Programare orientată pe obiecte

#7 JAVA
Noțiuni introductive despre Java

Adrian Runceanu
www.runceanu.ro/adrian

Curs 7

Limbajul JAVA
(Notiuni introductive)

Java - proprietăţi

- limbaj de programare de nivel înalt
- dezvoltat de firma Sun Microsystems (in prezent proprietate a firmei ORACLE)
- folosit în special datorită portabilităţii sale

Caracteristici

- orientat exclusiv pe obiecte
- portabilitate mare
- sintaxă preluată din limbajul C++
- execuţia propriu-zisă este făcută de o maşină virtuală (**JVM**)
- pune la dispoziţia utilizatorului o bibliotecă de clase foarte puternică şi complexă

Distribuţii Java

1. Java Developement Kit (JDK):
   - compilatorul Java (javac)
   - maşina virtuală Java (Java Virtual Machine – JVM)
   - debugger (jdb)
   - viewer de applet-uri (appletviewer)
   - alte utilitare

2. Java Runtime Environment (JRE):
   - maşina virtuală Java (JVM)
   - viewer-ul de applet-uri (appletviewer)
     - stand-alone sau într-un browser

Diferenţe faţă de limbajul C++

1. structură pur obiectuală
   - nu există funcţii şi variabile globale
   - nu există structuri, enumerări şi uniuni (*struct*, *enum*, *union* în C++)

1. tip de date nativ boolean (*true* şi *false*)

1. tip de date nativ *String* – lucrul cu şiruri de caractere

Diferenţe - continuare

4. mecanisme pentru tratarea structurată a excepţiilor
   – simplificarea structurării programelor
   – permit un control mai bun asupra erorilor apărute în timpul rulării programului
5. obiectele sunt instanţiate **doar** dinamic – folosind operatorul *new*
6. nu există pointeri (nici către obiecte, nici către metode)
   – obiectele se accesează prin referinţele returnate de operatorul ***new***

Diferenţe - continuare

7. nu suportă moştenire multiplă
   – se compensează prin existenţa interfeţelor
8. obiectele alocate dinamic nu trebuie dezalocate explicit
   – dezalocarea se face explicit, de către un mecanism de *garbage-collection*
   – determină automat când un obiect nu mai este folosit (nu mai există referinţe către el)
   – elimină necesitatea folosirii unui destructor
9. mecanisme pentru programare concurentă - *programe cu mai multe fire de execuţie care ruleaza în paralel*

Noţiunile de “clasă” şi “obiect”

- **clasa** –tip de date nou, creat de programator
  - caracteristicile clasei sunt descrise prin folosirea altor tipuri de date
  - permite folosirea şi de metode, pe lânga date
  - *membrii pot avea mai multe atribute de accesibilitate*

- **obiect** – instanţiere a unei clase
  - *se pot instanţia doar dinamic (folosind operatorul new)*
  - fiecare obiect are o zonă de memorie proprie

Compilarea, executarea

- **compilarea** unui program Java:
  - se realizează cu comanda:

    javac nume_fişier.java
  - fişierul **trebuie** să aibă acelaşi nume cu clasa principală (cea care conţine funcţia **main**)
  - rezultă unul sau mai multe fişiere cu extensia **.class**

- **rularea** unui program Java:
  - se realizează cu comanda:

    java nume_fişier
  - fişierul executat trebuie să fie cel care conţine funcţia **main**

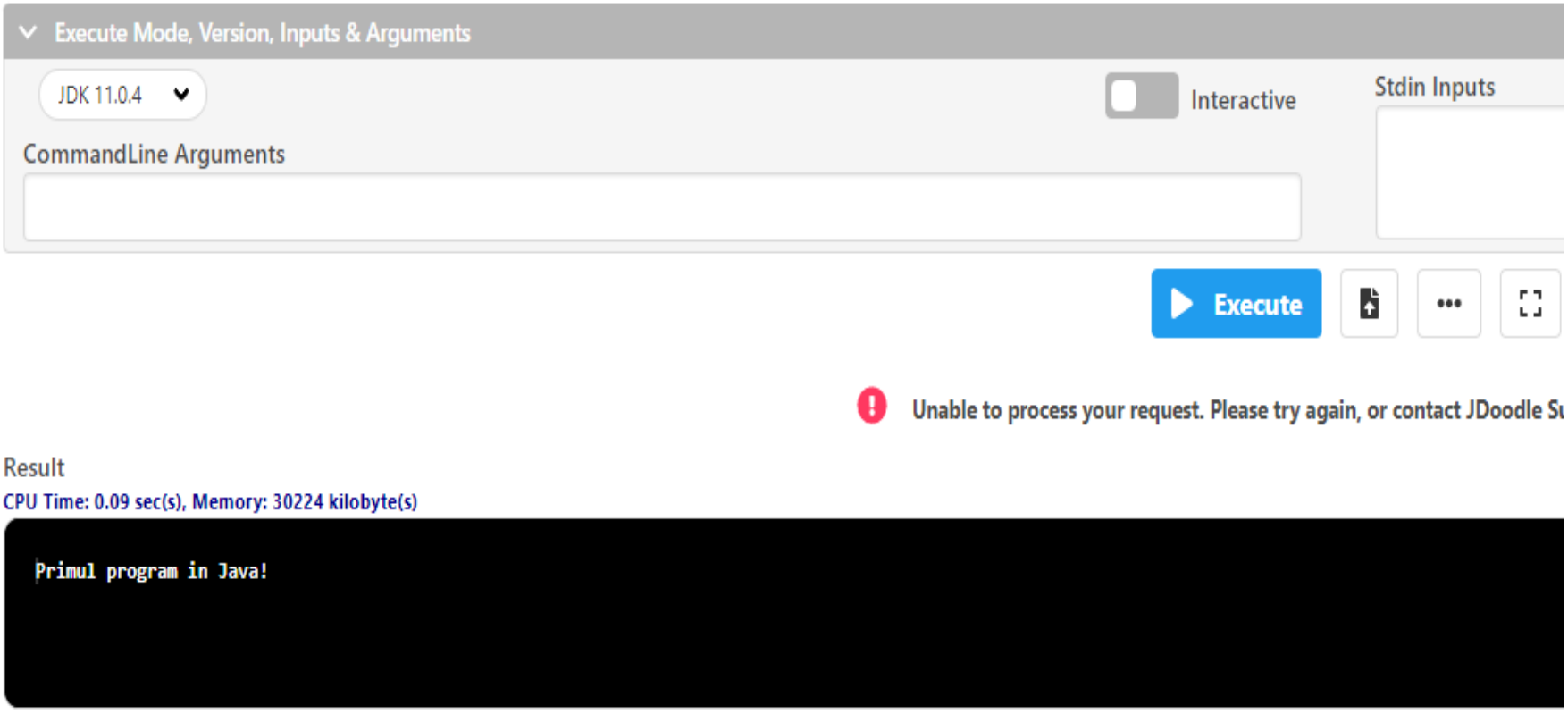
Exemplu de program Java

```
public class PrimulProgram
{
    public static void main (String[ ] args)
    {
        System.out.println ("Primul program in Java!");
    }
}
```

execuţia programului începe de la funcţia **main**

- trebuie să fie **public**
- **static** -> nu există un obiect instanţiat din clasa respectivă
- tipul returnat: **void**
- se poate lansa în execuţie folosind parametrii (**args**)

```java
public class PrimulProgram
{
    public static void main (String[ ] args)
    {
        System.out.println ("Primul program in Java!");
    }
}
```



Exemplu de program – instantierea unui obiect

```
class Student
{
    private String nume;
    private int varsta;

    public Student (String nume, int varsta)
    {
        this.nume = nume;
        this.varsta = varsta;
    }

public void afisare ()
    {
        System.out.println ("Dl. "+ nume + " are " + varsta + " ani. ");
    }
}
```

Exemplu de program – instantierea unui obiect (continuare)

```
public class ExempluStudent
{
    public static void main (String[ ] args)
    {
        Student st = new Student ("Mihai", 25);
        st.afisare();
    }
}
```

- folosirea membrilor private şi public
- folosirea pointer-ului *this*
- alocare dinamică de memorie pentru obiectul instanţiat
- apelarea implicită a constructorului
- apelarea funcţiei afisare
- apariţia *garbage-collector*-ului

```java
class Student
{
    private String nume;
    private int varsta;

    public Student (String nume, int varsta)
    {
        this.nume = nume;
        this.varsta = varsta;
    }

public void afisare ()
    {
        System.out.println ("Dl. "+ nume + " are " + varsta + " ani. ");
    }
}
public class ExempluStudent
{
    public static void main (String[ ] args)
    {
        Student st = new Student ("Mihai", 25);
        st.afisare();
    }
}
```

Execute Mode, Version, Inputs & Arguments

JDK 11.0.4

Interactive

Stdin Inputs

CommandLine Arguments

Execute

Unable to process your request. Please try again, or contact JDoodle Support

Result

CPU Time: 0.15 sec(s), Memory: 35600 kilobyte(s)

```
Dl. Mihai are 25 ani.
```

Exemplu de program – introducerea de la tastatură

folosirea clasei System;

```
public class IntroducereStudent
{
    public static void main (String[ ] args) throws IOException
    {
            System.out.print ("Introduceti numele studentului: ");
            BufferedReader b1 = new BufferedReader (new InputStreamReader (System.in));
            String nume = b1.readLine ();
            System.out.print ("Introduceti varsta studentului: ");
            BufferedReader b2 = new BufferedReader (new InputStreamReader (System.in));
            String varsta_str = b2.readLine ();
            int varsta = Integer.parseInt(varsta_str);
            Student st = new Student (nume, varsta);
            st.afisare ();
    }
}
```

Limbajul JAVA

1. **Tipuri de programe implementate de Java**
2. **Etapele dezvoltării unei aplicaţii Java**
3. **Structura unei aplicaţii Java**
4. **Metode**
5. **Elemente de bază ale limbajului Java**

Tipuri de programe implementate de Java

Cu ajutorul limbajului Java se pot dezvolta doua tipuri de programe:

1. *Programe Java care se executa individual prin intermediul unui interpretor Java.*
   Acestea se incadreaza in programele “clasice” scrise in diverse limbaje de programare, cum ar fi: C/C++, Pascal, etc.
   Acest tip de programe Java sunt denumite **aplicatii**.
2. *Programe Java care se executa in interiorul unui navigator Internet(browser), dintr-un document HTML.*
   Acest tip de programe Java sunt denumite **applet-uri**.

Limbajul Java

1. **Tipuri de programe implementate de Java**
2. **Etapele dezvoltării unei aplicaţii Java**
3. **Structura unei aplicaţii Java**
4. **Metode**
5. **Elemente de bază ale limbajului Java**

Etapele dezvoltarii unei aplicatii Java

1. **Editarea setului de instructiuni de programare** cu ajutorul unui editor de texte.

In acest fel este creat un fisier sursa, care are extensia ***.java***.

2. **Compilarea programului**

Pentru aceasta operatie se lanseaza in executie un program special, denumit **compilator Java**.

Compilatorul analizeaza codul sursa al programului din punct de vedere sintactic, semnaland eventualele erori.

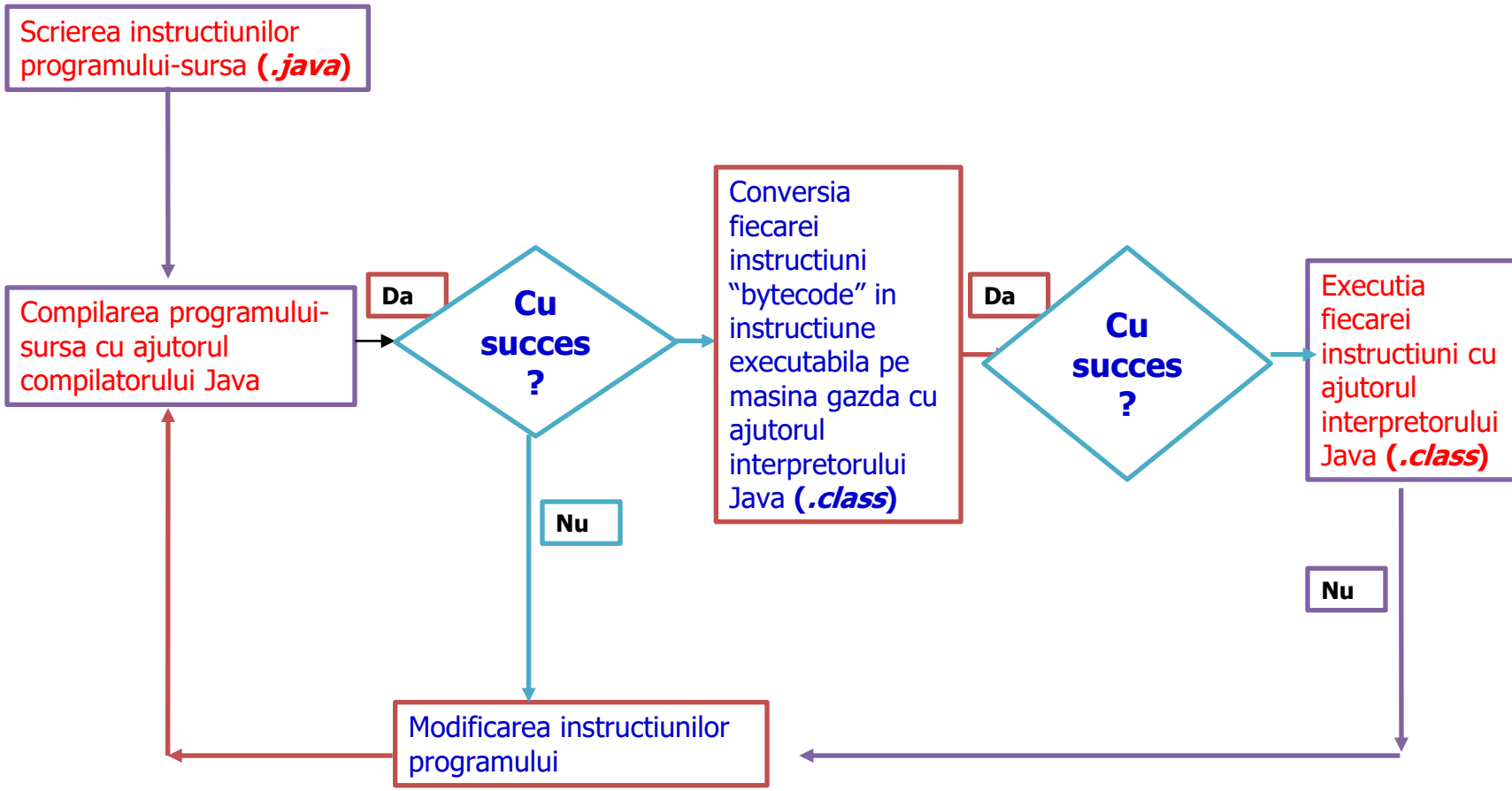
- Daca programul nu contine erori sintactice, compilatorul traduce acest program in codul masina pentru **masina virtuala Java** (un fel de calculator intr-un alt calculator).
- Rezultatul compilarii este unul sau mai multe **fisiere de tip "bytecode"**- *o secventa de instructiuni de asamblare pentru masina virtuala Java*.
- *Fisierul "bytecode" nu depinde de masina gazda si nici de sistemul de operare pe care va fi executat programul.*
- Masina virtuala mai este cunoscuta si ca **interpretor Java sau executor (runtime) Java**.
- Extensia fisierului "bytecode" rezultat in urma compilarii are extensia *.class*.

3. **Conversia (transformarea), de catre interpretorul Java, a instructiunilor "bytecode" in instructiuni inteligibile masinii gazda care apoi sunt executate**.

- *Conversia are loc la lansarea executiei* si anume instructiune cu instructiune.

- In acest mod este asigurata portabilitatea si independenta de platforma.

- Un *dezavantaj* poate fi *considerat timpul mai mare de executie*.



Procesul de compilare si executie a unei aplicatii Java

Limbajul Java

1. **Tipuri de programe implementate de Java**
2. **Etapele dezvoltării unei aplicaţii Java**
3. **Structura unei aplicaţii Java**
4. **Metode**
5. **Elemente de bază ale limbajului Java**

Structura unei aplicatii Java

- *O aplicatie Java este compusa din una sau mai multe **clase** care interactioneaza intre ele prin intermediul metodelor.*

- In grupul de clase, care formeaza o aplicatie Java, trebuie sa existe o clasa care sa contina o *metoda statica* avand numele **main**.

- Atunci cand se executa o aplicatie Java, masina virtuala va cauta si invoca automat metoda statica avand numele **main**.

Nota:

- **Metoda main** poate fi considerata ca fiind echivalentul Java a **functiei main** din C/C++.
- Cel mai simplu program Java (care nu face nimic) arata astfel:

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

Nota:

✔ Tipul parametrilor metodei (functiei) **main** cat si tipul metodei, *static void*, sunt obligatorii.

Limbajul Java

1. **Tipuri de programe implementate de Java**
2. **Etapele dezvoltării unei aplicaţii Java**
3. **Structura unei aplicaţii Java**
4. **Metode**
5. **Elemente de bază ale limbajului Java**

Metode

- **Metodele** folosite in programele Java reprezinta in mare masura echivalentul functiilor sau procedurilor din alte limbaje de programare.
- O definitie completa a conceptului de metoda va fi data intr-un curs separat cand se va discuta despre conceptele de clasa si obiect.
- O metoda se defineste prin:
    1. antetul metodei
    2. corpul metodei

1. Antetul metodei contine:

- numele metodei
- tipul valorii returnate de metoda
- si o lista de parametri (eventual vida), fiecare avand un tip precizat, prin care metoda comunica cu exteriorul ei, incadrata intre paranteze rotunde.

2. Corpul metodei este constituit din declaratii si instructiuni care executa diferite sarcini.

- Toate declaratiile si instructiunile din corpul metodei sunt incadrate intre paranteze acolade.

Sintaxa definirii unei metode este:

```
<tip_rezultat> nume_metoda
(<lista_parametri>)
{
    <corpul_metodei>
}
```

Nota:

- Prin prefixarea metodelor cu ajutorul cuvintelor cheie *public static* se poate spune ca metodele sunt aproape similare cu functiile din limbajul C++.
- Aceasta tehnica nu trebuie utilizata insa in mod abuziv.

- Instructiunea return este utilizata pentru a intoarce o valoare catre codul apelant.

Sintaxa instructiunii return este:

```
return
[<expresie>];
```

unde:

- <expresie> - specifica valoarea returnata de metoda; in anumite situatii pentru iesirea fortata dintr-o metoda care are tipul *void* se foloseste *instructiunea return fara <expresie>*.

- Daca tipul metodei este *void* atunci nu se returneaza nici o valoare si deci instructiunea return nu este necesara.

La apelul unei metode parametrii actuali (reali) sunt trecuti in parametrii formali utilizand exclusiv **transmiterea prin valoare**.

Nota:

- *Java permite supraincarcarea (overloading) numelui metodelor.*
- *Aceasta inseamna ca mai multe metode cu acelasi nume pot fi declarate in cadrul aceleiasi clase atata timp cat lista de parametri formali difera ca numar si tip.*

Utilizarea claselor de obiecte din pachetele predefinite (pachete API) Java

- Pot exista unele operatii care sunt frecvent utilizate in cadrul unei aplicatii Java:
  - citirea datelor
  - scrierea datelor
  - operatii matematice (extragerea radicalului, calculul modulului, signatura unui numar, etc.)
  - desenarea de obiecte grafice, etc

  pentru care nu exista instructiuni specifice ale limbajului Java.

- Mediul de programare Java 2 SDK ofera peste 70 de pachete predefinite, printre care, mai des utilizate sunt:

  a) **java.lang** care ofera clase fundamentale pentru limbajul Java:
  - clasa Integer
  - clasa Math
  - clasa System
  - clasa String, etc;

  b) **java.io** care ofera modalitati de citire/scriere a datelor prin intermediul fluxurilor de date, a fisierelor, etc

  c) **java.util** care contine clase pentru utilizarea colectiilor de date abstracte de tip stiva si coada, pentru manevrarea datelor calendaristice si a timpului

d) **java.util.jar** pentru citirea/scrierea fisierelor in format *jar* (Java Archive)

e) **java.math** care ofera clase specializate in calcul matematic

f) **java.text** care ofera clase pentru manevrarea textului, a datelor calendaristice, a timpului si a mesajelor intr-o maniera independenta de limba utilizata

g) **java.net** care pune la dispozitie clase pentru implementarea aplicatiilor de retea

Directiva *import* pentru utilizarea claselor din pachetele API Java in propria aplicatie:

- Sa presupunem ca se doreste folosirea unei **clase oarecare C** dintr-un **pachet P**.

- Atunci referirea la acea clasa in cadrul programului se face cu numele **P.C**, adica plasand numele pachetului urmat de semnul punct (**.**) inaintea numelui clasei.

- Utilizarea permanenta a numelui pachetului din care fac parte clasele poate crea dificultati de scriere a programului.

- Pentru a evita acest lucru se foloseste directiva **import**.

Sintaxa directivei **import** este:

```
import <nume_pachet>.*<nume_clasa>;
```

sau

```
import <nume_pachet>.*
```

Pentru a doua forma a directivei, toate clasele din pachet vor putea fi abreviate cu numele lor neprecedat de numele pachetului.

Directiva import

- *Directivele **import** trebuie sa apara inainte de orice declarare a unei clase.*
- Pachetul ***java.lang*** este automat inclus in intregime.
- Acesta este motivul pentru care putem folosi prescurtari de genul **Integer.parseInt ()**

Crearea de pachete cu clase de obiecte proprii aplicatiei. Instructiunea *package*

- Daca dorim sa includem o clasa de obiecte intr-un pachet trebuie sa realizam urmatoarele:
  - sa scriem o instructiune **package** inaintea unei eventuale directive **import**;
  - apoi, sa stocam fisierul cu extensia *.class* ce contine clasa respectiva in directorul care are numele pachetului.

Sintaxa instructiunii **package** este:

```
package
<nume_pachet>;
```

Limbajul Java

1. **Tipuri de programe implementate de Java**
2. **Etapele dezvoltării unei aplicaţii Java**
3. **Structura unei aplicaţii Java**
4. **Metode**
5. **Elemente de bază ale limbajului Java**

Elemente de baza ale limbajului Java

1. Setul de caractere folosit de limbajul Java

- Limbajul **Java** lucreaza in mod nativ folosind setul de caractere Unicode.
- Acesta este un standard international care inlocuieste vechiul set de caractere ASCII.
- Vechiul standard ASCII este insa un subset al setului Unicode, ceea ce inseamna ca vom regasi caracterele ASCII cu exact aceleasi coduri ca si mai inainte.
- **Java** foloseste setul Unicode in timpul executiei aplicatiilor dar si in timpul compilarii acestora.

- La citirea fisierului sursa, compilatorul **Java** foloseste ***secventele escape Unicode***.

- Secventele escape sunt *secvente de caractere ASCII care incep cu caracterul backslash (****).*

- Pentru *secventele escape Unicode*, al doilea caracter din secventa trebuie sa fie caracterul *u* sau *U*.

- Dupa caracterul *u* sau *U* urmeaza o combinatie de patru cifre hexazecimale care formeaza impreuna doi octeti de memorie reprezentand un caracter Unicode.

- De exemplu, cifrele de la 0 la 9 sunt reprezentate prin secvente escape Unicode de la \u0030 la \u0039 si sunt interpretate ca cifre ISO-LATIN-1.

- Spatiile albe folosite in programele **Java** sunt:
  - caracterul blanc (spatiu)
  - tab
  - return (retur de linie)
  - line-feed (linie noua)
  - return + line-feed

Anumite caractere sunt reprezentate prin secvente escape speciale:

- \n = \u000a - linie noua
- \b = \u0008 - backspace
- \t = \u0009 - tab
- \r = \u000d - return
- \\ = \u005c - backslash
- \” = \u0022 - ghilimele
- \’ = \u0027 - apostrof

2. Identificatori

✔ Identificatorii, intalniti si sub denumirea de nume simbolice, au rolul de a denumi elemente ale programului **Java**:

- constante
- variabile
- clase
- metode, etc.

✔ Din punct de vedere sintactic, *un identificator este constituit dintr-o succesiune nelimitata de litere si cifre Unicode*, primul caracter fiind obligatoriu o litera (inclusiv '_').

- Limbajul Java este “case-sensitive”, adica *face diferenta intre literele mici si literele mari*.

- De aceea, identificatorii sunt diferiti in functie daca sunt scrisi cu litere mici si mari.

- *Cuvintele-cheie (keywords) sunt identificatori speciali, cu inteles predefinit, care pot fi utilizati numai in constructii sintactice in care sunt specificati.*
- De exemplu: if, while, etc.

- Toate cuvintele-cheie se scriu numai cu litere mici.

3. Separatori

Separatorii au rolul de a separa unitatile sintactice:

- Ca separatori “generali” se utilizeaza caracterele albe: spatiu (‘ ‘), TAB (‘\t’), sfarsit de linie (‘\n’) si comentariile.

- *Separatorii specifici* sunt folositi, de exemplu, pentru a separa unele constructii sintactice:
  - variabilele sunt separate prin caracterul virgula (‘,’)
  - Alti *separatori specifici* sunt **( ) { } [ ]**

- Delimitatorii sunt folositi, de exemplu, pentru:

  - a delimita sfarsitul unei instructiuni sau al unei declaratii - caracterul punct si virgula (‘;’);

  - a delimita o constanta de tip caracter - caracterul apostrof (‘);

  - a delimita constantele sir de caractere(ghilimelele).

4. Comentarii

✔ *Comentariile sunt texte care vor fi ignorate de compilator, dar au rolul de a explicita si de a face mai lizibil pentru programator anumite secvente de program.*

✔ In Java exista trei tipuri de comentarii:

1. o succesiune de caractere incadrata intre **/*** si ***/** ; aceste comentarii pot fi formate din mai multe linii;
2. o succesiune de caractere pe mai multe linii care tin de documentatie, incadrate intre **/**** si ***/**; textul dintre cele doua secvente este automat mutat in documentatia aplicatiei de catre *generatorul automat de documentatie (javadoc)*;
3. o succesiune de caractere care incepe cu **//** si se termina la sfarsitul unei singure linii.

5. Variabile

- **Variabila** este *o zona temporara de stocare, rezidenta in memoria RAM, care are un nume simbolic (identificator) si stocheaza un anumit tip de date*.
- Ea poate fi modificata pe parcursul executiei programului.
- In ciuda denumirii, in **Java** exista variabile care isi pot modifica valoarea si variabile care nu si-o pot modifica, numite **variabile finale**.
- Asupra variabilelor finale se va reveni ulterior dupa intelegerea conceptelor de clasa si de obiecte.

5. Variabile

- Pentru utilizarea unei variabile intr-un program **Java** trebuie ca aceasta sa fie declarata.

- La *declararea variabilei* trebuie specificat:
  - numele simbolic al variabilei
  - tipul acesteia
  - si, eventual, o valoare initiala care se atribuie variabilei.

Tipurile primitive de date definite in Java

- Un tip de date defineste:
  - multimea de valori pe care variabila poate sa le stocheze
  - modul de reprezentare a acestora in memorie
  - si setul de operatii pe care programul poate sa le realizeze cu aceste date.

- In limbajul **Java** a fost definita exact modalitatea de reprezentare a tipurilor primitive de date in memorie.

- In acest fel, **variabilele Java devin independente de platforma hardware si software pe care lucreaza**.

Tipuri de date predefinite(de baza)

- De asemenea, **Java** defineste o valoare implicita pentru fiecare tip de date, in cazul in care variabila de un anumit tip nu a primit nici o valoare de la utilizator.

- Este o practica buna, insa se recomanda ca programele sa nu depinda niciodata de aceste initializari implicite.

- Regula ar fi deci, urmatoarea: *nici o declaratie de variabile, fara initializare.*

Limbajul **Java** accepta urmatoarele 9 tipuri de baza:

	Tipul de date Java
1.	byte
2.	short
3.	int
4.	long
5.	float
6.	double
7.	char
8.	boolean
9.	void

Tabelul de mai jos prezinta o descrie generala a tipurilor primitive de date utilizate de **Java**.

Tip	Valori	Reprezentare in memorie
byte	[-128, 127]	Intreg pe 1 byte
short	[-32768, 32767]	Intreg pe 2 bytes
int	[2.147.483.648, 2.147.483.648]	Intreg pe 4 bytes
long	$[-2^{63}, 2^{63}-1]$	Intreg pe 8 bytes
float	6 cifre semnificative $[10^{-46}, 10^{38}]$	Virgula mobila pe 4 bytes
double	15 cifre semnificative $[10^{-324}, 10^{308}]$	Virgula mobila pe 8 bytes
char	coduri Unicode	Pe 2 bytes
boolean	false sau true	Pe un bit

1. **Tipul boolean** este folosit pentru memorarea unei valori de adevar sau fals.

- In Java aceste doua valori le vom nota prin constantele (literali) **true** si respectiv **false**.
- *Orice variabila booleana nou creata primeste automat valoarea implicita **false***.

2. **Tipul char** este folosit pentru a reprezenta caractere de tip Unicode.

- *Orice variabila de tip caracter nou creata primeste automat ca valoare implicita caracterul **null** al codului Unicode, "**\u0000**"*.

3. **Tipurile de date intregi** sunt folosite pentru memorarea unor valori intregi cu semn.

✔Conventia folosita de **Java** pentru valorile intregi cu semn este reprezentarea in complement fata de doi.

✔Tipurile de date intregi sunt:

1. ***byte***
2. ***short***
3. ***int***
4. ***long***

✔*Orice variabila de tip intreg (byte, short, int si long) nou creata primeste automat valoarea implicita 0.*

4. **Tipurile de date reale** sunt folosite pentru memorarea unor valori reale sub forma de mantisa si caracteristica.

✔Tipurile de date reale sunt:

1. ***float***
2. ***double***

✔Valoarea implicita pentru variabilele de tip *float* este 0.0f, iar pentru variabilele de tip *double* este 0.0d.

In Java sunt definite cateva **valori reale speciale**:

1. Valoarea NaN (Not a Number) se obtine atunci cand se efectueaza o operatie a carei rezultat nu este definit, de exemplu 0.0 / 0.0.
2. Valori folosite pe post de infinit pozitiv si negativ.

Aceste valori pot rezulta in urma unor calcule.

- Valorile descrise la pct. 1. si 2. *sunt definite sub forma de constante* si in ierarhia de clase predefinite Java, si anume in clasa *java.lang.Float* si respectiv in clasa *java.lang.Double*.
- Numele constantelor este:
  - NaN
  - POSITIVE_INFINITY
  - NEGATIVE_INFINITY

Pentru tipurile intregi si intregi lungi, precum si pentru tipurile flotante exista definite clase in ierarhia de clase predefinite **Java** si anume:

- *java.lang.Integer* - pentru tipul *int*
- *java.lang.Long* - pentru tipul *long*
- *java.lang.Float* - pentru tipul *float*
- *java.lang.Double* - pentru tipul *double*

✔In fiecare dintre clase numerice prezentate sunt definite doua constante care reprezinta valorile minime si maxime pentru tipurile respective.

✔Aceste doua constante se numesc in mod uniform **MIN_VALUE** si **MAX_VALUE**.

5. Tipul **void**

- ✔ Tipul **void** este un tip special, pentru care multimea valorilor este vida.

- ✔ Acest tip se utilizeaza cand este necesar sa se specifice absenta oricarei valori.

- ✔ De exemplu: pentru tipul de data a metodelor care nu intorc nici un rezultat, cum a fost cazul metodei *main ().*

Constante

- ✔ *O **constanta** este folosita pentru a exprima in program o valoare pe care o poate lua tipurile primitive de date si tipul sir de caractere.*

- ✔ **Constantele intregi** pot fi reprezentate in bazele 10, 16 sau 8.
- ✔ Constantele intregi pot fi intregi normale sau lungi.
- ✔ Constantele lungi se recunosc prin faptul ca se termina cu sufixul *l* sau *L*.
- ✔ Pentru a reprezenta o constanta intreaga in baza 16 trebuie sa se adauge prefixul 0x sau 0X in fata numarului.
- ✔ Pentru a reprezenta o constanta intreaga in baza 8 trebuie sa se adauge prefixul 0 (cifra zero) in fata numarului.

- ✔ **Constantele reale** care se pot reprezenta in memoria calculatorului sunt numere rationale din intervalele specificate la tipurile *float* si *double*.

- ✔ Constantele reale pot fi specificate in notatia obisnuita sau in format stiintific.

- ✔ Sufixul care indica tipul *float* poate fi **f** sau **F** iar sufixul care indica tipul *double* poate fi **d** sau **D**.

- ✔ Daca nu este specificat nici un sufix, valoarea implicita este de tip *double*.

- ✔**Constantele de tip caracter** sunt utilizate pentru a reprezenta caracterele Unicode.
- ✔Reprezentarea se face fie folosind o litera sau o cifra, fie o secventa *escape*.
- ✔Caracterele care au reprezentare grafica pot fi scrise intre apostrofuri.
- ✔Pentru cele care nu au reprezentare grafica, se folosesc secventele *escape* sau *secventele escape predefinite* prezentate deja.
- ✔Intern, Java interpreteaza constantele de tip caracter ca pe un numar (codul Unicode al caracterului respectiv).
- ✔Ulterior, functiile de scriere vor transforma acest numar in caracterul corespunzator.

- ✔**Constantele de tip sir de caractere** sunt cuprinse intre ghilimele.
- ✔Caracterele care formeaza sirul de caractere pot fi caractere grafice sau secvente escape ca cele prezentate la constantele de tip caracter.
- ✔Daca se doreste introducerea de caractere terminatoare de linie intr-un sir de caractere, atunci se foloseste *secventa escape* '\n' in sirul de caractere respectiv.

Observatie:

Un *sir de caractere este*, de fapt, *o instanta a clasei de obiecte* **String** declarata standard in pachetul *java.lang*.

Sintaxa folosita pentru declararea de variabile este:

<tip> <nume_v1> [= <expresie>] [, <nume_v2> [= <expresie2>] ...];

unde:

- <tip> - specifica tipul de data al variabilelor;
- <nume_v1>, <nume_v2>, ... - specifica numele simbolic al variabilelor care se declara (adica, identificatorii);
- <expresie1>, <expresie2>, ... - specifica o expresie de initializare; expresia trebuie sa fie evaluabila in momentul declararii; sunt optionale si sunt folosite pentru a atribui unor variabile anumite valori initiale.

Nota:

Se pot declara simultan mai multe variabile de acelasi tip, separand numele lor prin virgula.

- ✔ *O variabila trebuie sa fie declarata imediat inainte de a fi folosita*.
- ✔ Locul unde este declarata o variabila determina domeniul de vizibilitate si semnificatia ei.
- ✔ Limbajul **Java** permite si definirea de constante.
- ✔ Modul cum se face definirea constantelor va fi prezentata intr-un curs separat destinat descrierii atributelor statice.

Exemple de declaratii de variabile ce pot fi folosite intr-un program:

```
int a, b=7, c=8;
char g;
float x=b*5.6, y;
```

Întrebări?