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

1. **Introducere in abordarea orientata spre obiecte (OO)**

   1.3. Caracteristicile si principiile abordarii OO

Orientarea spre Obiecte (OO)

- **Orientarea** spre
  - **modelarea** (abstractizarea) informatica a **realitatii**
  - entitati bazate pe **responsabilitati** (roluri)
  - **incapsulare duala** (a structurilor de date si de comportament)
  - **mentinerea** si **ascunderea** unei **stari** interne
  - **colaborare** intre entitati (comunicare prin **mesaje**, interactiune **sociala**)
  - definirea unor **interfete** contractuale, componente ***black-box***, etc.

Modelarea si abstractizarea OO

Programul (sistemul software) **orientat spre obiecte**

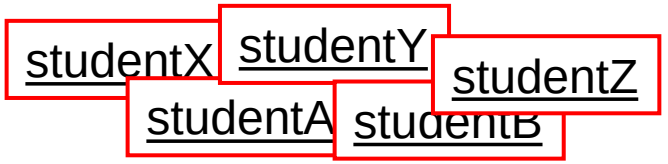
– reprezinta un **model informatic**

– al unei **parti** din **lumea reala**

– **elementele** care compun modelul

– sunt **construite prin analogie** cu **entitati care apar in lumea reala** (obiecte reale, concepte)

– sunt numite **obiecte software**



– trebuie **reprezentate** in limbajul de programare

Care sunt **constructiile** software care permit **reprezentarea obiectelor** software (similare) in limbaje de programare?

Clasificarea OO – entitati bazate pe responsabilitati (roluri)

Ca si in cazul obiectelor si conceptelor din lumea reala

- **obiectele** software pot fi **categorisite** (clasificate)
- **categoriile** (numite **clase**) **corespunzand**
  - diferitelor **responsabilitati** pe care le au
  - sau diferitelor **roluri** pe care le joaca **entitatile din lumea reala**
    - din care sunt **construite prin analogie** **obiectele** software

Clasele

- sunt constructiile software care
  - reprezinta **obiectele** software care au **responsabilitati / roluri similare**

Clasificarea OO – abstractizarea multimilor (AM) de obiecte

Clasa (**obiectelor** software)

– este o **constructie software complexa**

– **asemanatoare** structurilor de date din C (**struct**)

– **evoluata din ADT** (concretizare a ADT)

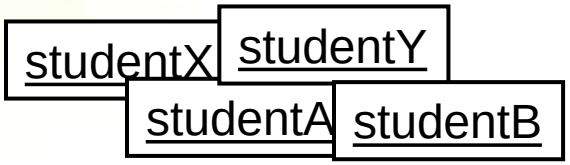
– si astfel **tip de date** cu **incapsulare duala**

– care **descrie** intr-o **forma abstracta**

– toate **obiectele software** de un **tip particular**

(care au **responsabilitati / roluri similare**)

obiecte software



AM ⇩ (**AOO**)

Student

Clasa (categorie, tip)

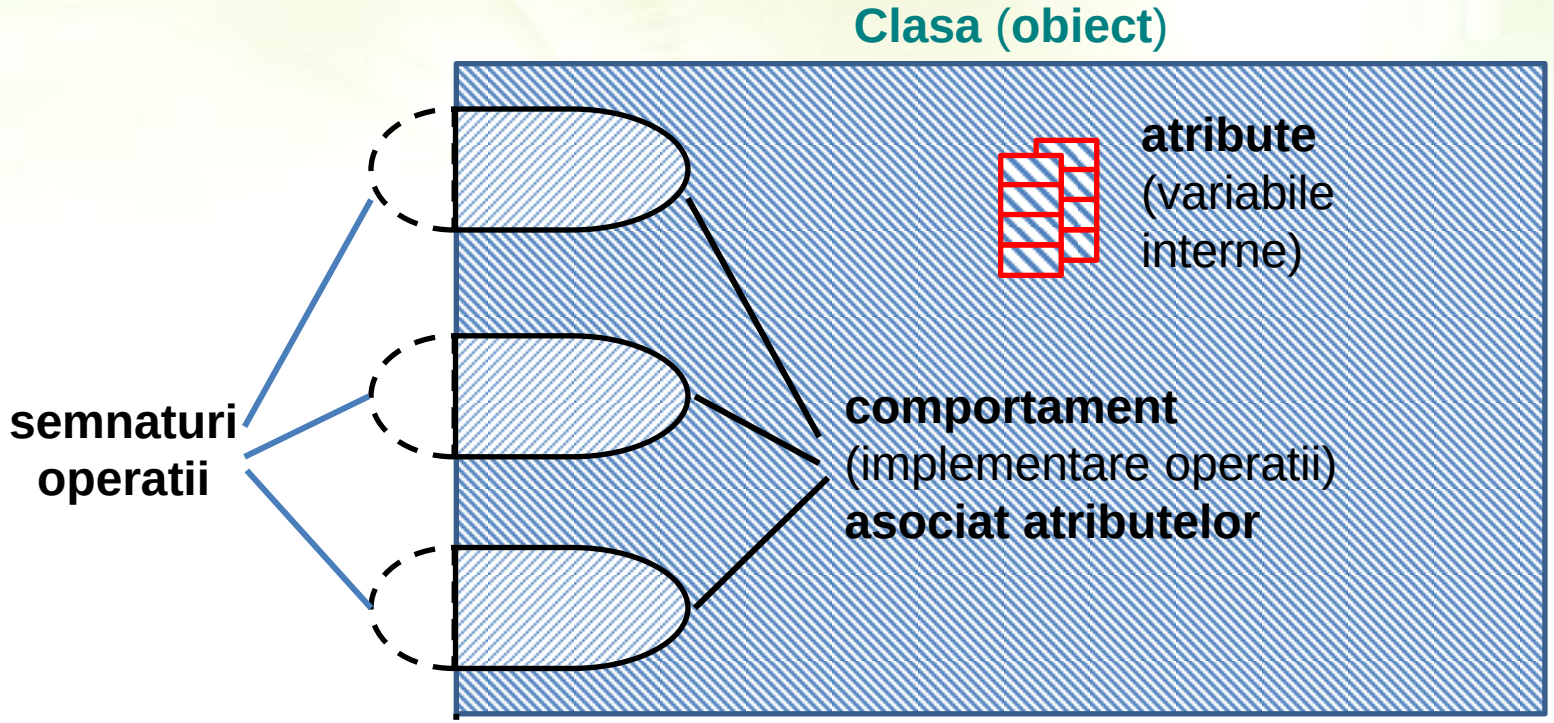
Clasificarea OO – incapsularea duala cu ascunderea detaliilor

Clasa fiind o constructie software **evoluata din ADT**

- **realizeaza o incapsulare** cu ascundere a detaliilor **duala**
  - a **informatiilor** (**datelor**)
    - **regrupand** elementele de date
      - numite **atribute**
    - implicit **CU ascunderea detaliilor** **(datelor, informatiilor)** !
  - a **comportamentului**
    - **regruparea** elementelor de comportament
      - numite **operatii**
    - implicit **CU ascunderea detaliilor** **(implementarii)**
- «**transmite**» **obiectelor** aceasta **incapsulare** cu ascundere a detaliilor **duala**

Incapsularea duala cu ascunderea detaliilor

Incapsularea oferita de clase (si **obiecte**) este o forma de **incapsulare duala**



Interfata (setul de semnaturi ale operatiilor) – ideal este **publica, vizibila, accesibila**

Reprezentarea datelor si implementarea (setul de atribute si setul de implementari ale operatiilor) – ideal sunt **private, ascunse, inaccesibile**

Incapsularea duala cu ascunderea detaliilor

Incapsularea OO – oferita de clase si **obiecte**

- **–** inseamna **ascunderea detaliilor** interne
  - **–** reprezentarea **datelor** (set de **atribute**)
  - **–** si **implementarea** (set de **coduri ale operatiilor**)

in spatele unei **interfete publice** (set de **semnaturi ale operatiilor**)

Exemplu in Java

```
public class Student {
  // Campuri (atribute) private (inaccesibile codurilor exterioare)
  private String nume;
  private String[] cursuri;
  private int[] rezultate;

  // Metode (operatii) publice (accesibile tuturor codurilor exterioare)
  // Metoda stabilire nume
  public void setNume(String n)          { nume = n; }

  // Metoda stabilire cursuri
  public void setCursuri(String[] c)     { cursuri = c; }

  // Metoda stabilire rezultate
  public void setRezultate(int[] r)      { rezultate = r; }

  // Metoda obtinere nume
  public String getNume()                { return (nume); }

  // Metoda obtinere cursuri
  public String[] getCursuri()           { return (cursuri); }

  // Metoda obtinere rezultate
  public int[] getRezultate()            { return (rezultate); }
}
```

Informatii ascunse = stare ascunsa (campuri private)

Interfata publica = servicii oferite (semnaturi metode)

Implementare ascunsa = comportament ascuns (coduri interne ale metodelor)

Incapsularea duala cu ascunderea detaliilor

```
// Incapsuleaza informatiile si comportamentul unui Student.
public class Student {
  // Campuri (atribute) private (inaccesibile codurilor exterioare)
  private String nume;
  private String[] cursuri;
  private int[] rezultate;

  // Metode (operatii) publice (accesibile tuturor codurilor exterioare)
  // Metoda stabilire nume
  public void setNume(String n)          {  nume = n;  }

  // Metoda stabilire cursuri
  public void setCursuri(String[] c)     {  cursuri = c;  }

  // Metoda stabilire rezultate
  public void setRezultate(int[] r)      {  rezultate = r;  }

  // Metoda obtinere nume
  public String getNume()                {  return (nume);  }

  // Metoda obtinere cursuri
  public String[] getCursuri()           {  return (cursuri);  }

  // Metoda obtinere rezultate
  public int[] getRezultate()            {  return (rezultate);  }
}
```

Informatii ascunse = stare ascunsa (campuri private)

Interfata publica = servicii oferite (semnaturi metode)

Implementare ascunsa = comportament ascuns (coduri interne ale metodelor)

Exemplu in Java

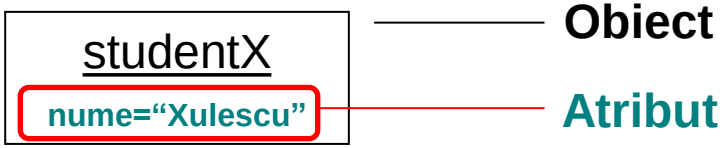
Clasificarea OO – incapsularea duala cu ascunderea detaliilor

Obiectele si conceptele sunt **categorisite / clasificate**

– atat **in lumea reala** cat si **in cea informatica** (software)

– pe baza **caracteristicilor esentiale** pe care le au (rezultate in urma abstractizarilor)

– **atribute** – elemente de **date**, variabile interne care caracterizeaza obiectele



– **operatii** – elemente de **comportament**, proceduri care pot fi efectuate asupra **atributelor**

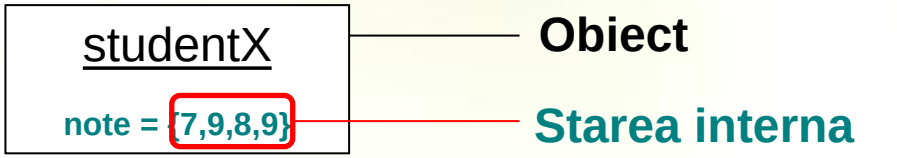
Clasele

– reprezinta **obiecte** software care au **caracteristici** esentiale (**atribute si operatii**) **similare**

Mentinerea si ascunderea unei stari interne

Ansamblul valorilor atributelor unui **obiect** la un **moment dat**

– reprezinta **starea** **interna** obiectului



Atributele

– sunt astfel “variabilele de stare” ale obiectului

– sunt prin natura lor **private, ascunse, inaccesibile**

– si dau astfel **starii** **obiectelor**

– calitatea de a fi **privata, ascunsa, inaccesibila** **direct**

– ea putand fi obtinuta **din exterior** doar **indirect** (**controlat**)

– **prin** intermediul apelurilor la **operatii**

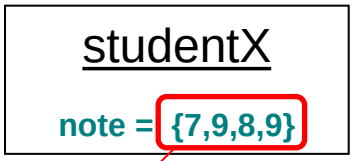
Mentinerea si ascunderea unei stari interne

Starea unui obiect **poate varia in timp**

– ca urmare a **comportamentului**

– care este **rezultatul executiei operatiilor**

– prin **apeluri** venite **de la alte obiecte**



Starea interna
la un **moment dat**
(initial)

timp

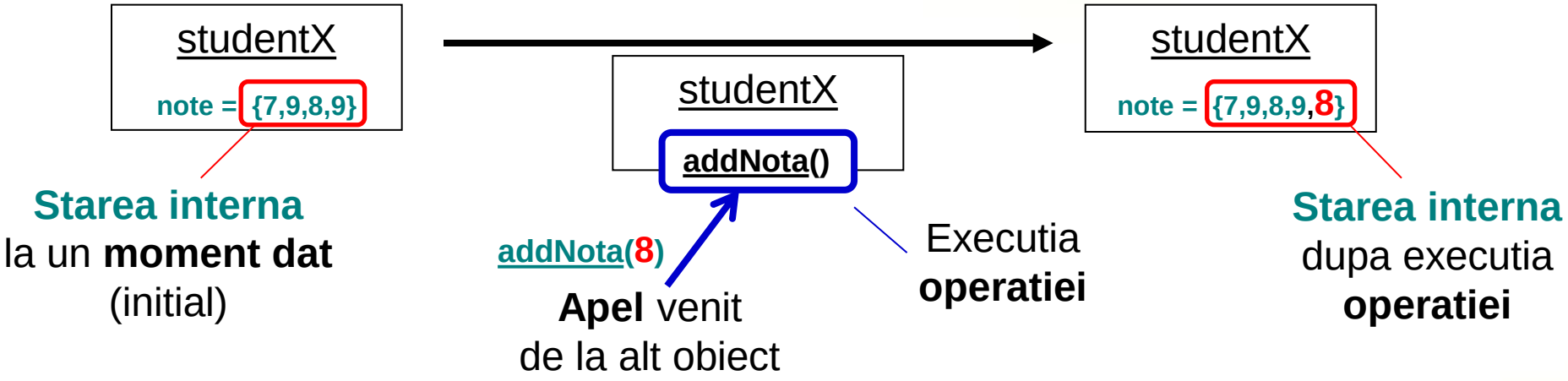
Mentinerea si ascunderea unei stari interne

Starea unui obiect **poate varia in timp**

– ca urmare a **comportamentului**

– care este **rezultatul** executiei **operatiilor**

– prin **apeluri** venite **de la alte obiecte**



Mentinerea si **ascunderea** unei **stari interne** sunt **calitati noi ale programelor**

– introduse de **orientarea spre obiecte**

Colaborarea intre entitati – comunicarea prin mesaje

In **lumea reala**

– **sarcinile** sunt realizate in **colaborare intre entitati** diverse

Abordarile actuale

– **OOP** – programarea **orientata spre obiecte** (derivata din conceptul **ADT**)

– **CBD** – dezvoltarea **bazata pe componente** (***black box***)

– **SOA** – **orientarea spre servicii** a arhitecturilor

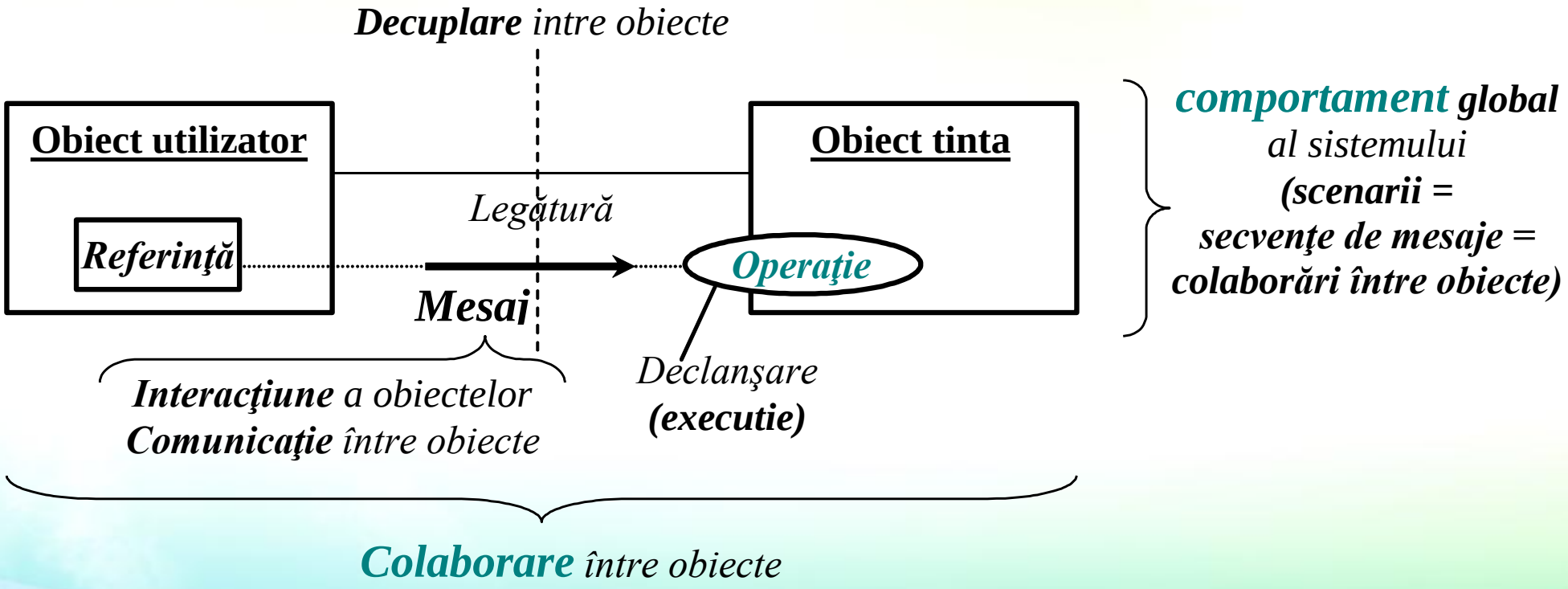
imita **modelele sociale, colaborative**

Colaborarea intre entitati – comunicarea prin mesaje

Comportamentul si **operatiile** (**functiile, procedurile**) in abordarea OO

– se reprezinta prin **forme de colaborare** (**comunicare**, **interactiune**)

– intre **obiectele** ce compun **programul** (sistemul software)

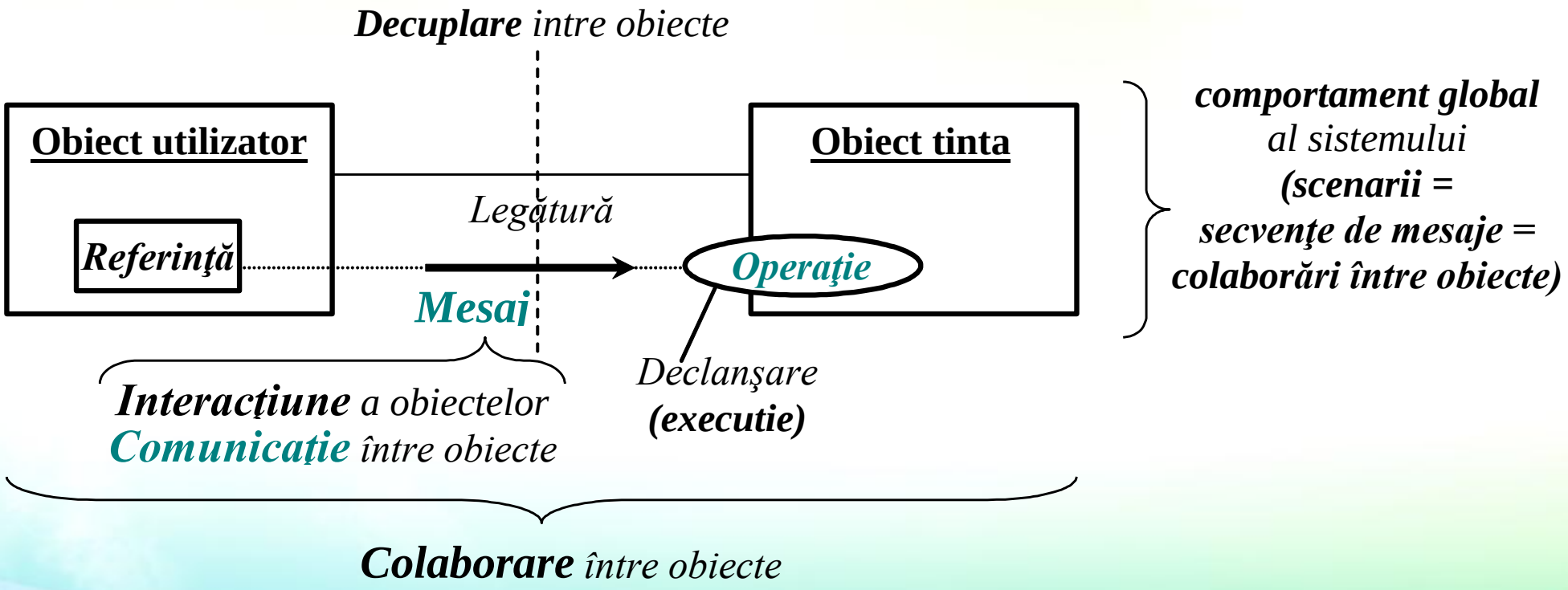


Colaborarea intre entitati – comunicarea prin mesaje

Operatiile (**functiile, procedurile**) in abordarea OO

– sunt **unitati de comunicare** (**interactiune**)

– numite **mesaje**



Abordarea orientata spre obiecte a dezvoltarii programelor

Constructia unui **program de calcul** (sistem *software*)

– este o **secventa** de **iteratii** de tip **divizare-reunire**, fiind necesare:

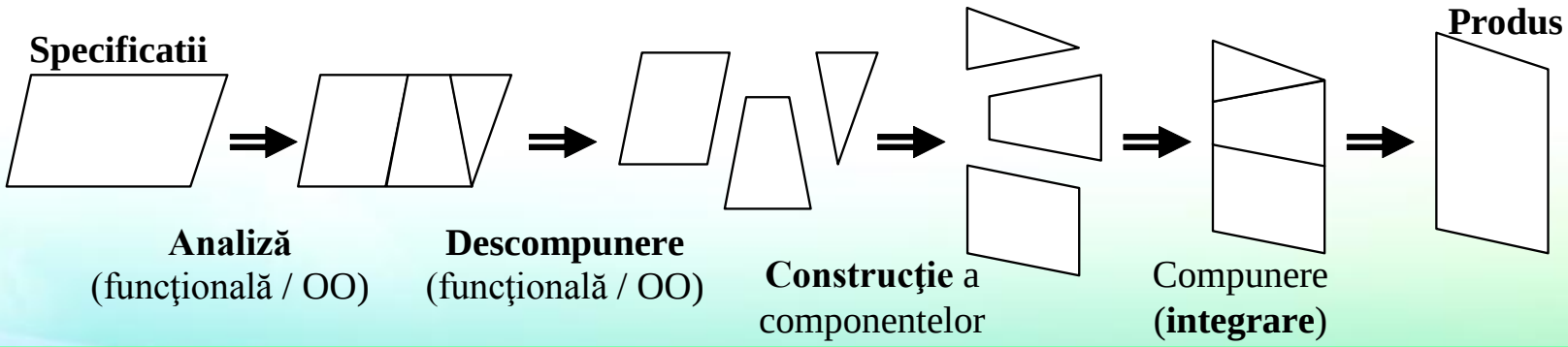
– **descompunerea** (**analiza**) pentru

– a **intelege problema** si

– a putea formula o **conceptie** a **solutiei**

– **compunerea** (**sinteza**) pentru

– a **construi solutia** (a materializa, a realiza efectiv conceptia)



Abordarea orientata spre obiecte a dezvoltarii programelor

Abordarea orientata spre obiecte (OO)

– propune **descompunerea** bazată pe

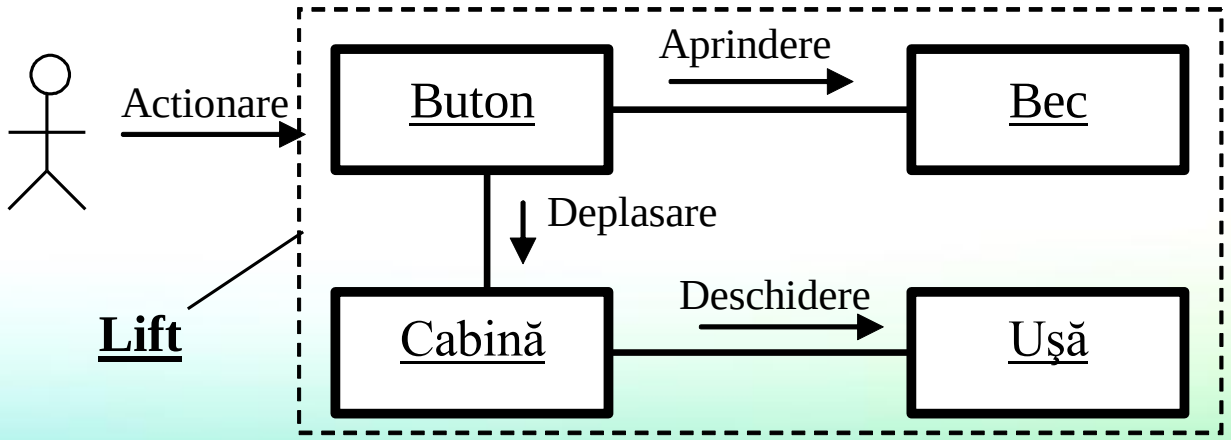
– **responsabilitati** si **delegare de responsabilitati** (nu doar functionala)

– **integrarea** a ceea **ce este** (structural) şi ceea **ce face** (comportamental) sistemul (nu doar a ceea ce face)

Cuplajul intre obiecte

– este obtinut prin trimiterea de **mesaje** (apelurile de **operatii**)

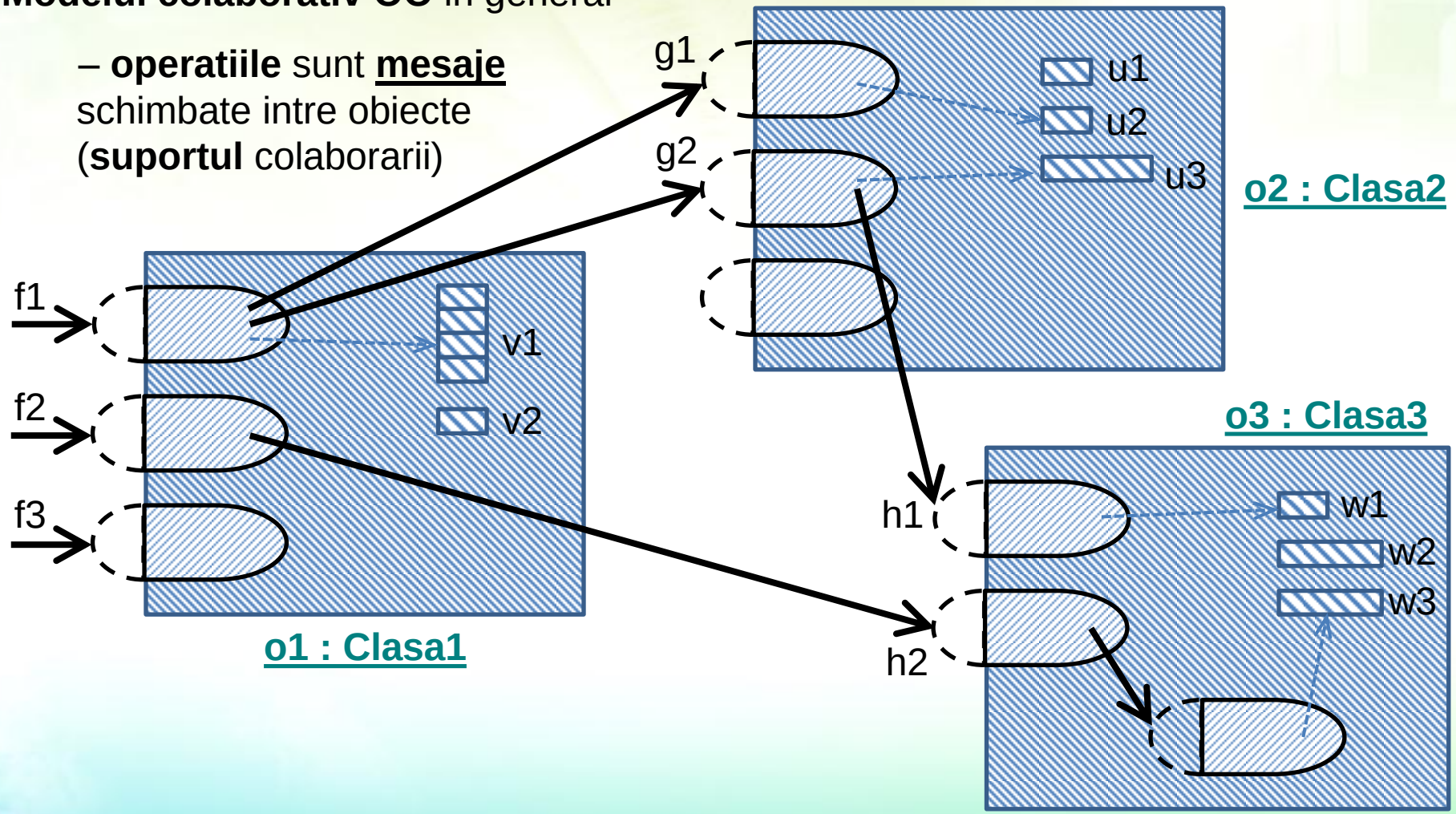
– si astfel **dinamic**



Abordarea orientata spre obiecte a dezvoltarii programelor

Modelul colaborativ OO in general

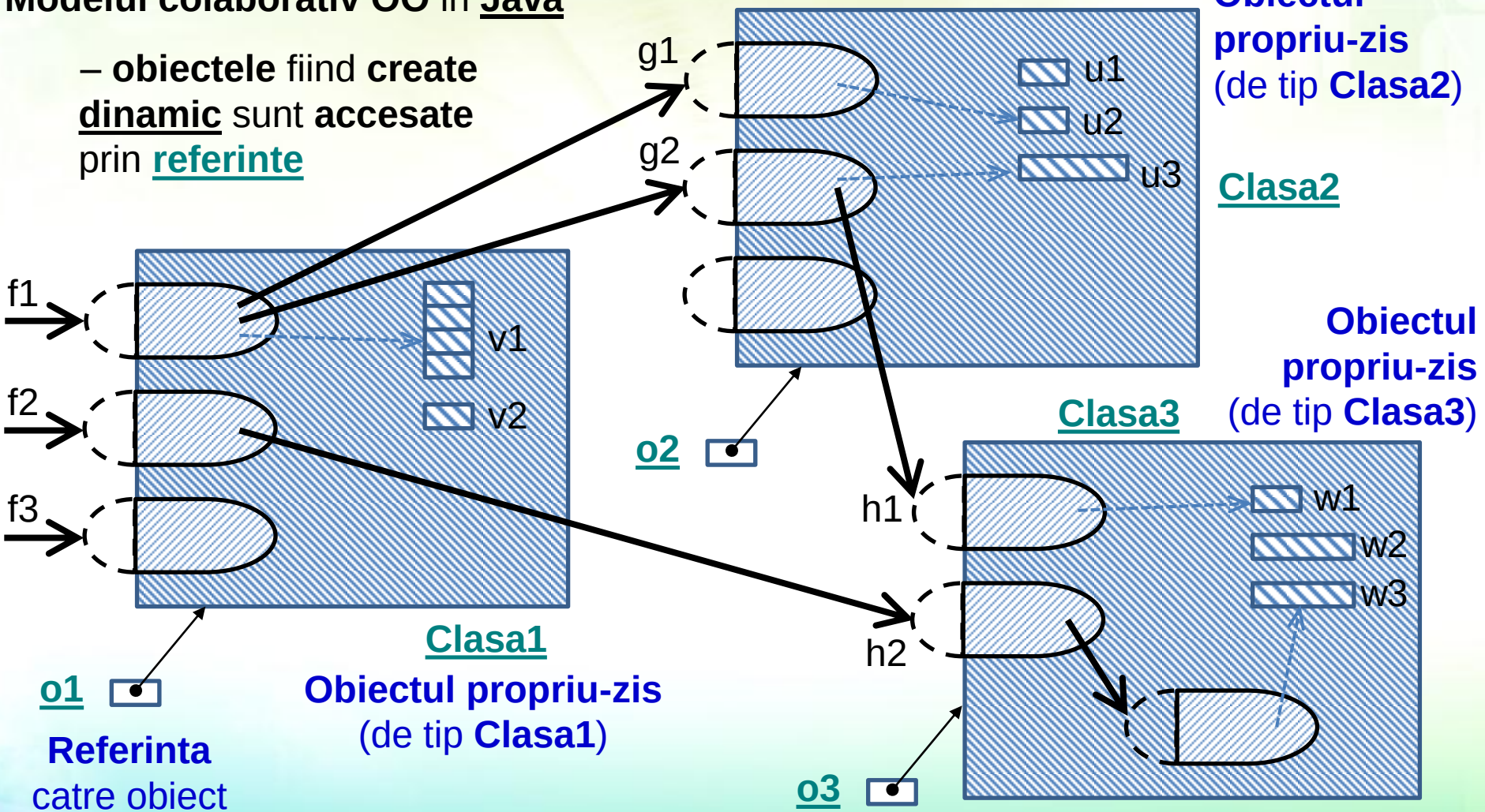
– **operatiile** sunt **mesaje** schimbate intre obiecte (**suportul** colaborarii)



Abordarea orientata spre obiecte a dezvoltarii programelor

Modelul colaborativ OO in **Java**

– **obiectele** fiind **create dinamic** sunt **accesate** prin **referinte**



Abordarea orientata spre obiecte a dezvoltarii programelor

Exemple de cod Java

```
System.out.println(..);
```

`System` = **clasa** Java din pachetul **java.lang** (importat implicit)

`out` = **atribut public static al clasei System**
(**variabila partajata de toate obiectele** clasei System)

= **referinta** catre **obiect din** clasa **PrintStream**

d.p.d.v. al clasificarii

d.p.d.v. structural

`println(..)` = **metoda a obiectelor** clasei **PrintStream**

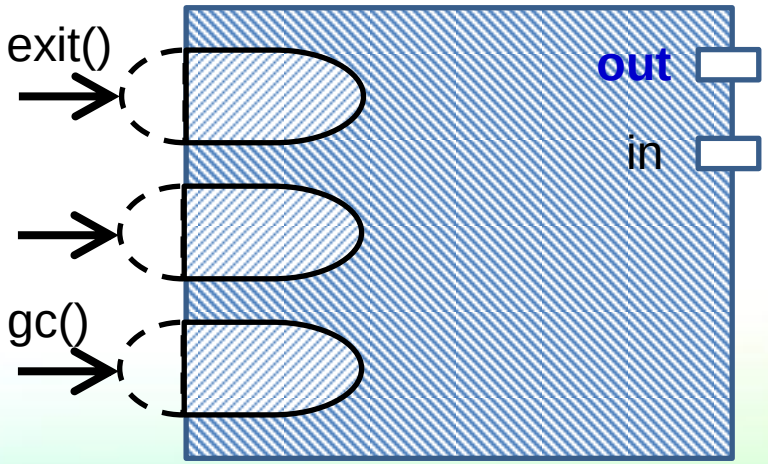
Abordarea orientata spre obiecte a dezvoltarii programelor

Exemple de cod Java

```
System.out.println(..);
```

```
// clasa Java din pachetul java.lang
public class System {

  // atribut public static al clasei System
  // obiect din clasa PrintStream
  public static PrintStream out;

  // .. restul codului clasei System
}
```

Clasa utilitara care **incapsuleaza facilitatile** portabile ale **sistemelor de operare** (apeluri “sistem” cum e exit(), console I/O)



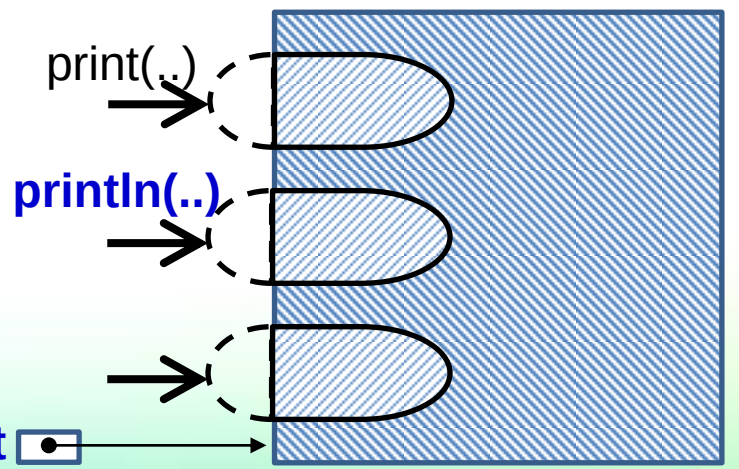
Abordarea orientata spre obiecte a dezvoltarii programelor

Exemple de cod Java

```
System.out.println(..);
```

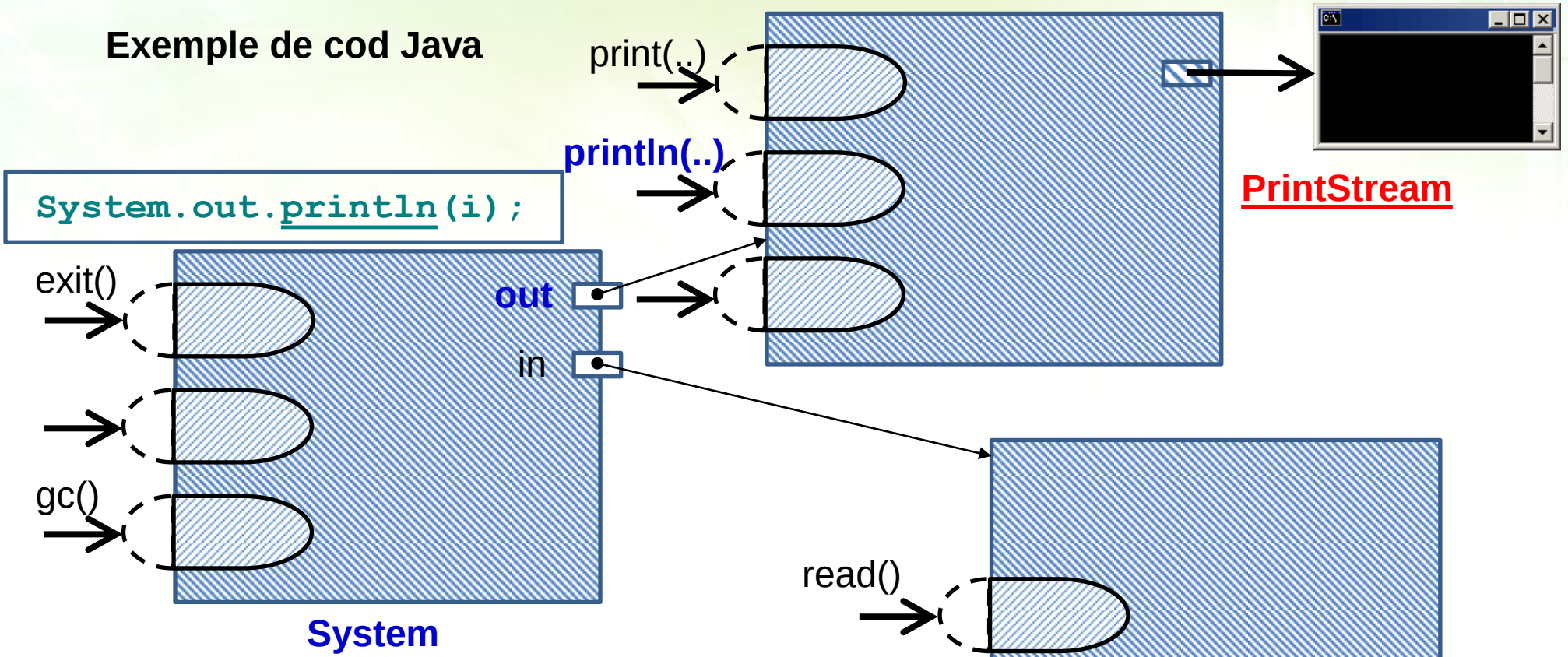
```
// clasa Java din pachetul java.io
public class PrintStream {

  // metoda a obiectelor clasei PrintStream
  public void println(..) {

    // .. codului metodei println()

  }

  // .. restul codului clasei PrintStream
}
```

Obiect al unei clase flux de iesire care **incapsuleaza** fluxul de date catre **consola standard de iesire**)



Abordarea orientata spre obiecte a dezvoltarii programelor

Exemple de cod Java



System = **clasa** Java
out = **atribut public static al clasei System**
= **obiect din** clasa **PrintStream**
println() = **metoda a obiectelor** clasei **PrintStream**

Abordarea orientata spre obiecte a dezvoltarii programelor

Exemple de cod Java

```
int i = System.in.read();
```

`System` = **clasa** Java din pachetul **java.lang** (importat implicit)

`in` = **atribut public static al clasei System**
(**variabila partajata de toate obiectele** clasei System)

= referinta catre **obiect din** clasa **InputStream**

d.p.d.v. al clasificarii

d.p.d.v. structural

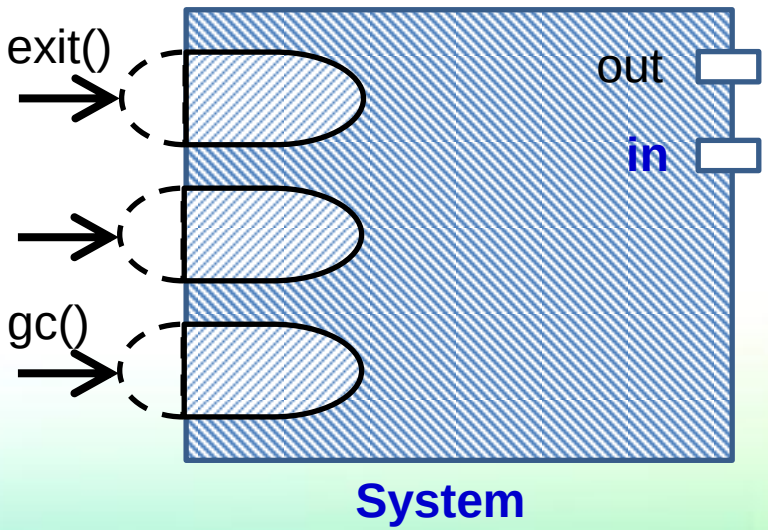
`read()` = **metoda a obiectelor** clasei **InputStream**

Abordarea orientata spre obiecte a dezvoltarii programelor

Exemple de cod Java

```
int i = System.in.read();
```

```
// clasa Java din pachetul java.lang
public class System {

  // atribut public static (partajat de obiecte) al clasei System
  // obiect din clasa InputStream
  public static InputStream in;

  // .. restul codului clasei System
}
```



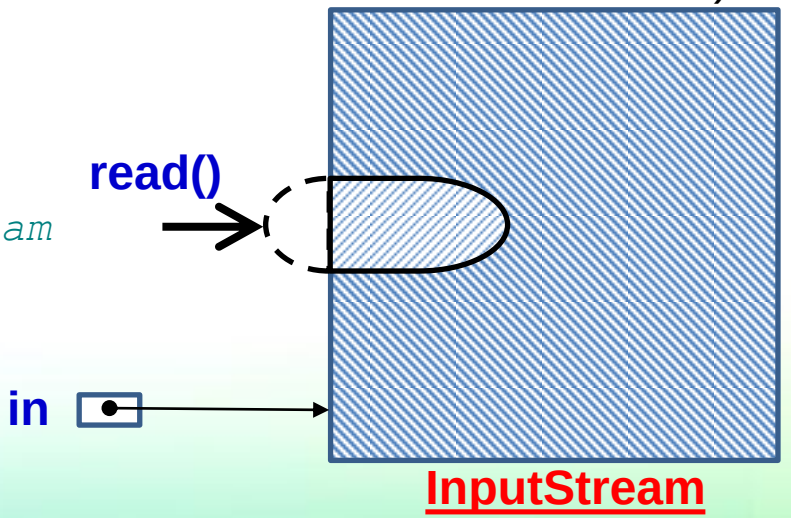
Abordarea orientata spre obiecte a dezvoltarii programelor

Exemple de cod Java

```
int i = System.in.read();
```

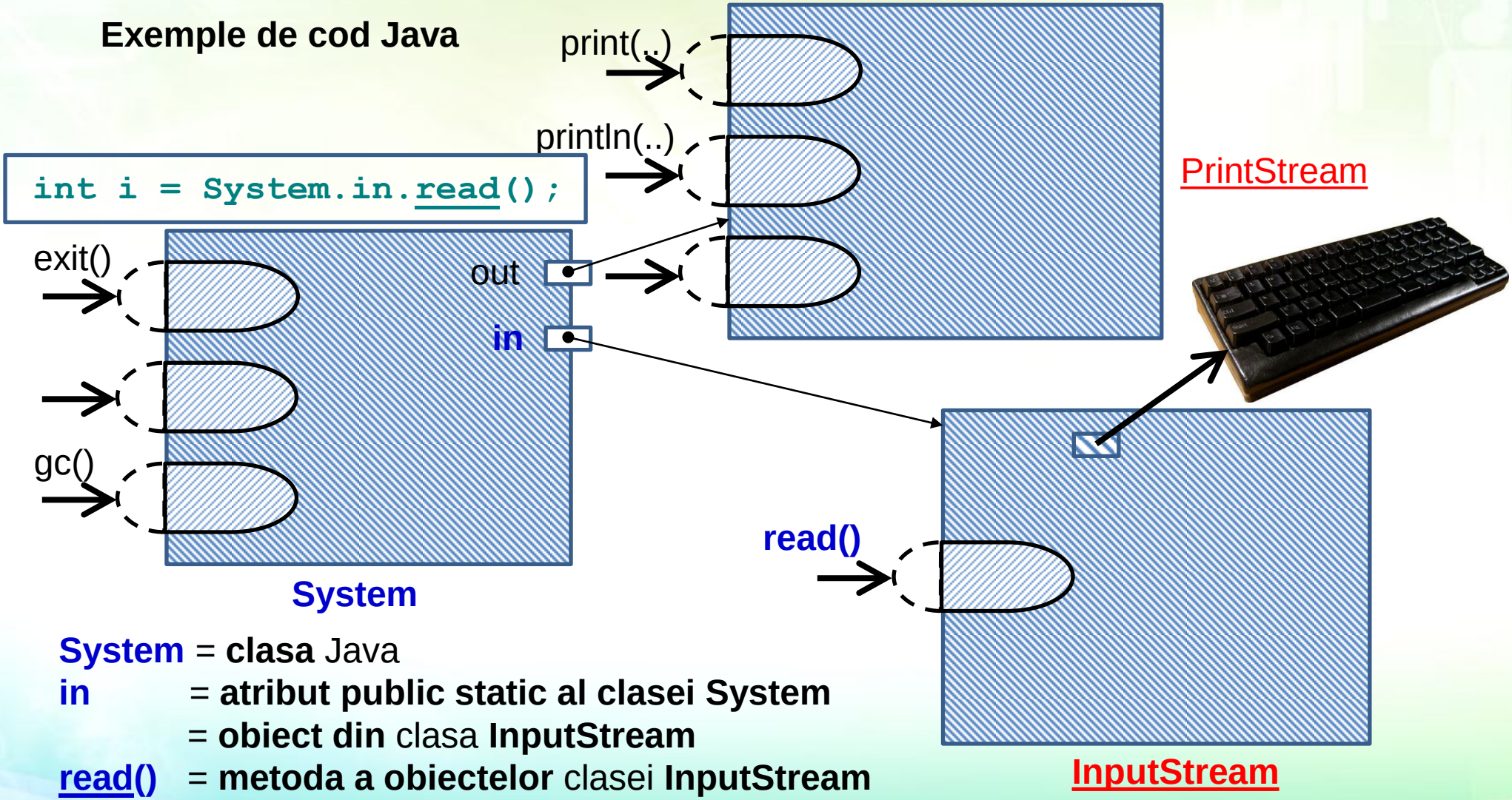
```
// clasa Java din pachetul java.io
public class InputStream {

  // metoda a obiectelor clasei InputStream
  public int read() {

    // .. codului metodei read()

  }

  // .. restul codului clasei InputStream
}
```

Obiect al unei clase flux de intrare care **incapsuleaza** fluxul de date dinspre **consola standard de intrare**)



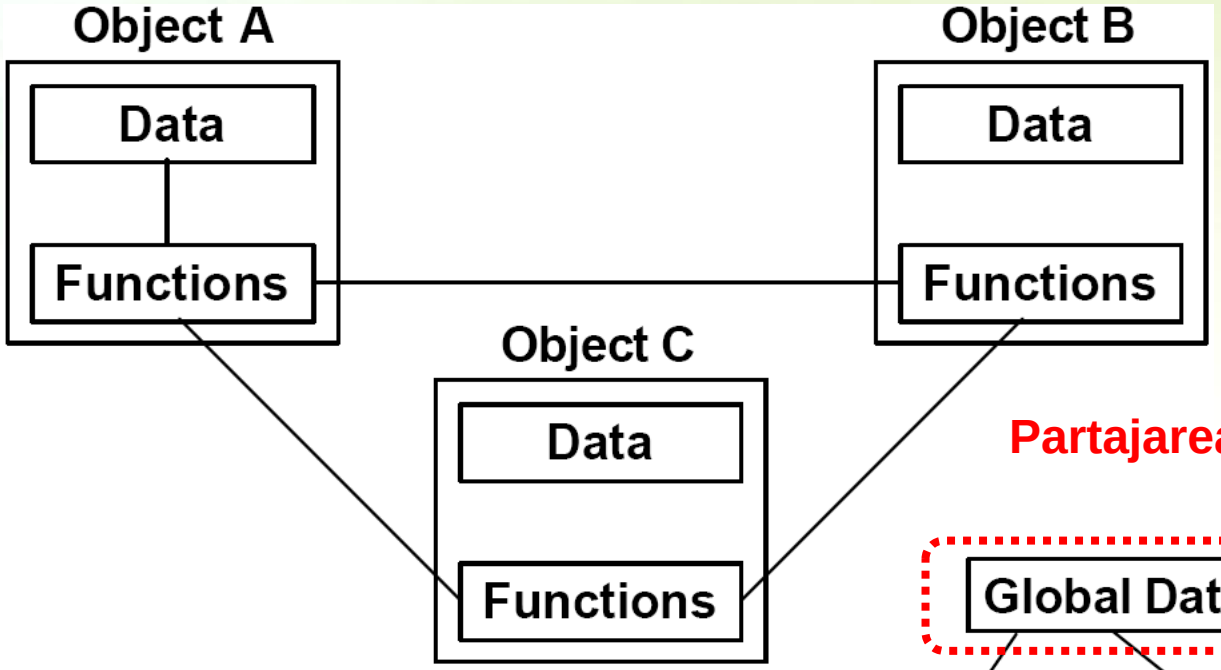
Abordarea orientata spre obiecte a dezvoltarii programelor

Exemple de cod Java



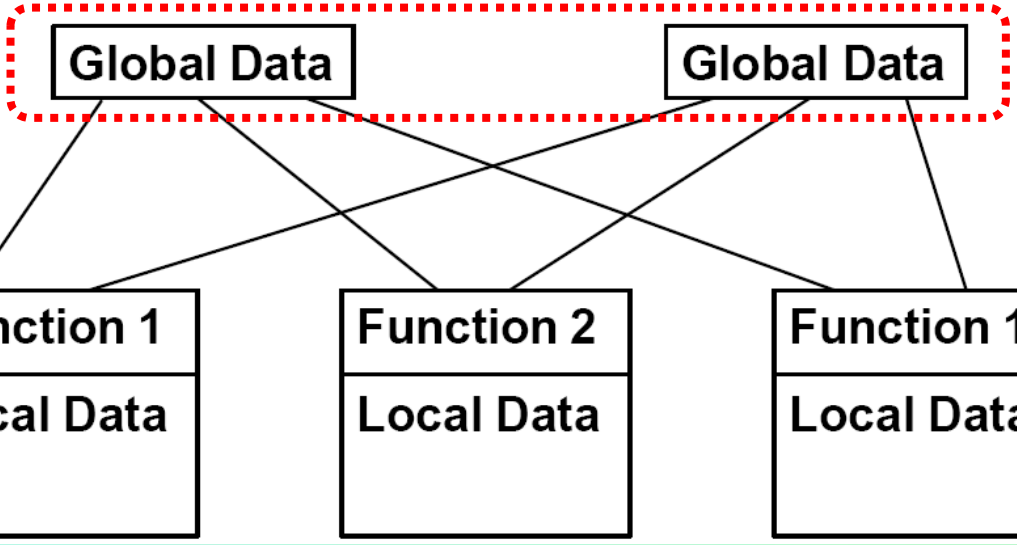
System = **clasa** Java
in = **atribut public static al clasei System**
= **obiect din** clasa **InputStream**
read() = **metoda a obiectelor** clasei **InputStream**

Abordarea orientata spre obiecte vs abordarea procedurala



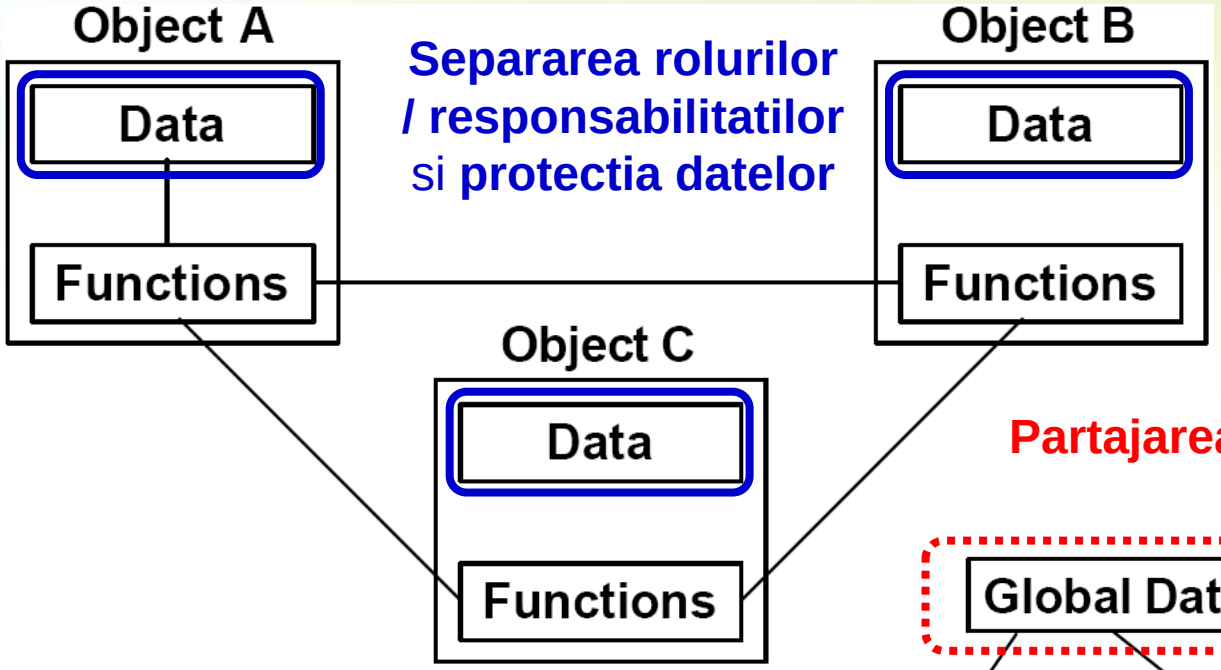
Abordarea OO

Partajarea datelor si responsabilitatilor creeaza probleme



Abordarea procedurala

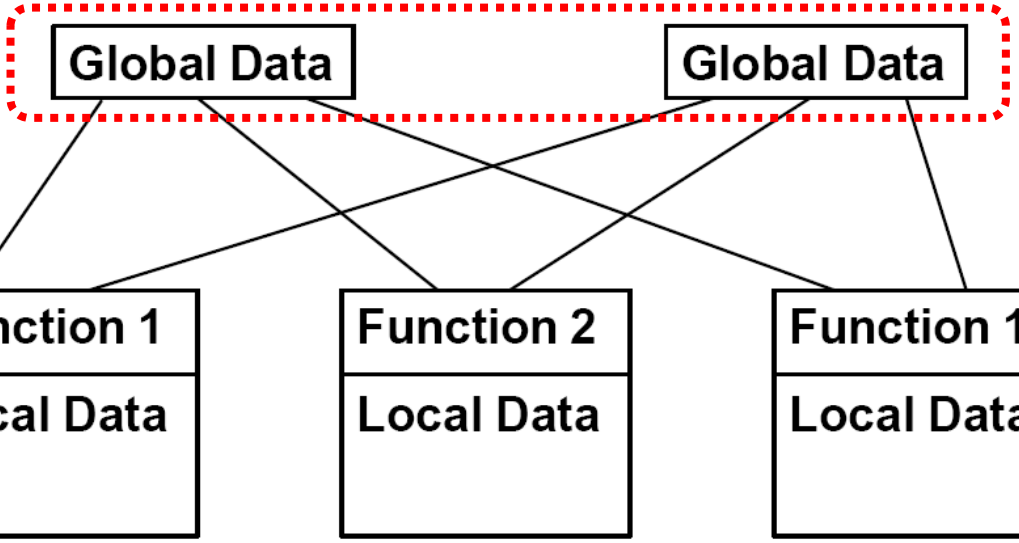
Abordarea orientata spre obiecte vs abordarea procedurala



Separarea rolurilor / responsabilitatilor si **protectia datelor**

Abordarea OO

Partajarea datelor si responsabilitatilor creeaza probleme



Abordarea procedurala

Programarea ca rezolvare de probleme

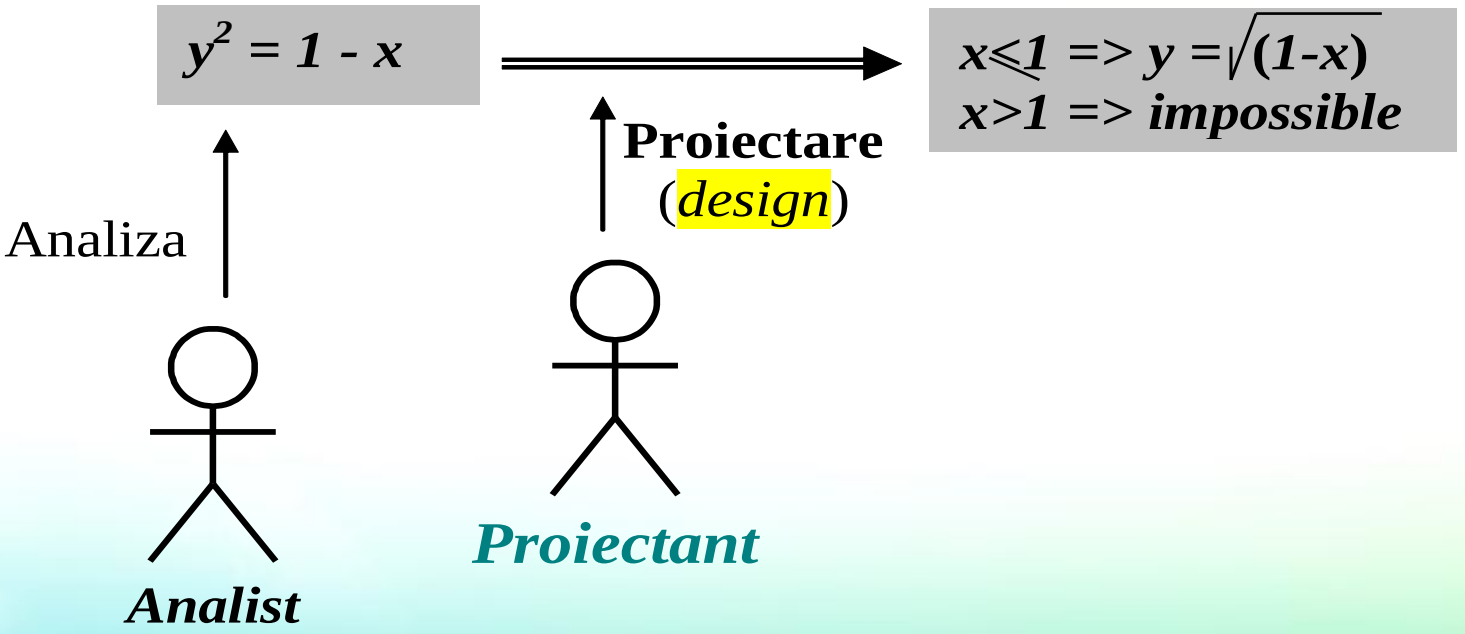
Problemele de rezolvat prin programare

– sunt **rezolvate** la nivel **conceptual**

– prin **proiectarea** unei **solutii**

abstractii

Problema *la nivel* ***conceptual*** ***Solutia*** *la nivel* ***conceptual***



Programarea ca rezolvare de probleme

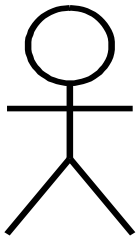
Solutia conceptuala a unei **probleme** de rezolvat prin programare

– este apoi **transformata** in pasi (instructiuni)

– care formeaza un **program** de calcul

Problema** la nivel **conceptual — ***Solutia** la nivel **conceptual***

Analiza $y^2 = 1 - x$ ⟹ $x \leqslant 1 \Rightarrow y = \sqrt{(1-x)}$; $x > 1 \Rightarrow$ ***impossible*** — Proiectare



Programare (codare si depanare)

*transformarea **conceptelor** in **pasi***

Programator

```
if (x > 1) {
   cout << "impossible" << endl;
} else {
   y = sqrt(1-x);
   cout << "y is " << y << endl;
}
```

Program

***Solutia** translatata **in limbaj** de programare*

Programarea ca rezolvare de probleme

Abordarea orientata spre obiecte a dezvoltarii programelor

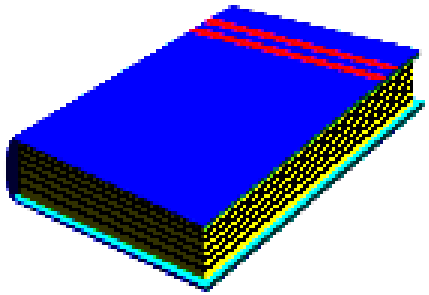
Incapsulare duala cu **ascunderea datelor** si a **comportamentului** (**clasa** de **obiecte**)

Domeniu al realitatii

domain concept

Analiza problemei

Entitate reala



Modelare / Abstractizare

Proiectarea solutiei

Book
title

visualization of domain concept

Implementarea (concretizarea) solutiei

Transformare constructii teoretice in constructii ale unui **limbaj**

representation in an object-oriented programming language

```
public class Book
{
private String title;

public Chapter getChapter(int) {...}
}
```

Constructii ale unui limbaj de programare OO

Orientarea spre obiecte (OO) este astfel o orientare spre

– **modelarea / abstractizarea (A)** informatica a **realitatii** (in **obiecte** si **clase**)

– **entitati** bazate pe **responsabilitati / roluri** (**clasele** si **obiectele**)

– **incapsularea (E) duala** – a **reprezentarii datelor** si a **comportamentului** (in **obiecte** si **clase**) cu **limitarea accesului**

– **mentinerea** si **ascunderea** unei **stari interne** (in **obiecte**)

– **colaborare** "sociala" intre entitati / **comunicare** prin **mesaje** (intre **obiecte**)

– definirea unor **interfete contractuale** (ale **claselor**) / componente ***black-box***

In plus, **OO utilizeaza si concepte mai avansate** **"A P I E"**

– **generalizarea claselor** in **superclase** si **specializarea claselor** in **subclase** prin **mostenire (I)**

– **polimorfismul (P)** – **selectia dinamica a comportamentului operatiei**

– **bazata pe** pozitia in **ierarhia de clase** a obiectului

Orientarea spre obiecte (OO) inseamna

- **modelarea / abstractizarea (A) realitatii**
- **entitati** cu **responsabilitati / roluri**
- **incapsularea (E) duala** (**date**+**comportament**) cu **limitarea accesului**
- **mentinerea** si **ascunderea** unei **stari interne**
- **colaborare** intre obiecte / **comunicare** prin **mesaje**
- **interfete contractuale** / componente ***black-box***

Concepte avansate **“A P I E”**

- **generalizarea** si **specializarea claselor** in **ierarhii** prin **mostenire (I)**
- **polimorfismul (P)**
  - **selectia dinamica a comportamentului operatiei**
    - **bazata pe** pozitia in **ierarhia de clase** a obiectului