTable:
        line 6: 0
        line 7: 8
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       9     0  this   Lorg/jvminternals/SimpleClass;
}
```

# Java binding

**Statički metodi, preklopljeni i privatni metodi se vezuju statički!**

Vezivanje poziva sa kodom statičkih metoda se vrši **statički** (early binding) , tj. u vreme kompajliranja.

Odlučivanje koji će metod biti pozvan se vrši na osnovu tipa reference.

**Nestatički metodi se vezuju dinamički!**

Vezivanje poziva sa kodom metoda koji pripadaju objektima se vrši **dinamički** (late binding), tj. u vreme izvršavanja.

Odlučivanje koji će metod biti pozvan se vrši na osnovu tipa objekta.