Programare orientată pe obiecte

12

JAVA
Clase. Variabile. Domeniul de vizibilitate a variabilelor (partea II)

Adrian Runceanu
www.runceanu.ro/adrian

2018

Curs 12

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

Clase. Variabile. Domeniul de vizibilitate a variabilelor

4. Domeniul de vizibilitate (acces) al variabilelor folosite in clasele de obiecte:
   4.1. Domeniul de vizibilitate al variabilelor locale
   4.2. Domeniul de vizibilitate al variabilelor clasei
   4.3. Modificatorii de acces (vizibilitate)
5. Metodele unei clase de obiecte:
   5.1. Definirea si apelul metodelor
   5.2. Modificatorii de metoda
6. Metode de instanta si metode de clasa

4.1. Domeniul de vizibilitate al variabilelor locale

Domeniul de vizibilitate al unei variabile locale este constituit din:

1. *partea din blocul in care variabila a fost declarata*, parte ce urmeaza declararii

2. precum si din *subblocurile blocului de instructiuni*, subblocuri care urmeaza declararii

4.1. Domeniul de vizibilitate al variabilelor locale

Variabilele locale exista si pot fi folosite numai pe perioada in care blocul unde sunt declarate este in curs de executie.

O variabila locala nu poate fi redeclarata nici in blocul in care a fost declarata, nici intr-un bloc inclus in acesta.

O variabila locala poate fi redeclarata in blocuri disjuncte.

4.1. Domeniul de vizibilitate al variabilelor locale

Parametrii unei metode sunt vizibili (folosibili) doar in corpul metodei respective, deoarece se comporta ca variabile locale.

Insa este posibil ca intr-un bloc dintr-o metoda sa se redeclare parametrul respectiv.

In acest caz, in partea din bloc ce urmeaza redeclararii, parametrul respectiv este considerat ca avand semnificatia data de redeclarare.

Clase. Variabile. Domeniul de vizibilitate a variabilelor

4. Domeniul de vizibilitate (acces) al variabilelor folosite in clasele de obiecte:
   4.1. Domeniul de vizibilitate al variabilelor locale
   4.2. Domeniul de vizibilitate al variabilelor clasei
   4.3. Modificatorii de acces (vizibilitate)
5. Metodele unei clase de obiecte:
   5.1. Definirea si apelul metodelor
   5.2. Modificatorii de metoda
6. Metode de instanta si metode de clasa

4.2. Domeniul de vizibilitate al variabilelor clasei

Variabilele de instanta si de clasa au un domeniu de vizibilitate extins la intreaga clasa in care au fost declarate, deci ele pot fi folosite de oricare dintre metodele din cadrul clasei fara a fi prefixate cu operatorul punct si numele instantei sau clasei.

- **Java** verifica existenta unei declaratii a unei variabile de instanta sau de clasa in clasa curenta.
- Daca **Java** nu gaseste declaratia variabilei in clasa curenta, o cauta in superclasa corespunzatoare.
- Daca o variabila de instanta sau de clasa este redeclarata intr-o metoda (este folosita ca un parametru sau ca o variabila locala) atunci declararea cea mai interioara este cea care primeaza cand se face o referire la acea variabila.
- Se spune ca variabila redeclarata cu un domeniu de vizibilitate interior “ascunde” (inlocuieste) valoarea originala a variabilei si poate produce erori greu de depanat.

De exemplu, urmatorul program (TestDomeniu.java):

```
class TestDomeniu  {
        int test = 10;
        void printTest(int test) {
                System.out.println("test = " + test);}
        public static void main (String args[]) {
                TestDomeniu st = new TestDomeniu();
                 st.printTest(20);
        }
}
```

curs11_1.java

```
1  package poo;
2
3  public class curs11_1 {
4      int test = 10;
5      void printTest(int test) {
6            System.out.println("test = " + test);
7      }
8      public static void main(String[] args) {
9
10         curs11_1 st = new curs11_1();
11         st.printTest(20);
12     }
13 }
```

- Programul are declarate doua variabile cu acelasi nume.
- Prima, o variabila de instanta, are numele *test* si a fost initializata cu valoarea 10.
- A doua este un parametru cu acelasi nume al metodei *printTest*, insa cu valoarea 20.
- Parametrul *test* din cadrul metodei *printTest* ascunde variabila de instanta *test*.

Metoda printTest apelata in metoda main afiseaza parametrul test cu valoarea 20 si nu variabila de instanta.

Se poate evita aceasta eroare folosind **referinta this** ca o referinta la obiectul curent.

Astfel:

this.test *refera variabila de instanta*

iar numele simplu **test** refera parametrul metodei printTest.

Programul de mai sus se poate, astfel, modifica (TestDomeniuThis.java) pentru a afisa valoarea 10 a variabilei de instanta si nu valoarea parametrului.

```
class TestDomeniuThis  {
        int test = 10;
        void printTest(int test)  {
                System.out.println("test = " + this.test);
        }
        public static void main (String args[]) {
                TestDomeniuThis st = new
TestDomeniuThis();
                 st.printTest(20);
        }
}
```

curs11_1.java

```
1  package poo;
2
3  public class curs11_1 {
4      int test = 10;
5      void printTest(int test) {
6            System.out.println("test = " + this.test);
7      }
8      public static void main(String[] args) {
9
10         curs11_1 st = new curs11_1();
11         st.printTest(20);
12     }
13 }
```

Clase. Variabile. Domeniul de vizibilitate a variabilelor

4. Domeniul de vizibilitate (acces) al variabilelor folosite in clasele de obiecte:
   4.1. Domeniul de vizibilitate al variabilelor locale
   4.2. Domeniul de vizibilitate al variabilelor clasei
   4.3. Modificatorii de acces (vizibilitate)
5. Metodele unei clase de obiecte:
   5.1. Definirea si apelul metodelor
   5.2. Modificatorii de metoda
6. Metode de instanta si metode de clasa

4.3. Modificatorii de acces (vizibilitate) ai variabilelor unei clase

In **Java** exista trei modificatori de vizibilitate ai variabilelor unei clase:

1. modificatorul **public**
2. modificatorul **protected**
3. modificatorul **private**

4.3. Modificatorii de acces (vizibilitate) ai variabilelor unei clase

1. **Modificatorul public** face ca *variabila respectiva sa fie accesibila, prin intermediul operatorului punct, oriunde este accesibila clasa variabilei*.
2. **Modificatorul protected** face ca *variabila respectiva sa fie accesibila in orice clasa din pachetul careia ii apartine clasa in care a fost declarata*. In acelasi timp, *variabila este accesibila in toate subclasele clasei date, chiar daca ele apartin altor pachete*.
3. **Modificatorul private** face ca *variabila respectiva sa fie accesibila doar in interiorul clasei in care a fost declarata*.

4.3. Modificatorii de acces (vizibilitate) ai variabilelor unei clase

Daca pentru o variabila a unei clase nu se precizeaza nici un modificator de acces din cei descrisi mai sus, atunci variabila respectiva devine **package-friendly**.

O variabila friendly este accesibila in pachetul din care face parte clasa in interiorul careia a fost declarata, dar nu este accesibila in subclasele clasei date daca acestea apartin altor pachete.

4.3. Modificatorii de acces (vizibilitate) ai variabilelor unei clase

Nota:
Modificatorii de acces (*public*, *protected*, *private*) sunt plasati primii in declaratia variabilei,
urmeaza apoi modificatorii care determina felul variabilei (*static*, *final*)
si apoi tipul de data al variabilei (referinta sau tip primitiv):

[<modificatori_acces>]
[<modificatori_variabila>] <tip_variabila>
<nume_variabila>

Clase. Variabile. Domeniul de vizibilitate a variabilelor

4. Domeniul de vizibilitate (acces) al variabilelor folosite in clasele de obiecte:
   4.1. Domeniul de vizibilitate al variabilelor locale
   4.2. Domeniul de vizibilitate al variabilelor clasei
   4.3. Modificatorii de acces (vizibilitate)
5. Metodele unei clase de obiecte:
   5.1. Definirea si apelul metodelor
   5.2. Modificatorii de metoda
6. Metode de instanta si metode de clasa

5.1. Definirea metodelor

Definitia unei metode cuprinde patru parti principale:

1. numele metodei
2. o lista de parametrii (definiti prin nume si tip)
3. tipul obiectului sau tipul primitiv de date returnat de metoda
4. corpul metodei

5.1. Definirea metodelor

Primele trei parti ale definitiei metodei formeaza ceea ce se numeste ***semnatura* metodei**.

In plus, o metoda mai poate contine:

- modificatorii care descriu proprietatile metodei si modul de lucru al acesteia
- clauze *throws* care indica exceptiile (erorile) pe care le poate semnala metoda

5.1. Definirea metodelor

Sintaxa definitiei unei metode este:

```
[<modificatori_acces>]
[<modificatori_metoda>] <tip_returnat>
<nume_metoda> ([<param1>, <param2>, …])
[<clauze_specifice>]
{
        <corpul_metodei>
}
```

5.1. Definirea metodelor

- <modificatori_acces> - specifica domeniul de vizibilitate (folosire sau acces) al metodei; modificatorul de acces este optional si poate fi: ***public, protected, private;***

- <modificatori_metoda> - specifica proprietatile metodei si modul de lucru al acesteia; modificatorul este optional si poate fi: ***static, abstract, final;***

- <tip_returnat> - specifica unul din tipurile primitive, un nume de clasa sau cuvantul cheie *void* (cand metoda nu returneaza nici o valoare);

5.1. Definirea metodelor

- <nume_metoda> - specifica numele metodei; este de preferat ca numele metodei sa inceapa cu o litera mica si daca numele metodei contine in interior mai multe cuvinte, aceste cuvinte sa inceapa cu o litera majuscula;
- <param1>, <param2>, … - specifica lista de parametrii ai metodei, care reprezinta un set de definitii de variabile separate prin virgula;
- <clauze_specifice> - specifica anumite clauze *throws* care indica exceptiile (erorile) pe care le poate semnala metoda; despre aceste clauze vom vorbi intr-un curs viitor;
- <corpul_metodei> - instructiuni, apelari de metode, etc.

5.1. Definirea metodelor

Observatii:
1. Daca *o metoda returneaza o referinta la un tablou de valori sau de obiecte*, trebuie folosite parantezele drepte (**[ ]**) fie dupa <tip_returnat>, fie dupa lista de parametri.

2. In afara cazurilor cand este declarata cu tipul *void*, *o metoda returneaza la terminarea sa o valoare de un anumit tip*.
Aceasta valoare trebuie specificata explicit intr-o instructiune **return**.

3. *In aceeasi clasa pot exista metode cu acelasi nume si acelasi tip al valorii returnate, dar care difera prin numarul si tipul parametrilor din lista de parametri*.
Acest mecanism poarta denumirea de **supraîncărcarea (*overloading*) metodei**.

Apelul metodelor

Apelul unei metode definita intr-o clasa de obiecte se realizeaza in mai multe moduri:

- *prin crearea si utilizarea unei instante a clasei in care a fost definita metoda sau a unei subclase a clasei respective* (ca regula generala de apel a unei metode);

În acest caz se foloseste operatorul punct (.), in stânga acestuia punandu-se numele instantei, iar in dreapa acestuia punandu-se numele metodei;

- *prin simpla folosire a numelui sau, in cazul in care clasa in care este apelata metoda este aceeasi cu clasa in care a fost definita*;
aceasta modalitate este folosita daca atat metoda apelanta cat si metoda apelata sunt fie numai metode de instanta, fie numai metode de clasa;

- *prin folosirea operatorului punct (.),* in stanga acestuia punandu-se numele clasei in care a fost definita, iar in dreapta acestuia punandu-se numele metodei;
aceasta modalitate este folosita numai daca metoda este definita ca metoda de clasa (modificatorul *static*).

Exemplul 1

Programele urmatoare (ClasaTablou1.java si ClasaTablou2.java) prezinta un exemplu de creare a unei clase care defineste o metoda numita *creareTablou*.

Acesta preia doua numere naturale (o limita inferioara si una superioara) si creaza un tablou unidimensional care contine toate numerele naturale aflate intre cele doua limite, inclusiv aceste limite.

Varianta de *apel a unei metode prin crearea si utilizarea unei instante a clasei* in care a fost definita metoda:

```
public class ClasaTablou1
{
        int [] creareTablou(int inf, int sup)
        {
        int [] tabl = new int[(sup - inf) +1];
        for (int i = 0 ; i < tabl.length; i++)
                                        tabl[i] = inf++;
        return tabl;
        }
```

```
public static void main(String args[])
{
   ClasaTablou1 unTablou = new ClasaTablou1();
   int [] tablou = unTablou.creareTablou(1,10);
   System.out.print("Tabloul: [ ");
   for (int i = 0; i < tablou.length; i++)
        System.out.print(tablou[i] + " ");
   System.out.println("]");
}
}
```

Rezultatul executiei programului este:

Tabloul: [1 2 3 4 5 6 7 8 9 10]

Exemplul 2

Varianta de *apel a unei metode prin simpla folosire a numelui metodei*, deoarece metoda este definita si apelata in aceeasi clasa.

Totusi metoda *creareTablou* trebuie sa fie declarata ca metoda de clasa (modificatorul *static*) pentru a putea fi apelata dintr-o alta metoda de clasa.

```
public class ClasaTablou2 {
        static int [] creareTablou(int inf, int sup)
        {
          int [] tabl = new int[(sup - inf) +1];
          for (int i = 0 ; i < tabl.length; i++)
                  tabl[i] = inf++;
          return tabl;
        }
        public static void main(String args[])
        {
                int [] tablou = creareTablou(1,10);
                System.out.print("Tabloul: [ ");
                for (int i = 0; i < tablou.length; i++)
                System.out.print(tablou[i] + " ");
                System.out.println("]");
        }
}
```

Rezultatul executiei programului este:

Tabloul: [1 2 3 4 5 6 7 8 9 10]

Clase. Variabile. Domeniul de vizibilitate a variabilelor

4. Domeniul de vizibilitate (acces) al variabilelor folosite in clasele de obiecte:
   4.1. Domeniul de vizibilitate al variabilelor locale
   4.2. Domeniul de vizibilitate al variabilelor clasei
   4.3. Modificatorii de acces (vizibilitate)
5. Metodele unei clase de obiecte:
   5.1. Definirea si apelul metodelor
   5.2. Modificatorii de metoda
6. Metode de instanta si metode de clasa

5.2. Modificatorii de metodă

Modificatorii de metoda specifica proprietati suplimentare pentru o metoda.

In **Java** exista mai multi modificatori de metoda:

- modificatorul ***static*** - pentru metode statice de clasa
- modificatorul ***abstract*** - pentru metode abstracte, ce vor fi descrise într-un curs viitor
- modificatorul ***final*** - pentru metode finale, ce vor fi descrise într-un curs viitor

Clase. Variabile. Domeniul de vizibilitate a variabilelor

4. Domeniul de vizibilitate (acces) al variabilelor folosite in clasele de obiecte:
   4.1. Domeniul de vizibilitate al variabilelor locale
   4.2. Domeniul de vizibilitate al variabilelor clasei
   4.3. Modificatorii de acces (vizibilitate)
5. Metodele unei clase de obiecte:
   5.1. Definirea si apelul metodelor
   5.2. Modificatorii de metoda
6. Metode de instanta si metode de clasa

6. Metode de instanţă şi metode de clasă

Metode de instanţă

Ca regula generala, *o metoda definita într-o clasa se poate apela prin crearea unei instante a clasei respective sau a unei subclase a clasei respective*.

Aceasta se datoreaza faptului ca metoda lucreaza cu o serie de variabile ale clasei care sunt memorate in interiorul instantei si care au valori diferite in instante diferite (numite variabile de instanta).

6. Metode de instanţă şi metode de clasă

Astfel de metode se numesc **metode ale instantelor clasei**.

Metodele de instanta sunt aplicate unui anume obiect, nu unei clase intregi.

Majoritatea metodelor definite într-o clasa sunt metode de instanta.

6. Metode de instanţă şi metode de clasă

Dupa cum stim deja, exista si un alt tip de variabile si anume variabilele de clasa sau variabilele statice care sunt comune tuturor instantelor clasei respective.

Aceste variabile pot fi accesate fara a avea nevoie de o instanta a clasei in care au fost declarate.

6. Metode de instanţă şi metode de clasă

In mod similar exista si **metode de clasa sau metode statice**.

Pentru a fi apelate, aceste metode, definite intr-o clasa, nu au nevoie sa fie creata o instanta a clasei respective sau a subclasei derivata din clasa respectiva.

Metodele de clasa sunt disponibile oricarei instante a clasei.

Metodele de clasa nu folosesc variabilele de instanta, in schimb pot sa foloseasca variabilele de clasa (variabile statice) declarate in interiorul clasei in care au fost definite.

6. Metode de instanţă şi metode de clasă

Intr-o metoda de clasa se pot apela metode de instanta dar cu precalificarea acestora cu numele instantei.
Pentru a defini metode de clasa se foloseste modificatorul ***static***, pozitionat in fata definitiei metodei, *la fel ca in cazul declararii variabilelor de clasa*.

Ca si in cazul variabilelor de clasa, o metoda de clasa poate fi apelata:

- fie conform regulii generale prin precalificarea numelui metodei cu numele instantei (despartite de operatorul punct)
- fie direct, prin precalificarea numelui metodei cu numele clasei (despartite de operatorul punct)

6. Metode de instanţă şi metode de clasă

De exemplu, **Java** contine clase pentru fiecare dintre tipurile de baza: **Byte**, **Integer**, **Long**, **Float**, **Double**, **Boolean**, **Character**, **Short**.

Fiecare din aceste clase contin metode care se aplica oricarei instante a clasei respective.

De exemplu, putem folosi metodele de clasa care realizeaza conversia obiectelor in tipuri primitive si invers (*parseInt*, *parseFloat,* etc sau *toString*).

6. Metode de instanţă şi metode de clasă

Metodele de clasa pot fi de asemenea folositoare pentru adunarea intr-un singur loc (o clasa) a unor metode generale.

De exemplu, clasa **Math** contine *un set larg de operatii matematice definite cu metode de clasa* - nu exista instante ale clasei **Math**.

Clase. Variabile. Domeniul de vizibilitate a variabilelor

7. Domeniul de vizibilitate (acces) al metodelor unei clase:
    - 7.1. Modificatori de acces
    - 7.2. Referinta *this*
8. Metode constructor:
    - 8.1. Caracteristici
    - 8.2. Supraîncărcarea metodelor constructor
    - 8.3. Cuvântul-cheie *this* pentru constructori
9. Iniţializatori statici

7.1. Modificatori de acces

- *O metoda este accesibila (apelabila) daca este definita in clasa din care este apelata sau intr-una din subclasele acesteia*.
- Atunci cand se apeleaza metoda unui obiect, **Java** cauta definitia metodei respective in clasa obiectului.
- Daca nu o gaseste, cauta mai sus in ierarhia de clase pana cand gaseste o definitie.
- In acelasi timp pentru a “vedea” o metoda si pentru a o putea apela, este nevoie sa avem drepturile de acces necesare (date de modificatorii de acces).

7.1. Modificatori de acces

Modificatorii de acces (vizibilitate) ai variabilelor unei clase

In Java exista trei modificatori de vizibilitate ai variabilelor unei clase:

1. modificatorul **public**
2. modificatorul **protected**
3. modificatorul **private**

7.1. Modificatori de acces

1. **Modificatorul public** face ca *metoda respectiva sa fie accesibila oriunde este accesibila clasa din care face parte metoda*.
2. **Modificatorul protected** face ca *metoda respectiva sa fie accesibila in orice clasa din pachetul careia ii apartine clasa in care a fost definita*. In acelasi timp, *metoda este accesibila in toate subclasele clasei date, chiar daca ele apartin altor pachete*.
3. **Modificatorul private** face ca *metoda respectiva sa fie accesibila doar in interiorul clasei in care a fost definita*.

7.1. Modificatori de acces

Daca pentru o metoda a unei clase nu se precizeaza nici un modificator de acces din cei descrisi mai sus, atunci metoda respectiva devine **package-friendly**.

O metoda friendly este accesibila in pachetul din care face parte clasa in interiorul careia a fost definita, dar nu este accesibila in subclasele clasei date daca acestea apartin altor pachete.

7.1. Modificatori de acces

Nota:

Modificatorii de acces (*public*, *protected*, *private*) sunt plasati primii in definitia metodei, urmeaza apoi modificatorii care determina felul metodei (*static*, *abstract*, *final*) si apoi semnatura metodei.

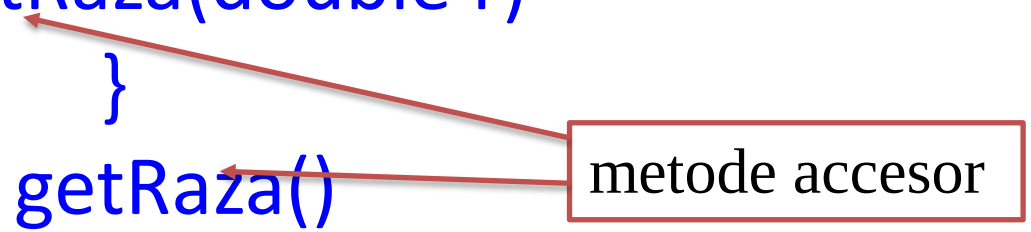
7.1. Modificatori de acces

Urmatorul program (TestCerc.java) ilustreaza modul de *folosire al variabilelor de instanta*, precum si al *metodelor de instanta*.

In clasa *Cerc* variabila de instanta este *raza* care este vizibila numai in clasa in care a fost declarata (are modificatorul *private*).

De aceea, accesul la aceasta variabila (pentru citire si modificare) se face numai prin intermediul metodelor *setRaza* si *getRaza* care sunt publice.

```
class Cerc
{
        private double raza;
        public void setRaza(double r)
        {       raza = r;     }
        public double getRaza()
        {       return raza;        }
        public double arie()
        {       return Math.PI * raza * raza;        }
        public double lungime()
        {       return 2 * Math.PI * raza;    }
}
```



```
public class TestCerc
{
        public static void main(String[] args) {
        Cerc cerculMeu = new Cerc();
        cerculMeu.setRaza(10);
        System.out.println("Raza=" +
cerculMeu.getRaza());
        System.out.println("Aria=" + cerculMeu.arie());
        System.out.println("Lungimea=" +
cerculMeu.lungime());
         }
 }
```

Clase. Variabile. Domeniul de vizibilitate a variabilelor

7. Domeniul de vizibilitate (acces) al metodelor unei clase:
    - 7.1. Modificatori de acces
    - 7.2. Referinta *this*
8. Metode constructor:
    - 8.1. Caracteristici
    - 8.2. Supraîncărcarea metodelor constructor
    - 8.3. Cuvântul-cheie *this* pentru constructori
9. Iniţializatori statici

7.2. Referinta *this*

Cuvantul-cheie **this** se refera la obiectul curent, adica obiectul a carei metoda a fost apelata.

Metoda poate folosi variabilele de instanta ale obiectului curent sau poate transmite obiectul curent ca parametru unei alte metode.

7.2. Referinta *this*

Exemple de folosire a cuvantului **this**:

t = this.x; // variabila de instanta x pentru acest obiect

this.resetRaza(this); // apeleaza metoda resetRaza, definita in clasa curenta si transmite obiectul curent

return this; // returneaza obiectul curent

7.2. Referinta *this*

In cele mai multe cazuri nu este nevoie sa se foloseasca explicit cuvantul-cheie **this**, deoarece este presupus.

De exemplu, ne putem referi atat la variabilele de instanta, cat si la apelurile de metode definite in clasa curenta prin simpla folosire a numelui lor, deoarece **this** este implicit folosit de aceste referinte.

7.2. Referinta *this*

De aceea, primele doua exemple se pot rescrie astfel:

t = x; // variabila de instanta x pentru acest obiect

resetRaza(this); // apeleaza metoda resetRaza, definita in clasa curenta

7.2. Referinta *this*

- Nu se omite cuvantul-cheie **this** daca in *domeniul de vizibilitate al obiectului curent au fost definite variabile locale cu acelasi nume ca cel al unei variabile de instanta* sau *au fost transmisi unei metode, a obiectului curent, parametrii cu acelasi nume ca cel al unei variabile de instanta*.
- Aceste aspecte au fost explicate la domeniul de vizibilitate al variabilelor clasei.
- *Nota*: Deoarece **this** este o referinta a instantei curente a clasei, trebuie sa se foloseasca doar in corpul unei definitii de metoda de instanta. *Metodele de clasa, declarate cu modificatorul* **static**, *nu pot folosi* **this**.

Clase. Variabile. Domeniul de vizibilitate a variabilelor

7. Domeniul de vizibilitate (acces) al metodelor unei clase:
    - 7.1. Modificatori de acces
    - 7.2. Referinta *this*
8. Metode constructor:
    - 8.1. Caracteristici
    - 8.2. Supraîncărcarea metodelor constructor
    - 8.3. Cuvântul-cheie *this* pentru constructori
9. Iniţializatori statici

8. Metode constructor

Pe langa metodele obisnuite, in clase se pot include si metode constructor.
O **metoda constructor** este o metoda apelata atunci cand obiectul este creat si initializat, folosind operatorul **new**.

Spre deosebire de alte metode, *o metoda constructor nu poate fi apelata direct in cadrul programului*;
Java apeleaza metodele constructor in mod automat.

8. Metode constructor

Atunci cand este folosit operatorul **new** pentru crearea unei instante a unei clase, **Java** executa trei activitati:

1. *aloca memorie pentru obiect*
2. *initializeaza variabilele de instanta ale obiectului fie la valorile initiale date de programator, fie la cele implicite* (*0* pentru numere, *null* pentru obiecte, *false* pentru valori booleene, si *'\0'* pentru caractere)
3. *apeleaza metodele constructor ale clasei*

8. Metode constructor

Daca la definirea clasei nu se furnizeaza nici un constructor, compilatorul creaza automat un constructor implicit care initializeaza fiecare membru al clasei cu valorile implicite.

Prin definirea unor metode constructor in clase:

- se pot seta valorile initiale ale variabilelor de instanta,
- se pot apela metode pe baza acestor variabile,
- se pot apela metode ale altor obiecte etc.

Clase. Variabile. Domeniul de vizibilitate a variabilelor

7. Domeniul de vizibilitate (acces) al metodelor unei clase:
    - 7.1. Modificatori de acces
    - 7.2. Referinta *this*
8. Metode constructor:
    - 8.1. Caracteristici
    - 8.2. Supraîncărcarea metodelor constructor
    - 8.3. Cuvântul-cheie *this* pentru constructori
9. Iniţializatori statici

8.1. Caracteristici

Metodele constructor au două caracteristici de bază:

1. *au întotdeauna acelaşi nume cu cel al clasei*
2. *nu returnează nici o valoare*

8.1. Caracteristici

Urmatorul program (TestCercCons.java) prezinta clasa *Cerc* care are trei variabile de instanta:

- *raza*
- si coordonatele centrului cercului, *x* si *y*

Clasa *Cerc* foloseste o metoda constructor pentru a-si initializa variabilele de instanta pe baza argumentelor primite de **new**.

8.1. Caracteristici


```
class Cerc
{
        private double raza;
        private int x, y;

        Cerc(int coordX, int coordY, double lungRaza)
        {
                x = coordX;
                y = coordY;
                raza = lungRaza;
        }
```

8.1. Caracteristici

```
public void setRaza(double r) {  raza = r;       }
public double getRaza()   {      return raza;  }
public int getX()     {      return x;      }
public int getY()     {      return y;      }

public double arie()
{       return Math.PI * raza * raza;     }
public double lungime()
{       return 2 * Math.PI * raza;        }
}
```

metode accesor

8.1. Caracteristici

```
public class TestCercCons
{
  public static void main(String[] args) {
          Cerc cerculMeu = new Cerc(3, 9, 20);
          System.out.println("Raza=" + cerculMeu.getRaza());
          System.out.println("Centrul cercului este in punctul: x= " +
cerculMeu.getX() + " y= " + cerculMeu.getY());
          System.out.println("Modificarea razei cercului");
          cerculMeu.setRaza(10);
          System.out.println("Raza=" + cerculMeu.getRaza());
          System.out.println("Aria=" + cerculMeu.arie());
          System.out.println("Lungimea=" + cerculMeu.lungime());
           }
 }
```

Clase. Variabile. Domeniul de vizibilitate a variabilelor

7. Domeniul de vizibilitate (acces) al metodelor unei clase:
    - 7.1. Modificatori de acces
    - 7.2. Referinta *this*
8. Metode constructor:
    - 8.1. Caracteristici
    - 8.2. Supraîncărcarea metodelor constructor
    - 8.3. Cuvântul-cheie *this* pentru constructori
9. Iniţializatori statici

8.2. Supraîncărcarea metodelor constructor

- Ca si metodele obisnuite, constructorii pot avea un numar diferit de parametri sau tipuri diferite pentru acestia desi au acelasi nume.
- Folosirea mai multor constructori cu acelasi nume dar cu parametrii care difera prin numar si/sau tip poarta denumirea de **supraincarcarea metodelor constructor**.
- Aceasta tehnica *ne permite sa cream un obiect cu proprietatile dorite sau ne da posibilitatea sa cream obiecte care sa isi seteze proprietatile pornind de la date de intrare diferite*.

Clase. Variabile. Domeniul de vizibilitate a variabilelor

7. Domeniul de vizibilitate (acces) al metodelor unei clase:
    - 7.1. Modificatori de acces
    - 7.2. Referinta *this*
8. Metode constructor:
    - 8.1. Caracteristici
    - 8.2. Supraîncărcarea metodelor constructor
    - 8.3. Cuvântul-cheie *this* pentru constructori
9. Iniţializatori statici

8.3. Cuvântul-cheie *this* pentru constructori

- Multe clase dispun de mai multi constructori care au un comportament similar.
- Putem folosi cuvantul-cheie **this** in cadrul unei metode constructor pentru a apela ceilalti constructori ai clasei.
- Apelul unei metode constructor definita in clasa curenta, folosind **this** se face astfel:

this(<arg1>, <arg2>, <arg3>)

unde:

- <arg1>, <arg2>, <arg3> - specifica parametrii metodei constructor.

Intotdeauna apelul lui **this** trebuie sa fie prima instructiune din metoda constructor, celelalte instructiuni urmand dupa aceasta.

8.3. Cuvântul-cheie *this* pentru constructori

Urmatorul program (TestCercCons.java) prezinta clasa *Cerc* care are trei variabile de instanta:

- *raza*
- si coordonatele centrului cercului, *x* si *y*

Clasa *Cerc* foloseste doua metode constructor:

unul in care sunt initializate variabilele de instanta pe baza datelor furnizate de parametrii lui **new**, si unul in care coordonatele *x* si *y* sunt preluate pe baza datelor furnizate de **new** dar variabila *raza* primeste valoarea prestabilita 1

8.3. Cuvântul-cheie *this* pentru constructori

```
class Cerc
{
        private double raza;
        private int x, y;
        Cerc(int coordX, int coordY, double lungRaza) {
                x = coordX;
                y = coordY;
                raza = lungRaza;
        }
        Cerc(int coordX, int coordY) {
                this(coordX, coordY, 1);
        }
```

metode de tip constructor

```
    public void setRaza(double r)
    {       raza = r; }
    public double getRaza()
    {       return raza; }
    public int getX()
    {       return x; }
    public int getY()
    {       return y; }
    public double arie()
    {       return Math.PI * raza * raza; }
    public double lungime()
    {       return 2 * Math.PI * raza;}
}
```

```
public class TestCercCons
{
        public static void main(String[] args) {
         System.out.println("Crearea obiectului cu
primul constructor");
         Cerc cerculMeu = new Cerc(3, 9, 20);
         System.out.println("Raza=" +
cerculMeu.getRaza());
         System.out.println("Centrul cercului este in
punctul: x= " +
         cerculMeu.getX() + " y= " + cerculMeu.getY());
```

```
System.out.println("Crearea obiectului cu al doilea
constructor");
        Cerc cerculMeu = new Cerc(3, 9);
        System.out.println("Raza=" + cerculMeu.getRaza());
        System.out.println("Centrul cercului este in punctul:
x= " + cerculMeu.getX() + " y= " + cerculMeu.getY());
System.out.println("Modificarea razei cercului");
        cerculMeu.setRaza(10);
        System.out.println("Raza=" + cerculMeu.getRaza());
        System.out.println("Aria=" + cerculMeu.arie());
        System.out.println("Lungimea=" +
cerculMeu.lungime());}
 }
```

Dupa executia programului pe ecran se afiseaza urmatoarele:

Crearea obiectului cu primul constructor
Raza=20.0
Centrul cercului este in punctul: x= 3 y= 9
Crearea obiectului cu al doilea constructor
Raza=1.0
Centrul cercului este in punctul: x= 3 y= 9
Modificarea razei cercului
Raza=10.0
Aria=314.1592653589793
Lungimea=62.83185307179586

Clase. Variabile. Domeniul de vizibilitate a variabilelor

7. Domeniul de vizibilitate (acces) al metodelor unei clase:
    - 7.1. Modificatori de acces
    - 7.2. Referinta *this*
8. Metode constructor:
    - 8.1. Caracteristici
    - 8.2. Supraîncărcarea metodelor constructor
    - 8.3. Cuvântul-cheie *this* pentru constructori
9. Iniţializatori statici

9. Iniţializatori statici

La încărcarea în memorie a unei clase (deci inainte de a fi creata prima instanta) sunt initializate automat toate variabilele statice declarate in interiorul clasei.
In plus, sunt apelati toti initializatorii statici ai clasei.

Un initializator static are urmatoarea sintaxa:

static {<set_instructiuni>}

9. Iniţializatori statici

Setul de instructiuni din initializatorul static este executat automat la incarcarea clasei in memorie.
De exemplu, putem defini un initializator static in felul urmator:

```
class A {
        static double a;
        static int b;

        static {
        a = Math.random(); // numar aleator intre 0.0 si 1.0
        b = (int) (a * 500);  // numar intreg intre 0 si 500
        }
        // restul clasei
}
```

variabile statice

initializator static

9. Iniţializatori statici

Declaratiile de variabile statice si initializatorii statici sunt executate in ordinea in care apar in clasa.

De exemplu:

```
class A {
        static int i  = 11;
        static {       i  += 100;
                       i %= 55;
        }
        static int j = i + 1;
        // restul clasei
}
```

- Valoarea finala a lui i va fi 1 ((11+100) % 55)
- iar valoarea lui j va fi 2

Întrebări?