C12: Java - Introducere

Cuprins:

- Introducere - descriere generala
- Tipuri de date, cuvinte cheie, clase, obiecte
- Structura unei aplicatii
- Cateva exemple

Bibliografie:

- Thinking in Java – 3rd Edition – Bruce Eckel
http://sina.sharif.edu/~abolhassani/courses/ap/tij3.pdf

- Un tutorial online:
http://www.tutorialspoint.com/java/index.htm

- Documentatie Oracle:
http://www.oracle.com/technetwork/java/index.html

C12: Java - Introducere

Pentru ce / de ce a aparut?

- Java a aparut in anii ‘90 ca urmare a unui proiect de cercetare care urmarea dezvoltarea de software avansat pentru o mare varietate de sisteme embedded si dispozitive de retea (**mediu de tip retele eterogene, distribuite**)
- De maxima importanta in aceste conditii este realizarea de aplicatii care consuma **minimul de resurse**, pot **functiona pe orice platforma**, pot sa **fie extinse dinamic**.
- **Scopul** a fost realizarea unei **platforme de operare in real-time, mica, robusta, portabila si distribuita**.

La ce s-a ajuns?

Ca urmare a acestui demers, a fost creata **platforma Java:** care s-a demonstrat ideala pentru **dezvoltarea de aplicatii sigure**, **distribuite**, **pentru retele in medii variind de la dispozitive de retea si embedded la aplicatii web, apps si aplicatii desktop**.

C12: Java - Introducere

Platforme de lucru Java

JSE(Standard Edition) –JDK (Java Development Kit) , JRE(Java Runtime Environment)
JDK - este platforma standard de lucru ce ofera suport pentru crearea de aplicatii.

J2ME(Micro Edition)
Platforma de lucru J2ME ofera suportul necesar scrierii de programe dedicate dispozitivelor mobile.

J2EE(Enterprise Edition)
Aceasta platforma ofera API-ul necesar dezvoltarii de aplicatii complexe, formate din componente ce trebuie sa ruleze in sisteme eterogene, cu informatiile memorate in baze de date distribuite, etc. Tot aici gasim si suportul necesar pentru crearea de aplicatii si servicii Web, bazate pe componente cum ar fi servleturi, pagini JSP, etc

Free download: http://www.oracle.com/technetwork/java/javase/downloads/index.html

1996 - JDK 1.0 (Sun) -> 2011 - Java SE 7 (Oracle) -> Java SE 17

C12: Java - Introducere

Caracteristicile principale ale limbajului Java :

- de nivel inalt (mai apropiat de limbajul uman decat C++) ->simplu
- familiar (foarte asemanator cu C/C++)
- puternic orientat obiect
- interpretat + independent de arhitectura -> portabil
- de inalta performanta
- distribuit
- robust
- sigur
- multithreaded
- dinamic.

➢ De nivel inalt

A fost dezvoltat avand ca model (printre altele) C++. Trecerea de la scrierea de cod in C/C++ in Java e relativ simpla.

Instructiunile de baza: if, while, for, case, etc; **tipurile primitive** (de baza), **operatorii aritmetici si pe biti etc** au fost preluati cu modificari nesemnificative.

Pune la dispozitie o multime de clase organizate in **pachete** specializate pentru operatii cu text (java.text), de intrare-iesire (java.io), pe baze de date, cu intefata grafica (java.awt) , in retea, etc. (Pachetele sunt organizate, la randul lor, in module incepand cu Java 9)
Aceste clase si metodele lor sunt prezentate si explicitate in: **API** (Application Programming Interface)

https://docs.oracle.com/javase/9/docs/api/index.html?overview-summary.html
https://docs.oracle.com/javase/7/docs/api/

Clasele si metodele au denumiri intuitive:

```
String sir; //declar o variabila de tip String cu numele sir

// instantiez un sir de caractere care contine textul “ceva”
sir = new String(“ceva");

char c=sir.charAt(2);   //v
```

➢ Simplu

Limbajul Java elimina multe dintre “bataile de cap” din C si C++.

- **Nu exista fisiere header si nici directive catre preprocesor** (de fapt nu exista preprocesor).

Ca atare, nu avem #include si nici #define.

- Conceptele de **structura (struct) si uniune (union) au fost eliminate – avem doar clase**.

Structura de baza in Java este clasa.

- **Limbajul Java nu permite lucrul cu pointeri, operatori de luare a adresei sau de indirectare**.

Nu putem face singuri managementul memoriei (dar, toate obiectele de tipuri neprimitive instantiate sunt, de fapt, alocate dinamic – vom vedea cand discutam tipuri de variabile).

- **Nu mai este posibila eliberarea spatiului de memorie alocat dinamic**.

De acest lucru se ocupa o componenta interna a mediului Java, numita colector de spatiu disponibil (**Garbage Collector**).

- **O functie nu poate sa aiba parametri impliciti.**

- In ceea ce priveste programarea obiectuala**, nu exista notiunile de supradefinirea operatorilor, functii si clase friend, destructori, mostenire multipla**; iar toate **metodele nestatice sunt tratate implicit ca fiind virtuale**. Aceste eliminari simplifica mult programarea cu clase.

➢ Puternic orientat obiect

- **Limbajul Java este pur obiectual, adica nu putem scrie un program fara clase** (ca in C++). Nu exista functii sau variabile definite in afara claselor (nici macar functia main).

- **Clasele sunt ierarhizate**, existand o clasa radacina/parinte (numita **Object**), considerata clasa de baza pentru toate clasele implementate deja si cele implementate de utilizator. (Implicit, la crearea unei clase noi, aceasta e derivata din Object chiar daca nu se precizeaza in mod explicit).

- Exista un **set puternic de clase predefinite**, organizate in pachete (**package**) -> module.

- **In afara tipurilor primitive: numerice (int, double, short, long, float, byte), char si boolean, toate tipurile de date sunt implementate ca o clasa**. Pana si tipurile primitive au si o implementare sub forma de clasa: int ->clasa Integer, etc.

- **Nu exista posibilitatea de implementare de functii friend** (acestea stricand incapsularea datelor).

➢ Interpretat

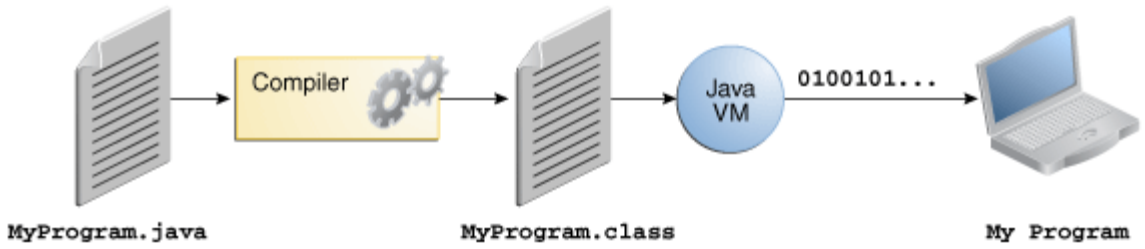
• Java este un limbaj compilat si interpretat

❖ **Compilatorul Java** (**javac**) nu produce cod-masina pentru un anumit procesor, ci un cod intermediar, numit **byte-code**, universal (nu pentru o anume masina).

❖ Pentru executie, avem nevoie de un **interpretor Java** (numit **java**) care traduce codul intermediar intr-un **cod-masina specific sistemului pe care se ruleaza aplicatia**.

Traducerea se face in momentul executiei codului: instructiune cu instructiune, si acest proces se numeste **interpretare**.

Daca o secventa de cod se executa de mai multe ori (de exemplu intr-o bucla for), codul respectiv va fi interpretat de fiecare data => **Java este mai lent decat C++** (acesta e doar un motiv)**.**



(sursa: http://docs.oracle.com/javase/tutorial/index.html)

interpretor + sistem run-time = masina virtuala Java (Java Virtual Machine).

Program Source Code	CPU secs	Elapsed secs	Memory KB	Code B	≈ CPU Load
fannkuch-redux					
Java 7	68.14	**17.25**	18,344	1282	99% 99% 100% 98% †
C++ g++	22.69	22.70	372	1150	0% 2% 100% 0% †
fasta					
Java 7	5.34	5.27	19,144	1507	66% 34% 1% 1%
C++ g++	4.19	4.20	1,080	1286	0% 0% 1% 100%
pidigits					
Java 7	3.11	2.97	22,096	938	5% 98% 1% 2%
C++ g++	2.30	2.30	1,660	682	1% 3% 100% 0%
fasta-redux					
Java 7	2.56	2.52	18,500	1443	3% 2% 0% 98%
C++ g++	1.71	1.71	836	1474	1% 0% 0% 100%
spectral-norm					
Java 7	17.66	4.50	18,652	950	98% 98% 98% 98%
C++ g++	10.06	2.53	500	1044	100% 100% 100% 100%
mandelbrot					
Java 7	27.77	7.05	72,064	796	98% 99% 98% 99%
C++ g++	15.50	3.90	32,292	840	99% 99% 99% 100%
reverse-complement					
Java 7	2.85	1.38	294,652	1661	85% 44% 37% 43%
C++ g++	1.04	0.68	227,944	2275	75% 25% 12% 47%
binary-trees					
Java 7	24.64	9.79	517,904	1007	79% 66% 60% 48%
C++ g++	15.40	4.77	132,948	919	94% 58% 87% 87%
regex-dna					
Java 7	24.40	8.51	711,372	929	81% 63% 78% 66%
C++ g++	5.64	3.90	193,828	695	11% 76% 35% 25%
n-body					
Java 7	22.60	22.59	18,384	1424	100% 0% 0% 0%
C++ g++	9.79	9.79	436	1545	1% 1% 100% 0%
k-nucleotide					
Java 7	1.28	0.43	33,612	1630	74% 88% 63% 77%
C++ g++		Make Error			

- **Desi mai lent, avantajul este ca byte-codul este independent de arhitectura**, deci compatibil cu platforme diverse.

Un program Java poate fi executat pe orice masina/sistem de operare care dispune de un subsistem **run-time Java** (implementarea mediului Java) si de un **interpretor Java**. (JRE)

Sursa: http://benchmarksgame.alioth.debian.org/u64q/java.php

Platforma Java

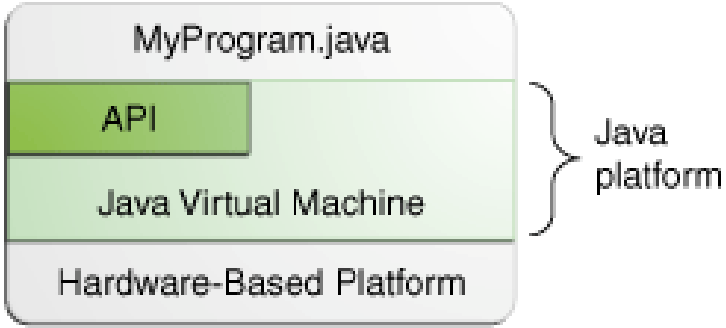
(O *platforma* = mediul hardware + software in care ruleaza un program.
Cele mai populare platforme: Microsoft Windows, Linux, Mac OS, Solaris OS …

Majoritatea platformelor sunt descrise ca o combinatie intre SO si hardware-ul pe care acesta e instalat.)

Platforma **Java** este **o platforma exclusiv software** care ruleaza pe alte platforme (bazate pe hardware).

Platforma Java are 2 componente: **JVM** (*Java Virtual Machine)* si **API**(*Java Application Programming Interface*).

API – o colectie ampla de componente software deja implementate care pun la dispozitie multe facilitati – grupate in module care contin pachete (care contin clase si intefete) – similare cu bibliotecile C/C++

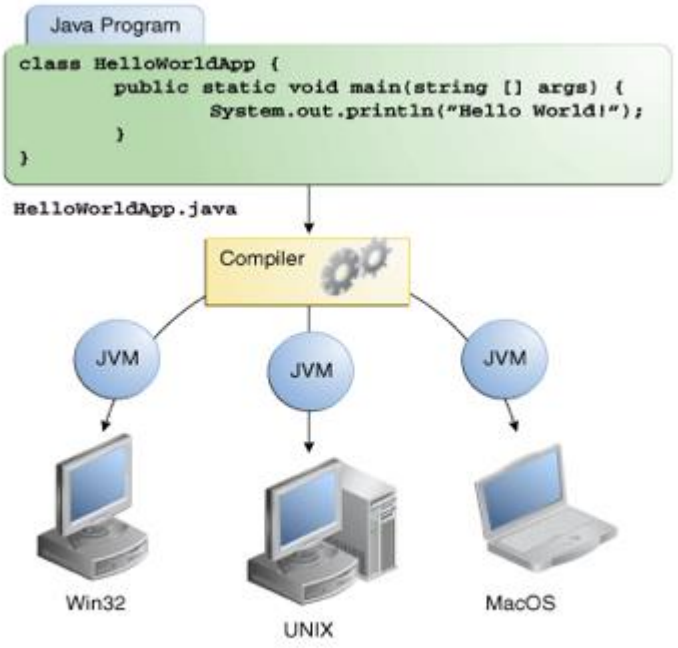


(sursa: http://docs.oracle.com/javase/tutorial/index.html)

➢ Independent de arhitectura

• Se spune despre aplicatiile Java ca sunt de tip ***"compile once, run everywhere"***.

Codul intermediar (byte-code-ul) este independent de arhitectura, deci poate rula oriunde exista o masina virtuala Java.



(sursa: http://docs.oracle.com/javase/tutorial/index.html)

•In legatura cu acest aspect, trebuie subliniat ca exista doua extreme: pe de-o parte se situeaza **limbajele de nivel inalt interpretate** (scripturi), foarte puternice, dar foarte lente (de ex. Perl), iar de cealalta parte **limbajele compilate** (de ex. C++), in care codul generat este puternic optimizat in functie de arhitectura masinii gazda.

•Java se situeaza undeva la mijloc: compilatorul produce un cod intermediar, iar acest cod este interpretat.

➢ **Portabil**

•**Independenta de arhitectura este aspectul principal al portabilitatii**, dar nu e singurul.
In Java nu exista chestiuni lasate la latitudinea implementarii.
De exemplu, toate tipurile numerice au dimensiunea si comportarea aritmetica precizata.
In C/C++ nu este asa, de exemplu, int este reprezentat pe 32 de biti sau pe 16.

In Java:

boolean	—	—	—
char	16-bit	Unicode 0	Unicode 2^{16}- 1
byte	8-bit	-128	+127
short	16-bit	-2^{15}	$+2^{15}—1$
int	32-bit	-2^{31}	$+2^{31}—1$
long	64-bit	-2^{63}	$+2^{63}—1$
float	32-bit	IEEE754	IEEE754
double	64-bit	IEEE754	IEEE754
void	—	—	—

- **Distribuit**

• **Java a fost destinat din start aplicatiilor in retele de calculatoare**.
Pachetul **java.net** ofera clase si metode pentru lucrul in retea.
De exemplu, a deschide un fisier pe sistemul local este o operatie la fel de simpla ca a deschide un fisier aflat pe un calculator din retea.
Se pot dechide URL-uri (locatii Internet), crea aplicatii client-server etc., direct din limbajul Java, fara a apela la aplicatii suplimentare.

- **Robust**

•**Java a fost creat pentru aparatura electronica de larg consum (home appliances)**, ceea ce a impus restrictii severe de fiabilitate, adica o robustete sporita.

•**Limbajul este strongly-typed** => la compilare se fac verificari ample ale tipurilor de date. Nu exista tipuri universale (ca void * in C). Modelul de memorie este fara pointeri, iar obiectele se creeaza numai prin constructori si alocare dinamica (cu new).

Eliberarea memoriei nu mai este sarcina programatorului, ci a colectorului de spatiu disponibil, ceea ce **elimina multe erori logice de programare**.

•**Indicii la operatiile cu tablouri se verifica implicit**, ceea ce exclude multe erori logice.
In C/C++ puteam avea un tablou: int a[10] si puteam scrie linistit a[12], fara a avea eroare la compilare.
Asa ceva nu se intampla in Java; va fi aruncata o exceptie.

• **Gestiune exceptiilor se face prin blocuri try-catch**.

➢ Sigur

• **Fiind destinat aplicatiilor de retea (de ex. Internet), Java are mecanisme de securitate** care sa impiedice accesul neautorizat, incarcarea unor programe suspecte etc.

De exemplu, in aplicatiile Internet se descarca programe Java (applet-uri) de la distanta, si acestea se executa pe masina locala.

Exista restrictii si verificari ample ale acestui gen de programe, care se fac inainte de rularea lor.

Clasele incarcate din retea sunt plasate in alt spatiu de memorie decat clasele incarcate de pe sistemul local; nu au acces la fisierele de pe client, etc

➢ Multithreaded

• **Limbajul Java include concepte si mecanisme pentru programarea concurenta**.
Un thread (fir de executie) este o unitate de program care se executa concurent cu alte asemenea unitati.
De exemplu, o aplicatie Java care efectueaza operatii de intrare-iesire, ar putea executa aceste operatii intr-un thread, iar calculele interne in alt thread.
Daca masina are un singur procesor, aceste threaduri isi vor partaja timpul de executie, insa timpii de asteptare necesari operatiilor de intrare-iesire nu vor mai fi pierduti, ci vor fi folositi de celalalt thread.

• Noutatea consta in faptul ca Java are mecanisme de acest gen la nivel de limbaj (apar si in C++ 11).

➢ Dinamic

• **Obiectele se incarca in memorie doar atunci cand este nevoie de ele**.
Similar, identificarea metodelor concrete care trebuie apelate (de exemplu, in cadrul suprascrierii unei metode intr-o clasa derivata), se face la executie.

Aceasta proprietate exista si la C++ in cazul metodelor virtuale.

In Java, ea este implicita (toate metodele nestatice sunt implicit virtuale).

C12: Java - Introducere

Principalele diferente fata de limbajul C++

1. Limbajul Java este pur obiectual

Asta inseamna ca nu pot exista variabile, obiecte, metode etc. in afara claselor ca in C++.

Nu exista notiunea de variabila globala.

Una dintre clasele care compun aplicatia trebuie sa contina o metoda cu numele ***main***, statica - punctul de start al aplicatiei.

Exemplu:

```
class MyClass {
        public static void main(String [] args) {//main are o singura instanta - publica
                System.out.println("Hello World!");
        }
}//nu se pune ; dupa clase
```

Acest program consta dintr-o definitie de clasa.
Clasa contine metoda ***main***, care indica locul din care interpretorul Java va incepe executia.
Metoda ***main*** trebuie sa fie *publica si statica*, ceea ce asigura o unica instanta a acesteia.
Neaparat, ea TREBUIE sa *fie de tip void*.

Metoda ***main*** apeleaza o functie de biblioteca, ***println***, care face parte din pachetul ***System.out*** - ***println*** afiseaza pe consola siruri de caractere (care in Java sunt obiecte de tip String), dar putem sa-i furnizam spre afisare ca parametru orice primitiva.

Metoda **main** are ca parametru un tablou de String-uri (parametrii din linia de comanda, la fel ca al C standard/C++).
De observat ca, spre deosebire de C, lipseste numarul de argumente.
Aceasta diferenta apare deoarece tablourile din Java sunt obiecte, iar orice tablou are un camp ***length***, care ii precizeaza lungimea.

2. Separare interfata – implementare

Un principiu de baza in programarea orientata - obiect este separatea interfata - implementare.

In C++ separararea e mai puternica decat in Java.

C++ permite ca interfata unei clase sa fi declarata intr-un fisier, separat de implementare . Fisierele header (contin interfata) se folosesc cu directiva # include.

Implementarea se face intr -un fisier cu acelasi nume ca al headerului , dar cu extensie .cpp. Sursa compilata se va gasi intr-un modul obiect cu extensia .o.

In **Java** - o clasa se salveaza intr-un fisier – **exact cu numele clasei si extensie . java.**
Nu avem headere.

Codul compilat –bytecod-ul – se salveaza intr-un fisier cu numele clasei si extensie **.class**.

Clasele client vor **importa** aceasta clasa.

3. In Java, fiecare clasa compilata este memorata intr-un fisier distinct.
Numele acestui fisier trebuie sa coincida cu cel al clasei, avand extensia .class.
Daca o clasa face parte dintr-un pachet, fisierul .class trebuie sa se afle intr-o structura de directoare care reflecta structura pachetului. Daca o clasa are nume complet:

ProiectX.ClasePentruPartea1.MyClass.class

atunci calea fisierului e:

ProiectX/ClasePentruPartea1/MyClass.class

Un fisier sursa Java trebuie sa aiba extensia .java.
El poate consta dintr-una sau mai multe definitii de clase.

Daca exista mai multe clase intr-un fisier java, doar una trebuie sa fie declarata public (adica disponibila in afara pachetului) si sa aiba un nume care coincide cu cel al fisierului sursa (mai putin extensia .java).

➔ In aceasta situatie se vor genera mai multe fisiere .class, cate unul pentru fiecare clasa.

***De exemplu**, sa presupunem ca fisierul A.java contine definitia claselor A (publica) si B. Din compilare vor rezulta fisierele A.class si A$B.class.*

Daca in **C++** avem mai multe clase in acelasi fisier - > se genereaza un singur modul obiect, continand tot codul compilat.

4. **Instructiunea package** specifica din ce pachet face parte clasa din fisierul curent.
Ea trebuie sa apara ca prima instructiune din fisierul sursa.

Clasa care face parte dintr-un pachet are acces la toate clasele (publice sau ne-publice) din acel pachet si la toate clasele/metodele care nu sunt private din toate celelalte pachete.

Daca o clasa este declarata ca facand parte dintr-un pachet cu nume, fisierul .class trebuie sa fie plasat intr-o structura de directoare adecvata.
De regula, compilatorul Java salveaza fisierul .class intr-o asemenea structura.

Daca se omite *instructiunea package din textul sursa, codul din acel fisier face parte dintr-un pachet predefinit anonim.*

Acest stil este util pentru programe simple si inseamna ca acel cod va fi interpretat ca fiind in directorul curent.

5. **In Java nu exista directiva #include** si niciun fel de directiva preprocesor (nu exista preprocesor).

Se foloseste instructiunea import care permite accesul la alte clase prin numele lor.

Instructiunile import trebuie sa apara dupa instructiunea package, dar inainte de orice definitie de clasa.

Exista mai multe forme de import:

```
import nume_pachet.nume_clasa;

import nume_pachet.*;
```

nume_pachet.* - se pot folosi toate clasele din acel pachet.

6. **In Java exista urmatoarele reguli de acces/vizibilitate la pachete, clase si membri ai claselor**

- Un pachet este disponibil daca fisierele si directoarele respective sunt disponibile.

- Toate clasele si interfetele dintr-un pachet sunt accesibile tuturor claselor si interfetelor din acel pachet. Nu este posibil sa definim clase care sa fie vizibile doar intr-un singur fisier sursa.

- Clasa declarata ***public*** intr-un pachet este accesibila din orice alt pachet. O clasa ***ne-publica*** nu este disponibila in afara pachetului din care face parte.

- Modificatori de acces pentru atribute/ metode:

Modificator	In clasa	In pachet	In subclase	In afara pachetului
Public	Da	Da	Da	Da
Protected	Da	Da	Da	Nu
Private	Da	Nu	Nu	Nu
Neprecizat/ default	Da	Da	Nu	Nu

7. **Tipuri primitive de date**

In Java **toate** variabilele de tipuri primitive sunt initializate automat.
Ultima coloana precizeaza initializarile implicite.

Tip	Contine	Valoare implicita
boolean	true, flase	false
char	caracter Unicode	\u0000
byte	intreg cu semn pe 1 octet	0
short	intreg cu semn pe 2 octeti	0
int	intreg cu semn pe 4 octeti	0
long	intreg cu semn pe 8 octeti	0
float	real simpla prec. (4 octeti)	0.0
double	real dubla prec. (8 octeti)	0.0

Tipurile numerice pot avea valori speciale :
+infinit, -infinit, -0 si NaN (Not a Number).

Valoarea NaN comparata (<, >, == etc.) cu orice alta valoare reala, inclusiv NaN, produce false.

Se pot utiliza metodele Float.isNan() sau Double.isNaN() pentru a testa daca o valoare este NaN.

8. **Tipurile non-primitive sunt:**

- **tablourile**
- **clasele predefinite (din API)**
- **clasele implementate de programator**

Variabilele de acest fel sunt manipulate prin **handleri** (“prin referinta“), adica adresa unui obiect sau tablou este memorata intr-o variabila, transmisa catre o metoda etc. Obiectele propriu zise sunt stocate in heap.

Doar tipurile primitive sunt tratate "prin valoare".

Atentie la modul in care transmiteti parametrii in functii!

Primitivele vor fi transmise prin valoare (daca veti modifica parametrul, modificarile nu se vor salva la iesirea din functie).

Restul variabilelor se transmit “prin referinta” (orice modificare se va face direct la adresa obiectului transmis ca parametru).

Tipurile non-primitive se initializeaza (li se creeaza spatiu de memorie) numai folosind cuvantul rezervat new urmat de un apel de constructor.

E posibil ca variabile diferite sa se refere la acelasi obiect (spatiu de memorie). De exemplu:

```
Persoana p, q; //sunt declarate variabilele p si q de tip Persoana
p = new Persoana();   //p este instantiat prin apel constructor- exista in memorie;
q = p;  //atribuire de handleri; q adreseaza acelasi obiect ca p
```

Aici p si q se refera la acelasi obiect (cel creat prin apelul constructorului).

Atribuirea de variabile ne-primitive este de fapt atribuire de referinte.
De exemplu, in secventa:

```
Persoana p, q;
p = new Persoana(“Ana”);
q = new Persoana(“Maria”);
p = q;
```

p contine o referinta catre obiectul referit de q, iar obiectul pe care il referea p este "pierdut".

OBS: In Java nu avem destructori.

Pentru a copia obiecte, utilizam metode de tip ***clone*** :

```
Vector<int> a, b;  // a si b sunt = null
b = new Vector<int>(); //se creaza un obiect de tip vector
b.add(2); b.add(3);  //se adauga elemente in vector cu crestere automata de spatiu
a = b.clone(); // nu se pot supradefini operatori de atribuire
```

sau constructori de copiere:

```
a = new Vector<int>(b);
```

iar pentru a testa egalitatea, metode de tip ***equals***:

```
if (a.equals(b)) { ... }
```

Valoarea implicita pentru toate referintele este null (semnifica absenta unei referinte la un obiect).

Intr-o metoda nestatica, avem la dispozitie referinta this (catre obiectul curent).
Avem acces la membrii clasei prin:

```
this.nume_membru
```

Vectori /tablouri

In C/C++:

```
int a[10];
int *a1=new int[10];
```

In Java:

```
//se aloca spatiu doar dinamic
        int []a=new int[10];
```

sau: int a[] =new int[10]; **se aplica si in continuare*

```
int a2[]={1,2,2,4};
```

```
int []a2={1,2,3};
```

```
// acces la un element a2[i]
```

```
//acces la un element a2[i]
```

```
int matr[][2]={{1,1},{1,2},{1,1}};
```

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

```
int **matr1;//trebuie alocat spatiu
//si initializate elementele cu 0

for (int i=0;i<2;i++)
{cout<<endl;
 for (int j=0;j<3;j++)
     cout<<matr1[i][j]<<" ";
  }
```

```
int [][]matr1=new int[2][3];
//automat toate elementele sunt 0

for (int i=0;i<matr1.length;i++)   //2
{  System.out.println();
   for (int j=0;j<matr1[i].length;j++)   //3
            System.out.print(matr1[i][j]+" ");
}
```

Exemplu program C++

```
#include <iostream>
using namespace std;

void sortare(int *v, int dim){
  int aux;
  for (int i=0;i<dim-1;i++)
            for (int j=i+1;j<dim;j++)
                        if(v[i]>v[j]){
                            aux=v[i];
                            v[i]=v[j];
                            v[j]=aux;
                         }
}

int main(){
   int v[]={5,3,2,1};

   sortare(v,4);

   for (int i=0;i<4;i++)
                        cout<<v[i]<<end;
   return 0;
}
```

```
public class Exemplu {
public /*static*/ void sortare(int []v, int dim){
//pot sa nu transmit dim si sa folosesc v.length
   int aux;
   for (int i=0;i<dim-1;i++)
            for (int j=i+1;j<dim;j++)
                        if(v[i]>v[j]){
                            aux=v[i];
                            v[i]=v[j];
                            v[j]=aux;
                        }
}

public static void main(String[] args){
   int v[]={5,3,2,1};   // int []v={5,3,2,1};
   Exemplu test=new Exemplu();
   test.sortare(v,4);//Exemplu.sortare(v,4);

   for (int i=0;i<4;i++)
        System.out.println(v[i]);
}

}
```

9. Cuvantul cheie static

Ca si in C++: cand se creaza o clasa – este descris modul in care vor arata si se vor comporta atributele si metodele.

Nimic nu se intampla efectiv pana cand nu se instantiaza un obiect de acel tip (i se aloca spatiu) si metodele se pot apela.

Aceasta afirmatie este adevarata, mai putin in 2 cazuri:

- dorim sa avem alocat spatiu pentru o variabila – folosita in comun de toate obiectele.
- avem nevoie de o metoda care nu are legatura directa cu atributele clasei si pe care vrem sa o folosim chiar daca nici un obiect nu a fost creat.

-> in aceste contexte se foloseste cuvantul **static**

OBS: Metodele statice nu au nevoie ca un obiect sa fie creat ca ele sa fie apelate, dar nici nu pot accesa direct atribute sau alte metode ne-statice.

```
class StaticTest {
static int i = 47;

//chiar daca se instantiaza doua obiecte de tip StaticTest  se va aloca doar o data spatiu
//pentru atributul i

public static void main(String[] args){

StaticTest obj1 = new StaticTest();
StaticTest obj2 = new StaticTest();

//Atat obj1.i cat si obj2.i se refera la aceeasi zona de memorie.
//In general, este de preferat ca referirea la atributele statice sa se faca cu numele clasei

StaticTest.i++;
// obj1.i cat si obj2.i au valoarea 48.

}
```

!Constructorul fara parametrii este generat in mod automat daca niciun alt constructor nu a fost implementat.

Manipularea atributelor statice se poate face cu metode statice :

```
class StaticFunc {
private static int i=0;

public static void incr() {
StaticFunc.i++;
}
//main
}
```

Metoda statica incr( ) modifica un membru static al clasei si se poate apela:

```
//in main
StaticFunc sf = new StaticFunc();
sf.incr();
```

sau :

```
StaticFunc.incr();
```

Cuvantul static este esential pentru definirea metodei main( ) – punctul de intrare in program.

10. **In Java nu putem defini constante simbolice** (nu avem preprocesor) **si nici variabile cu atributul const (mai mult, cuvantul const nu exista in niciun context)**.

Se foloseste atributul ***final***, care inseamna ca variabila, dupa ce e intializata cu o valoare, nu mai poate fi apoi modificata.

De regula, dorim ca o constanta sa fie instantiata o singura data, ca atare folosim si atributul ***static***.

```
public class FiguraGeometrica{
        .........
        public static final double PI = 3.14159.....;
        .........
}
```

C12: Java - Introducere

Unde “traiesc” variabilele?

- **invocarea metodelor** apare in stiva programului

- variabilele de tipuri primitive declarate intr-o metoda sau ca parametri/tip de date returnat ai/al metodei isi vor salva valorile in stack

- variabilele de tipuri neprimitive – au doar handler-ul/referinta in stack, obiectul efectiv “traieste” in **heap**)

```
//..
public int functie(int i)  //cand va fi apelata va fi incarcata in stack impreuna cu parametrul
{                          // i transmis prin valoare; similar pentru valoarea returnata
int aux = i;               //aux va fi incarcat in stack (valoare)
altaFunctie(aux);          //altaFunctie si aux sunt incarcate in stack
String s=new String(“ceva in heap”); // handlerul s pe stack, obiectul String in heap
return aux;
}
//..
```

C12: Java - Introducere

Un exemplu simplu – clasa Produs (Produs.java)

```
class Produs {
   private String nume;    //declar o variabila instanta (atribut al clasei); fara initializare
   private int pret;          //declar o variabila instanta (atribut al clasei)
  //vizibilitatea atributelor si metodelor se precizeaza pentru fiecare element in parte

  public Produs(String s, int n) {
            this.nume=new String(s);
            this.pret=n;
    }

  public Produs() {//pot, ca si in C++, sa supradefinesc functii
           this("default",0);//aici se apeleaza constructorul cu parametri pentru Produs
                          //apelul trebuie sa se faca pe prima linie a constructorului
  } //nu se genereaza automat daca am implementat alt constructor
  public Produs(Produs a) { //creez obj nou – copie a lui a
            this(new String(a.nume), a.pret);
    } //nu se genereaza automat ca in C++
      // obiectele nu se transmit niciodata prin valoare; sunt “pointeri”
  public void clone(Produs a) { //copiez valorile atributelor din obiectului a in this
            nume=new String(a.nume);
            pret=a.pret;
    }
```

```
    public void setNume(String n){
            nume=new String(n);
    }

    public void afisare()
    {       System.out.println("Produsul:");
            System.out.println("  "+nume);//afisez doua stringuri concatenate
            System.out.println("  "+pret);
            System.out.println();
            }

public static void main(String[]args) //main-ul e in clasa
{           Produs p=new Produs();//instantiez un obiect
            p.afisare();
            Produs p1=new Produs("carte",50);
            p1.afisare();
            p=p1;//atribuire de handleri;
                  // p si p1 adreseaza aceasi zona de memorie
            p1.setNume("alta carte");
            p.afisare(); //modificarea lui p1 se reflecta si in p; am folosit =
             Produs p2=new Produs();
             p2.clone(p1); //copie profunda
             //  Produs p2=new Produs(p1);
}//main
}//clasa
```

```
Produsul:
 default
 0

Produsul:
 carte
 50

Produsul:
 alta carte
 50

Produsul:
 alta  carte
 50
```

Alt exemplu de “compozitie”

```
//in fisierul Adresa.java

public class Adresa {
//atributele pot avea valori default
   private String str=new String();
   private int nr=0;
   private char[]cod=new char[6];

  public Adresa() {//am deja valori default
            //pentru atribute
   }

    public Adresa(String s, int n, char[]c) {
            str=new String(s);
            nr=n;
            for (int i=0;i<6;i++) cod[i]=c[i];
   }

   public Adresa(Adresa a) {
            //this(a.str, a.nr, a.cod); //sau:
            str=new String(a.str);
            nr=a.nr;
            for (int i=0;i<6;i++) cod[i]=a.cod[i];
   }

public void afisare(){
  System.out.println("adresa:");
  System.out.print("Strada "+str+" nr. "+nr+
                                        " cod: ");
    for (int i=0;i<6;i++)
          System.out.print(cod[i]);
    System.out.println();
    System.out.println();
    }
}
```

```
//in fisierul Persoana.java
public class Persoana{
private Adresa adr=new Adresa();
private String nume=new String("default");//instantierea atributelor
private int varsta=0;

public Persoana(){
//am instantiat deja atributele si au valori default
}

public Persoana(String n, int v,Adresa a){
  nume=new String(n);
  varsta=v;
  adr=new Adresa(a); //vreau o copie profunda
}

public Persoana(Persoana p) {
  this(p.nume, p.varsta, p.adr);
  //nume=new String(p.nume);
  //varsta=p.varsta;
  //adr=new Adresa(p.adr); //vreau o copie profunda
}
```

```
public int getVarsta(){
   return varsta;
}

public void setVarsta(int v){
    this.varsta=v;
}

public void afisare(){
    System.out.print("Persoana "+nume+" cu varsta "+varsta+" locuieste la ");
    adr.afisare();
}

public static void sortare(Persoana []v){ //metoda statica; nu transmit ca parametru
 Persoana aux;                                  // lungimea vectorului
 for (int i=0;i<v.length-1;i++)
            for (int j=i+1;j<v.length;j++)
                        if(v[i].getVarsta()>v[j].getVarsta())
                                                {aux=v[i];
                                                v[i]=v[j];
                                                v[j]=aux;}
}
```

```
public static void main(String[]args) //main-ul e in clasa Persoana
{
char[]c={'c','1','c','1','c','1'};
Adresa a=new Adresa("Undeva",2,c);
a.afisare();

Persoana p=new Persoana("Ana",20,a);

Persoana []vec=new Persoana[3];//tablou cu dimensiune 3 – contine handleri de tip Persoana
//fiecare obiect din vector trebuie instantiat

vec[0]=new Persoana(p);
p.setVarsta(p.getVarsta()-3);

vec[1]=new Persoana(p);

vec[2]=new Persoana("Ion",2,a);

Persoana.sortare(vec);

for (int i=0;i<3;i++) vec[i].afisare();
}//de la main

}//nu se pune ; dupa clasa
```

```
adresa:
Strada Undeva nr. 2 cod: c1c1c1

Persoana Ion cu varsta 2 locuieste la adresa:
Strada Undeva nr. 2 cod: c1c1c1

Persoana Ana cu varsta 17 locuieste la adresa:
Strada Undeva nr. 2 cod: c1c1c1

Persoana Ana cu varsta 20 locuieste la adresa:
Strada Undeva nr. 2 cod: c1c1c1
```

Pentru a realiza aplicatii

1. Instalati JDK-ul pentru sistemul de operare pe care il folositi

Downloadati de aici:
https://www.oracle.com/technetwork/java/javase/downloads/index.html

2. Instalati un IDE – recomand pentru inceput Jcreator (cautati versiunea pentru SO vostru), altfel - > Eclipse sau IntelliJ IDEA

Downloadati de aici:
http://www.jcreator.org/download.htm
https://eclipse.org/downloads/

3. Pentru API (documentatia in care sunt descrise toate clasele si metodele organizate pe pachete) folositi linkul:

https://docs.oracle.com/en/java/javase/11/

C12: Java - Introducere

IDE- uri (Integrated Development Environment)

JCreator - gratuit – interfata simpla – recomandat in faza de inceput - *

Eclipse – gratuit - *

IntelliJ IDEA – gratuit - *

NetBeans - gratuit

BEA Workshop

BlueJ - gratuit

CodeGuide - comercial

DrJava - gratuit

JBuilder - comercial

JDeveloper - comercial, platforma multipla

KDevelop - gratuit (platforma GNU/Linux, Cygwin)