

Lab1: Noțiuni introductive

Cuprins

1. O scurtă introducere
2. Caracteristici ale limbajului:
3. Crearea unei aplicații simple
 - Observații
 - Cum se realizează compilarea?
 - Cum se realizează rularea programului?
 - Folosirea argumentelor din linia de comandă
4. Elemente de bază ale limbajului Java
 - 4.1. Comentarii
 - 4.2. Identificatori
 - 4.3. Constante (Literali)
 - 4.4. Tipuri primitive
 - Observații
 - 4.5. Declararea variabilelor
 - 4.6. Operatori
 - Observații
 - 4.7. Instrucțiuni
5. Instalare IDE
6. Exerciții
6. Recomandări

1. O scurtă introducere

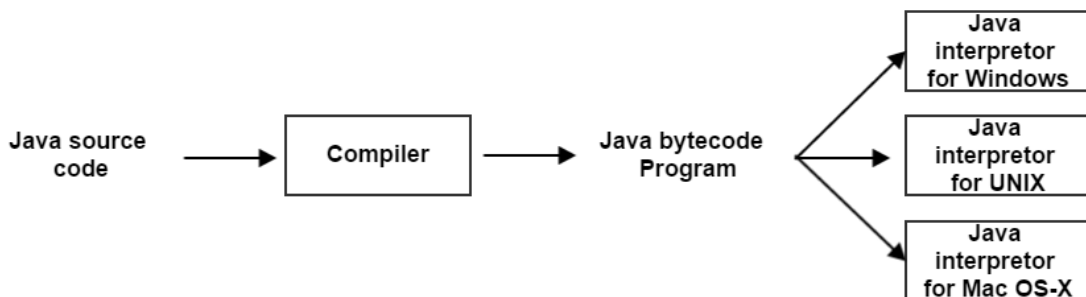
- Java este un limbaj de nivel înalt, dezvoltat în cadrul firmei Sun Microsystems, cumpărată ulterior de Oracle Corporation.
- Dezvoltarea limbajului a început în anul 1991, de către James Gosling, iar prima versiune a fost făcută public în 1995, prin Java 1.0.
- Principalul principiu care stă la baza limbajului este: *“Write once, run everywhere”*, ceea ce înseamnă că programele Java compilate pot rula pe orice platformă ce suportă Java, fără a fi nevoie de recompilare (mai multe detalii mai jos).
- Java este unul dintre cele mai populare limbaje de programare, fiind folosit la dezvoltarea unor tehnologii în cele mai diverse domenii. Tehnologiile Java au fost grupate în așa-numitele platforme de lucru, ce conțin seturi de librării scrise în Java, împreună cu diverse programe utilitare, utilizate pentru dezvoltarea de aplicații sau componente destinate unei anumite categorii de utilizatori. Printre acestea, se numără:
 - *J2SE (Java SE)* - Java Standard Edition, platforma standard de lucru ce oferă suport pentru crearea de aplicații independente și appleturi;
 - *J2ME (Java ME)* - Java Micro Edition - folosită pentru programarea dispozitivelor mobile
 - *J2EE (Java EE)* - Java Enterprise Edition, platforma folosită pentru crearea de aplicații complexe, formate din componente ce trebuie să ruleze în medii eterogene, dezvoltarea unor aplicații și servicii Web etc.

2. Caracteristici ale limbajului:

- **Simplitate:**
 - s-a eliminat lucrul liber cu pointeri, aritmetica pe pointeri (în spate, Java Virtual Machine, despre care vom vorbi în cele ce urmează, lucrează cu pointeri la obiecte, iar noi folosim pointeri implicați pentru zone de memorie alocate pentru obiectele respective);
 - nu există moștenirea multiplă, așa cum este definită în C++;
 - s-a eliminat supraîncărcarea operatorilor;
 - nu mai există funcții independente, variabile globale;
 - se realizează colectarea automată a “reziduurilor”. Java permite crearea explicită de obiecte, de tipul unei clase. Distrugerea acestora este preluată de *colectorul de reziduuri* (garbage collector), care marchează obiectele care nu mai sunt folosite

și eliberează spațiul ocupat de ele; eliberarea se face periodic sau atunci când spațiul disponibil curent nu mai satisface o nouă cerere de alocare de memorie.

- **Complet orientat pe obiecte.** Unitatea de programare fundamentală în Java este *clasa*.
- **Portabilitate:** “Write once, run everywhere”
- **Securitate**
- **Ușurința în ceea ce privește lucrul în rețea**
- **Java este compilat și interpretat**, adică:
 - un program sursă Java trebuie mai întâi compilat
 - rezultatul compilării este un fișier “*bytecode*” - o secvență de instrucțiuni de asamblare pentru așa-numita *mașina virtuală Java* (JVM), care nu depinde de mașina gazdă pe care va fi executat programul (asigurând portabilitatea și independența de platformă).
 - acesta poate fi transferat pe orice mașină. Fiecare mașină gazdă capabilă să execute programe Java dispune de un *interpretor*, care convertește reprezentarea “bytecode” în instrucțiuni mașină proprii, care sunt apoi executate; conversia are loc la lansarea executării.



3. Crearea unei aplicații simple

Să se afișeze argumentele date de la linia de comandă.

```
class ArgumentsPrinter {
    public static void main(String[] args) {
        System.out.println("Au fost trimise " + args.length + " argumente");
        for(int i = 0; i < args.length; ++i) {
            System.out.println(args[i]);
        }
    }
}
```

Observații

- *Clasa* este unitatea de programare fundamentală în Java. Orice clasă este alcătuită din *câmpuri*, *metode* (ce formează împreună membrii clasei) și *constructori*.
- Toate aplicațiile Java conțin o *clasă principală* în care trebuie să existe metoda `main`. Aceasta este metoda principală: la executarea programului, sistemul detectează și execută metoda `main` a clasei al cărei nume coincide cu numele fișierului bytecode rezultat în urma compilării clasei care conține metoda principală.
- Clasa de mai sus nu are specificat niciun modificador de acces. Implicit, o astfel de clasă se numește “*package-private*” (vom vorbi mai multe despre modificatorii de acces în cele ce urmează și în laboratoarele viitoare).
- Este *indicat*, dar nu obligatoriu în acest caz, ca numele fișierului sursă să coincidă cu numele clasei care conține clasa principală. Dacă, în schimb, un fișier conține o clasă *publică* (un fișier poate conține cel mult o clasă publică), atunci numele acestuia **trebuie** să coincidă cu numele clasei publice.

Cum se realizează compilarea?

- Se folosește compilatorul Java, `javac`.
- Apelul compilatorului se va face pentru fișierul ce conține clasa principală a aplicației. Acesta va genera câte un fișier separat *pentru fiecare clasă a programului*, cu extensia `.class`, amplasate implicit în același director cu fișierele sursă.
- Presupunând că fișierul ce conține codul sursă de mai sus se numește `FirstApp.java`, apelul se va face:

```
javac FirstApp.java
```

și în cazul unei compilări reușite se va genera fișierul `ArgumentsPrinter.class`



Compilați programul Java de mai sus din linia de comandă. Adăugați modificadorul `public` clasei `ArgumentsPrinter` și observați rezultatul compilării. Corectați problema.



Pentru a putea compila din linia de comandă pe Windows, trebuie setată variabila de mediu `JAVA_HOME`, care este calea către folderul unde este Java JDK instalat (de exemplu: `C:\Program Files\Java\jdk-version`), iar binarul `javac` trebuie adăugat în `PATH`.



Pentru Linux, este necesară adaugarea în fișierul `.bashrc` a liniei:

```
export JAVA_HOME=/usr/lib/jvm/java-version
```

Cum se realizează rularea programului?

- Se folosește interpretorul `java`, apelat pentru unitatea de compilare corespunzătoare clasei principale, cu omiterea extensiei `.class`.

```
java numeClasa [listăDeArgumente]
```

unde *numeClasă* este numele clasei principale și *listăDeArgumente* reprezintă o listă opțională de argumente, primite de aplicație ca parametru al metodei `main`.

Ex: `java ArgumentsPrinter 1 2 3`



Interpretorul are ca prin argument numele clasei principale și nu numele unui fișier, deci un apel de tipul: `java C:\Users\labs\ArgumentsPrinter` nu este corect! Trebuie să fim poziționați în directorul ce conține fișierul compilat.

Folosirea argumentelor din linia de comandă

- Un program Java poate primi oricâte argumente din linia de comandă în momentul lansării ei, separate prin spații.
- În cazul în care un argument conține spații, acesta trebuie specificat între ghilimele.
- Interpretorul va parcurge linia de comandă cu care a fost lansată aplicația și, în cazul în care există argumente, le trimite aplicației sub forma unui vector de șiruri. Acesta este primit ca parametru al metodei `main`.



În cazul în care unele argumente sunt valori numerice, ele vor trebui convertite din șiruri de caractere în numere. Acest lucru se realizează prin apelarea unor metode de tipul `parseTipNumeric`, din clasa corespunzătoare tipului în care vrem să facem conversia. Mai multe detalii în cele ce urmează.

4. Elemente de bază ale limbajului Java

4.1. Comentarii

Java suportă trei tipuri de comentarii:

- comentarii tradiționale:

```
/* Acesta este un comentariu,  
ce se poate extinde pe mai multe linii. */
```

- comentarii de sfârșit de linie:

```
// Alt comentariu la finalul liniei
```

- comentarii pentru documentație[1]:

```
/**
 * Acesta este un comentariu folosit de
 * JavaDoc pentru generarea documentatiei.
 * Poate contine taguri descriptive, cum ar fi:
 * @param name Descrierea parametrului name
 * @return Descrierea rezultatului întors
 * @see referință Link catre un alt element de doc
 * @version version etc.
 */
```

4.2. Identificatori

- În Java, identificatorii încep cu o literă (este permis și simbolul ‘_’), și pot continua cu litere și / sau cifre.
- Prin litere, înțelegem atât literele obișnuite ale limbii engleze, mici și mari, cât și pe cele din alte limbi.
- Un identificator poate avea orice lungime. Este indicat să se folosească identificatori cât mai sugestivi pentru rolul lor, pentru a facilita înțelegerea codului.
- Anumiți identificatori au o semnificație predefinită, ce nu poate fi modificată. Aceștia sunt *cuvintele - cheie* ale limbajului. Printre acestea, amintim:

abstract	boolean	break	byte	case
catch	char	class	continue	default
do	double	else	extends	final
finally	float	for	if	implements
import	instanceof	int	interface	long
new	null	package	private	protected
public	return	short	static	super
switch	synchronized	this	throw	
throws	try	void	while	

4.3. Constante (Literali)


a. Literalii întregi:

- Sunt acceptate 3 baze de numerație: baza 10, baza 16 (ex. **0xA**) și baza 8 (ex. **0**, **012**)
- Pot fi de două tipuri:
 - *normali*: se reprezintă pe 32 de biți: 0, 1, 10, 25 etc.
 - *lungi*: se reprezintă pe 64 de biți și se termină în caracterul L sau l - 0L, 1L, 10L.

b. Literalii reali (în virgulă mobilă):

- **dublii**: numere zecimale ce conțin punctul zecimal și pot fi urmate de un exponent prefixat cu e sau E; sunt reprezentați pe 64 de biți;
- **normali**: au aceeași formă ca cei dublii, dar au în plus sufixul f sau F; sunt reprezentanți pe 32 de biți.

c. Literalii booleeni:

- sunt `true` și `false`
-  Literalii întregi nu mai reprezintă valori booleene ca în C

d. Literalii de tip caracter:

- un singur caracter sau o secvență escape între apostrofuri: `'\n'`
- Secvențele escape sunt folosite pentru a înlocui caractere sau acțiuni:

Secvența	Utilizare
<code>\b</code>	Backspace
<code>\t</code>	Tab orizontal
<code>\n</code>	Line feed (linie nouă)
<code>\f</code>	Form feed (pagină nouă)
<code>\r</code>	Carriage return (început de rând)
<code>\"</code>	Ghilimele
<code>\'</code>	Apostrof
<code>\uxxxx</code>	Caracter Unicode

e. Literalii de tip șir de caractere:

- Sunt cuprinși între ghilimele
- Sunt instanțe ale clase `String`, declarată standard în Java.
- Ex: `""`, `"abc"`, `"abc\n"` etc.

4.4. Tipuri primitive

a. Tipuri întregi

- Java oferă patru tipuri de întregi: `byte`, `short`, `int`, `long`
- Sunt definite ca valori cu semn, reprezentate pe 8, 16, 32, respectiv 64 de biți.

b. Tipuri reale

- Există două tipuri reale: `float` și `double`, reprezentate pe 32 și 64 biți.


c. Tipul boolean


- Conține valorile `true` și `false`.


d. Tipul char

- Variabilele de tipul `char` sunt reprezentate pe 16 biți și pot primi ca valoare orice simbol din codul Unicode.

Observații

-  Pentru fiecare tip primitiv, există o clasă corespunzătoare (numită **clasă înfășurătoare** - *wrapper class*), care pune la dispoziție anumite metode utile în cazul conversiilor de date și nu numai.
- Acestea sunt:
 - `int` ↔ `Integer`
 - `double` ↔ `Double`
 - `long` ↔ `Long`
 - `short` ↔ `Short`
 - `byte` ↔ `Byte`
 - `boolean` ↔ `Boolean`
 - `char` ↔ `Character`

 Consultați documentația clasei `Integer`[\[2\]](#) și observați metodele puse la dispoziție. Scrieți o aplicație care să primească de la linia de comandă două valori întregi `a` și `b` și care calculează a^b . Afișați rezultatul în baza de numerație 2.

 Scrieți o aplicație care filtrează dintre toate argumentele primite din linia de comandă numai numerele întregi și afișează:

- numerele prime

- suma tuturor numerelor
- suma radicalilor numerelor; afișați un mesaj corespunzător în cazul în care un argument este negativ. Consultați clasa `Math` din pachetul `java.lang`.
- pentru argumentele care nu sunt convertibile la `Integer`, afișați un mesaj corespunzător.



Pentru a se realiza conversia din șiruri de caractere în numere, se utilizează metodele de tipul `parseTipNumeric`, din clasa înfășurătoare corespunzătoare tipului în care vrem să facem conversia:

```
Integer.parseInt(șir)
Double.parseDouble(șir) etc.
```



Aceste metode pot produce excepții de tipul `NumberFormatException`, în cazul în care argumentul dat nu poate fi convertit la un număr de tipul corespunzător. De aceea, este necesară *tratarea excepțiilor* cu ajutorul instrucțiunii `try - catch - finally`:

```
int number;
try {
    number = Integer.parseInt(args[0]);
} catch (NumberFormatException nfe) {
    System.out.println(number + " is not a number");
}
```

4.5. Declararea variabilelor

- În Java, o variabilă se declară prin tipul ei, urmat de un nume (identificator). Tipul unei variabile poate fi un tip primitiv sau un tip referință (clase, interfețe, vectori)

```
[<modificatori>] <tip> <listă identificatori>;
```

unde <modificatori> poate fi: `public`, `private`, `protected` (modificatori de acces), `final` sau `static`.

- În funcție de locul unde sunt declarate, variabilele se împart în următoarele categorii:
 - *Variabile membre*: declarate în interiorul unei clase, vizibile pentru toate metodele clasei respective și pentru alte clase, în funcție de nivelul lor de acces
 - *Variabile locale*: declarate într-o metodă sau într-un bloc de cod, vizibile doar în metoda / locul respectiv.

- *Parametrii ai metodelor*: vizibili doar în metoda respectivă
- *Parametrii de la tratarea excepțiilor*

4.6. Operatori

- Vezi curs
- Dăm, mai jos, câteva observații legate de unii operatori.



Observații

- **Operatori aritmetici: +, -, *, /, %:**
 - Orice valoare ce depășește limita admisă este redusă modulo acea limită, deci nu există depășiri (overflow și underflow)
 - Operatorul % este definit prin: $(x / y) * y + x \% y == x$.
- **Operatorii de incrementare și decrementare: ++, --**
 - Pot fi aplicați pe operanzilor numerici, atât în formă prefixată, cât și în formă postfixată: ++x, respectiv x++.
 - Diferența dintre cele două forme constă în faptul că incrementarea lui x este realizată înainte, respectiv după folosirea lui x în contextul în care apare.
 - Ex:

```

x = 3;
3 * ++x == 12
3 * x++ == 9
x == 4

```
- Expresiile x++ și ++x sunt echivalente cu $x = x + 1$, cu excepțiile:
 - sunt *atomice* - considerate o unică operație și nu două: o incrementare, urmată de o atribuire. Important atunci când mai multe fire de execuție acționează asupra aceleiași variabile.
 - operandul x este evaluat o singură dată.
- **Operatorii de atribuire: =, +=, -=, *=, /=, %=**
 - Operatorii += etc sunt scrieri prescurtate pentru: $x = x + y$ etc, cu excepția faptului că evaluarea lui x se face o singură dată
- **Operatori booleani: ||, &&, !**

- la evaluare se face *scurtcircuitare*, adică nu se continuă evaluarea unei expresii dacă valoarea ei finală devine evidentă (dacă într-o conjuncție logică primul termen este false, nu se mai evaluează ceilalți termeni)
- **Precedența operatorilor:**
 - Ordinea în care are loc efectuarea prelucrărilor determinate de operatori este (de la prioritate maximă la prioritate minimă):
 - operatorii postfix
 - operatorii unari de incrementare/decrementare, operatorii + și - unari, operatorul de negație !
 - operatorul **new** de creare de obiecte și cel de conversie: (tip) expresie
 - operatorii multiplicativi: * / %
 - operatorii aditivi: + -
 - operatorii relaționali și instanceof
 - operatorii de egalitate: == !=
 - operatorul &
 - operatorul |
 - conjuncția logică &&
 - disjuncția logică ||
 - operatorul condițional (? :)
 - operatorii de atribuire.

4.7. Instrucțiuni

- Vezi curs

5. Instalare IDE

- Pentru Java există mai multe medii de dezvoltare. În funcție de preferințe, instalați [Netbeans](#) sau [Eclipse](#).
- Pentru a crea proiecte Java, consultați tutorialele:
 - [NetBeans IDE Java Quick Start Tutorial](#)
 - Pentru Eclipse, un tutorial cuprinzător este [Eclipse IDE - Tutorial](#)

6. Exerciții



Citiți de la tastatură un număr n , urmat de n numere reale. Afișați suma obținută prin aproximarea fiecărui număr la cel mai mic întreg mai mare decât acesta.



Pentru a citi de la tastatură, folosim clasa `Scanner`, din pachetul `java.util`. Consultați documentația și observați metodele puse la dispoziție.

```
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
```



Deoarece clasa `Scanner` se află în pachetul `java.util`, acesta trebuie importat.

```
import java.util.Scanner;
```



Consultați clasa `String`. Pentru un șir de caractere dat de la tastatură, verificați dacă acesta este palindrom și afișați un mesaj corespunzător.



*Se dau de la tastatură două numere m și n , mai mari decât 3, ce definesc un interval $[m, n]$. Pentru fiecare număr par din interval, verificați conjectura lui Goldbach: "*Orice număr întreg par mai mare decât 3 poate fi scris ca suma a două numere prime*".

6. Recomandări

- Încercați să respectați convențiile de scriere în Java atunci când scrieți cod. Mai multe detalii în următorul [pdf](#).
- Folosiți nume sugestive pentru clase, variabile, constante.
- Încercați să documentați fiecare clasă și fiecare metodă netrivială (JavaDoc).
- Pentru fiecare clasă, încercați să vă împărțiți codul în metode care să facă un *singur* lucru. Acest lucru va fi util atunci când vreți să reutilizați codul sau să faceți refactoring unei metode.
- În general, încercați să scrieți câte o clasă pe fișier.