Programare orientată pe obiecte

#9 JAVA
Obiecte și referințe

https://www.runceanu.ro/adrian/activitate-didactica/

Curs 9

Limbajul JAVA.
Obiecte și referințe

1. Terminologia de baza pentru programarea orientata obiect:
   1.1. Obiecte reale si reprezentarea lor in calculator
   1.2. Proprietati (atribute) specifice
   1.3. Set de operatii specifice

2. Despre obiecte si referinte in Java:
   2.1. Variabile de tip referinta
   2.2. Operatii care se pot aplica referintelor
   2.3. Declararea si crearea obiectelor
   2.4. Operatorul new
   2.5. Gestiunea memoriei si colectarea de gunoaie (garbage collection)
   2.6. Transmiterea de parametri referinta la obiecte catre metodele unui obiect

1. Terminologia de baza pentru programarea orientata obiect

Obiectele fizice din lumea reala sau notiunile sunt reprezentate in memoria calculatorului in asa fel incat informatiile specifice lor sa fie pastrate la un loc si sa se poata prelucra ca un tot unitar.

1. Pentru a reprezenta in memoria interna obiecte fizice sau notiuni, este necesar sa analizam intregul **set de proprietati (atribute) specifice** fiecaruia dintre acestea si sa il reprezentam prin numere intr-o zona compacta de memorie.

- Va trebui insa sa avem intotdeauna o imagine clara a deosebirii fundamentale dintre un obiect fizic sau o notiune si reprezentarea acestora in memoria calculatorului.
- De exemplu, *in memoria calculatorului este simplu sa cream un nou obiect, identic cu altul deja existent, prin simpla duplicare a zonei de memorie folosita de obiectul pe care dorim sa il duplicam*.
- In realitate insa, este mult mai greu sa obtinem o copie identica a unui obiect fizic.

2. Atunci cand analizam un obiect fizic sau o notiune pentru a le reprezenta in memoria calculatorului, trebuie sa analizam, nu numai proprietatile fiecaruia dintre acestea dar, si **setul de operatii specifice care pot fi executate asupra lor sau cu ajutorul lor**.

- Setul de operatii specifice unui obiect real sau unei notiuni va fi reprezentat in memoria calculatorului avand in vedere valorile care formeaza proprietatile obiectului.
- Apoi, acest set de operatii va fi descompus in operatii numerice predefinite in calculator.

De exemplu, dorim sa reprezentam in memoria calculatorului un obiect care sa descrie o *minge de forma sferica, de o anumita culoare aflata in spatiu*.

Pentru aceasta reprezentare, vom defini:

- ✓ o valoare pentru raza sferei
- ✓ o valoare pentru culoarea mingii
- ✓ si 3 valori care sa reprezinte coordonatele x, y si z relativ la un sistem de axe dat

aceste valori numerice vor face parte din setul de proprietati ale obiectului *minge*.

vom defini operatiile specifice obiectului *minge*, cum ar fi:

- ✓ *mutarea in spatiu a obiectului si umflarea obiectului*
- ✓ de exemplu, la definirea operatiei care reprezinta mutarea in spatiu a obiectului *minge* este suficient sa folosim operatiile predefinite cu numere pentru a modifica valorile coordonatelor x, y, z.

In concluzie, din punct de vedere al programarii calculatoarelor, un obiect este o reprezentare in memoria calculatorului:

1. a proprietatilor
2. a setului de operatii specifice unui obiect real sau unei notiuni

Observatie:

Descrierea proprietatilor si a setului de operatii specifice ale obiectului depind de problema de rezolvat.

In exemplul de mai sus, obiectul *minge* este insuficient descris si pentru a simula in calculator obiectul real este nevoie de proprietati suplimentare cum ar fi:

- materialul din care este confectionat mingea, etc,
- precum si de multe operatii in plus, cum ar fi:
  - aruncarea mingei
  - deformarea mingei
  - etc.

Daca problema de rezolvat nu necesita anumite proprietati si operatii ale unui obiect real, este preferabil sa nu le definim in obiectul folosit pentru reprezentarea in memoria calculatorului.

Cu alte cuvinte, vom defini acelasi obiect real in diferite feluri pentru a-l reprezenta in memoria interna, in functie de scopul problemei de rezolvat.

1. Terminologia de baza pentru programarea orientata obiect:
   1.1. Obiecte reale si reprezentarea lor in calculator
   1.2. Proprietati (atribute) specifice
   1.3. Set de operatii specifice

2. Despre obiecte si referinte in Java:
   2.1. Variabile de tip referinta
   2.2. Operatii care se pot aplica referintelor
   2.3. Declararea si crearea obiectelor
   2.4. Operatorul new
   2.5. Gestiunea memoriei si colectarea de gunoaie (garbage collection)
   2.6. Transmiterea de parametri referinta la obiecte catre metodele unui obiect

Categorii de operatii care se pot aplica variabilelor referinta

1. *Prima categorie de operatii, cu variabile referinta, permite examinarea si manipularea valorii referinta.*

În Java, singurele operatii care sunt permise asupra referintelor (cu o singura exceptie pentru *String*-uri) sunt:

1. atribuirea prin intermediul operatorului **=**
2. comparatia prin intermediul operatorilor **==** si **!=**

- De exemplu, prin atribuirea *nr2 = nr1* vom face ca *nr2* sa refere acelasi obiect pe care il refera *nr1*, iar expresia *nr1 == nr2* este adevarata, deoarece ambele referinte stocheaza aceeasi valoare a adresei de memorie la care se afla obiectul referit.

Observatie:

- Alte limbaje de programare, cum ar fi C, definesc notiunea de **pointer** care este **similara cu** cea a unei **variabile referinta**.
- *In limbajul C este permisa aritmetica pointerilor, pe cand in Java aceasta nu este permisa*.

2. A doua categorie de operatii, cu variabile referinta, se aplica obiectului care este referit.

Exista trei actiuni fundamentale care pot fi realizate:

1. aplicarea unei conversii explicite de tip
2. accesul la o variabila a obiectului sau apelul unei metode prin operatorul punct (.)
3. verificarea daca obiectul referit are un anumit tip cu ajutorul operatorului *instanceof*

- *In Java, un obiect este orice variabila care nu este de tip primitiv.*

Variabilele obiect sunt **manipulate prin referinte**

Variabilele de tipuri primitive sunt **manipulate prin valoare**

- Obiectul in sine este stocat undeva in memorie, iar *variabila referinta stocheaza adresa de memorie a obiectului*.

- Astfel, *variabila referinta devine un nume pentru acea zona de memorie*.

Operatorul punct (.)

- *Este folosit pentru a apela o metoda care se aplica unui obiect.*

- De exemplu, sa presupunem ca avem un obiect ce apartine clasei *Cerc* (adica de tip *Cerc*) care defineste metoda *arie*.
- Daca variabila *cerculMeu* este o referinta catre un obiect de tip *Cerc*, atunci putem calcula aria cercului referit prin apelul metodei *arie* astfel:

```
Cerc cerculMeu;
Double arieCerc = cerculMeu.arie();
```

- *Operatorul punct poate fi folosit si pentru a accesa atributele individuale ale unui obiect, daca cel care a proiectat obiectul permite acest lucru.*

Declararea si crearea obiectelor

Declararea si crearea (initializarea) obiectelor se realizeaza in doi pasi de catre compilatorul Java:

1. *Se declara o variabila referinta la obiect (instanta a unei clase).*

- Prin declaratia unei referinte se aloca zona de memorie necesara stocarii referintei in sine (adica a unei adrese de memorie).
- In consecinta, dupa ce se declara o variabila referinta, aceasta va contine valoarea *null*, ceea ce inseamna ca referinta inca nu indica o instanta valida.

Sintaxa folosita pentru **declararea unei variabile referinta catre un obiect** este:

<nume_clasa> <nume_obiect>;

unde:

- <nume_clasa> - *reprezinta numele clasei dupa modelul careia se doreste crearea unei instante*;
- <nume_obiect> - *reprezinta numele unei variabile referinta la un obiect oarecare de tipul <nume_clasa>; acesta variabila referinta la un obiect contine valoarea null*.

2. Se aloca, efectiv, zona de memorie pentru obiectul declarat la pasul 1 si se initializeaza obiectul cu ajutorul operatorului **new**.

In acest fel, se creaza o noua instanță a clasei date.

Sintaxa folosita pentru **alocarea memoriei si crearea efectiva a unei instante a clasei** este:

```
<nume_obiect> = new <nume_constructor> ();
```

sau

```
<nume_obiect> = new <nume_constructor> (arg1, arg2, …, argn );
```

unde:

- <nume_obiect> - reprezinta numele variabilei referinta declarata la pasul 1;

- <nume_constructor> () sau <nume_constructor> (arg1, arg2, …) - metoda speciala a clasei obiectului, numita *constructor,* care are acelasi nume cu cel al clasei obiectului urmat de paranteze rotunde vide sau care pot contine argumente; numarul si tipul argumentelor folosite sunt definite de clasa de obiecte.

Nota:

- Parantezele rotunde plasate dupa <nume_constructor> sunt importante si ele nu trebuie omise chiar daca sunt fara argumente.

- Daca se foloseste prima forma a operatorului **new**, in care *constructorul* este urmat de paranteze goale, este creat un obiect simplu fara a se initializa anumite valori ale variabilelor de instanta; de regula acest tip de constructor este implicit.

Daca se foloseste a doua forma a operatorului **new**, in care *constructorul* este urmat de paranteze care contin argumente, atunci aceste argumente determina valorile initiale ale variabilelor de instanta sau ale altor variabile folosite de obiectul astfel creat.

Nota:

- De cele mai multe ori programatorii combina declararea si crearea efectiva a obiectului intr-o singura instructiune, astfel:

```
<nume_clasa> <nume_obiect> = new
<nume_constructor>();
```

sau

```
<nume_clasa> <nume_obiect> = new
<nume_constructor>(arg1, arg2, …);
```

Exemple in care *constructorul* este folosit fara argumente:

1.
```
Random nrAleatoare;
nrAleatoare = new Random();
```
sau
```
Random nrAleatoare = new Random();
```

2.
```
Copil c;
c = new Copil();
```
sau
```
Copil c = new Copil();
```

Urmatoarele exemple prezinta modalitati de crearea a unor obiecte cu anumite valori initiale, adica folosind argumente in metoda speciala numita *constructor*:

1.
Punct pt = new Punct(0,0);

Obiectul de tip *Punct* cu numele *pt* este construit cu doua argumente pentru coordonatele initiale ale punctului.

2.
Cerc cerculMeu = new Cerc (0, 0, 10);
Obiectul de tip *Cerc* cu numele *cerculMeu* este construit cu trei argumente, doua pentru coordonatele initiale ale centrului si unul pentru lungimea razei.

Un alt exemplu de creare a mai multor tipuri de obiecte care folosesc diferite tipuri de argumente impreuna cu operatorul **new** si metoda speciala a clasei numita ***constructor***:

- **clasa Random**, care face parte din *pachetul java.util*, creaza obiecte folosite pentru generarea de numere aleatoare intr-un program.

- Aceste obiecte sunt numite *generatoare de numere aleatoare* si au valori cuprinse in intervalul [0, 1).

- De fapt, un obiect de tip **Random** extrage un numar dintr-o multime foarte mare de numere.

- Aceasta tehnica este denumita *generare de numere pseudo-aleatoare* si este folosita in diferite limbaje de programare.

- Pentru a extrage un numar diferit din secventa de numere aleatoare, obiectul de tip **Random** trebuie sa primeasca o valoare initiala denumita “sămânță” (“seed”).

- Aceasta “sămânță” poate fi transmisa obiectului la construirea sa.

Programul urmator (NrAleator.java) creaza obiecte de tip **Random** folosind operatorul **new** in doua moduri cu si fara argumente ale ***constructorului*.**

```
import java.util.*;
class NrAleator
{
   public static void main(String args[ ])
        {
           Random r1, r2;
```

```
r1 = new Random();
System.out.println("Valoarea  aleatoare 1: " +
    r1.nextDouble());          // valoarea afisata se schimba
int numar1 = (int) (r1.nextDouble() * 11);
System.out.println("Intreg aleator 1 in intervalul 0 si
    10: " + numar1);
 r2 = new Random(8600000);
System.out.println("Valoarea  aleatoare 2: " +
    r2.nextDouble());          // valoarea afisata nu se schimba
int numar2 = (int) (r2.nextDouble() * 11);
System.out.println("Intreg aleator 2 in intervalul 0 si
    10: " + numar2);
    }
}
```

Dupa executia programului, se afiseaza:

Problems @ Javadoc Declaration Console

```
<terminated> NrAleator [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (04.11.2015, 06:36:53)
Valoarea  aleatoare 1: 0.48501282808015755
Intreg aleator 1 in intervalul 0 si 10: 2
Valoarea  aleatoare 2: 0.7258826590374842
Intreg aleator 2 in intervalul 0 si 10: 9
```

Problems @ Javadoc Declaration Console

```
<terminated> NrAleator [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (04.11.2015, 06:37:30)
Valoarea  aleatoare 1: 0.14431768790079136
Intreg aleator 1 in intervalul 0 si 10: 4
Valoarea  aleatoare 2: 0.7258826590374842
Intreg aleator 2 in intervalul 0 si 10: 9
```

Problems @ Javadoc Declaration Console

```
<terminated> NrAleator [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (04.11.2015, 06:38:00)
Valoarea  aleatoare 1: 0.38102280429577884
Intreg aleator 1 in intervalul 0 si 10: 7
Valoarea  aleatoare 2: 0.7258826590374842
Intreg aleator 2 in intervalul 0 si 10: 9
```

- In acest exemplu doua obiecte **Random** diferite sunt create folosind argumente diferite pentru *constructorul* clasei Random introdus dupa operatorul **new**.
- Primul obiect, cu numele *r1*, a fost creat cu **new Random()** fara argumente, deci acest obiect are ca “sămânța” ceasul calculatorului (ora curenta).
- Apelarea metodei **nextDouble()** a obiectului de tip **Random** produce extragerea urmatoarei valori din secventa de numere pseudo-aleatoare.
- Valoarea obtinuta prin folosirea metodei **nextDouble()** difera de la o executie la alta a programului.

- Al doilea, cu numele *r2*, a fost creat cu **new Random(8600000)** deci, cu un argument dat de un numar intreg.

- De aceea, numarul aleator obtinut prin folosirea metodei **nextDouble()** ramane intotdeauna acelasi, indiferent de cate ori se executa programul.

- Acest lucru poate fi folositor pentru crearea unor date de test pentru programe.

Nota:

In programul de mai sus s-au mai introdus doua instructiuni care inmultesc un numar aleator cu 11 si stocheaza produsul ca intreg:

```
int numar1 = (int) (r1.nextDouble() * 11);
int numar2 = (int) (r2.nextDouble() * 11);
```

Intregul continut de variabilele *numar1* si *numar2* va fi un numar aleator cuprins intre 0 si 10.

Rolul operatorului new

La folosirea operatorului **new** se executa urmatoarele:

1. *se creaza o noua instanta a clasei date*
2. *se aloca memorie pentru aceasta instanta*
3. *se apeleaza o metoda speciala a clasei numita constructor*

- *Constructorii* reprezinta *metode speciale pentru crearea si initializarea noilor instante ale claselor*.

- Constructorii:
  1. initializeaza noul obiect si variabilele sale
  2. creeaza orice alte obiecte de care are nevoie obiectul creat
  3. realizeaza orice alte operatii de care obiectul are nevoie la initializarea sa
- Intr-o clasa pot exista mai multe definitii de *constructori*, fiecare avand un numar diferit de argumente sau de tipuri.

- Când se foloseste operatorul **new**, se pot specifica diferite argumente in lista de argumente si va fi apelat *constructorul* corespunzator pentru acele argumente.

- Intr-o clasa pot fi definiți oricâți constructori se doresc pentru a implementa comportamentul clasei.

Gestiunea memoriei și colectarea de gunoaie (garbage collection)

Gestiunea memoriei in Java se face dinamic si automat.

- Atunci cand se creaza un obiect nou, Java aloca automat o zona de memorie de dimensiunea corespunzatoare obiectului.
- Nu trebuie sa se aloce explicit memorie pentru obiecte.
- Deoarece gestiunea memoriei in Java se face automat, nu este nevoie sa dezalocam explicit memoria ocupata de obiect atunci cand am terminat de lucru cu aceasta.
- In Java *cand un obiect din memorie nu mai este referit de nici o variabila, memoria pe care o consuma va fi eliberata automat*.
- Aceasta tehnica se numeste *colectare de gunoaie*.

Interpretarea operatorului de atribuire (=) pentru referinte

- Daca x si y sunt referinte (de tipuri compatibile) atunci instructiunea

x = y

- inseamna ca valoarea adresei stocata in *y* este transferata in variabila referinta *x*.
- Rezulta ca, dupa operatia de atribuire, *x* va referi acelasi obiect ca si *y*.
- *Ceea ce se copiaza in acest caz sunt adrese.*
- Obiectul pe care *x* il referea inainte de operatia de atribuire nu mai este referit de *x* dupa operatia de atribuire.
- Daca *x* a fost singura referinta catre acel obiect, atunci obiectul respectiv nu mai este referit de nici o variabila si este disponibil pentru colectarea de gunoaie.

Nota:

Sa retinem faptul ca, *obiectele nu se copiaza prin operatorul de atribuire (=).*

De exemplu, secventa de instructiuni de mai jos:

```
Cerc cerc1 = new Cerc (0,0,10); // un cerc de raza 10
Cerc cerc2 = cerc1;
```

Ne spune ca s-a construit un singur obiect de tip *Cerc*, cu numele *cerc1* (prima instructiune) si ca *cerc2* este un alt nume dat pentru *cerc1* (a doua instructiune).

Transmiterea de parametri referinta la obiecte catre metodele unui obiect

- În Java transmiterea parametrilor se face prin valoare.
- Datorita acestui fapt, parametrii actuali (de apel) se transpun in parametri formali, din definitia metodei, folosind atribuirea obisnuita.
- Daca parametrul transmis este un tip referinta, atunci se stie deja ca, prin atribuire, atat parametrul formal, cat si parametrul de apel vor referi acelasi obiect.
- Orice metoda aplicata obiectului referit prin parametrul formal este, implicit, aplicata si obiectului referit prin parametrului de apel.

De exemplu, sa presupunem ca dorim sa modificam raza unui cerc folosind o metoda denumita *modifica()*.

Secventa de cod care apeleaza metoda *modifica()* este:

```
int v = 4;
Cerc cerc = new Cerc();
modifica(cerc, v);
```

Metoda *modifica()* de mai jos primeste ca parametri o referinta la un obiect de tip *Cerc* si o valoare intreaga:

```
public void  modifica (Cerc cerculMeu, int val)
{
   cerculMeu.setRaza (val);   // modifica raza obiectului
   apelant
   val += 4;      // nu are nici un efect asupra parametrului
   actual
}
```

Observatie:

- Metoda *setRaza()* modifica raza unui cerc oarecare si codul-sursa al acesteia nu este necesar a fi prezentat in acest context.

- Secventa de cod care apeleaza metoda *modifica()* va avea ca efect modificarea razei obiectului *cerc* la *4*, deoarece atat variabila referinta *cerc* cat si variabila referinta *cerculMeu* indica acelasi obiect, dar valoarea lui *v* va ramane nemodificata.

Interpretarea operatorului de egalitate (==) pentru referinte

Doua variabile referinta sunt egale via **==** daca ele refera acelasi obiect (sau ambele sunt valori *null*).

De exemplu:

```
Cerc cerc1 = new Cerc(0, 0, 15);  //un cerc de raza 15
Cerc cerc2 = new Cerc(0, 0, 15);  //un cerc de raza 15
Cerc cerc3 = cerc2;
```

In acest caz avem doua obiecte: un obiect cu numele *cerc1* si un obiect cu doua nume *cerc2* si *cerc3*.

Expresia `cerc2 == cerc3` este adevarata.

Expresia `cerc1 == cerc2` este falsa, desi variabila *cerc1* si variabila *cerc2* refera obiecte care au valori egale (ambele sunt cercuri cu centrul in origine si raza 15).

Un exemplu simplu de aplicatie Java care foloseste variabile si metode de instanta

Programul urmator (Copil.java) ilustreaza cum sunt definite variabilele si metodele de instanta intr-o clasa numita *Copil*, cum este folosit operatorul **new** pentru crearea unei instante a clase *Copil* cu numele *c* si cum sunt apelate metodele de instanta in cadrul aplicatiei.
Variabilele de instanta sunt definite astfel:

```
String culoare_piele;
String sex;
boolean flamand;
```

- Doua dintre variabile care desemneaza culoarea pielii si sexul contin obiecte de tip *String* (sir).
- Un obiect *String* in Java este creat folosind una din clasele standard din biblioteca de clase Java.
- Clasa *String* este folosita pentru pastrarea textului si pentru diferite functii de manipulare a textului.
- A treia variabila desemneaza starea de flamand a copilului si pastreaza doua valori: *true* (pentru flamand) si *false* (pentru hranit).
- Comportamentul clasei *Copil* poate fi dat de o multitudine de metode care sa descrie actiuni pe care le-ar putea face un copil (se hraneste, se joaca, se imbraca etc).

Totusi, in program sunt definite doar doua metode:

1. una pentru a hrani copilul (denumita *hranescCopil*)
2. una pentru a verifica si afisa atributele (proprietatile) copilului (denumita *afisezAtributeCopil*)

Iata codul sursa al aplicatiei:

```
class Copil
{
    String culoare_piele;
    String sex;
    boolean flamand;
    void hranescCopil()
    {
       if (flamand==true)
          {System.out.println("Bun - lapte");
                        flamand = false;                }
       else
           System.out.println("Nu, multumesc - am mancat deja");
    }
```

```
void afisezAtributeCopil()
{
      System.out.println("Acesta este un copil de sex " +sex+" si " +
                                                    culoare_piele +" .");
      if (flamand == true)    System.out.println("Copilul este flamand.");
        else    System.out.println("Copilul este satul");
}
public static void main(String args[ ]) {
          Copil c = new Copil();
          c.culoare_piele = "alb";
          c.sex = "masculin";
          c.flamand = true;
          System.out.println("Atribute copil");
          c.afisezAtributeCopil();
          System.out.println("Hranesc copilul");
          c.hranescCopil();
          System.out.println("Atribute copil");
          c.afisezAtributeCopil();
          System.out.println("Hranesc copilul");
          c.hranescCopil();
     }
}
```

Întrebări?