Lucrare de laborator nr. 8
Clase derivate

1. Scopul lucrării

În această lucrare se studiază un concept fundamental în programarea orientată pe obiecte: *moştenirea*.

2. Breviar teroretic

Moştenirea este capacitatea prin care dezvoltăm o nouă clasă plecând de la o clasă existentă. Noua clasă obţinută prin moştenire de la clasa existentă, se cheamă **clasă derivată** (**clasă fiu** sau **subclasă**). Clasa existentă din care prin moştenire derivăm subclasa se cheamă **clasă de bază** (sau **clasă părinte** sau **supraclasă**).

Moştenirea este o trăsătură fundamentală în POO. Ea permite să dezvoltăm mai rapid noi clase pe baza claselor existente. Astfel, în loc să proiectăm de la zero o nouă clasă, căutăm între clasele existente o clasă asemănătoare şi noua clasă o derivăm din aceasta. Acest stil de programare se cheamă **programare prin diferenţe (** în noua clasă programăm doar diferenţele faţă de clasa existentă).

Subclasarea unor clase existente este utilă pentru a nu "reinventa" cod deja creat. Dacă între o clasa nouă care trebuie proiectată şi o clasă existentă este o relaţie de tipul "is_a" ("este_un" sau "este_o") noua clasă poate fi obţinută prin moştenire de la clasa existentă. Astfel , având clasa Fruct şi dacă se cere să scriem o nouă clasă, clasa Măr, în loc să definim clasa Măr de la început, putem să o derivăm din clasa existentă Fruct, pentru că un măr este un fruct.

În biblioteca grafică în pachetul **javax.swing** există **clasa JFrame,** ce modelează o fereastră grafica (cu bara de titlu, cu cele trei butoane tipice si cu comportament de fereastra). Daca vrem sa facem o aplicație in care intr-o fereastra avem mai multe componente grafice, atunci clasa necesară în această aplicaţie va subclasa (va moşteni) clasa JFrame.

Pentru a implementa moştenirea se foloseşte cuvântul cheie **extends**.

Exemplu

class FereastraTestGrilă extends Jframe

Legat de moştenire există un specificator de acces ce se aplică membrilor unei clase (variabile de instanţă, constructori, metode) şi anume specificatorul de acces **protected**.

Membrii unei superclase declaraţi **protected** pot fi accesaţi doar de subclasele ei. Aceste subclase pot face parte din alte package-uri de clase decât package-ul clasei de bază.

Clasele ce fac parte din acelaşi package ca şi superclasa, chiar dacă nu o subclasează (nu o moştenesc) au acces şi la membrii protected.

În mod evident, la membrii **private** nu au acces nici subclasele.

În Java, spre deosebire de limbajul C++, nu există **moştenire multiplă**, adică în Java o subclasă nu poate avea decât un singur părinte. În Java, o clasă extinde o singură clasă (nu moşteneşte decât de la o singură clasă), dar poate să implementeze mai multe interfeţe.

Subclasa moşteneşte de la superclasă membrii **protected** sau **public** ai superclasei.

În subclasă pot fi adăugate noi variabile de instanţă (care nu există în superclasă) şi noi metode sau subclasa poate să redefinească anumite metode moştenite de la superclasă.

Constructorii nu sunt moşteniţi de la superclasă la subclasă.

Din subclasă se poate apela constructorul clasei de bază folosind apelând **super( )**, cu parametrii necesari.

În cazul în care se apelează cu **super(...)** constructorul clasei de bază, această instrucţiune trebuie să fie prima instrucţiune din metoda respectivă (ce conţine apelul cu **super**).

Strămoşul tuturor claselor din Java este clasa **Object**.

Chiar dacă o clasă nu extinde în mod explicit o altă clasă (nu foloseşte cuvântul cheie extends), ea moşteneşte în mod implicit clasa Object.

În clasa Object sunt definite metode ce sunt moştenite de toate clasele din Java. Iata cateva din acestea::

- metoda **equals( )**

  public boolean equals(Object o)

- metoda **toString( )**

public String toString( )
se foloseşte pentru a da o reprezentare sub formă de String unui obiect.

În declararea unui obiect din clasa derivată poate fi folosit ca tip şi clasa de bază. Invers, nu se poate.
Exemplu:

Dacă avem clasa Fereastra derivată din clasa JFrame:

class Fereastra extends JFrame { …. }

Putem instanţia un obiect Fereastra , astfel:

Fereastra f=new Fereastra( );

sau:

JFrame f=new Fereastra( );

3. Probleme rezolvate

Problema 1

Să se construiască clasa ContBancar, folosită pentru a modela un cont bancar, ce are ca variabilă de instanţă privată, variabila *suma*, (suma de bani din cont). Ca metode:

- constructorul;
- adauga(), ce are ca parametru un număr real x, valoarea ce se adaugă în cont;
- extrage(), ce are ca parametru un număr real x, valoarea ce se extrage din cont, şi care scoate ca rezultat *true*, dacă se poate face extragerea (suma >= x), şi *false* în caz contrar;
- getSuma(), ce returnează valoarea variabilei private suma;
- afisare(), ce afişează valoarea sumei de bani din cont.

Pe baza clasei ContBancar se va dezvolta prin derivare (moştenire) clasa ContBancarExtins, în care se va adăuga o nouă variabilă de instanţă: rata dobânzii anuale şi o nouă metodă: metoda adaugaDobandaLunara(), ce adaugă în cont dobânda calculată după trecerea unei luni. În clasa ContBancarExtins se va redefini şi metoda afisare(), astfel încât să se afişeze şi rata dobânzii. De asemenea, în această nouă clasă se va defini constructorul, prin care se iniţializează suma de bani din cont şi rata dobânzii.
Să se scrie şi o clasă de test pentru clasa ContBancarExtins.

```
class ContBancar
{
 private double suma;
```

```
 public ContBancar(double S)
 {
        suma=S;
 }

 public void adauga(double S)
 {
        suma=suma+S;
 }

 public boolean extrage(double S)
 {
        if(S>suma)return false;
        suma=suma-S;
        return true;
 }

 public double getSuma()
 {
        return suma;
 }

 public void afisare()
 {
        System.out.println("suma="+suma);
 }
}

class ContBancarExtins extends ContBancar
{
private double rd;//rata dobanzii anuale
public ContBancarExtins(double S,double rata)
{
        super(S);
        rd=rata;
}

public void adaugaDobandaLunara()
{
        double S=this.getSuma();
```

```
        double dobanda=S*rd/12;
        this.adauga(dobanda);
}

public void afisare()
{
        System.out.println("suma="+this.getSuma());
        System.out.println("rata dobanzii="+rd);
}
}

class TestCont
{
public static void main(String args[])
{
        ContBancarExtins c=new ContBancarExtins(1000,0.12);
        c.adauga(1000);
        c.adaugaDobandaLunara();
        c.afisare();
}
}
```

Problema 2

Să se construiască clasa Cerc, ce are ca variabilă de instanţă privată, un număr întreg r, ce reprezintă raza unui cerc. În această clasă avem ca metode:

- constructorul, ce face iniţializarea razei;
- getRaza(), ce returnează raza;
- calculArie(),ce returnează aria cercului;
- calculPerimetru(),ce returnează perimetrul cercului;
- suntEgale(), ce are ca parametru un Cerc c, şi care returnează *true* dacă cercul curent este egal cu cercul c (au aceeaşi rază).
- afisare(), ce afişează raza cercului.

Din clasa Cerc se va deriva clasa CercExtins, în care se vor adăuga ca variabile de instanţă x şi y: coordonatele centrului şi se vor redefini metodele suntEgale() (cercurile sunt egale când au aceeaşi rază şi aceleaşi coordonate ale centrului), şi afisare() (pe lângă rază, va afişa şi coordonatele centrului)

Să se scrie şi o clasă de test pentru clasa CercExtins.

```
class Cerc
{
  private int raza;

  public Cerc(int x)
  {
   raza=x;
  }

  public int getRaza()
  {
   return raza;
  }

  public double calculArie()
  {
   return Math.PI*raza*raza;
  }

  public double calculPerimetru()
  {
   return 2*Math.PI*raza;
  }

  public boolean suntEgale(Cerc c)
  {
   if(this.raza==c.raza)return true;
   else return false;
  }

 public void afisare()
 {
   System.out.println("raza="+raza);
 }
}

class CercExtins extends Cerc
{
  private int x;
  private int y;
```

```
 public CercExtins(int r,int x0, int y0 )
 {
  super(r);
  x=x0;
  y=y0;
 }

 public boolean suntEgale(CercExtins c)
{
 if((this.getRaza()==c.getRaza())&&(this.x==c.x)&&(this.y==c.y))
 return true;
 else return false;
}

 public void afisare()
{
 System.out.println("raza="+this.getRaza());
 System.out.println("x="+x);
 System.out.println("y="+y);
 }
}

class TestCercExtins
{
 public static void main (String args[])
 {
  CercExtins c=new CercExtins(3,0,1);
  System.out.println("Aria= "+c.calculArie());
  CercExtins c1=new CercExtins(3,0,10);
  System.out.println("Sunt egale= "+c.suntEgale(c1));
 }
}
```

Problema 3

Să se construiască clasa Punct, folosită pentru a modela un punct în plan, ce are ca variabile de instanţă x şi y, coordonatele unui punct în plan. Ca metode:

- constructorul;
- muta(), ce are doi parametrii dx şi dy, pe baza cărora noile coordonate ale punctului devin: x+dx, y+dy;

- compara(), ce are ca parametru un punct p, şi care returnează *true*, dacă punctul curent (cel referit prin *this*) este egal cu punctul p, şi *false* în caz contrar;
- distantaOrigine(), ce returnează distanţa între punctul curent şi origine;
- distanta(), ce are ca parametru un punct p, şi care returnează distanţa între punctul curent şi punctul p;
- getX() ce returnează valoarea coordonatei x;
- getY() ce returnează valoarea coordonatei y;
- setX() ce setează valoarea coordonatei x;
- setY() ce setează valoarea coordonatei y;
- afisare() ce afişează coordonatele punctului.

Pe baza clasei Punct, se va dezvolta prin moştenire, clasa PunctColor, în care se va adăuga o nouă variabilă de instanţă de tipul String: culoarea punctului şi noile metode getCuloare() ce returnează culoarea punctului şi setCuloare() ce setează culoarea punctului. Se vor redefini metodele compara() şi afişare() şi noul constructor.
Să se scrie şi o clasă de test pentru clasa PunctColor.

```
class Punct
{
 private int x;//coordonata x a punctului
 private int y;

 public Punct(int x,int y)
 {
   this.x=x;
   this.y=y;
 }

 public int getX()
 {
   return x;
 }

 public int getY()
 {
        return y;
 }
```

```
public void setX(int x)
{
        this.x=x;
}
public void setY(int y)

{
        this.y=y;
}

public void afisare()
{
        System.out.println("x="+x);
        System.out.println("y="+y);
}

public void muta(int dx, int dy)
{
        x=x+dx;
        y=y+dy;
}

public boolean compara(Punct p)
{
        if((x==p.x)&&(y==p.y))
          return true;
        else return false;
}

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

public double distanta(Punct p)
{
        double dx=this.x-p.x;
        double dy=this.y-p.y;
        double dist=Math.sqrt(dx*dx+dy*dy);
```

```
        return dist;
 }
}

class PunctColor extends Punct
{
 private String culoare;

 public PunctColor(int x, int y, String culoare)
 {
  super(x,y);
  this.culoare=culoare;
 }

 public String getCuloare()
 {
        return culoare;
 }

 public void setCuloare(String culoare)
 {
        this.culoare=culoare;
 }

 public void afisare()
 {
        System.out.println("x="+getX());
        System.out.println("y="+getY());
        System.out.println("culoare="+culoare);
 }

 public boolean compara(PunctColor p)
 {
        if((this.getX()==p.getX())&&
          (this.getY()==p.getY())&&
          (this.culoare==p.culoare))
         return true;
        else return false;
 }
}
```

```
class TestPuncte
{
  public static void main (String args[])
  {
        PunctColor p=new PunctColor(0,1,"negru");
        p.muta(1,1);
        p.afisare();
  }
 }
```

Problema 4

Să se dezvolte clasa Persoana ce are ca variabile de instanţă numele şi prenumele unei persoane şi vârsta ei, şi ca metode:

- constructorul ce face iniţializările;
- getNume(), ce returnează numele;
- getPrenume(), ce returnează prenumele;
- afisare(), ce afişează informaţiile despre persoană.

Din clasa Persoană se va deriva clasa Student, ce are în plus ca variabile de instanţă, numele facultaţii pe care o urmează şi numărul matricol. În clasa Student se va dezvolta un nou constructor şi se va redefini metoda afisare(). Se vor adăuga în plus metodele:

- getFacultate();
- getNumărMatricol().

Să se scrie apoi o aplicaţie pe baza clasei Student, în care se vor citi de la tastatură N studenţi, ce se vor memora într-un vector. Se vor afişa câţi studenţi au prenumele ”Ion”.

```
import javax.swing.*;
class Persoana
{
  private String nume;
  private String prenume;
  private int varsta;

  public Persoana(String n, String p, int v)
  {
   nume=n;
   prenume=p;
   varsta=v;
```

```
  }

  public String getNume()
  {
  return nume;
  }

  public String getPrenume()
  {
  return prenume;
  }

  public void afisare()
  {
  System.out.println(nume+" "+prenume+" : "+varsta);
  }
}

class Student extends Persoana
{
  private String numeFacultate;
  private int nrMatricol;

  public Student(String n, String p, int v, String facult, int nrMatr)
  {
   super(n,p,v);
   numeFacultate=facult;
   nrMatricol=nrMatr;
   }

  public String getFacultate()
  {
  return numeFacultate;
  }

  public int getNumarMatricol()
  {
  return nrMatricol;
  }
}
```

```
class TestStudenti
{
  public static void main(String args[])
  {
   final int N=2;
   int i;
   Student s[]=new Student[N];
   for(i=0;i<N;i++){
     String nume=JOptionPane.showInputDialog("nume=");
     String prenume=JOptionPane.showInputDialog("prenume=");
     int varsta=
    Integer.parseInt(JOptionPane.showInputDialog("varsta="));
    String facultate=JOptionPane.showInputDialog("facultate=");
    int  nrMatr=Integer.parseInt(JOptionPane.showInputDialog(
                                        "nr.  matricol="));
   s[i]=new Student(nume,prenume,varsta,facultate,nrMatr);
   }
   int contor_ion=0;
   for(i=0;i<N;i++){
     String prenumeCrt=s[i].getPrenume();
     if(prenumeCrt.compareTo("Ion")==0)contor_ion++;
   }
   System.out.println(contor_ion);
  }
}
```

3. Probleme propuse

Problema 1

Dându-se clasa Cerc, definită în problema 2, să se deriveze din aceasta clasa CercColor, în care se va adăuga ca variabilă de instanţă culoarea cercului (de tip String). Se va scrie noul constructor şi se vor adăuga metodele getCuloare() şi setCuloare() şi se vor redefini metodele suntEgale() (cercurile sunt egale când au aceeaşi rază şi aceeaşi culoare), şi afisare() (pe lângă rază, va afişa şi culoarea cercului)
Să se scrie şi o clasă de test pentru clasa CercColor.

Problema 2

Să se construiască clasa Punct3D, folosită pentru a modela un punct în spaţiu, ce are ca variabile de instanţă x, y, z, coordonatele unui punct în spaţiu. Ca metode:

- constructorul;
- muta(), ce are trei parametrii dx, dy şi dz, pe baza cărora noile coordonate ale punctului devin: x+dx, y+dy, z+dz;
- compara(), ce are ca parametru un punct p, şi care returnează *true*, dacă punctul curent (cel referit prin *this*) este egal cu punctul p, şi *false* în caz contrar;
- distanta(), ce are ca parametru un punct p, şi care returnează distanţa între punctul curent şi punctul p;
- getX() ce returnează valoarea coordonatei x;
- getY() ce returnează valoarea coordonatei y;
- getZ() ce returnează valoarea coordonatei z;
- afisare() ce afişează coordonatele punctului.

Pe baza clasei Punct3D, se va dezvolta clasa Punct3DColor, în care se va adăuga o nouă variabilă de instanţă de tipul String: culoarea punctului şi o nouă metodă getCuloare() ce returnează culoarea punctului. Se vor redefini metodele compara() şi afişare() şi noul constructor.

Să se scrie şi o clasă de test pentru clasa Punct3DColor.

Problema 3

Să se modifice clasa TestPuncte din problema rezolvată numărul 3, astfel încât se va citi de la tastatură un număr natural N, se vor citi de la tastatură N puncte color în vectorul p[ ] şi apoi se va afişa dacă toate punctele introduse sunt diferite între ele sau nu.