2011 - 2012

Programare **O**rientata **spre O**biecte
(*Object-Oriented Programming*)

a.k.a. Programare Obiect-Orientata

Titular curs: Eduard-Cristian Popovici

Suport curs: *http://discipline.elcom.pub.ro/POO-Java/*

1. **Introducere in abordarea orientata spre obiecte (OO)**

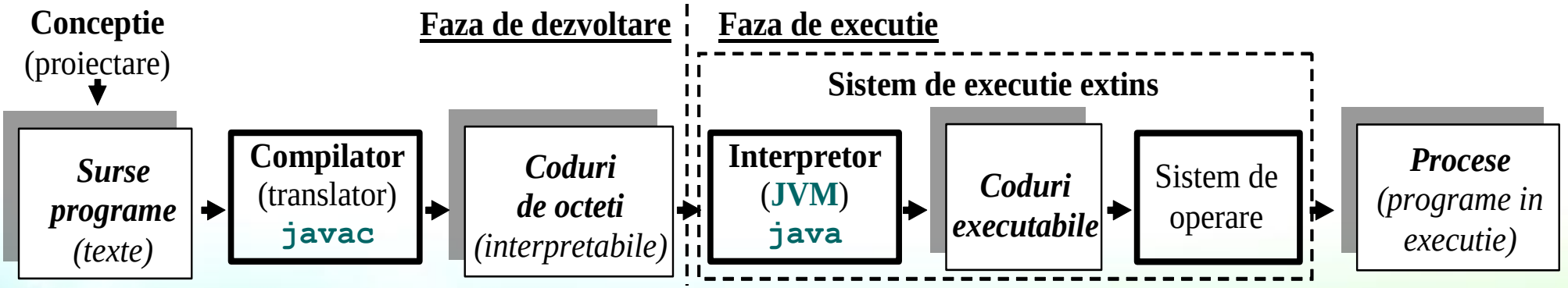
   1.4. Scurta recapitulare a programarii procedurale/structurate (Introducere in limbajul Java)

Masina virtuala Java si dezvoltarea programelor Java

Masina virtuala Java

In dezvoltarea programelor Java

- **codurile sursa** sunt translatate (**compilate** cu `javac`) din limbajul Java
  - in coduri numite **coduri de octeti** (*bytecodes*) executabile pe procesorul JVM
- **apoi** codurile de octeti sunt **interpretate**
  - adica **executate de interpretorul Java** (`java`), parte **din JVM**
    - prin **apeluri** ale JVM **catre sistemul de operare** al sistemului hardware



Dezvoltarea programelor Java

Masina virtuala Java

– este un **calculator abstract**

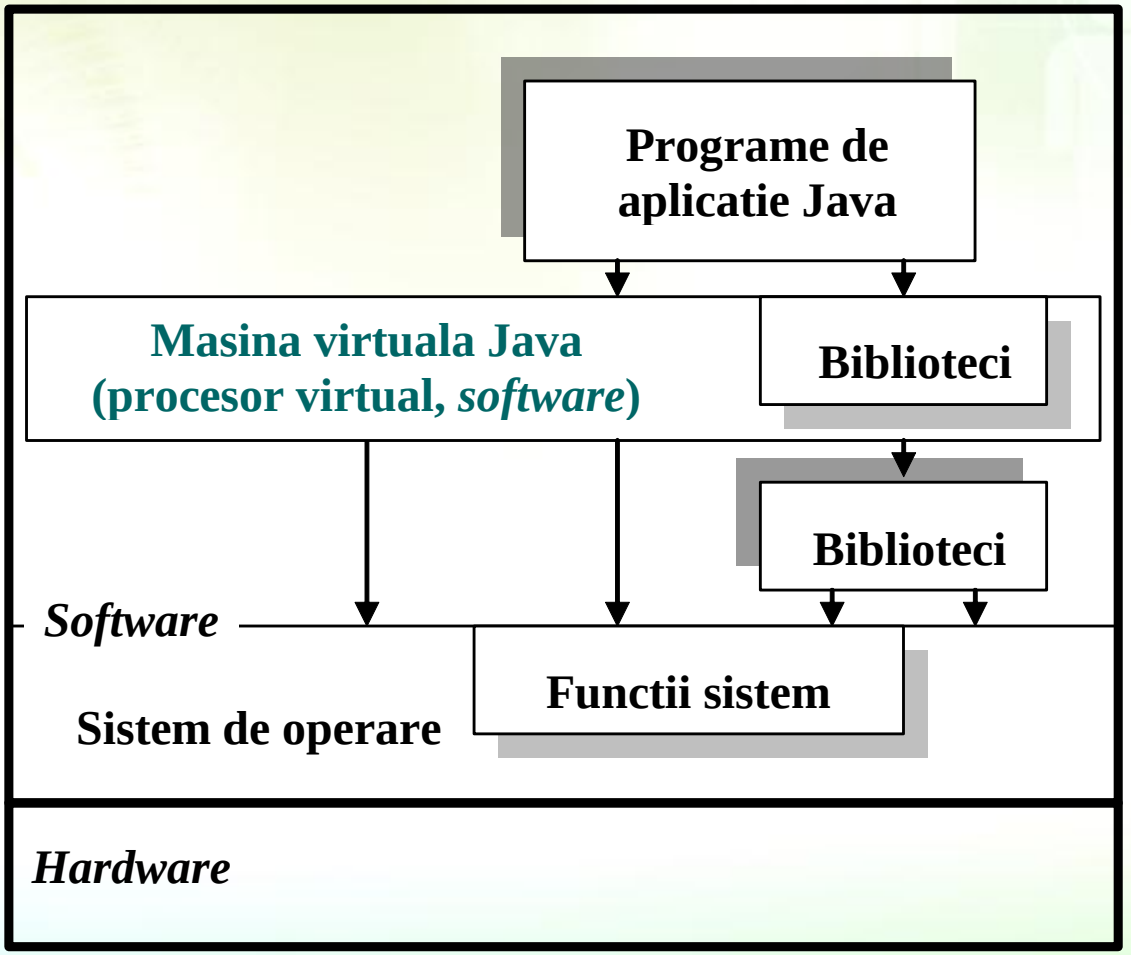
– adica un **procesor software**

– care **apeleaza la sistemul de operare** al sistemului hardware

– **nu la** sistemul **hardware** (la care are doar indirect acces)

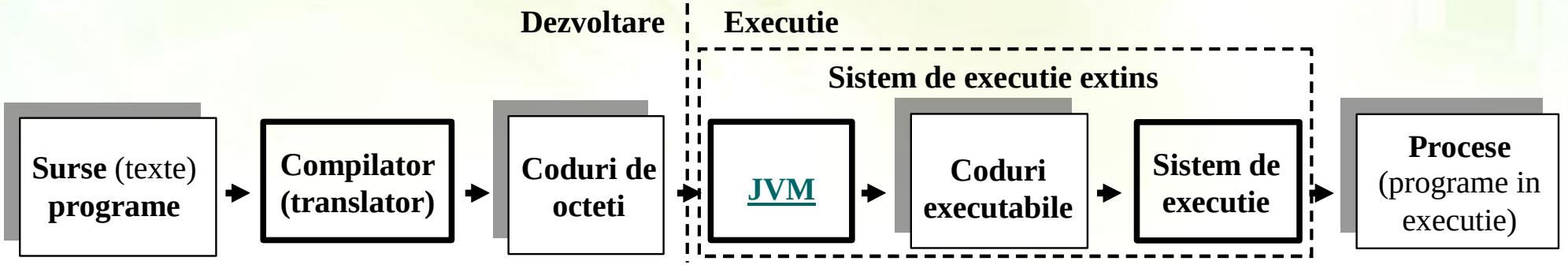
– ofera suport pentru **portabilitatea programelor / independenta de platforma**

– stand astfel **la baza** realizarii **limbajului de programare / tehnologiei Java**

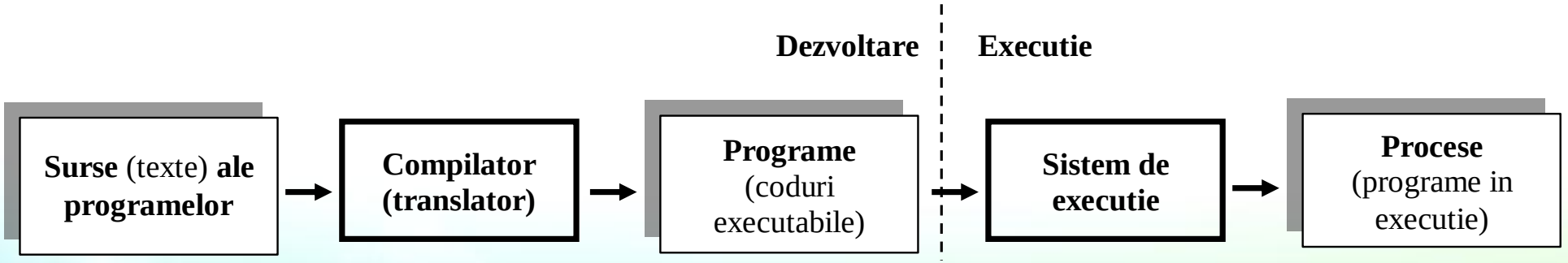


Dezvoltarea programelor Java vs dezvoltarea traditionala

Cazul unui program Java

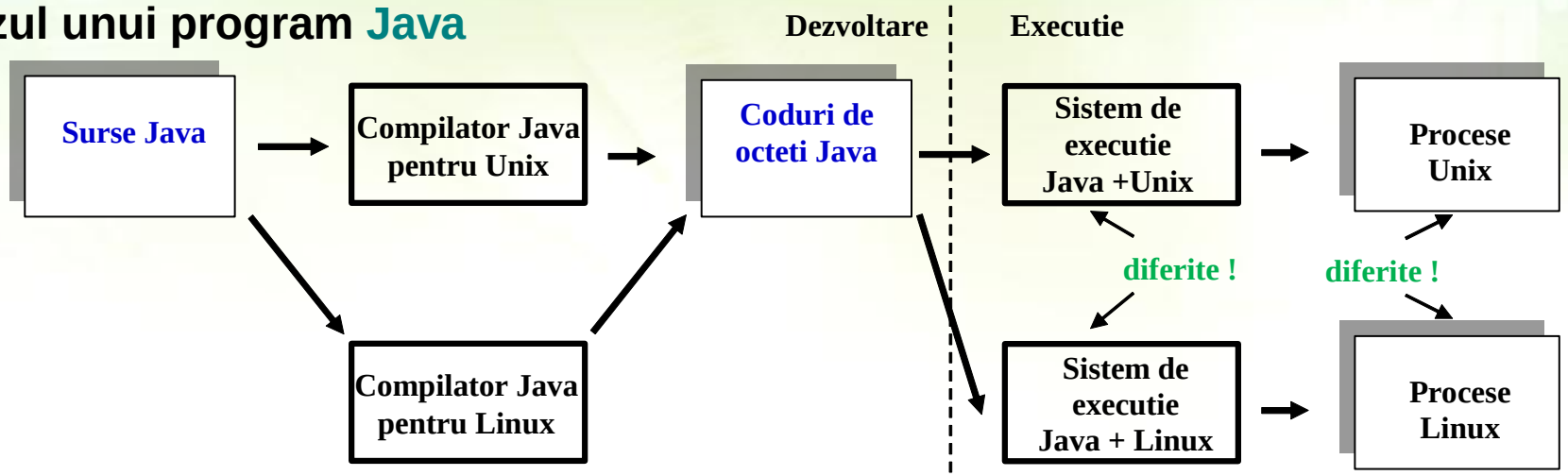


Cazul unui program C, C++, etc.

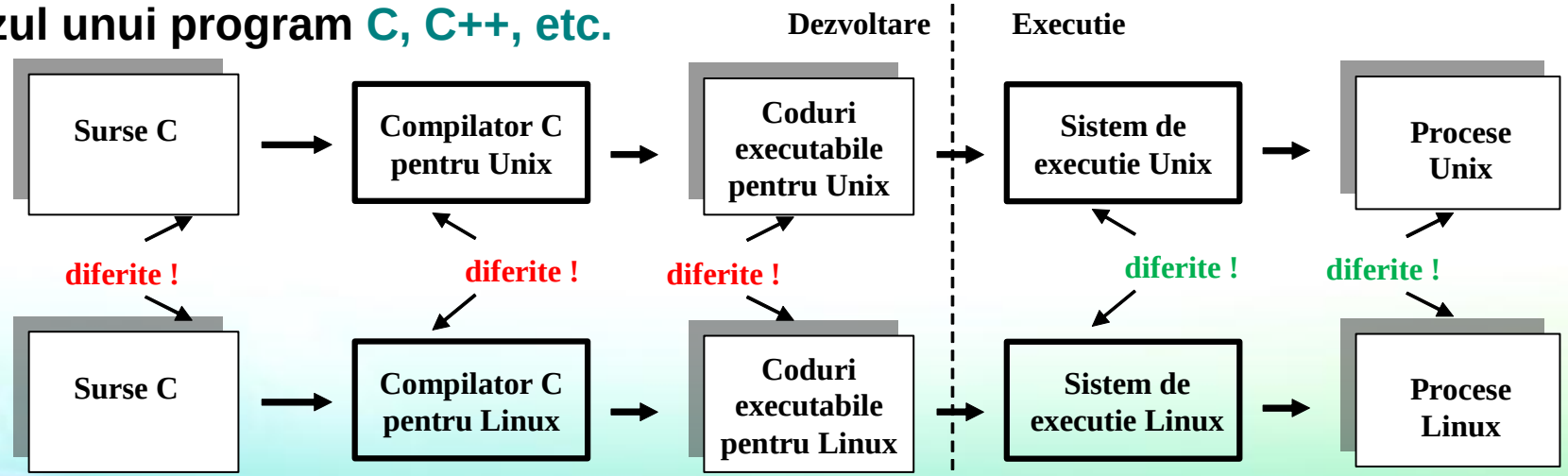


Dezvoltarea programelor Java vs dezvoltarea traditionala

Cazul unui program Java



Cazul unui program C, C++, etc.



Conventii Java

Conventii ale limbajului Java

Cuvintele care **incep cu litera mica** sunt

- **cuvinte rezervate** (neutilizate sau valori literale) sau **cheie**

de ex. `int`

`abstract`	`const` (neutilizat)	`final`	`int`	`public`	`throw`
`assert` (din 1.4)	`continue`	`finally`	`interface`	`return`	`throws`
`boolean`	`default`	`float`	`long`	`short`	`transient`
`break`	`do`	`for`	`native`	`static`	`true`
`byte`	`double`	`goto` (neutilizat)	`new`	`strictfp` (din 1.2)	`try`
`case`	`else`	`if`	`null`	`super`	`void`
`catch`	`enum` (din 5.0)	`implements`	`package`	`switch`	`volatile`
`char`	`extends`	`import`	`private`	`synchronized`	`while`
`class`	`false`	`instanceof`	`protected`	`this`	

- sau **variabile**, daca **numele NU este urmat de paranteze**

de ex. `razaCercului`

- **metode (functii)**, daca **numele este urmat de paranteze**

de ex. `arieCerc(..)`

Conventii ale limbajului Java

Cuvintele care **incep cu litera mare** sunt

- **clase**, daca **numele NU este urmat de paranteze**

  de ex. `String`

- **constructori**, daca **numele ESTE urmat de paranteze**

  (**functii** care au **acelasi nume cu clasa**, folosite pentru **crearea/initializarea obiectelor**)

  de ex. `String(..)`

Se observa faptul ca

- **in cazurile de mai sus NU se folosesc separatori intre multi-cuvinte** ci

  - **toate cele de dupa primul incep cu litera mare**

    de ex. `C` in cazul `razaCercului`

Cuvintele formate **DOAR din litere mari si despartite prin *underscore*** ("_") sunt

- **constante** (care in Java sunt "variabile nemodificabile", de ex. `PI_PATRAT`

Cuvinte cheie Java

OO = tine de orientarea spre obiecte,
exceptii = tine de tratarea exceptiilor,
bold = existent si in limbajul C

abstract (**OO**)	finally (**exceptii**)	public (**OO**)
boolean	**float**	**return**
break	**for**	**short**
byte	**if**	**static**
case	implements (**OO**)	super (**OO**)
catch (**exceptii**)	import	**switch**
char	instanceof (**OO**)	synchronized
class (**OO**)	**int**	this (**OO**)
continue	interface (**OO**)	throw (**exceptii**)
default	**long**	throws (**exceptii**)
do	native	transient
double	new (**OO**)	try (**exceptii**)
else	package	**void**
extends (**OO**)	private (**OO**)	**volatile**
final (**OO**)	protected (**OO**)	**while**

Variabile si tipuri de date Java

Date si variabile in Java

Programul in sens clasic se ocupa cu **prelucrari** asupra unor **date**

```
1  int suma;              // declaratia (tipului) variabilei suma
2  suma = 0;              // initializarea variabilei suma
3  for (int i=1; i<=10; i++) {
4     suma = suma + i;    // utilizarea variabilei (citire+scriere valoare)
5  }
```

Datele sunt reprezentate ca **variabile** (**locatii** de **memorie** cu **nume**)

Variabila este definita prin:

- **numele** ei, care o identifica si este un *alias* pentru adresa numerica (de exemplu, suma)
- **valoarea** continuta (de exemplu, suma contine pe rand valorile: 0, 1, 3, 6, .., 55)
- **locatia** in care e continuta valoarea (in cazul suma, locatia ocupa in Java 4B=32b)
- **adresa** numerica (inaccesibila in anumite limbaje, cum este Java)
- **tipul** de date (de exemplu, suma este de tip int)

Tipuri de date in Java

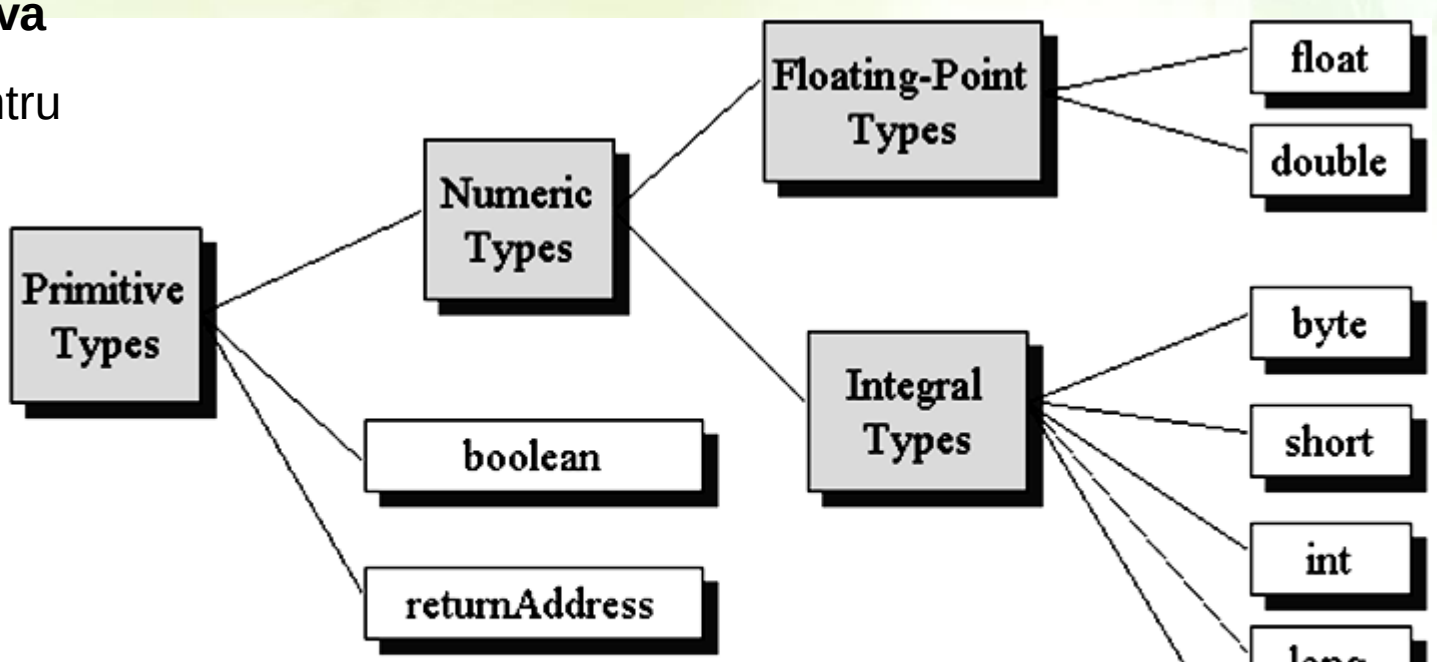
Tipul de date

- este o **descriere abstracta** a unui **grup de entitati asemanatoare**
- specifica **structura variabilelor** si **domeniul de definitie al valorilor**, adica:
  - **spatiul de memorie alocat** pentru stocarea valorii
  - **gama / spatiul / multimea valorilor** posibile
  - **formatul valorilor literale/de tip imediat** (de ex., sufixul f pentru valori de tip float)
  - **conventiile privind conversiile** catre alte tipuri: **direct** (implicit, prin **extindere**) sau **explicit** (prin *cast*, prin **trunchiere**)
  - **valorile implicite** (daca este cazul)
  - **operatorii asociati** (permisi) – tin de partea de **prelucrare** asupra datelor

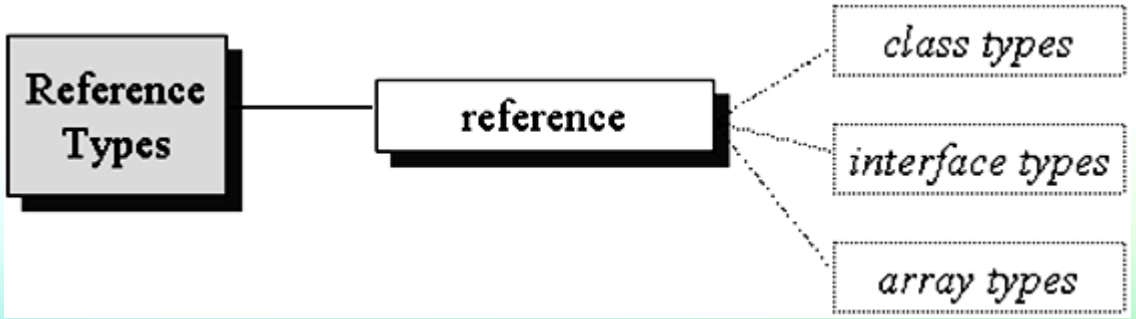
Tipuri de date in Java

Tipurile de date Java

– **primitive** (pentru **variabile** create **static** in **stiva**)



– **referinta** (pentru **variabile** create **dinamic** in ***heap***)



Tipurile de date primitive Java

Tipuri de date primitive in Java

Categorie	Tip	Valoare implicita	Spatiu memorie	Gama valori	Conversii explicite (cast, trunchiere)	Conversii implicite (extindere)
Valori intregi cu semn	byte	0	8 biti (1B)	-128 … 127	La char	La short, int, long, float, double
	short	0	16 biti (2B)	-32768 … 32767	La byte, char	La int, long, float, double
	int	0	32 biti (4B)	-2147483648 … 2147483647	La byte, short, char	La long, float, double
	long	0l	64 biti (8B)	-9223372036854775808 …9223372036854775807	La byte, short, int, char	La float, double
Valori in virgula mobila cu semn	float	0.0f sau 0.0F	32 biti (4B)	+/-1.4E-45 … +/-3.4028235E+38, +/-infinity, +/-0, NaN	La byte, short, int, long, char	La double
	double	0.0 echivalent 0.0d sau 0.0D	64 biti (8B)	+/-4.9E-324 … +/-1.7976931348623157E+308, +/-infinity, +/-0, NaN	La byte, short, int, long, float, char	Nu exista (nu sunt necesare)
Caractere codificate UNICODE	char	\u0000 (null)	16 biti (2B)	\u0000 … \uFFFF	La byte, short	La int, long, float, double
Valori logice	boolean	false	1 bit folosit din 32 biti	true, false	Nu exista (nu sunt posibile)	Nu exista (nu sunt posibile)

Tipuri de date in Java

Exemple de conversii intre tipurile primitive:

– intre valori **intregi**

– care dintre urmatoarele coduri ar genera eroare si de ce?

```
int i = 10;
byte b;
short s;
long l;
b = i;
s = i;
l = i;
i = l;
```

Tipuri de date in Java

Exemple de conversii intre tipurile primitive:

– intre valori **intregi**

```
int i = 10;
byte b;
short s;
long l;
// Ar genera eroare:                    // Coduri corectate:
//   b = i;                             b = (byte) i;
//   s = i;                             s = (short) i;
// Nu genereaza eroare:
l = i;
// Dar genereaza eroare:
//   i = l;                             i = (int) l;
```

Tipuri de date in Java

Exemple de conversii intre tipurile primitive:

– intre valori **intregi** si valori **char**

– care dintre urmatoarele coduri ar genera eroare si de ce?

```
int i = 10;
byte b = 100;
long l = i;
char c;
c = b;
c = i;
c = l;
b = c;
i = c;
l = c;
```

Tipuri de date in Java

Exemple de conversii intre tipurile primitive:

– intre valori **intregi** si valori **char**

```
int i = 10;
byte b = 100;
long l = i;
char c = 100;
// Ar genera eroare:
//   c = b;
//   c = i;
//   c = l;
//   b = c;
i = c;
l = c;
```

```
// Coduri corectate:
c = (char) b;
c = (char) i;
c = (char) l;
b = (byte) c;
```

Tipuri de date in Java

Exemple de conversii intre tipurile primitive:

– intre valori **intregi** si valori **cu virgula**

– care dintre urmatoarele coduri ar genera eroare si de ce?

```
int i = 10;
long l = i;
double d = 1.0;
float f = 2.0;
f = d;
d = f;
f = i;
i = f;
```

Tipuri de date in Java

Exemple de conversii intre tipurile primitive:

– intre valori **intregi** si valori **cu virgula**

```
int i = 10;
long l = i;
double d = 1.0;
// Ar genera eroare:                    // Coduri corectate:
// float f = 2.0;                       float f = 2.0f;
// f = d;                               f = (float) d;
d = f;
f = i;
// Ar genera eroare:
l = f;                                  l = (long) f;
```

Exemple introductive (lucrarea 1 de laborator)

Exemplu introductiv (lucrarea 1 de laborator)

```
public class Salut {                              // declaratia clasei
  public static void main(String[] args) {        // declaratia unei metode
    System.out.println("Buna ziua!");               // corpul metodei
  }                                                 // incheierea corpului metodei
}                                                 // incheierea corpului clasei
```

Cuvintele cheie de mai sus au, in general, urmatoarele semnificatii:

- **`public`**: specificator (calificator, modificator) al *modului de acces* la clase, metode (functii) si atribute (variabile avand drept scop clasele)
- **`class`**: declara o *clasa Java* (tip de date complex)
- **`static`**: specificator (calificator, modificator) al caracterului *de clasa* al unei metode sau al unui atribut (in lipsa lui, caracterul implicit al unei metode sau al unui atribut este *de obiect*)
- **`void`**: specifica faptul ca metoda nu returneaza nimic

Exemplu introductiv (lucrarea 1 de laborator)

```
public class Salut {
  public static void main(String[] args) {
    System.out.println(“Buna ziua!”); }
}
```

In particular, **cuvintele cheie de mai sus au urmatoarele semnificatii**:

- **`public`** din linia 1: codul clasei `Salut` poate fi accesat de orice cod exterior ei
- **`class`**: declara clasa Java `Salut`
- **`public`** din linia 2: codul metodei `main()` poate fi accesat de orice cod exterior ei
- **`static`**: metoda `main()` este o metoda cu caracter *de clasa* (nu cu caracter *de obiect*)
- **`void`**: metoda `main()` nu returneaza nimic

Exemplu introductiv (lucrarea 1 de laborator)

```
public class Salut {
  public static void main(String[] args) {
    System.out.println(“Buna ziua!”); }
}
```

Operatorii utilizati in programul de mai sus sunt:

- operatorul de **declarare a blocurilor** (acolade: “{“ si “}”),
- operatorul **listei de parametri** ai metodelor (paranteze rotunde: “(“ si “)”),
- operatorul de **indexare a tablourilor** (paranteze drepte: “[“ si “]“),
- operatorul de **calificare a numelor** (punct: “.“),
- operatorul de **declarare a sirurilor de caractere** (ghilimele: “”“ si “”“),
- operatorul de **sfarsit de instructiune** (punct si virgula: “;“).

Exemplu introductiv (lucrarea 1 de laborator)

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

Compilare (cu compilatorul **javac** si argument numele fisierului sursa Salut.java)

```
directorcurent> javac Salut.java
directorcurent> java Salut
Buna ziua!
directorcurent>
```

Interpretare (interpretorul **java** este programul executat de fapt, numele clasei Salut fiind doar un argument al lui)

Exemplu de program cu argumente primite din linia de comanda

```
1   public class SumaArgumenteIntregi {
2     public static void main(String[] args)  {
3       System.out.println("Au fost primite " + args.length + " argumente");
4
5       if (args.length > 0) {
6         int suma = 0;
7         for (int index = 0; index < args.length; index++) {
8           suma = suma + Integer.parseInt(args[index]);
9         }
10        System.out.println("Suma valorilor primite este " + suma);
11      }
12      else {
13        System.out.println("Utilizare tipica:");
14        System.out.println("\t java SumaArgumenteIntregi 12 31 133 -10");
15      }
16    }
17  }
```

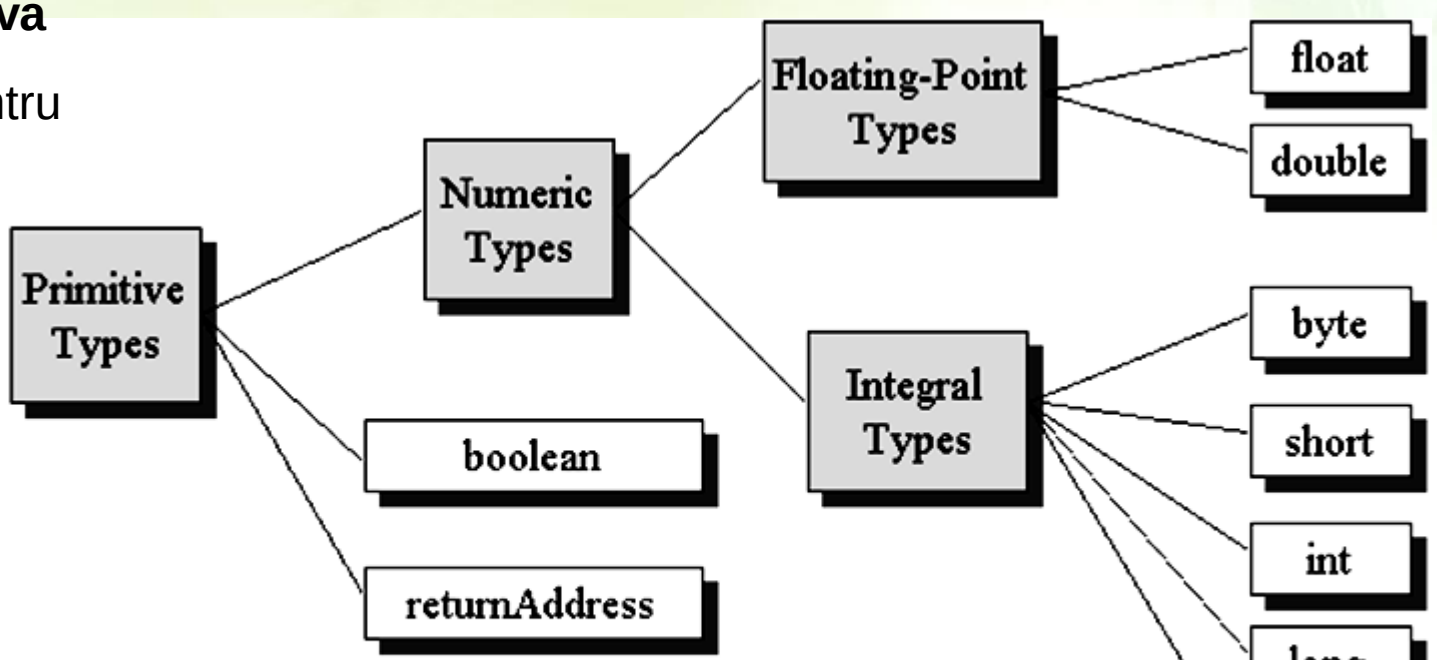
```
directorcurent> javac SumaArgumenteIntregi.java
directorcurent> java SumaArgumenteIntregi 12 31 133 -10
Au fost primite 4 argumente
Suma valorilor primite este 166
directorcurent>
```

Tipurile de date referinta Java

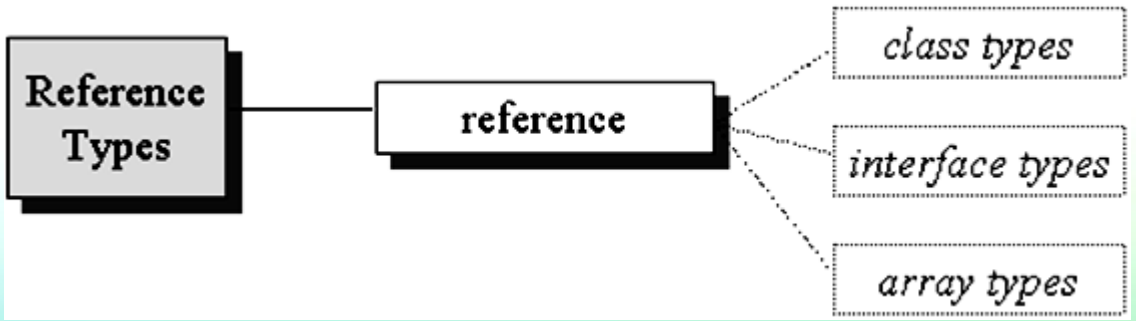
Tipuri de date in Java

Tipurile de date Java

– **primitive** (pentru **variabile** create **static** in **stiva**)



– **referinta** (pentru **variabile** create **dinamic** in ***heap***)



Tipuri referinta in Java

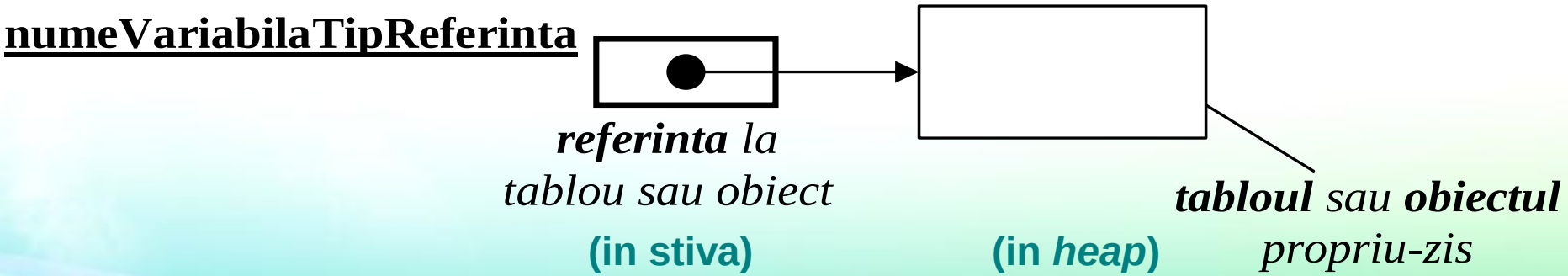
- tipul **tablou**
- tipul **clasa**
- tipul **interfata**

Variabilele de tip referinta sunt:

- variabile **tablou** - al caror tip este un **tablou**
- variabile **obiect** - al caror tip este o **clasa** / o **interfata**

Variabilele de tip referinta contin:

- **referinta** catre **tablou sau obiect** (creata in momentul declararii)
- **tabloul / obiectul propriu-zis** (creat in mod **dinamic**, cu **new**)



Tipuri referinta in Java

- **programatorul nu are acces la continutul referintelor**

  (in alte limbaje, cum sunt C/C++, pointerii si referintele pot fi accesate si tratate ca orice alta variabila)

  - **programatorul are acces doar la continutul tablourilor / obiectelor** referite
  - accesul la continutul tablourilor / obiectelor este permis **doar prin intermediul referintelor catre ele**

- o **valoare posibila** pentru referinte este si **null**, semnificand referinta “catre nimic”
  - simpla declarare a variabilelor referinta conduce la initializarea implicita a referintelor cu valoarea null

Tablourile Java

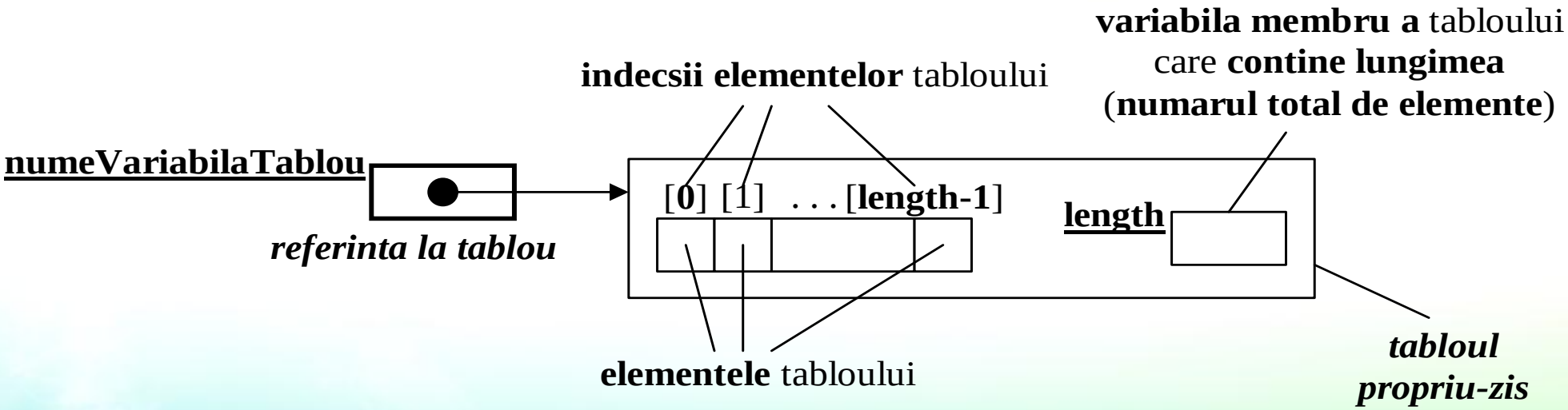
Tablourile in Java

Tabloul Java

- **structura** care contine **mai multe valori de acelasi tip**, numite **elemente**

Lungimea tabloului (numarul de elemente)

- **fixa**, stabilita **in momentul crearii tabloului** (cu operatorul **new**)
- este un **camp** (***field***, variabila membru) **al tabloului**



Tablourile in Java

Pentru a obtine numarul de elemente ale unui tablou se foloseste:

```
// Obtinerea dimensiunii tabloului de argumente pasate de utilizator
int numarArgumentePasateDeUtilizator = args.length;
```

Pentru a se crea un tablou cu valorile 1, 2, 3 se foloseste **sintaxa simplificata**:

```
// Crearea unui tablou de 3 valori intregi, varianta simplificata
int[] tab = { 1, 2, 3 };
```

Acelasi efect se obtine folosind **sintaxa complexa** pentru **crearea unui tablou**:

```
// Crearea unui tablou de 3 valori intregi, varianta complexa
int[] tab = new int[3]; // declararea variabilei si alocarea memoriei
tab[0]= 1;              // popularea tabloului
tab[1]= 2;              // popularea tabloului
tab[2]= 3;              // popularea tabloului
```

Tablourile in Java

Exemplu de utilizare

```
1    int[] t;

2    t = new int[6];

3    int[] v;

4    v = t;

5    int[] u = {1,2,3,4};

6    t[1] = u[0];

7    v = u;
```

Care este efectul fiecareia dintre liniile de cod de mai sus?

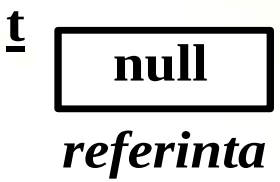
Tablourile in Java

Exemplu de utilizare

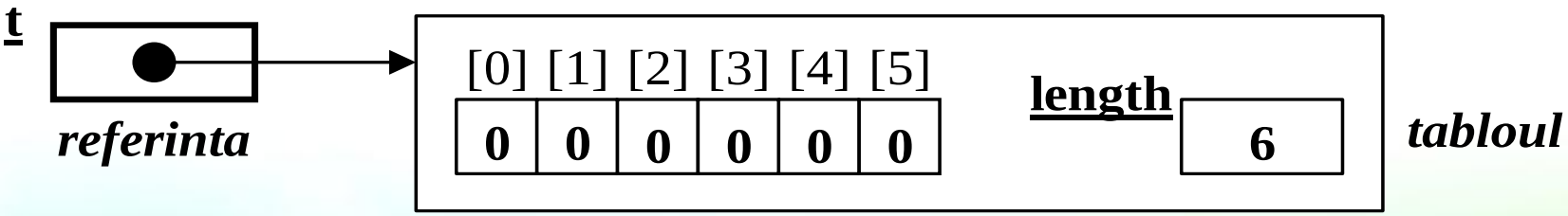
```
1    int[] t;                    // declarare simpla

2    t = new int[6];             // alocare si initializare

3    int[] v;                    // declarare simpla

4    v = t;                      // copiere referinte

5    int[] u = {1,2,3,4};        // declarare, alocare, initializare

6    t[1] = u[0];                // atribuire intre elemente

7    v = u;                      // copiere referinte
```

Tablourile in Java

Dupa executia liniei 1: `int[] t;`

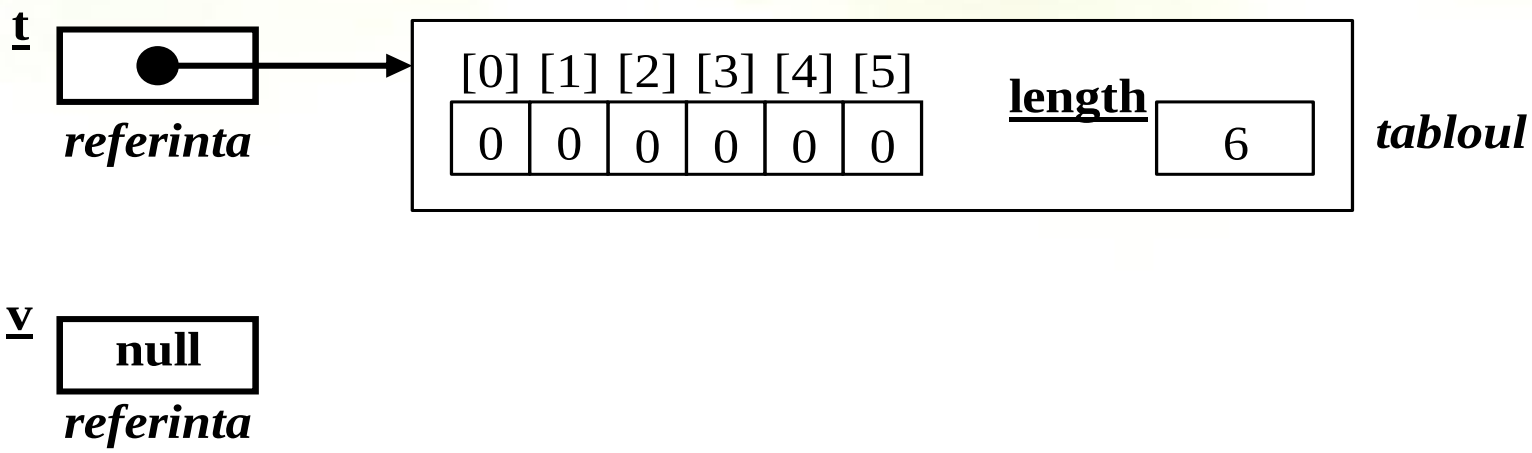


Dupa executia liniei 2: `t = new int[6];`



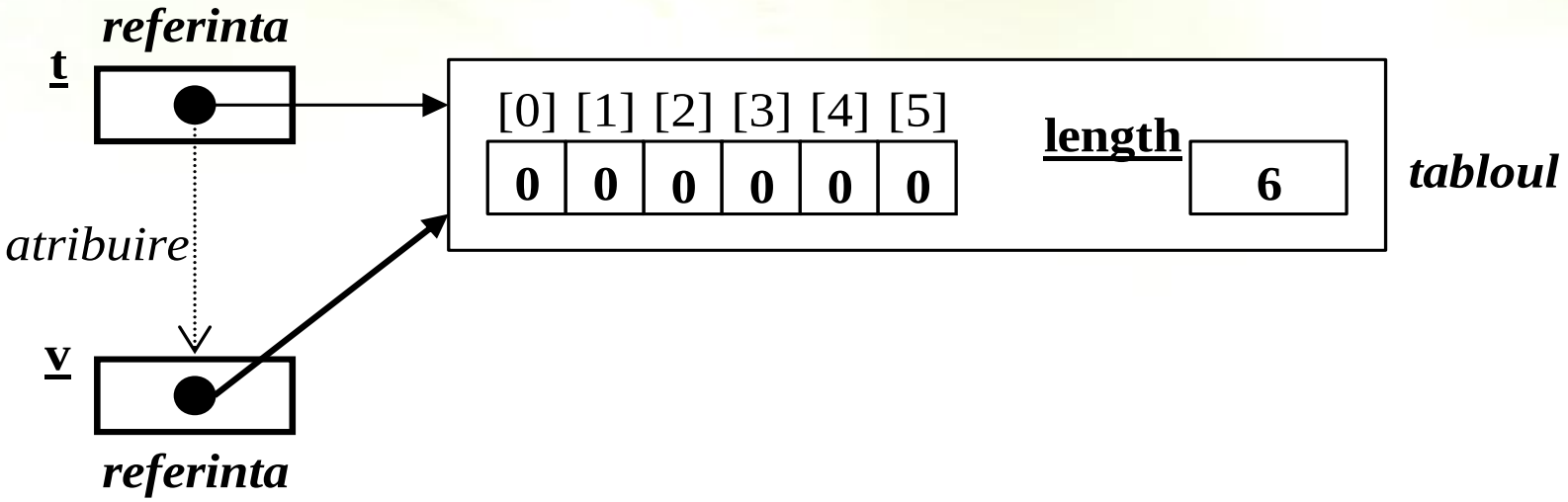
Tablourile in Java

Dupa executia liniei 3: `int[] v;`



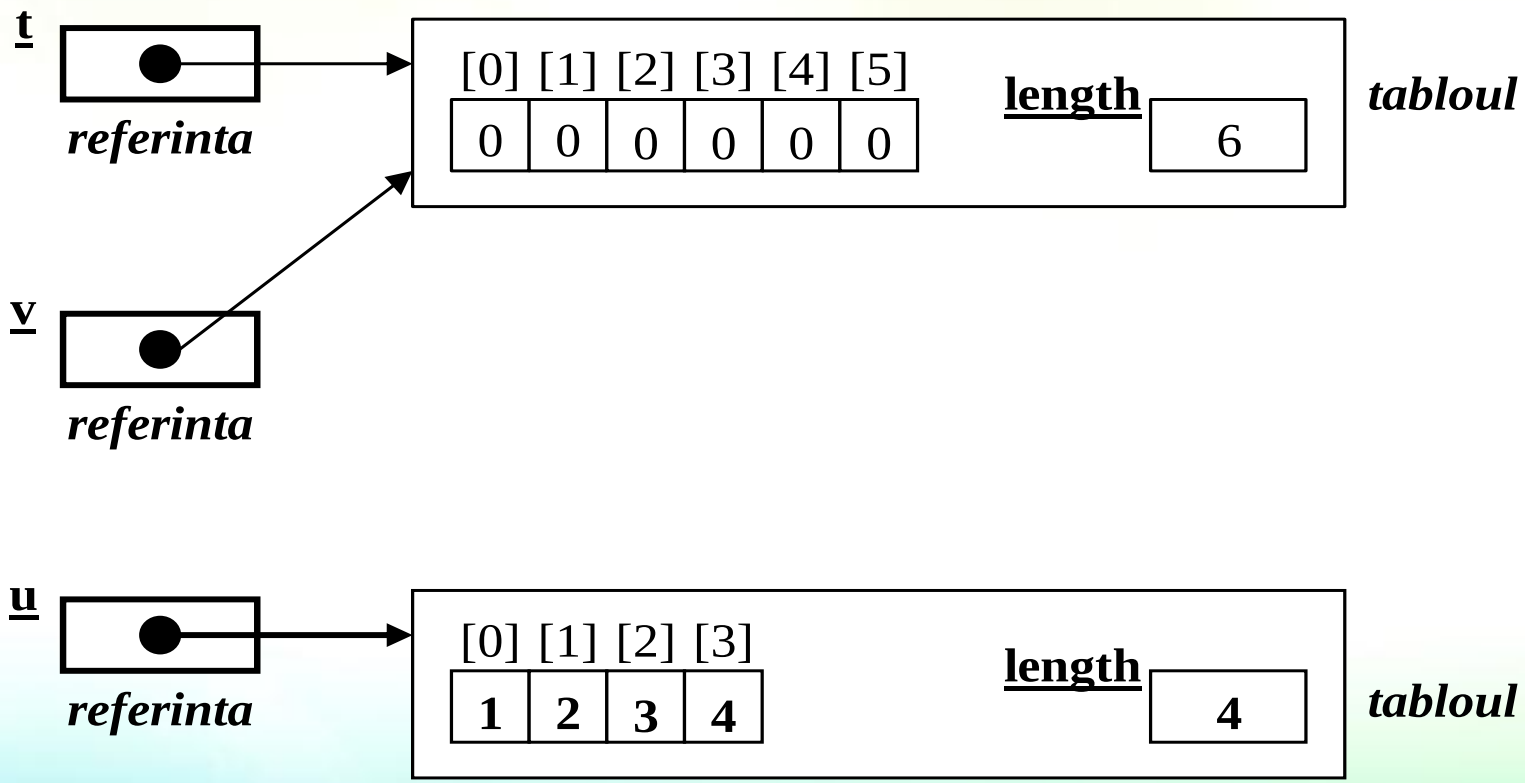
Tablourile in Java

Dupa executia liniei 4: `v = t;`



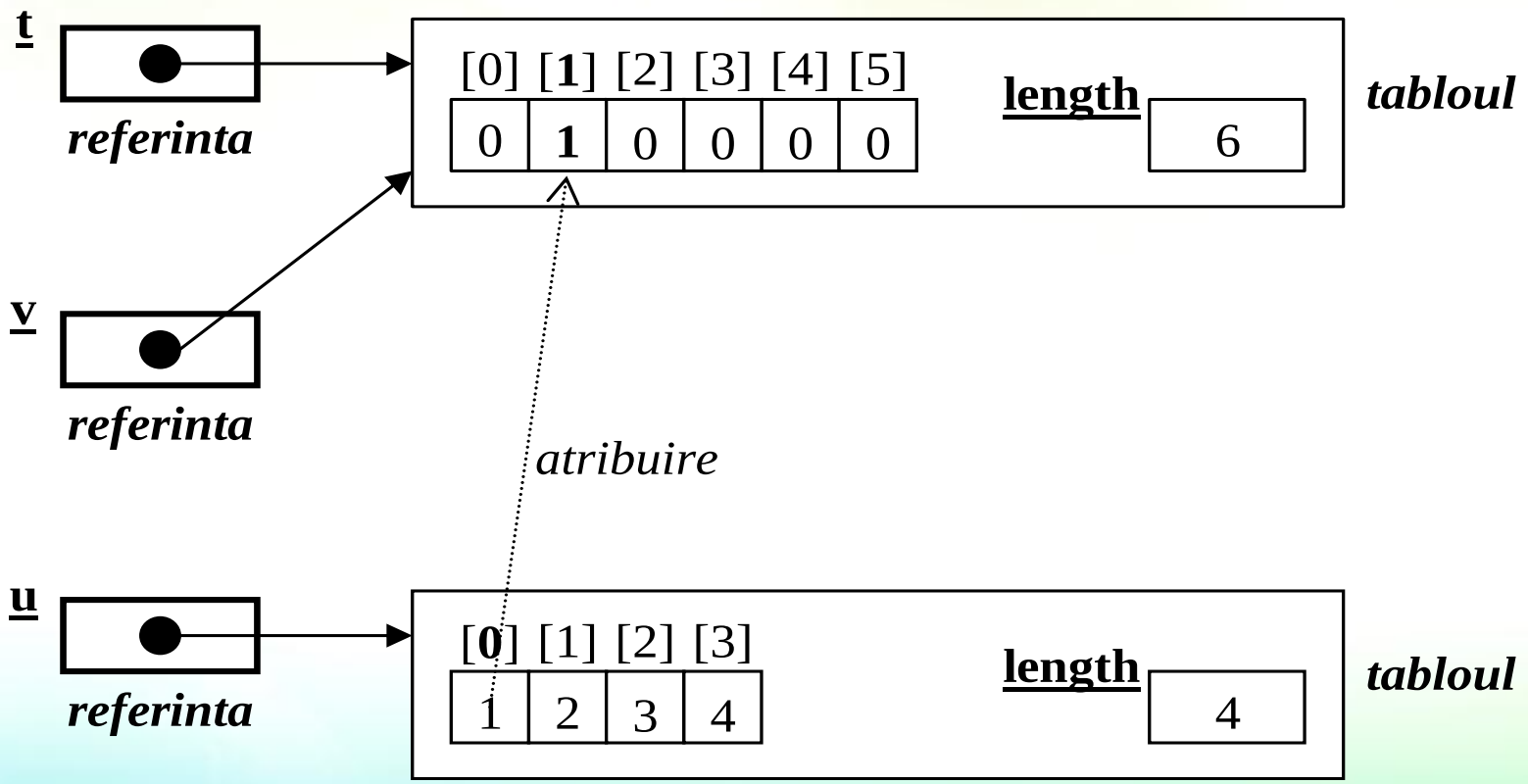
Tablourile in Java

Dupa executia liniei 5: `int[] u = {1, 2, 3, 4};`



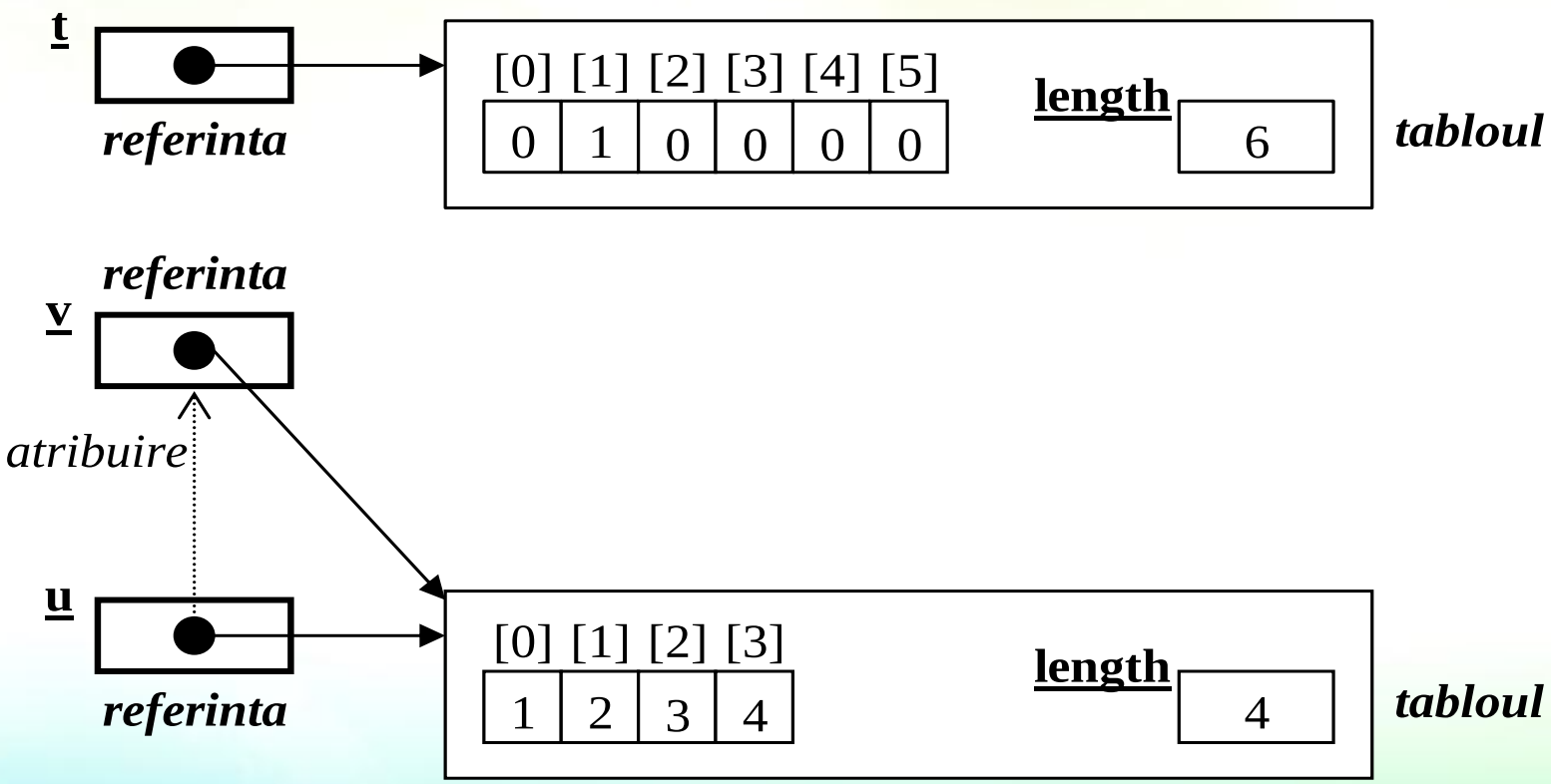
Tablourile in Java

Dupa executia liniei 6: `t[1] = u[0];`



Tablourile in Java

Dupa executia liniei 7: `v = u;`



Functiile (metodele) Java

Functii - necesitatea existentei acestora

Tot codul intr-o metoda (se observa **redundanta**)

```
// Program care afiseaza un "raport" format din doua parti
// incadrate si separate prin linii orizontale formate din 50 caractere
public class Raport01 {
  public static void main(String[] args) {
    final int LATIME = 50;                  // variabila finala (constanta!!)

    for (int i = 1; i <= LATIME; i++) System.out.print('-');
    System.out.println();                   // „traseaza o linie" de 50 de caractere

    System.out.println("Prima parte a raportului");
    for (int i = 1; i <= LATIME; i++) System.out.print('-');
    System.out.println();                   // „traseaza o linie" de 50 de caractere

    System.out.println("A doua parte a raportului");
    for (int i = 1; i <= LATIME; i++) System.out.print('-');
    System.out.println();                   // „traseaza o linie" de 50 de caractere
  }
}
```

Cate functii distincte vedeti mai sus?

Care dintre ele sunt apelate si care sunt definite?

Functii - necesitatea existentei acestora

Delegarea functionala (pentru **eliminarea redundantelor** si **modularizarea sarcinilor**) - catre o **metoda de tip static (a clasei, nu a obiectelor clasei)**

```
// Program care afiseaza un "raport" format din doua parti
// incadrate si separate prin linii orizontale formate din 50 caractere
public class Raport02 {

 private static void linie() { // definitia metodei (structura de program)
    final int LATIME = 50;
    for (int i = 1; i <= LATIME; i++) System.out.print('-');
    System.out.println();          // „traseaza o linie" de 50 de caractere
 }

 public static void main(String[] args) {
    linie();                                          // apelul metodei
    System.out.println("Prima parte a raportului");
    linie();                                          // apelul metodei
    System.out.println("A doua parte a raportului");
    linie();                                          // apelul metodei
 }
}
```

Cate functii distincte sunt mai sus? **Cum se modifica ierarhia apelurilor functiilor?**

Functii – parametri si argumente

Utilizarea unor parametri si **primirea argumentelor** (pentru **flexibilitatea utilizarii** si **genericitatea/reutilizabilitatea codului**)

```
// Program care afiseaza un "raport" format din doua parti
public class Raport03 {

 private static void linie(int latime) {              // definitia metodei
    for (int i = 1; i <= latime; i++) System.out.print('-');
    System.out.println();// „traseaza o linie" cu nr variabil de caractere
 }
 public static void main(String[] args) {
    final int LATIME_IMPLICITA = 50;
    linie(LATIME_IMPLICITA);                          // apelul metodei

    System.out.println("Prima parte a raportului");
    linie(LATIME_IMPLICITA - 5);                      // apelul metodei

    System.out.println("A doua parte a raportului");
    linie(LATIME_IMPLICITA);                          // apelul metodei
 }
}
```

Cum ar putea fi reutilizata metoda linie()?

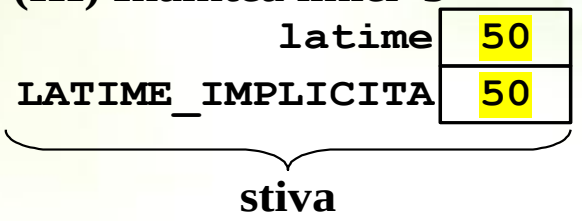
Functii - parametri si argumente

Starile succesive ale stivei in varianta **Raport03**

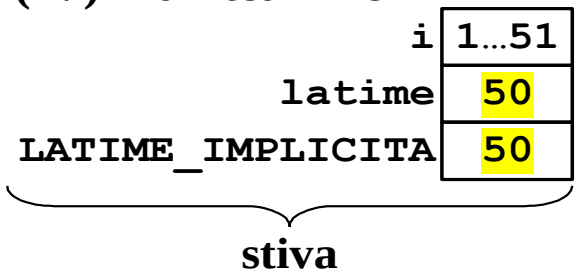
(I) Inaintea liniei 6

(II) Inaintea liniei 8

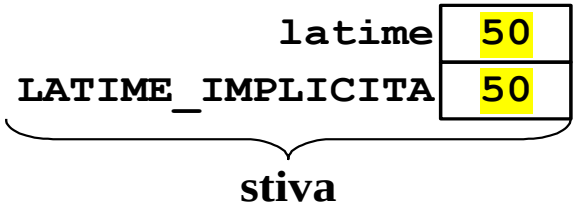
(III) Inaintea liniei 3



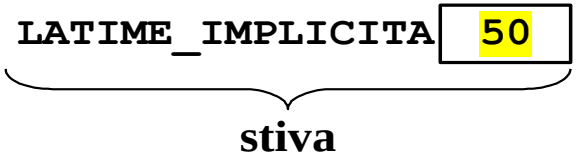
(IV) Inaintea liniei 4



(V) Inaintea liniei 5



(VI) Inaintea liniei 9



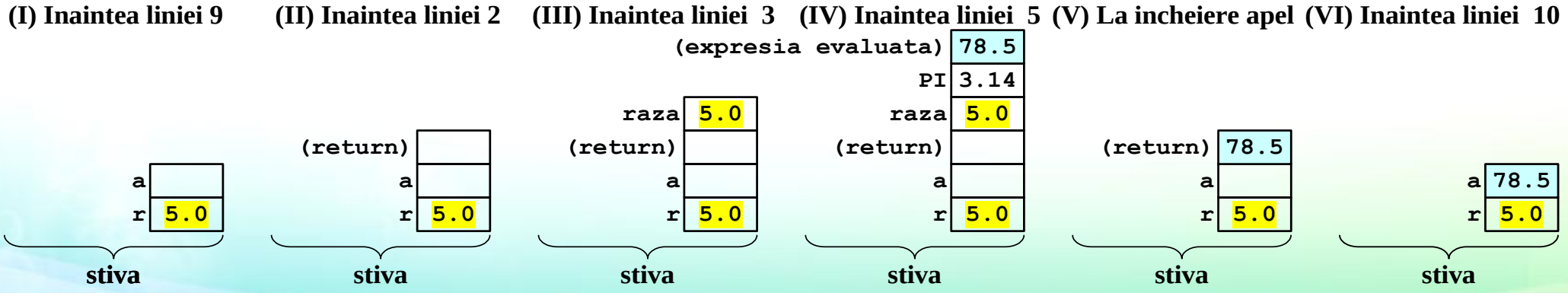
In Java - **stiva** (***stack***) contine **variabilele de tip primitiv** (byte, double, char, etc,)
si referintele la tablouri/obiecte

- **zona *heap*** contine **tablourile/obiectele propriu-zise**
(create dinamic cu **new**)

Functii - returnarea unor valori

```
// Program care calculeaza aria unui cerc in functie de raza
public class Cerc {
  private static double arie(double raza) {      // definitia metodei
     final double PI = 3.14159;                  // variabila finala (constanta!!)
     return 3.14159 * raza * raza;               // returnarea unei valori
  }
  public static void main(String[] args) {
     double r = 5.0;                            // variabila locala r
     double a;                                  // variabila locala a
     a = arie(r);                               // apelul metodei
     System.out.println("Un cerc de raza " + r + " are aria " + a + ".");
  }
}
```

Starile succesive ale stivei



Functii - pasarea argumentelor prin valoare (copie a valorii primite)

Cazul pasarii unei **valori primitive** de tip **int**

```
// Program care incrementeaza o valoare intreaga
public class C1 {

  // declaratie (semnatura) metoda inc()
  public static void inc(int i) {

    i++;  // i este parametru formal (pe scurt, parametru)

  }

  public static void main(String[] args) {

    int x = 10;

    inc(x);                          // apel metoda inc()

    System.out.println("x=" + x); // x este parametru actual (argument)

  }                                  // Rezultat final: x = 10
}
```

Care este evolutia valorilor variabilelor **i** si **x**, in **stiva**?

Functii - pasarea argumentelor prin valoare (copie a valorii primite)

Cazul pasarii unui **tablou** de tip **int[]**

```
// Program care incrementeaza un element al unui tablou
public class C2 {
  // primeste o copie a valorii referintei, asa incat refera acelasi tablou
  public static void inc(int[] i) {
    i[0]++;     // este incrementat primul element al tabloului
  }

  public static void main(String[] args) {
    int[] x = {10};                    // tablou cu un element, referit de x
    inc(x);                               // este pasata valoarea referintei

    System.out.println("x[0]=" + x[0]); // primul element al tabloului
  }                                       // Rezultat final: x[0] = 11
}
```

Care este evolutia valorilor variabilelor **i** si **x**, in **stiva** si in ***heap***?

Structuri de control al programului Java

Structuri de control al programului – decizie simpla

Structura:

```
<expresieBooleana> ? <expresie1> : <expresie2>
```

este echivalenta cu:

```
if (<expresieBooleana>)
   <expresie1>        // executata daca <expresieBooleana>==true
else
   <expresie2>        // executata daca <expresieBooleana>==false
```

In Java **expresia din paranteza trebuie sa fie logica**:

- sa fie **evaluata** la o **valoare de tip boolean** (**true** sau false)
- **nu poate fi de tip intreg** (ca in C, C++, etc.)

Structuri de control al programului – decizie multipla if else if

```
Date today = new Date();
if (today.getDay() == 0)
    System.out.println("Este duminica.");
else if (today.getDay() == 1)
    System.out.println("Este luni.");
else if (today.getDay() == 2)
    System.out.println("Este marti.");
else if (today.getDay() == 3)
    System.out.println("Este miercuri.");
else if (today.getDay() == 4)
    System.out.println("Este joi.");
else if (today.getDay() == 5)
    System.out.println("Este vineri.");
else
    System.out.println("Este sambata.");
```

Structuri de control al programului – decizie multipla switch case

```
Date today = new Date();
switch (today.getDay()) {
   case 0:    // duminica
     System.out.println("Este duminica.");
     break;
   case 1:    // luni
     System.out.println("Este luni.");
     break;
   case 2:    // marti
     System.out.println("Este marti.");
     break;
   case 3:    // miercuri
     System.out.println("Este miercuri.");
     break;
   case 4:    // joi
     System.out.println("Este joi.");
     break;
   case 5:   // vineri
     System.out.println("Este vineri.");
     break;
   default:  // sambata
     System.out.println("Este sambata.");
}
```

Structuri de control al programului - iteratii (bucle)

```
// repetare cat timp <expresieBooleana> == true
for (<initializare>; <expresieBooleana>; <actualizare>)
   <instructiuneExecutataRepetat>
```

```
// repetare cat timp <expresieBooleana> == true
while (<expresieBooleana>)
   <instructiuneExecutataRepetat>
```

```
// repetare cat timp <expresieBooleana> == true
do {
   <instructiuneExecutataRepetat>
} while (<expresieBooleana>);
```

Cum pot fi echivalate for si while?

Care e diferenta intre while si do..while?

Structuri de control al programului – break si continue

```
While (boolean expression) {
        statement1
        statement2
    if(boolean expression)
        break;
        statement3
}
statement4
```

```
While (boolean expression) {
        statement1
        statement2
    if(boolean expression)
        continue;
        statement3
}
statement4
```

Cum se poate iesi in C/C++ dintr-o bucla interna alteia?

Ce alternative ar exista?

Structuri de control al programului – break si continue

In C/C++ se iese dintr-o bucla interna alteia folosind **goto <eticheta>**

In Java se folosesc **break <eticheta>** si **continue <eticheta>**

```
outsideLoop: for( ... ) {
    ...
    while( ... ) {
        ...
        if ( ... ) {
            ...
            break outsideLoop;
        } // end if

        if ( ... ) {
            ...
            continue outsideLoop;
        } // end if
        ...
    }  // end while
    ...
}  // end for
```

Operatori binari pentru valori intregi

Operator	Operatie	Exemplu
=	**Atribuire**	a = b
==	**Egalitate**	a == b
!=	**Inegalitate**	a != b
<	**Mai mic decat**	a < b
<=	**Mai mic sau egal cu**	a <= b
>=	**Mai mare sau egal cu**	a >= b
>	**Mai mare decat**	a > b
+	**Adunare**	a + b
-	**Scadere**	a - b
*	**Inmultire**	a * b
/	**Impartire**	a / b
%	**Modul**	a % b
<<	**Deplasare la stanga**	a << b
>>	**Deplasare la dreapta**	a >> b
>>>	**Deplasare la dreapta cu umplere cu zero**	a >>> b
&	**SI pe biti**	a & b
|	**SAU pe biti**	a | b
^	**XOR pe biti**	a ^ b

Operatori unari pentru valori intregi

Operator	Operatie	Exemplu
-	**Negare unara**	-a
~	**Negare logica pe biti**	~a
++	**Incrementare**	a++ sau ++a
--	**Decrementare**	a-- sau --a

Operatori pentru valori booleene

Operator	Operatie	Exemplu
!	**Negare**	!a
&&	SI **conditional**	a && b
||	SAU **conditional**	a || b
==	**Egalitate**	a == b
!=	**Inegalitate**	a != b
?:	**Conditional**	a ? expr1 : expr2

Secvente escape

Secventa	Utilizare
`\b`	***Backspace***
`\t`	***Tab* orizontal**
`\n`	***Line feed***
`\f`	***Form feed***
`\r`	***Carriage return***
`\"`	**Ghilimele**
`\'`	**Apostrof**
`\\`	***Backslash***
`\uxxxx`	**Caracter Unicode numarul `xxxx`**

Delimitatorii de comentariu din Java

Incep ut	*Sfarsit*	*Scop*
`/*`	`*/`	Textul continut este tratat ca un comentariu.
`//`	(nimic)	Restul liniei este tratata ca un comentariu.
`/**`	`*/`	Textul continut este tratat ca un comentariu de catre compilator, si *poate folosit de catre JavaDoc pentru a genera automatic documentatie*.