

# **PROGRAMARE ORIENTATĂ PE OBIECTE**



# CUPRINS CURS

- INTRODUCERE ÎN LIMBAJUL JAVA
- PROGRAMARE ORIENTATĂ PE OBIECTE – CONCEPTE GENERALE
- PROGRAMARE ORIENTATĂ PE OBIECTE ÎN LIMBAJUL JAVA
- EXCEPȚII
- FLUXURI (INTRĂRI-IEȘIRI)
- INTERFEȚE
- CLASE ABSTRACTE
- CLASE INCLUSE
- COLECȚII
- INTERFETE GRAFICE (AWT, SWING)
- APPLET-uri
- DESENARE

# BIBLIOGRAFIE



- Curs practic de Java, Cristian Frăsinaru
- Programare orientată pe obiecte in Java, Florian Moraru, Carmen Odubășteanu
- Java de la 0 la Expert, Ștefan Tanasa, s.a.
- Java - o perspectiva pragmatică, Irina Athanasiu, s.a.
- Java Tutorial,  
[www.java.sun.com/docs/books/tutorial](http://www.java.sun.com/docs/books/tutorial)
- Thinking in Java, Bruce Eckel,  
[www.bruceckel.com](http://www.bruceckel.com)

# Introducere în Java



- Tehnologia Java
- Primul program
- Structura lexicală
- Tipuri de date
- Variabile
- Instrucțiuni
- Ce este un pachet ?
- Pachetele standard (J2SDK)
- Importul unei clase sau interfețe
- Importul la cerere
- Importul static
- Vectori
- Șiruri de caractere
- Argumente de la linia de comandă

# Limbajul de programare Java

- Simplitate
- Ușurință în crearea de aplicații complexe
- Robustețe – nu există pointeri, administrarea automată a memoriei, GC
- Complet orientat pe obiecte
- Securitate
- Neutralitate arhitecturală
- Portabilitate
- Compilat și interpretat
- Performanță
- Modelat după C și C++

# Platforme de lucru Java

- J2SE (Standard Edition)

Aplicații independente, appleturi, Java Web Start

- J2ME (Micro Edition)

Programarea dispozitivelor mobile

- J2EE (Enterprise Edition)

- Aplicații complexe, pe mai multe nivele pentru sisteme eterogene, cu baze de date distribuite, etc.
- Aplicații și servicii Web: servleturi, pagini JSP, etc.

- Distribuțiile Java sunt oferite gratuit

- <http://java.sun.com>
- J2SE 1.7 SDK

# Compilat și interpretat

Limbajele de programare, în funcție de modul de execuție a aplicațiilor:

- Interpretate
  - + : simplitate, portabilitate
  - : viteza de execuție redusă
- Compilate
  - + : execuția extrem de rapidă
  - : lipsa portabilității

Java: compilat + interpretat

Compiler: sursă - cod de octeți

Interpreter: execută codul de octeți

Cod octeți  $\neq$  Cod mașină

Cod mașină - procesor

Cod octeți - JVM

JVM = Java Virtual Machine (mediul de execuție al aplicațiilor Java)

# Primul program

## 1. Scriererea codului sursă

```
class FirstApp
{
    public static void main( String args[])
    {
        System.out.println("Hello world!");
    }
}
```

Definire clasă – class *numeclasa*

- Funcția principală a unei aplicații Java propriu-zise - *public static void main( String args[])*
- Afișare - *System.out.println*



# Primul program



## 2. Salvarea fișierelor sursă

`C:\intro\FirstApp.java`

## 3. Compilarea aplicației

`javac FirstApp.java`

va rezulta: `FirstApp.class`

## 4. Rularea aplicației

`java FirstApp`

- Se poate folosi un IDE (mediu integrat de dezvoltare) pentru a realiza toți pașii anteriori (JCreator, Eclipse, NetBeans, etc.)

# Structura lexicală

Setul de caractere: Unicode

- înlocuiește ASCII
- un caracter se reprezintă pe 2 octeți
- conține 65536 de semne
- compatibil ASCII : primele 256 caractere sunt cele din ASCII
- structurat în blocuri:  
Basic, Latin, Greek, Arabic, Gothic,  
Currency, Mathematical, Arrows, Musical,  
etc.
- referire prin cod hexa:  
    \ uxxxx  
    \u03B1 -\u03C9: α - ω
- <http://www.unicode.org>

# Cuvinte cheie

- Cuvintele rezervate sunt, cu câteva excepții, cele din C++.
  - abstract, double, int, strictfp
  - boolean, else, interface, super
  - break, extends, long, switch
  - byte, final, native, synchronized
  - case, finally, new, this
  - catch, float, package, throw
  - char, for, private, throws
  - class, goto\*, protected, transient
  - const\*, if, public, try
  - continue, implements, return, void
  - default, import, short, volatile
  - do, instanceof, static, while

Incepând cu 1.5: enum.

# Identificatori

- Sunt secvențe nelimitate de litere și cifre Unicode plus simbolul “\_”, ce încep cu o literă sau “\_”.
- Identificatorii nu au voie să fie identici cu cuvintele rezervate.

Exemple: a1, FirstApp, factorial, etc.

## Convenție:

- **Nume de clasa:** prima literă mare (Complex)
- **Nume de metodă:** prima literă mică (aduna, adunaComplex)
- **Nume variabilă:** prima literă mică (var1)
- **Nume constantă:** prima literă mare sau tot numele cu litere mari (Pi, PI)

Obs: *dacă identificatorul este format din mai mulți atomi lexicali, atunci primele litere ale celorlalți atomi se scriu cu majuscule.*

# Constante

- Intregi (10, 16 -0x, 8-0)
  - normali - se reprezintă pe 4 octeți (32 biți)
  - lungi - se reprezintă pe 8 octeți (64 biți) și se termină cu caracterul L (sau l).
- Flotați: 1.0, 2e2, 3f, 4D
  - să aibă cel puțin o zecimală după virgulă
  - să fie în notație exponențială
  - să aibă sufixul F sau f pentru valorile normale - reprezentate pe 32 biți, respectiv D sau d pentru valorile duble - reprezentate pe 64 biți.
- Logici: true, false

# Constante

- Character: 'J', 'a', 'v', 'a', '\n'
- Character sau secvențe escape (permit specificarea caracterelor care nu au reprezentare grafică și reprezentarea unor caractere speciale precum backslash, apostrof, etc.)
- Secvențele escape predefinite în Java sunt:
  - '\b' : Backspace (BS)
  - '\t' : Tab orizontal (HT)
  - '\n' : Linie nouă (LF)
  - '\f' : Pagină nouă (FF)
  - '\r' : Inceput de rând (CR)
  - '\"' : Ghilimele
  - '\'' : Apostrof
  - '\\' : Backslash

# Constante

- Şiruri de caractere: "Text"
  - format din zero sau mai multe caractere între ghilimele. Caracterele care formează şirul pot fi caractere grafice sau secvenţe escape.
- Separatori: indică sfârşitul unei unităţi lexicale şi începutul alteia.  
( ) [ ] ; , .

Observaţie: instrucţiunile sunt separate prin ;

# Operatori

- atribuirea:

=

- matematici:

+, -, \*, /, %, ++, --

lval op= rval: x += 2 n -= 3

x++, ++x, n--, --n

- logici:

&&(and), ||(or), !(not)

- relaționali: <, <=, >, >=, ==, !=

- pe biți:

&(and), |(or), ^ (xor), ~ (not)

- de translație: <<, >>, >>> (shift la dreapta fără semn)

- if-else:

expresie-logica ? val-true: val-false



# Operatori

- operatorul , (virgulă) folosit pentru evaluarea secvențială a operațiilor:  
`int x=0, y=1, z=2;`
- operatorul + pentru concatenarea șirurilor  
`String s1="Ana";`  
`String s2="mere";`  
`int x=10;`  
`System.out.println(s1 + " are " + x + " " + s2);`
- operatori pentru conversii (cast) : (tip-de-data)  
`int a = (int)'a';`  
`char c = (char)96;`  
`int i = 200;`  
`long l = (long)i; //widening conversion`  
`long l2 = (long)200;`  
`int i2 = (int)l2; //narrowing conversion`

# Comentarii

Există trei feluri de comentarii:

- Comentarii pe mai multe linii, închise între `/*` și `*/`.
- Comentarii pe mai multe linii care țin de documentație, închise între `/**` și `*/`.
  - Textul dintre cele două secvențe este automat mutat în documentația aplicației de către generatorul automat de documentație javadoc.
- Comentarii pe o singură linie, care încep cu `//`.

# Tipuri de date

Tipurile primitive:

- aritmetice
  - întregi: byte (1 octet), short(2), int (4), long (8)
  - reale: float (4), double (8)
- caracter: char (2)
- logic: boolean (true și false)

Tipul referință:

- Vectorii, clasele și interfețele
- Valoarea unei variabile de acest tip este, spre deosebire de tipurile primitive, o referință (adresă de memorie) către valoarea sau mulțimea de valori reprezentată de variabila respectivă.

Nu există: pointer, struct și union.

# Variabile

- Declararea variabilelor:

Tip numeVariabila;

- Inițializarea variabilelor:

Tip numeVariabila = valoare;

- Declararea constantelor:

final Tip numeVariabila;

final double PI = 3.14;

int valoare = 100;

long numarElemente = 12345678L;

String bauturaMeaPreferata = "apa";

# Categorii variabile

- a. Variabile membre ale unei clase, declarate în interiorul unei clase, vizibile pentru toate metodele clasei respective cât și pentru alte clase în funcție de nivelul lor de acces
- b. Parametrii metodelor, vizibili doar în metoda respectivă
- c. Variabile locale, declarate într-o metodă, vizibile doar în metoda respectivă
- d. Variabile locale, declarate într-un bloc, vizibile doar în blocul respectiv.
- e. Parametrii de la tratarea excepțiilor

# Categorii variabile

```
class Exemplu {  
    int a;  
    public void metoda(int b) {  
        a = b;  
        int c = 10;  
        for(int d=0; d < 10; d++) {  
            c --;  
        }  
        try {  
            a = b/c;  
        } catch(ArithmeticException e) {  
            System.err.println(e.getMessage());  
        }  
    }  
}
```

# Instrucțiuni



- Instrucțiuni de decizie:  
if-else, switch-case
- Instrucțiuni de salt (repetitive):  
for, while, do-while
- Instrucțiuni pentru tratarea excepțiilor:  
try-catch-finally, throw
- Alte instrucțiuni:  
break, continue, return, label:

# Instrucțiuni de decizie

- **if-else**

```
if (expresie-logica) {  
    ...  
}
```

**Sau:**

```
if (expresie-logica) {  
    ...  
} else {  
    ...  
}
```

- **switch-case**

```
switch (variabila) {  
case valoare1:  
    ...  
    break;  
case valoare2:  
    ...  
    break;  
default:  
    ...  
}
```



# Instrucțiuni de salt (repetitive)

- **for**

```
for(initializare; expresie-logica; pas-iteratie) {  
    //Corpul buclei  
}
```

Exemplu:

```
for(int i=0, j=100 ; i < 100 && j > 0; i++, j--) {  
    ...  
}
```

- **while**

```
while (expresie-logica) {  
    ...  
}
```

- **do-while**

```
do {  
    ...  
} while (expresie-logica);
```

# Alte instrucțiuni

- `break`: părăsește forțat corpul unei structuri repetitive.
- `continue`: termină forțat iterația curentă a unui ciclu și trece la următoarea iterație.
- `return [valoare]`: termină o metodă și, eventual, returnează o valoare.
- `numeEticheta`: definește o etichetă.
- Nu există `goto`
- Pot fi definite etichete folosite astfel:
  - `break numeEticheta`
  - `continue numeEticheta`

# Exemplu de folosire a etichetelor

```
i=0;
eticheta:
while (i < 10) {
    System.out.println("i="+i);
    j=0;
    while (j < 10) {
        j++;
        if (j==5) continue eticheta;
        //sau: if (j==5) break eticheta;
        System.out.println("j="+j);
    }
    i++;
}
```

# Ce este un pachet ?



Pachet = Colecție de clase și interfețe

Scopul:

- Organizarea lucrului
- Găsirea și utilizarea mai ușoară a claselor
- Evitarea conflictelor de nume
- Controlul accesului

# Pachetele standard (J2SDK)

- **java.lang** - clasele de bază ale limbajului Java
- **java.io** - intrări/ieșiri, lucrul cu fișiere
- **java.util** - clase și interfețe utile
- **java.applet** - dezvoltarea de appleturi
- **java.awt** - interfața grafică cu utilizatorul
- **java.awt.event** - tratare evenimente
- **java.beans** - scrierea de componente reutilizabile
- **java.net** - programare de rețea
- **java.sql** - lucrul cu baze de date
- **java.rmi** - execuție la distanță
- **java.security** - mecanisme de securitate
- **java.math** - operații matematice cu numere mari
- **java.text** - lucrul cu texte, date și numere independent de limbă
- **java.lang.reflect** - introspecție
- **javax.swing** - interfața grafică cu utilizatorul, mult îmbogățită față de AWT.

# Folosirea membrilor unui pachet

1. specificarea numelui complet:

**numePachet.NumeClasa.**

Button - numele scurt al clasei

java.awt - pachetul din care face parte

java.awt.Button - numele complet al clasei

Exemplu:

```
java.awt.Button b1 = new java.awt.Button("OK");
```

```
java.awt.Button b2 = new java.awt.Button("Cancel");
```

```
java.awt.TextField tf1 =
```

```
    new java.awt.TextField("Neplacut");
```

```
java.awt.TextField tf2 =
```

```
    new java.awt.TextField("Tot neplacut");
```

# Importul unei clase sau interfețe

## 2. importul clasei respective

**import numePachet.numeClasa;**

//Pentru exemplul nostru:

import java.awt.Button;

import java.awt.TextField;

...

Button b1 = new Button("OK");

Button b2 = new Button("Cancel");

TextField tf1 = new TextField("Placut");

TextField tf2 = new TextField("Foarte placut");

//Problema:

import java.awt.Button;

import java.awt.TextField;

import java.awt.Rectangle;

import java.awt.Line;

import java.awt.Point;

import java.awt.Polygon;

# Importul la cerere

## 3. importul întregului pachet

```
import numePachet.*;
```

//Pentru exemplul nostru:

```
import java.awt.*;
```

...

```
Button b = new Button("OK");
```

```
Point p = new Point(0, 0);
```

```
import java.awt.C*; = eroare
```

Importate automat:

- pachetul java.lang
- pachetul curent
- pachetul implicit (fără nume)



# Ambiguități

```
import java.awt.*;  
// Contine clasa List  
import java.util.*;  
// Contine interfata List  
...  
List x; //Declaratie ambigua  
java.awt.List a = new java.awt.List(); //corect  
java.util.List b = new ArrayList(); //corect
```

# Importul static

- Referirea constantelor statice ale unei clase:

`import static numePachet.NumeClasa.*;`

// Inainte de versiunea 1.5

```
import java.awt.BorderLayout.*;
```

...

```
fereastră.add(new Button(), BorderLayout.CENTER);
```

// Incepand cu versiunea 1.5

```
import java.awt.BorderLayout.*;
```

```
import static java.awt.BorderLayout.*;
```

...

```
fereastră.add(new Button(), CENTER);
```

# Vectori

- Declararea

```
Tip[] numeVector; sau  
Tip numeVector[];
```

- Instanțierea

```
numeVector = new Tip[nrElemente];
```

- Inițializarea (opțional)

```
String culori[] = {"Rosu", "Galben"};
```

```
v = new int[10];
```

```
//aloca spatiu pentru 10 intregi: 40 octeti
```

```
c = new char[10];
```

```
//aloca spatiu pentru 10 caractere: 20 octeti
```

# Vectori

- Declararea și instanțierea pot fi făcute simultan:

`Tip[] numeVector = new Tip[nrElemente];`

- Primul indice al unui vector este 0, deci pozițiile unui vector cu n elemente vor fi cuprinse între 0 și n - 1.
- Nu sunt permise construcții de genul *Tip numeVector[nrElemente]*, alocarea memoriei făcându-se doar prin intermediul operatorului new.

`int v[10]; //illegal`

`int v[] = new int[10]; //corect`

# Tablouri multidimensionale

Tablou multidimensional = vector de vectori.

Tip `mat[][] = new Tip[nrLinii][nrColoane];`

- *mat[i]* este linia i a matricii și reprezintă un vector cu nrColoane elemente iar *mat[i][j]* este elementul de pe linia i și coloana j.
- Dimensiunea unui vector  
Variabila **length**:  
`int []a = new int[5];`  
`// a.length are valoarea 5`  
`int m[][] = new int[5][10];`  
`// m[0].length are valoarea 10`

# Copierea vectorilor

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

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

```
// Varianta 1
```

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

```
    b[i] = a[i];
```

```
// Varianta 2
```

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

```
// Nu are efectul dorit
```

```
    b = a;
```

# Sortarea vectorilor - clasa Arrays

Metode din java.util.Arrays:

- sort - (QuickSort -  $O(n \log(n))$ )  
int v[]={3, 1, 4, 2};  
java.util.Arrays.sort(v);  
// Sorteaza vectorul v  
// Acesta va deveni {1, 2, 3, 4}
- binarySearch
- equals
- fill

Vectori cu dimensiune variabilă și eterogeni:

- Vector, ArrayList, etc. din pachetul java.util.

# Argumente de la linia de comandă

- Trimiterea argumentelor

```
java NumeAplicatie [arg0 arg1 . . .]
```

```
java Test Java "Hello Duke" 1.5
```

- Primirea argumentelor:

```
public static void main (String args[]){
```

```
    /* args[0] va fi "Java"
```

```
    args[1] va fi "Hello Duke"
```

```
    s.a.m.d.
```

```
    */
```

```
}
```

- Numărul argumentelor:

```
public static void main (String args[]) {
```

```
    int numarArgumente = args.length ;
```

```
}
```



# Exemplu

```
public class Salut {  
    public static void main (String args[]) {  
        if (args.length == 0) {  
            System.out.println( "Numar insuficient de  
argumente!");  
            System.exit(-1); //termina aplicatia  
        }  
        String nume = args[0]; //exista sigur  
        String prenume;  
        if (args.length >= 1)  
            prenume = args[1];  
        else  
            prenume = ""; //valoare implicita  
        System.out.println("Salut " + nume + "      " +  
prenume);  
    }  
}
```

# Afişarea argumentelor

```
public class Afisare {  
    public static void main (String[] args) {  
        for (int i = 0; i < args.length; i++)  
            System.out.println(args[i]);  
    }  
}
```

**java Afisare Hello Java**

**Hello**

**Java**

**java Afisare "Hello Java"**

**Hello Java**

# Argumente numerice

- Argumentele de la linia de comandă sunt siruri de caractere
- Transformarea şir - număr se face cu metode de tipul:
  - Integer.parseInt
  - Double.parseDouble
  - Float.parseFloat, etc.

[illegible]

# Șiruri de caractere

- `char[]`

- `String`

`String s = "abc";`

`String s = new String("abc");`

`char data[] = {'a', 'b', 'c'};`

`String s = new String(data);`

- `StringBuffer`

Un șir de caractere constant (nu se doresc schimbări în porțiuni din conținutul său pe parcursul execuției programului) va fi declarat de tipul `String`, altfel va fi declarat de tip `StringBuffer`. `StringBuffer` pune la dispoziție metode pentru modificarea conținutului șirului, cum ar fi: `append`, `insert`, `delete`, `reverse`.

# Adrese utile



- [www.java.com](http://www.java.com)  
Toată lumea
- [www.java.net](http://www.java.net)  
Comunitate
- [java.sun.com](http://java.sun.com)  
Dezvoltatori
- [jguru.com](http://jguru.com)  
Cursuri
- [jars.com](http://jars.com)  
Resurse
- [javaworld.com](http://javaworld.com), [javareport.com](http://javareport.com)  
Articole
- [javaalmanac.com](http://javaalmanac.com)  
Exemple de cod
- [google:java](https://www.google.com/search?q=java)