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 POO 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

Obiectul cursului si relatia cu alte cursuri

- **Programarea**
  - **Masinile programabile** (suportul programarii) – curs **CID** si **AMP** (sem II)
  - **Programele** de calcul (tinta programarii) – curs **PC** si **SDA** (anul 1)
  - Programarea ca **rezolvare de probleme –** curs **Inginerie *Software* (?)**
- **Orientarea spre Obiecte (OO)**
  - **Evolutia catre OO**
    - Masina programabila si **codul masina** – curs **CID** si **AMP**
    - Limbajele de **asamblare** – curs **AMP**
    - Limbajele de **nivel inalt** (pre-OO) – curs **PC** si **SDA**
    - Tipurile de date abstracte (**ADT**) – curs **SDA**
  - **Orientarea** spre **modelarea realitatii**, entitati bazate pe **responsabilitati** (roluri), **incapsulare duala**, **interfete** (specificare), componente ***black-box***, **servicii**, etc.

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

   1.2. Evolutia catre abordarea OO

Evolutia catre abordarea Orientata spre Obiecte (OO)

- **Evolutia catre OO**
  - Masina programabila si **codul masina** – curs **CID** si **AMP**
  - Limbajele de **asamblare** – curs **AMP**
  - Limbajele de **nivel inalt** (pre-OO) – curs **PC** si **SDA**
    - Modelare si **abstractizare** (I)
    - **Incapsulare**, modularizare si **ascunderea detaliilor**
    - Programare **structurata**
    - Programare **procedurala**
  - Tipurile de date abstracte (**ADT**) – curs **SDA**
    - Incapsularea **duala**
    - Modelare si **abstractizare** (I)

Orientarea spre Obiecte (OO)

- **Evolutia catre OO**
  - **Incapsulare**, modularizare si **ascunderea detaliilor**

Incapsulare, modularizare si ascunderea detaliilor

Incapsularea are in teoria programarii **doua sensuri**

– **regrupare**: ——— ***Inside the box***

– **gruparea unor elemente** intre care exista legaturi

– **intr-o singura entitate noua**

– care poate astfel sa fie **referita** printr-un **singur nume**

– **limitarea accesului** sau **ascunderea detaliilor** (*details hiding*):

– **includerea** unui **lucru**

In / Out

Inside the box

– intr-un **alt lucru**

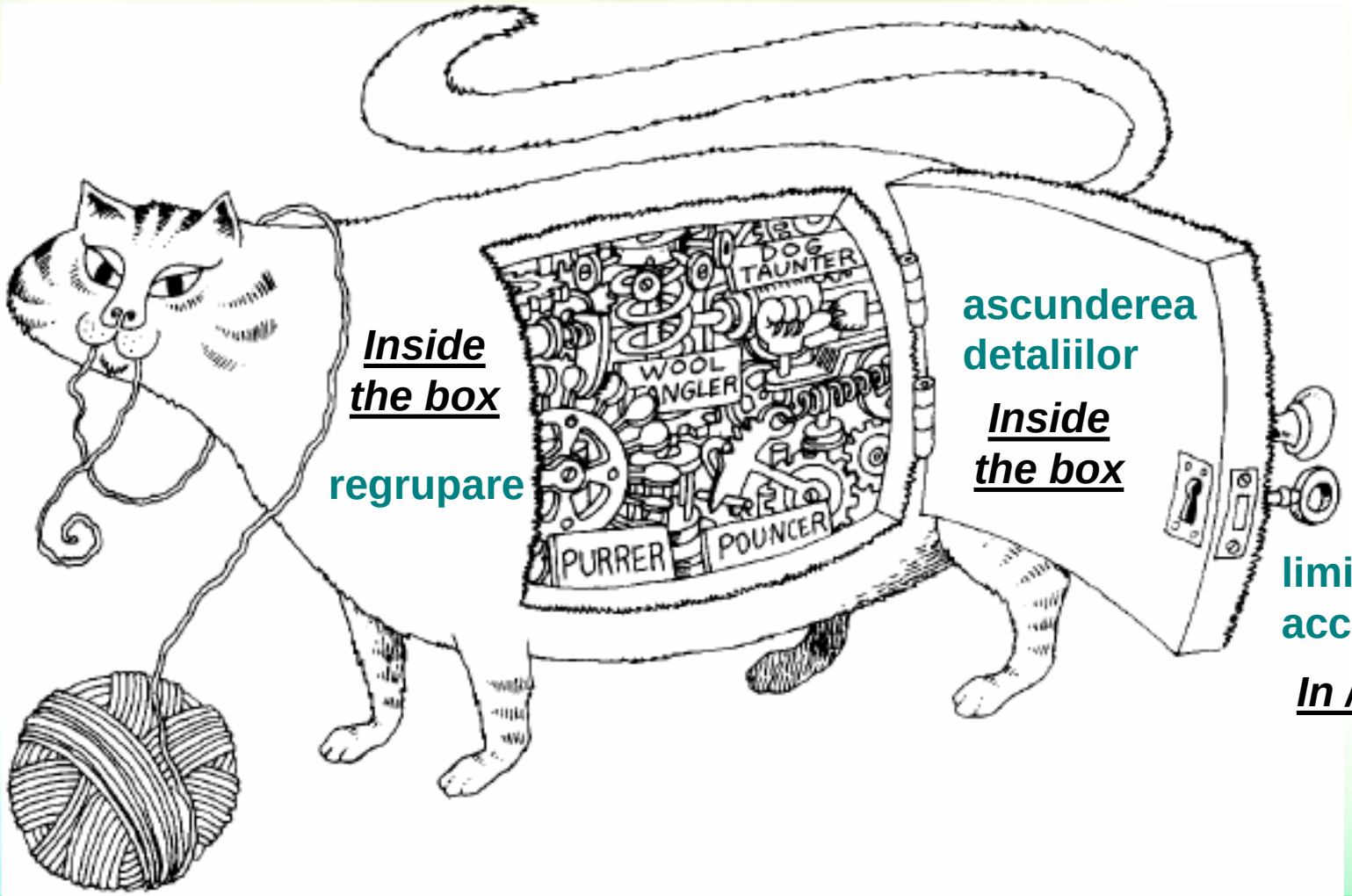
– astfel incat **lucrul inclus** sa **fie privat (inaccesibil, invizibil)**

Cele doua sensuri ale incapsularii pot fi **combinate** cu **abstractizarea** unei entitati

– rezultatul fiind o **entitate** de tip ***black box***

Incapsulare, modularizare si ascunderea detaliilor

Incapsularea ca regrupare si limitare a accesului (ascundere a detaliilor)



Incapsulare, modularizare si ascunderea detaliilor

Incapsularea rezultatului **unei abstractizari**

– este asadar o **combinatie** a celor **doua sensuri** ale incapsularii

– este procesul de **regrupare** a **elementelor** (detaliilor esentiale) unei **abstractii** intr-o **entitate noua** cu **nume propriu**, in urma caruia

– se **separa interfata** **contractuala** a abstractiei ——— ***In / Out***

– **detaliile** necesare **interactiunilor externe**

– care trebuie sa fie **publice**, **accesibile, vizibile**

– **de implementarea** sa ——— ***Inside the box***

– **detaliile** necesare **reprezentarii interne**

– care trebuie sa fie **private**, **inaccesibile**

– **rezultatul** fiind o **entitate** de tip ***black box***, concept aplicabil

– **modulelor** functionale, **obiectelor** si **componentelor** software

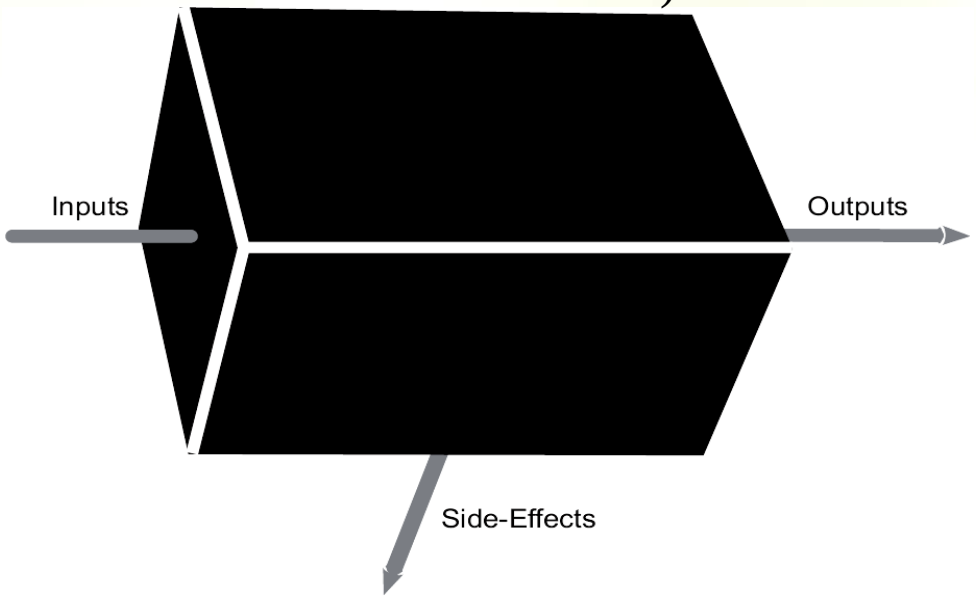
Incapsulare, modularizare si ascunderea detaliilor

Un program / o componenta a unui program ca **servicii** oferite de un ***black box***

Black Box

(***abstractizare, incapsulare cu ascunderea detaliilor***)

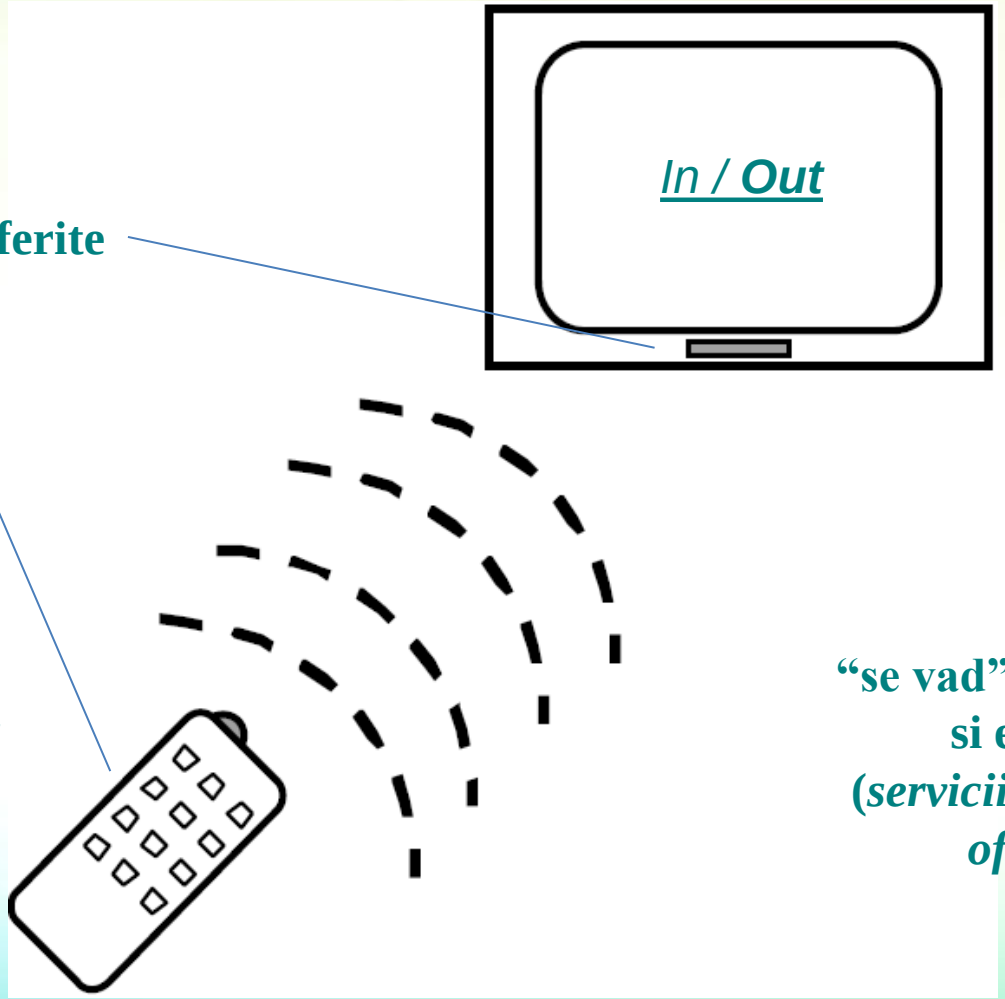
Utilizator (***uman / alta componenta***)



"se vad" doar intrarile, iesirile, si efectele colaterale (*serviciile furnizate, interfetele oferite si necesare*)

Incapsulare, modularizare si ascunderea detaliilor

Servicii oferite de un ***black box***



Black Box (***incapsulare, abstractizare,*** *ascundere a detaliilor*)

"se vad" doar intrarile, iesirile, si efectele colaterale (*serviciile furnizate, interfetele oferite si necesare*)

Incapsulare, modularizare si ascunderea detaliilor

Beneficiile incapsularii *black box*

– permite **imbunatatirea** **mecanismelor interne** ale unei componente **fara impact** asupra altor componente

– permite **inlocuirea** **componentei** cu o alta care ofera **aceeasi interfata publica**

– **protejeaza integritatea** **componentei**

– impiedicandu-I pe utilizatori sa aduca detaliile interne ale componentei in stari **invalide** sau **inconsistente**

– **reduce complexitatea** **perceputa** de utilizatori (**programatori**)

– care sunt eliberati de **necesitatea cunoasterii** detaliilor

– **reduce complexitatea** **sistemului** marind astfel robustetea

– prin **limitarea interdependentelor** intre componente software

Incapsulare, modularizare si ascunderea detaliilor

Incapsularea informatica

– inseamna **regruparea** elementelor unei **abstractii informatice**

– care poate reprezenta **date** si/sau **comportament**

Incapsularea informatiilor (datelor)

– inseamna **regruparea** elementelor de date in

– **structuri /tipuri de date complexe** (structuri, uniuni, enumerari, etc.)

– si este **implicit** o incapsulare **FARA ascundere a detaliilor (informatiilor)**

Incapsularea comportamentului

– inseamna **regruparea** elementelor de comportament in

– **module** (subprograme, rutine, proceduri, functii, operatii, metode)

– si este **implicit** o incapsulare **CU ascundere a detaliilor (implementarii)**

Incapsulare, modularizare si ascunderea detaliilor

Incapsularea informatiilor (datelor)

- inseamna **regruparea** unor elemente de date simple
  - in **structuri / tipuri de date complexe**
    - **numele** structurii / tipului formand ”**interfata**” care permite accesul
    - **elemente de date regrupate** formand “**detaliile interne**”
  - **FARA ascundere a detaliilor** (**informatiilor**, **datelor**)
- permite crearea unor **tipuri de date noi**
  - pornind de la cele fundamentale / elementare / primitive
- tipurile de date noi (eventual parte a unor biblioteci)
  - sunt fie **predefinite** in cadrul limbajului
  - fie **definite de programator**

Incapsulare, modularizare si ascunderea detaliilor

Incapsularea informatiilor (datelor) in C

Pornind de la portiunea de program urmatoare

```
int numarCont;
char *prenume;
char *nume;
float sold;

numarCont = 14378;
prenume = "Ics";
nume = "Igrec";
sold = 3300.00;
```

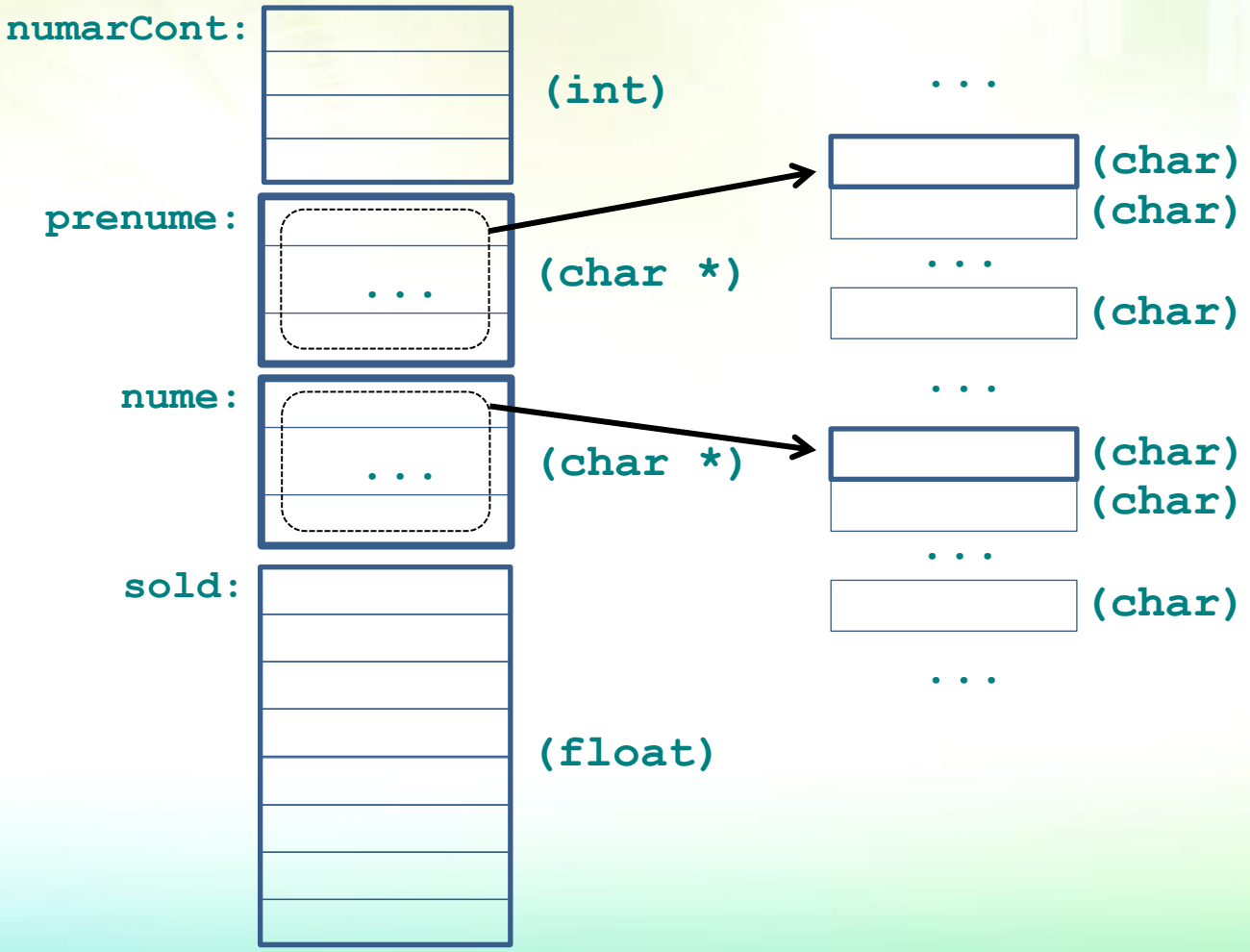
Cum s-ar putea **incapsula aceste elemente de date?**

Cum ar arata **programul rescris** (*refactored*) in acest caz**?**

Incapsulare, modularizare si ascunderea detaliilor

Incapsularea informatiilor (datelor) in C

```
int numarCont;
char *prenume;
char *nume;
float sold;
```

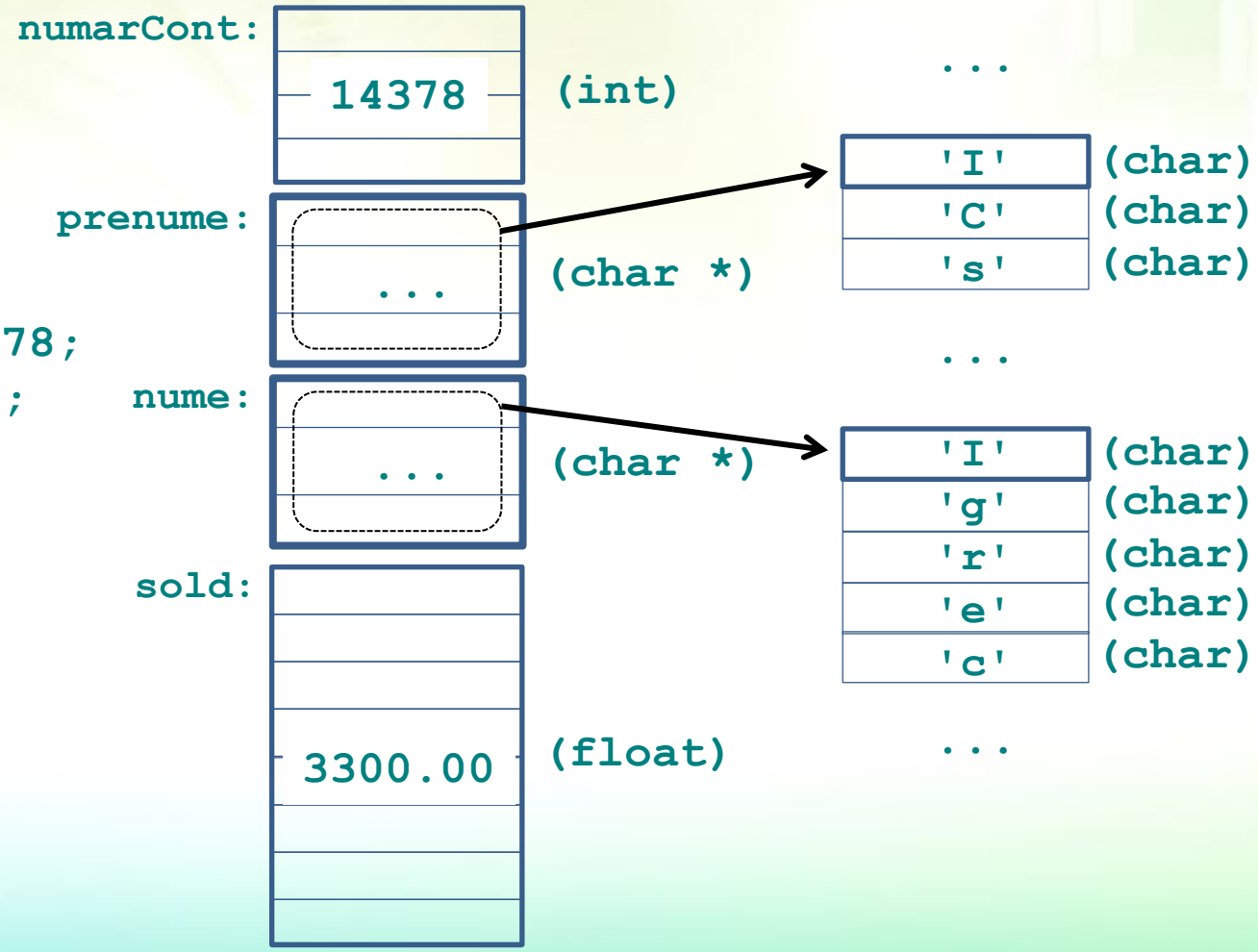


Primele 4 linii de cod

Incapsulare, modularizare si ascunderea detaliilor

Incapsularea informatiilor (datelor) in C

```
int numarCont;
char *prenume;
char *nume;
float sold;

numarCont = 14378;
prenume = "Ics";
nume = "Igrec";
sold = 3300.00;
```



Toate cele 8 linii de cod

Incapsulare, modularizare si ascunderea detaliilor

Posibila incapsulare a informatiilor (datelor) in C

```
struct Cont {          // tip de date NOU cu nume ("Cont")
    int numarCont;     // camp de tip int (intreg)
    char *prenume;     // camp de tip pointer la caractere
    char *nume;        // camp de tip pointer la caractere
    float sold;        // camp de tip float (real)
};
struct Cont c;         // declaratia (tipului) variabilei c
```

Incapsularea in acest caz este **FARA ascundere a detaliilor (informatiilor, datelor)!**

Doar regrupare!

Incapsulare, modularizare si ascunderea detaliilor

Posibila incapsulare a informatiilor (datelor) in C

```
struct Cont {        // tip de date NOU cu nume ("Cont")
   int numarCont;    // camp de tip int (intreg)
   char *prenume;    // camp de tip pointer la caractere
   char *nume;       // camp de tip pointer la caractere
   float sold;       // camp de tip float (real)
};
struct Cont c;       // declaratia (tipului) variabilei c
struct Cont c1;      // reutilizarea numelui tipului de date
```

Incapsularea permite insa
reutilizarea tipului nou de date

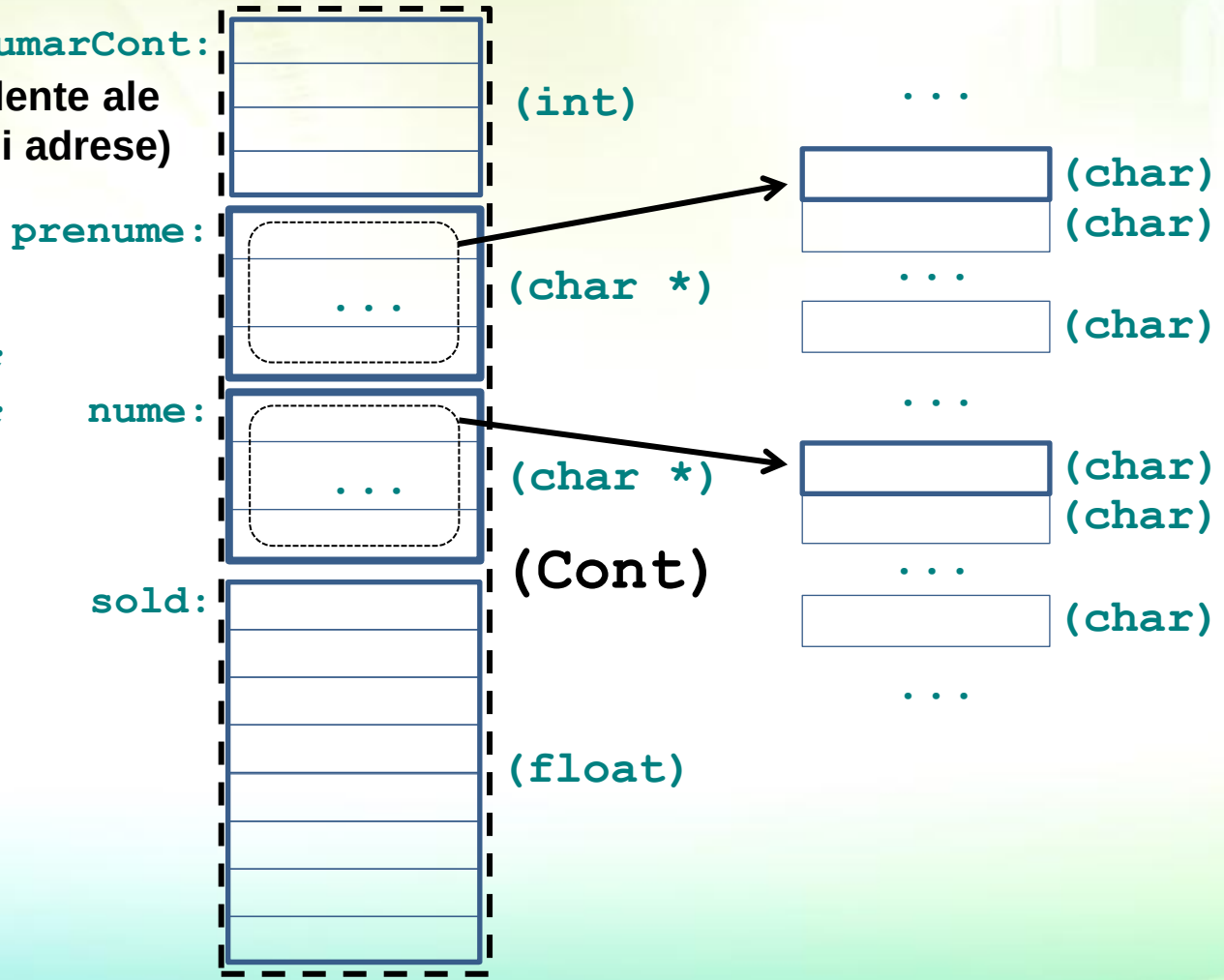
Incapsulare, modularizare si ascunderea detaliilor

Posibila incapsulare a informatiilor (datelor) in C

```
struct Cont {
    int numarCont;
    char *prenume;
    char *nume;
    float sold;
};
struct Cont c;
```

Primele 4 linii de cod

`c = c.numarCont;`
(echivalente ale aceleiasi adrese)

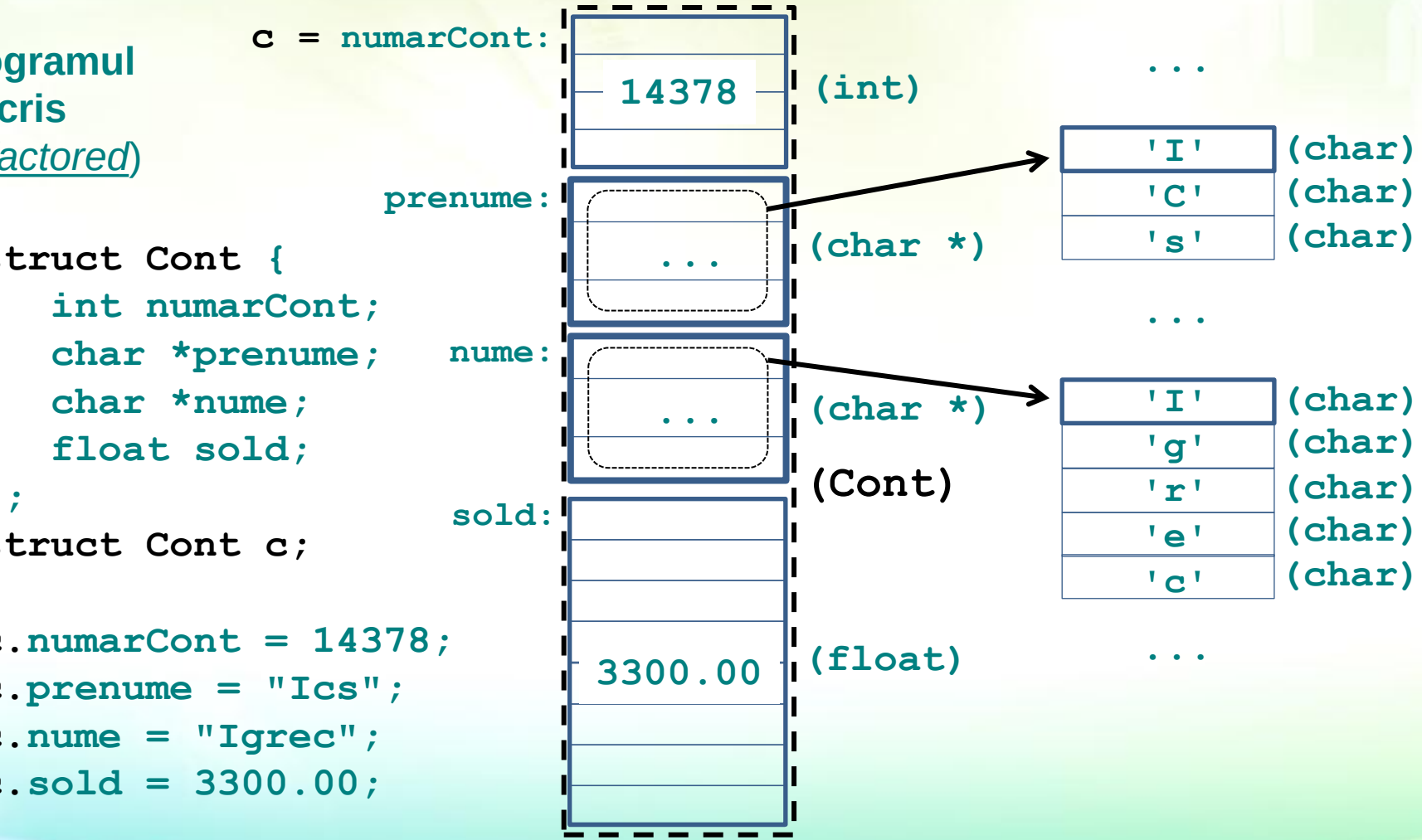


Incapsulare, modularizare si ascunderea detaliilor

Posibila incapsulare a informatiilor (datelor) in C

Programul rescris (*refactored*)

```
struct Cont {
    int numarCont;
    char *prenume;
    char *nume;
    float sold;
};
struct Cont c;

c.numarCont = 14378;
c.prenume = "Ics";
c.nume = "Igrec";
c.sold = 3300.00;
```



Incapsulare, modularizare si ascunderea detaliilor

Posibila simplificare a utilizarii incapsularii informatiilor (datelor) in C

```
typedef struct {    // tip de date NOU
   int numarCont;   // camp de tip int (intreg)
   char *prenume;   // camp de tip pointer la caractere
   char *nume;      // camp de tip pointer la caractere
   float sold;      // camp de tip float (real)
} Cont;             // nume dat tipului de date NOU

Cont c;      // declaratia tipului variabilei c
             // intr-o forma simplificata
```

Incapsulare, modularizare si ascunderea detaliilor

Ascunderea detaliilor poate avea mai multe **sensuri / forme**

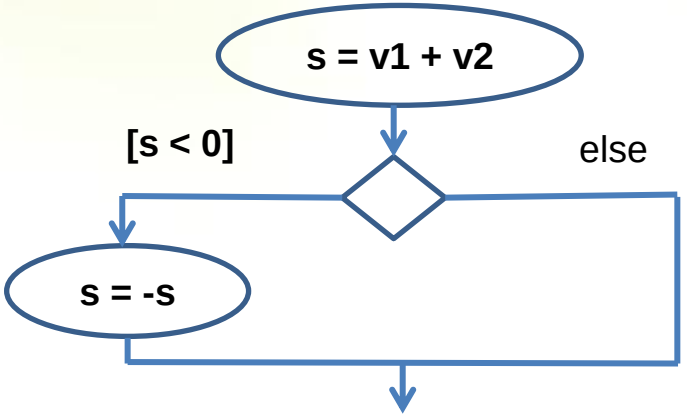
- ascunderea **informatiilor** (**datelor**)
  - reprezinta **ascunderea detaliilor** necesare **reprezentarii interne a structurilor de date**
  - **NU se regaseste** implicit in **incapsularea informatiilor (datelor)**
- ascunderea **implementarii**
  - reprezinta **ascunderea detaliilor** necesare **reprezentarii interne** ale unei **abstractizari**
    - rezultate in urma **separarii interfetei** necesara interactiunilor
      - care trebuie sa fie publica, accesibila, vizibila
    - de **implementarea** ei
  - se regaseste **implicit** in **incapsularea comportamentului** in **module functionale**

Incapsulare, modularizare si ascunderea detaliilor

Incapsularea comportamentului in C

Pornind de la portiunea de program pentru **"*calculul modulului sumei a doua valori*"**

```
s = v1 + v2; // suma
if (s<0)     // structura decizie
    s = -s;  // modul
```



Cum s-ar putea **incapsula acest comportament?**

Incapsulare, modularizare si ascunderea detaliilor

Posibila incapsulare a codului pentru **"*calculul modulului sumei a doua valori*"**

```
// declaratie functie - incapsulare comportament
int modululSumei() {    // interfata publica (semnatura)
    int s;              // implementare ascunsa
    s = v1 + v2;        // acces la variabilele globale v1 si v2
    if (s<0) s = -s;
    return s;
}
... {
    // ...
    s = modululSumei(); // apel functie (delegare functionala)
}
```

Cum ar arata intreg **programul rescris** (*refactored*) in acest caz**?**

Incapsulare, modularizare si ascunderea detaliilor

Posibil program bazat pe incapsularea anterioara

```
#include <stdio.h>
int v1 = 3, v2 = 5;              // 2 variabile globale
int main(void) {
    int s;                       // variabila locala
    s = modululSumei();          // fara argumente
    printf("modulul sumei %d cu %d este %d\n", v1, v2, s);
    return 0;
}
int modululSumei() {             // fara parametri
   int s;                        // variabila locala
   s = v1 + v2;
   if (s<0) s = -s;
   return s;
}
```

Cum ar putea fi facuta functia modululSumei() mai generala?

Incapsulare, modularizare si ascunderea detaliilor

Posibila incapsulare a codului pentru **"*calculul modulului sumei a doua valori*"**

```
// declaratie functie - incapsulare comportament
int modululSumei(int a, int b) {   // declaratii parametri
   int s;                  // implementare ascunsa
   s = a + b;              // acces la parametri
   if (s<0) s = -s;
   return s;
}
... {
   // ...
   s = modululSumei(v1,v2);//apel functie(delegare functionala)
}
```

Cum ar arata **programul rescris** (*refactored*) in acest caz**?**

Incapsulare, modularizare si ascunderea detaliilor

Alta posibila incapsulare pentru **"*calculul modulului sumei a doua valori*"**

```
#include <stdio.h>
int v1 = 3, v2 = 5;                  // 2 variabile globale
int main(void) {
    int s;                           // variabila locala
    s = modululSumei(v1, v2);        // 2 argumente
    printf("modulul sumei %d cu %d este %d\n", v1, v2, s);
    return 0;
}
int modululSumei(int a, int b) {  // 2 parametri
   int s;                            // variabila locala
   s = a + b;
   if (s<0) s = -s;
   return s;
}
```

Ce posibila problema vedeti in codul de mai sus **din punctul de vedere al variabilelor?**

Incapsulare, modularizare si ascunderea detaliilor

Cum poate sa apara o problema?

```
#include <stdio.h>
int v1 = 3, v2 = 5;                  // 2 variabile globale NEASCUNSE
int main(void) {
    int s;                           // variabila locala
    s = modululSumei(v1, v2);        // 2 argumente
    v1 = 7;                          // al doilea acces la v1
    printf("modulul sumei %d cu %d este %d\n", v1, v2, s);
    return 0;
}
int modululSumei(int a, int b) { // 2 parametri
   int s;                            // variabila locala
   s = a + b;                        // primul acces la v1
   if (s<0) s = -s
   return s;
}
```

- Variabila s **nu** creeaza nici o problema!
- **Ce afiseaza insa programul?**

Incapsulare, modularizare si ascunderea detaliilor

Problema apare din cauza accesului neingradit la variabilele globale

```
#include <stdio.h>
int v1 = 3, v2 = 5;                  // 2 variabile globale NEASCUNSE
int main(void) {
    int s;                           // variabila locala
    s = modululSumei(v1, v2);        // 2 argumente
    v1 = 7;                          // al doilea acces la v1
    printf("modulul sumei %d cu %d este %d\n", v1, v2, s);
    return 0;
}
int modululSumei(int a, int b) { // 2 parametri
   int s;                            // variabila locala
   s = a + b;                        // primul acces la v1
   if (s<0) s = -s
   return s;
}
```

Incapsulare, modularizare si ascunderea detaliilor

Modulul software

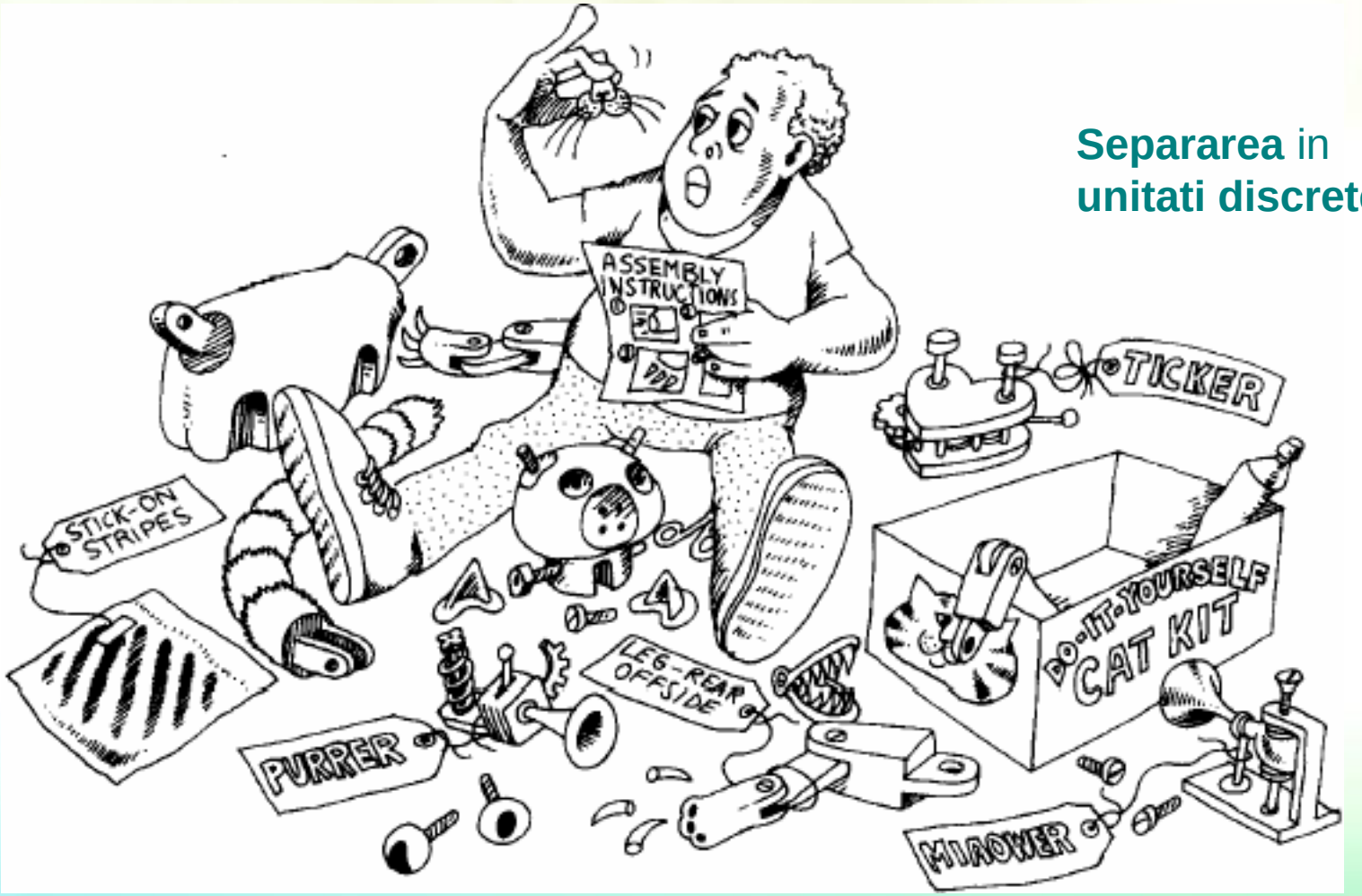
- **parte separata** din program / **unitate de cod** (functie, obiect, fisier, etc.)
  - care **interactioneaza** cu alte module
  - **doar prin** intermediul unor **interfete** clar **definite**
    - ca **intrari si iesiri** (cum ar fi: valori returnate, liste de argumente, semnaturi ale operatiilor, etc.)
- conceptual reprezinta o **separare a preocuparilor** (*separation of concerns*)
  - **introducerea limitelor** logice intre module **usurand modificarile**

Modularizarea (programarea modulara, 1972 – David Parnas)

- **tehnica de proiectare** a programelor prin **descompunerea** lor în module
- este necesara in special in **programe mari, complicate**
- are ca scop cresterea masurii in care **programul** este **compus** din **module**

Incapsulare, modularizare si ascunderea detaliilor

Modularizarea – proiectare prin **descompunerea în module**



Separarea in
unitati discrete

Incapsulare, modularizare si ascunderea detaliilor

Incapsulare vs modularizare informatica

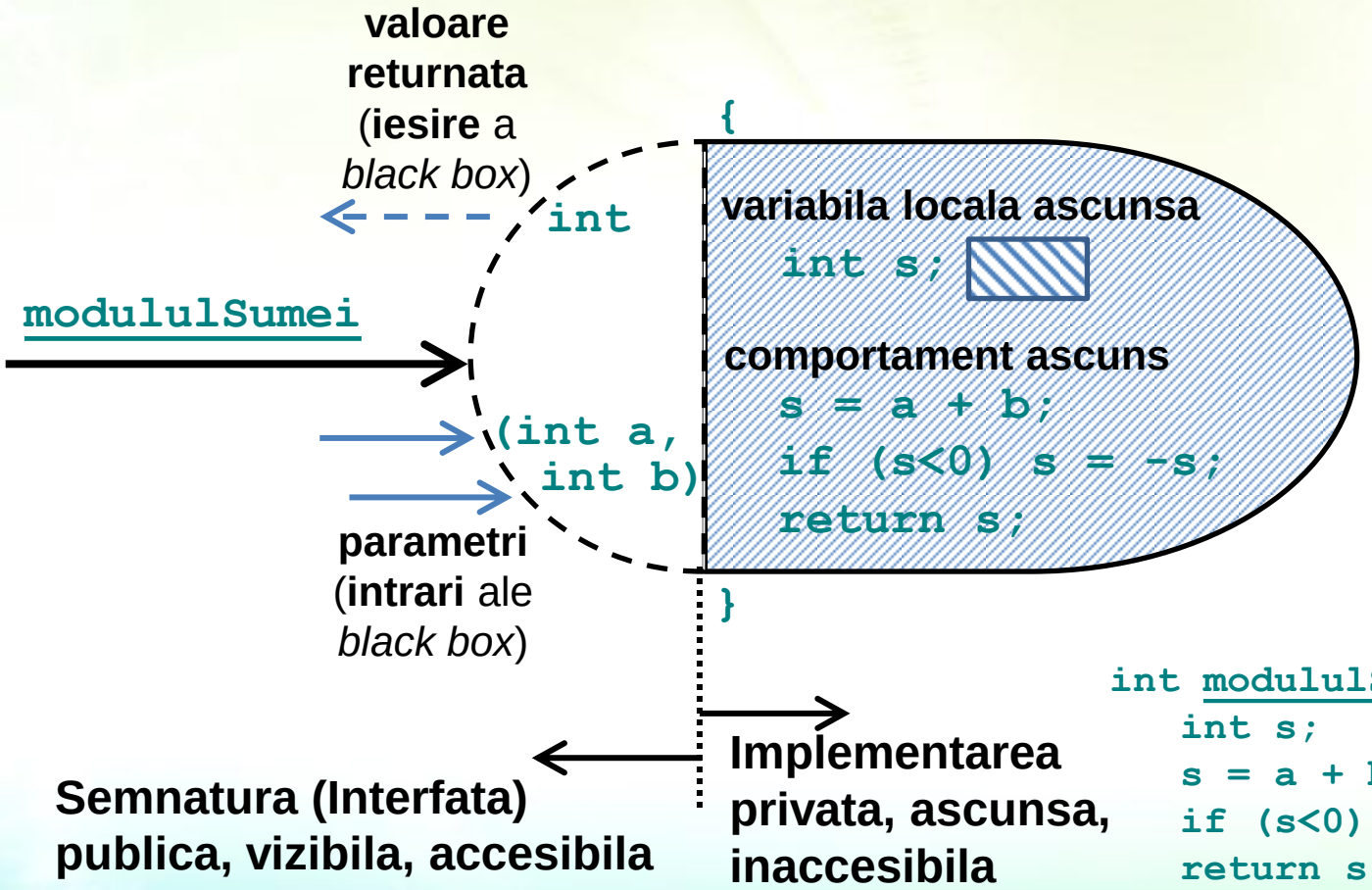
- **incapsularea** insista pe
  - **regruparea** elementelor unei **abstractii** informatice
- **modularizarea** insista pe
  - **descompunerea** unui program in **unitati de program** numite **module**

Ambele

- folosesc **separarea interfetei** de **implementare**
  - pentru a introduce **limite** intre entitatile care rezulta
  - si pot astfel genera **componente** de tip ***black box***
- ca efect al acestei separari
  - usureaza **managementul** **complexitatii** si al **schimbarii**

Incapsulare, modularizare si ascunderea detaliilor

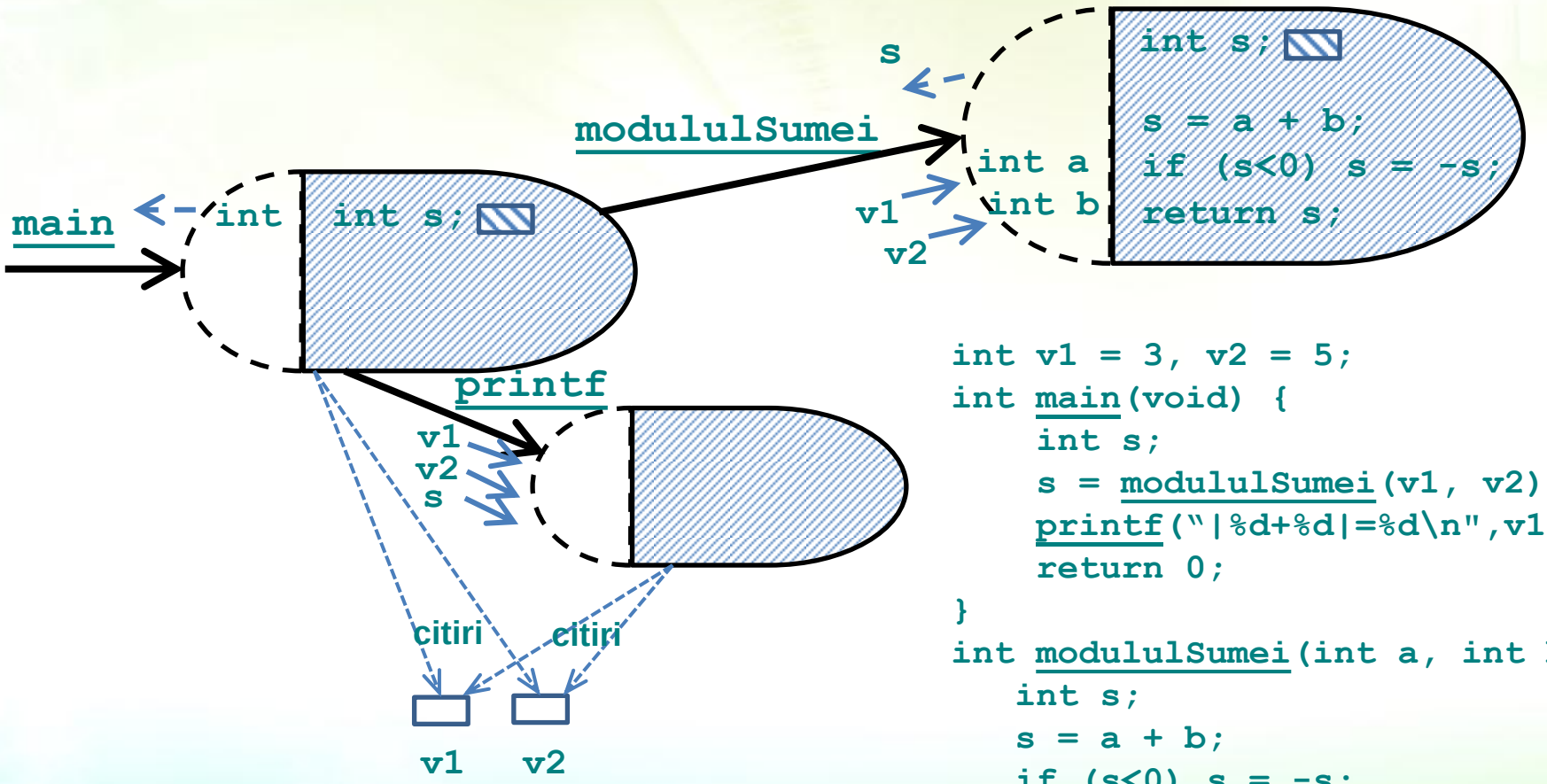
Functia modululSumei() vazuta ca o componenta *black box*




```
int modululSumei(int a, int b) {
    int s;
    s = a + b;
    if (s<0) s = -s;
    return s;
}
```

Incapsulare, modularizare si ascunderea detaliilor

Functia modululSumei() vazuta ca o componenta *black box*

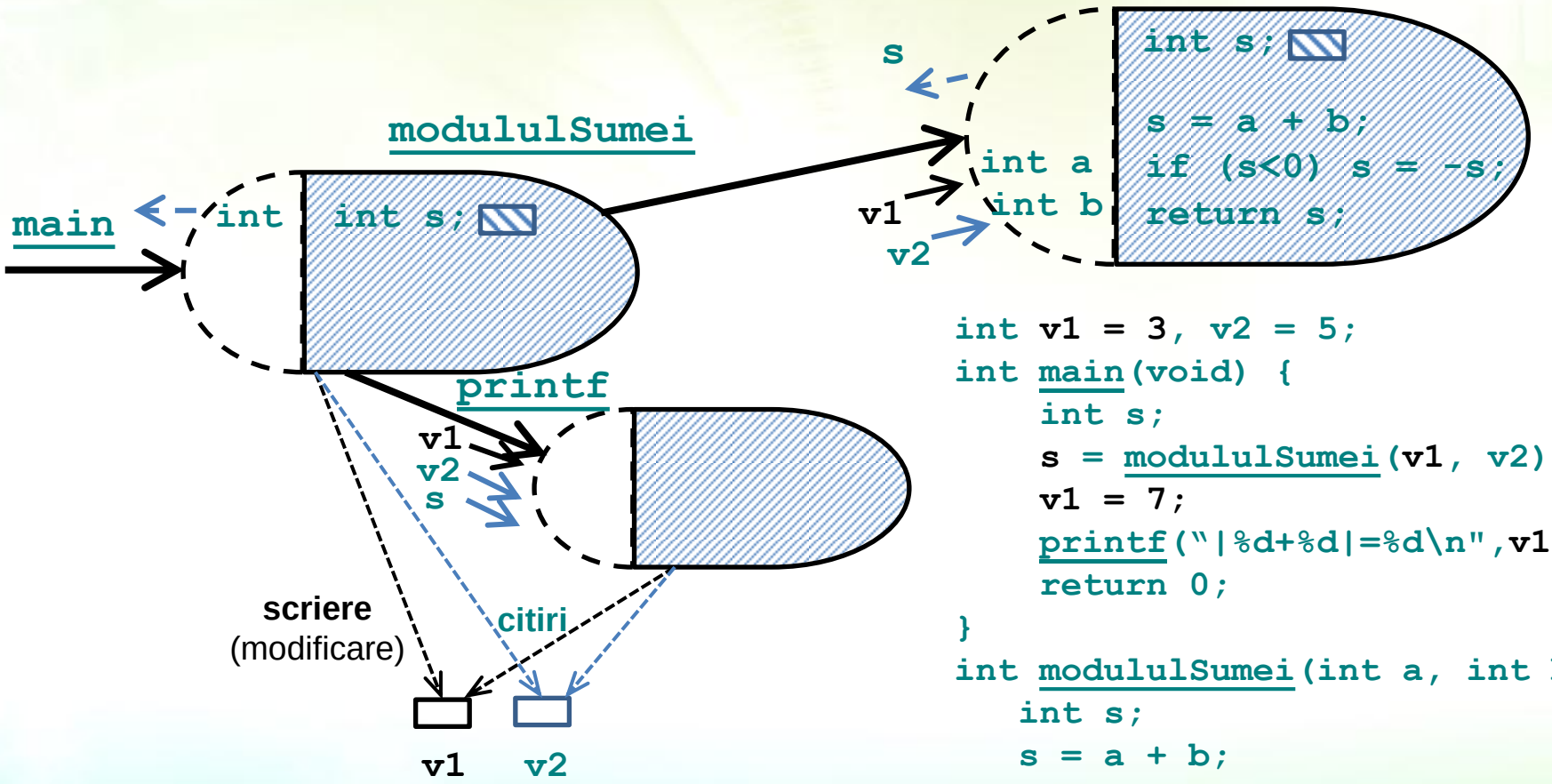


Variabile globale publice, vizibile, accesibile oricui

```
int v1 = 3, v2 = 5;
int main(void) {
    int s;
    s = modululSumei(v1, v2);
    printf("|%d+%d|=%d\n",v1,v2,s);
    return 0;
}
int modululSumei(int a, int b) {
   int s;
   s = a + b;
   if (s<0) s = -s;
   return s;
}
```

Incapsulare, modularizare si ascunderea detaliilor

Problemele create de accesul neingradit la variabile globale




```
int v1 = 3, v2 = 5;
int main(void) {
    int s;
    s = modululSumei(v1, v2);
    v1 = 7;
    printf("|%d+%d|=%d\n",v1,v2,s);
    return 0;
}
int modululSumei(int a, int b) {
   int s;
   s = a + b;
   if (s<0) s = -s;
   return s;
}
```

⇒ ***Pierderea consistentei variabilei v1***

Exemple de **teme pentru eseu (nou: 08.10.2010)**

- Abstractizarea informatica (trecere in revista cu exemple)

- Abstractizarea informatica (motivatie, beneficii)

- Abstractizarea orientata spre obiecte (motivatie, beneficii)

- Incapsularea informatica (trecere in revista cu exemple)

- Incapsularea informatica (motivatie, beneficii)

- Incapsularea orientata spre obiecte (motivatie, beneficii)

- Modularizarea informatica (trecere in revista cu exemple)

- ... (urmeaza a fi definite si alte titluri)

Orientarea spre Obiecte (OO)

- **Evolutia catre OO**
  - Programarea **structurata**

Programarea structurata

Programarea structurata

– este programarea in care **logica** **unui program**

– este o **structura compusa** din **sub-structuri similare**

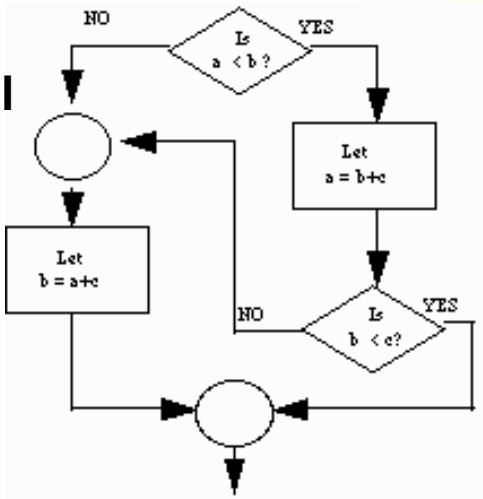
– intr-un **numar limitat** de **moduri**

– **structura logica** face **programul**

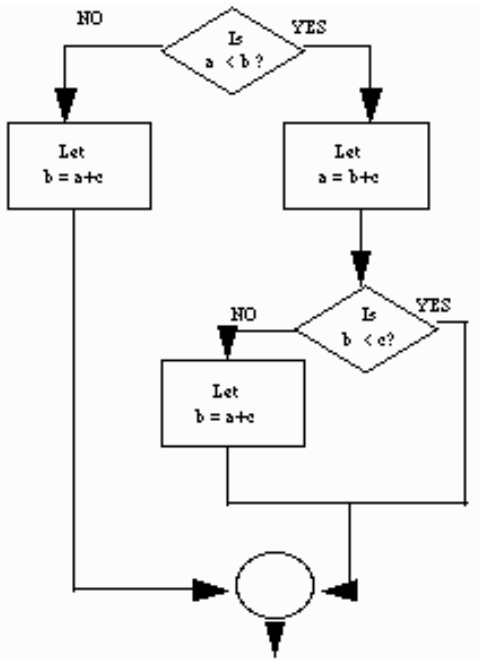
– **eficient**

– usor de **inteles**

– usor de **modificat**



FLOWCHART USING THE GOTO STATEMENT



STRUCTURED FLOWCHART WITH GOTO REMOVED

Programarea structurata

Programarea structurata

– un exemplu este **blocul de instructiuni**

– cu rol de **incapsulare** a unei secvente de instructiuni

– adica **regrupare** a secventei de instructiuni

– **fara nume** prin care poate fi referit, si **fara intrari si iesiri**

– **scopul** (**domeniul de existenta al**) **variabilelor** locale **declarate in interiorul unui bloc**

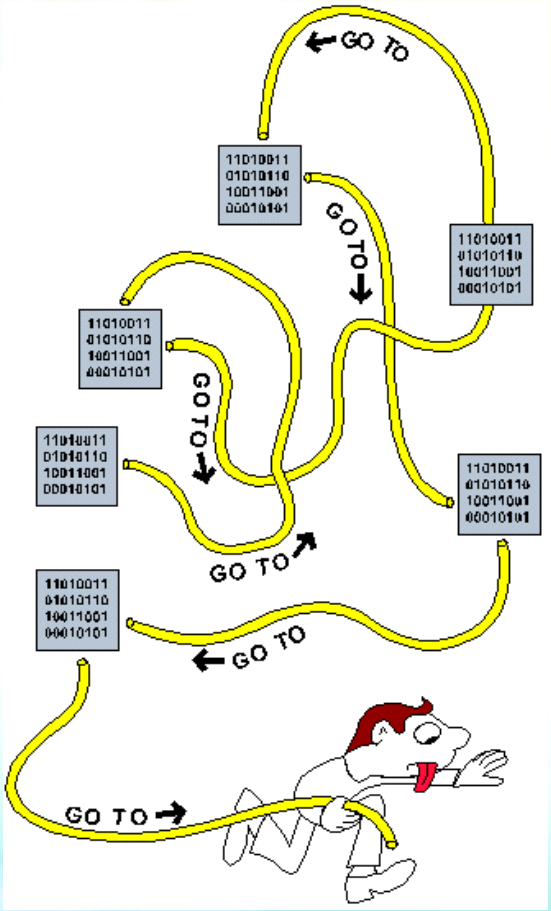
– este **din locul declararii** si **pana la finalul blocului**

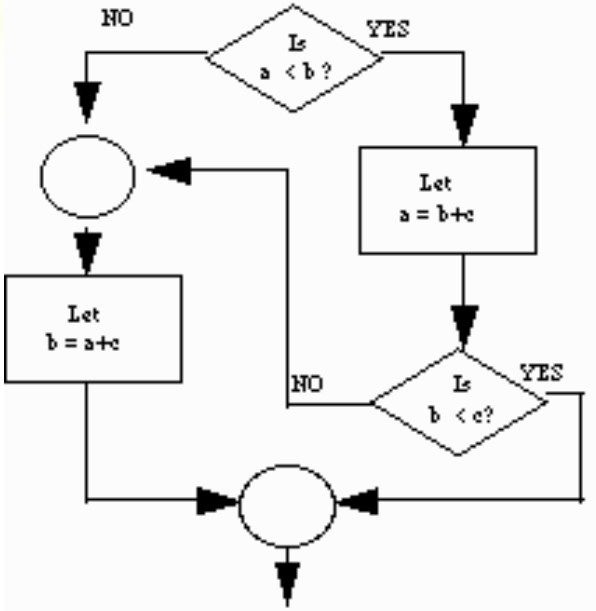
```
..   // aici v NU exista

{   // BEGIN in Pascal

    ..          // aici v NU exista

    int v; // declaratie tip

    ..          // aici v exista

}   // END in Pascal

..   // aici v NU exista
```

Programarea structurata

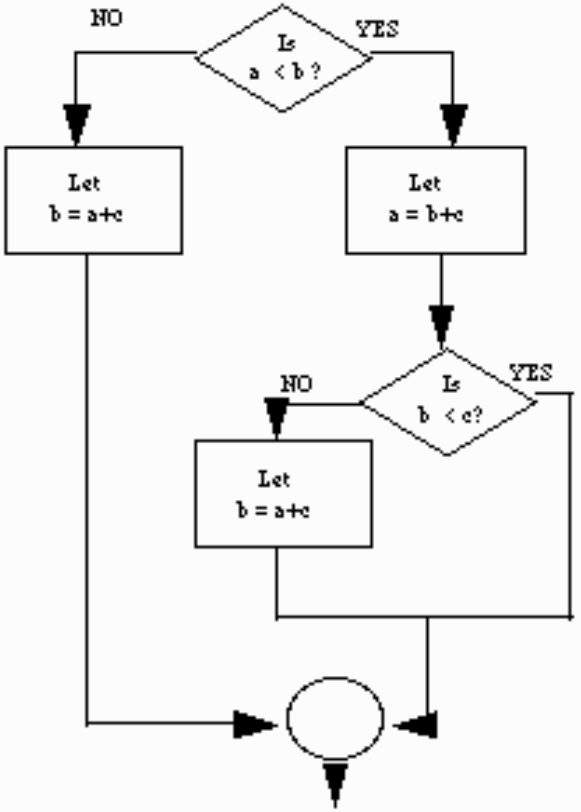
Programarea structurata

– **descurajeaza utilizarea** instructiunilor de tip "**Go To eticheta**"





FLOWCHART USING THE GOTO STATEMENT



STRUCTURED FLOWCHART WITH GOTO REMOVED

Programarea structurata

Programarea structurata

– la **cel mai jos nivel** se afla **structuri de control al programului** simple, ierarhice:

– **secventa**

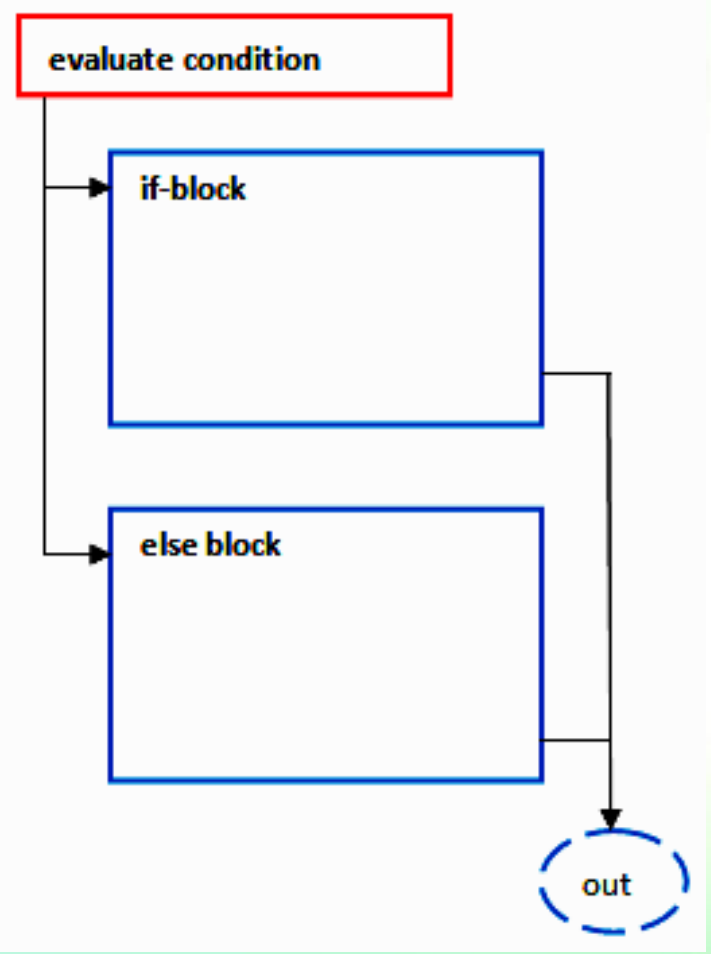
– executia **in ordine** a instructiunilor

– **selectia conditionata**

– **decizia** privind instructiunea care urmeaza a fi executata dintr-un numar de instructiuni

– **in functie de starea curenta** a programului (**if**, **if..else**, if..then..else..endif, **switch**, **case**, etc.)

decizie / selectie

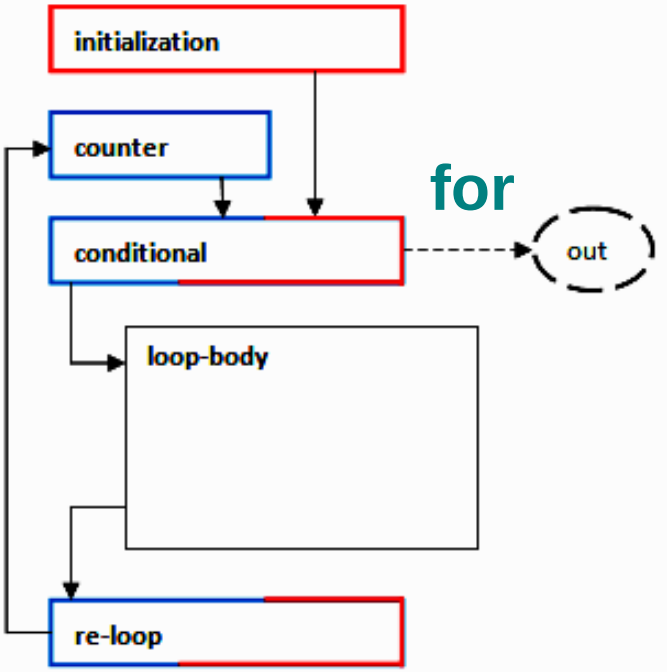


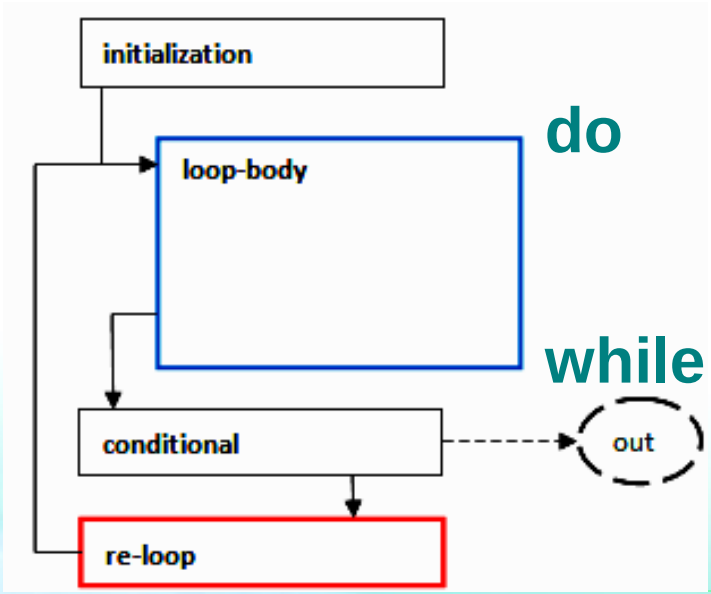
Programarea structurata

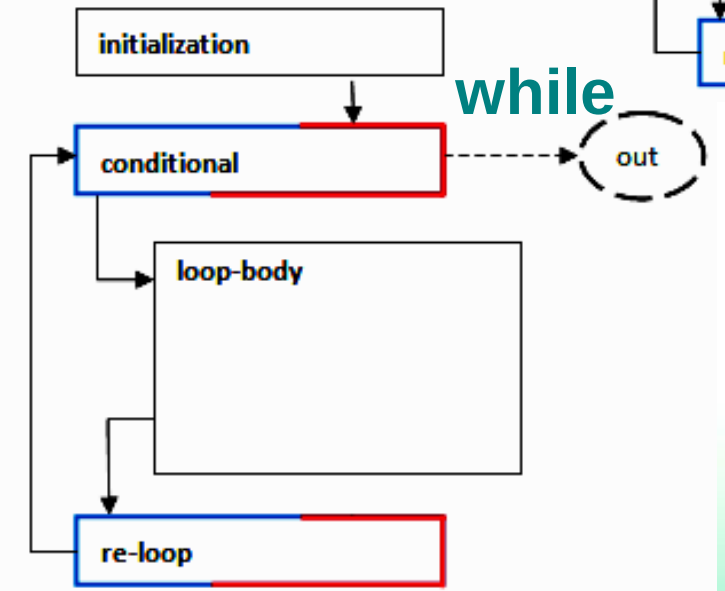
– **repetitia conditionata**

– o instructiune este executata **pana cand este aplicata tuturor elementelor dintr-o colectie** (**for**)

– sau **cat timp/pana cand** programul **atinge o anumita stare** (**while,** do..until, **do..while**)







Orientarea spre Obiecte (OO)

- **Evolutia catre OO**
  - Programarea **procedurala**

Programarea procedurala

Programarea procedurala

- este o forma avansata de **programare structurata**
  - care foloseste **conceptele structurale** de (**apel de**) **procedura**

Procedura

- este o forma de **incapsulare** si **modularizare** a **comportamentului** (executiei)
  - in jurul unui **bloc de cod**, care **regrupeaza** o **secventa de instructiuni**
  - oferind o **interfata publica** (**semnatura**) a blocului, formata din
    - **nume** prin care poate fi **apelat**
    - eventual **intrari** (argumente) **si iesiri** (valori returnate)
  - **ascunzand implicit detaliile** de **implementare** ale blocului
    - prin **scopul** limitat la nivelul **blocului** de cod al **variabilele locale**
    - si **neoferind alt acces** din exterior **la instructiunile regrupate**

Programarea procedurala

Denumiri alternative

– **procedura** – sugereaza ideea de **modalitate** de executie **standard** sau recomandata

– **rutina** – sugereaza ideea de **utilizare repetata**, de **reutilizare**

– **subprograme** – sugereaza ideea de **parte** dintr-un program, **modul**

– **subrutine** – combina ideea de **reutilizare** cu ideea de **modularizare**

– **functie** – sugereaza ideea de **dependenta** intre **intrari** si **iesiri** si originea matematica

– **metoda** – sugereaza ideea de **modalitate** de realizare/executie

Procedura ofera

– **abstractizare**, **modularizare** si **incapsulare** a **comportamentului** (executiei)

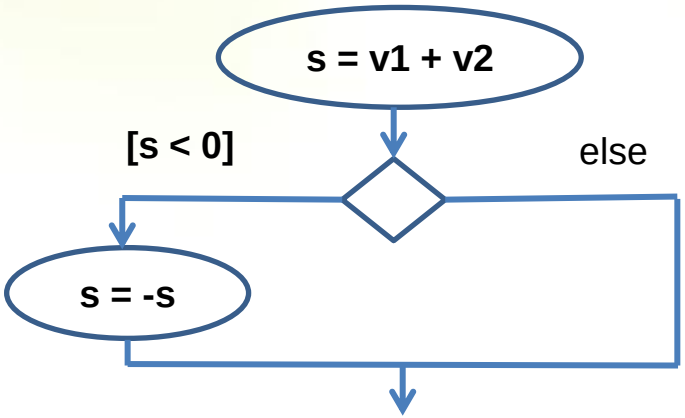
– facilitand **reutilizabilitatea**, **delegarea**, usurinta **intelegerii** si **modificarii**, **protectia**, etc., aspectelor **comportamentale** ale programelor

Programarea procedurala

Modularizarea comportamentului in C

Pornind de la portiunea de program pentru **"*calculul modulului sumei a doua valori*"**

```
s = v1 + v2; // suma
if (s<0)     // structura decizie
    s = -s;  // modul
```



Cum s-ar putea **modulariza calculul modulului?**

Cum ar arata **programul rescris** (*refactored*) in acest caz**?**

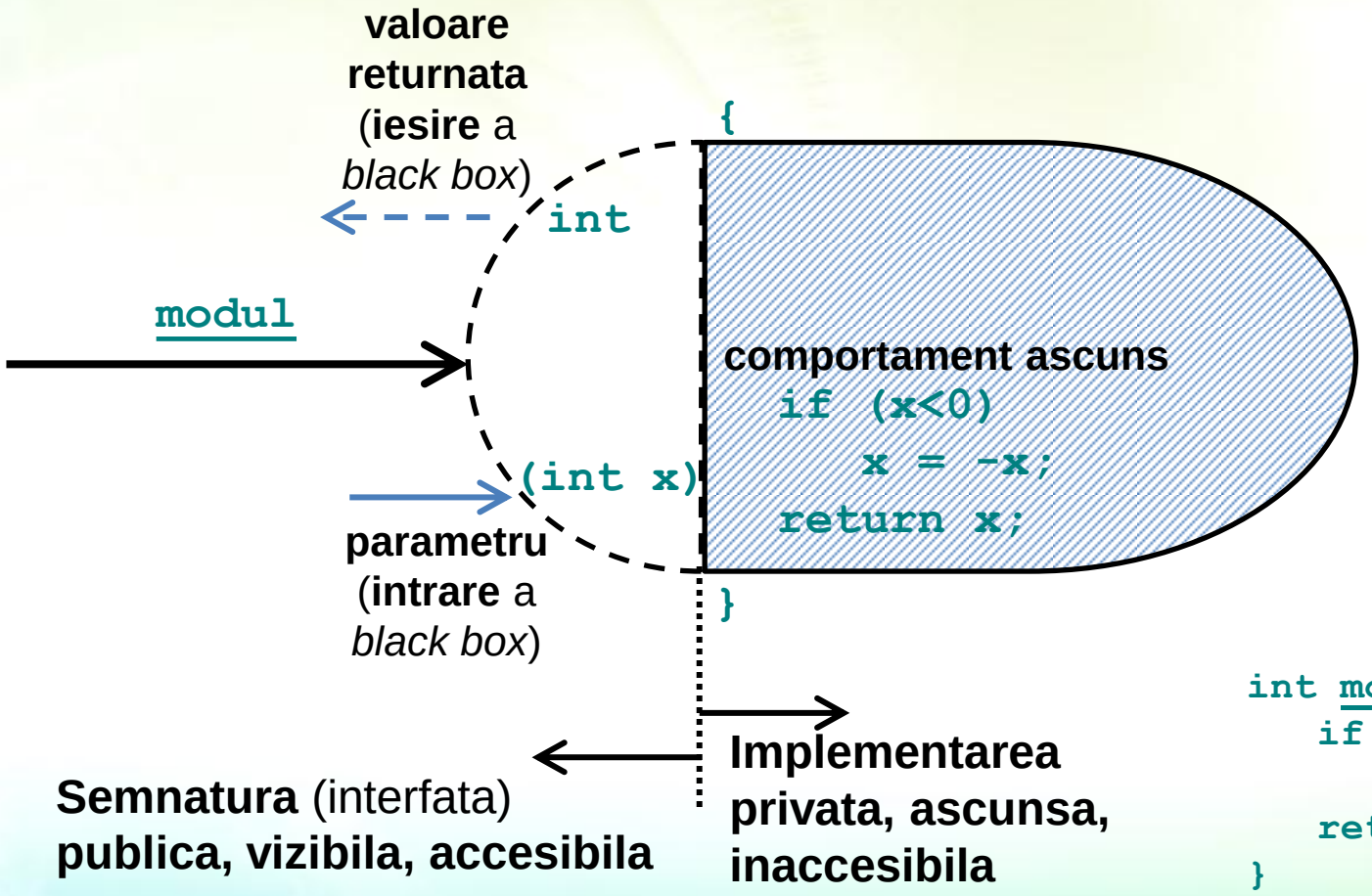
Programarea procedurala

Posibila modularizare a programului pentru **"*calculul modulului sumei a doua valori*"**

```
// declaratie functie (modul de comportament)
int modul(int x) {   // interfata publica (semnatura)
   if (x<0)          // implementare ascunsa
      x = -x;
   return x;
}
... {
   // ...
   s = v1 + v2;
   s = modul(s);     // apel functie (delegare functionala)
}
```

Programarea procedurala

Functia modul() vazuta ca modul procedural si componenta *black box*




```
int modul(int x) {
    if (x<0)
        x = -x;
    return x;
}
```

Programarea procedurala

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

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

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

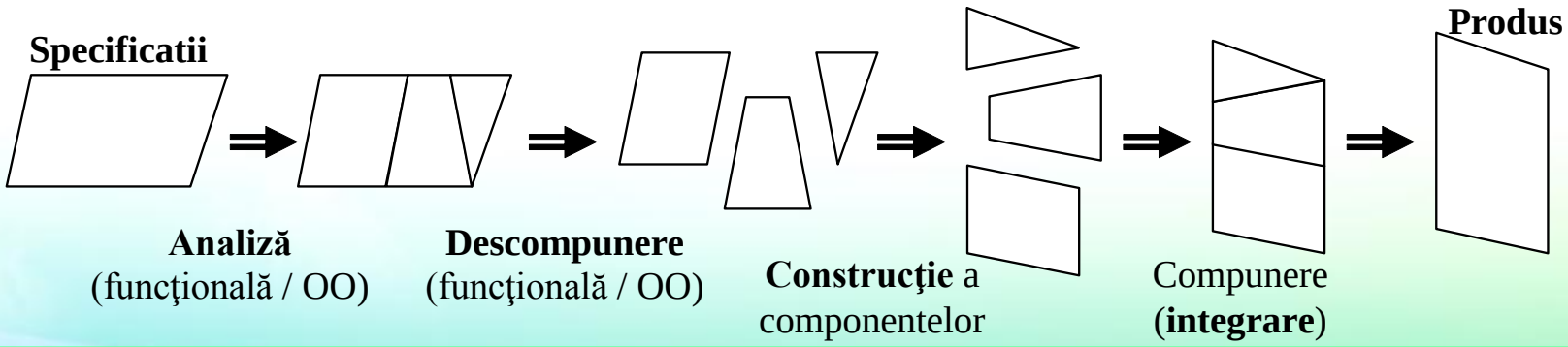
– a **intelege problema** si

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

– **compunerea** (**sinteza**) pentru

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

Cum corespund fazele de mai jos cu exemplul ecuatiei matematice?



Programarea procedurala

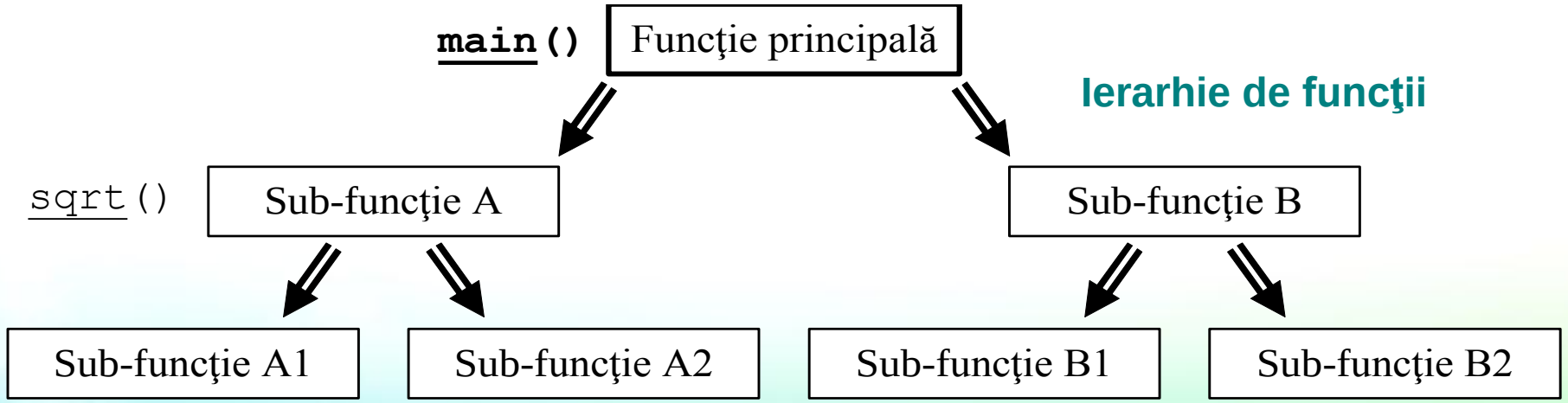
Procesul de **descompunere** traditional **procedural** (non-OO)

– este dirijat de **criteriul procedural** (functional):

– **identificarea funcţiilor**,

– **descompunerea** funcţiilor în **subfuncţii**,

aplicat în mod **recursiv**, ducand la elemente simple, atomice, direct implementabile (proceduri, funcţii, etc.)



Programarea procedurala

Abordarea procedurala

– propune **descompunerea** bazată pe ceea **ce face** programul

– considera **programul** ca fiind un **proces imens**

– care trebuie descoperit si **descompus**

– si odata **subproblemele rezolvate** trebuie **combinate solutiile** pentru a construi solutia intregii probleme

Mecanismele de integrare sunt

– **apelurile** de functii (**delegarile functionale**) si **ierarhizarea** lor

Arhitectura programului reflecta DOAR **functiile** lui (**functiile decid structura**)

⇒ **evolutiile functionale** pot implica **modificari structurale** puternice

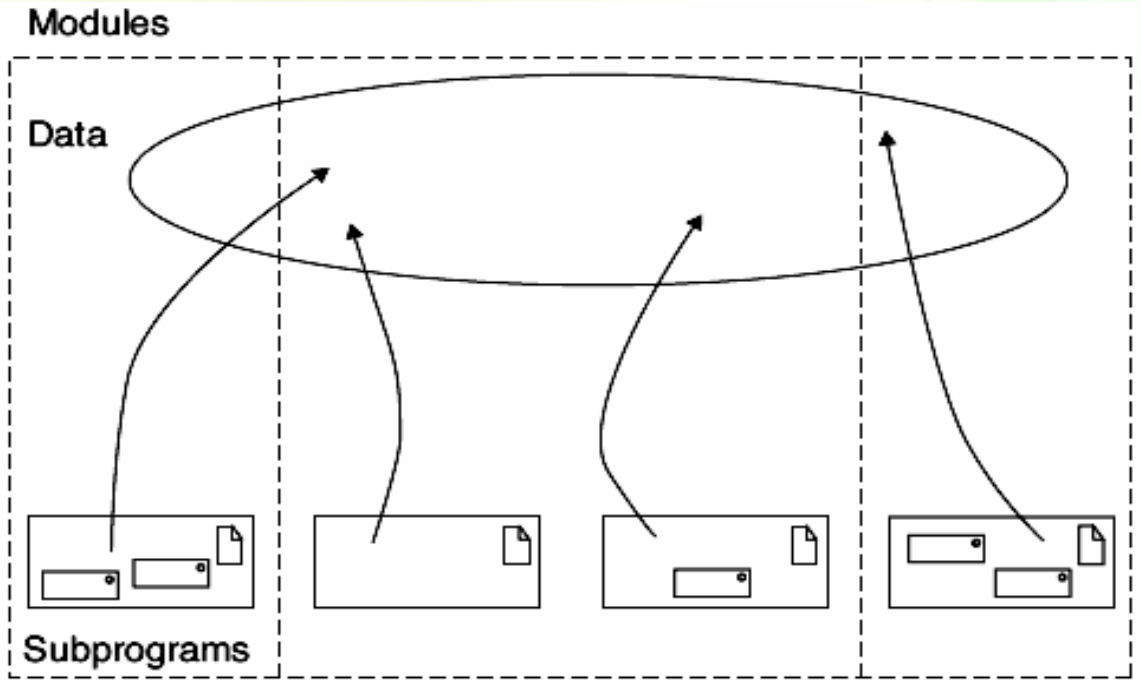
Pentru rezultate bune functiile cerute **trebuie sa fie bine identificate** si **stabile** in timp (ceea ce **se intampla foarte rar**)

Programarea procedurala

Abordarea descompunerii procedurale se bazeaza pe:

– **programare structurata** si

– **modularizare functionala**



Datele globale insa

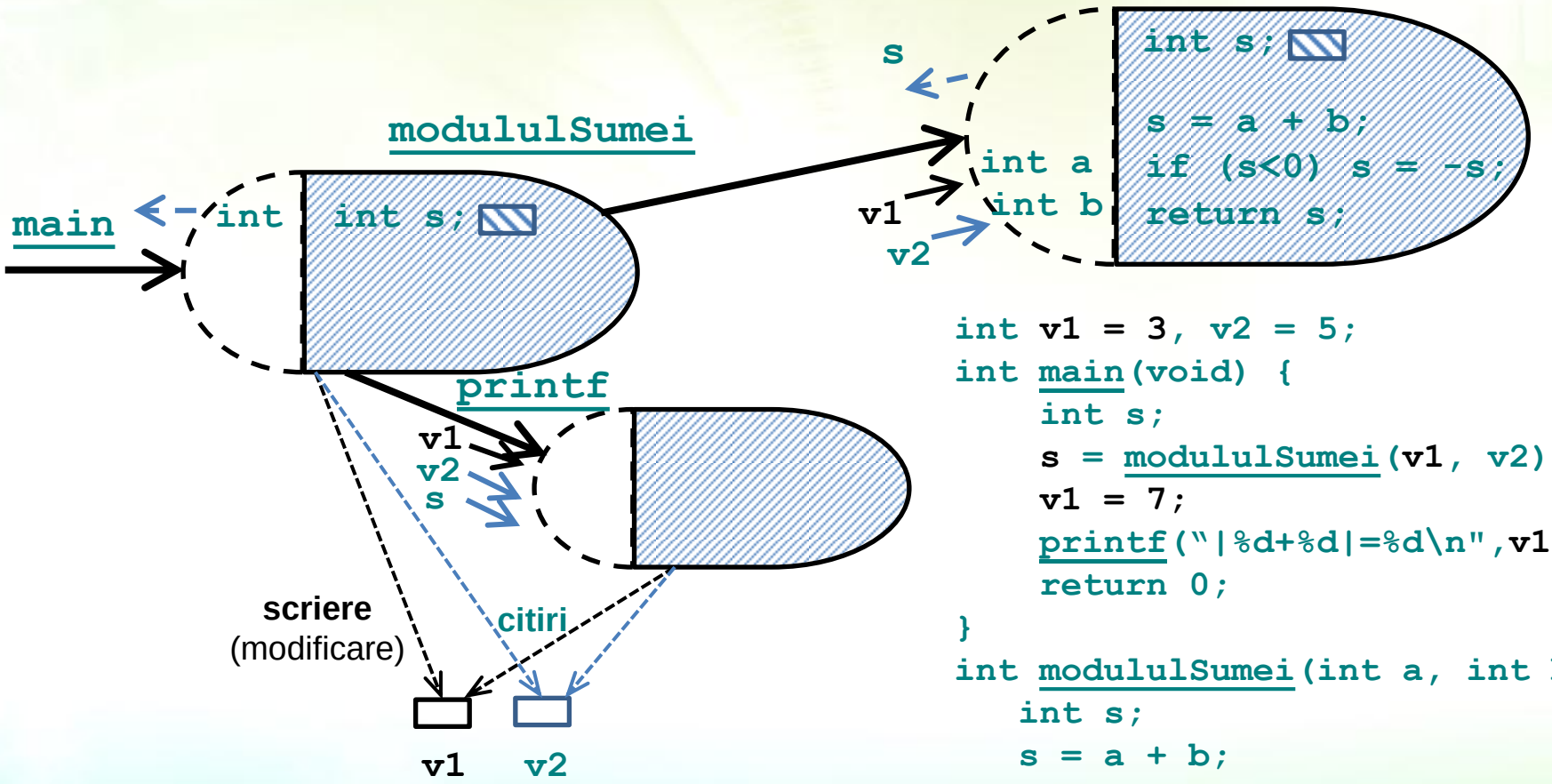
– **nu sunt protejate** si pot deveni usor **eronate** sau **incoerente**

In plus, **orice modificare intr-o structura de date**

– **se propaga in toate functiile** care o folosesc

Programarea procedurala

Problemele create de accesul neingradit la variabile globale




```
int v1 = 3, v2 = 5;
int main(void) {
     int s;
     s = modululSumei(v1, v2);
     v1 = 7;
     printf("|%d+%d|=%d\n",v1,v2,s);
     return 0;
}
int modululSumei(int a, int b) {
    int s;
    s = a + b;
    if (s<0) s = -s;
    return s;
}
```

⇒ ***Pierderea consistentei variabilei v1***

Orientarea spre Obiecte (OO)

- **Evolutia catre OO**
  - Tipurile de date abstracte (**ADT**)
    - curs **SDA** (anul 1)

Curs 2 – sapt.3 /
Curs 3 – sapt.4

Tipurile de date abstracte (ADT)

Tipul de date abstract (ADT)

– este definit ca un **model** (abstract)

– creat pentru a **captura esenta** unui **domeniu al problemei** (subdomeniu al **realitatii**)

– cu scopul de a fi **transformat** inttr-un **program de calcul**

– exemple: **coada**, **lista**, **stiva**, **arbore**, **graf**, etc.

– poate fi definit si ca **entitate** care consta

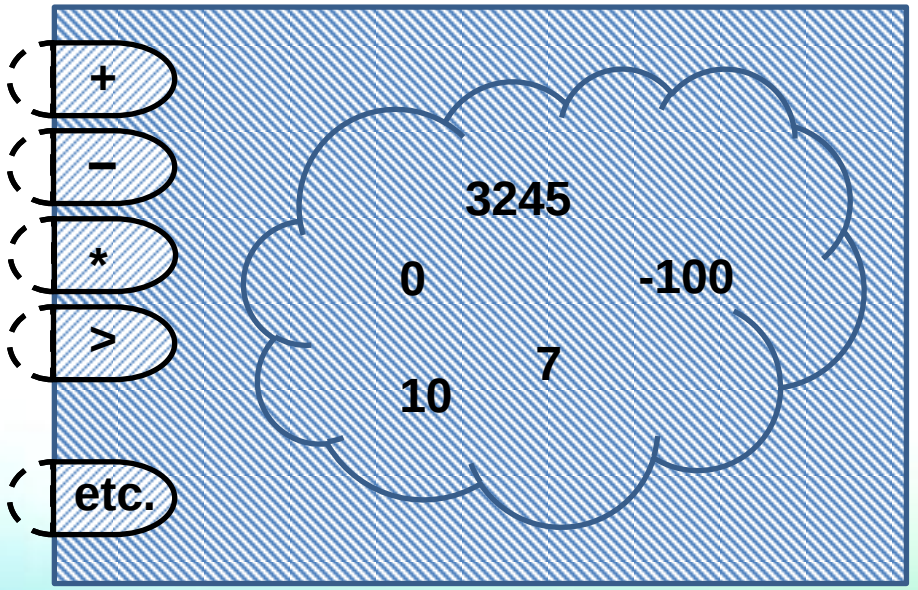
– dintr-o **multime** de **valori** si

– dintr-o **colectie** de **operatii** care **prelucreaza** acele valori

Tipurile de date abstracte (ADT)

Tipul de date abstract (ADT) – exemple

- **Integer** consta din
  - o **multime** de **valori** pozitive si negative de numere intregi si 0
  - **colectie** de **operatii** care prelucreaza acele valori, cum ar fi
    - adunare, scadere, inmultire, comparatii (egalitate, inegalitati, etc.)



Tipurile de date abstracte (ADT)

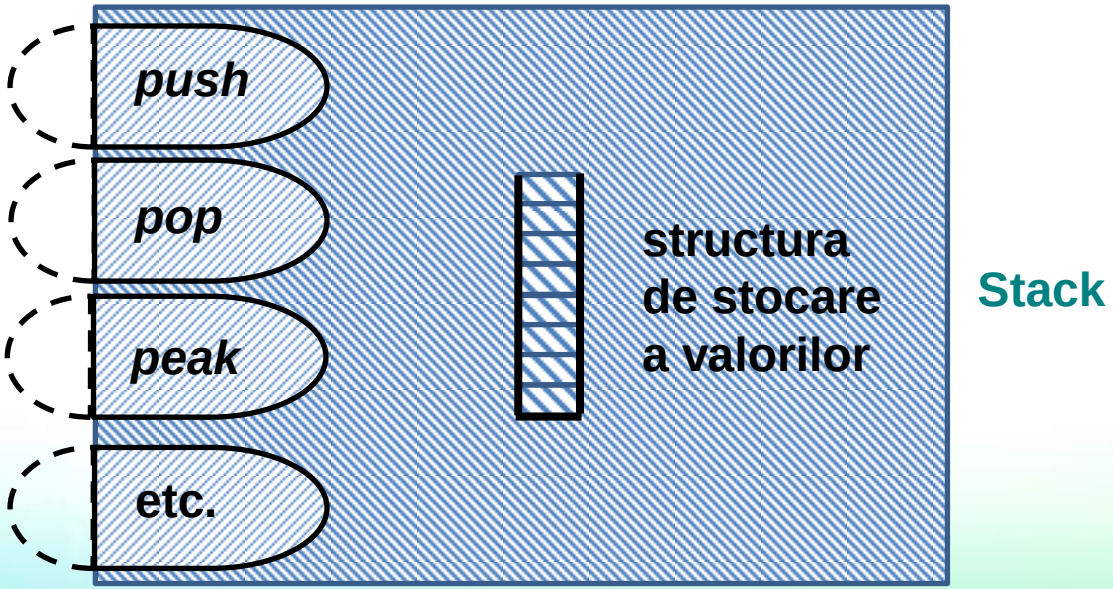
Tipul de date abstract (ADT) – exemple

– ***Stack* (stiva)** consta din

– structura de stocare a **valorilor** (cu regula “ultimul adaugat, primul extras”)

– **colectie** de **operatii** care prelucreaza acele valori, cum ar fi

– ***push*** (adaugare), ***pop*** (extragere), ***peak*** (ultima valoare adaugata)



Tipurile de date abstracte (ADT)

Tipul de date abstract (ADT)

– este **important** in programare deoarece

– ofera o **cale clara si riguroasa** de a **specifica**

– **datele** pe care un **program** trebuie sa le **prelucreze**

– si **modurile in care** programul **trebuie sa isi prelucreze datele**

– **fara referire la detalii**

– despre **modul de reprezentare** a datelor sau

– despre **cum sunt implementate** operatiile

Tipurile de date abstracte (ADT)

Tipul de date abstract (ADT)

– odata **inteles** si **documentat** un **ADT**

– serveste ca **specificatie** pe care programatorii o pot utiliza ca **ghid pentru alegerea reprezentarii** datelor **si a implementarii** operatiilor

– si ca **standard** pentru **asigurarea corectitudinii** programelor

– o **realizare** (**concretizare**) a unui **ADT**

– ofera **reprezentari ale valorilor** din multimea de valori

– ofera **algoritmi pentru operatiile sale**

– si este denumita **tip de date**

Tipurile de date abstracte (ADT)

Tipurile de date abstracte (ADT)

– pot fi **implementate** prin **tipuri de date** sau **structuri de date specifice**

– in mai multe moduri si in mai multe **limbaje de programare**

– sau **descrise** intr-un **limbaj de specificare formala**

– sunt adesea **implementate** ca **module**

– **interfata modulului** declarand **procedurile** care corespund **operatiilor** ADTului, uneori cu comentarii care descriu **constrangerile** acestora

– strategie de **ascundere a informatiilor** care

– permite ca **implementarea** unui modul sa fie **schimbata fara a afecta** programele client / utilizator

Incapsularea duala

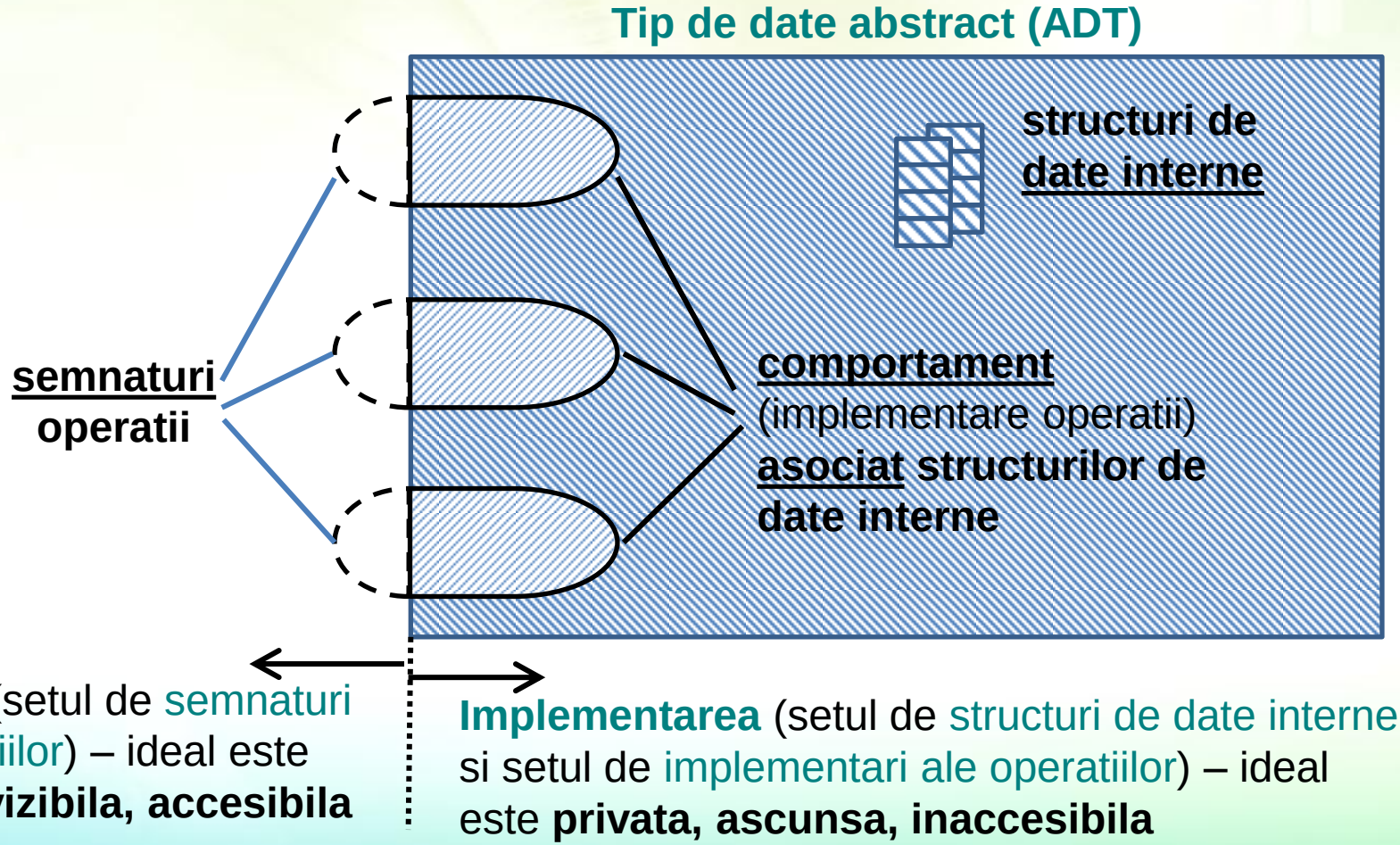
Incapsularea oferita de ADTuri este o forma de **incapsulare duala**

- a **informatiilor** (**datelor**)
  - **regruparea** elementelor de date in
    - **structuri / tipuri de date complexe**

      (stive, cozi, liste, arbori, etc.)
  - ideal **CU ascunderea detaliilor (datelor, informatiilor)**
- a **comportamentului**
  - **regruparea** elementelor de comportament in
    - **sub-module procedurale** (**operatii**)
  - implicit **CU ascunderea detaliilor (implementarii)**

Incapsularea duala

Incapsularea oferita de ADTuri este o forma de **incapsulare duala**



Orientarea spre Obiecte (OO)

- **Evolutia catre OO**
  - Modelare si **abstractizare (II)**

Modelare si abstractizare

Abstractizarea in general poate fi **aplicata**

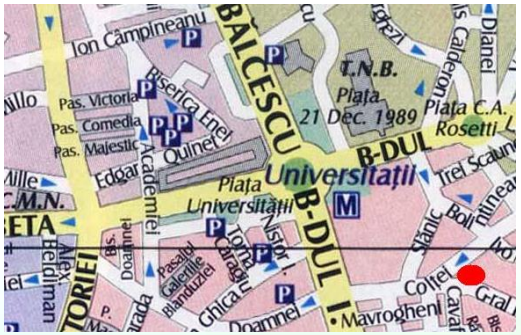
– unor **entitati** (**AE**)

– din care sunt obtinute **modele esentiale abstracte**

– care sunt **perspective** **simplificate** **ale entitatilor**

– de ex. un **teritoriu** ➡ o **harta**

AE ➡



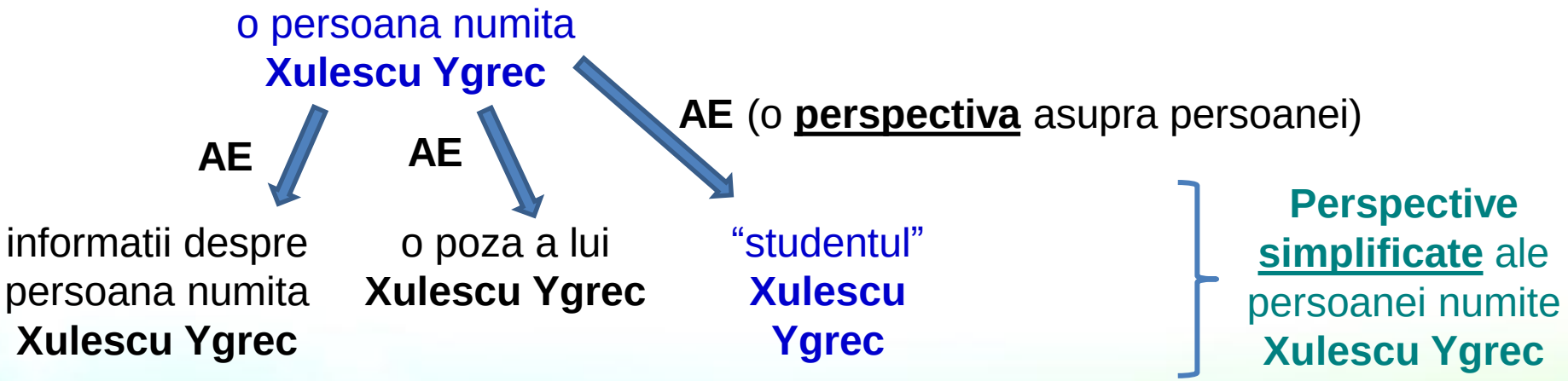
Modelare si abstractizare

Abstractizarea in general poate fi **aplicata**

– unor **entitati** (**AE**)

– din care sunt obtinute **modele esentiale abstracte**

– care sunt **perspective** **simplificate** **ale entitatilor**



Modelare si abstractizare

Abstractizarea informatica (**AI**)

– **aplicata** unor **entitati** (**AE**)

– produce **modele informatice** **(esentiale, abstracte)**

– de exemplu

– **pornind de la perspectiva "student"** a persoanei **Xulescu Ygrec**

studentul
Xulescu Ygrec

AE → inregistrari intr-o baza de date despre studentul **Xulescu Ygrec**

AE → **altele?**

Abstractizari informatice (**AI**) ale studentului numite **Xulescu Ygrec**

Modelare si abstractizare

Abstractizarea orientata spre obiecte (**AOO**)

– caz particular de **abstractizare informatica** (**AI**)

– **aplicata** unei **entitati** (**AE**)

– produce un **element al modelului OO** al realitatii

– numit ***obiect software*** sau pe scurt **obiect**

studentul
Xulescu Ygrec

AE → inregistrari intr-o baza de date despre studentul **Xulescu Ygrec**

AE → obiectul software **pXY**

Abstractizari informatice (**AI**) ale studentului numite **Xulescu Ygrec**

Modelare si abstractizare

Abstractizarea in general poate fi **aplicata**

– unor **multimi de entitati** (**AM**)

– din care sunt obtinute **categorii**, **tipuri**, **clase** de entitati

– care **descriu <u>ceea ce este comun</u> multimii de entitati**

– fiind **generalizari** **(vederi de ansamblu)** ale multimilor de entitati

AM ⇩

Conceptul, categoria, etc., de
“Student”

Modelare si abstractizare

Revenind la abstractizarile aplicate unor entitati (AE)

o persoana numita
Xulescu Ygrec

AE (o **perspectiva** asupra persoanei)

studentul
Xulescu
Ygrec

Perspectiva simplificata a persoanei numite **Xulescu Ygrec**

Modelare si abstractizare

Combinand abstractizarile entitatilor (AE) su abstractizarea multimii (AM)

AE (**perspectiva** asupra persoanelor)

AE

studentul **Xulescu Ygrec**

studentul **Wescu Zet**

studentul **…**

Perspectiva simplificata a persoanelor (din categoria "Student")

(**vedere de ansamblu** asupra studentilor) **AM**

Conceptul, categoria, etc., de **"Student"**

Abstractizare a tuturor persoanelor care sunt studenti (**descriere a ceea ce este comun tuturor studentilor**)

Modelare si abstractizare

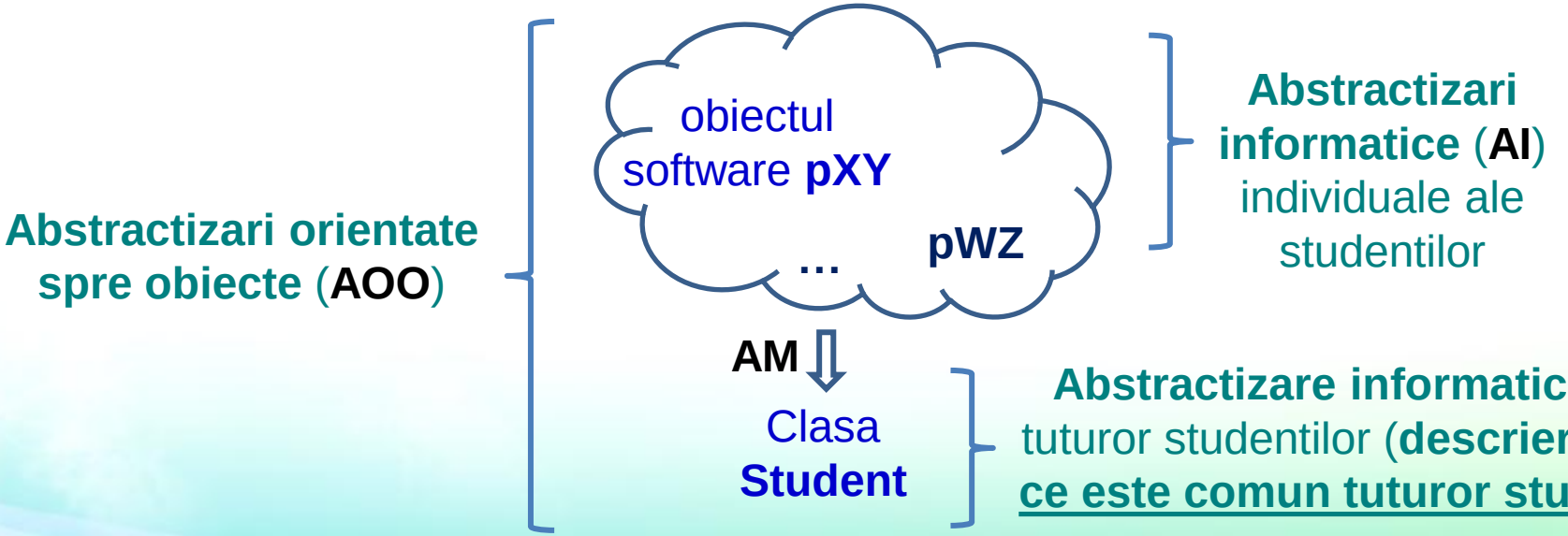
Abstractizarea <u>orientata spre obiecte</u> (AOO)

– caz particular de **abstractizare informatica** (**AI**)

– **aplicata** unei **multimi de entitati** (**AM**)

– produce **categorii**, **tipuri** ale modelului OO

– numite **<u>clase de obiecte</u>** software sau pe scurt **<u>clase</u>**



Modelare si abstractizare

Combinand abstractizarile entitatilor (AE) su abstractizarea multimii (AM)

AE **perspectiva**

AE

obiectul software **pXY**

pWZ

...

Abstractizari informatice (AI) individuale ale studentilor

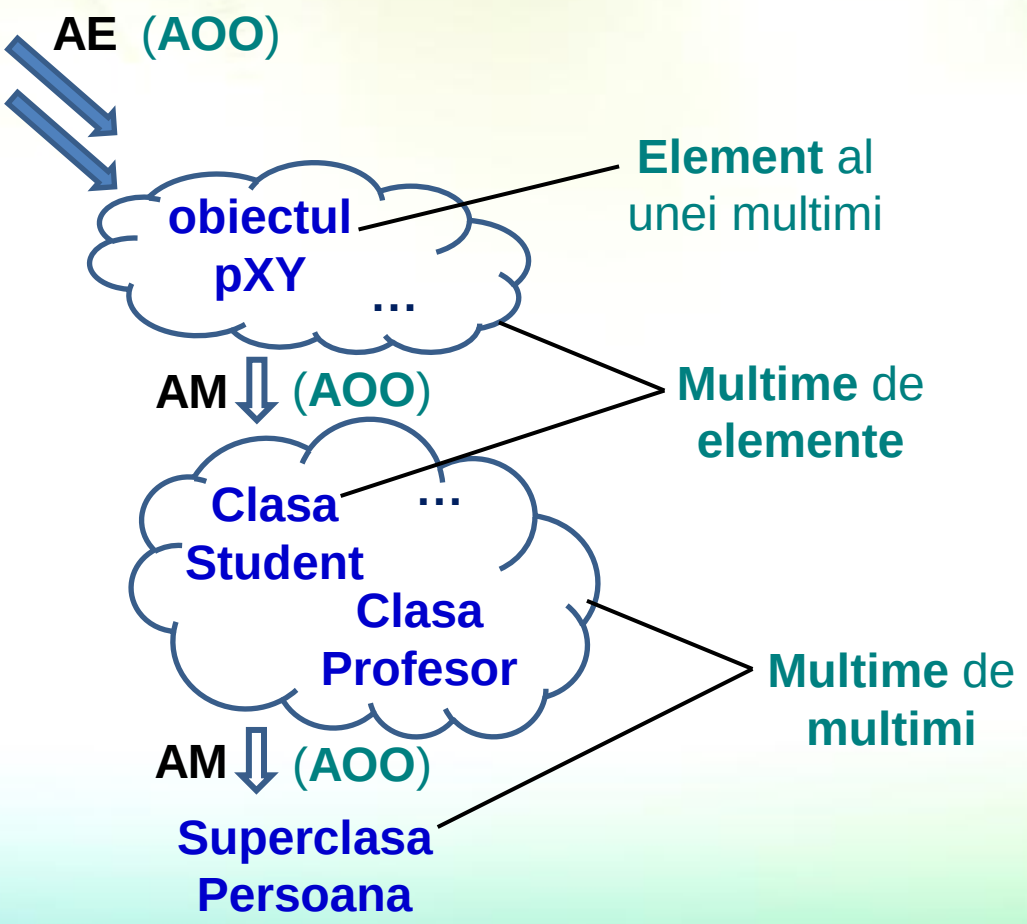
vedere de ansamblu **AM**

Clasa **Student**

Abstractizare informatica (AI) a tuturor studentilor (**descriere a ceea ce este comun tuturor studentilor**)

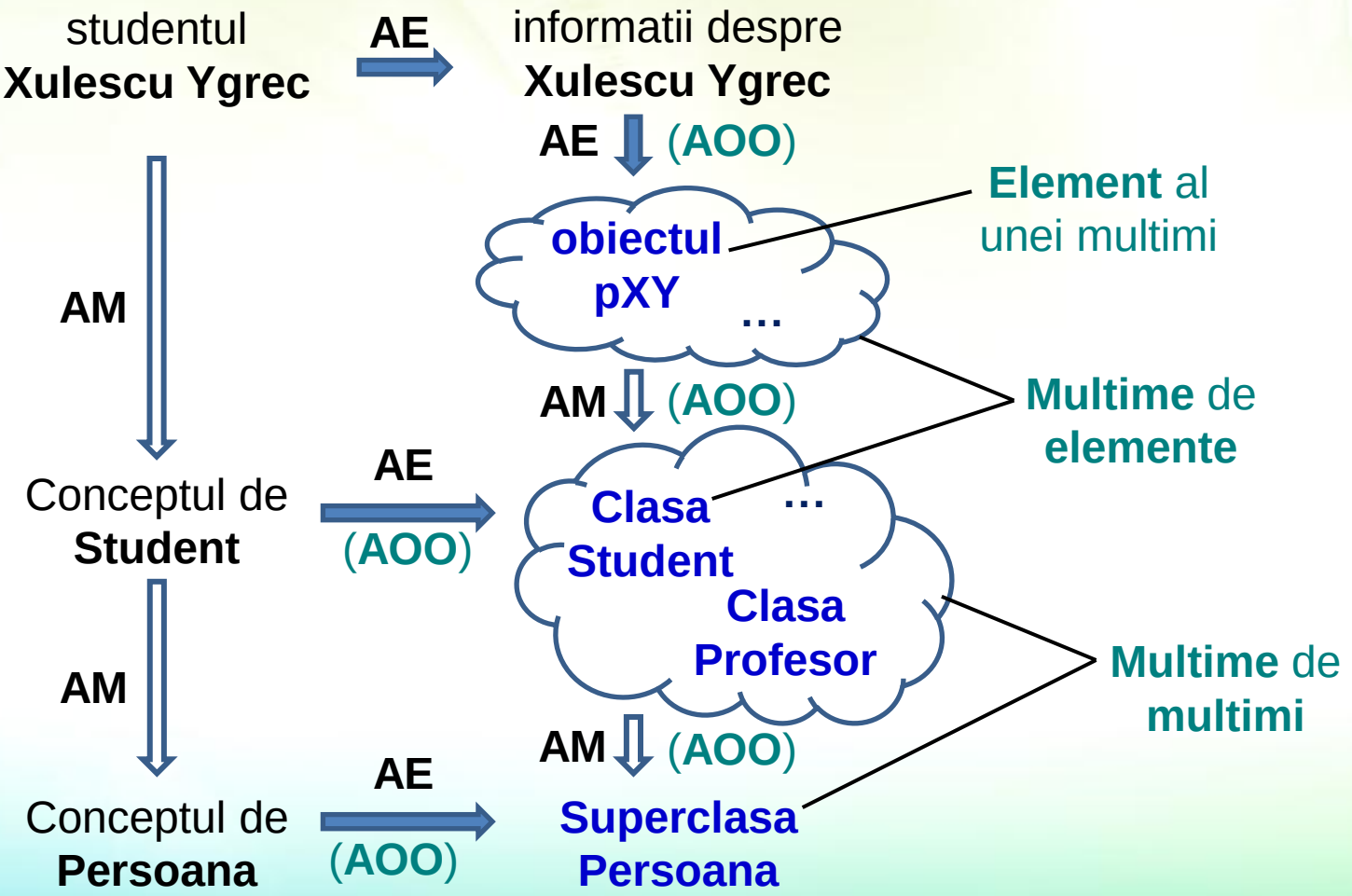
Modelare si abstractizare

Mergand mai departe, putem abstractiza multimile de multimi ca superclase



Modelare si abstractizare

Paralela intre abstractizarile din lumea reala si abstractizarile informatice OO



Modelare si abstractizare

Revenire asupra exemplului de abstractizare "*masina de calcul si limbajele*"

Entitate abstractizata	*Model abstract*	*Model si mai abstract*	*Modelul cel mai abstract*
Masina de calcul condusa de **coduri masina** + **Limbajul masina**	**Masina de calcul** programata la **nivel de asamblare** + **Limbajul de asamblare**	**Masina de calcul** programata la **nivel inalt** + **Un limbaj de nivel inalt**	**Masina de calcul** programata **orientat spre obiecte** + **Un limbaj orientat spre obiecte**
(**limbajul masinii** de calcul)	(**echivalent uman** al limbajului masinii)	(**forma concentrata** a limbajului masinii, **apropiata de cea umana**)	(**forma intuitiva din punct de vedere uman**)

CONCRET ⟵ ***Ierarhie de abstractizari*** ⟶ **ABSTRACT**

Modelare si abstractizare

Revenire asupra exemplului de abstractizare "*masina de calcul si limbajele*"

Entitate abstractizata	*Model abstract*	*Model si mai abstract*	*Modelul cel mai abstract*
Masina de calcul condusa de **coduri masina** + **Limbajul masina**	**Masina de calcul** programata la **nivel de asamblare** + **Limbajul de asamblare**	**Masina de calcul** programata la **nivel inalt** + **Un limbaj de nivel inalt**	**Masina de calcul** programata **orientat spre obiecte** + **Un limbaj orientat spre obiecte**
Ex. de concepte: – coduri instructiuni – registre – locatii memorie – operanzi – adrese	**Ex. de concepte:** – mnemonici instructiuni – registre – variabile – salturi – operanzi – call procedura	**Ex. de concepte:** – decizii – variabile – structuri de date – expresii – apeluri functii – blocuri de cod – pointeri	**Ex. de concepte:** – obiecte, clase – operatii, atribute – interfete – mesaje – stari – colaborari – referinte

Modelare si abstractizare

Revenire asupra exemplului de abstractizare "*entitatile lumii reale*"

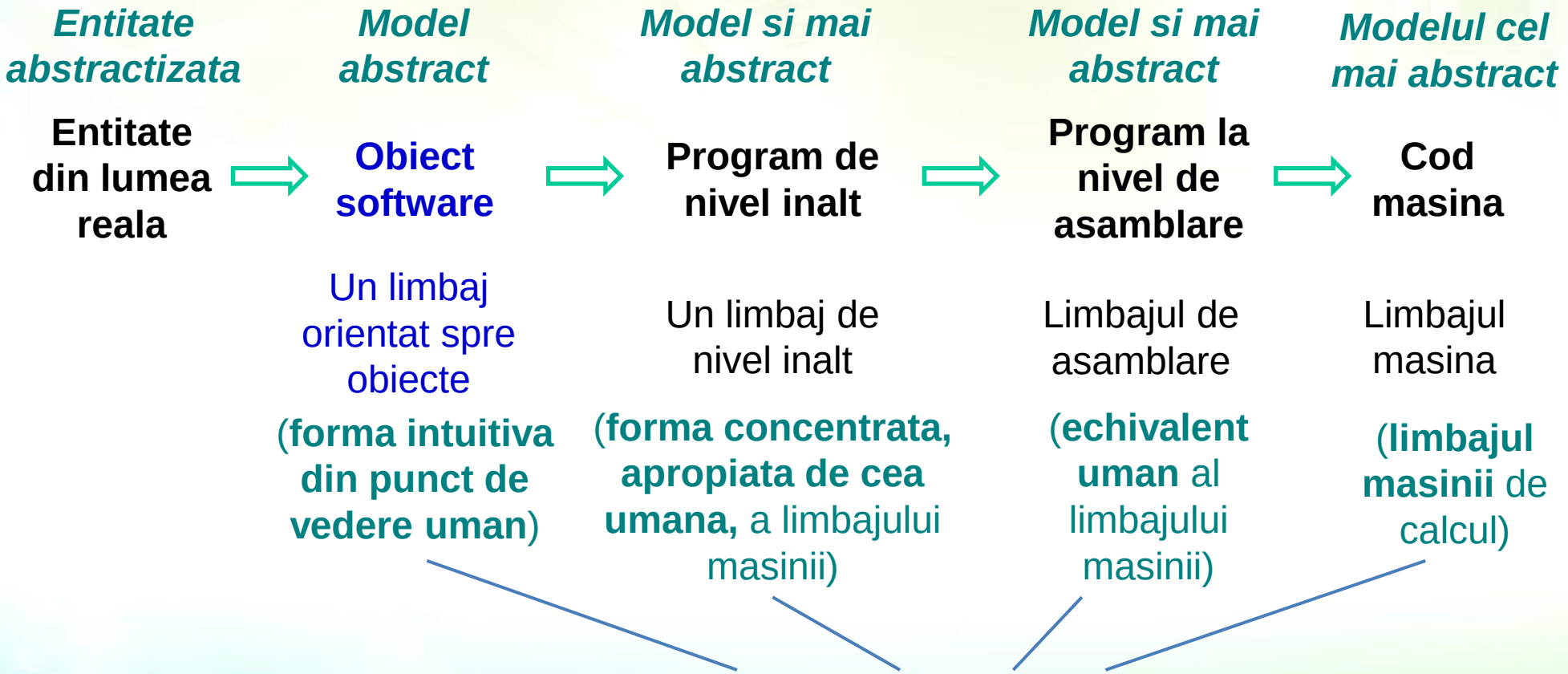
Entitate abstractizata

Entitate din lumea reala ⇨ **Cum poate fi abstractizata informatic?**

CONCRET

Modelare si abstractizare

Revenire asupra exemplului de abstractizare "*entitatile lumii reale*"



Modelare si abstractizare

Recapitulare a exemplelor de abstractizare care reflecta relativitatea conceptului

Limbaj de programare	Format	Controlul executiei realizat prin	Folosit in	Cat de abstract este	
				pentru masina	pentru om
cod masina	numeric binar	**salturi** conditionate, iteratii catre etichete	masina de calcul	deloc	**extrem de mult**
asamblare	alfa-numeric	**salturi** conditionate, iteratii simplecatre etichete	programarea la nivel asamblare	destul de mult	foarte mult
procedural (nivel inalt)	alfa-numeric	**decizii** (simple+multiple), iteratii (mai multe tipuri)	programarea la nivel inalt	mult	mult
orientat spre obiecte (OO)	alfa-numeric	**colaborarea** obiectelor plus decizii si iteratii	programarea OO	**foarte mult**	**destul de putin**