**2011** - 2012

# **P**rogramare **O**rientata **spre O**biecte
# (*Object-Oriented Programming*)

a.k.a. Programare Obiect-Orientata

Titular curs: Eduard-Cristian Popovici

Suport curs: *http://discipline.elcom.pub.ro/POO-Java/*

## 2. Orientarea spre obiecte in limbajul Java

### 2.4. Generalizare, specializare si mostenire

## Clasa vazuta ca o generalizare (abstractizare) a obiectelor

**Clasa**

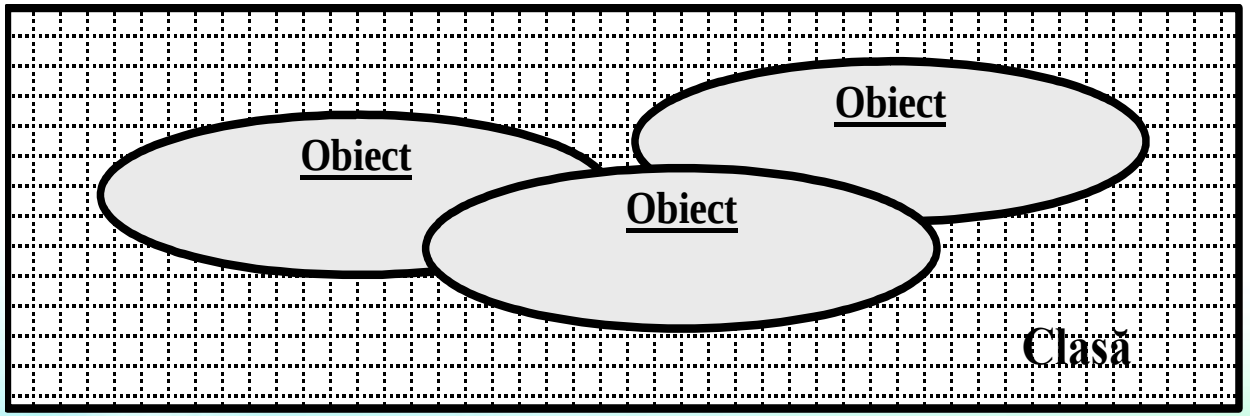
- contine **generalitatile** (abstractiile generale)

**Obiectele**

- contin **particularitatile** (detaliile particulare)

**Clasa**

- vazuta ca **multime de obiecte**



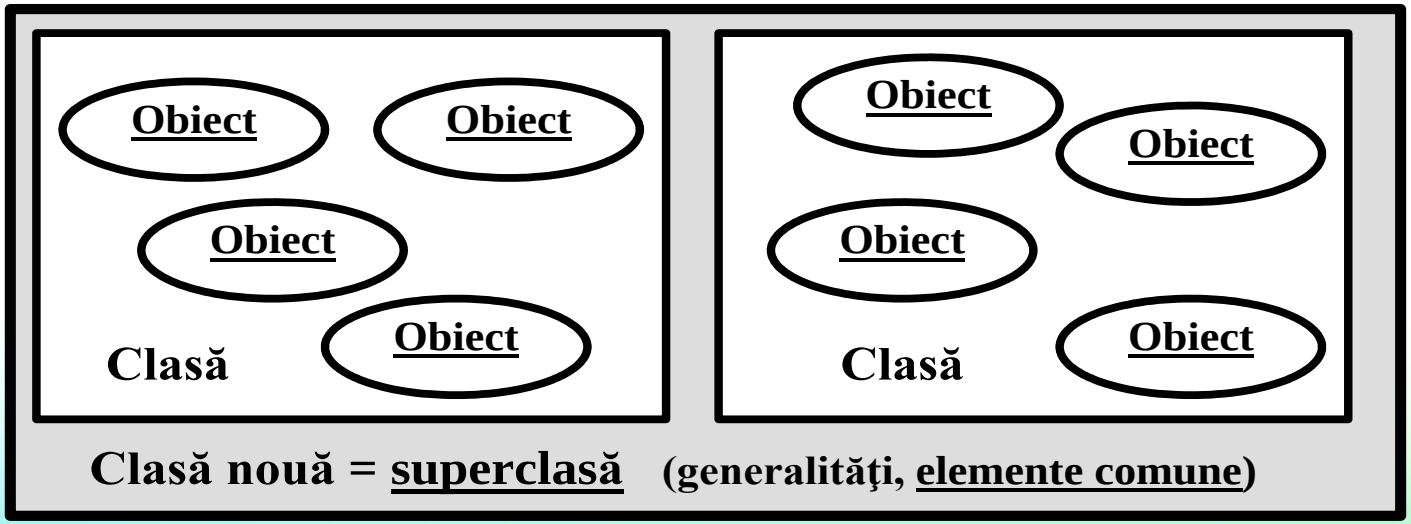

= generalităţi (elemente, aspecte, caracteristici comune)

= particularităţi (elemente, aspecte, caracteristici diferite)

## Generalizarea si specializarea

**Generalizarea**

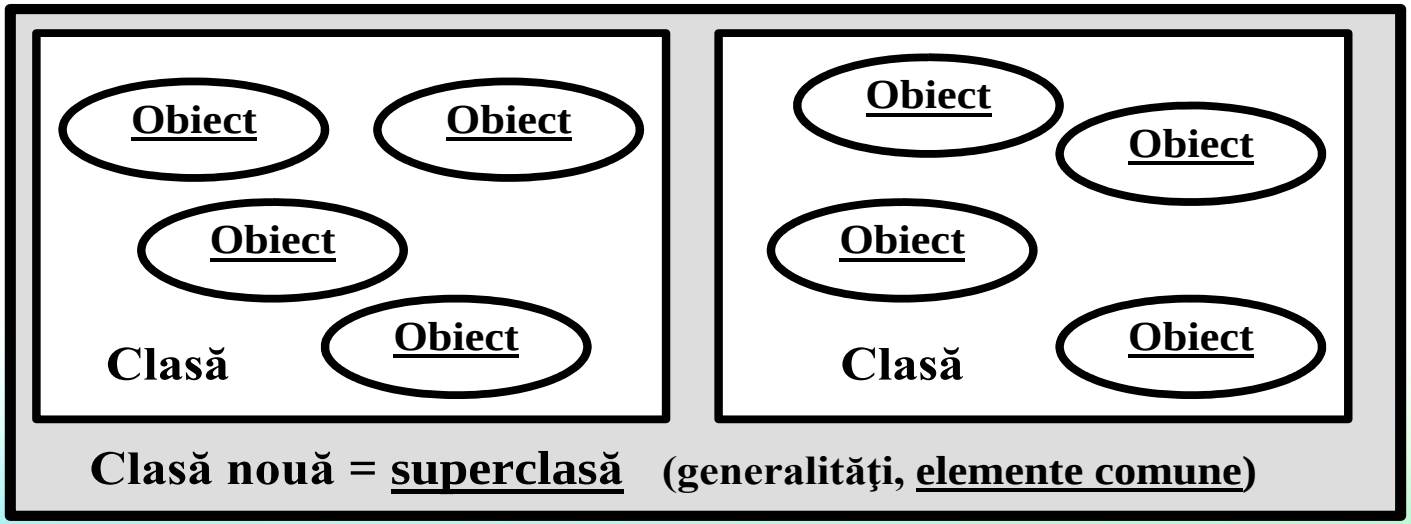
- inseamna extragerea elementelor comune (atribute, operaţii şi constrângeri) ale unui ansamblu de clase
  - într-o **nouă clasă** mai generală, denumită **superclasă** (parinte)
- superclasa este o abstracţie a subclaselor (descendentilor) ei
- rezulta o ierarhie de clase bazata pe generalizare in care
  - arborii de clase sunt construiţi pornind de la frunze



## Generalizarea si specializarea

**Generalizarea**

- este utilizată cand elementele modelului au fost identificate
  - pentru a obţine o descriere generica a soluţiilor
- semnifică "este un (fel de)“
  - un obiect dintr-o subclasa este un (fel de) obiect din superclasa
- priveşte clasele vazute ca multimi (NU produce legaturi intre obiecte)
  - subclasa este o submultime de obiecte ale superclasei



## Generalizarea si specializarea

Generalizarea actioneaza in OO la **doua niveluri**:

- **clasele** sunt **generalizari ale** **ansamblurilor de obiecte**
  - un obiect este de felul specificat de o clasa
- **superclasele** sunt **generalizari de** **clase**
  - obiectele de felul specificat in clasa sunt si de felul specificat in superclasa

**Limbajele** “orientate spre obiecte” (**OO**)

- ofera **ambele mecanisme** de generalizare
- de ex. Java, C++, C#

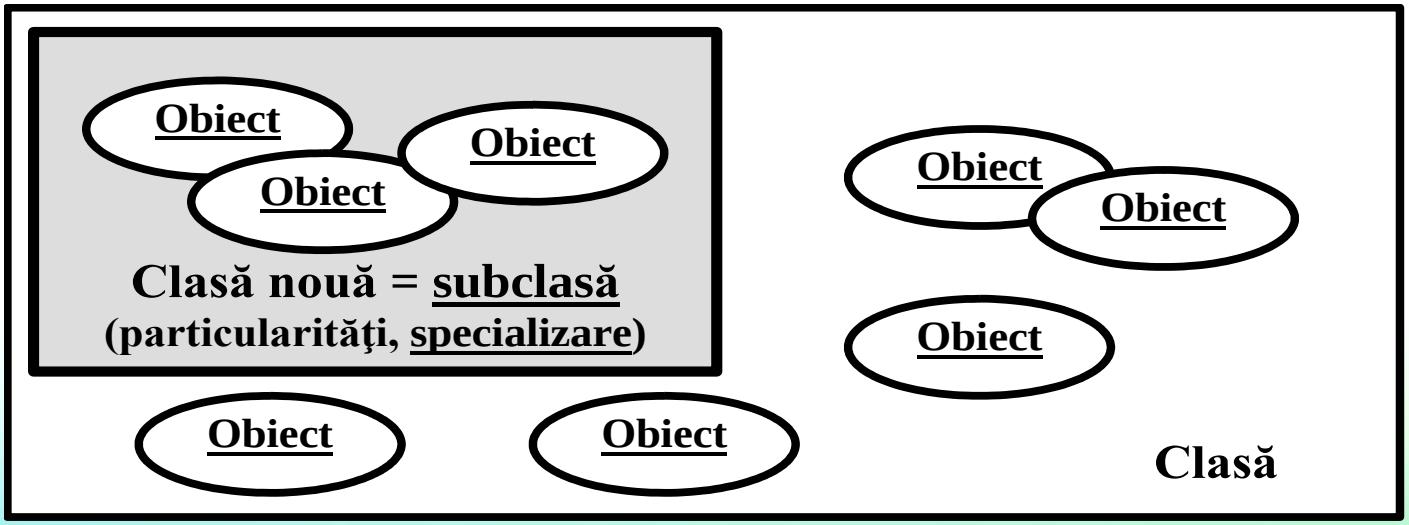
**Limbajele** care ofera **doar constructii** numite **obiecte** (eventual **clase**) se pot numi

- limbaje “**care lucreaza cu**” **obiecte** (si eventual **clase**)

## Generalizarea si specializarea

**Specializarea**

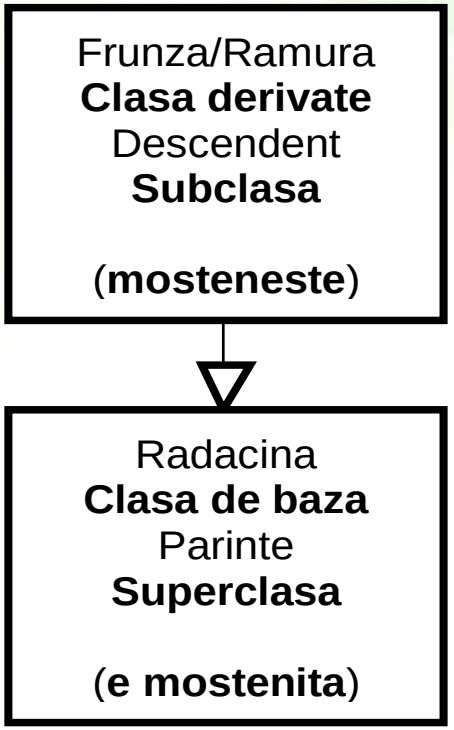
- inseamna capturarea particularităţilor / elementelor distincte ale unui ansamblu de obiecte ale unei clase existente
  - noile caracteristici fiind reprezentate într-o **nouă clasă** mai specializată, denumită **subclasă**
- este utilă pentru extinderea coerentă a unui ansamblu de clase
  - noile cerinţe fiind încapsulate în subclase care extind coerent si uniform funcţiile existente



## Mostenirea – mecanism suport pentru generalizare/specializare

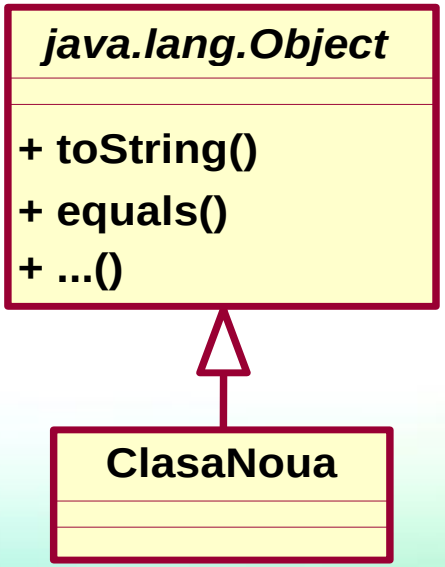
**Moştenirea**

- tehnică de generalizare oferită de limbajele OO pentru a construi o clasă pornind de la una sau mai multe alte clase
  - partajând atributele si operaţiile într-o ierarhie de clase




**Orice clasa Java care nu extinde** prin mostenire in mod **explicit o alta**

- **extinde implicit** clasa **Object** (radacina ierarhiei de clase Java)
  - care **contine metodele necesare tuturor obiectelor Java** (*equals()*, *toString()*, *clone()*, etc.)

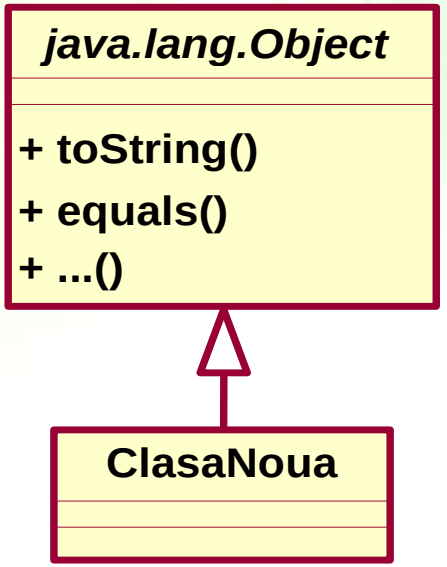


## Mostenirea – mecanism suport pentru generalizare/specializare

**Orice clasa Java care nu extinde** prin mostenire **explicit o alta**

- **extinde implicit** clasa **Object** (radacina ierarhiei de clase Java)

  - care contine metodele necesare tuturor obiectelor Java (*equals()*, *toString()*, etc.)




Declaratia:

```
class NumeClasa {
    // urmeaza corpul clasei ...
```

**este echivalenta cu**:

```
class NumeClasa extends Object {
    // urmeaza corpul clasei ...
```

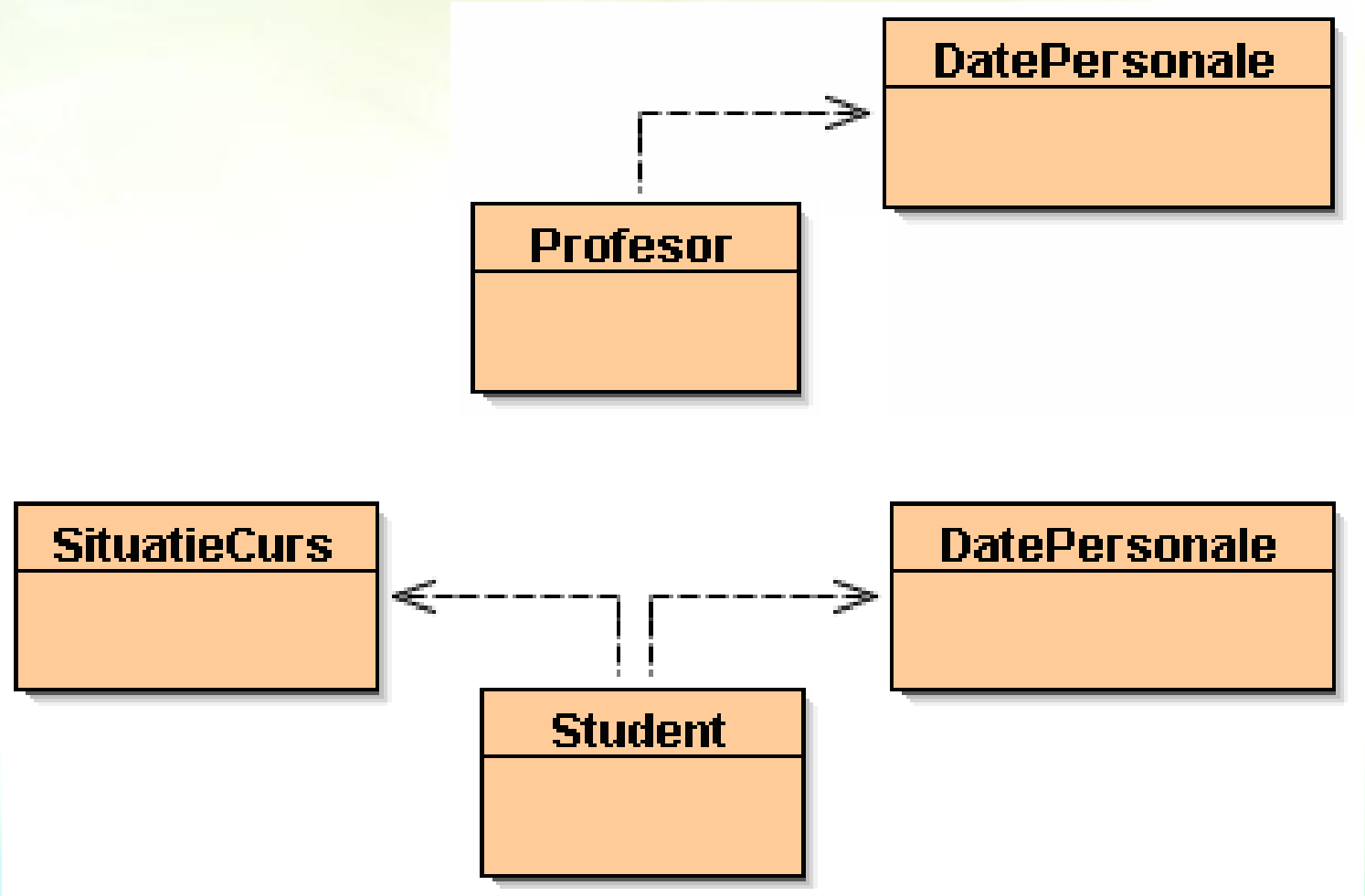
## Exemplu de clasa specializata

```
public class Profesor {   // Incapsuleaza informatiile despre un Profesor
   // Campuri ascunse
   private DatePersonale date;
   private String titlu;

   // Constructori
   public Profesor(String nume, String initiale, String prenume,int anNastere){
      date = new DatePersonale(nume, initiale, prenume, anNastere);
   }
   // Interfata publica si implementarea ascunsa
   public void setTitlu(String t) {
      titlu = new String(t); // copiere „hard" a obiectului primit ca parametru
   }
   public String toString() {                  // forma „String" a campurilor
      return ("Profesorul " + date + " are titlul " + titlu);
   }
   public static void main(String[] args) {
      // Crearea unui nou Profesor, initializarea campurilor noului obiect
      Profesor pr = new Profesor("Nulescu", "Ion", "A.", 1960);
      pr.setTitlu("Lector Dr.");
      // Utilizarea informatiilor privind Profesorul
      System.out.println(pr.toString());    // afisarea formei „String"
   }
}
```

## Exemplu de clasa specializata

```
public class Student { // Incapsuleaza informatiile despre un Student
   // Campuri ascunse
   private DatePersonale date;
   private SituatieCurs[] cursuri;
   private int numarCursuri = 0;       // initializare implicita
   // Constructori
   public Student(String nume, String initiale, String prenume, int anNastere){
      date = new DatePersonale(nume, initiale, prenume, anNastere);
      cursuri = new SituatieCurs[10]; // se initializeaza doar date si cursuri
   }
   // Interfata publica si implementarea ascunsa
   public void addCurs(String nume) { // se adauga un nou curs
      cursuri[numarCursuri++] = new SituatieCurs(nume);
   }
   public void notare(int numarCurs, int nota) {
      cursuri[numarCurs].notare(nota); // se adauga nota cursului specificat
   }
   public String toString() {          // forma „String" a campurilor
      String s = "Studentul " + date + " are urmatoarele rezultate:\n";
      for (int i=0; i<numarCursuri; i++)
         s = s + cursuri[i].toString() + "\n";
      return (s);
   }
}
```

## Relatiile dintre clasele anterioare



## Exemplu de clasa care generalizeaza clasele anterioare

```
public class Persoana {     // Incapsuleaza informatiile despre o Persoana

   // Campuri ascunse
   protected DatePersonale date;

   // Constructori
   public Persoana(String nume, String initiale, String prenume,int anNastere){
      date = new DatePersonale(nume, initiale, prenume, anNastere);
   }

   // Interfata publica si implementarea ascunsa
   public String toString() {                    // forma „String" a campurilor
      return (date.toString());
   }

   public static void main(String[] args) {

      // Crearea unei noi Persoane, initializarea campurilor noului obiect
      Persoana p = new Persoana("Julescu", "Ion", "C.", 1965);

      // Utilizarea informatiilor privind Persoana
      System.out.println(p.toString());     // afisarea formei „String"
   }
}
```

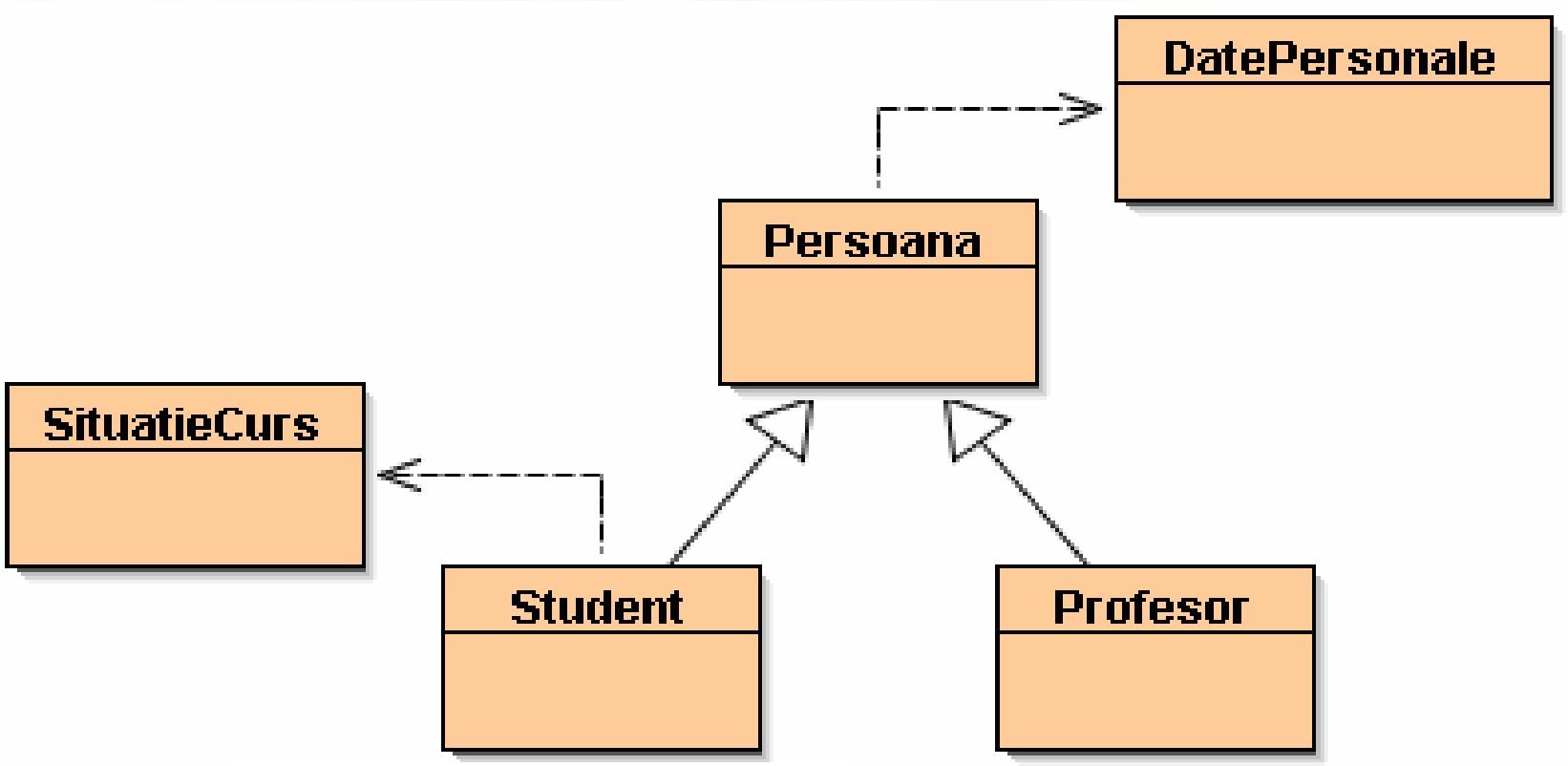
## Exemplu de clasa rescrisa pentru a mosteni clasa generala

```
public class Profesor extends Persoana {

  // Campuri ascunse
  private String titlu;

  // Constructori
  public Profesor(String nume, String initiale, String prenume,int anNastere){
    super(nume, initiale, prenume, anNastere);  // apel constructor supraclasa
  }                                             // (reutilizare cod/delegare)
  // Interfata publica si implementarea ascunsa
  public void setTitlu(String t) {
     titlu = new String(t); // copiere „hard" a obiectului primit ca parametru
  }
  public String toString() {                    // forma „String" a campurilor
     return ("Profesorul " + date + " are titlul " + titlu);
  }
  public static void main(String[] args) {
     // Crearea unui nou Profesor, initializarea campurilor noului obiect
     Profesor pr = new Profesor("Nulescu", "Ion", "A.", 1960);
     pr.setTitlu("Lector Dr.");
     // Utilizarea informatiilor privind Profesorul
     System.out.println(pr.toString());    // afisarea formei „String"
  }
}
```

## Exemplu de clasa rescrisa pentru a mosteni clasa generala

```
public class Student extends Persoana {
   // Campuri ascunse
   private SituatieCurs[] cursuri;
   private int numarCursuri = 0;       // initializare implicita

   // Constructori
   public Student(String nume, String initiale, String prenume,int anNastere){
      super(nume, initiale, prenume, anNastere); // apel constructor supraclasa
      cursuri = new SituatieCurs[10];  // se initializeaza doar date si cursuri
   }
   // Interfata publica si implementarea ascunsa
   public void addCurs(String nume) { // se adauga un nou curs
      cursuri[numarCursuri++] = new SituatieCurs(nume);
   }
   public void notare(int numarCurs, int nota) {
      cursuri[numarCurs].notare(nota); // se adauga nota cursului specificat
   }
   public String toString() {          // forma „String" a campurilor
      String s = "Studentul " + date + " are urmatoarele rezultate:\n";
      for (int i=0; i<numarCursuri; i++)
         s = s + cursuri[i].toString() + "\n";
      return (s);
   }
}
```

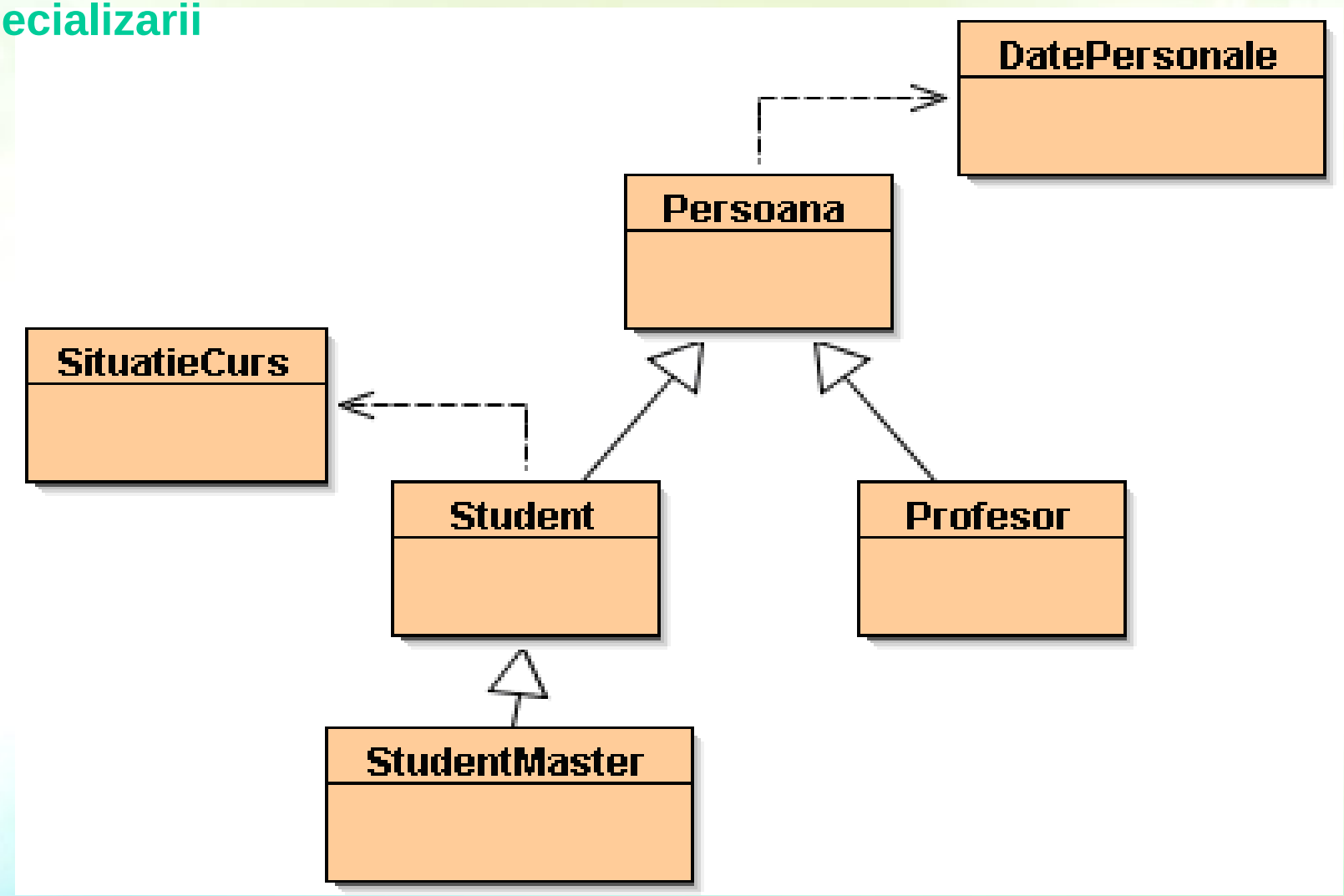
## Relatiile dintre clasele anterioare dupa introducerea generalizarii



## Exemplu de clasa care specializeaza clasa anterioara

```
public class StudentMaster extends Student {

   // Campuri ascunse
   private String specializare; // specializare obtinuta anterior (la licenta)

   // Constructori
   public StudentMaster(String nume, String initiale, String prenume,
                        int anNastere) {
      super(nume, initiale, prenume, anNastere); // apel constructor supraclasa
   }

   // Interfata publica si implementarea ascunsa
   public void setSpecializare(String spec) {  // se stabileste specializarea
      specializare = new String(spec);   // copiere „hard" a obiectului primit
   }

   public String toString() {                  // forma „String" a campurilor
      String s = "Studentul " + date + " cu specializarea " + specializare +
                 " are urmatoarele rezultate:\n";
      for (int i=0; i<numarCursuri; i++)   s =s + cursuri[i].toString() + "\n";
      return (s);
    }
}
```

## Relatiile dintre clasele anterioare dupa introducerea generalizarii si a specializarii



## Exemplu de clasa tinta

**Clasa tinta** numita **Radio** care

- **simuleaza planurile** pentru **crearea unui obiect radio**

```
public class Radio {

   // camp (atribut, variabila membru)
   // definire tablou pentru asociere butoane cu frecvente
   protected double[] stationNumber = new double[5];

   // metoda (operatie, functie membru)
   // definire asociere buton cu frecventa
   public void setStationNumber(int index, double freq) {
      stationNumber[index] = freq;
   }

   // metoda (operatie, functie membru)
   // definire selectie frecventa
   public void playStation(int index) {
      System.out.println("Playing the station at " + stationNumber[index] + " Mhz");
   }
}
```

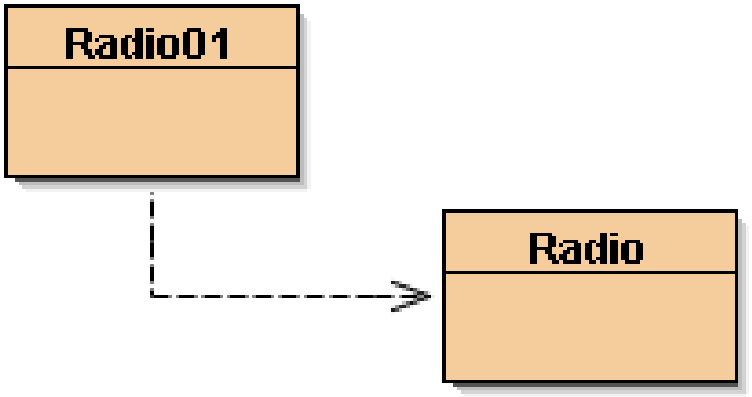
**( explicatii la http://discipline.elcom.pub.ro/POO-Java/Esenta POO - Obiecte Clase Incapsulare.pdf**

**– adaptare dupa http://www.developer.com/java/article.php/935351 )**

## Exemplu de clasa utilizator pentru clasa tinta anterioara

**Clasa utilizator** numita **Radio01** care

- **creeaza** si **foloseste** un **obiect al clasei** Radio
- **testeaza clasa** Radio




```
public class Radio01 {

   public static void main(String[] args){

      // creare obiect
      Radio myObjRef = new Radio();

      // apel asociere buton cu frecventa
      myObjRef.setStationNumber(3, 93.5);

      // apel selectie frecventa
      myObjRef.playStation(3);
   }

}     // Rezultat: Playing the station at 93.5 MHz
```

## Exemplu de clasa de baza

**Pregatirea clasei Radio** pentru a **simula planurile** de **creare a unui obiect radio-casetofon**

```
public class Radio {

  // definire tablou pentru asociere butoane cu frecvente
  protected double[] stationNumber = new double[5];

  // definire variabila care indica prezenta casetei
  protected boolean tapeIn = false;                          // tapeIn = true

  // definire asociere buton cu frecventa
  public void setStationNumber(int index, double freq) {
     stationNumber[index] = freq;
  }

  // definire selectie frecventa
  public void playStation(int index) {
    System.out.println("Play Radio");
    if (!tapeIn) {                                           // tapeIn == false
      System.out.println("  Playing the station at " + stationNumber[index] + " Mhz");
    } else {                                                 // tapeIn == true
      System.out.println("  Remove the tape first!");
    }
  }
}
```

## Exemplu de clasa extinsa

**Prima varianta a clasei Combo** care **simuleaza planurile** de **creare a unui obiect radio-casetofon**

```
public class Combo extends Radio {
    public void insertTape() {
        System.out.println("Insert Tape");
        tapeIn = true;                                    // tapeIn = true
        System.out.println("  Tape is in");
        System.out.println("  Radio is off");
    }
    public void removeTape() {
        System.out.println("Remove Tape");
        tapeIn = false;                                   // tapeIn = false
        System.out.println("  Tape is out");
        System.out.println("  Radio is on");
    }
    public void playTape() {
        System.out.println("Play Tape");
        if (!tapeIn) {                                    // tapeIn == false
           System.out.println("  Insert the tape first!");
        } else {                                          // tapeIn == true
           System.out.println("  Tape is playing");
        }
    }
}
```

## Exemplu de clasa utilizator pentru clasa extinsa anterioara

**Clasa utilizator** numita **Radio02** care **creeaza** si **foloseste** un **obiect al clasei** Combo, si **testeaza clasa** Combo

```
public class Radio02 {

    public static void main(String[] args){

        // creare obiect
        Combo myObjRef = new Combo();

        myObjRef.setStationNumber(3,93.5);
        myObjRef.playStation(3);
        myObjRef.insertTape();

        myObjRef.playStation(3);

        myObjRef.removeTape();
        myObjRef.playStation(3);

        myObjRef.playTape();

        myObjRef.insertTape();
        myObjRef.playTape();
    }
}
```

**Rezultatul** executiei **Radio02**

```
Play Radio
  Playing the station at 93.5 Mhz
Insert Tape
  Tape is in
  Radio is off

Play Radio
  Remove the tape first

Remove Tape
  Tape is out
  Radio is on
Play Radio
  Playing the station at 93.5 Mhz

Play Tape
  Insert the tape first

Insert Tape
  Tape is in
  Radio is off
Play Tape
  Tape is playing
```

## Exemplu de clasa de baza

### Varianta cu rescrierea metodelor si polimorfism: clasa Radio initiala

```
public class Radio {

    // camp (atribut, variabila membru)
    // definire tablou pentru asociere butoane cu frecvente
    protected double[] stationNumber = new double[5];

    // metoda (operatie, functie membru)
    // definire asociere buton cu frecventa
    public void setStationNumber(int index, double freq) {
        stationNumber[index] = freq;
    }

    // metoda (operatie, functie membru)
    // definire selectie frecventa
    public void playStation(int index) {
        System.out.println("Playing the station at " + stationNumber[index] + " Mhz");
    }
}
```

## Exemplu de clasa extinsa

**Varianta cu rescrierea metodelor si polimorfism: Radio ramane nemodificata**

```
public class Combo extends Radio {

    // definire variabila care indica prezenta casetei
    private boolean tapeIn = false;                          // tapeIn = true

    public void insertTape() {
        System.out.println("Insert Tape");
        tapeIn = true;                                        // tapeIn = true
        System.out.println("  Tape is in");
        System.out.println("  Radio is off");
    }

    public void removeTape() {
        System.out.println("Remove Tape");
        tapeIn = false;                                       // tapeIn = false
        System.out.println("  Tape is out");
        System.out.println("  Radio is on");
    }
```

## Exemplu de clasa extinsa

**Varianta cu rescrierea metodelor si polimorfism: Radio ramane nemodificata**

```
    public void playTape() {
        System.out.println("Play Tape");
        if (!tapeIn) {                                          // tapeIn == false
            System.out.println("  Insert the tape first!");
        } else {                                                // tapeIn == true
            System.out.println("  Tape is playing");
        }
    }

    public void playStation(int index) {     // metoda rescrisa (polimorfa)
        System.out.println("Play Radio");
        if (!tapeIn) {                                           // tapeIn == false
            System.out.println("  Playing the station at"+stationNumber[index]+"Mhz");
        } else {                                                 // tapeIn == true
            System.out.println("  Remove the tape first!");
        }
     }
}
```

**Rezultatul** executiei **Radio02 va fi acelasi!** **Insa nu a fost necesara modificarea Radio!**