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?

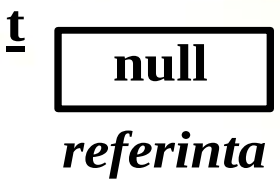
Tablourile in Java

Exemplu de utilizare

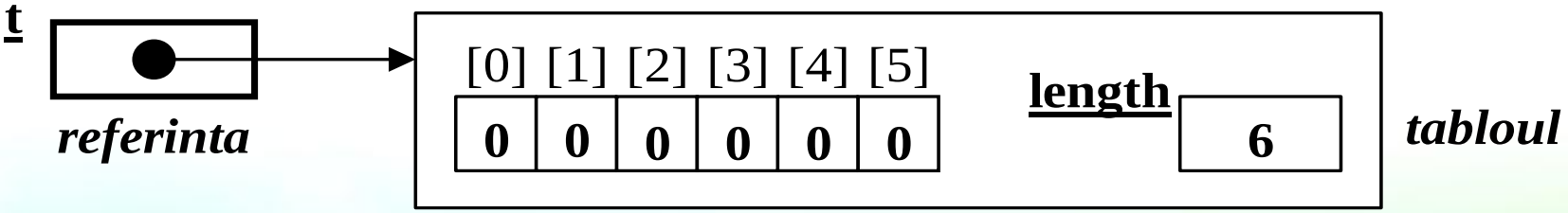
```
1    int[] t;                    // declarare simpla

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;`

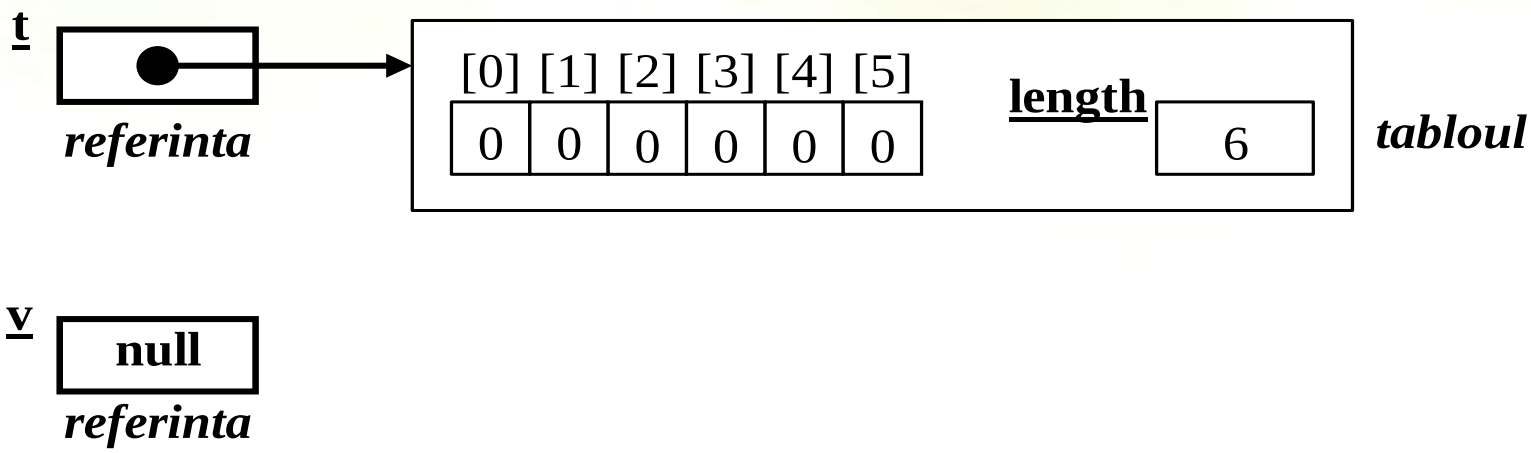


Dupa executia liniei 2: `t = new int[6];`



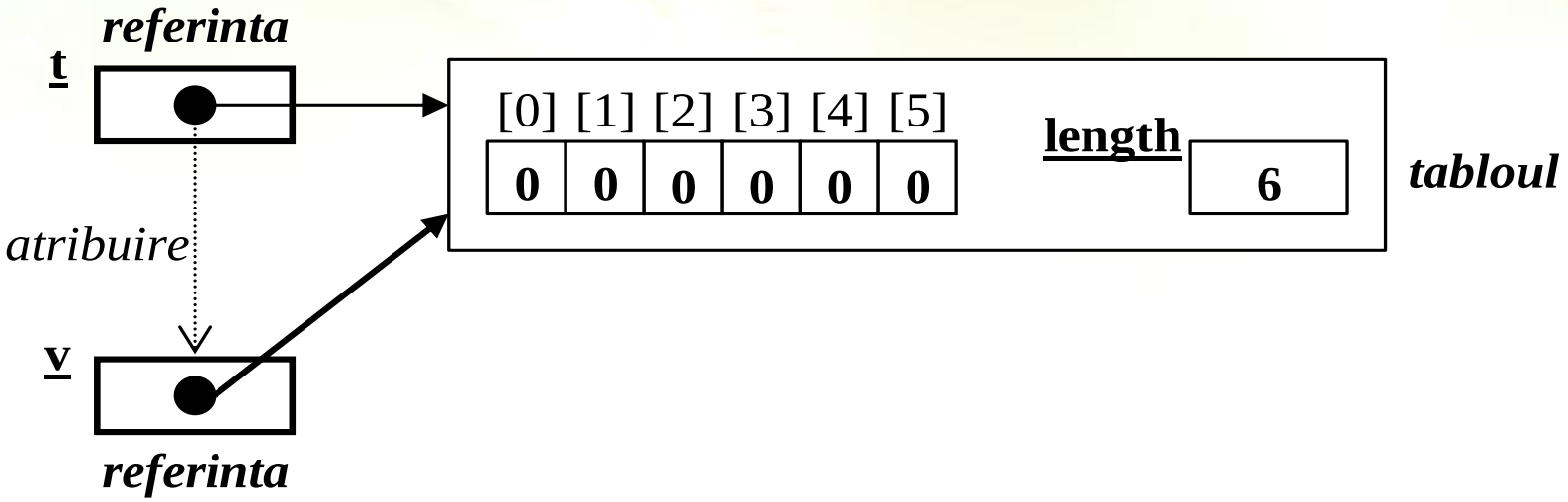
Tablourile in Java

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



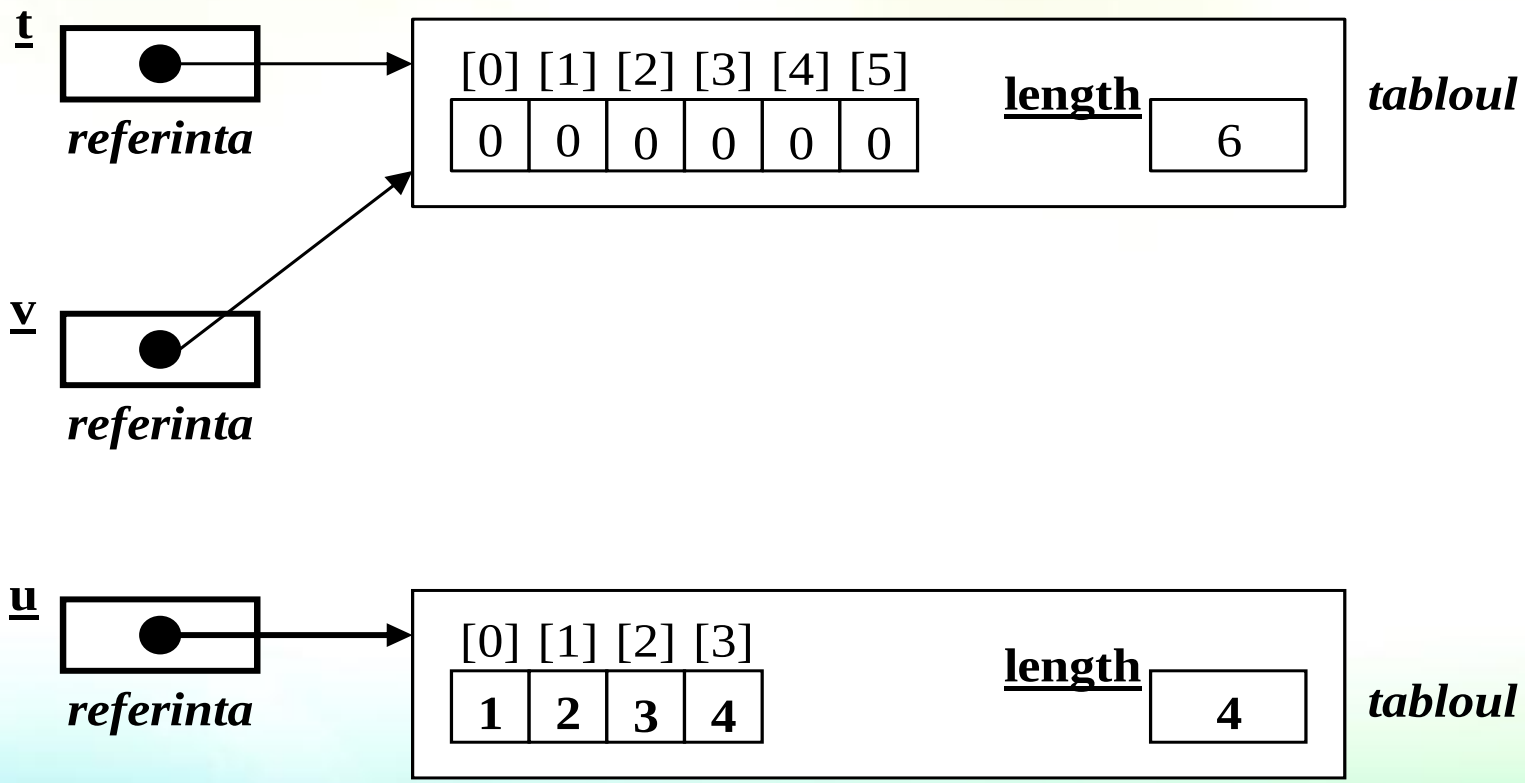
Tablourile in Java

Dupa executia liniei 4: `v = t;`



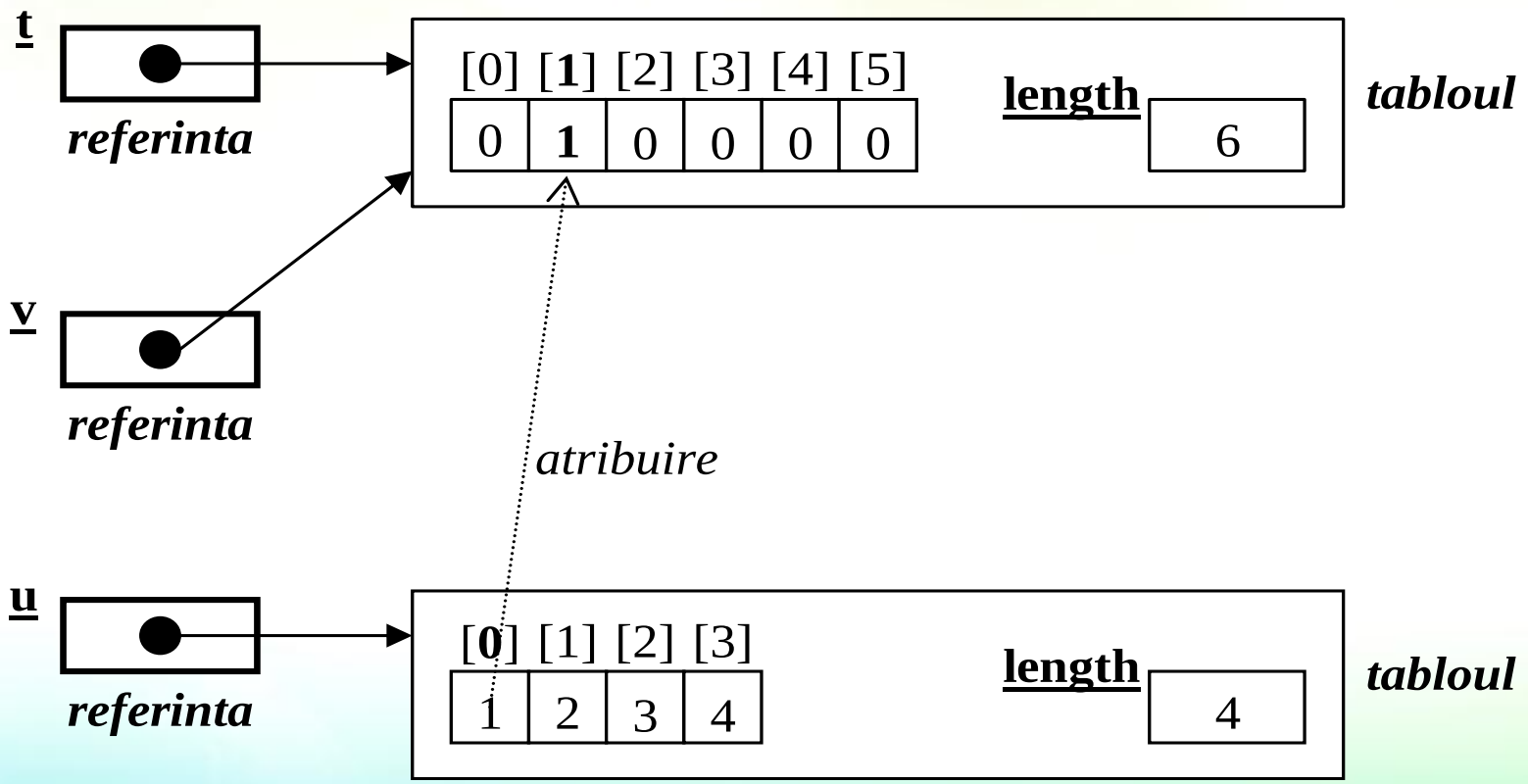
Tablourile in Java

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



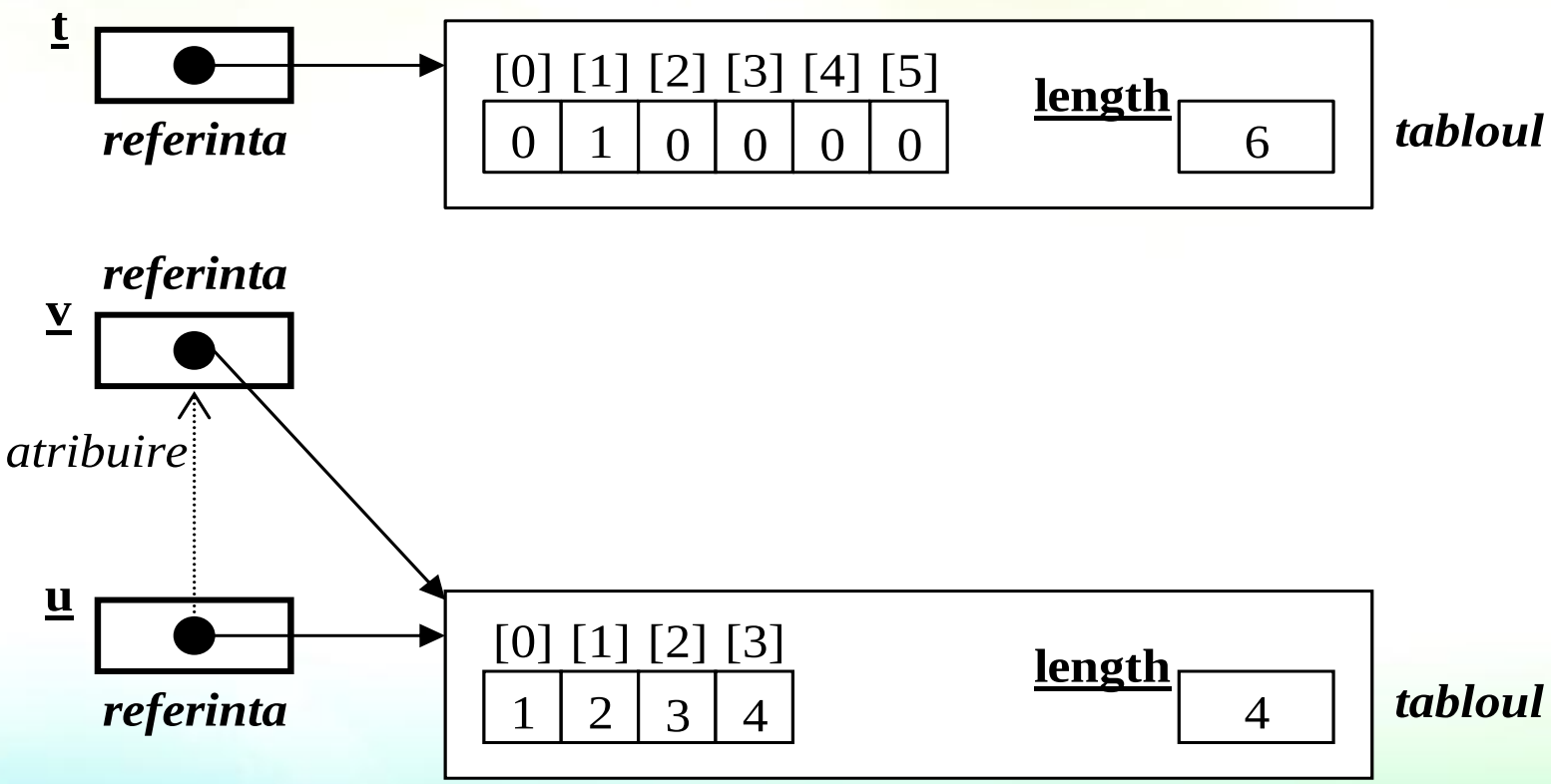
Tablourile in Java

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()?

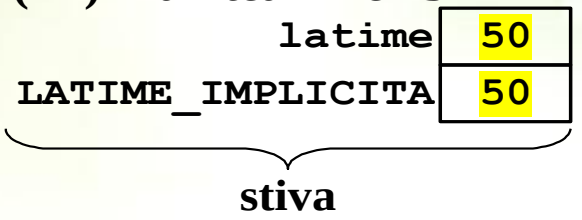
Functii - parametri si argumente

Starile succesive ale stivei in varianta **Raport03**

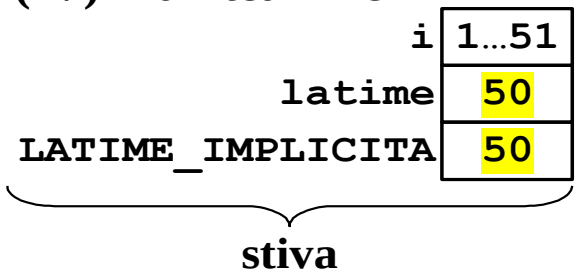
(I) Inaintea liniei 6

(II) Inaintea liniei 8

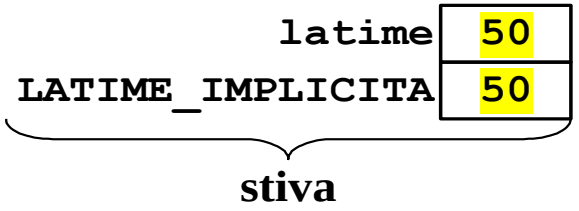
(III) Inaintea liniei 3



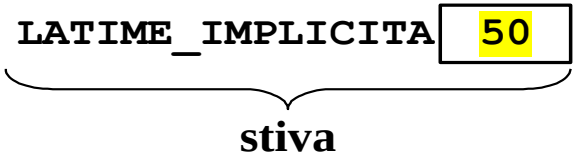
(IV) Inaintea liniei 4



(V) Inaintea liniei 5



(VI) Inaintea liniei 9



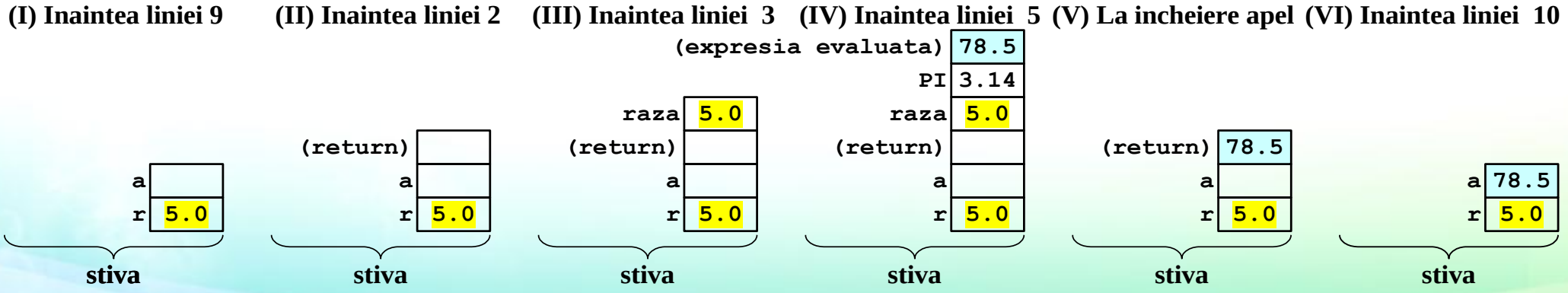
In Java - **stiva** (***stack***) contine **variabilele de tip primitiv** (byte, double, char, etc,)
si referintele la tablouri/obiecte

- **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

Inceput	*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*.