

Tehnici avansate de programare

Marinescu-Ghemeci Ruxandra

verman@fmi.unibuc.ro



Programa



Programa

- ▶ **Introducere in limbajul Java**
 - Elemente de bază
 - Colecții. Tipuri generice
 - Extinderi. Interfețe

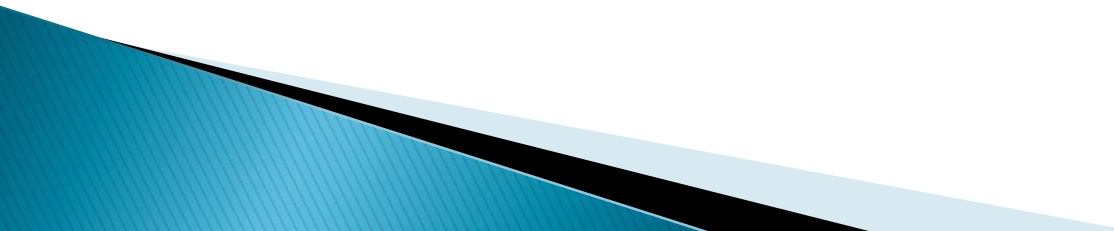
Programa

- ▶ **Tehnici de programare:**
 - Greedy
 - Divide et Impera
 - Programare dinamica
 - Backtracking
 - Branch and Bound
- ▶ **Algoritmi euristici. Algoritmi probabiliști. Algoritmi genetici**
- ▶ **NP-completitudine**
- ▶ **Principiul lui Dirichlet**

Obiectiv general

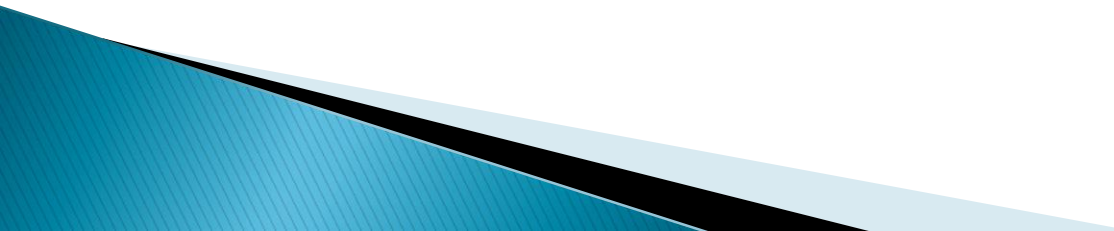
- ▶ Însușirea principalelor tehnici de elaborare a algoritmilor și a tipurilor de probleme la care se pretează acestea

Obiective specifice

- ▶ cunoașterea principalelor tehnici de programare
 - ▶ abilități de utilizare a tehnicii potrivite
 - ▶ abilități de justificare a corectitudinii algoritmilor propuși și de a estima eficiența acestora
 - ▶ însușirea elementelor de bază ale limbajului Java, utilizarea corectă a structurilor de date și algoritmilor puși la dispoziție de acest limbaj pentru implementarea algoritmilor elaborați
- 

Objective. Motivații

► Java

- elemente de bază
 - concepte POO
 - lucrul cu structuri de date
 - cursuri viitoare
 - portabil, de actualitate, simplu
 - numeroase facilități
- 

Obiective. Motivații

► Tehnici de programare

- algoritmi eficienți

"Perhaps the most important principle for the good algorithm designer is to refuse to be content" –

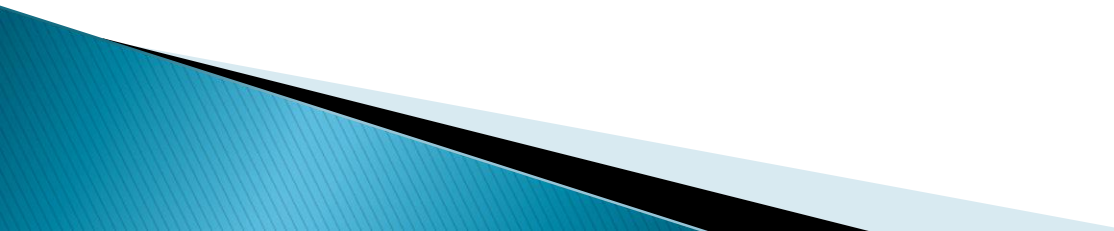
Aho, Hopcroft, and Ullman, The Design and Analysis of Computer Algorithms

Obiective. Motivații

► Tehnici de programare

- algoritmi eficienți

Exemple de probleme

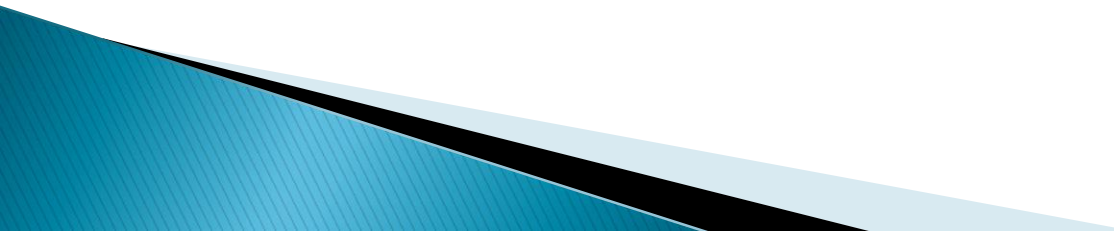
- Aflarea minimului și maximului dintr-un vector
 - Cele mai apropiate două puncte dintr-o mulțime de puncte dată
 - Numărul de inversiuni dintr-un vector
 - Înmulțirea a două numere / matrice
- 

Obiective. Motivații

► Tehnici de programare

- algoritmi corecți

Exemple de probleme

- Dată o mulțime de intervale, să se determine o submulțime de cardinal maxim de intervale care nu se suprapun
 - Dată o mulțime de intervale, fiecare interval având asociată o pondere, să se determine o submulțime de intervale care nu se suprapun având ponderea totală maximă
- 

Obiective. Motivații

► Tehnici de programare

- algoritmi eficienți (chiar dacă există soluții evidente polinomiale – se poate mai bine?)
- corectitudinea algoritmilor – demonstrații
- probleme dificile \rightarrow NP-completitudine
- pentru ce tipuri de probleme se aplica metodele

Obiective. Motivații

► Tehnici de programare

- algoritmi eficienți (chiar dacă există soluții evidente polinomiale – se poate mai bine?)
- corectitudinea algoritmilor – demonstrații
- probleme dificile –> NP-completitudine
- pentru ce tipuri de probleme se aplica metodele
- analiza algoritmilor
- structuri de date

Objective. Motivații

- numeroase aplicații
 - Bioinformatică, procesare texte, imagini
 - Geometrie computațională
 - Căutare web, similitudini, aliniere
 - Probleme de planificare
 - Proiectare, jocuri, strategii
 - Baze de date – arbori de căutare optimi
- probleme interviuri
- licență...

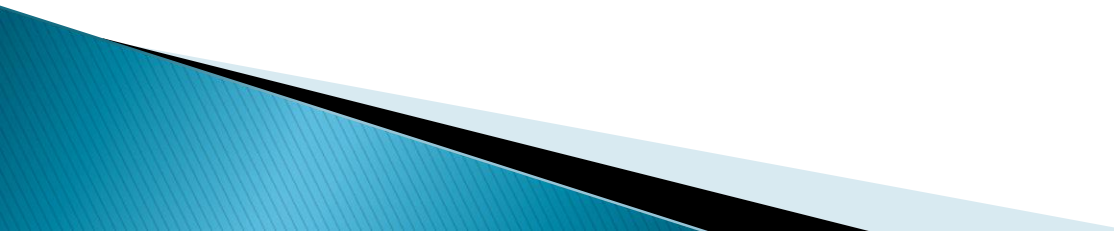
Resurse

- ▶ <http://moodle.fmi.unibuc.ro/course/>

BIBLIOGRAFIE

- ❖ Jon Kleinberg, Éva Tardos, **Algorithm Design**, Addison–Wesley 2005
<http://www.cs.princeton.edu/~wayne/kleinberg-tardos/>
- ❖ Horia Georgescu. **Tehnici de programare**. Editura Universității din București 2005
- ❖ S. Dasgupta, C.H. Papadimitriou, U.V. Vazirani, **Algorithms**, McGraw–Hill, 2008

BIBLIOGRAFIE

- ❖ T.H. Cormen, C.E. Leiserson, R.R. Rivest – **Introducere in algoritmi**, Mit Press, trad. Computer Libris Agora
 - ❖ Leon Livovschi, Horia Georgescu. **Sinteza și analiza algoritmilor**. 1986
 - ❖ Dana Lica, Mircea Pașoi, **Fundamentele programării**, L&S Infomat
- 

BIBLIOGRAFIE

- ❖ **coursera.org**

Algorithms, Part II – Princeton University

Algorithms: Design and Analysis – Stanford University

- ❖ **MIT** <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/>

- ❖ **infoarena.ro**

BIBLIOGRAFIE – Java

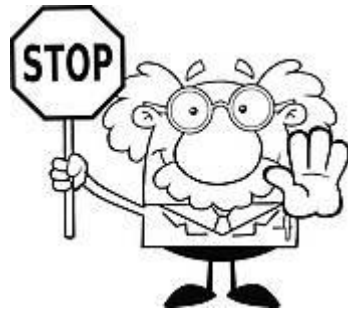
- ❖ <http://www.oracle.com/technetwork/java/javase/documentation/index.html>
- ❖ Ivor Horton – **Beginning Java 2, Java 7 Edition**, Wiley Pub., 2011
- ❖ Horia Georgescu. **Introducere în universul Java**, Editura Tehnică București 2002
- ❖ Ștefan Tanasă, Cristian Olaru, Ștefan Andrei, **Java de la 0 la expert**, ediția a II-a, Polirom 2007
- ❖ M. Naftalin, P. Wadler – **Java Generics and Collections**, O'Reilly, 2007

Consultații

- ▶ verman@fmi.unibuc.ro
- ▶ sala 318 (catedra de informatică)

Consultații

- ▶ verman@fmi.unibuc.ro
- ▶ sala 318 (catedra de informatică)



- învățat pe de rost
- copy – paste

Evaluare



Evaluare

- ▶ 50% laborator + 50% examen scris
- ▶ **Condiții:**
 - Nota test laborator ≥ 5
 - Nota laborator ≥ 5

Evaluare

► Laborator

- 1 / 2 activitate + 1 / 2 test în ultima săptămână
 - **Activitate:**
 - teme (5 teme, 2 se pot înlocui cu nota de la test)
 - teme suplimentare

Evaluare

▶ Laborator

- 1 / 2 activitate + 1 / 2 test în ultima săptămână
 - **Activitate:**
 - teme (5 teme, 2 se pot înlocui cu nota de la test)
 - teme suplimentare

▶ Examen scris – în ultima săptămână la curs (Verificare)

▶ Model subiect – moodle

Limbajul JAVA

- ▶ Tehnologii – grupate în platforme de lucru
 - J2SE (Standard Edition)
 - J2ME (Micro Edition)
 - J2EE (Enterprise Edition) ...

- ▶ Distribuțiile Java – gratuit

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

► Distribuțiile Java – gratuit

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

- Java SE 8
- jdk

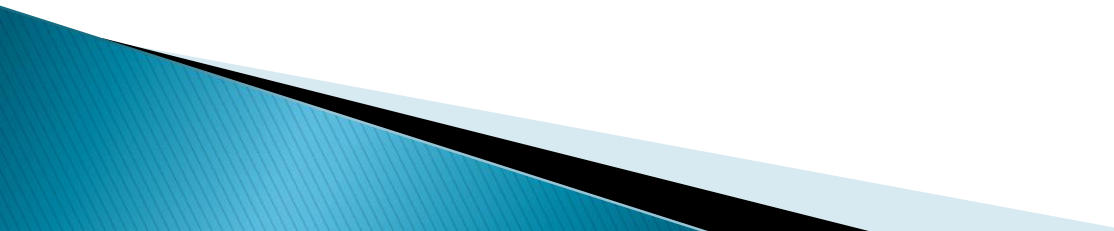
- ▶ Distribuțiile Java – gratuit

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

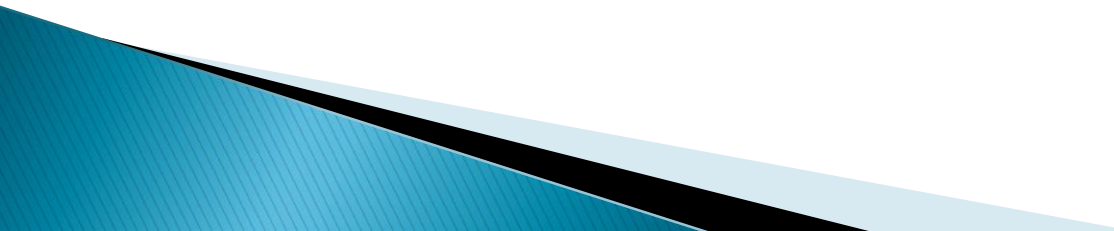
- ▶ Medii de programare Java

- NetBeans
- Eclipse
- JCreator , BlueJ ...

Caracteristici

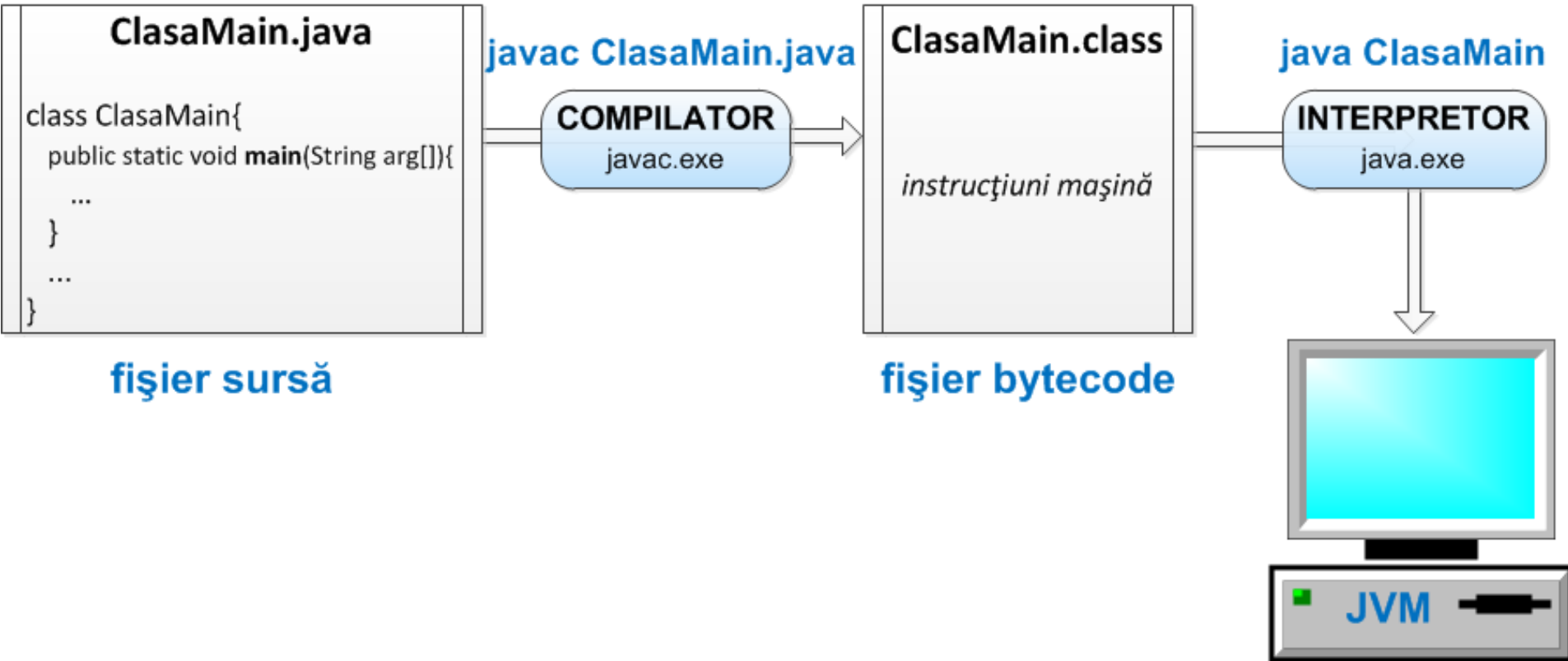
- ▶ Orientat pe obiecte
 - ▶ Simplitate
 - ▶ Colectorul de reziduuri (*garbage collector*)
 - ▶ Securitate ridicată
- 

Caracteristici

- ▶ Orientat pe obiecte
 - ▶ Simplitate
 - ▶ Colectorul de reziduuri (*garbage collector*)
 - ▶ Securitate ridicată
 - ▶ Proiectat pentru lucru în rețea
 - ▶ Posibilitatea lansării mai multor fire de executare
- 

Caracteristici

► Compilat și interpretat



Exemplu

Exemplu

```
class Unu{  
    public static void main(String arg[]) {  
        System.out.println("Prima clasa");  
    }  
}
```

Salvare: Unu.java

Compilare: javac Unu.java ➡ Unu.class

Rulare: java Unu

<https://docs.oracle.com/javase/8/docs/api/java/lang/System.html>

Elemente de bază ale limbajului

Comentarii

- ▶ de sfârșit de linie: `//`
- ▶ generale: `/* */`
- ▶ de documentare: `/** */`

Variabile

```
[modificatori] tip lista_identificatori;
```

```
int x;
```

```
public static int y;
```



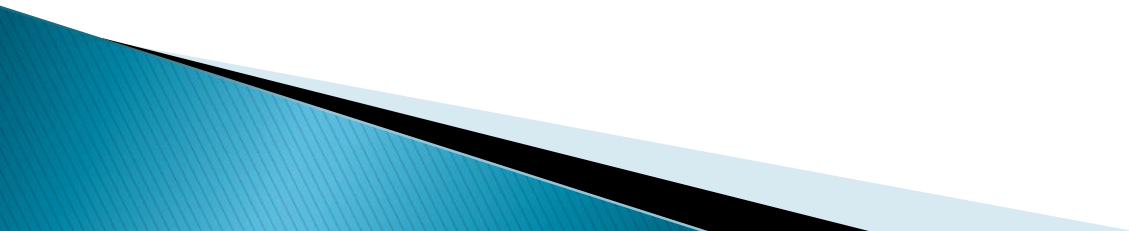
Variabile

- ▶ O variabilă poate fi de un tip
 - **primitiv** – numeric, caracter etc.
 - **referință** – vectori, clase, interfețe

Variabile

- ▶ **Categorii de variabile** – în funcție de locul în care sunt declarate
 - **Variabile membre (câmpuri)** – în interiorul unei clase
 - **Variabile locale** – într-o metodă sau într-un bloc de cod
 - **Parametrii metodelor**
 - **Parametrii de la tratarea excepțiilor**

Tipuri primitive



Tipuri primitive

- ▶ Tipuri întregi: `byte`, `short`, `int`, `long`

Tipuri primitive

- ▶ Tipuri întregi: `byte`, `short`, `int`, `long`

Literali întregi

- normali (de tip `int`)
- lungi (de tip `long`) – au sufixul `'l'` sau `'L'`.

```
int x = 456789;
```

```
long z1 = 4999999999999999L;
```

Tipuri primitive

- ▶ Tipuri întregi: `byte`, `short`, `int`, `long`

Literali întregi

- normali (de tip `int`)
- lungi (de tip `long`) – au sufixul `'l'` sau `'L'`.

```
int x = 456789;
```

```
long z1 = 4999999999999999L;
```

```
long z2 = 49_999_999_999_999L; //-java 7
```

Tipuri primitive

- ▶ Tipuri întregi: `byte`, `short`, `int`, `long`

Literali întregi

- normali (de tip `int`)
- lungi (de tip `long`) – au sufixul `'l'` sau `'L'`.

```
int x = 456789;
```

```
long z1 = 4999999999999999L;
```

```
long z2 = 49_999_999_999_999L; //-java 7
```

```
byte y1 = 456789; //possible loss of precision
```

```
byte y2 = 1;
```

Tipuri primitive

- ▶ Tipuri întregi: `byte`, `short`, `int`, `long`

Literali întregi

- normali (de tip `int`)
- lungi (de tip `long`) – au sufixul `'l'` sau `'L'`.

```
int x = 456789;
```

```
long z1 = 4999999999999999L;
```

```
long z2 = 49_999_999_999_999L; //-java 7
```

```
byte y1 = 456789; //possible loss of precision
```

```
byte y2 = 1;
```

```
int x1 = 0xA1B, x2=071, x3=0b100;
```

Tipuri primitive

- ▶ Tipuri întregi: `byte`, `short`, `int`, `long`

În calcule tipurile `byte` și `short` sunt convertite la `int`

Tipuri primitive

- ▶ Tipuri în virgulă mobilă: `float`, `double`

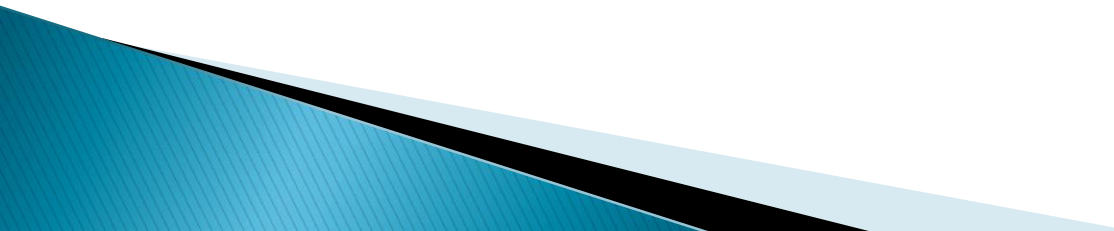
Literali reali

- dubli
- normali – sufixul 'f' sau 'F'.

```
double d = 12345.2;
```

```
float f = 12345.2; //possible loss of precision
```

```
float f1 = 12345.2f;
```



Tipuri primitive

► **char**:

```
char c='a';
```

```
char c1='\u0061';
```

```
char c2='\n';
```

```
char c3='\\';
```


Tipuri primitive

► **char**:

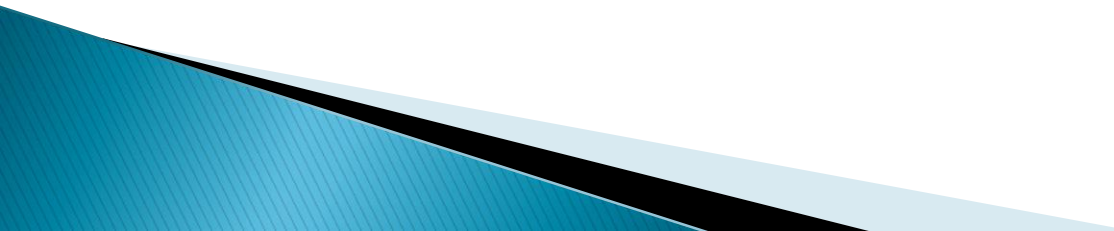
```
char c='a';
```

```
char c1='\u0061'; // UNICODE
```

```
char c2='\n';
```

```
char c3='\\';
```

```
int i=c; // DA
```



Tipuri primitive

- ▶ **boolean:** true, false

Tipuri primitive

- ▶ **boolean:** true, false

```
boolean b=true;
```

```
int i=b; // NU
```

```
int x=1;
```

```
...
```

```
if (x==1) // NU if(x)
```

Operatori

► **Operatori aritmetici** $+$ $-$ $*$ $/$ $\%$

Operatori

- ▶ **Operatori aritmetici** $+$ $-$ $*$ $/$ $\%$
- ▶ **Operatorii de incrementare și decrementare** $++$ $--$

Operatori

- ▶ **Operatori aritmetici** + - * / %
- ▶ **Operatorii de incrementare și decrementare** ++ --
- ▶ **Operatori logici** || && !

Operatori

- ▶ **Operatori aritmetici** + - * / %
- ▶ **Operatorii de incrementare și decrementare** ++ --
- ▶ **Operatori logici** || && !
- ▶ **Operatori pe biți** | & ^
 - operatorii de translație (shift): <<, >>, >>>

Operatori

```
int f = 0b10001;
```

```
//testare bit
```

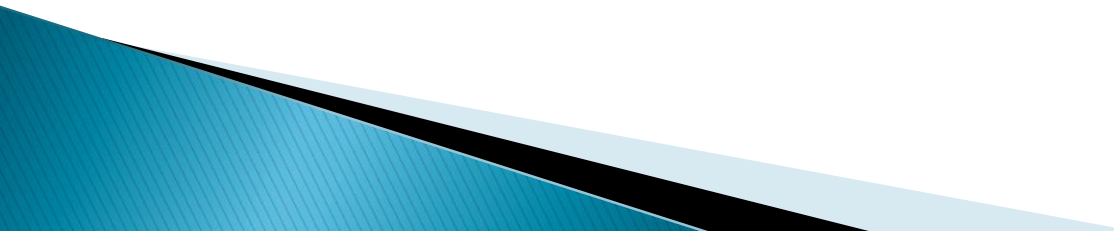
```
int mask = 0b100;
```

```
if( (f& mask) == 0)
```

```
    System.out.println("bitul 3 este 0");
```

```
else
```

```
    System.out.println("bitul 3 este 1");
```



Operatori

```
int b1 = -3;
```

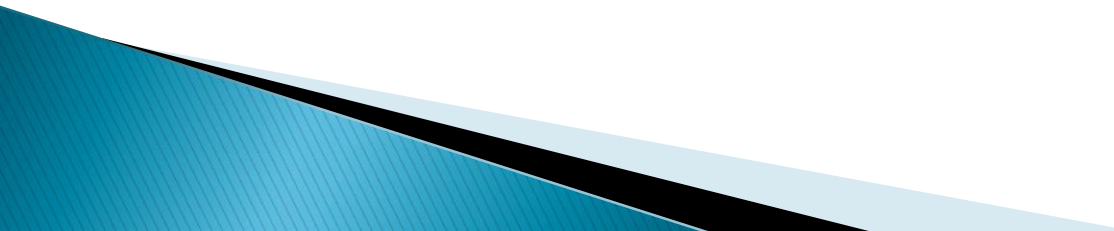
```
System.out.println(Integer.toString(b1));
```

```
System.out.println(b1>>1);
```

```
System.out.println(b1<<1);
```

```
System.out.println(Integer.toString(b1>>>24));
```

```
System.out.println(b1>>>24);
```



Operatori

- ▶ **Operatorii relaționali** $>$ $>=$ $==$ $<$ $<=$ $!=$
- ▶ **Operatorii de atribuire** $=$ $+=$ $-=$ $/=$ $*=$ $\%=$ $<<=$
 $>>=$ $>>>=$ $\&=$ $|=$ $\wedge=$
- ▶ **Operatorul condițional** $?:$ $(\text{cond} ? e1 : e2)$

Operatori

- ▶ **Operatorul de conversie a tipurilor** (tip) expresie

Operatori

- ▶ **Operatorul de conversie a tipurilor** (tip) expresie
- ▶ **Operatorul + pentru lucrul cu șiruri**

```
int u=2, v=4;
```

```
System.out.println(u+v+" suma");
```

```
System.out.println("suma "+u+v);
```

Operatori

- ▶ Operatorul de conversie a tipurilor (tip) expresie
- ▶ Operatorul + pentru lucrul cu șiruri
- ▶ Operatori pentru referințe

ob **instanceof** Clasa

```
String sir="abc";
```

```
System.out.println(sir instanceof String);
```

```
System.out.println(sir instanceof Object);
```

► Precedența operatorilor

$x - y + z$

$x = y = z = 0;$

► Precedența operatorilor

$x - y + z$

$x = y = z = 0;$

► Instrucțiuni

Diferențe C++

- ▶ Tipul `boolean`

```
int x=1;
```

```
...
```

```
if (x==1)    //NU if (x)
```


Diferențe C++

- ▶ for pentru obiecte iterabile

```
for (tip identificador : obiect_iterabil)  
    instrucțiuni
```

Diferențe C++

- ▶ for pentru obiecte iterabile

```
for (tip identificador : obiect_iterabil)  
    instrucțiuni
```

- **Exemplu:** Afișarea elementelor unui tablou unidimensional de numere întregi:

```
for(int i=0;i<a.length;i++)  
    System.out.print(a[i]+" ");
```

Diferențe C++

- ▶ for pentru obiecte iterabile

```
for (tip identificador : obiect_iterabil)  
    instrucțiuni
```

- **Exemplu:** Afișarea elementelor unui tablou unidimensional de numere întregi:

```
for(int i=0;i<a.length;i++)  
    System.out.print(a[i]+" ");  
sau
```

Diferențe C++

- ▶ for pentru obiecte iterabile

```
for (tip identificador : obiect_iterabil)  
    instrucțiuni
```

- **Exemplu:** Afișarea elementelor unui tablou unidimensional de numere întregi:

```
for (int i=0; i<a.length; i++)  
    System.out.print(a[i]+" ");
```

sau

```
for (int x:a)  
    System.out.print(x+" ");
```

Clase

Exemplu

Clase

▶ Clase

```
[modifier] class NumeClasaDefinita [extends  
NumeClasa] [implements NumeInterfete]{  
    corp;  
}
```

Clase

▶ Clasa

```
[modifier] class NumeClasaDefinita [extends  
NumeClasa] [implements NumeInterfete]{  
    corp;  
}
```

▶ Câmpuri

```
[modificatori] tip lista_identificatori;
```


Clase

▶ Clasa

```
[modifier] class NumeClasaDefinita [extends  
NumeClasa] [implements NumeInterfete]{  
    corp;  
}
```

▶ Câmpuri

```
[modifieri] tip lista_identificatori;
```

▶ Metode

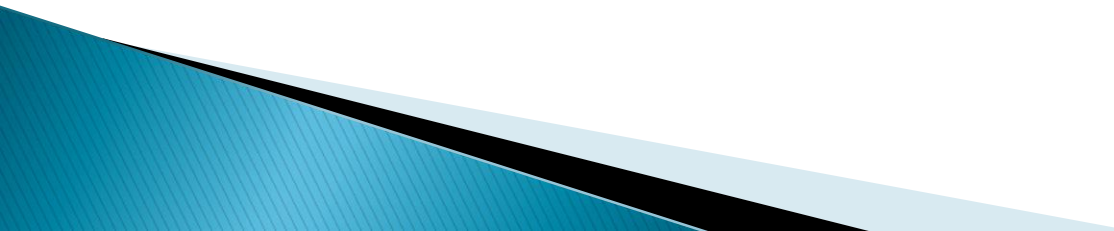
```
[modifieri] tip_returnat numeMetoda(lista_parametrii)
```



Clase

- ▶ **Constructor** – seamănă cu metodele, dar numele lor este obligatoriu numele clasei și nu întorc valori.
 - **Unul dintre constructori este automat invocat la crearea unui obiect de tipul clasei respective**

Clase

- ▶ Clasele sunt considerate **tipuri**.
 - ▶ Entitățile al căror tip este o clasă se numesc **obiecte** (instanțieri, instanțe) ale clasei respective
 - ▶ Crearea obiectelor se face cu ajutorul operatorului **new**.
- 

Clase – exemplu

Clase – exemplu

```
class C {
```

```
}
```

Clase – exemplu

```
class C {  
    // câmpuri  
    int x;  
    boolean y;
```

```
}
```

Clase – exemplu

```
class C {  
    // câmpuri  
    int x;  
    boolean y;  
  
    // constructori  
    C() { x=2; }
```

```
}
```

Clase – exemplu

```
class C {  
    // câmpuri  
    int x;  
    boolean y;  
  
    // constructori  
    C() { x=2; }
```

- y primește valoare implicită

```
}
```


Clase – exemplu

```
class C {  
    // câmpuri  
    int x;  
    boolean y;  
  
    // constructori  
    C() { x=2; }  
    C(int a, boolean b) { x=a; y=b; }  
  
}
```

Clase – exemplu

```
class C {  
    // câmpuri  
    int x;  
    boolean y;  
  
    // constructori  
    C() { x=2; }  
    C(int a, boolean b) { x=a; y=b; }  
  
    // metode  
    int met() {  
        if (y) return x;  
        else return x+1;  
    }  
  
}
```

Clase – exemplu

```
class C {  
    // câmpuri  
    int x;  
    boolean y;  
  
    // constructori  
    C() { x=2; }  
    C(int a, boolean b) { x=a; y=b; }  
  
    // metode  
    int met() {  
        if (y) return x;  
        else return x+1;  
    }  
    void met(int x) {  
        if (this.x==x) y=true;  
    }  
  
}
```

Clase – exemplu

```
class C {  
    // câmpuri  
    int x;  
    boolean y;  
  
    // constructori  
    C() { x=2; }  
    C(int a, boolean b) { x=a; y=b; }  
  
    // metode  
    int met() {  
        if (y) return x;  
        else return x+1;  
    }  
    void met(int x) {  
        if (this.x==x) y=true;  
    }  
  
}
```



supraîncărcare

Clase – exemplu

```
class C {  
    // câmpuri  
    int x;  
    boolean y;  
  
    // constructori  
    C() { x=2; }  
    C(int a, boolean b) { x=a; y=b; }  
  
    // metode  
    int met() {  
        if (y) return x;  
        else return x+1;  
    }  
    void met(int x) {  
        if (this.x==x) y=true;  
    }  
}
```



referință la obiectul curent


Clase – exemplu

```
class C {  
    // câmpuri  
    int x;  
    boolean y;  
  
    // constructori  
    C() { x=2; }  
    C(int a, boolean b) { x=a; y=b; }  
  
    // metode  
    int met() {  
        if (y) return x;  
        else return x+1;  
    }  
    void met(int x) {  
        if (this.x==x) y=true;  
    }  
    void afis(){  
        System.out.println(x+" "+y);  
    }  
}
```

Clase – exemplu

```
class C {  
    // câmpuri  
    int x;  
    boolean y;  
  
    // constructori  
    C() { x=2; }  
    C(int a, boolean b) { x=a; y=b; }  
  
    // metode  
    int met() {  
        if (y) return x;  
        else return x+1;  
    }  
    void met(int x) {  
        if (this.x==x) y=true;  
    }  
    void afis(){  
        System.out.println(x+" "+y);  
    }  
}
```

concatenare – orice tip de
date se poate transforma în
șir de caractere



Clase – exemplu

```
class MainC {
    public static void main(String arg[]){

    }
}
```


Clase – exemplu

```
class MainC {  
    public static void main(String arg[]){  
        //Declaratia și crearea unui obiect de tipul C:  
        C ob1; //declararea  
  
    }  
}
```

Clase – exemplu

ob1 – referință către un obiect

```
class MainC {  
    public static void main(String arg[]){  
        //Declaratrea și crearea unui obiect de tipul C:  
        C ob1; //declararea  
  
    }  
}
```

Clase – exemplu

```
class MainC {  
    public static void main(String arg[]){  
        //Declaratia și crearea unui obiect de tipul C:  
  
        C ob1; //declararea  
        ob1 = new C(); //crearea  
  
    }  
}
```

Clase – exemplu

```
class MainC {  
    public static void main(String arg[]){  
        //Declaratia și crearea unui obiect de tipul C:  
  
        C ob1; //declararea  
        ob1 = new C(); //crearea  
  
    }  
}
```

Clase – exemplu

```
class MainC {  
    public static void  
        //Declaratia si crearea  
        C ob1; //declaratia  
        ob1 = new C();  
}
```

- se creează o nouă instanță a clasei
- se alocă memorie pentru ea
- este invocat constructorul corespunzător

```
}
```

Clase – exemplu

```
class MainC {  
    public static void main(String arg[]) {  
        //Declarată și crearea unui obiect de tipul C:  
  
        C ob1; //declarată  
        ob1 = new C(); //crearea  
  
    }  
}
```

- dacă în clasa C nu există constructor declarat explicit, se presupune că “există” un constructor fără parametri, care nu prevede nici o acțiune

Clase – exemplu

```
class MainC {  
    public static void main(String arg[]){  
        //Declaratia și crearea unui obiect de tipul C:  
  
        C ob1; //declararea  
        ob1 = new C(); //crearea  
  
        C ob2 = new C(1,true); //declarare si creare  
  
    }  
}
```

Clase – exemplu

```
class MainC {  
    public static void main(String arg[]){  
        //Declaratia și crearea unui obiect de tipul C:  
  
        C ob1; //declararea  
        ob1 = new C(); //crearea  
  
        C ob2 = new C(1,true); //declarare si creare  
  
        //Invocarea metodelor:  
  
        ob1.afis();  
  
    }  
}
```


Clase – exemplu

```
class MainC {  
    public static void main(String arg[]){  
        //Declaratia și crearea unui obiect de tipul C:  
  
        C ob1; //declararea  
        ob1 = new C(); //crearea  
  
        C ob2 = new C(1,true); //declarare si creare  
  
        //Invocarea metodelor:  
  
        ob1.afis();  
        System.out.println(ob1.met());  
        ob1.met(2);  
        ob1.afis();  
        System.out.println(ob1.met());  
        ob2.afis();  
    }  
}
```

```

class C {
    int x;
    boolean y;
    C() { x=2; }
    C(int a, boolean b) {
        x=a; y=b; }
    int met() {
        if (y) return x;
        else return x+1;
    }
    void met(int x) {
        if (this.x==x)
            y=true;
    }
    void afis(){
        System.out.println(x+" "
            +y);
    }
}

```

```

class MainC {
    public static void main(String arg[]){

        C ob1; //declararea
        ob1 = new C(); //crearea

        C ob2 = new C(1,true);

        ob1.afis();
        System.out.println(ob1.met());
        ob1.met(2);
        ob1.afis();

        System.out.println(ob1.met());

        ob2.afis();

    }
}

```

Apelarea câmpurilor/metodelor

```
C ob;// variabila referinta
```

```
ob.x = 1;    //NU
```

Apelarea câmpurilor/metodelor

```
C ob; // variabila referinta
```

```
ob=new C();
```

```
ob.x;
```

```
ob.met (...);
```

Apelarea câmpurilor/metodelor

```
C ob;// variabila referinta
```

```
ob=new C();
```

```
ob.x;
```

```
ob.met (...);
```

```
ob=null;
```

```
ob.x = 1; //NullPointerException
```



Apelarea câmpurilor/metodelor

Câmpuri și metode statice



Apelarea câmpurilor/metodelor

Câmpuri și metode statice

- ▶ **câmp static** – unic pe clasă, comun tuturor obiectelor ce sunt instanțe ale clasei
- ▶ **metodă statică**

Apelarea câmpurilor/metodelor

Câmpuri și metode statice

- ▶ **câmp static** – unic pe clasă, comun tuturor obiectelor ce sunt instanțe ale clasei
- ▶ **metodă statică**

`C.x;`

`C.met (...);`


```
class ExpStatic{  
    static int nr=1;  
    int x=1;
```

```
class ExpStatic{  
    static int nr=1;  
    int x=1;  
    static void cresteNr(){ nr++; }  
    void cresteX(){ x++; } //NU static
```

```
class ExpStatic{  
    static int nr=1;  
    int x=1;  
    static void cresteNr(){ nr++; }  
    void cresteX(){ x++; } //NU static  
    void afis(){ System.out.println(nr+" "+x ); }
```

```
class ExpStatic{
    static int nr=1;
    int x=1;
    static void cresteNr(){ nr++; }
    void cresteX(){ x++; } //NU static
    void afis(){ System.out.println(nr+" "+x ); }
    public static void main(String s[]){
        ExpStatic.cresteNr();
        System.out.println(ExpStatic.nr);

    }
}
```

```
class ExpStatic{
    static int nr=1;
    int x=1;
    static void cresteNr(){ nr++; }
    void cresteX(){ x++; } //NU static
    void afis(){ System.out.println(nr+" "+x ); }
    public static void main(String s[]){
        ExpStatic.cresteNr();
        System.out.println(ExpStatic.nr);
        ExpStatic ob1,ob2;
        ob1=new ExpStatic();
        ob2=new ExpStatic();

    }
}
```

```
class ExpStatic{
    static int nr=1;
    int x=1;
    static void cresteNr(){ nr++; }
    void cresteX(){ x++; } //NU static
    void afis(){ System.out.println(nr+" "+x ); }
    public static void main(String s[]){
        ExpStatic.cresteNr();
        System.out.println(ExpStatic.nr);
        ExpStatic ob1,ob2;
        ob1=new ExpStatic();
        ob2=new ExpStatic();
        ob1.cresteNr();
        ob1.cresteX();
        ob1.afis();

    }
}
```

```
class ExpStatic{
    static int nr=1;
    int x=1;
    static void cresteNr(){ nr++; }
    void cresteX(){ x++; } //NU static
    void afis(){ System.out.println(nr+" "+x ); }
    public static void main(String s[]){
        ExpStatic.cresteNr();
        System.out.println(ExpStatic.nr);
        ExpStatic ob1,ob2;
        ob1=new ExpStatic();
        ob2=new ExpStatic();
        ob1.cresteNr();
        ob1.cresteX();
        ob1.afis();
        ob2.afis();
    }
}
```

Tablouri

Tablouri unidimensionale

- ▶ Declaraarea:

```
tip[] a;
```

```
tip a[];
```



Declaraarea unui tablou nu are drept consecință crearea sa (! un tablou este un tip referință)

Tablouri unidimensionale

- ▶ Declararea:

```
tip[] a;  
tip a[];
```



Declaraarea unui tablou nu are drept consecință crearea sa (! un tablou este un tip referință)

- ▶ Crearea

```
a = new tip[n];
```

Tablouri unidimensionale – Exemplu

```
int a[];  
a = new int[5];  
a[0] = 1;
```

Tablouri unidimensionale

- ▶ Declaraarea și crearea pot fi făcute și simultan :

```
tip[] a = new tip[n];
```

```
tip a[] = new tip[n];
```

sau printr-o inițializare efectivă:

```
int[] a = {0,3,2,5,1}
```



Tablouri unidimensionale

- ▶ Declaraarea și crearea pot fi făcute și simultan :

```
tip[] a = new tip[n];
```

```
tip a[] = new tip[n];
```

sau printr-o inițializare efectivă:

```
int[] a = {0,3,2,5,1}
```

- ▶ `tip` poate fi un tip primitiv sau tip referință (clasă, interfață)

Tablouri unidimensionale

- ▶ Componentele tabloului: $a[i]$, cu $i = 0..n-1$

Tablouri unidimensionale

- ▶ Componentele tabloului: `a[i]`, cu `i = 0..n-1`
- ▶ Lungimea tabloului: `a.length`

Tablouri unidimensionale

- ▶ Componentele tabloului: `a[i]`, cu `i = 0..n-1`
- ▶ Lungimea tabloului: `a.length`
- ▶ `ArrayIndexOutOfBoundsException`

Tablouri unidimensionale

- ▶ **Afișarea** elementelor unui tablou unidimensional de numere întregi:

```
for (int i=0; i<a.length; i++)  
    System.out.print(a[i]+" ");
```

Tablouri unidimensionale

- ▶ **Afișarea** elementelor unui tablou unidimensional de numere întregi:

```
for (int i=0; i<a.length; i++)  
    System.out.print(a[i]+" ");
```

sau

```
for (int x:a)  
    System.out.print(x+" ");
```

Tablouri unidimensionale

- ▶ **for** pentru obiecte iterabile

```
for (tip identificator : obiect_iterabil)  
    instrucțiuni
```

Tablouri unidimensionale – Exemple

- ▶ Interschimbarea conținutului a două tablouri:

```
int[] a = {1,2,3,4}, b = {11,12,13}, c;  
c = a; a = b; b = c;
```

Tablouri unidimensionale – Exemple

- ▶ Copierea elementelor unui vector a în vectorul b

```
int a[] = {1, 2, 3, 4};  
int b[] = new int[4];
```

Tablouri unidimensionale – Exemple

- ▶ Copierea elementelor unui vector a în vectorul b

```
int a[] = {1, 2, 3, 4};
```

```
int b[] = new int[4];
```

```
// Varianta 1 - Nu are efectul dorit
```

```
b = a;
```



Tablouri unidimensionale – Exemple

- ▶ Copierea elementelor unui vector a în vectorul b

```
int a[] = {1, 2, 3, 4};
```

```
int b[] = new int[4];
```

```
// Varianta 1 - Nu are efectul dorit
```

```
b = a;
```

```
System.out.println(a[0]+" "+b[0]);
```

```
b[0] = 5;
```

```
System.out.println(a[0]+" "+b[0]);
```

```
a[0] = 6;
```

```
System.out.println(a[0]+" "+b[0]);
```

Tablouri unidimensionale – Exemple

- Copierea elementelor unui vector a în vectorul b

```
int a[] = {1, 2, 3, 4};  
int b[] = new int[4];
```

```
// Varianta 2
```

```
for(int i=0; i<a.length; i++)  
    b[i] = a[i];
```

```
// Varianta 3
```

```
System.arraycopy(a, 0, b, 0, a.length);
```



Tablouri unidimensionale de “obiecte”

► `Clasa a[];`

Tablouri unidimensionale de “obiecte”

▶ `Clasa a[];`

`a[0] = new Clasa(); //NU`

Tablouri unidimensionale de “obiecte”

Tablou de referințe (adrese)

▶ `Clasa a[] = new Clasa[4];`

Tablouri unidimensionale de “obiecte”

▶ `Clasa a[] = new Clasa[4];`

`a[0].met(); //eroare rulare`

Tablouri unidimensionale de “obiecte”

```
▶ Clasa a[] = new Clasa[4];  
    a[0]= new Clasa();  
    a[0].met(); //corect
```

Tablouri unidimensionale de “obiecte”

► `Clasa a[] = new Clasa[4];`

```
for(int i=0; i<a.length; i++)  
    a[i] = new Clasa();
```

Tablouri unidimensionale – Exemple

- ▶ În clasa **Arrays** din pachetul **java.util** există metode (statice) utile pentru lucrul cu tablouri :

- fill(int[] a, int fromIndex, int toIndex, int val)

- sort(int[] a)

etc

<http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html>

Tablouri multidimensionale

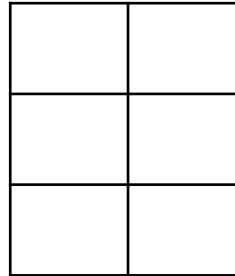
- ▶ tablouri unidimensionale ale căror elemente sunt tablouri unidimensionale etc

`a[indice1] ... [indicen]`

Tablouri multidimensionale

► Tablouri bidimensionale

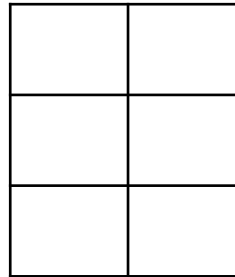
```
int[][] a = new int[3][2];
```



Tablouri multidimensionale

- ▶ Tablouri bidimensionale

```
int a[][] = new int[3][2];
```



- ▶ Avem

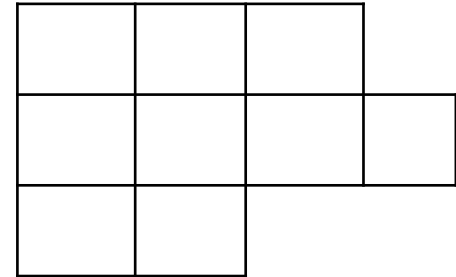
```
a.length = 3
```

```
a[1].length = 2
```

Tablouri multidimensionale

► Tablouri bidimensionale

```
int[][] a = new int[3][];  
a[0] = new int[3];  
a[1] = new int[4];  
a[2] = new int[2];
```

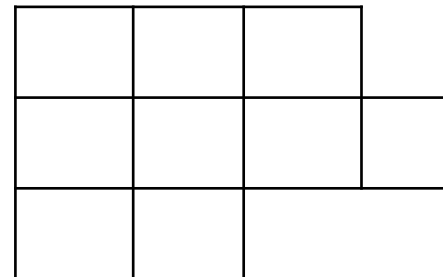


► Avem `a[1].length=4`

Tablouri multidimensionale

▶ Tablouri bidimensionale

```
int[][] a = new int[3][];  
a[0] = new int[3];  
a[1] = new int[4];  
a[2] = new int[2];
```



- ▶ Avem `a[1].length=4`
- ▶ La aceeași structură se poate ajunge și printr-o inițializare efectivă:

```
int[][] a = { {0,1,2}, {1,2,3,4}, {2,3} };  
int a[][] = { {0,1,2}, {1,2,3,4}, {2,3} };
```

Tablouri multidimensionale

- ▶ Exemplu. Afișarea elementelor unui tablou bidimensional cu for-each

```
int x[][]={{1,2,3,7},{4,5},{8}};
```

```
for(int[] linie:x){  
    for(int elem:linie)  
        System.out.printf("%5d",elem);  
    System.out.println();  
}
```

Clase înfășurătoare

Clase înfășurătoare

- ▶ Pentru fiecare tip primitiv există o clasă corespunzătoare (clasă înfășurătoare – **wrapper class**)
 - **int** – **Integer**
 - **char** – **Character**
 - **double** – **Double**, **long** – **Long**, **short** – **Short**, **byte** – **Byte**, **boolean** – **Boolean**

Clase înfășurătoare

- ▶ Pentru fiecare tip primitiv există o clasă corespunzătoare (clasă înfășurătoare – **wrapper class**)
 - **int** – **Integer**
 - **char** – **Character**
 - **double** – **Double**, **long** – **Long**, **short** – **Short**, **byte** – **Byte**, **boolean** – **Boolean**
- ▶ Utilitate

Clase înfășurătoare

- ▶ Pentru fiecare tip primitiv există o clasă corespunzătoare (clasă înfășurătoare – **wrapper class**)
 - `int` – `Integer`
 - `char` – `Character`
 - `double` – `Double`, `long` – `Long`, `short` – `Short`, `byte` – `Byte`, `boolean` – `Boolean`
- ▶ Utilitate
 - `Integer.MAX_VALUE`
 - `int x = Integer.parseInt("123")`
 - `Colecții`

Clase înfășurătoare

- ▶ Trecerea int - Integer (versiuni <5)

```
int i = 1;
```

```
Integer wi = new Integer(i);
```

Clase înfășurătoare

- ▶ Trecerea int - Integer (versiuni <5)

```
int i = 1;
```

```
Integer wi = new Integer(i);
```

```
int j = wi.intValue();
```

```
System.out.println(j);
```

Clase înfășurătoare

- ▶ Trecerea int – Integer (versiuni <5)

```
int i = 1;
```

```
Integer wi = new Integer(i);
```

```
int j = wi.intValue();
```

```
System.out.println(j);
```

- ▶ Implicit: **autoboxing** / **unboxing**

Clase înfășurătoare

- ▶ Trecerea int – Integer (versiuni <5)

```
int i = 1;  
Integer wi = new Integer(i);  
int j = wi.intValue();  
System.out.println(j);
```

- ▶ Implicit: **autoboxing / unboxing**

```
int i = 1;  
Integer wi = i;  
int j = wi;  
System.out.println(j);
```

Clase înfășurătoare



Nu este indicată folosirea claselor înfășurătoare în operații aritmetice, ci doar în lucrul cu colecții

```
class ExpAutoboxing{
    static void autoboxing() {
        Long suma=0L;
        for(long i=0;i<Integer.MAX_VALUE/10;i++)
            suma+=i;
    }
    static void primitiv() {
        long suma=0L;
        for(long i=0;i<Integer.MAX_VALUE/10;i++)
            suma+=i;
    }
}
```

```
class ExpAutoboxing{
    static void autoboxing() {
        Long suma=0L;
        for(long i=0;i<Integer.MAX_VALUE/10;i++)
            suma+=i;
    }
    static void primitiv() {
        long suma=0L;
        for(long i=0;i<Integer.MAX_VALUE/10;i++)
            suma+=i;
    }
    public static void main(String arg[]){
        long start=System.currentTimeMillis();
        autoboxing();
        System.out.println(System.currentTimeMillis()-start);
        start=System.currentTimeMillis();
        primitiv();
        System.out.println(System.currentTimeMillis()-start);
    }
}
```


Citirea de la tastatură

Citirea de la tastatură

- ▶ `System.in`

Citirea de la tastatură

- ▶ `System.in`
- ▶ `System.in.read() ;`

Citirea de la tastatură

- ▶ `System.in`
- ▶ `System.in.read()` ;

```
try{  
    char c = (char)System.in.read();  
}  
catch(Exception ioe){  
}
```

Citirea de la tastatură

- ▶ Clasa **Scanner** din pachetul **java.util**

Citirea de la tastatură

- ▶ Clasa **Scanner** din pachetul **java.util**
- ▶ Diferite surse: tastatură, fișier, String

Citirea de la tastatură

- ▶ Clasa **Scanner** din pachetul **java.util**
- ▶ Diferite surse: tastatură, fișier, String
- ▶ În mod predefinit un obiect de tip **Scanner** extrage entități delimitate prin **caractere albe** și apoi încearcă să le interpreteze în modul cerut

Citirea de la tastatură – clasa Scanner

- ▶ Pentru tipurile **primitive** de date există metodele
 - `nextByte()`
 - `nextShort()`
 - `nextInt()`
 - `nextLong()`
 - `nextFloat()`
 - `nextDouble()`
 - `nextBoolean()`

Citirea de la tastatură – clasa Scanner

- ▶ Pentru a testa dacă **sunt disponibile valori** de anumit tip există metode ca
 - `hasNextInt()`
 - `hasNextDouble()` etc.

Citirea de la tastatură – clasa Scanner

- ▶ `hasNext()` , `next()`

tipul rezultatului întors de metoda `next()` este `String`)

- ▶ `nextLine()` , `hasNextLine()`

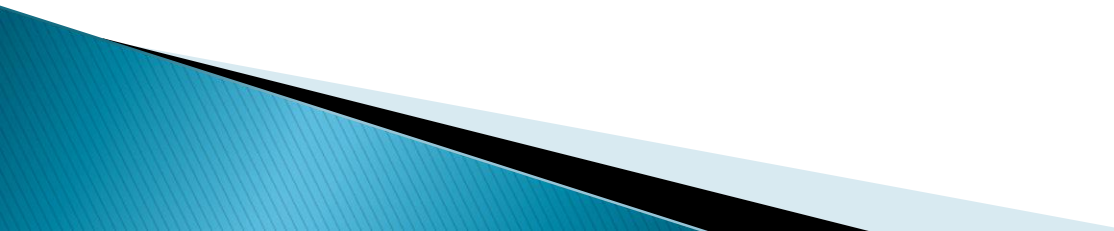
Citirea de la tastatură – clasa Scanner

▶ `import java.util.Scanner;`

Citirea de la tastatură – clasa Scanner

- ▶ `import java.util.Scanner;`
- ▶ `Scanner sc = new Scanner(System.in);`

Citirea de la tastatură – clasa Scanner

- ▶ `import java.util.Scanner;`
 - ▶ `Scanner sc = new Scanner(System.in);`
 - ▶ `int x = sc.nextInt();`
 - ▶ `double d = sc.nextDouble();`
 - ▶ `String s = sc.next();`
- 

Citirea de la tastatură – clasa Scanner

Exemplu



Citire/scriere din fișier

- ▶ `java.io.File`

Citire/scriere din fișier

- ▶ `java.io.File`
- ▶ `Scanner` – citire
- ▶ `PrintWriter` – scriere

Transmiterea parametrilor

Transmiterea parametrilor



La invocarea metodelor se folosește
apelul prin valoare

Transmiterea parametrilor



La invocarea metodelor se folosește **apelul prin valoare**

- ▶ Variabila de tip primitiv “revine” la valoarea pe care o avea înainte de apel
- ▶ Variabila de tip referință “revine” la **adresa** pe care o memora înainte de apel

```
class C{
    int a;

    C(){
    }

    C(int a1){
        a=a1;
    }

    void afis(){
        System.out.println(a);
    }
}
```

```
class TestParam{
    static void modif(C ob){
        ob.a++; //modific camp
    }

    static void modifOb(C ob){
        ob = new C(5); //modific adresa
    }

    static void creste(int x){
        x++; //modific tip primitiv
    }
}
```

```
class C{
    int a;

    C(){
    }

    C(int a1){
        a=a1;
    }

    void afis(){
        System.out.println(a);
    }
}
```

```
class TestParam{
    static void modif(C ob){
        ob.a++; //modific camp
    }

    static void modifOb(C ob){
        ob = new C(5); //modific adresa
    }

    static void creste(int x){
        x++; //modific tip primitiv
    }

    public static void main(String a[]){
        int x=1;
        creste(x);
        System.out.println(x);
        C ob = new C(1); ob.afis();
        modifOb(ob); ob.afis();
        modif(ob); ob.afis();

    }
}
```

```
import java.util.*;
class Tablou {
    static void met(int[] a) {
        a[0]= 7;
        a = new int[5];
        Arrays.fill(a,0,4,1);
    }
}
```

```
}
```

```
import java.util.*;
class Tablou {
    static void met(int[] a) {
        a[0]= 7;
        a = new int[5];
        Arrays.fill(a,0,4,1);
    }
    public static void main(String[] s) {
        int[] a = {1,2,3,4};
        for (int i=0 ; i<a.length; i++)
            System.out.print(a[i]+" ");
        System.out.println();
        met(a);
        for (int el:a)
            System.out.print(el+" ");
    }
}
```

