C7

Clase şi obiecte

- Paradigme de programare
- Programarea orientata pe obiect, avantaje si principii
- Clasa, obiectul, definitii, creare si utilizare
- Incapsulare
- Modelare OO

Obiective

Dupa parcurgerea acestui curs ar trebuie sa puteti:

- ☐ intelege conceptul de paradigma orientata pe obiect
- ☐ descrie avantajele si principiile POO
- ☐ intelege si opera cu clase si obiecte
- ☐ intelege modul de abordare a scrierii unei aplicatii orientate pe obiect prin prisma

Ce este programarea orientată pe obiect (POO)

- este o metodologie (paradigmă) de proiectare a programelor
- este inspirată din realitate şi se bazează pe conceptul de modelare pe obiect
- trecerea de la realitate la model se face prin abstractizare
- câteva caracteristici:
  - ☐ asigură reutilizarea şi adaptarea simplă a codului;
  - ☐ conduce la un cod mai puţin vulnerabil la manipulări eronate.

Programarea orientată pe obiect (object oriented programming, în engleză) este o metodologie de scriere a programelor. În programare, modul de abordare a scrierii aplicaţiei, adică metoda, mai poartă şi denumirea de paradigmă (vine din grecescul "paradeigma" care înseamnă "a demonstra"). Fiecare paradigmă a fost pusă la punct cu un scop precis, câteva dintre caracteristicile celei obiectuale sunt:

- reutilizarea codului;
- întreţinere uşoară;
- adaptarea codului cât mai simplă la situaţii noi;
- minimizarea vulnerabiltăţii codului la utilizarea greşită din partea programatorilor care în refolosesc.

Într-o descriere foarte generală, în programarea orientată pe obiect, aplicația de realizat este privită în termenii manipulării unor entități numite obiecte şi nu prin prisma acțiunilor care trebuiesc implementate (cum se face în programarea structurată).

POO are drept sursă lumea înconjurătoare. Omul, în viaţa de toate zilele, îşi propune să fie cât mai concret în descrierea lumii care îl înconjoară. El, dă nume scurte obiectelor înconjurătoare pentru ca să le poată identifica cât mai exact. De exemplu, spunem LOGAN, în loc de automobilul produs la Uzina de Automobile din Piteşti, în colaborare cu Renault. Toate aceste nume particulare ascund însă în spatele lor denumiri mai generale cum sunt: automobile, atomi, stele, oceane, oameni etc. Deseori, un obiect are la bază altele. Trăim, astfel, într-o lume orientată pe obiect. O descriere a termenului de obiect este dată de Booch prin "Un obiect are stare, comportament şi identitate; structura şi comportamentul similar al obiectelor este definit prin clasele lor comune; termenii de instanţă şi obiect sunt echivalenţi". Mai general, un obiect este ceva căruia i se poate aplica un concept. Conceptul este o idee sau o notaţie comună, mai multor obiecte înconjurătoare.

Obiectul, din viaţa de toate zilele, în programarea orientată pe obiect este implementat ca o structură de dată (abstractă) încapsulată cu un grup de subrutine denumite metode, care acţionează asupra

datelor. Structura obiectului rezultă ca urmare a abstractizării. Abstractizarea implică o generalizare, prin ignorarea sau ascunderea detaliilor, în vederea separării unor proprietăţi sau caracteristici ale unei entităţi de obiectul fizic sau de conceptul pe care îl descrie. Ca urmare, în programarea orientată pe obiect, căutarea şi definirea proprietăţilor obiectelor din lumea reală definesc specificul acestei paradigme.

Proiectarea orientată pe obiect conduce la o aplicație formată din colecții de obiecte care cooperează. Între obiectele aplicaţiei se stabilesc relaţii de legatură, iar cooperarea între obiecte se face prin mesaje transmise între acestea.

Etapele tipice acestei paradigme de proiectare sunt: identificarea claselor și a obiectelor, identificarea semanticii acestora, identificarea relațiilor și descrierea interfețelor și a implementărilor de clase și de obiecte.

Câteva nume care au avut contribuţii importante în crearea şi dervoltarea POO sunt:

- **Grady Booch**, de la Rational, care a dezvoltat renumitul instrument de modelare numit "*Rose*";
- **Kirsten Nygaard** şi **Ole-Johan Dahl**, inventatorii limbajului *Simula*, primul limbaj de programare orientat pe obiect şi inventatorii proiectării orientate pe obiect;
- **Adelle Goldberg**, **Kay Alan** şi **Dan Ingallas** fondatorii limbajului obiectual *Smalltalk* şi coautori ai lui "*Smalltalk*-80";
- **Brax Cox**, creatorul limbajului *Objective-C*, care implementează în limbajul C facilităţile legate de identificarea obiectelor şi mecanismul de mesaje din Smalltalk;
- **Meyer Bertrand** creatorul limbajului *Eiffel*;
- **Bjarne Stroustrup**, inventatorul limbajului *C++*, o extensie a limbajului C, probabil, unul dintre cele mai populare limbaje de programare orientate pe obiect.

Conceptul de obiect - definiţii

- depinde de contextul în care se lucrează:
  - **filozofic**: o entitate ce se poate recunoaşte ca fiind distinctă de altele - are identitae proprie
  - **proiectarea oo**: o abstractizare a unui obiect din lumea reală
  - **teoria tipurilor**: o structură de date împreună cu funcţiile de prelucrare a lor
- se defineşte prin atribute şi operaţii
- atributele definesc starea, operaţiile permit modificarea stării.

Din punctul de vedere al aplicaţie din care face parte obiectul el trebuie să asigură o parte din funcţiile necesare funcţionării întregii aplicaţii. Termenul de **modelare a obiectelor** se foloseşte pentru a descrie procedura de căutare a obiectelor prin care să descriem problema de rezolvat.

Obiectele sunt descrise prin **atribute** şi **operaţii**. **Atributele** sunt caracteristici care se modifică, iar **operaţiile** sunt acţiuni pe care obiectul le poate face. De exemplu, pisica are culoare, rasă şi masă (greutate), acestea ar fi câteva dintre atribute ei, ea pot prinde şoareci, mânca, dormi sau mieuna, acestea ar fi câteva dintre acţiunile specifice pisicii.

Într-un **model de obiecte** toate datele sunt stocate sub formă de atribute ale obiectelor. Atributele unui obiect vor putea fi manipulate prin acţiuni sau operaţii. Singura modalitate de modificare a atributelor este prin utilizarea operaţiilor. Atributele pot fi uneori obiecte. Funcţionarea la nivelul unui model de obiecte este definită prin operaţii. Un obiect poate accesa şi utiliza operaţiile unui al obiect. Operaţiile sunt cele ce modifică starea obiectului. Modelarea orientată pe obiect este despre găsirea obiectelor şi a dependenţelor între acestea. Dependenţele între obiecte reprezintă modul în care ele se aranjează, în sensul legăturilor - a relaţiilor, a asocieriilor - care se formează, pentru a ne soluţiona problema. De regulă, indentificarea obiectelor unui model de obiecte se face pe bază de substantive şi verbe. Substativele vor fi obiecte, iar verbele operaţii.

Relaţii între obiecte

Agregarea, cunoscută în literatura de specialitate sub prescurtarea de relaţie “has-a”, apare atunci când un obiect este compus din mai multe sub-obiecte. Comportamentul obiectului complex este definit prin comportamentul părţilor componente, distincte şi aflate în interacţiune. În procesul de descompunere al unui obiect în obiecte mai simple, acestea, deseori, vor putea fi reutilizate în alte aplicaţii. Agregarea ne permite deci, reutilizarea componentelor unei aplicaţii într-o altă aplicaţie.

Delegarea, cunoscută în literatura de specialitate sub prescurtarea de relaţie “uses-a”, apare atunci când un obiect, format total sau parţial din alte obiecte, lasă obiectele componente să îi definească comportamentul. Obiectul nu are un comportament implicit, la nivel de interacţiune cu alte obiecte, acesta fiind moştenit de la sub-obiectele lui.

Conceptul de clasă

- clasa este o generalizare a unei mulţimi de obiecte
- definiţia unei clase se face prin specificarea:
  - ☐ variabilelelor care îi definesc starea - variabile de instanţă;
  - ☐ metodele prin care se modifică starea - metode de instanţă;
  - ☐ relaţiile cu alte clase.

```
class nume_clasa extends nume_parinte {
//STARE - ATRIBUTE
    tip v_1;
    . . .
    tip v_n;
//METODE - OPERATII
    tip metoda_1(lista parametri) {
        corp_1
    }
    . . .
    tip metoda_k(lista parametri) {
        corp_k
    }
} //TERMINARE definitie de clasa
```

Clasa reprezintă un termen general asociat procesului de clasificare. Scopul clasificării este cel de încadrare a unui obiect într-o colecţie de obiecte similare ce poartă denumirea de clasă. Toate obiectele clasei au aceleaşi atribute şi fac aceleaşi operaţii, dar starea lor diferă şi au o identitate proprie. Un obiect particulat poartă denumirea de instantă. Din punctul de vedere al programatorului clasa reprezintă un nou tip de date. Clasa este o definiţie statică, prin care descriem un grup de obiecte. Clasa este un şablon (un tip), un concept, în timp de obiectele există în realitate (în timpul rulării programului). De exemplu, `String` reprezintă o clasă, iar variabilele de tipul `String` reprezintă obiecte ale aplicaţiei. Clasa `String` este una singură, în timp ce numărul obiectelor de tipul `String` nu este limitat. În procesul de proiectare a claselor apare situaţia în care mai multe clase distincte au părţi comune. Părţile comune pot fi cuprinse într-o singură clasă şi moştenite la nivelul unor clase ulterioare. **Moştenirea**, cunoscută în literatura de specialitate sub prescurtarea de relaţie “is-a”, apare atunci când o clasă are ca părinte o altă clasă. Prin moştenire, noua clasă preia toate caracteristicile clasei moştenite, la care mai poate adăuga unele noi. Astfel, noua clasă o extinde pe cea moştenită, fiind o specializare a acesteia. Moştenirea creşte claritatea proiectului şi productivitatea, deoarece noi clase pot fi create pe baza unora existente.

Definiţia unei clase presupune descrierea precisă a formei şi a naturii acesteia. Pentru aceasta trebuie specificate datele (atributele) pe care le conţine şi codul (operaţiile) care va opera cu respectivele date.

Datele, se vor stoca în variabile ce poartă denumirea de **variabile** de instanţă, iar operaţiile se implementează în **metode**. Variabilele şi metodele unei clase poartă denumirea comună de **membri**. Variabilele de instanţă au această denumire deoarece sunt proprii fiecărei instaţieri de clasă, adică fiecărui obiect din clasa respectivă. Datele unui obiect sunt distincte, separate de datele unui alt obiect din aceeaşi clasă.

Exemplul alăturat defineşte în clasa Punct:

- două variabile de instanţă:
```
private double x;
private double y;
```
- doi constructori:
```
Punct()
Punct(double abscisa ...)
```
- şase metode:
```
public void setX(double abscisa)
public void setY(double ordonata)
public double x()
public double y()
public double distantaOrigine()
public String toString()
```

Clasa `Grafica`, are metoda `main()`, este punctul de plecare al aplicaţiei şi defineşte două obiecte, P1 şi P2, din clasa `Punct`.

Aici se prezintă:

- modul în care se **declară obiectele** unei clase:
```
Punct p1;
```
- modul în care se **alocă spaţiu unui obiect**, prin folosirea operatorului `new`:
```
p1 = new Punct();
```
- modul în care **se referă o metodă publică** dintr-un obiect:
```
p1.setX(12);
```

```
      public class Punct {
/*Atributele clasei,
permit stocarea STARII */
    private double x;
    private double y;

/*Constructorii clasei,
permit initializarea obiectelor din
clasa Punct */

  Punct() {
      setX(0);
      setY(0);
  }

  Punct(double abscisa, double ordonata)
{
      setX(abscisa);
      setY(ordonata);
  }

/*Metodele clasei,
permit accesul si modificarea starii
obiectelor din clasa Punct */

  public void setX(double abscisa) {
      x = abscisa;
  }

  public void setY(double ordonata) {
      y = ordonata;
  }

  public double x() {
      return x;
  }

  public double y() {
          return y;
  }

  public double distantaOrigine() {
      return Math.sqrt(x*x+y*y);
  }

   public String toString() {
      return "<" + x + "," + y + ">";
  }
}

public class Grafica {
 public static void main(String[] args){
   Punct p1;
   Punct p2 = new Punct(-1.,7.);
   p1 = new Punct();
   System.out.println("p1 = " + p1);
   System.out.println("p2 = " + p2);
   p1.setX(12);
   p2.setY(13.345);
   System.out.println("p1 = " +
p1.toString());
  System.out.println("p2 = " + p2);
 }
}
```

Crearea obiectelor

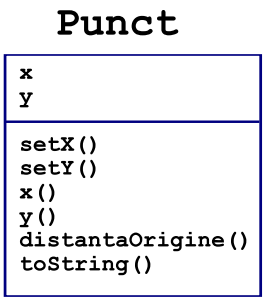
- **obiectele se creează cu operatorul `new`**

```
refObiect = new nume_clasa();
```

- **new realizează următoarele acțiuni:**
  - ☐ **alocă memorie pentru nou obiect;**
  - ☐ **apelează o metodă specială de inițializare numită constructor;**
  - ☐ **întoarce o referință la noul obiect**.

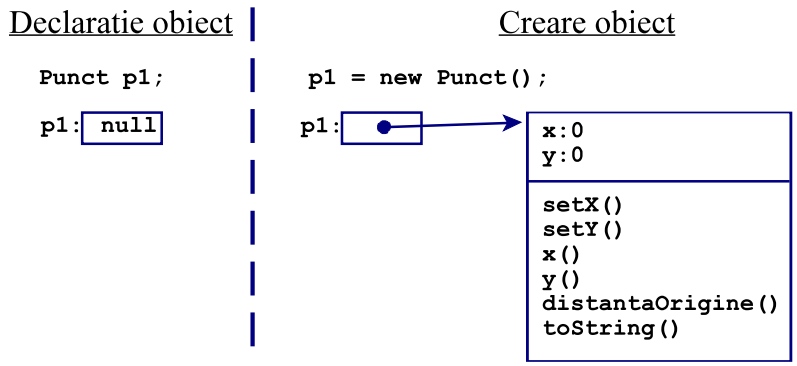
În Java obiectele se creează cu operatorul `new`. În aplicația anterioară se crează două instanțe ale clasei `Punct` prin liniile de cod:

```
Punct p2 = new Punct(-1,7);
p1 = new Punct();
```



Prima linie declară şi creează variabila obiect `p2`, iar cea de a doua creează variabila `p1`, ea fiind deja declarată anterior. Operatorul new are ca efect alocarea de spațiu în RAM pentru obiect şi are ca rezultat o referință către obiectul creat. Respectiva referință se va stoca în variabila obiect, mai sus `p1` şi `p2`, prin care vom putea accesa toți membrii obiectului în cauză.

Reprezentarea grafică a clasei `Punct` se poate da sub forma alăturată. Pe baza acestei clase vor crea două variabile obiect sau instanțe, `p1` şi `p2`. Spațiul de date pentru instanța `p1` va fi reprezentat grafic pentru a descrie etapele parcurse la crearea unui obiect. În momentul declarării obiectului `p1` prin linia `Punct p1;` în RAM se alocă spațiu în RAM pentru o referință căte un obiect. Spațiul de date alocat în RAM constă într-o locație cu numele `p1` în care este stocată o valoarea specială `null`. Această valoarea specială arată faptul că obiectului încă nu i s-a alocat spațiu. În etapa următoare, operatorul `new` alocă spațiu pentru obiect pe baza definiției de clasă, apelează constructorul clasei pentru a inițializa variabilele de instantă `x` şi `y` cu valorile 0 şi 0, apoi întoarce o referință (o adresă de RAM ce se reprezintă grafic ca o săgeată cu vârful de la instanță către spațiul alocat obiectului) către nou obiect ce va fi stocată în



variabila obiect `p1`.

Încapsularea (encapsulation) şi vizibilitatea membrilor

- **vizibilitatea membrilor este controlată prin specificatorii de acces:**
  - `public`
  - `private`
  - `protected`
- **încapsularea asigură:**
  - **protecția datelor prin accesul la acestea numai prin intermediul unei intefețe publice (`public`)**
  - **izolarea părților vizibile prin interfață de implementarea internă (`private`)**
- **avantajele încapsulării:**
  - **nu pot fi realizate modificări ilegale ale datelor, deoarece accesul direct la ele este restricționat**
  - **gruparea datelor şi a operațiilor ce se pot realiza cu ele permit depistarea uşoară a eventualelor probleme**
  - **modificările interne ale clasei nu se reflectă şi în afara ei (dacă se păstrează aceeaşi interfață), ca urmare restul programului nu trebuie modificat**

Membrii `public` vor fi vizibili pentru orice porțiune de cod din aplicație. Membrii `private` vor fi vizibili numai pentru membrii clasei din care face parte. Membrii `protected` sunt vizibili numai prin moştenire.

Condițiile de implementare corectă a încapsulării:

- toate variabilele de instanță sunt declarate cu `private`;
- numai metodele `public` ale obiectului vor putea fi utilizate pentru accesul la datele private ale acestuia;

În continuare se prezintă o astfel de implementare a clasei `Punct`:

În această implementare `x`, `y` şi `distanta` sunt toate declarate `private` , deci invizibile pentru un utilizator al clasei.

Modificarea valorilor acestor variabile de insstanţă se face numai sub controlul metodelor:

- automat la crearea instanţierea obiectelor prin acţiunea automată a constructorilor: `Punct()`, `Punct(double x, double y)`
- prin apelul metodelor: `setX(double x)`, `setY(double y)`

Vizualizarea valorilor stocate în variabilele provate se face cu metodele: `x()`, `y()` şi `distantaOrigine()`.

```
public class Punct {
//Campuri
  private double x;
  private double y;
  private double distanta;

//Constructori
  Punct() {
      setX(0);
      setY(0);
      distanta = 0;
  }

  Punct(double x, double y) {
      setX(x);
      setY(y);
      actualizareDistanta();
  }

// Metode
  public void setX(double x) {
      this.x = x;
      actualizareDistanta();
  }

  public void setY(double y) {
      this.y = y;
      actualizareDistanta();
  }

  public double x() {
      return x;
  }

  public double y() {
      return y;
  }

  public double distantaOrigine() {
      return distanta;
  }

  private void actualizareDistanta()
{
       distanta =
Math.sqrt(x*x+y*y);
    }

   public String toString() {
       return "<" + x + "," + y +
">";
  }
}
```

C7- Întrebari

1. Enumerati 4 avantaje ale POO.
2. Care sunt principiile POO?
3. Ce este clasa si cum se defineste?
4. Ce este obiectul, cum se acceseaza membrii?
5. Ce este comunerea, exemplificati?
6. Ce este mostenirea, exemplificati?

BIBLIOGRAFIE

1. http://www.oracle.com/technetwork/java/javase/documentation/index.html

2. http://docs.oracle.com/javase/6/docs/

3. Stefan Tanasa, Cristian Olaru, Stefan Andrei, Java de la 0 la expert, Polirom, 2003, ISBN: 973-681-201-4.

4. Herber Schild, Java 2 - The Complete Reference, Fourth Edition, Osborne, 2001, ISBN: 0-07-213084-9.

5. Deitel H.M., Deitel P. J., Java - How to programm, Fith Edition, Prentice Hall, 2003, ISBN: 0-13-120236-7.

6. http://www.east.utcluj.ro/mb/mep/antal/downloads.html