2010 - 2011

Programare **O**rientata **spre O**biecte
(*Object-Oriented Programming*)

a.k.a. Programare Obiect-Orientata

Titular curs: Eduard-Cristian Popovici

Suport curs: ***http://electronica08.curs.ncit.pub.ro/****course/view.php?id=113*

Suport curs vechi: *http://discipline.elcom.pub.ro/POO-Java/* si
http://electronica07.curs.ncit.pub.ro/course/view.php?id=132

Continut curs Programare Orientata spre Obiecte (in Java)

1. Introducere in abordarea orientata spre obiecte (OO)
- 1.1. Obiectul cursului si relatia cu alte cursuri
- 1.2. Evolutia catre abordarea OO
- 1.3. Caracteristicile si principiile abordarii OO
- 1.4. Scurta recapitulare a programarii procedurale/structurate (introducere in limbajul Java)

2. Orientarea spre obiecte in limbajul Java
- 2.1. Obiecte si clase. Metode (operatii) si campuri (atribute)
- 2.2. Particularitati Java. Clase de biblioteca Java (de uz general)
- 2.3. Clase si relatii intre clase. Asociere, delegare, agregare, compunere
- **2.4. Generalizare, specializare si mostenire**
- 2.5. Clase abstracte si interfete Java
- 2.6. Polimorfismul metodelor
- 2.7. Clase pentru interfete grafice (GUI) din biblioteca Java Swing

3. Programarea la nivel socket cu Java (pe platforma Java SE)
- 3.1. Clase pentru fluxuri de intrare-iesire (IO)
- 3.2. Introducere in Protocolul Internet (IP) si stiva de protocoale IP
- 3.3. Socketuri flux (TCP) Java.
- 3.4. Clase Java pentru programe multifilare. Servere TCP multifilare
- 3.5. Socketuri datagrama (UDP) Java

2. Orientarea spre obiecte in limbajul Java

2.4. Generalizare, specializare si mostenire

Clasa vazuta ca o generalizare (abstractizare) a obiectelor

Clasa

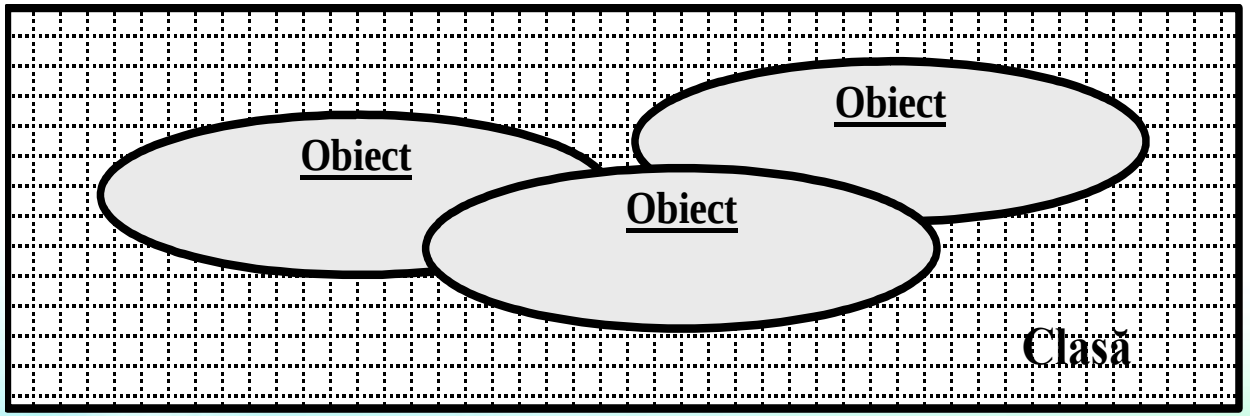
- contine **generalitatile** (abstractiile generale)

Obiectele

- contin **particularitatile** (detaliile particulare)

Clasa

- vazuta ca **multime de obiecte**



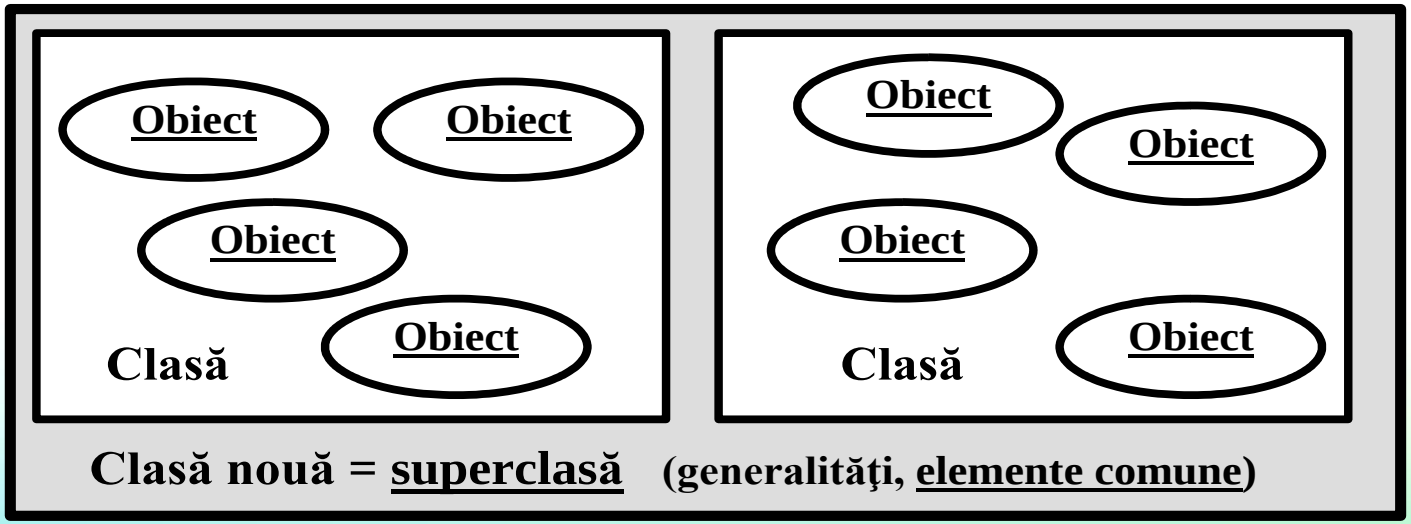
= generalităţi (elemente, aspecte, caracteristici comune)

= particularităţi (elemente, aspecte, caracteristici diferite)

Generalizarea si specializarea

Generalizarea

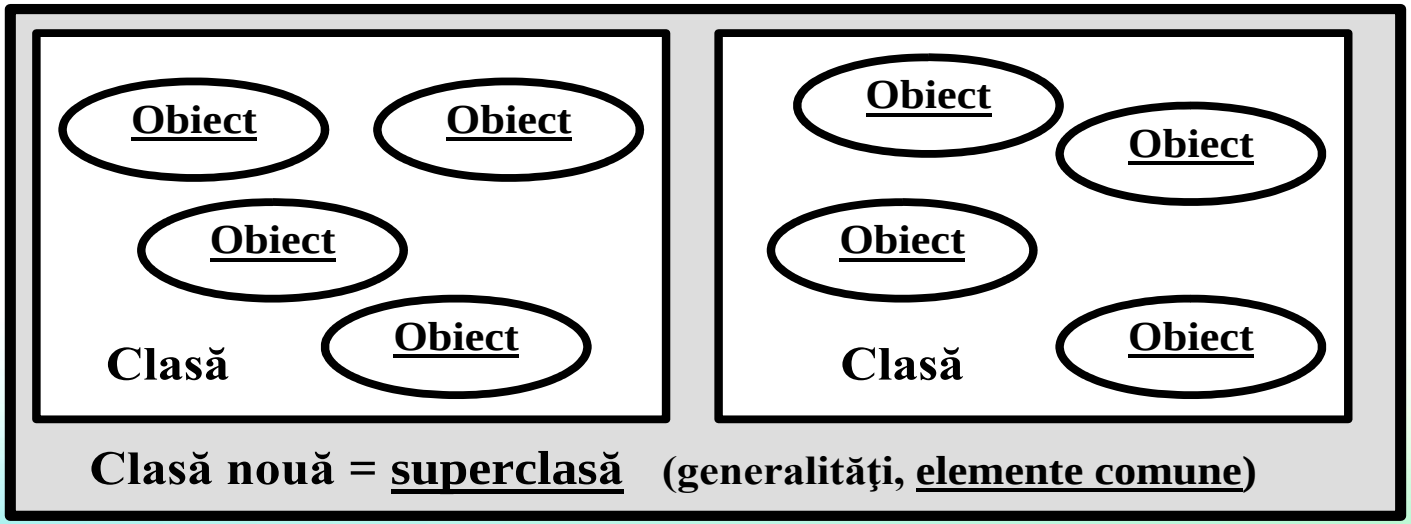
- inseamna extragerea elementelor comune (atribute, operaţii şi constrângeri) ale unui ansamblu de clase
  - într-o **nouă clasă** mai generală, denumită **superclasă** (parinte)
- superclasa este o abstracţie a subclaselor (descendentilor) ei
- rezulta o ierarhie de clase bazata pe generalizare in care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

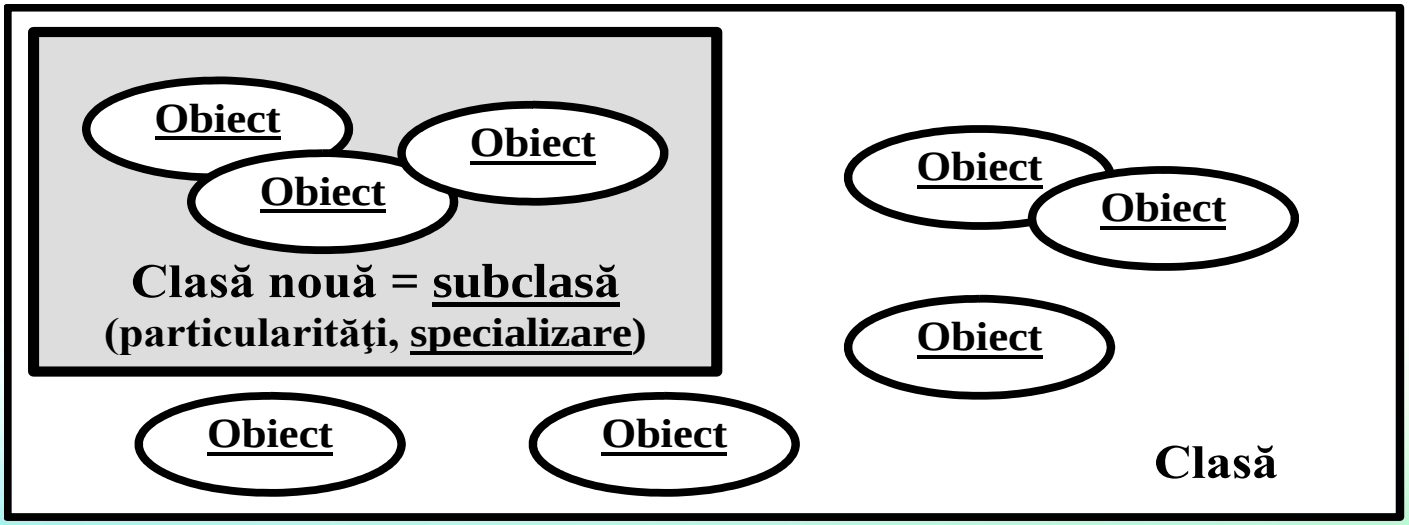
Limbajele care ofera **doar constructii** numite **obiecte** (eventual **clase**) se pot numi

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

Generalizarea si specializarea

Specializarea

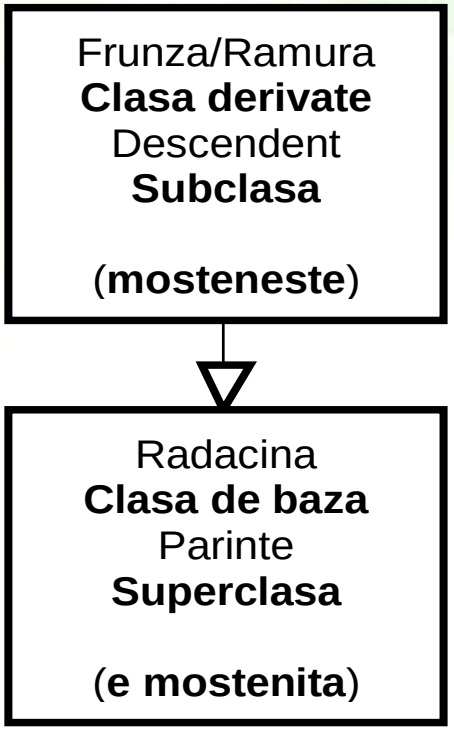
- inseamna capturarea particularităţilor / elementelor distincte ale unui ansamblu de obiecte ale unei clase existente
  - noile caracteristici fiind reprezentate într-o **nouă clasă** mai specializată, denumită **subclasă**
- este utilă pentru extinderea coerentă a unui ansamblu de clase
  - noile cerinţe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



Mostenirea – mecanism suport pentru generalizare/specializare

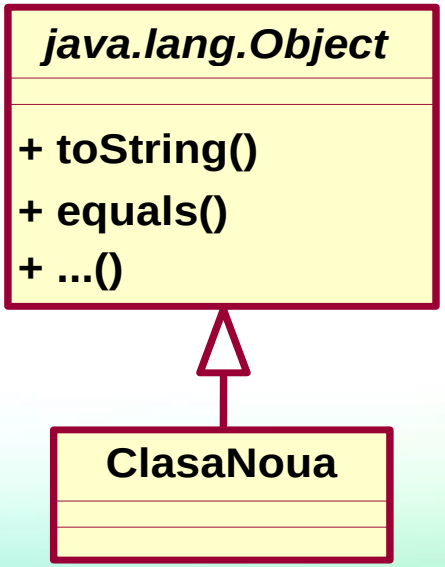
Moştenirea

- 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.)

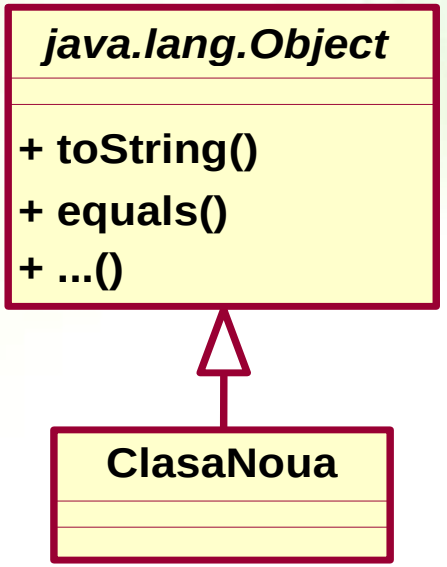


Mostenirea – mecanism suport pentru generalizare/specializare

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 ...
```

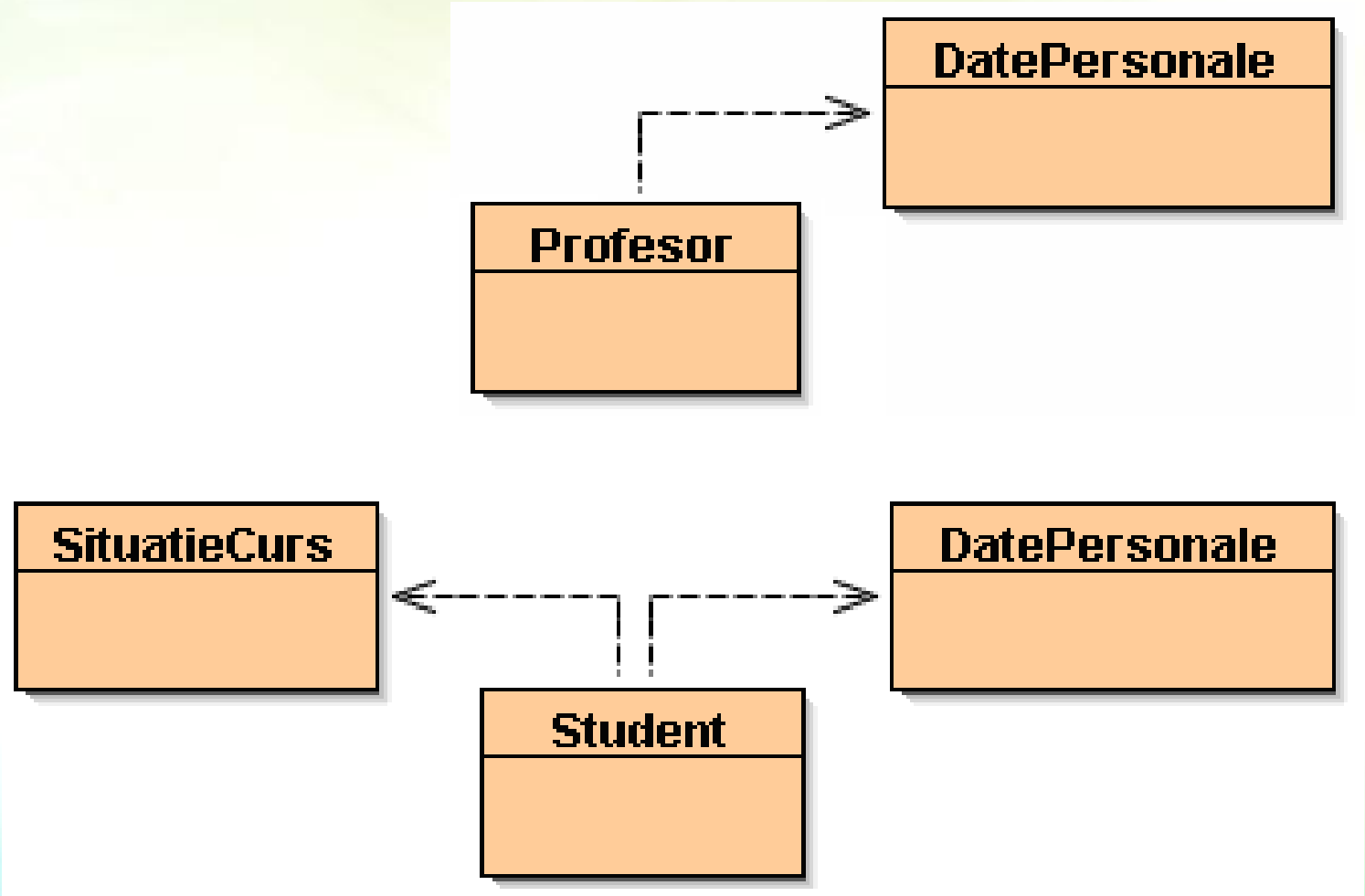
Exemplu de clasa specializata

```
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"
   }
}
```

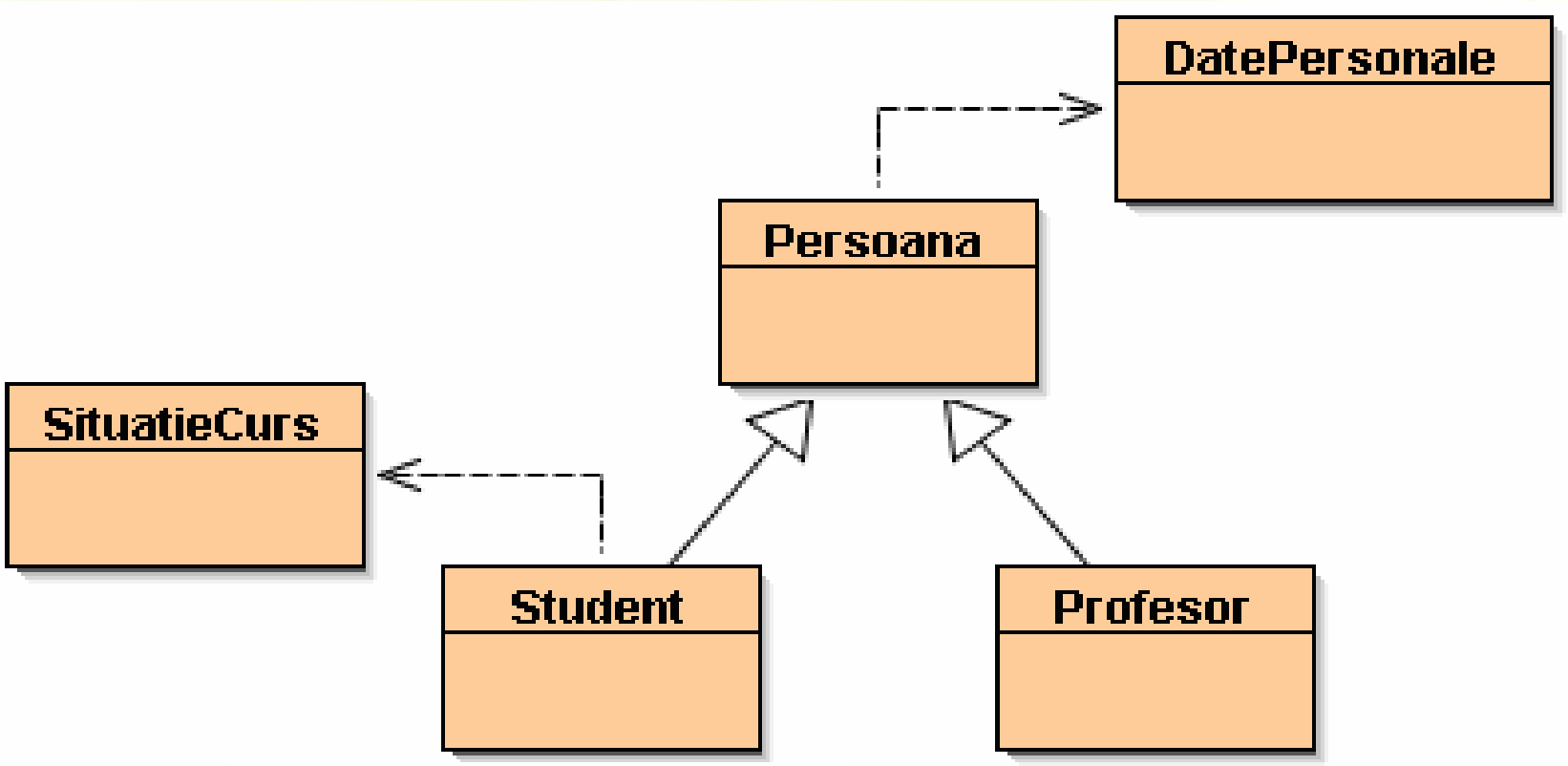
Exemplu de clasa rescrisa pentru a mosteni clasa generala

```
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