Programare avansata pe obiecte - laborator 1 (231)

Butan Silvia
silvia.butan@endava.com
butan.silvia@gmail.com

1) **Kit de dezvoltare JDK**
    a) Download open-source build from: https://jdk.java.net/13/
    b) Extract the zip file into a folder: eg. C:\Program Files\Java\jdk-13.0.2
    c) Add the location of the bin folder of the JDK installation to the PATH variable in System Variables
    d) Set JAVA_HOME: Under System Variables, click New -> Enter the variable name as JAVA_HOME -> Enter the variable value as the installation path of the JDK eg. C:\Program Files\Java\jdk-13.0.2
    e) Open up the Command Prompt and type `java -version`

```
openjdk 13.0.2 2020-01-14
OpenJDK Runtime Environment (build 13.0.2+8)
OpenJDK 64-Bit Server VM (build 13.0.2+8, mixed mode, sharing)
```

2) **Java Development Environment - BlueJ (pentru la**
    a) Download and Install BlueJ from: **https://www.bluej.org/**

3) **Git - distributed version control system**
    a) Install git from: https://git-scm.com/download/win

4) **GitHub**
    a) Create a Github account on: https://github.com/join?source=header
    b) Create a new repository ‘pao-labs’ on: https://github.com/new
    c) Clone the repository:
        i) Open **cmd** and run the following command:
            - `git clone https://github.com/<<git-username>>/pao-labs.git`

Programare avansata pe obiecte - laborator 1

Limbajul Java prezinta urmatoarele caracteristici:

- Este un limbaj de programare orientat spre obiecte
- Este un limbaj portabil => independent de platforma
- Programele Java sunt interpretate => este utilizata o masina virtuala Java (JVM) care interpretează un cod compilat al programelor

.java (fisier cod sursa) ---> **javac** (compilator) ---> **.class** (bytecode) ---> **java** (interpretor, emulator de cod bytecode = **JVM**) ---> **OS** (sistem de operare)

- Este un limbaj case sensitive

Exemple:

1. **Metoda principala**

```
/*
 * class = keyword (cuvant cheie), cuvant din vocabularul limbajului java
 *
 * HelloWorld = identificator, trebuie sa respecte următoarele reguli:
 *
 * - Nu poate începe cu cifra, dar poate contine cifre
 * - Nu poate contine spații și operatori (sau caractere speciale: #)
 * - Nu pot fi folosite cuvinte cheie
 *
 * main = metoda principala - entry point al programului
 */

class HelloWorld {

    public static void main(String[] args) {
        System.out.print("Hello World");
    }
}
```

2. **Tipuri primitive**

   Pentru a lucra cu date avem nevoie de zone de memorie, iar pentru a lucra cu zone de memorie în limbajul Java trebuie sa ne gandim la tipul de date stocat: numere, valori de adevărat sau fals, caractere sau obiecte.

   Numerele, valorile de adevărat sau fals și caracterele sunt numite **Tipuri primitive. =>**

   byte, short, int, long, float, double, char, boolean

```
public class Exemplul1 {

    public static void main(String[] args) {
        // tip identificator
        // byte, short, int, long, float, double, char, boolean
```

```java
// String --- nu este primitiv

byte q1; // declarare zona memorie
// byte -> 256 ---> [-128, 127]
q1 = 10;
// pentru a da o valoare zonei de memorie se folosește operatorul "="

short s1 = 10;
int i = 10; // 4 bytes
long a = 10; // 8 bytes

// Diferenta intre byte, short, int si long o reprezinta cantitatea de date

int i2 = 054; // baza 8
int i3 = 0XFF; // baza 16
int i4 = 0xFF; // baza 16
int i5 = 0b10110; // baza 2

long b = 99999999999999L; // L = literal

double d1 = 10.5;

float f1 = 10.5F;
// diferenta dintre double si float este de precizie (nr zecimale după
// virgulă)
float f2 = (float) 10.5;
// În Java orice valoare cu virgula este considerată de tip double =>
// => folosim un literal (f,F) sau operatorul de conversie

int f3 = (int) 10.5;

boolean k1 = false;
boolean k2 = true;
// true, false = cuvinte cheie (keywords)

char w1 = 'a';
// caracterele în Java se pun între ghilimele simple ''
char w2 = '\n';
char w3 = '\u1011';

int r1,r2,r3;
// se pot declara mai multe zone de memorie cu virgula între ele
```

```
        int r4,r5 = 10, r6;
    }
}
```

3. **Operatori de comparare:**

```
public class Exemplul2 {

    public static void main(String[] args) {
        // operatori de comparare: < > <= >= == !=
        // < > <= >= --- folosiţi cu tip de date numeric

        boolean b1 = true == false;
        boolean b2 = 3 >10;

        int x = 10;
        int y = 20;

        boolean b3 = y <= x;

        // operatorii de comparare sunt folosiţi ca şi condiţii în structurile de
        // control
        // structurile de control ne ajuta sa luăm decizii / repetăm instrucţiuni

        if (x > 10) { // condiţia trebuie sa fie boolean

        }
    }
}
```

```
public class Exemplul3 {

    public static void main(String[] args) {
        int x = 10;
        int y = 20;

        if (y / x < 5) {
            System.out.print("A");
        } else {
            System.out.print("B");
```

```
        }
    }
}
```

4. **Utilizarea pachetelor**

   Clasele Java sunt grupate în **pachete**. Declarația de import îi spune compilatorului ce pachet să caute pentru a găsi o clasă.

   Există un pachet special în Java numit `java.lang`. Acest pachet este special prin faptul că este importat automat.

   Până acum, tot codul pe care l-am scris în acest laborator a fost inclus în pachetul implicit, "**default**". Acesta este un pachet special, fără nume. O buna practica este de a numi întotdeauna pachetele pentru a evita conflictele și pentru a permite celorlalți să reutilizeze codul.

```java
import java.util.Random;

public class Exemplul4 {

    public static void main(String[] args) {
        Random r = new Random();
        System.out.print(r.nextInt(10));
    }

}
```

```java
package com.example.lab1;

import java.util.Scanner;

public class Exemplul5 {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); // The Scanner class is used to
        // get user input, and it is found in the java.util package.
        System.out.println("Enter username:");
        String user = scanner.nextLine();
    }
```

5. Obiecte, Wrapper Classes

O **clasa** este un **blueprint** prin care descriem un **obiect**.

Clasele Wrapper oferă o modalitate de a utiliza tipuri de date primitive (int, byte, short, char, double, float, long, boolean) ca obiecte.

Primitive Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

```
package com.example.lab1;

public class Exemplul6 {

    public static void main(String[] args) {
        Integer myInt = 5;
        Double myDouble = 5.99;
        Character myChar = 'A';

        System.out.println(myInt.intValue());
        System.out.println(myDouble.doubleValue());
        System.out.println(myChar.charValue());
    }
}
```

```
public class Exemplul7 {
  public static void main(String[] args) {
    Integer myInt = 100;
```

```
    String myString = myInt.toString();
    System.out.println(myString.length());
  }
}
```

Aplicaţii:

1. **Scrieţi un program Java care afişează numerele impare de la 1 la 99.**

```
Expected Output:
1
3
5
7
9
....
```

2. **Scrieţi un program Java care compară două numere.**

```
Input Data:
Input first integer: 25
Input second integer: 39

Expected Output:
25 != 39
25 < 39
25 <= 39
```

3. **Fiind dat un număr n, scrieţi o metodă care însumează toţi multiplii de trei şi cinci până la n (inclusiv).**
4. **Scrieţi o metodă care calculează factorialul unui număr dat.**
5. **Scrieţi o metodă care verifică dacă un număr este număr prim.**
6. **Scrieţi o metodă care returnează elementul n al secvenţei Fibonacci.**
7. **Scrieţi o metodă care calculează cel mai mare factor prim al unui număr dat.**