**<http://www.ms.sapientia.ro/~manyi/teaching/oop/oop_romanian/curs3/curs3.html>**

**Modelare cu obiecte**

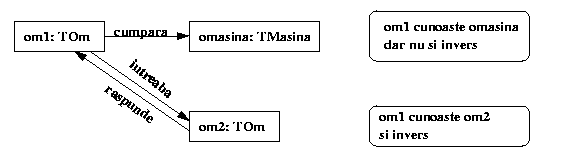
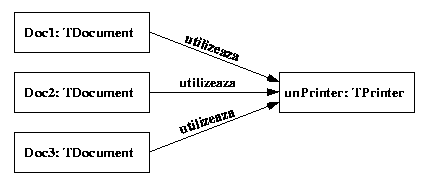
1. Legatura dintre obiecte, diagrame de obiecte
2. Diagrama de clase
3. Mostenire. Legare statica.
4. Exemple

**1. Legatura dintre obiecte, diagrame de obiecte**

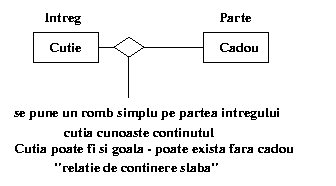
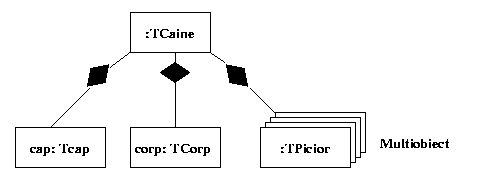
Obiectele pot conlucra doar daca exista legaturi intre ele. Legaturiledintre obiecte sunt de doua feluri:

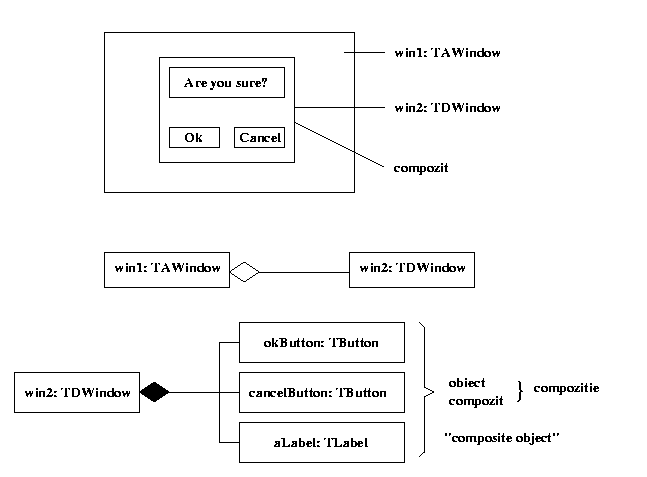
* de cunoastere (*aquaintance relationship*)
* de continere(*aggregation relationship*)

Doua obiecte sunt in legatura de cunoastere, daca existaindependent si cel putin unul cunoaste pe celalalt.  
Doua obiecte sunt in legatura de continere, daca unul dintreele contine fizic pe celalalt. (intreg+parte)  
Exemple de legaturi de cunoastere:

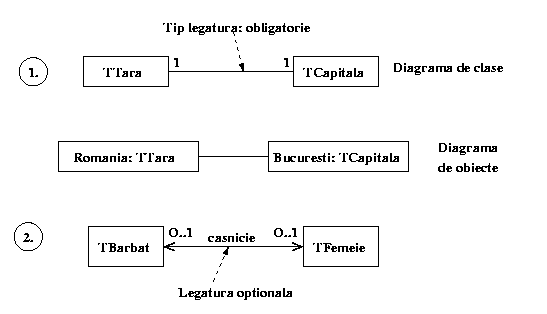
*Relatia de continere:*  
Daca se distruge obiectul container atunci automat se distrug si obiectele continute. Relatia de continere este o relatie mai puternica decat relatia de cunoastere. Relatia de continere poate fi o relatie de continere slaba sau una puternica. In cazul relatiei de continere slaba, intregul isi cunoaste partile, dar poate exista independent de partea continuta. Ca exemplu vom considera un obiect Cutie, care poate sa contina un cadou. Cutia existasi fara cadou. Pentru acest tip de relatie se utilizeaza un romb neumplutpe partea intregului.

In cazul relatiei de continere puternica utilizam un romb umplut tot pepartea intregului, dar in acest caz intregul nu exista independent de partilelui. Ca exemplu consideram obiectele TCaine, TCap, TCorp, TPicior si diagramaurmatoare pentru relatia de continere puternica.  


Diagramele de mai sus se numesc ***diagrame de obiecte.***

**2. Diagrame de clase**



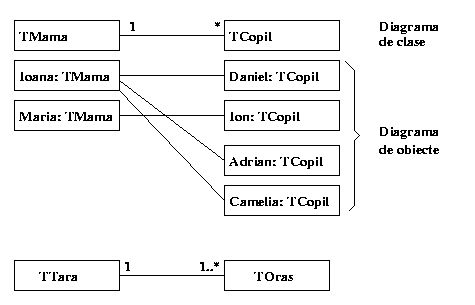
Lagaturile de pe diagramele de mai sus sunt de doua tipuri

1. legaturi obligatorii (intre tara si captala), nu exista tara fara capitala si invers
2. legaturi optionale cum ar fi legatura de casnicie intre barbat si femeie

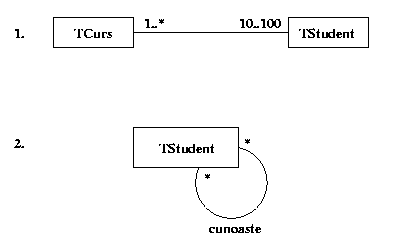
Legaturile dintre clase pot fi clasificate si dupa gradul legaturilor.Aici putem avea urmatioarele situatii:

* legatura unu la unu (*one to one association*)
* legatura unu la mai multe (*one to many association*)
* lagatura mai multe la mai multe(*many to many association*)

Exemplele de mai sus au fost din prima categorie, adica legaturi unu launu.   
Exemple de legaturi unu la mai multe:



Exemple de legaturi "mai multe la mai multe"



In cazul diagramei de mai sus \* inseamna ca numarul cursurilor poate fioricat de mare, iar cursurile se fac doar daca avem cel putin 10 cursanti.Capaciatatea salilor fiind limitata, cursurile se tin maximum pentru 100de studenti. In cazul celei de-a doua diagrame avem o legatura de cunoasterecu semnificatia ca orice student poate sa cunoasca orice alt student.

**Realizarea legaturilor**

Obiectul de tip client intotdeauna trebuie sa cunoasca obiectul de tipserver. Cel care initiaza cererea trebuie sa cunoasca pe cel caruia adreseazacererea. In cazul scrierii aplicatiilor acest lucru se realizeaza cu ajutorulunei  referinte. Exemplu:

class TClient{

    TServer server;

    ...

}

class TServer

    ...

}

**Scrierea unei aplicatii OO**

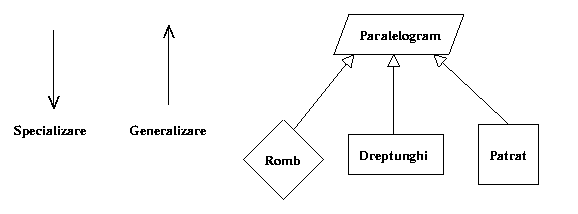
*Pas. 1. Specificarea detaliata a problemei astfel incat programul safie realizabil*  
*Pas. 2. Faza de proiectare+implementare*

* *Proiectare:*
  + *Diagrame de obiecte*
  + *Diagrame  de clase*
* *Implementare:*

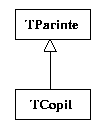
*Implementarea claselor intr-unlimbaj*  
*Pas. 3. Testare*

**Mostenire**

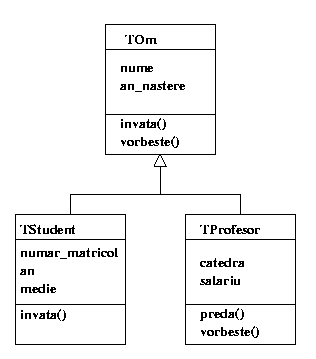
Prima data definim doua concepte: specializarea si generalizarea.  
*Specializarea* este procesul prin care se adauga noi caracteristiciunui obiect existent.  
*Generalizarea* este procesul prin care din descrierea mai multorobiecte scoatem in evidenta cele comune.



In exemplul de mai sus rombul, dreptunghiul si patratul sunt paralelogramespeciale. De exemplu dreptunghiul este un paralelogram cu un unghi dreptunghic.  
Mostenirea este procesul de specializare. Clasa care se extinde senumeste clasa parinte, iar clasa derivata se numeste clasa copil.



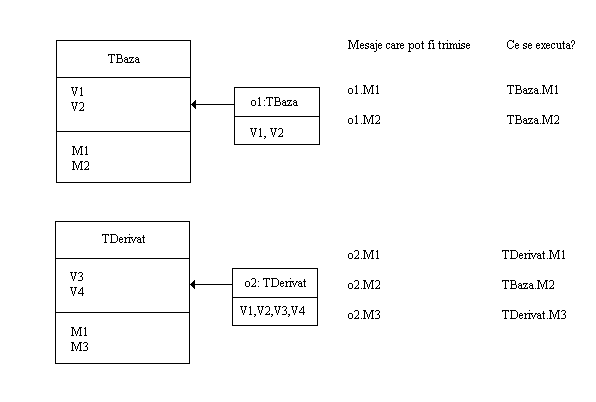
Prin mostenire se adauga noi proprietati unei clase existente si se potredefini anumite caracteristici ai clasei existente.  
Notatia UML pentru mostenire:



In exemplul de mai sus clasa TOm are doua campuri de date, numele si anul nasterii, respectiv doua metode invata() si vorbeste(). Din aceasta clasa de baza derivam doua alte clase noi, clasa TStudent respectiv clasa TProfesor. Aceste clase noi adauga alte campuri de date si metode respectiv specializeaza anumite metode ale clasei de baza.  
Astfel clasa TStudent va avea cinci campuri de date: nume, anul nasterii, numar matricol, an de studii, medie si doua metode inavata() si vorbeste(). Metoda invata() a fost specializata (redefinita) deoarece un student invata altfel decat un om obisnuit. Probabil ca invata mai mult si invata in anumite perioade ale anului. In cazul clasei TProfesor apar doua date in plus fata de clasa parinte. Apare catedra la care apartine si salariul pe care primeste pentru activitatea desfasurata. Clasa defineste o noua metoda preda(),care caracterizeaza numai obiectele de tip TProfesor si redefineste metodevorbeste() deoarece un profesor ar trebui sa vorbeasca mai clar si maicoerent decat un om obisnuit. In concluzie prin mostenire putem face urmatoarele:

* *se adauga campuri de date noi*
* *se adauga metode noi*
* *se redefinesc anumite metode ale clasei parinte*

Observatie: Daca se adauga campuri de date noi, atunci trebuie adaugatesi metode noi care realizeaza accesul la aceste date.  
Exista limbaje de programare care permit si mostenirea multipla cumar fi C++, iar altele care permit doar derivarea de la o singura clasaparinte: Java, Smalltalk, Delphi.



**Apelul constructorilor in cazul mostenirii**

Consideram exemplul urmator:  
class A{  
   A(){  
       System.out.println("Constructorul clasei A");  
   }  
}  
  
class B extends A{  
   B(){  
       System.out.println("Constructorul clasei B");  
   }  
}  
  
  
public class p4{  
   public static void main( String args[] )  
   {  
       B b = new B();  
  
   }  
}

Rezultatul executiei va fi:

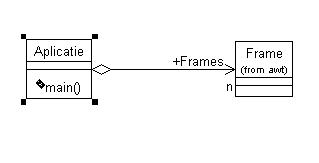
Constructorul clasei A  
Constructorul clasei B

Deci putem afirma ca constructorul suprclasei se apeleaza intotdeauna automat, daca aceasta are un constructor fara argumente. In caz contrar trebuie apelat explicit furnizandu-i parametrii corespunztori. Exemplul urmator ilustreaza acest lucru:  
  
class A{  
   A( String a){  
       System.out.println("Constructorul clasei A "+a);  
   }  
}  
  
class B extends A{  
   B( String b){  
       super( b );  
       System.out.println("Constructorul clasei B "+b);  
   }  
}  
  
  
public class p5{  
   public static void main( String args[] )  
   {  
       B b = new B("Hi");  
  
   }  
}

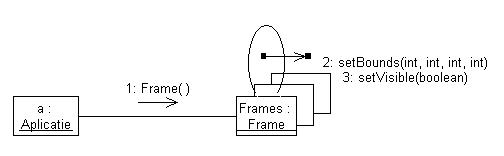
**Legare statica**

Legare statica inseamna ca adresa metodei se cunoaste inca din timpul compilarii. Deci in cazul unei constructii nume\_obiect.metoda() compilatorul stie exact care metoda va fi executata. Daca clasa careia apartine obiectul nu areo asemenea metoda atunci compilatorul cauta  metoda in ierarhia declase. Se va apela metoda din clasa cea mai apropiata cu acelasi nume    Aceasta cautare este facuta in faza compilarii si acest tip de legare senumeste legare statica.

**Exemplu 1:** Sa se scrie o aplicatie care afiseaza pe ecran treiferestre cu titlurile: Unu, Doi, Trei si cu dimensiunile 100x100.   
**Diagrama de clase:**



**Diagrama de colaborare:**



**Codul sursa Java:**

//Source file: Aplicatie.java

import java.awt.Frame;

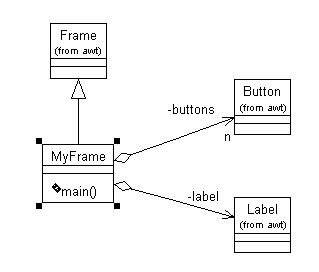
public class Aplicatie  
{  
   public Frame Frames[];

   public Aplicatie()  
   {  
      int i;  
      String titles[]={"Unu", "Doi","Trei"};  
      Frames = new Frame[ 3 ];  
      for(i=0;i<3;i++)   
      {  
          Frames[i ] = new Frame(titles[ i ]);  
          Frames[i ].setBounds(100\*i,100\*i,100,100);  
          Frames[i ].setVisible( true );  
      }

   }

   public static void main(String[] args)   
   {  
    Aplicatie a = new Aplicatie();   
   }  
}

**Exemplu 2:** Sa se scrie o aplicatie care se ruleaza intr-o fereastra.Fereastra va contine o eticheta si doua butoane.



**Sursa Java:**

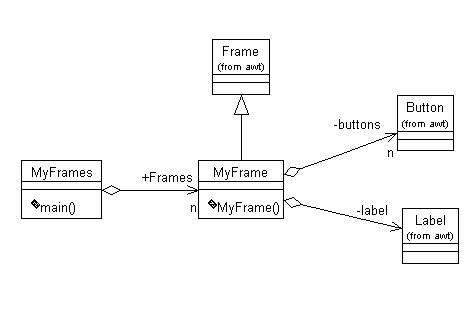
import java.awt.Frame;  
import java.awt.Button;  
import java.awt.Label;

public class MyFrame extends Frame  
{  
   private Button buttons[];  
   private Label label;

   public MyFrame()  
   {  
    // Crearea butoanelor si adaugarea lor la container  
    buttons = new Button[ 2 ];  
    buttons[ 0 ]= new Button("Buton 1");   
    add( buttons[ 0 ], "North" );   
    buttons[ 1 ]= new Button("Buton 2");   
    add( buttons[ 1 ],"South" );  
    // Crearea etichetei si adaugarea ei la container  
    label = new Label("Eticheta");   
    add( label, "Center" );  
   }

   public static void main(String[] args)   
   {  
     MyFrame f = new MyFrame();   
     f.setBounds(10,10, 300, 300 );  
     f.setVisible( true );  
   }  
}

**Exemplu 3**: Sa se scrie o aplicatie care afiseaza cinci ferestrede tip MyFrame astfel incat sa nu se suprapuna.



**Sursa Java:**

//Source file: MyFrames.java

public class MyFrames  
{  
   public MyFrame Frames[];

   public MyFrames()  
   {  
    int i;  
    Frames = new MyFrame[5];  
    for( i=0; i<5; i++)  
    {  
      Frames[ i ] = new MyFrame(Integer.toString( i ));  
      Frames[ i ].setBounds(i\*100,i\*100,50,50);   
      Frames[ i ].setVisible( true );  
    }  
   }

   public static void main(String[] args)   
   {  
    MyFrames f = new MyFrames();  
   }  
}  
   
 