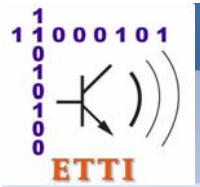
2009 - **2010**

Tehnologii de Programare in Internet (TPI / RST)

Titulari curs: **Mihnea Magheti**, Eduard-Cristian Popovici

Suport curs: *http://discipline.elcom.pub.ro/tpi/*

Moodle: *http://electronica07.curs.ncit.pub.ro/course/category.php?id=3*



Structura cursului

Continut curs TPI

1. Introducere in tehnologiile Internet

2. Introducere in tehnologiile desktop (SE) Java

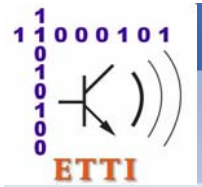
- 2.1. Elemente de baza. Tipuri de date referinta. Clase de biblioteca
- 2.2. Clase pentru fluxuri de intrare-iesire (IO)

3. Programarea la nivel socket in Java

- 3.1. Introducere in Protocolul Internet (IP) si stiva de protocoale IP
- 3.2. Socketuri flux (TCP) Java si programe multifilare (threads)
- 3.3. Socketuri datagrama (UDP) Java

4. Tehnologii Java de programare a aplicatiilor Web (EE) Java

- 4.1. Tehnologii client. Miniaplicatii Java (applet-uri)
- 4.2. Clase pentru interfete grafice cu utilizatorul (AWT, Swing)
- 4.3. Platforma Java EE. Arhitectura si tehnologiile implicate
- 4.4. Tehnologii server. Tehnologia Java Servlet
- 4.5. Tehnologia Java ServerPages (JSP)
- 4.6. Accesul la baze de date prin tehnologii Java (JDBC, Hibernate)
- 4.7. Tehnologii avansate (frameworks, componente EJB, Servicii Web)



Structura cursului

2. Introducere in tehnologiile desktop Java

2.1. Elemente de baza. Tipuri de date referinta. Clase de biblioteca

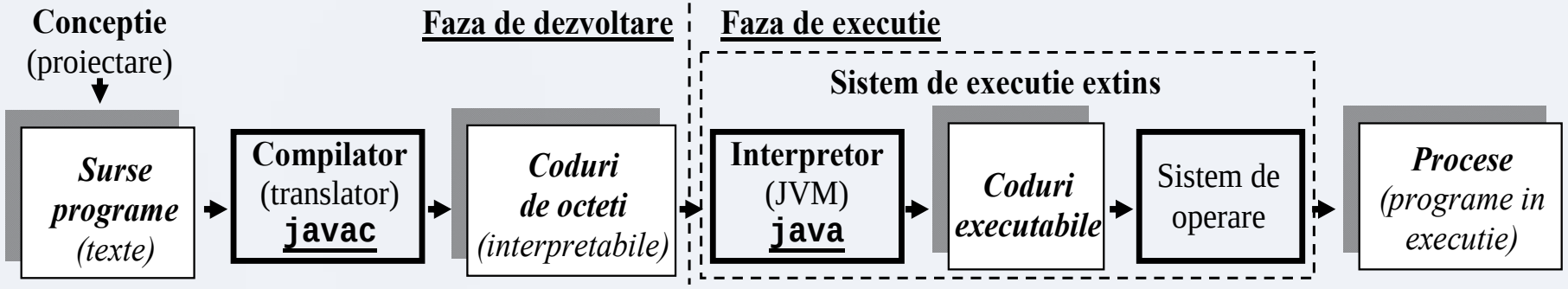
Elemente de baza ale programarii in limbajul Java

Dezvoltarea programelor Java

In dezvoltarea **programelor Java**

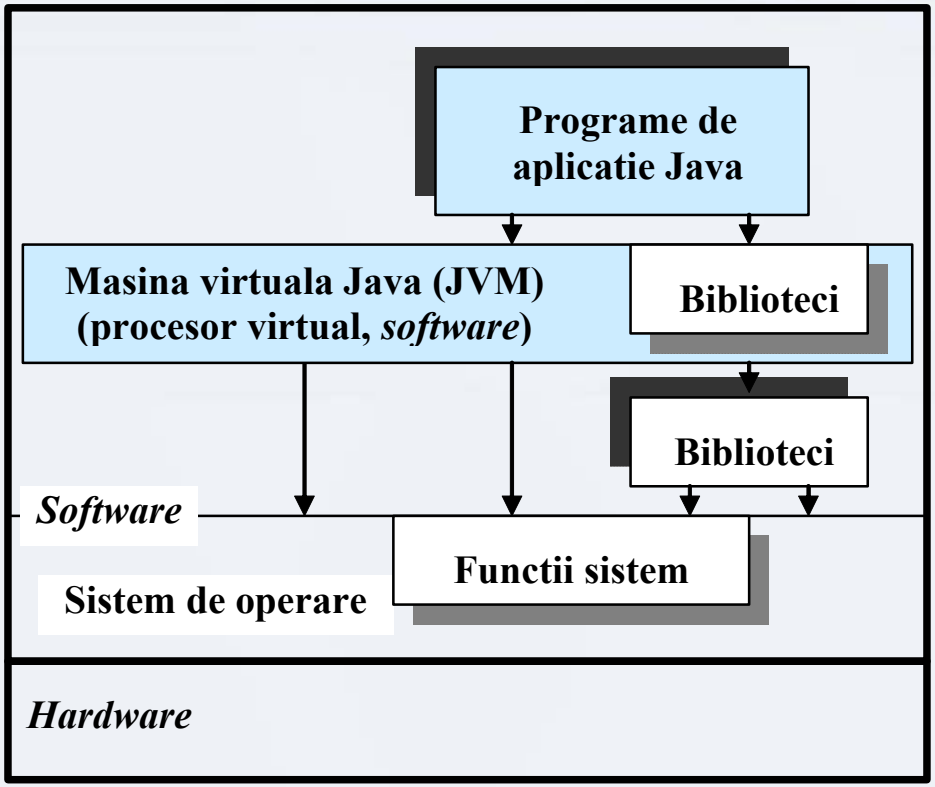
- *codurile sursa* sunt ***compilate*** (translatate) de la limbajul Java la coduri **executabile pe procesorul software Java** (JVM = *Java Virtual Machine*), numite coduri de octeti (*bytecodes*),

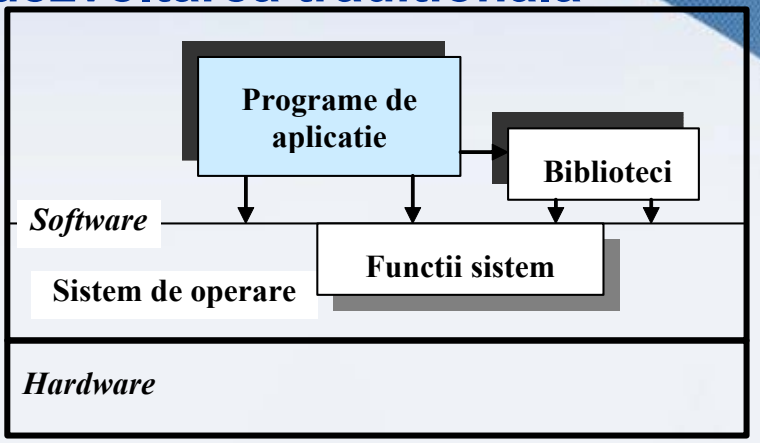
- apoi *codurile de octeti* sunt ***interpretate*** (executate de interpretorul Java, parte din JVM - **procesorul virtual** care face apelurile catre sistemul de operare)

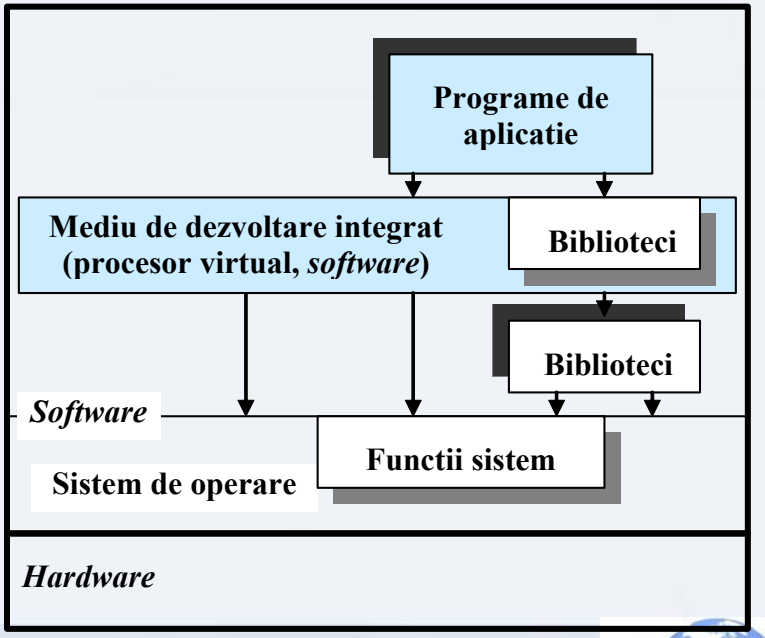


2. Introducere in tehnologiile Java SE

Dezvoltarea programelor Java vs. dezvoltarea traditionala

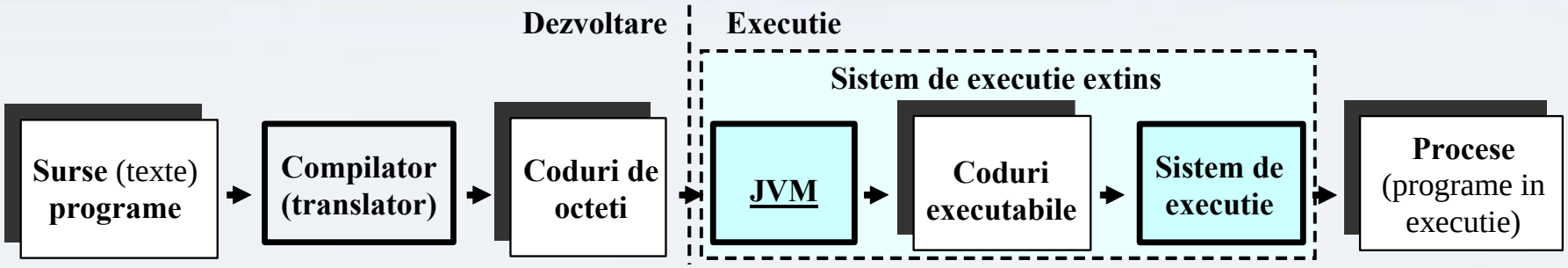




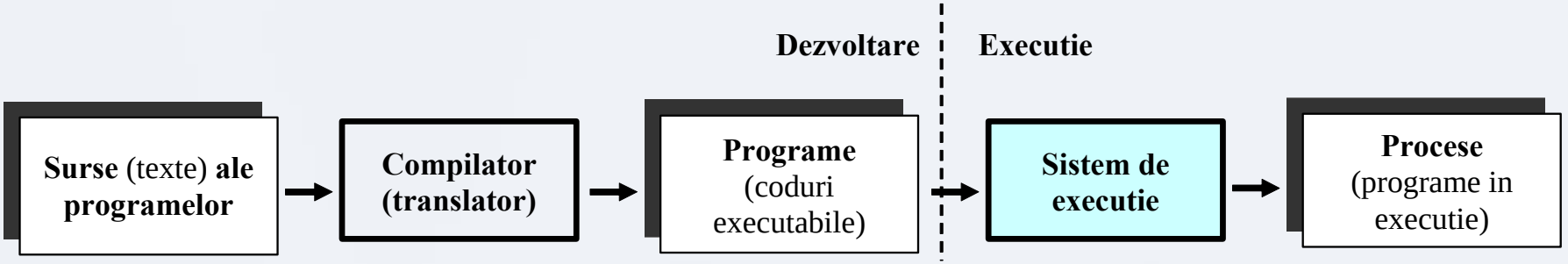


Dezvoltarea programelor Java vs dezvoltarea traditionala

Cazul unui program **Java**

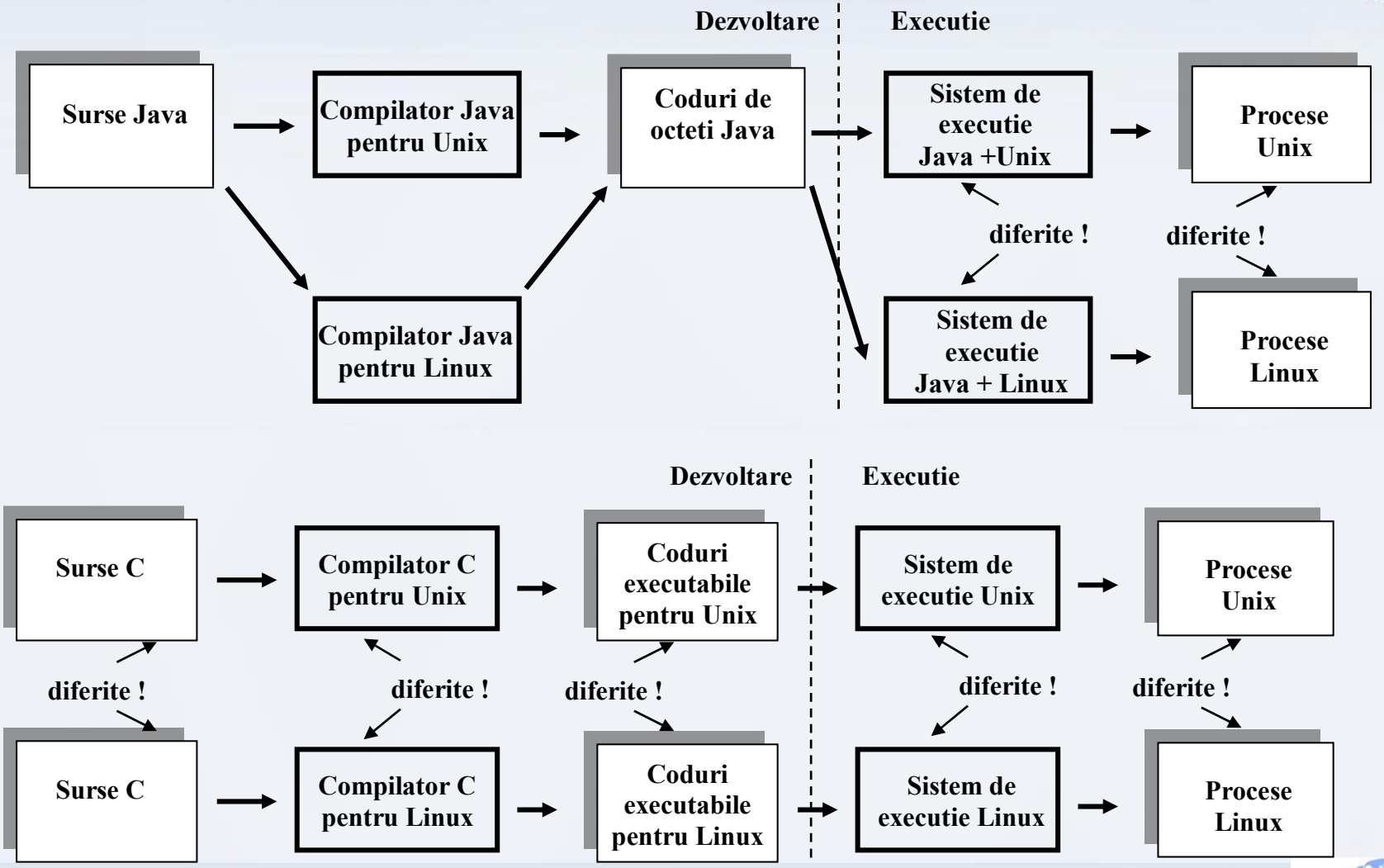


Cazul unui program **C, C++, Pascal, etc.**



2. Introducere in tehnologiile Java SE

Dezvoltarea programelor Java vs dezvoltarea traditionala



Date si variabile in Java

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

```
1  int suma;              // declaratia (tipului) variabilei - cod Java
2  suma = 0;              // initializarea variabilei
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)

O variabila are:

- **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, 10, 15, 21, 28, 36, 45, 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**

Tipul de date **specifica structura variabilelor** si **domeniul de definitie al valorilor:**

- **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

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:
//   b = i;
//   s = i;
// Nu genereaza eroare:
l = i;
// Dar genereaza eroare:
//   i = l;
```

```
// Coduri corectate:
b = (byte) i;
s = (short) i;
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:
// float f = 2.0;
// f = d;
d = f;
f = i;
// Ar genera eroare:
// l = f;
```

```
// Coduri corectate:
float f = 2.0f;
f = (float) d;


l = (long) f;
```

Exemplu introductiv / recapitulativ

```
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 are *caracter de obiect*)

`void`: specifica faptul ca metoda nu returneaza nimic

Exemplu introductiv / recapitulativ

```
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*, globala la nivelul clasei **`Salut`** (nu cu *caracter de obiect*)

`void`: metoda `main()` nu returneaza nimic

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**

Exemplu introductiv / recapitulativ

```
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: "**;**").

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
%	Modulo	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 binari 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.
//		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*.

Exemplu introductiv / recapitulativ

```
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)

Tablourile in Java

Tipuri referinta in Java

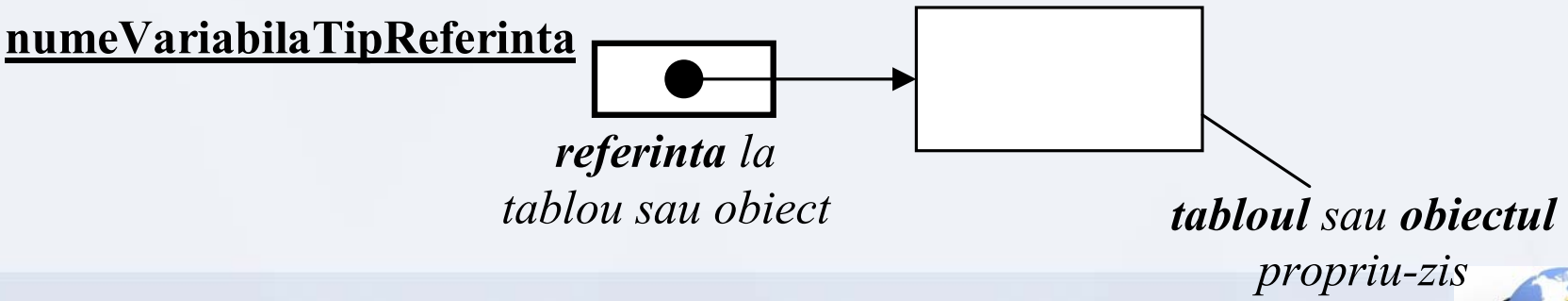
- tipul **tablou**
- tipul **clasa**
- tipul **interfata**

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)

  - **are acces doar la continutul tablourilor / obiectelor referite**

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

- o valoare pentru referinte este si **null**, semnificand **referinta "catre nimic"**

  - **simpla declarare a variabilelor referinta** conduce la **initializarea implicita** a referintelor **cu** valoarea **null**

referinta *catre nimic*

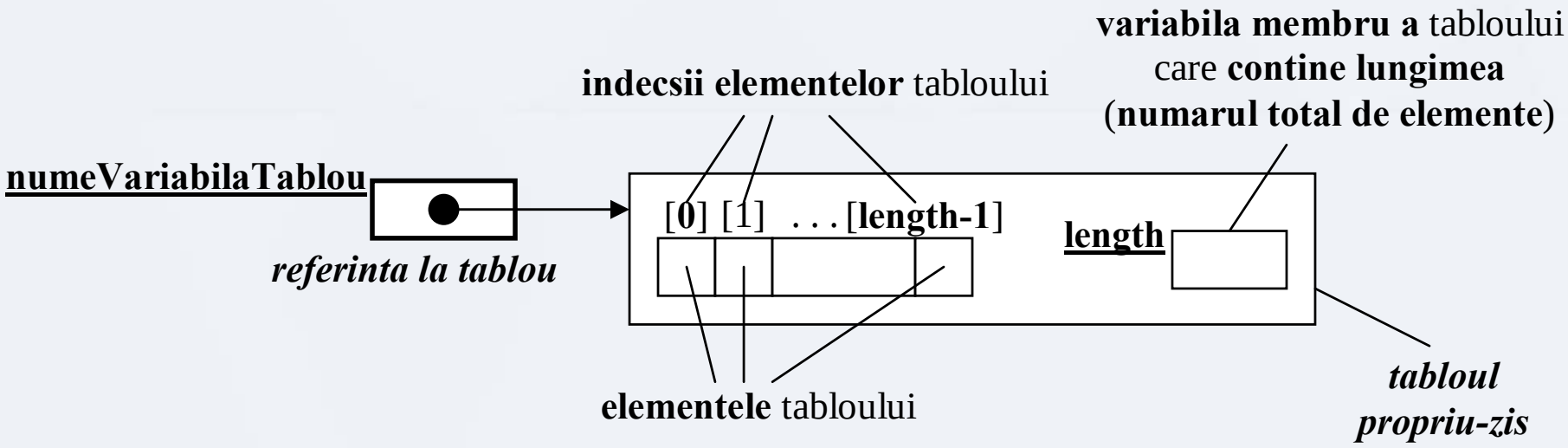
Tablourile in Java

Tabloul Java

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

Lungimea tabloului

- are **valoare fixa** stabilita in momentul crearii tabloului (cu operatorul **new**)
- este o **variabila membru** a tabloului



Cum arata un tablou de 5 elemente de tip int?

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 }; // declarare, alocare memorie si populare
```

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
```

Pentru **conversia unei valori de la `String` la `int`** se foloseste sintaxa:

```
// Conversia unei valori de la tip String la tip int
int numarStudenti = Integer.parseInt("25");
```

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]); // conversie
9         }                                                // String -> int
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  }
```

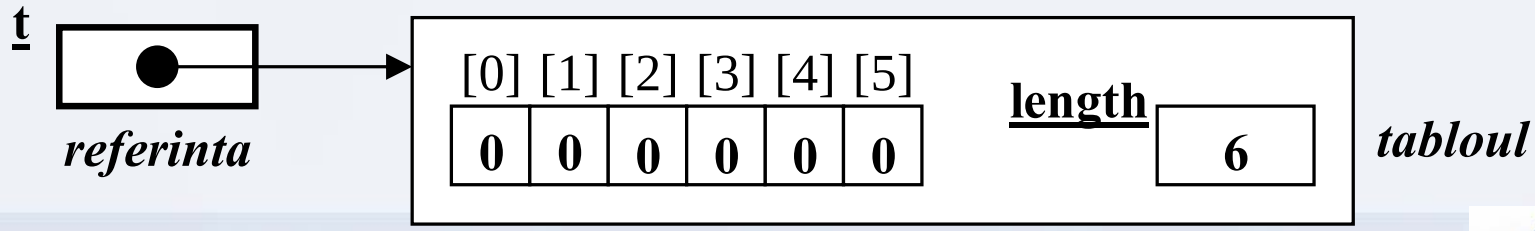
```
directorcurent> javac SumaArgumenteIntregi.java
directorcurent> java SumaArgumenteIntregi 12 31 133 -10
Au fost primite 4 argumente
Suma valorilor primite este 166
directorcurent>
```

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
```

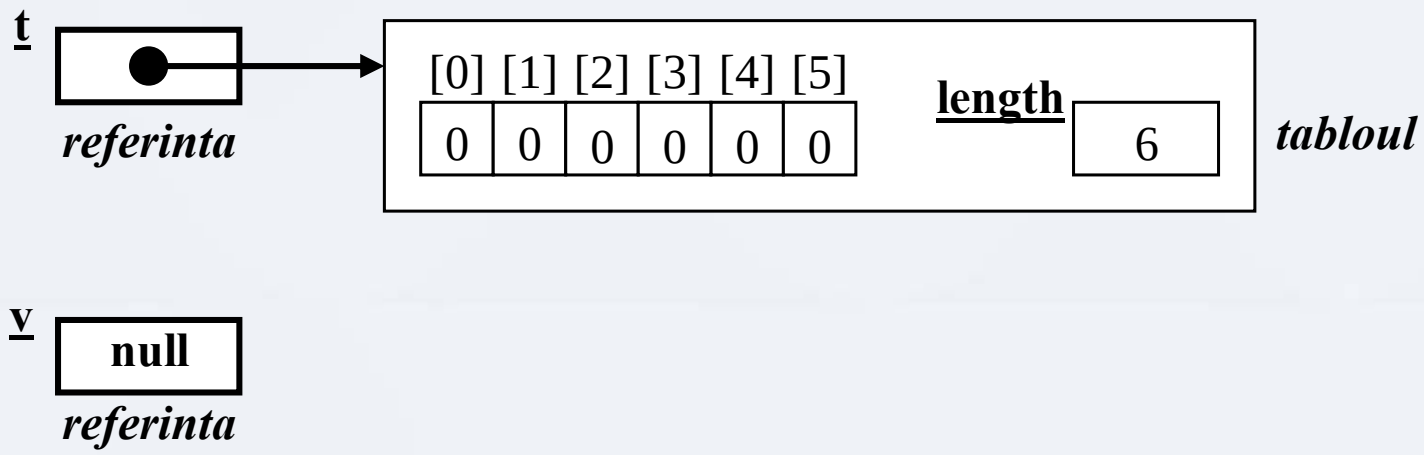
Dupa executia liniei 2:



Tablourile in Java

Dupa executia liniei 3:

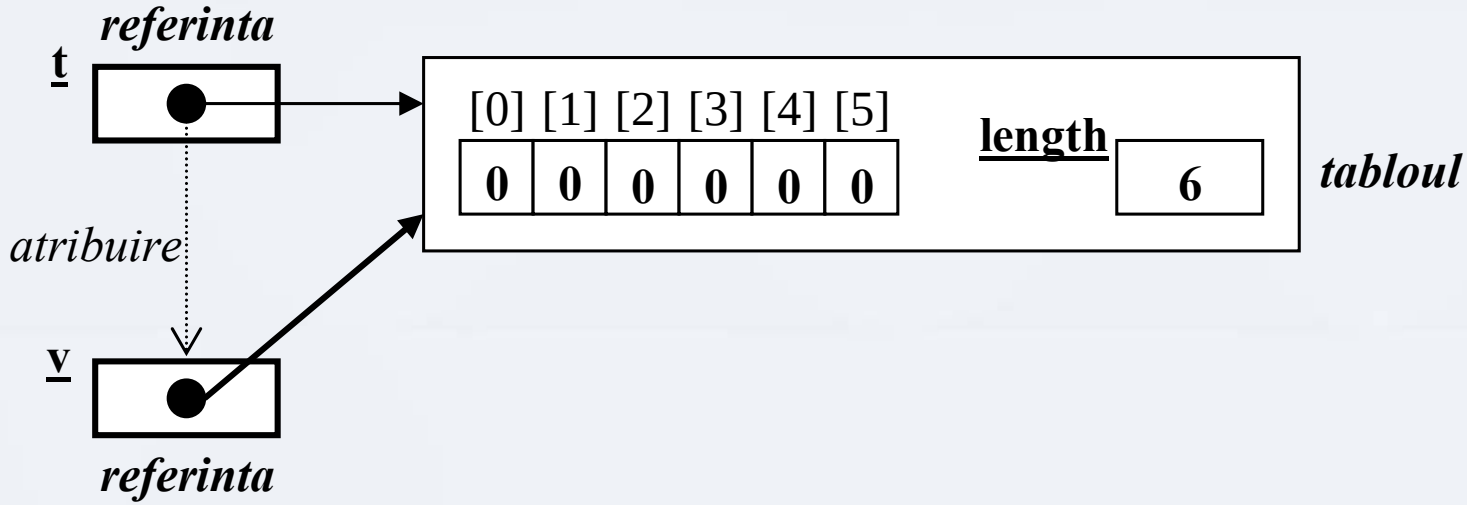
```
3   int[] v;                    // declarare simpla
```



Tablourile in Java

Dupa executia liniei 4:

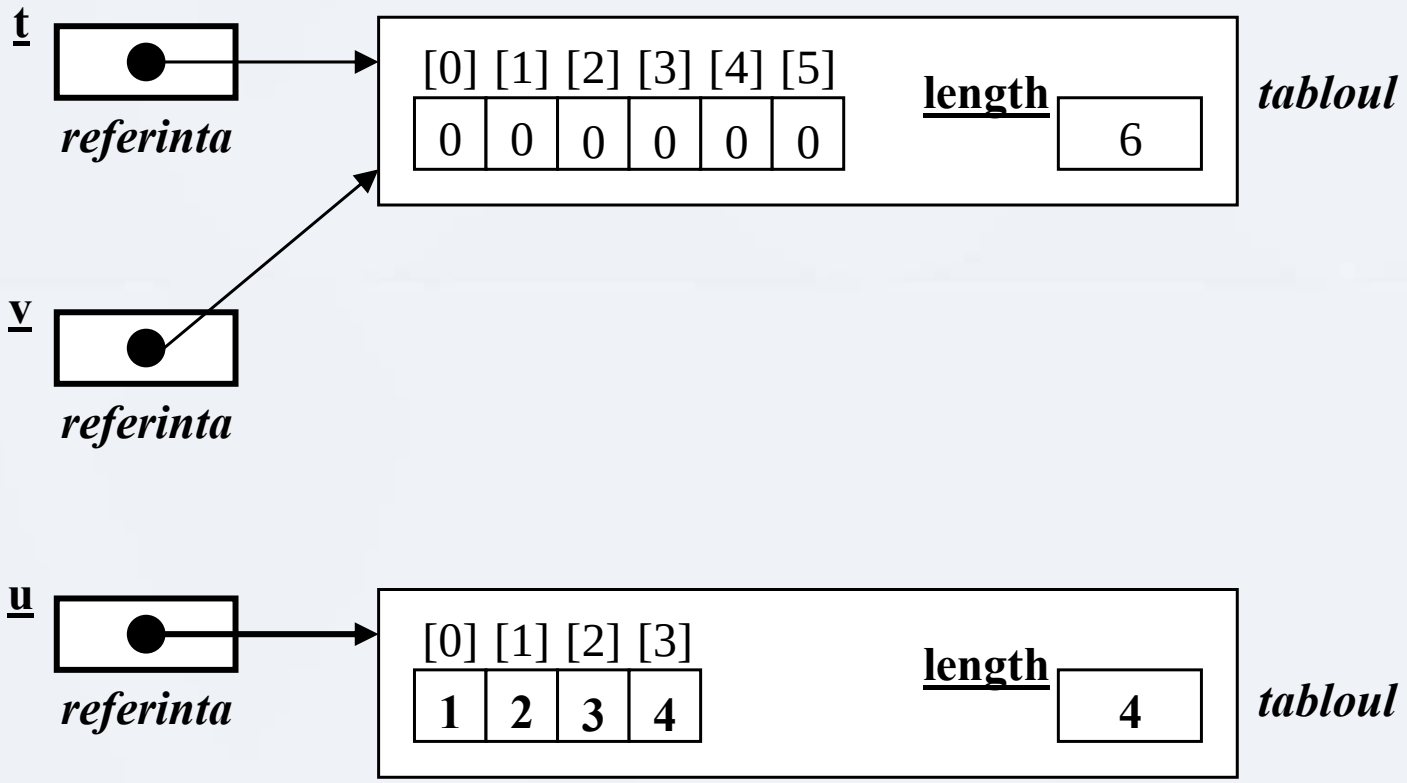
```
4    v = t;                    // copiere referinte
```



Tablourile in Java

Dupa executia liniei 5:

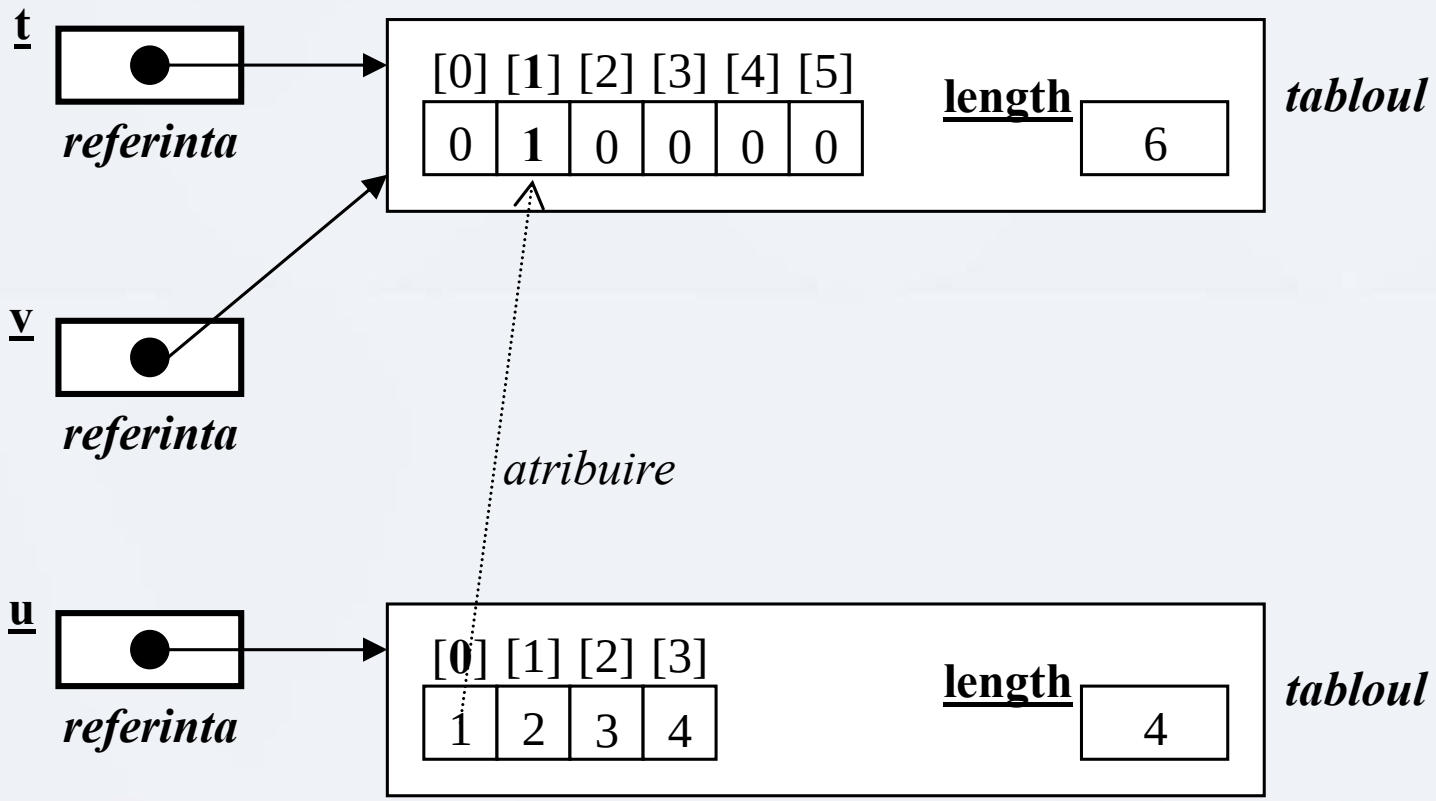
```
5   int[] u = {1,2,3,4}; // declarare+alocare+initializare
```



Tablourile in Java

Dupa executia liniei 6:

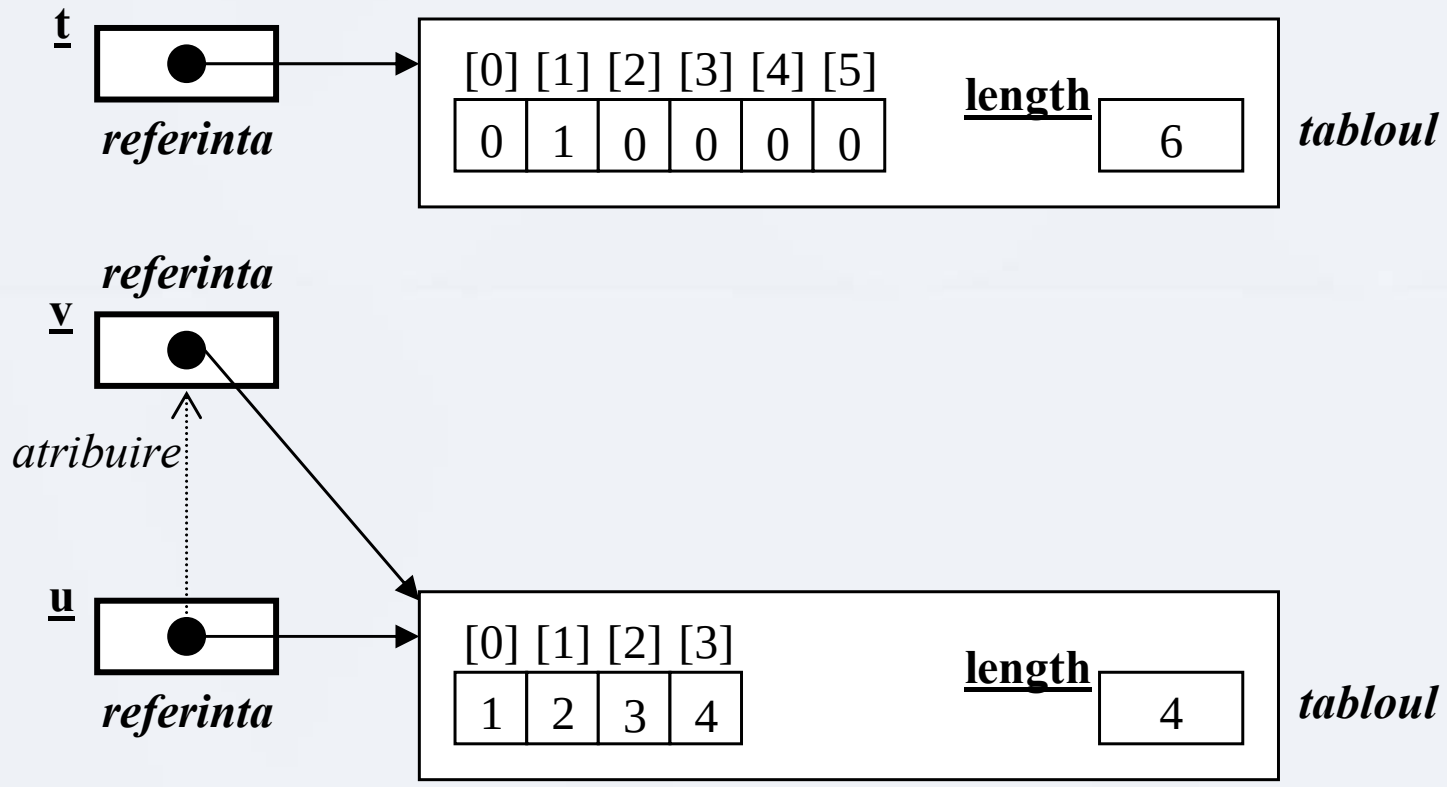
```
6    t[1] = u[0];                    // atribuire intre elemente
```



Tablourile in Java

Dupa executia liniei 7:

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



Tablourile in Java

Determinarea anilor bisecti (divizibili cu 4) – **3 variante, doar una corecta**

```
// Obtinerea anului testat printr-o fereastra grafica
int anulTestat = Integer.parseInt(JOptionPane.showInputDialog("Anul:"));


// Verificarea divizarii cu 4 - Varianta folosita in limbajul C
//   if (!(anulTestat % 4)) System.out.println("Anul testat este bisect");
//   In Java genereaza eroare:  (incompatible types
//                               found   : int
//                               required: boolean)
//   Expresia din paranteza trebuie sa fie tip boolean


// Verificarea divizarii cu 4 - Varianta corecta in limbajul Java
if ((anulTestat % 4)==0) System.out.println("Anul testat este bisect");


// Varianta incorecta in limbajul Java
//   if ((anulTestat % 4)=0) System.out.println("Anul testat e bisect");
//   In Java genereaza eroare:  (incompatible types
//                               required: variable
//                               found   : value)
//   Trebuie folosit "==" in loc de "="
```

Tablourile in Java

Pentru **a obtine de la utilizator o valoare** de tip **sir de caractere (`String`)** in timpul executiei programului, printr-o fereastra grafica, se poate folosi sintaxa:

```
// Obtinerea numelui utilizatorului folosind fereasta de dialog
String nume = JOptionPane.showInputDialog("Introduceti numele dvs:");
```

Determinarea anilor bisecti - varianta cu structura de program `if..else`

```
// Verificarea divizarii cu 4 - Varianta corecta in limbajul Java
if ((anulTestat % 4)==0)
    System.out.println("Anul testat este bisect");
else
    System.out.println("Anul testat nu este bisect");
```

Totusi, determinarea anilor bisecti ca fiind **divizibili cu 4 nu este corecta din punct de vedere conceptual.** Varianta corecta presupune determinarea anilor **divizibili cu 4 dar nu cu 100, sau divizibili cu 400**

Tablourile in Java

Determinarea anilor bisecti (divizibili cu 4 dar nu cu 100, sau divizibili cu 400)

```
// Verificarea divizarii cu 4
if ((anulTestat%4)==0)                    // multiplu de 4

   // Verificarea divizarii cu 100
   if ((anulTestat%100)==0)               // multiplu de 100

      // Verificarea divizarii cu 400
      if ((anulTestat%400)==0)            // multiplu de 400
          System.out.println("Anul testat este bisect");

      else                                // nu e multiplu de 400, ci e de 100
          System.out.println("Anul testat nu este bisect");

   else                                   // nu e multiplu de 100, ci e de 4
      System.out.println("Anul testat este bisect");

else                                      // nu e multiplu de 4
   System.out.println("Anul testat nu este bisect");
```

Cum se poate rescrie codul cu mai putine structuri de program if...else?

Metodele (functiile membru) 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 pentru modularizarea sarcinilor) - catre o metoda statica fara parametri

```
// 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
 }
}
```

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
// incadrate si separate prin linii orizontale formate din 50 caractere
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
 }
}
```

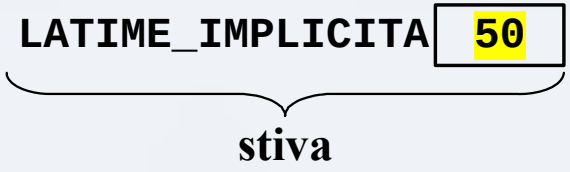
Care sunt parametrii (variabilele folosite in definitiile functiilor) si care sunt argumentele (valorile pasate in apelurile functiilor)?

Functii - parametri si argumente

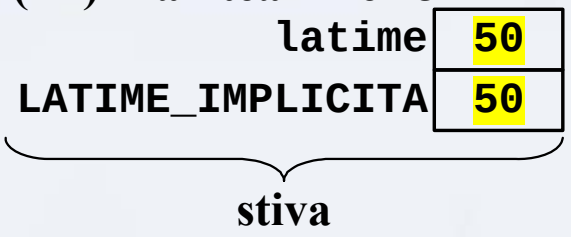
Starile succesive ale stivei in versiunea Raport03

(I) Inaintea liniei 6

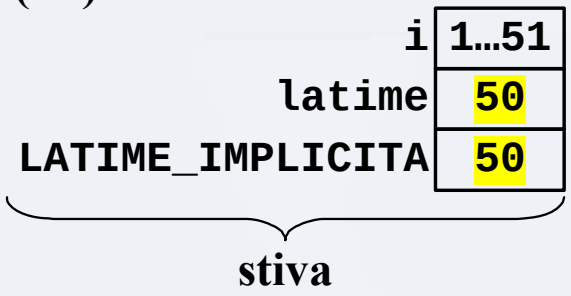
(II) Inaintea liniei 8



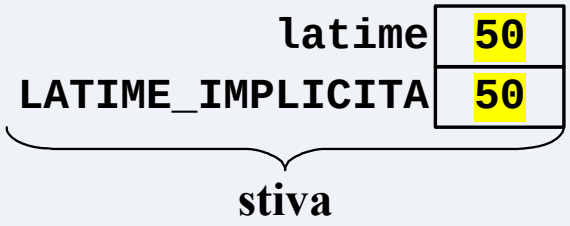
(III) Inaintea liniei 3



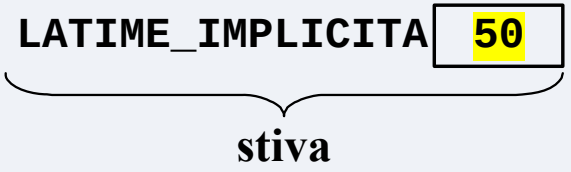
(IV) Inaintea liniei 4



(V) Inaintea liniei 5



(VI) Inaintea liniei 9



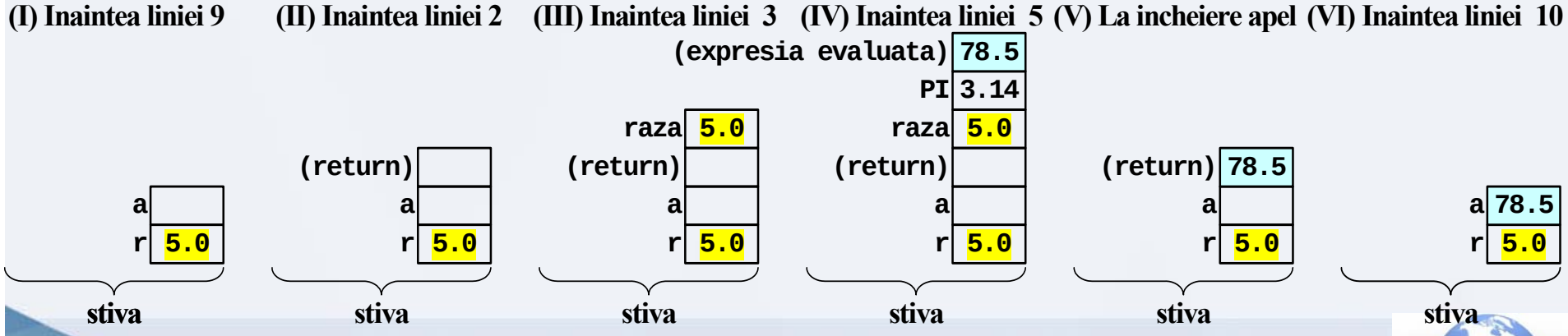
In Java - **stiva** (*stack*) contine **variabilele de tip primitiv** (byte, double, char, etc,) si **referintele la tablouri/obiecte**

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

Functii - returnarea unor valori

```
1   // Program care calculeaza aria unui cerc in functie de raza
2   public class Cerc {
3     private static double arie(double raza)       // definitia metodei
4        final double PI = 3.14159;                 // variabila finala
5        return PI * raza * raza;                   // returnarea unei valori
6     }
7     public static void main(String[] args) {
8        double r = 5.0;                            // variabila locala r
9        double a;                                  // variabila locala a
10       a = arie(r);                               // apelul metodei
11       System.out.println("Un cerc de raza " + r + " are aria " + a + ".");
12    }
13  }
```

Starile succesive ale stivei



Pasarea argumentelor prin valoare (copia valorii primite)

Cazul pasarii unei valori primitive

```
public class C1 {                  // Program care incrementeaza o valoare intreaga
  public static void inc(int i) {  // declaratie (semnatura) metoda inc()
    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 (sau argument)
  }                                         // Rezultat: x = 10
}
```

Cazul pasarii unui tablou

```
public class C2 {       // Program care incrementeaza un element al unui tablou
  public static void inc(int[] i) { // primeste o copie a referintei cu aceeasi
                                    // valoare, asa incat refera acelasi tablou
    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 referinta (valoarea ei)
    System.out.println("x[0] = " + x[0]); // Rezultat: x[0] = 11
  }
}
```

Structuri de control al programului

Structuri de control al programului – decizie/selectie 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.");
}
```

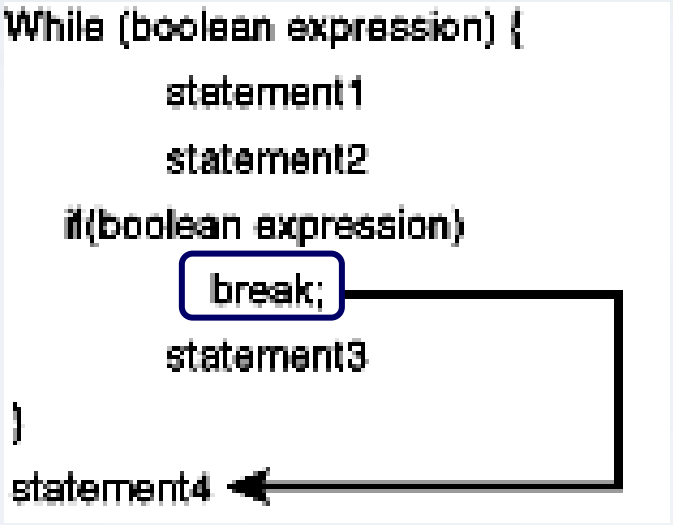
Structuri de control al programului – iteratii (bucle)

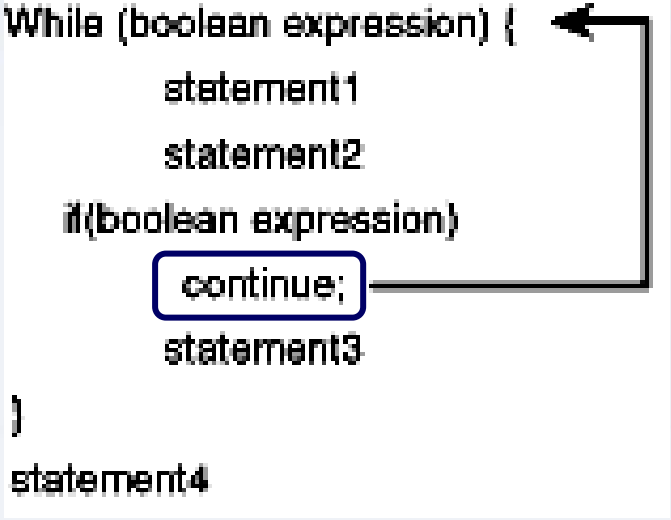
```
// repetare cat timp <expresieBooleana> == true
for (<initializare>; <expresieBooleana>; <actualizare>)
   <instructiuneExecutataRepetat>

// 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





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>**

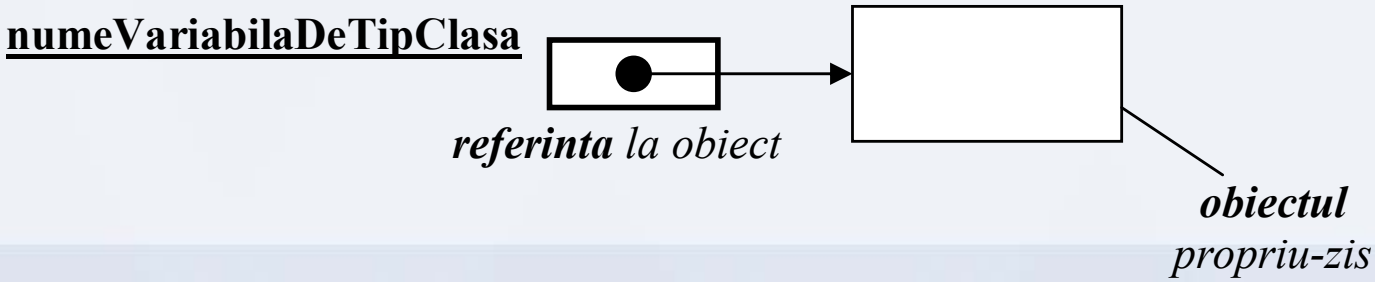
In Java se folosesc **break <eticheta> si respectiv continue <eticheta>**

```
outsideLoop: for( ... ) {
   ...
   while( ... ) {
      ...
      if (<condition1>) {
         ...
         break outsideLoop;
      } // end if

      if (<condition2>) {
         ...
         continue outsideLoop;
      } // end if
      ...
   }  // end while
   ...
}  // end for
```

Clase si obiecte Java

Clasa

- **tip de date**
  - **tipar** dupa care sunt construite variabile numite **obiecte** dar si
  - **domeniu de definitie** (asemanator unei **multimi**) pentru obiecte
- **structura complexa** care **reuneste**
  - **elemente de date**, numite - **atribute** in orientarea spre obiecte
    - **campuri** in Java, si
  - **algoritmi**, numiti - **operatii** in orientarea spre obiecte
    - **metode** in Java
- **tip referinta** in Java - **obiectele** sunt **accesate** printr-o **referinta**, care contine **adresa obiectului propriu-zis**

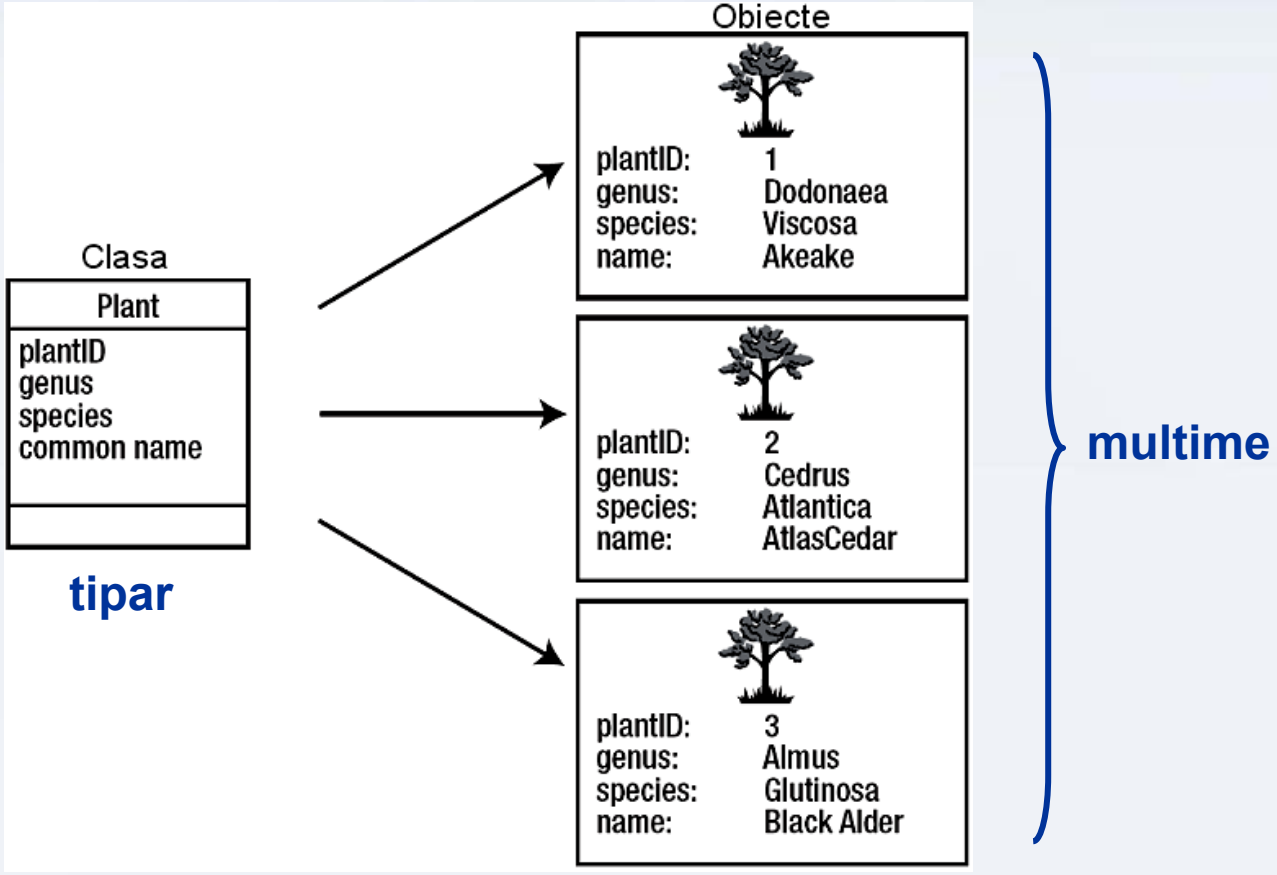


Obiectul

- **reprezentare abstractă** a unor **entități reale** sau **virtuale**
- caracterizată de:
  - **identitate**, prin care acesta este deosebit de alte obiecte
    - implementata in Java ca variabila **referinta catre obiect**
  - **comportament** accesibil altor obiecte
    - implementat ca set de **operatii** (**metode,** functii membru)
  - **stare internă** ascunsă, proprie
    - implementata ca set de **atribute** (**campuri,** variabile membru)
- este un **exemplu specific** al unei clase, numit **instanta** a clasei

Clasa si obiectele

Obiectul este **exemplu specific** al unei clase, numit **instanta** a clasei



Clasa este

- **tipar** dupa care sunt construite variabile numite **obiecte**
- **domeniu de definitie** (asemanator unei **multimi**) pentru obiecte

Clasa

Definitia clasei in Java (simplificata)

```
public class Nume {  // declaratie tip/structura de date

   // declaratie atribut (variabila membru, camp Java)
   tip atribut;

   // semnatura operatie (metoda Java)
   tipReturnat operatie(tipParametru parametruFormal) {

      // corpul functiei membru (metodei)
      // returneaza valoare de tipul tipReturnat
    }
}
```

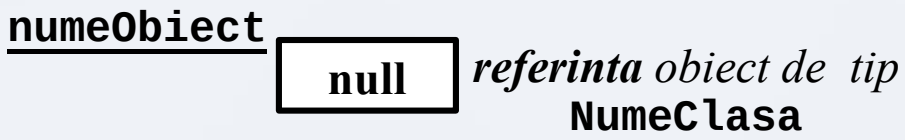
In UML clasa definita mai sus se reprezinta astfel

Nume
atribut : tip
operatie(parametruFormal : tipParametru) : tipReturnat

Obiectul

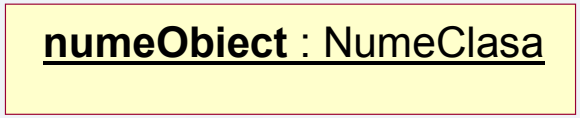
Pentru a trata distinct obiectele e necesara utilizarea unor **nume** diferite, care in Java sunt **variabile referinta** la/catre obiecte

```
// declararea variabilei referinta la obiect
NumeClasa  numeObiect;
```



Prin **simpla declarare** se **creaza doar spatiul necesar variabilei referinta**, care are valoarea implicita **null** (referinta catre nimic)

In UML obiectul declarat mai sus se reprezinta astfel

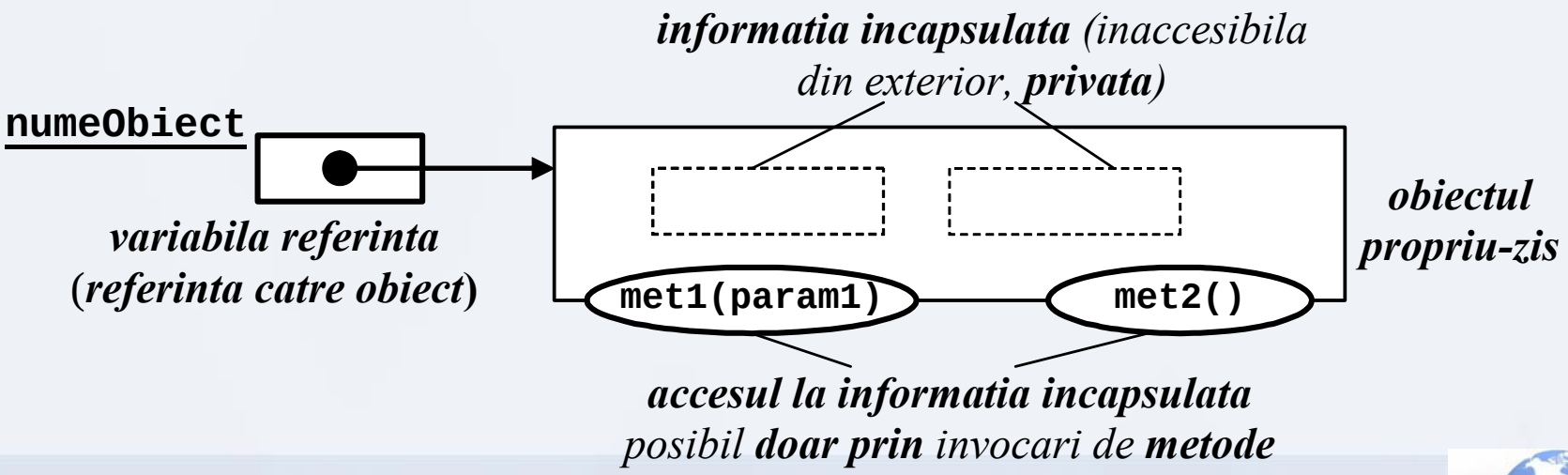


Obiectul

Alocarea memoriei pentru obiect si **atribuirea referintei catre acesta** se realizeaza cu ajutorul

- **constructorului clasei** obiectului si al
- operatorului **new** de alocare dinamica a memoriei

```
// crearea dinamica a obiectului
numeObiect = new NumeClasa(tipParametru parametruActual);
```



Clasa si obiectele

Constructorul Java este un tip special de **functie**, care

- are **acelasi nume cu numele clasei** in care este declarat,

- este **utilizat pentru a initializa orice nou obiect** de acel tip (stabilind valorile campurilor/ atributelor obiectului, in momentul crearii lui dinamice),

- **nu returnează** nici o valoare,

- are **aceleasi**

  - **niveluri de accesibilitate**,

  - **reguli de implementare** a corpului si

  - **reguli de supraincarcare a numelui**

ca si metodele obisnuite

```
NumeClasa(tipParametru parametruActual);
```

Clasa si obiectele

Constructorul

- **stabileste valorile initiale** ale tuturor **atributelor** unui **obiect nou**
- **ducand** astfel **obiectul** in **starea initiala**

In Java nu este neaparat necesara scrierea unor constructori pentru clase, deoarece

- **un constructor implicit** este **generat automat** de sistemul de executie (DOAR) **pentru o clasa care nu declara explicit** constructori

Acest constructor **nu face nimic** (nici o initializare, implementarea lui continand un bloc de cod vid: { })

```
NumeClasa() {}
```

De aceea

- **orice initializare dorita explicit impune** scrierea unor **constructori**

Exemplu de clasa

```
public class Point {
   // atribute (variabile membru)
   private int x;
   private int y;

   // operatie care initializeaza atributele = constructor Java
   public Point(int abscisa, int ordonata) {
      x = abscisa;
      y = ordonata;
   }

   // operatii care modifica atributele = metode (functii membru)
   public void moveTo(int abscisaNoua, int ordonataNoua) {
      x = abscisaNoua;
      y = ordonataNoua;
   }
   public void moveWith(int deplasareAbsc, int deplasareOrd) {
      x = x + deplasareAbsc;
      y = y + deplasareOrd;
   }
   // operatii prin care se obtin valorile atributelor = metode
   public int getX() {  return x;  }
   public int getY() {  return y;  }
}
```

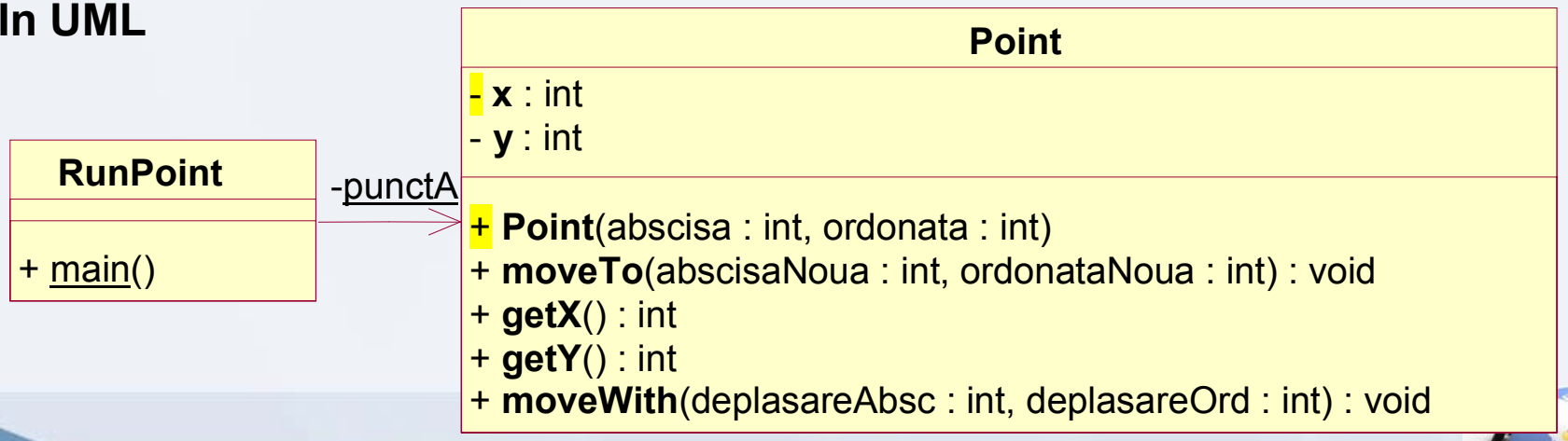
Declaratii (specificare) **atribute**

Semnaturi (declaratii, specificari) **operatii** + **Implementari** (corpuri) **operatii**

Exemplu de clasa utilizator pentru clasa anterioara

```
// clasa de test pentru clasa Point
public class RunPoint {
   private static Point punctA;                    // atribut de tip Point

   public static void main(String[] args) {        // declaratie metoda
      // corp metoda
      punctA = new Point(3, 4);  // alocare si initializare atribut punctA

      punctA.moveTo(3, 5);       // trimitere mesaj moveTo() catre punctA

      punctA.moveWith(3, 5);     // trimitere mesaj moveWith() catre punctA
   }
}
```

In UML



Exemplu de creare si utilizare a unui obiect

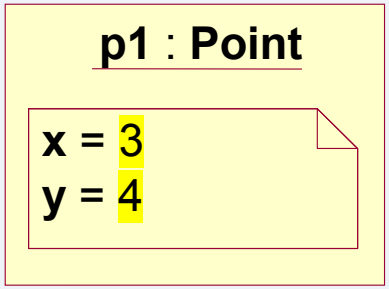
Crearea unui obiect

- numit **p1** de tip **Point**
- ale carui **atribute** au **valorile x=3** si respectiv **y=4**

```
// crearea dinamica si initializarea obiectului
Point p1 = new Point(3, 4);
```

Obiectul numit **p1**

- **abstractizeaza si**
- **incapsuleaza informatiile** care privesc un **punct in plan** de coordonate **{3, 4}**



Starea initiala a obiectului **p1** este perechea de coordonate **{3, 4}**

Exemplu de creare si utilizare a unui obiect

Schimbarea starii obiectului **p1** din **{3, 4}** in **{3, 5}**

- **prin deplasarea ordonatei** (departarea cu 1 de abscisa)

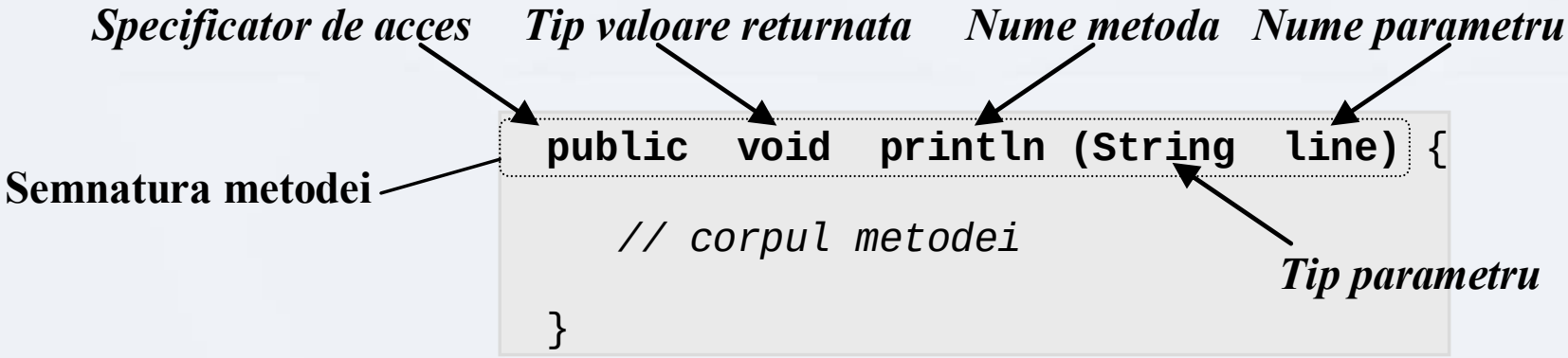
```
// crearea dinamica si initializarea obiectului
Point p1 = new Point(3, 4);

// schimbarea starii obiectului
p1.moveTo(3, 5);
```

Metode (operatii) si campuri (atribute) Java

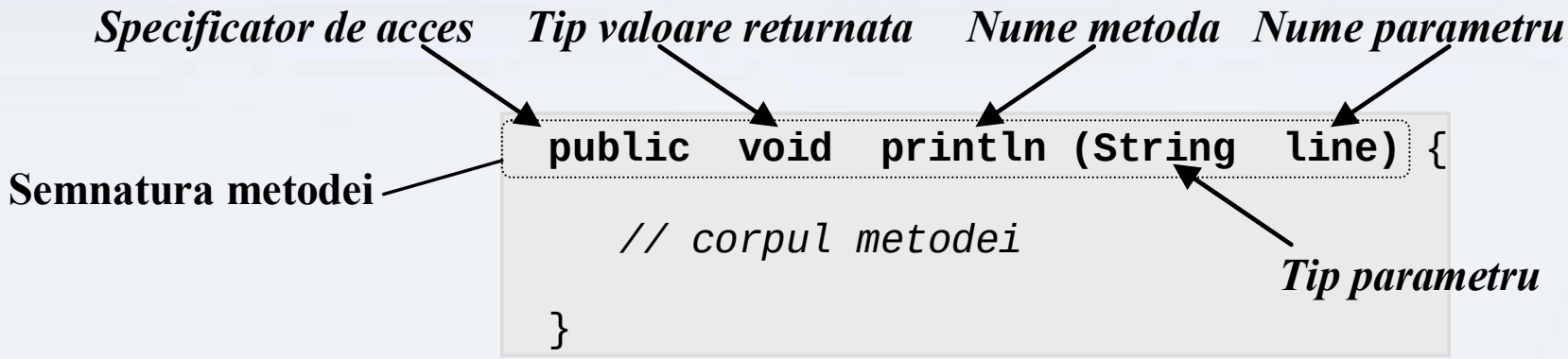
Operatiile (functiile membru, metodele Java)

Definitia unei metode contine 2 parti:

- **semnatura** (antetul, declaratia) si
- **corpul** (blocul, segmentul, secventa de instructiuni a implementarii)



Operatiile (functiile membru, metodele Java)



Semnatura metodei specifica:

- **numele** metodei

- lista de **parametri formali** (numarul, ordinea, tipul si numele lor)

- **tipul valorii returnate** (daca **nu** returneaza nici o valoare, **tipul** valorii returnate este declarat **void**)

- **specificatori** ai unor proprietati explicite (**modificatori** ai proprietatilor implicite)

Operatiile (functiile membru, metodele Java)

Tipul valorii returnate poate fi

- unul dintre **cele 8 tipuri primitive** Java (byte, short, int, long, float, double, boolean si char), sau
- unul dintre **cele 3 tipuri referinta** (tablourile, clasele si interfetele Java).

Corpul metodei

- contine **secventa de instructiuni** care
- specifica **pasii necesari indeplinirii sarcinilor** (evaluari expresii, atribuiri, decizii, iteratii, apeluri metode)

Returnarea valorilor

- este **specificata in codul metodelor** prin instructiunea **return** urmata de o **expresie**
- care poate fi **evaluata la o valoare** de **tipul declarat** in semnatura

Operatiile (functiile membru, metodele Java)

Declaratiile (semnaturile) metodelor

- pot include **liste** de declaratii de **parametri**

Parametrii

- specifica **valorile de intrare** necesare metodelor pentru a fi executate
- sunt **variabile care au ca scop intregul corp al metodei**
- declarate ca orice variabila, folosind formatul **tipVariabila numeVar**
- **se numesc parametri formali** sau simplu **parametri**

Apelurile metodelor

- pot include **liste** de **valori date parametrilor**
- valori care **trebuie sa corespunda ca tip celor declarate**

Valorile pasate metodelor in momentul apelurilor

- se numesc **parametri actuali** sau simplu **argumente**

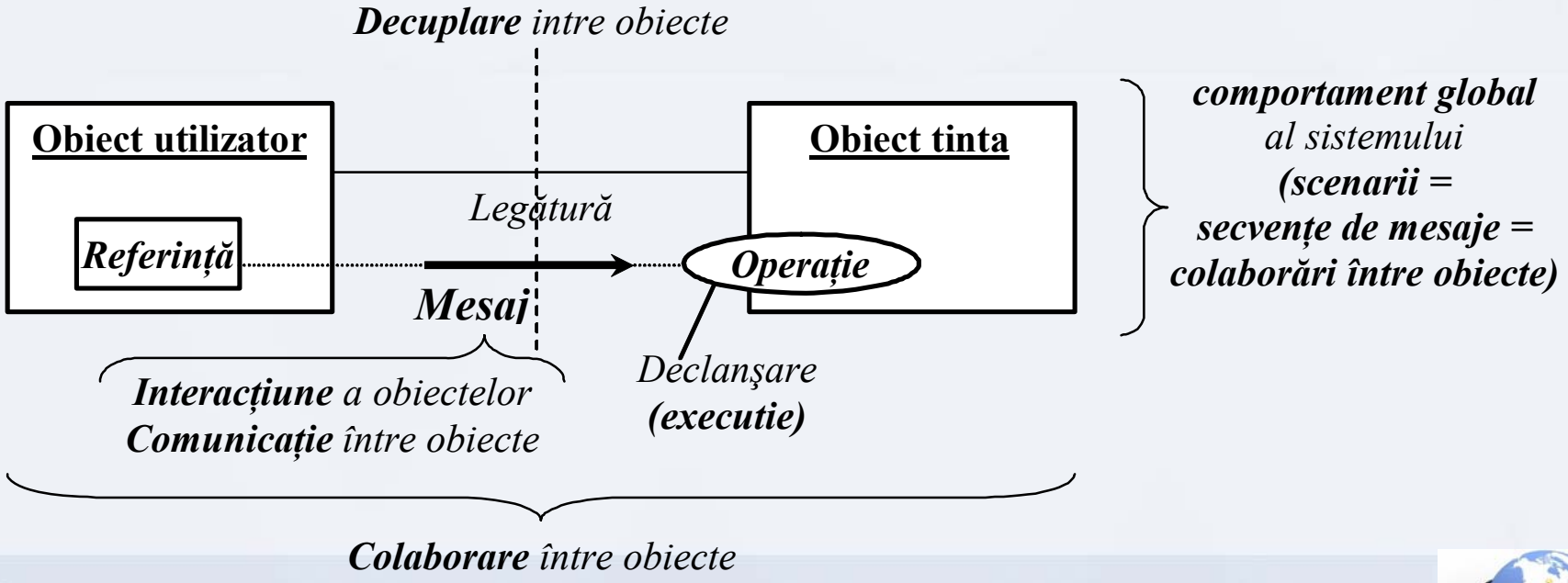
Operatiile vazute ca mesaje schimbate intre obiecte

Obiectele comunica prin mesaje

- **mesajul** fiind **apelul** (invocarea) **metodei altui obiect**

Mesajul

- realizeaza **cuplajul dinamic**
- **intre obiecte** (care sunt **create** la randul lor **dinamic**)



Reluarea problemei pasarii argumentelor prin valoare

Cazul pasarii unei valori primitive

```
public class C1 {                  // Program care incrementeaza o valoare intreaga
  public static void inc(int i) {  // declaratie (semnatura) metoda inc()
    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 (sau argument)
  }                                        // Rezultat: x = 10
}
```

Cazul pasarii unui tablou

```
public class C2 {       // Program care incrementeaza un element al unui tablou
  public static void inc(int[] i) { // primeste o copie a referintei cu aceeasi
                                    // valoare, asa incat refera acelasi tablou
    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 referinta (valoarea ei)
    System.out.println("x[0] = " + x[0]); // Rezultat: x[0] = 11
  }
}
```

Reluarea problemei pasarii argumentelor prin valoare

Cazul pasarii unui obiect care contine un camp public (accesibil oricarui cod exterior) – caz in care se poate vorbi de **"lucrul cu" obiecte!**

```
// Program care incrementeaza un camp (atribut) public al unui obiect
public class C3 {

  public static void inc(ClasaInt i) { // primeste o copie a referintei cu
                                       // aceeasi valoare, asa incat refera acelasi obiect
    i.camp++;                          // e incrementat campul continut in obiect
  }

  public static void main(String[] args) {
     ClasaInt x = new ClasaInt();//obiect referit de x,continand camp tip int
     x.camp = 10;                // initializat cu valoarea 10
     inc(x);                     // este pasata referinta (valoarea ei)
     System.out.println("x.camp = " + x.camp);  // Rezultat: x.camp = 11
   }
}

class ClasaInt {   // Clasa noua care ofera un atribut public
  public int camp;
}
```

Reluarea problemei pasarii argumentelor prin valoare

Cazul pasarii unui obiect care contine un camp privat (inaccesibil oricarui cod exterior) **si metode de acces** – caz in care vorbim de **"orientare spre" obiecte!**

```
// Program care incrementeaza un camp (atribut) private al unui obiect
public class C4 {

  public static void inc(ClasaInt i) { // primeste o copie a referintei cu
                                       // aceeasi valoare, asa incat refera acelasi obiect
    i.setCamp(i.getCamp()+1); // e incrementat campul incapsulat in obiect
  }

  public static void main(String[] args) {
    ClasaInt x = new ClasaInt(); //obiect referit de x,continand camp tip int
    x.setCamp(10);               // initializat cu valoarea 10
    inc(x);                      // este pasata referinta (valoarea ei)
    System.out.println("x.getCamp()= " + x.getCamp()); // Rez: x.getCamp()=11
  }
}

class ClasaInt { // Clasa noua care ofera un atribut privat si metode publice
  private int camp;
  public void setCamp(int c) { camp = c; }
  public int getCamp() { return camp; }
}
```

Structura unei clase Java

Structura unei clase Java

```
 1  import java.util.Vector;                    // clase importate
 2  import java.util.EmptyStackException;
 3  public class Stack                 // declaratia clasei
 4  {                                  // inceputul corpului clasei
 5    private Vector elemente;                  // atribut (variabila membru)
 6    public Stack() {                          // constructor
 7      elemente = new Vector(10);              // (functie de initializare)
 8    }
 9    public Object push(Object element) {      // metoda
10      elemente.addElement(element);           // (functie membru)
11      return element;
12    }
13    public synchronized Object pop(){         // metoda
14      int lungime = elemente.size();          // (functie membru)
15      Object element = null;
16      if (lungime == 0)
17        throw new EmptyStackException();
18      element = elemente.elementAt(lungime - 1);
19      elemente.removeElementAt(lungime - 1);
20      return element;
21    }
22    public boolean isEmpty(){                 // metoda
23      if (elemente.size() == 0)               // (functie membru)
24        return true;
25      else
26        return false;
27    }
28  }                                  // sfarsitul corpului clasei
```

Structura unei clase Java

Declaratia unei clase ([ ] semnifica element optional)

```
[public] [abstract] [final] class NumeClasa [extends NumeSuperclasa]
                        [implements  NumeInterfata [, NumeInterfata]]
   {
      // Corp clasa
   }
```

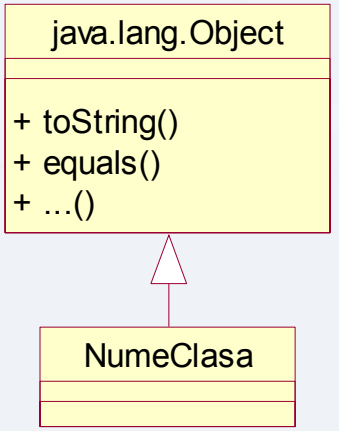
Element al declaratiei clasei	Semnificatie
public	**Orice cod exterior are acces la membrii clasei**
abstract	Clasa nu poate fi instantiata (**din ea nu pot fi create direct obiecte, ci doar din subclasele ei non-abstracte**)
final	Clasa **nu poate avea subclase**
class *NumeClasa*	**Numele clasei** este ***NumeClasa***
extends *NumeSuperClasa*	**Clasa extinde** o superclasa ***NumeSuperClasa*** (este o subclasa a clasei ***NumeSuperClasa***)
implements *NumeInterfata*	**Clasa implementează** o interfata ***NumeInterfata***

Structura unei clase Java

Declaratia unei clase

Dacă elementele opționale nu sunt declarate compilatorul Java presupune **implicit** despre clasa curent declarata ca:

- **doar clasele din acelasi director (pachet) cu clasa curenta au acces** la membrii clasei curente (**prietenie de pachet**),
- **este instantiabila** (se pot crea obiecte avand ca tip clasa curenta),
- **poate avea subclase** (create extinzand clasa curenta),
- **extinde clasa Object** (radacina ierarhiei de clase Java) si nu implementeaza nici o interfata.



Declaratia unui atribut ([ ] semnifica element optional)

```
[nivelAcces] [static] [final] tipAtribut numeAtribut;
```

Structura unei clase Java

Declaratia unui atribut ([ ] semnifica element optional)

Element al declaratiei	Semnificatie
`public`	**Orice cod exterior clasei are acces** la atribut
`protected`	**Doar codul exterior din sublcase sau aflat in acelasi director are acces** la atribut
`private`	**Nici un cod exterior nu are acces** la atribut
`static`	Are **caracter global, de clasa (e o variabila creata static, odata cu clasa, a carei locatie unica este partajata de toate obiectele clasei)**
`final`	Valoarea atributului nu poate fi modificata dupa initializare (este o **constanta)**
`transient`	Semnificatia tine de serializarea obiectelor
`volatile`	Previne compilatorul de la efectua anumite optimizari asupra atributului
tipAtribut numeAtribut	Tipul este ***tipAtribut*** iar numele este ***numeAtribut***
`[=valInitiala];`	Eventuala initializare

Structura unei clase Java

Declaratia unui atribut

Dacă elementele opționale nu sunt declarate compilatorul Java presupune **implicit** ca:

- **doar clasele din acelasi director cu clasa curenta au acces** la atributul curent,
- **atributul are caracter de obiect** (fiecare obiect din clasa curenta are un astfel de atribut nepartajat cu alte obiecte, **creat dinamic** in momentul crearii obiectului),
- **valoarea atributul poate fi modificata oricand** (este o **variabila**).

Declaratia unui constructor ([ ] semnifica element optional)

```
[nivelAcces] NumeClasa( listaParametri ) {
    // Corp constructor
}
```

Structura unei clase Java

Declaratia unui constructor ([ ] semnifica element optional)

Element al declaratiei	Semnificatie
`public`	Orice cod exterior clasei are acces la constructor
`protected`	Doar codul exterior din sublcase sau aflat in acelasi director are acces la constructor
`private`	Nici un cod exterior nu are acces la constructor
`NumeClasa`	Numele constructorului este ***`NumeClasa`***
`( ` ***`listaParametri`*** ` )`	Lista de parametri primiti de constructor, despartiti prin virgule, cu formatul ***`tipParametru numeParametru`***

Dacă elementele opționale nu sunt declarate compilatorul Java presupune **implicit** despre constructorul curent declarat ca

- **doar clasele din acelasi director cu clasa curenta au acces** la el.

Structura unei clase Java

Declaratia unei metode ([ ] semnifica element optional)

```
[nivelAcces] [static] [abstract] [final] [native] [synchronized]
        tipReturnat numeMetoda ( [listaDeParametri] )
                                  [throws NumeExceptie [,NumeExceptie] ]
{
  // Corp metoda
}
```

Dacă elementele opționale nu sunt declarate se presupune implicit ca:

- doar **codurile claselor din acelasi director cu clasa curenta au acces** la metoda curenta,
- **are caracter de obiect** (este creata dinamic in momentul crearii obiectului),
- **este implementata** (are corp),
- **poate fi rescrisa** (reimplementata) in subclase (create extinzand clasa curenta),
- **este implementata in Java**
- **nu are protectie la accesul concurent** la informatii partajate
- **nu are parametri**,
- **nu "arunca"** (declanseaza) exceptii.

Structura unei clase Java

Element al declaratiei metodei	Semnificatie
`public`	**Orice cod exterior clasei are acces** la metoda
`protected`	**Doar codul exterior din sublcase sau aflat in acelasi director are acces** la metoda
`private`	**Nici un cod exterior nu are acces** la metoda
`static`	Are **caracter global, de clasa** (este **creata static**, odata cu clasa)
`abstract`	**Nu are implementare** (trebuie implementata in subclase) si **impune declararea `abstract` a clasei din care face parte** (prin urmare clasa din care face parte nu poate avea instante)
`final`	**Nu poate fi rescrisa implementarea** metodei
`native`	Metoda **implementata in alt limbaj**
`synchronized`	**Are protectie la accesul concurent** la informatii partajate
`tipReturnat numeMetoda`	Tipul returnat este ***`tipReturnat`*** **iar numele** ***`numeMetoda`***
`(` ***`listaParametri`*** **`)`**	Lista de parametri primiti de metoda, despartiti prin virgule, cu formatul ***`tipParametru numeParametru`***
`throws` ***`NumeExceptie;`***	**Metoda arunca** exceptia ***`NumeExceptie`***

Scopul variabilelor

Scopul variabilelor

Scopul variabilelor (vizibilitatea lor in interiorul clasei):

- reprezintă portiunea de cod al clasei în care variabila este accessibilă si
- determină momentul în care variabila este creată şi distrusă.

Exista **4 categorii de scop al variabilelor Java:**

1. Camp sau atribut sau variabilă membru (*member variable*)

- este membrul unei clase sau al unui obiect,
- poate fi declarată oriunde în clasă, dar nu într-o metodă,
- e disponibilă în tot codul clasei

2. Variabilă locală (*local variable*)

- poate fi declarată oriunde într-o metodă sau într-un bloc de cod al metodei
- e disponibilă în codul metodei, din locul de declarare şi până la sfârşitul blocului in care e declarata

Scopul variabilelor

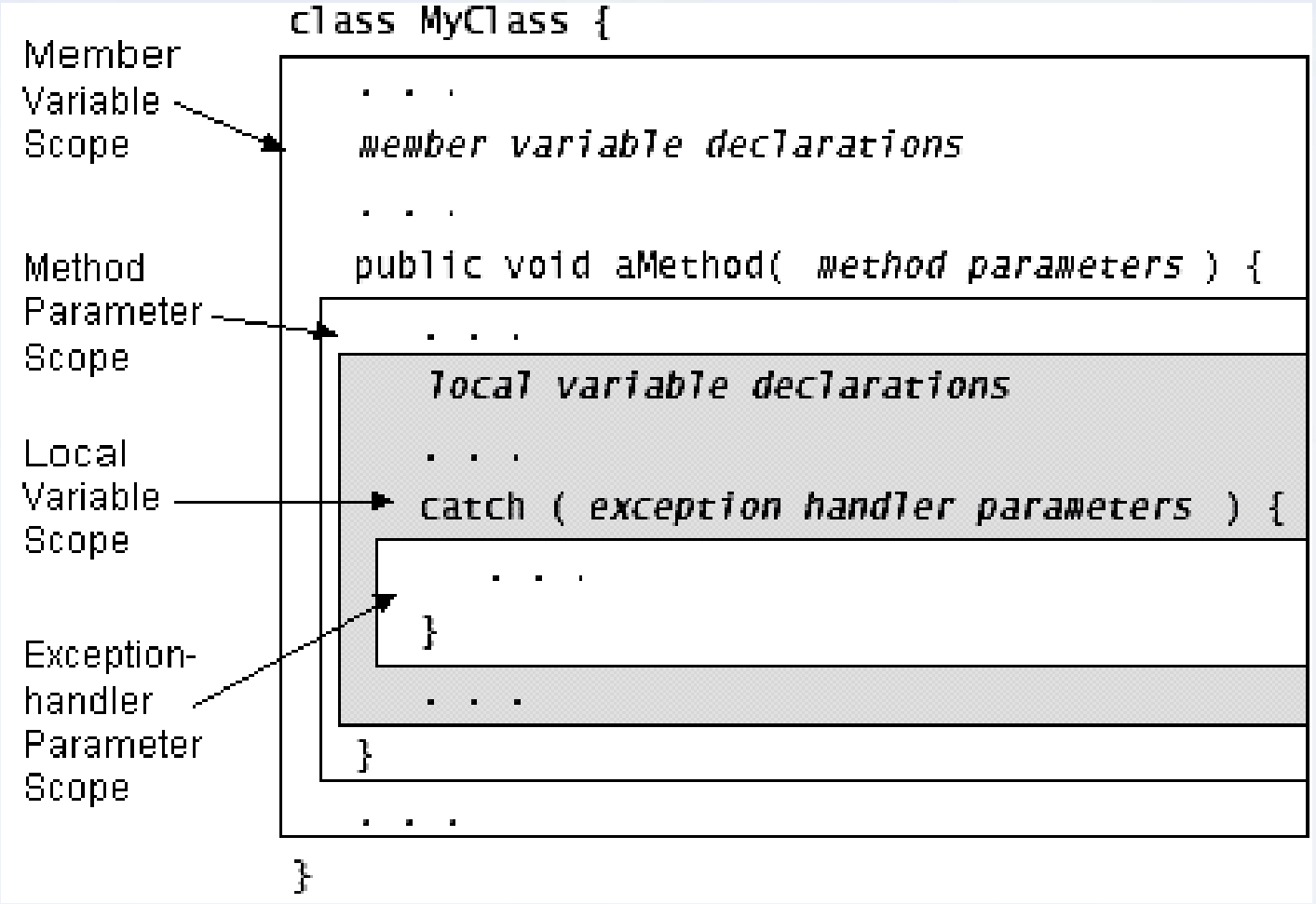
3. Parametru al unei metode

- este argumentul formal al metodei
- este utilizat pentru a se pasa valori metodei
- este disponibil în întreg codul metode

4. Parametru al unui handler de excepție

- este argumentul formal al handler-ului de excepție
- este utilizat pentru a se pasa valori handler-ului de excepție
- este disponibil în întreg codul handler-ului de excepție

Scopul variabilelor



Scopul variabilelor

```
 1  public class Complex          // declaratia clasei
 2  {                             // inceputul corpului clasei
 3    private double real;              // real = atribut (camp)
 4    private double imag;              // imag = atribut (camp)
 5
 6    public void setReal(double real) { // metoda, real = parametru
 7      this.real = real;                 // real = atribut, real = parametru
 8    }
 9    public void.setImag(double imag) { // metoda, imag = parametru
10      this.imag = imag;                 // imag = atributul, imag = parametru
11    }
12
13    public static void main(String[] args) {    // metoda, args = parametru
14      double real = Double.parseDouble(args[0]);  // real = variabila locala
15      double imag = Double.parseDouble(args[1]);  // imag = variabila locala
16
17      Complex c = new Complex();                  // c = variabila locala
18      c.setReal(real);  // echiv cu c.real = real // c, real = var. locale
19      c.setImag(imag);                            // c, imag = var. locale
20
21      System.out.println("{" + c.real +          // c.real = atributul lui c
22                         ", " + c.imag + "}");    // c.imag = atributul lui c
23    }
24  }                             // sfarsitul corpului clasei
```

Clase de biblioteca Java

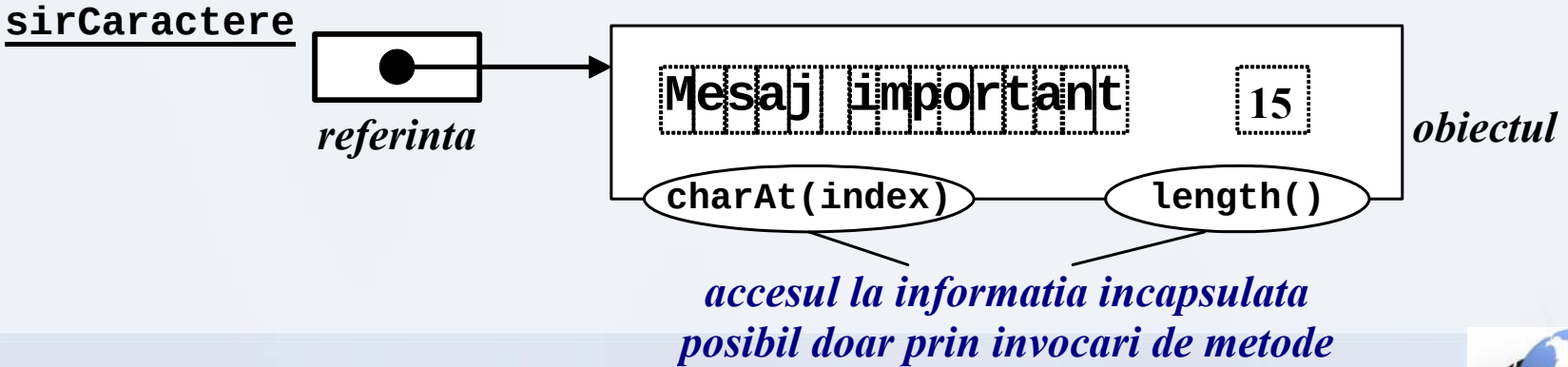
Clasa String

- **incapsuleaza siruri de caractere** – in **obiecte nemodificabile** (***immutable***)
- face parte **din pachetul de clase implicite** (***java.lang***)
- **crearea unei referinte** la obiect de **tip String**, numita **sirCaractere**

```
String  sirCaractere;  // initializata implicit cu null
```

- **crearea dinamica** a unui **obiect tip String** (incapsuleaza “Mesaj important”):

```
// alocare si initializare
sirCaractere = new String(“Mesaj important”);
```



Clasa String

- **accesul la un caracterul de index 0** (primul caracter):

```
sirDeCaractere.charAt(0)        // prin metoda charAt()
```

- **accesul la informatia privind numarul de caractere** al sirului incapsulat (lungimea sirului):

```
sirDeCaractere.length()         // prin metoda length()
```

- **pentru comparatie, cazul unui tablou de caractere** (**in Java este diferit de un sir de caractere**):

```
char[] tablouCaractere = {'M','e','s','a','j',
                          ' ','i','m','p','o','r','t','a','n','t',};
```

- **accesul la un caracterul de index 0** (primul caracter):

```
tablouCaractere[0] // prin index si operator de indexare
```

- **accesul la informatia privind numarul de caractere** (lungimea tabloului):

```
tablouCaractere.length          // prin camp length
```

Clasa String

Lucrul cu obiecte de tip String

```
 1  // variabile referinta
 2   String a;          // referinta la String neinitializata
 3   String b = null;   // referinta la String initializata explicit cu null
 4
 5   // constructie siruri de caractere utilizand constructori String()
 6   String sirVid = new String();       // sirVid.length() = 0, sirVid = ""
 7
 8   byte[] tabByte = {65, 110, 110, 97};          // coduri ASCII
 9   String sirTablouByte = new String(tabByte);  // sirTablouByte = "Anna"
10
11   char[] tabChar = {'T', 'e', 's', 't'};
12   String sirTabChar = new String(tabChar);     // sirTabChar = "Test"
13
14   String s = "Sir de caractere";
15   String sir = new String(s);                  // sir = "Sir de caractere"
```

Clasa String

Lucrul cu obiecte de tip String

```
 1   // constructie siruri de caractere utilizand metode de clasa
 2   boolean adevarat = true;
 3   String sirBoolean = String.valueOf(adevarat);   // sirBoolean = "true"
 4
 5   char caracter = 'x';
 6   String sirChar = String.valueOf(caracter);      // sirChar = "x"
 7
 8   char[] tab2Char = {'A', 'l', 't', ' ', 't', 'e', 's', 't'};
 9   String sirTab2Char = String.valueOf(tab2Char);  // sirTabChar2="Alt test"
10
11   int numar = 10000;
12   String sirInt = String.valueOf(numar);          // sirInt = "10000"
13
14   double altNumar = 2.3;
15   String sirDouble = String.valueOf(altNumar);    // sirDouble = "2.3"
```

Clasa String

Lucrul cu obiecte de tip String

```
 1   // echivalente functionale - 1
 2   char[] caractere = {'t', 'e', 's', 't'};
 3   String sir = new String(caractere);
 4   // echivalent cu    String sir = String.valueOf(caractere);
 5
 6   // echivalente functionale - 2
 7   char[] caractere = {'t', 'e', 's', 't', 'a', 'r', 'e'};
 8   String sir = new String(caractere, 2, 5);
 9   // echivalent cu    String sir = String.valueOf(caractere, 2, 5);
10
11   // echivalente functionale - 3
12   String original = "sir";
13   String copie = new String(original);
14   // echivalent cu    String copie = original.toString();
15   // echivalent cu    String copie = String.valueOf(original);
16   // echivalent cu    String copie = original.substring(0);
17
18   // complementaritati functionale
19   String sir = "test";
20   byte[] octeti = sir.getBytes();
21   String copieSir = new String(octeti);
```

Clasa String – exemplu de analiza lexicala (*parsing*)

```
 1  public class CautareCuvinteCheie1 {
 2     public static void main(String[] args) {
 3
 4        String textAnalizat = "The string tokenizer class allows "
 5                            + “application to break a string into tokens.";
 6        String[] cuvinteCheie = { "string" , "token" };
 7        // Pentru toate cuvintele cheie cautate
 8        for (int i=0; i<cuvinteCheie.length; i++) {
 9           String text = textAnalizat;
10           int pozitie=0;
11
12           // Daca un anumit cuvant cheie este gasit intr-un anumit text
13           // Varianta cu String.indexOf()
14           while ( text.indexOf(cuvinteCheie[i]) > -1 ) {
15              pozitie = pozitie + text.indexOf(cuvinteCheie[i])+1;
16
17              // Informeaza utilizatorul (indicand si pozitia)
18              System.out.println("Cuvantul cheie \"" + cuvinteCheie[i] +
19                 "\" a fost gasit in text pe pozitia " + pozitie + "\n");
20              text = text.substring(text.indexOf(cuvinteCheie[i])+1);
21           }
22        }
23     }
24  }
```

Clasa StringBuffer – alternativa cu obiecte modificabile

```
 1  class ReverseString {
 2     public static String reverseIt(String source) {
 3        int i, len = source.length();
 4        StringBuffer dest = new StringBuffer(len);
 5
 6        for (i = (len - 1); i >= 0; i--)
 7           dest.append(source.charAt(i));
 8        return dest.toString();
 9     }
10   }
11   public class StringsDemo {
12     public static void main(String[] args) {
13        String palindrome = "ele fac cafele";
14        String reversed = ReverseString.reverseIt(palindrome);
15        System.out.println(reversed);
16     }
17   }          // se va afisa elefac caf ele
```

Codul:

```
x = "a" + 4 + "c";
```

este compilat ca:

```
x = new StringBuffer().append("a").append(4).append("c").toString();
```

Clasa Integer

Lucrul cu obiecte de tip Integer

- **incapsuleaza intregi int** – in **obiecte nemodificabile** (***immutable***)

```
1   // declarare variabile de tip intreg
2   // int – primitiv
3   int     i, j, k;      // intregi ca variabile de tip primitiv
4   // Integer – obiect care incapsuleaza un int
5   Integer m, n, o;      // intregi incapsulati in obiecte Integer
6
7   // si variabile de tip String
8   String  s, r, t;      // siruri de caractere (incapsulate in obiecte)
9
10  // constructia intregilor incapsulati utilizand constructori ai clasei
11  i = 1000;
12  m = new Integer(i);         // echivalent cu   m = new Integer(1000);
13
14  r = new String("30");
15  n = new Integer(r);         // echivalent cu   n = new Integer("30");
16
17  // constructia intregilor incapsulati utilizand metode de clasa
18  t = "40";
19  o = Integer.valueOf(t);     // echivalent cu   o = new Integer("40");
```

Clasa Integer

Lucrul cu obiecte de tip Integer

```
1   // conversia intregilor incapsulati la valori numerice primitive
2   // obiectul m incapsuleaza valoarea 1000
3   byte  iByte = m.byteValue();          // diferit de 1000! (trunchiat)
4
5   int   iInt = m.intValue();            // = 1000
6
7   float iFloat = m.floatValue();        // = 1000.0F
8
9   double iDouble = m.doubleValue();     // = 1000.0
10
11  // conversia valorilor intregi primitive la siruri de caractere
12  String douaSute = Integer.toString(200);  // metoda de clasa (statica)
13
14  String oMieBinary = Integer.toBinaryString(1000); // metoda de clasa
15
16  String oMieHex = Integer.toHexString(1000);       // metoda de clasa
17
18  // conversia sirurilor de caractere la valori intregi primitive
19  int oSuta = Integer.parseInt("100");       // metoda de clasa (statica)
```

Tratarea exceptiilor – blocurile try {} catch (ex) {}

In programul urmator

- in **cazul** in care **argumentul nu are format intreg**

- **apelul metodei parseInt() genereaza o exceptie** de tip **NumberFormatException (definita in pachetul java.lang),**

- **care trebuie tratata exceptia cu un bloc:**

```
 1  public class VerificareArgumenteIntregi {
 2      public static void main(String[] args) {
 3        int i;
 4
 5        for ( i=0; i < args.length; i++ ) {
 6          try {
 7            System.out.println(Integer.parseInt(args[i]));
 8          }
 9          catch (NumberFormatException ex) {
10            System.out.println("Argumentul " + args[i] +
11                               " nu are format numeric intreg");
12          }
13        }
14      }
15    }
```

Tratarea exceptiilor – blocurile try {} catch (ex) {}

Formatul general al blocului de tratare a unei exceptii de tip **NumberFormatException:**

```
 1      try {
 2
 3         // aici este plasata secventa de cod
 4         // care poate genera exceptia
 5
 6      }
 7      catch (NumberFormatException ex) {
 8
 9         // aici este plasata secventa de cod
10         // care trateaza exceptia
11
12      }
```

Tratarea exceptiilor – blocurile try {} catch (ex) {}

```
 1  public class ClasificareArgumenteConsola {
 2
 3     // stabilirea la lansare a valorilor, ca argumente ale programelor
 4     public static void main(String[] args) {
 5       int i;
 6       for ( i=0; i < args.length; i++ ) {
 7
 8         try {
 9           int intreg = Integer.parseInt(args[i]);
10           System.out.println("Argumentul " + intreg + "are format intreg");
11         }
12         catch (NumberFormatException ex1) {
13
14           try {
15             double real = Double.parseDouble(args[i]);
16             System.out.println("Argumentul " + real + " are format real");
17           }
18           catch (NumberFormatException ex2) {
19             System.out.println("Argumentul " + args[i] + " nu este numar");
20           }
21         }
22       }
23     }
24   }
```