

Programare orientată pe obiecte

#11

JAVA

Clase. Variabile. Domeniul de vizibilitate a variabilelor

Adrian Runceanu

www.runceanu.ro/adrian

Curs 11

Clase. Variabile. Domeniul de vizibilitate a variabilelor (partea I)



Clase. Variabile. Domeniul de vizibilitate a variabilelor

1. Concepte fundamentale ale programării orientate obiect în Java:

1.1. Încapsulare

1.2. Moștenire

1.3. Polimorfism

2. Crearea claselor de obiecte:

2.1. Definirea claselor

2.2. Modificatorii pentru tipurile de clasa

2.3. Modificatorii de acces

3. Variabilele (campurile) clasei de obiecte:

3.1. Declararea variabilelor de instanta

3.2. Declararea variabilelor de clasa

3.3. Declararea constantelor

1. Concepte fundamentale ale programarii orientate obiect (OOP) in Java

In **Java** un obiect este o **variabila complexa** care se caracterizeaza prin:

1. o **structura**, descrisa de attributele (proprietatile) sale
2. o **stare**, descrisa de valorile pe care le ia la un moment dat attributele sale
3. un **set de operatii** prin intermediul carora se poate manevra (accesa sau modifica) starea sa

Clase. Variabile. Domeniul de vizibilitate a variabilelor

1. Concepte fundamentale ale programării orientate obiect în Java:

1.1. Încapsulare

1.2. Moștenire

1.3. Polimorfism

2. Crearea claselor de obiecte:

2.1. Definirea claselor

2.2. Modificatorii pentru tipurile de clasa

2.3. Modificatorii de acces

3. Variabilele (campurile) clasei de obiecte:

3.1. Declararea variabilelor de instanta

3.2. Declararea variabilelor de clasa

3.3. Declararea constantelor

1.1. Conceptul de încapsulare

- Obiectul trebuie privit ca o unitate atomică la care utilizatorul nu ar trebui să aibă acces direct.
- Acest principiu al POO este cunoscut sub numele de “ascunderea informației”.
- Acest principiu spune ca *un obiect poate fi accesat numai prin intermediul metodelor care au fost furnizate împreună cu obiectul.*

1.1. Conceptul de încapsulare

- In legatura cu principiul “ascunderii informatiei”, programarea orientata obiect a introdus conceptul de încapsulare.
- **Încapsularea** înseamnă *gruparea datelor și a operațiilor asupra acestor date în același întreg (agregat) având grijă să se ascundă detaliile de implementare (proiectare-realizare) ale acestui întreg*.
- Deci, datele sunt “ascunse”, iar accesul la aceste date se realizeaza numai prin intermediul metodelor încapsulate cu ele.

Clase. Variabile. Domeniul de vizibilitate a variabilelor

1. Concepte fundamentale ale programării orientate obiect în Java:

1.1. Încapsulare

1.2. Moștenire

1.3. Polimorfism

2. Crearea claselor de obiecte:

2.1. Definirea claselor

2.2. Modificatorii pentru tipurile de clasa

2.3. Modificatorii de acces

3. Variabilele (campurile) clasei de obiecte:

3.1. Declararea variabilelor de instanta

3.2. Declararea variabilelor de clasa

3.3. Declararea constantelor

1.2. Moștenirea

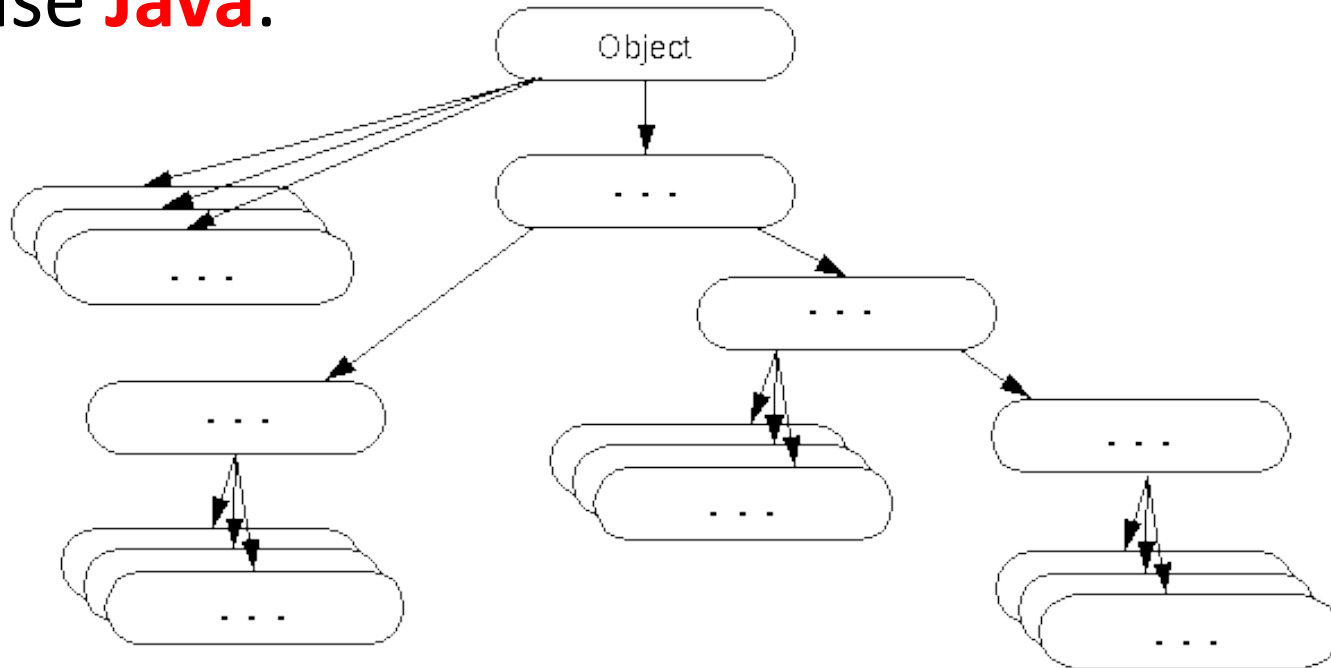
- este un alt concept fundamental al POO.
- Moștenirea *permite unei clase sa moștenească atributele și metodele unei alte clase existente.*
- Prin moștenire, o clasă nouă dobândește imediat tot comportamentul unei clase existente.
- Această clasă nouă se numește clasă derivată din clasa existentă.

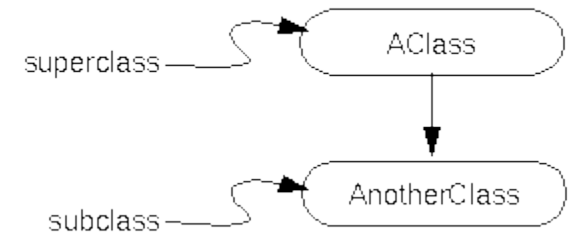
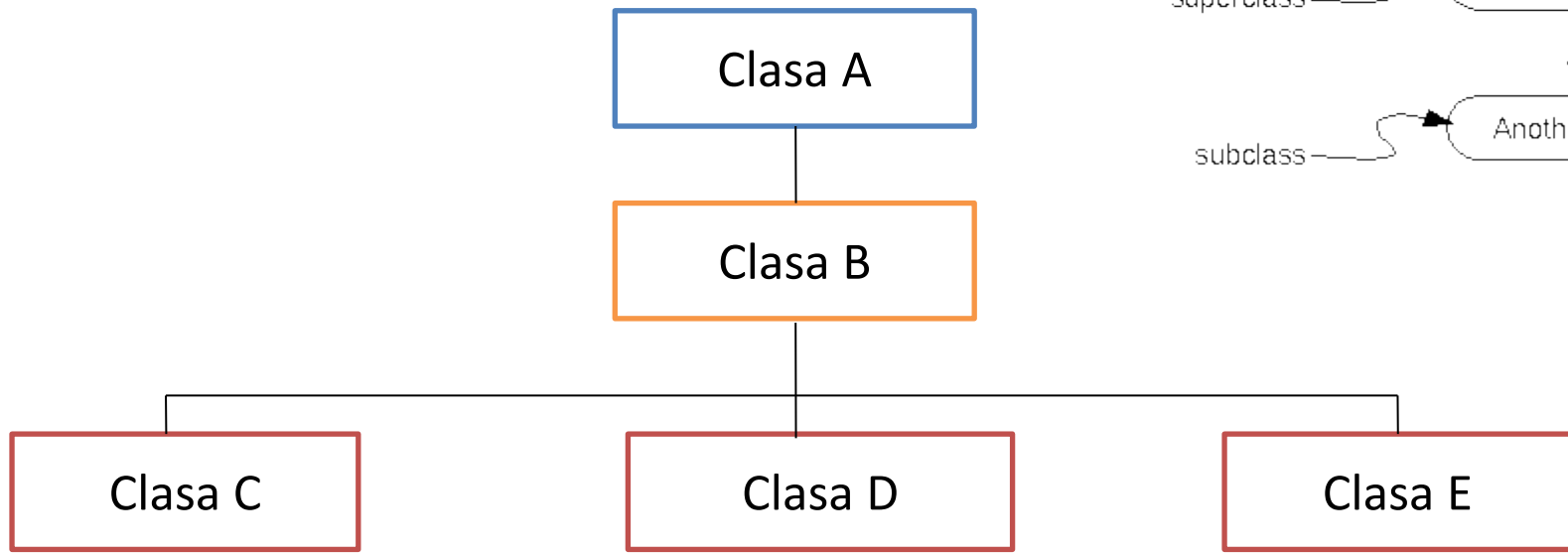
1.2. Moștenirea

- *O clasa de obiecte derivata dintr-o alta clasa existenta pastreaza toate proprietatile si metodele acesteia din urma aducand, in plus, proprietati si metode noi.*
- Prin mostenire, toate clasele sunt aranjate într-o ierarhie strictă.

1.2. Moștenirea

- De exemplu, într-o ierarhie strictă sunt aranjate și clasele provenite din biblioteca de clase **Java**.





- Clasa A este superclasa a clasei B
- Clasa B este subclasa a clasei A
- Clasa B este superclasa pentru clasele C, D si E
- Clasele C, D si E sunt subclase ale clasei B

1.2. Moștenirea

- In ierarhia de clase, **clasa care mosteneste alta clasa** este denumita **subclasa**, iar **clasa care isi ofera mostenirea** se numeste **superclasa**.
- ***Moștenirea*** *dă* **posibilitatea extinderii funcționalității unui obiect.**
- Cu alte cuvinte se pot crea noi clase de obiecte care sa extindă proprietățile și metodele clasei originale.
- Vom reveni asupra mecanismului de moștenire într-un curs viitor.

Clase. Variabile. Domeniul de vizibilitate a variabilelor

1. Concepte fundamentale ale programării orientate obiect în Java:

1.1. Încapsulare

1.2. Moștenire

1.3. Polimorfism

2. Crearea claselor de obiecte:

2.1. Definirea claselor

2.2. Modificatorii pentru tipurile de clasa

2.3. Modificatorii de acces

3. Variabilele (campurile) clasei de obiecte:

3.1. Declararea variabilelor de instanta

3.2. Declararea variabilelor de clasa

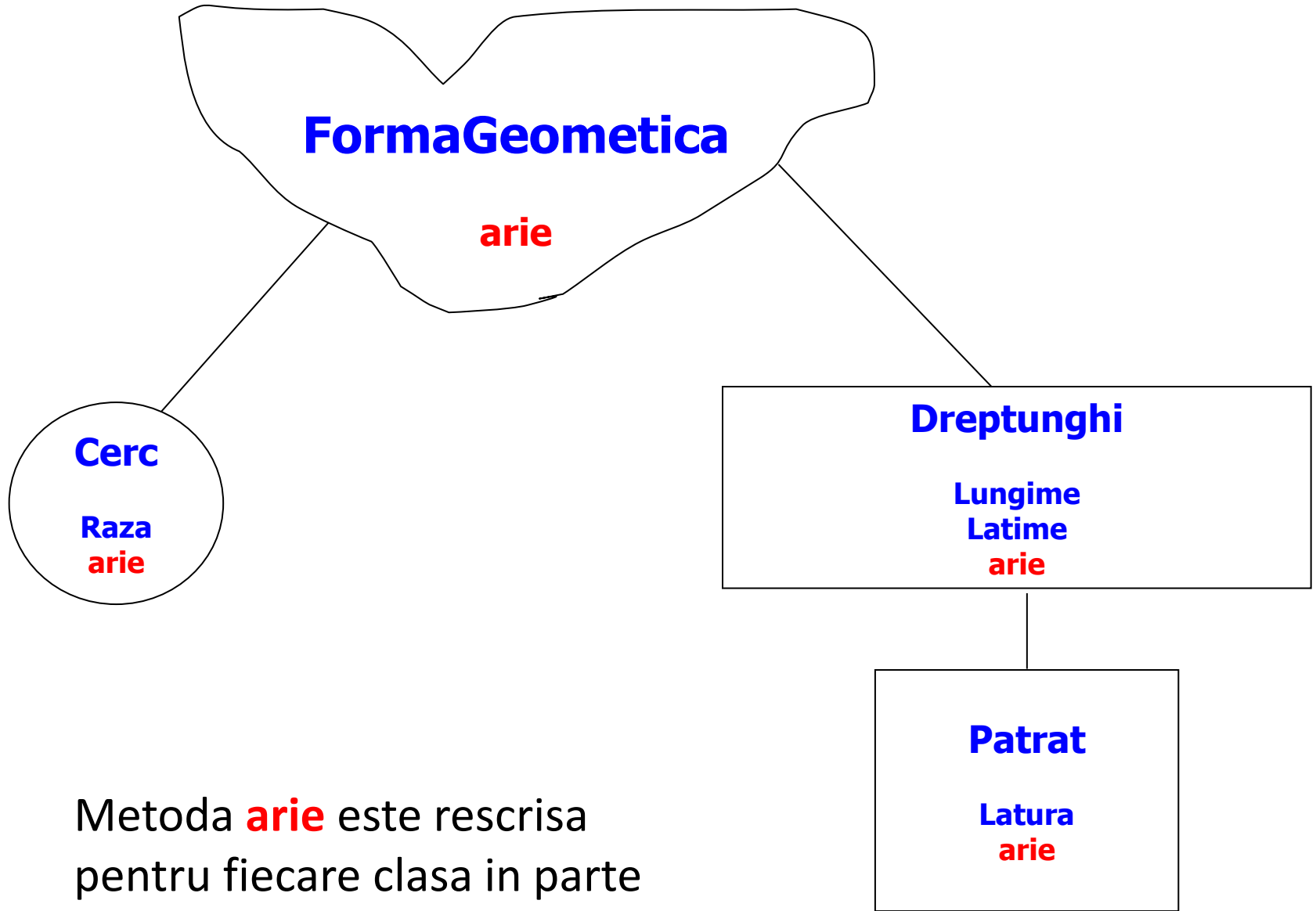
3.3. Declararea constantelor

1.3. Polimorfismul

- este al treilea concept fundamental in POO.
- *Polimorfismul reprezintă capacitatea de a apărea sub diferite forme.*
- De exemplu, în lumea reala, apa apare sub formă solidă, sub formă lichidă sau sub formă gazoasă.
- În **Java**, polimorfismul înseamnă că o singură variabilă referință de tipul unei superclase poate fi folosită pentru a referi mai multe obiecte (instanțe) din clase derivate direct sau indirect din aceeași superclasă, în diferite momente ale execuției unui program.

1.3. Polimorfismul

- Unele dintre proprietatile si metodele definite in superclasa pot fi redefinite (rescrise) in subclasele de obiecte derivate.
- Redefinirea proprietatilor si metodelor in subclasele derivate direct sau indirect dintr-o superclasa ne da, de fapt, o mare flexibilitate in constructia ierarhiei de clase pentru o problema de rezolvat, pentru ca nici o proprietate sau metoda definita intr-un punct al ierarhiei nu este impusa definitiv pentru clasele derivate din acest punct direct sau indirect.
- Vom reveni asupra mecanismului de polimorfism într-un curs viitor.



Metoda **arie** este rescrisa pentru fiecare clasa in parte

1. Concepte fundamentale ale programării orientate obiect în Java

Conceptele fundamentale sunt folosite pentru a îndeplini unul din principalele scopuri ale POO și anume **reutilizarea codului** (refolosirea obiectelor sau refolosirea unor programe).

Clase. Variabile. Domeniul de vizibilitate a variabilelor

1. Concepte fundamentale ale programării orientate obiect în Java:

- 1.1. Încapsulare
- 1.2. Moștenire
- 1.3. Polimorfism

2. Crearea claselor de obiecte:

- 2.1. Definirea claselor
- 2.2. Modificatorii pentru tipurile de clasa
- 2.3. Modificatorii de acces

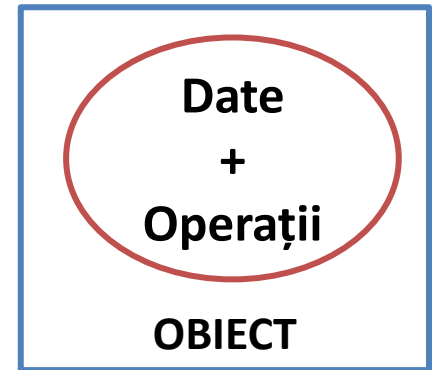
3. Variabilele (campurile) clasei de obiecte:

- 3.1. Declararea variabilelor de instanta
- 3.2. Declararea variabilelor de clasa
- 3.3. Declararea constantelor

2. Crearea claselor de obiecte

O clasă conține membri care pot fi:

- 1. attribute (câmpuri, date)**
- 2. metode (funcții, operații)**



Metodele pot acționa asupra atributelor și pot apela alte metode.

2.1. Definirea claselor de obiecte

O clasă este definită prin cuvântul cheie ***class*** urmat de numele clasei.

Sintaxa folosită este:

```
[<modificatori_acces>] [<modificatori_clasa>]  
  class <nume_clasa> [<clauze_specifice>]  
{  
  <corpul_clasei>  
}
```

- **<modificatori_acces>** - specifică domeniul de vizibilitate (folosire sau acces) al clasei; modificatorul de acces este opțional și poate fi: *public*;
- **<modificatori_clasa>** - specifică tipul clasei definite; modificatorul clasei este optional și poate fi: *abstract, final*;
- **<nume_clasa>** - specifică numele clasei de obiecte; este de preferat ca *numele clasei să înceapă cu o literă majusculă și dacă numele clasei conține în interior mai multe cuvinte, aceste cuvinte să înceapă cu o literă majusculă*;

- **<clauze_specifice>** - specifică anumite clauze referitoare la poziția pe care o ocupă clasa în ierarhia de clase din care face parte (clauza *extends*) sau dacă această clasă folosește o interfață (clauza *implements*); despre aceste clauze vom vorbi într-un curs viitor.
- **<corpul_clasei>** - variabilele clasei (de instanță și de clasă) și metodele care lucrează cu acestea, numite la un loc **membrii clasei**.

Observatie: Conținutul (corpul) unei clase nu este o succesiune de instrucțiuni.

2.2. Modificatorii pentru tipurile de clasa

Tipuri de clase

O clasă poate fi:

1. **abstractă**, caz în care folosim modificatorul ***abstract***
2. **finală**, caz în care folosim modificatorul ***final***

2.2. Modificatorii pentru tipurile de clasa

1. In cazul in care declaram o **clasă de obiecte** ca fiind **abstractă**, compilatorul va interzice instantierea acestei clase, adica *nu se permite crearea de obiecte din aceasta clasa*.

2. In cazul in care declaram o **clasă de obiecte** ca fiind **finală**, *compilatorul va interzice ca pornind de la aceasta clasă să se definească subclase*.

Nota: În cazul in care se declara, în acelasi timp, o clasa de obiecte ca fiind abstracta si finala, eroarea va fi semnalata la compilare, pentru ca cei doi modificatori se exclud.

2.3. Modificatorii de acces

- Sunt cuvinte rezervate ce controlează accesul celorlalte clase la membrii unei clase.
- Modificatorii de acces pentru variabilele și metodele unei clase sunt:
 1. `public`
 2. `protected`
 3. `private`
 4. și cel implicit(***package-friendly***)
- Nivelul lor de acces este dat în tabelul urmator:

2.3. Modificatorii de acces

Modificator de <u>acces</u>	Clasă	Subclasă	Pachet
private	✘		
protected	✘	✘	✘
public	✘	✘	✘
Implicit (package-<u>friendly</u>)	✘	✘	

2.3. Modificatorii de acces

Modificatorii au următoarele semnificații:

1. **private** – un membru declarat private este accesibil doar în clasa în care este declarat;
2. **protected** – un membru declarat protected este accesibil oricărei clase care aparține aceluiași pachet ca și clasa în care este declarat membrul; de asemenea, este accesibil și oricărei subclase a clasei respective;
3. **public** – un membru declarat public este accesibil oricărei clase indiferent de locul unde se află ea.
4. Dacă modificatorul unui membru lipsește, se consideră implicit un modificator "**package-friendly**" care dă acces oricărei clase din pachetul respectiv.

2.3. Modificatorii de acces

Exemple:

```
private int real, imaginar;  
protected String secret;  
public float[] elemente_vector;  
long x, y, z;  
private void metodaInterna();  
public void setReal(int real);
```

Observație: În cazul în care declarăm un membru "**protected**" atunci accesul la acel membru din subclasele clasei în care a fost declarată variabila depinde și de pachetul în care se găsește subclasa: dacă sunt în același pachet accesul este permis, dacă nu sunt în același pachet accesul nu este permis decât pentru obiecte de tipul subclasei.

2.3. Modificatorii de acces

- In cazul in care declaram o clasa de obiecte ca fiind publică, atunci *aceasta clasa poate fi folosita (accesata) si din exteriorul pachetului din care face parte.*
- Daca o clasa nu este declarata ca fiind de tip **public** atunci ea va putea fi folosita (accesata) doar de clasele din cadrul aceluiasi pachet.
- Acest tip de acces la o clasa se numeste **package-friendly** si este implicit in **Java**.

2.3. Modificatorii de acces

Nota:

- *Toate clasele care nu fac parte din nici un pachet, sunt considerate automat ca facand parte din acelasi pachet implicit.*
- Ca o consecinta, accesul de tip **friendly** se aplica pentru toate aceste clase.
- Acesta este motivul pentru care vizibilitatea nu este afectata daca se omite modifierul **public** pentru clasele care nu fac parte dintr-un pachet.
- Totusi, aceasta modalitate de folosire a accesului de tip **friendly** nu este recomandata.

Clase. Variabile. Domeniul de vizibilitate a variabilelor

1. Concepte fundamentale ale programării orientate obiect în Java:

1.1. Încapsulare

1.2. Moștenire

1.3. Polimorfism

2. Crearea claselor de obiecte:

2.1. Definirea claselor

2.2. Modificatorii pentru tipurile de clasa

2.3. Modificatorii de acces

3. Variabilele (campurile) clasei de obiecte:

3.1. Declararea variabilelor de instanta

3.2. Declararea variabilelor de clasa

3.3. Declararea constantelor

3.1. Variabilele (câmpurile) clasei de obiecte

Declararea variabilelor de instanță

- O variabila este considerata ca *variabila de instanță* *dacă este declarată în afara metodelor clasei respective*, *însă valorile variabilelor de instanță se stochează individual în instanțele (obiectele) clasei.*
- Rezulta ca, acestea pot lua valori diferite (se pot modifica) pentru fiecare obiect al clasei respective.
- Toate variabilele de instanță se declara imediat dupa prima linie a definitiei clasei (si dupa acolada deschisa {).

3.1. Variabilele (câmpurile) clasei de obiecte

De exemplu:

```
class Complex {  
    double real, imag;  
    Complex() {  
        real=imag=0;  
    }  
    Complex(double r, double i ) {  
        real=r; imag=i;  
    }  
    Complex(double r) {  
        real=r; imag=0;  
    }  
}
```

Clasa *Complex* contine doua variabile: *real* si *imag*

Deoarece aceste variabile sunt declarate in afara oricarei metode, ele sunt ***variabile (campuri) de instanta***

Clase. Variabile. Domeniul de vizibilitate a variabilelor

1. Concepte fundamentale ale programării orientate obiect în Java:
 - 1.1. Încapsulare
 - 1.2. Moștenire
 - 1.3. Polimorfism
2. Crearea claselor de obiecte:
 - 2.1. Definirea claselor
 - 2.2. Modificatorii pentru tipurile de clasa
 - 2.3. Modificatorii de acces
3. Variabilele (campurile) clasei de obiecte:
 - 3.1. Declararea variabilelor de instanta
 - 3.2. Declararea variabilelor de clasa
 - 3.3. Declararea constantelor

3.2. Declararea variabilelor de clasă

- O variabilă este considerată ca **variabilă de clasă** *dacă este declarată în afara metodelor clasei folosind modifierul **static**.*
- Valoarea variabilei de clasă rămâne aceeași pentru toată clasa în ansamblul ei.
- Rezultă ca, *variabila de clasă are o singură valoare comună tuturor instanțelor clasei în care este declarată.*

3.1. Variabilele (câmpurile) clasei de obiecte

De exemplu:

```
class Punct{  
    int x, y; // variabile de instanta  
    static int contor_puncte=0; // variabila de clasa  
  
    Punct(int xx, int yy){  
        x=xx; y=yy;  
        contor_puncte++;  
    }  
}
```

Clasa *Punct* contine doua variabile de instanta(x si y) si o **variabila de clasa**(contor_puncte)

3.2. Declararea variabilelor de clasă

- Modificarea valorii variabilei de clasă din interiorul unui obiect face ca modificarea să se reflecte în toate celelalte instanțe ale clasei existente în acel moment al execuției programului.
- **Variabilele de clasă** sunt folosite *la comunicarea între diferite obiecte ale aceleiași clase sau pentru păstrarea unor date comune la nivelul întregii clase.*
- Variabilele statice sunt initializate la încărcarea codului specific unei clase și există chiar și dacă nu există nici o instanță a clasei respective.

3.2. Declararea variabilelor de clasă

O *variabila statică poate fi accesată*:

- fie conform regulii generale prin crearea și utilizarea unei instanțe a clasei din care provine (și precalificarea variabilei cu numele instanței)
- fie direct, prin precalificarea variabilei cu numele clasei

3.2. Declararea variabilelor de clasă

Referirea (accesul) la variabilele de instanța și de clasă

Accesarea unei variabile de instanță sau de clasă se realizează în mai multe moduri:

- prin *crearea și utilizarea unei instanțe a clasei în care a fost declarată variabila sau a unei subclase a clasei respective* (ca regulă generală)
- în acest caz se folosește operatorul punct (`.`), în stânga acestuia punându-se numele instanței, iar în dreapta acestuia punându-se numele variabilei

3.2. Declararea variabilelor de clasă

- prin *simpla folosire a numelui sau, in cazul in care clasa in care este accesata variabila este aceeași cu clasa in care a fost declarata;*
- prin folosirea operatorului punct (.), in stanga acestuia punandu-se numele clasei in care a fost declarata, iar in dreapta acestuia punandu-se numele variabilei; aceasta modalitate este folosita numai cand variabila este declarata ca variabila de clasa.

3.2. Declararea variabilelor de clasă

Exemplu:

- Clasa Punct conține două variabile instanță și o variabilă clasă.
- Constructorul clasei Punct inițializează variabilele instanță și incrementează variabila clasă.
- Cu alte cuvinte, variabila contor_puncte va număra câte "puncte" (instanțe ale clasei Punct) se creează.
- În clasa Curs11_3 se creează trei obiecte de tipul Punct și se afișează valoarea variabilei clasă contor_puncte.
- Se observa că variabila contor_puncte este accesată prin intermediul numelui clasei.

Exemplu

```
curs11_3.java ✖
package curs11;

class Punct{
    int x, y; // variabile de instanta
    static int contor_puncte=0; // variabila de clasa

    Punct(int xx, int yy){
        x=xx; y=yy;
        contor_puncte++;
    }
}

public class curs11_3 {
    public static void main(String[] args) {
        Punct p1=new Punct(5, 5);
        Punct p2=new Punct(25, 25);
        Punct p3=new Punct(35, 35);
        System.out.println(Punct.contor_puncte);
    }
}

Problems @ Javadoc Declaration Console ✖
<terminated> curs11_3 [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (01.12.2015, 19:43:55)
3
```

3.2. Declaraarea variabilelor de clasă

Variabile locale

Declaraarea variabilelor locale

- Stim că, prin bloc *înțelegem o secvență (eventual, vidă) de instrucțiuni, cuprinsă între acolade.*
- De asemenea, orice bloc poate conține la rândul său un alt bloc, putând astfel lua naștere o structură imbricată de blocuri.

3.2. Declararea variabilelor de clasă

- Prin **variabila locală** înțelegem *o variabilă declarată în interiorul unei metode, unui bloc, inclusiv într-o instrucțiune for*.
- De asemenea, *parametrii metodei* sunt considerați variabile locale.

Observatie:

- *Numele variabilelor de instanta, de clasa sau locale este de preferat sa inceapa cu o litera mica, iar fiecare cuvânt incepand cu al doilea, in cazul in care exista, sa inceapa cu majuscule.*

Vizibilitatea diferitelor tipuri de variabile dintr-o clasa

```
public class Example
{
    private int var;
    ...

    private void method1()
    {
        int var;
        ... var
        ... this.var
    }

    private void method2()
    {
        ... var
    }
    ...
}
```

instance variable → `private int var;`

local variable → `int var;`

refers to local variable, NOT instance variable → `... var`

refers to instance variable → `... this.var`

refers to instance variable → `... var`

<http://algs4.cs.princeton.edu/12oop/>

Clase. Variabile. Domeniul de vizibilitate a variabilelor

1. Concepte fundamentale ale programării orientate obiect în Java:

1.1. Încapsulare

1.2. Moștenire

1.3. Polimorfism

2. Crearea claselor de obiecte:

2.1. Definirea claselor

2.2. Modificatorii pentru tipurile de clasa

2.3. Modificatorii de acces

3. Variabilele (campurile) clasei de obiecte:

3.1. Declararea variabilelor de instanta

3.2. Declararea variabilelor de clasa

3.3. Declararea constantelor

3.3. Constante

Declararea constantelor locale si ale clasei

- *Constanta este un tip special de variabila a carei valoare nu se modifica niciodata pe parcursul executiei programului.*
- In Java, se pot declara constante pentru toate tipurile de variabile:
 1. de instanta
 2. de clasa
 3. sau locale

3.3. Constante

- Pentru a declara o constanta se foloseste modificatorul *final*.
- De asemenea, constantei i se atribuie o valoare care nu se modifica pe parcursul executiei programului.
- In cazul constantelor locale, atribuirea valorii acestora se face la declarare sau inainte de a fi folosita.

3.3. Constante

- Se mentioneaza ca, modifierul *final* este singurul care poate fi atasat unei constante locale.

De exemplu:

```
final float pi = 3.141592;
```

```
final numarMaxim = 12567;
```

Observatie:

Numele constantelor locale este de preferat sa inceapa cu o litera mica, iar fiecare cuvint incepand cu al doilea, in cazul in care exista, sa inceapa cu o majuscula.

3.3. Constante

- In cazul constantelor care definesc attributele unei clase, atribuirea valorii acestora se face fie prin initializare, in afara corpului tuturor metodelor clasei, fie prin constructori.
- Constantele pot fi folosite pentru denumirea diferitelor stari ale unui obiect (instanta) sau pentru testarea acestor stari.

3.3. Constante

- De exemplu, sa presupunem ca avem o eticheta *text* care poate fi aliniata la stanga, la dreapta sau centrat.
- Aceste constante se pot declara astfel:

```
final int LEFT = 0;  
final int RIGHT = 1;  
final int CENTER = 2;
```

3.3. Constante

Observatii:

- ✓ Daca constanta defineste un atribut al unei intregi clase (adica, al tuturor instantelor clasei) atunci modifierul ***final*** este folosit impreuna cu modifierul ***static***.
- ✓ Numele constantelor care definesc proprietatile clasei este de preferat ***sa fie scris in totalitate cu majuscule, iar daca sunt mai multe cuvinte in componenta unui nume de constanta, atunci acestea sa fie separate prin linie de subliniere (_) ca in exemplul urmator.***

3.3. Constante

- Programul urmator ([LotoConstante.java](#)) ilustreaza modul de folosire al constantelor pentru attributele clasei.
- In jocul de loterie, se selecteaza saptamanal sase numere de la 1 la 49.
- *Programul alege aleator numere pentru 1000 de jocuri si afiseaza apoi de cate ori a aparut fiecare numar in cele 1000 de jocuri.*
- Se definesc trei constante de tip intreg pentru clasa [LotoConstante](#):
 - NUMERE care are valoarea 49,
 - NUMERE_PE_JOC care are valoarea 6
 - si JOCURI are valoarea 1000.

3.3. Constante

```
import java.io.*;
public class LotoConstante
{
    static final int NUMERE = 49;
    static final int NUMERE_PE_JOC = 6;
    static final int JOCURI = 1000;
    // genereaza numere de loterie intre 1 si 49
    // afiseaza numarul de aparitii al fiecarui numar
    public static void main(String[] args) throws
    IOException
    {
```

3.3. Constante

```
// genereaza numerele
int[] numere = new int[50];
for (int i = 0; i < numere.length; ++i)
    numere[i] = 0;
for (int i = 0; i < JOCURI; ++i)
    for (int j = 0 ; j < NUMERE_PE_JOC; ++j)
        numere[(int) (Math.random() * NUMERE) + 1]++;
// numara aparitiile de numere pe joc
// afisare rezultate
for (int k = 1; k <= NUMERE; ++k)
    System.out.println(k + ": " + numere[k]);
}
}
```



```

1  import java.io.*;
2  public class LotoConstante
3  {
4      static final int NUMERE = 49;
5      static final int NUMERE_PE_JOC = 6;
6      static final int JOCURI = 1000;
7      // genereaza numere de loterie intre 1 si 49
8      // afiseaza numarul de aparitii al fiecarui numar
9      public static void main(String[] args) throws IOException
10     {
11         // genereaza numerele
12         int[] numere = new int[50];
13         for (int i = 0; i < numere.length; ++i)
14             numere[i] = 0;
15         for (int i = 0; i < JOCURI; ++i)
16             for (int j = 0 ; j < NUMERE_PE_JOC; ++j)
17                 numere[(int) (Math.random() * NUMERE) +1]++;
18         // numara aparitiile de numere pe joc
19         // afisare rezultate
20         for (int k = 1; k <= NUMERE; ++k)
21             System.out.println(k + ": " + numere[k]);
22     }
23 }

```

Result

CPU Time: 0.10 sec(s), Memory: 36864 kilobyte(s)

```

1: 140
2: 128
3: 127
4: 97
5: 110
6: 143
7: 123
8: 103
9: 114
10: 124
11: 113
12: 131
13: 123
14: 128
15: 129
16: 118
17: 128
18: 122
19: 113
20: 123
21: 114
22: 112
23: 121
24: 135
25: 125
26: 132
27: 123
28: 122
29: 115
30: 133
31: 134
32: 119
33: 114
34: 107
35: 136
36: 125
37: 147
38: 130
39: 138
40: 113
41: 124
42: 136
43: 117
44: 122
45: 116
46: 130
47: 108
48: 91
49: 124

```

Întrebări?