

# Programarea orientată pe obiecte

SERIA CB



## JDK

**Java Development Kit (JDK)** reprezintă o implementare a limbajului Java, o mașină virtuală alcătuită din Java Virtual Machine (JVM) și instrumente utile pentru dezvoltarea și testarea programelor scrise în Java și care rulează pe platforma Java.

Pe lângă JDK, trebuie folosit și un mediu de dezvoltare integrat (**Integrated Development Environment – IDE**) - NetBeans (Apache NetBeans, n.d.), Eclipse (Eclipse Foundation, n.d.), IntelliJ IDEA (Jetbrains, n.d.). Editarea, compilarea si rularea programelor Java sunt astfel integrate într-o interfață grafică unică.

## GENERALITĂȚI

### CARACTERISTICI JAVA

- Usurinta in utilizare si invatare*
- Independenta de platforma*
- Extensibilitate ca biblioteci disponibile*
- Volum foarte mare de informatii si de resurse disponibile pentru invatare.*
- Robustete* – detectarea erorilor de catre compilator si gestionarea situatiilor de exceptie
- Alocarea memoriei* se face eficient prin intermediul unui mecanism de tip stiva
- Rularea pe mai multe fire de executie* si executarea mai multor task-uri simultan

### EDITII JAVA

- Standard Edition* (SE; Oracle)
- Enterprise Edition* (EE; Oracle)
- Micro Edition* (ME; Oracle)
- Versiuni specifice, de exemplu: *Card* (Oracle) – pentru aplicații ce rulează pe smart cards sau device-uri cu memorie sau capabilități de procesare limitate – *Java TV* (Oracle) – o versiune a lui Java ME ce implementează soluții pentru sisteme TV sau mediabox-uri.

```
1  public class Main {
2      public static void main(String[] args) {
3          // Afisare mesaj la consola
4          System.out.println("Acesta este un exemplu!");
5      }
6  }
```

## COMPILAREA ȘI EXECUȚIA JAVA

**Linia 1** definește o clasă. Fiecare program Java trebuie să aibe cel puțin o clasă. Fiecare clasă are un nume. Prin convenție, numele clasei trebuie să înceapă cu literă mare. În acest exemplu, numele clasei este **Main**.

**Linia 2** definește metoda `main`. Pentru a rula o clasă, aceasta trebuie să conțină o metoda numită `main`. Programul este executat din metoda `main`.

**Linia 3** conține un comentariu. Acestea pot fi de două tipuri: pe linie (`//`) sau în bloc (`/* */`).

Pe **Linia 4** este afișat un mesaj la consolă. Cuvintele cheie rezervate limbajului sunt `class`, `public`, `static`, `void`. Limbajul Java este case sensitive. Astfel, este greșit să se înlocuiască cuvântul `main` cu `Main`.

După scrierea programului, acesta trebuie compilat înainte de a fi executat. Dacă programul are erori de compilare, acestea trebuie rezolvate și programul recompilat. Dacă programul are erori la rulare (runtime) sau nu furnizează rezultatul corect, el trebuie modificat, recompilat și executat din nou. Codul Java este scris într-un fișier sursă. Compilatorul traduce fișierul sursă într-un fișier **bytecode**.

Următoarea comandă compilează `Main.java`:

```
javac Main.java
```

Înainte de compilare și rulare, trebuie instalat și configurat JDK-ul. Dacă nu sunt erori de sintaxă, compilatorul generează un **fișier bytecode cu extensia .class**. Astfel, pentru exemplul de mai sus, se generează fișierul `Main.class`.

## TIPURI DE DATE NUMERICE

Java are 8 tipuri de date primitive pentru valorile numerice, caracter și boolean (a se vedea tabelul).

Când se încearcă să se stocheze o valoare care se apropie de limita maximă a intervalului tipului de date, se produce un **overflow**. Trebuie avut grijă, întrucât Java nu generează avertizări sau erori referitoare la acest lucru. Pe de altă parte, atunci când se stochează o valoare în virgulă mobilă foarte mică (foarte aproape de zero), se produce **underflow**. Aici nu se solicită atenția programatorului deoarece Java aproximează aceste valori foarte mici la 0.

## ELEMENTE SPECIFICE DE LIMBAJ

### Identificatori

Identificatorul este o secvență de **litere, cifre, liniuța de subliniere ( \_ ) și caracterul dolar ( \$ )**, dintre care primul caracter nu trebuie să fie cifră. Un identificator poate fi `true`, `false` sau `null`, de orice lungime și nu trebuie să se confunde cu un cuvânt rezervat limbajului.

Identificatorii pot numi **variabile, constante, metode, clase și pachete**.

**Nu este recomandat totuși să se folosească \$ ca prim caracter.**

### Variabile

Variabilele sunt utilizate pentru a stoca valori ce se vor folosi mai târziu în program.

Declararea unei variabile este o modalitate prin care se precizează compilatorului cantitatea de memorie pe care trebuie să o aloce în program, bazându-se pe tipul de date al variabilei.

Declararea se face astfel:

```
datatype variableName;
```

Exemplu:

```
int count;
float salary;
double x, y, z;
```

Prin convenție, numele unei variabile se scrie **cu litere mici**.

Dacă numele variabilei este alcătuit din mai multe cuvinte concatenate, atunci **se scrie cu literă mare prima literă începând de la al doilea cuvânt**.

# Elementele limbajului Java



## Constante

Valoarea unei variabile se schimbă de-a lungul execuției unui program, dar cea a unei constante nu.  
Pentru a declara o constantă se folosește sintaxa:

```
final datatype NUMECONSTANTA = VALOARE;
final double PI = 3.1415926;
```

`final` este un cuvânt cheie din Java care se utilizează pentru a declara o constantă.

Constantele se scriu **cu literă mare** și au ca avantaj faptul că nu trebuie să se repete în scrierea programului aceeași valoare, aduce lizibilitate codului iar, în cazul în care valoarea ei trebuie schimbată, acest lucru se va produce într-un singur loc în program.

## Operatorii numerici

Operatorii numerici sunt **+**, **-**, **/**, **\***, și **%**.

Când ambii operanzi sunt numere întregi, rezultatul operației de împărțire este un întreg.

De exemplu  $5 / 2 = 2$

Dacă unul dintre operanzi este număr real, atunci rezultatul este și el un număr real. De exemplu  $5.0 / 2 = 2.5$ . Operatorul **%** poate fi folosit și pentru numere negative:  $-5 \% 2 = -1$ .

Acești operatori pot fi combinați cu operatorul de atribuire **=**, rezultând: **+=**, **-=**, **/=**, **\*=** și **%=**. Asemănător limbajului C, avem și operatorii de incrementare prefixați și postfixați **++** și **--**.

## Conversii de tip

Când se încearcă executarea unei operații în care avem un număr întreg și un număr real, Java convertește în mod automat numărul întreg la tipul real. Spre exemplu,  $2 * 5.0$  va fi echivalent cu  $2.0 * 5.0$ . Se poate atribui o valoare unei variabile care acoperă o plajă mai largă de valori. În sens invers nu se poate, atâta timp cât nu se folosește operația de conversie de tip (eng., „**type casting**”). Există două tipuri de casting: implicit la un tip mai general (eng., „**widening**” – se realizează automat, fără a se preciza acest lucru explicit la trecerea de la un tip de date mai specific la unul mai gneral) și explicit la un tip mai specific (eng., „**narrowing**” – trebuie obligatoriu precizat).

### Exemplu widening:

```
int numarIntreg = 9;
double numReal = numarIntreg;
System.out.println(numarIntreg);
// Se va afisa 9
System.out.println(numReal);
// Se va afisa 9.0
```

### Exemplu narrowing:

```
int x = (int) 1.5;
```

În exemplul de mai sus, valoarea `double 1.5` este convertită la tipul `int`, prin urmare se va trunchia la partea întreagă, deci va fi egală cu 1 și se va atribui variabilei `x`. Acesta este un caz de *narrowing casting* ce trebuie explicitat în mod obligatoriu prin plasarea tipului între paranteze.

## Tipul String

Pentru a reprezenta mai multe caractere se folosește tipul `String`. Atunci când se atribuie o valoare de tipul `String` unei variabile, ea trebuie încadrată între ghilimele. **String este de fapt o clasă predefinită în limbajul Java**, alături de `System`, `Scanner` etc. **String nu este un tip de bază sau primitiv, ci este cunoscut ca un tip de referință**. Cel mai des întâlnit operator cu care lucrează stringurile este operatorul de concatenare **+** (cu variația **+=**). Astfel, pot fi concatenate atât două string-uri, cât și un string cu o valoare non-string. Cea din urmă este automat convertită în acest caz la tipul `String`.

| Tip    | Interval de valori   | Dimensiune          |
|--------|--|---------------------|
| byte   | -2 <sup>7</sup> (-128) până la 2 <sup>7</sup> -1 (127)                         | 8 biti              |
| short  | -2 <sup>15</sup> (-32768) până la 2 <sup>15</sup> - 1 (32767)                  | 16 biti             |
| int    | -2 <sup>31</sup> până la 2 <sup>31</sup> - 1                                   | 32 biti             |
| long   | -2 <sup>63</sup> până la 2 <sup>63</sup> - 1                                   | 64 biti             |
| float  | Negative: -3.4E+38 pana la -1.4E-45<br>Pozitive: 1.4E-45 pana la 3.4 E+38      | 32 biti<br>IEEE 754 |
| double | Negative: -1.79E+308 pana la -4.9E-324<br>Pozitive: 4.9E-324 pana la 1.79E+308 | 64 biti<br>IEEE 754 |

În Java există noțiunea de **atribuire expresie**.

```
System.out.println(x = 1);
```

este echivalent cu

```
x = 1;
```

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

Atribuirea `x = y = z = 2` este corectă și este echivalentă cu:

```
x = 2;
```

```
y = 2;
```

```
z = 2;
```

Într-o atribuire, tipul de date al variabilei din stânga trebuie să fie compatibil cu cel al variabilei din dreapta. De exemplu, atribuirea `int x = 1.0` este incorectă deoarece `x` este de tipul `int` iar `1.0` este de tipul `double`.

## Tipul caracter

Tipul `char` denotă un singur caracter, reprezentat pe **16 biți**, ce trebuie încadrat între apostroafe.

Modalitatea de codificare a caracterelor în limbajul Java poartă denumirea de Unicode, fiind o schemă de codificare dezvoltată de Unicode Consortium pentru procesarea textelor în diverse limbaje de programare. S-a observat că cele 65 536 caractere posibile în reprezentarea pe 16 biți nu sunt suficiente pentru a reprezenta toate caracterele posibile. Standardul Unicode a fost extins până la 1112064 caractere. **Caracterele Unicode sunt exprimate în hexazecimal de la '\u0000' la '\uFFFF', și includ caracterele din standardul ASCII.** Câteva exemple de conversii la tipul `int` sunt exemplificate în continuare.

```
char ch = (char) 66.9;
System.out.println(ch);
```

În exemplul de mai sus se va afișa valoarea 'B'.

```
int i = '0' + '1';
System.out.println(i);
```

Va afișa valoarea 95 deoarece lui '0' îi corespunde codul 47, iar lui '1' îi corespunde codul 48.

```
int i = (int) 'a';
System.out.println(i);
```

Va afișa valoarea 97.

## Instrucțiuni

În limbajul Java, instrucțiunea de decizie **if** are aceeași formă ca în C sau C++, cu diferența că evaluarea unei expresii conduce la valorile de adevăr `true` sau `false`, care sunt de tipul `boolean`. Secvențele de mai jos sunt echivalente:

```
if(x % 2 == 0)
    par = true;
else
    par = false;
```

Echivalentă cu:

```
boolean par = x % 2 == 0
```

Operatorii logici sunt **!** (NOT), **&&** (ȘI), **||** (SAU) și **^** (SAU EXCLUSIV).

Similar se întâmplă cu instrucțiunea **switch** și cu cea **condițională**, care utilizează operatorul ternar **?:**

**expresie\_booleană ? expresie1 : expresie2;**

```
max = (a > b) ? a : b;
```

```
System.out.println((x % 2 == 0) ?
"par" : "impar");
```

**Instrucțiunile repetitive din limbajul Java sunt while, do while și for**, cu formă similară celei din limbajele C sau C++. Pentru a încheia iterația curentă și a trece automat la următoarea se folosește instrucțiunea **continue**, iar pentru a încheia ciclul repetitiv se utilizează **break**.



# Afișarea datelor cu format

Pentru afișarea datelor cu format se utilizează metoda

```
System.out.printf
```

Cei mai des întâlniți specificatori de format sunt:

- %b** – **boolean**
- %c** – **character**
- %s** – **String**
- %d** – **întreg**
- %f** – **număr real**
- %e** – **real în notatie științifică**

Exemple de specificatori de format cu precizie:

- **%4c** – afișează caracterul și lasă 3 spații înaintea lui.
- **%5d** – afișează numărul întreg de dimensiune cel puțin 5. Dacă numărul de cifre este mai mic decât 5, se adaugă spații înainte. Dacă numărul de cifre este mai mare decât 5, dimensiunea lui se mărește automat.
- **%10.2f** – afișează numărul în virgulă mobilă, de dimensiune cel puțin 10, incluzând virgula și două cifre după virgulă. Deci, sunt 7 cifre alocate înainte de virgulă. Dacă numărul de cifre de dinainte de virgulă este mai mic de 7, se adaugă spații înainte de număr. Dacă numărul de cifre este mai mare decât 7, dimensiunea este automat incrementată.
- **%12s** – dacă dimensiunea șirului de caractere este mai mică de 12, se adaugă spații înaintea lui, altfel se suplimentează dimensiunea.

Tipul datelor afișate trebuie să fie compatibil cu specificatorii. Dacă vrem să afișăm valoarea reală 40.0, trebuie să folosim %f sau %e. Dacă încercăm să afișăm valoarea 40, nu putem folosi %f sau %e. Semnul % denotă un specificator. Dacă doriți să afișați literalul %, folosiți %%.

## APLICAȚII

1. Compilați și rulați din terminal aplicația *MyHello.java*, ce afișează mesajul *Hello, World!*. Aveți nevoie de un compilator (puteți folosi **javac**)

Comenzile sunt:

```
javac MyHello.java
java MyHello
```

Va afișa:

```
Hello, World!
```

2. Se consideră următoarea secvență:

```
boolean yes = true;
boolean no = false;
int loVal = -999;
int hiVal = 999;
double grade = 87.5;
double amount = 50.0;
String hello = "world";
```

Care este rezultatul următoarelor expresii?

```
yes == no || grade > amount
amount == 40.0 || 50.0
hiVal != loVal || loVal < 0
true || hello.length() > 0
hello.isEmpty() && yes
grade <= 100 && !false
!yes || no
grade > 75 > amount
amount <= hiVal && amount >= loVal
no && !no || yes && !yes
```

3. Ce se afișează la rularea clasei Demo?

```
public class Demo {
    public static void main(String[] args) {
        float a = 100.25f;
        long b = (long)a;
        System.out.println("value of a: "+a);
        System.out.println("value of b:"+b);
        int c = (int)b;
        System.out.println("value of c:"+c);
        byte d = (byte)c;
        System.out.println("value of d:"+d);
    }
}
```

4. Ce afișează următoarea secvență?

```
int a = 5;
System.out.println(a + -a - a++ % 10);
System.out.println(a - a + --a / 10);
for (int i = 2; i < 5;) {
    i++;
    a += a;
}
System.out.println("a = " + a);
```