

2010 - 2011

Programare Orientata spre Obiecte (*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 (attribute)

- 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

2.1. Obiecte si clase. Metode (operatii) si campuri (attribute)

- **Obiecte si clase**

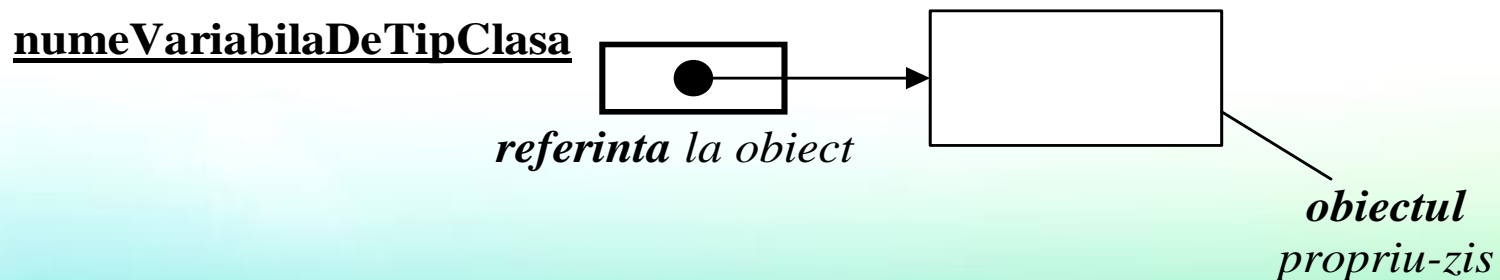
- **Clasa si obiectul**
- **Constructorii**
- Exemple de cod
- **Campurile** Java (atributele) si starea obiectului
- **Metodele** Java (operatiile) si comportamentul obiectului
- **Semnaturile** metodelor
- Metodele ca **mesaje**, **colaborarea** intre obiecte
- **Crearea si legarea** obiectelor

Obiecte si clase

- Clasa si obiectul

Clasa

- **tip de date**
 - **tipar** dupa care sunt construite variabile numite obiecte dar si
 - **domeniu de definitie** (asemanator unei multimi) pentru obiecte
- **structura complexa** care reuneste
 - elemente de date, numite - **attribute** in **orientarea spre obiecte**
- **campuri** in **Java**, si
 - algoritmi, numiti - **operatii** in **orientarea spre obiecte**
- **metode** in **Java**
- in **Java** este **tip referinta** - obiectele sunt accesate printr-o **referinta**, care contine adresa obiectului propriu-zis

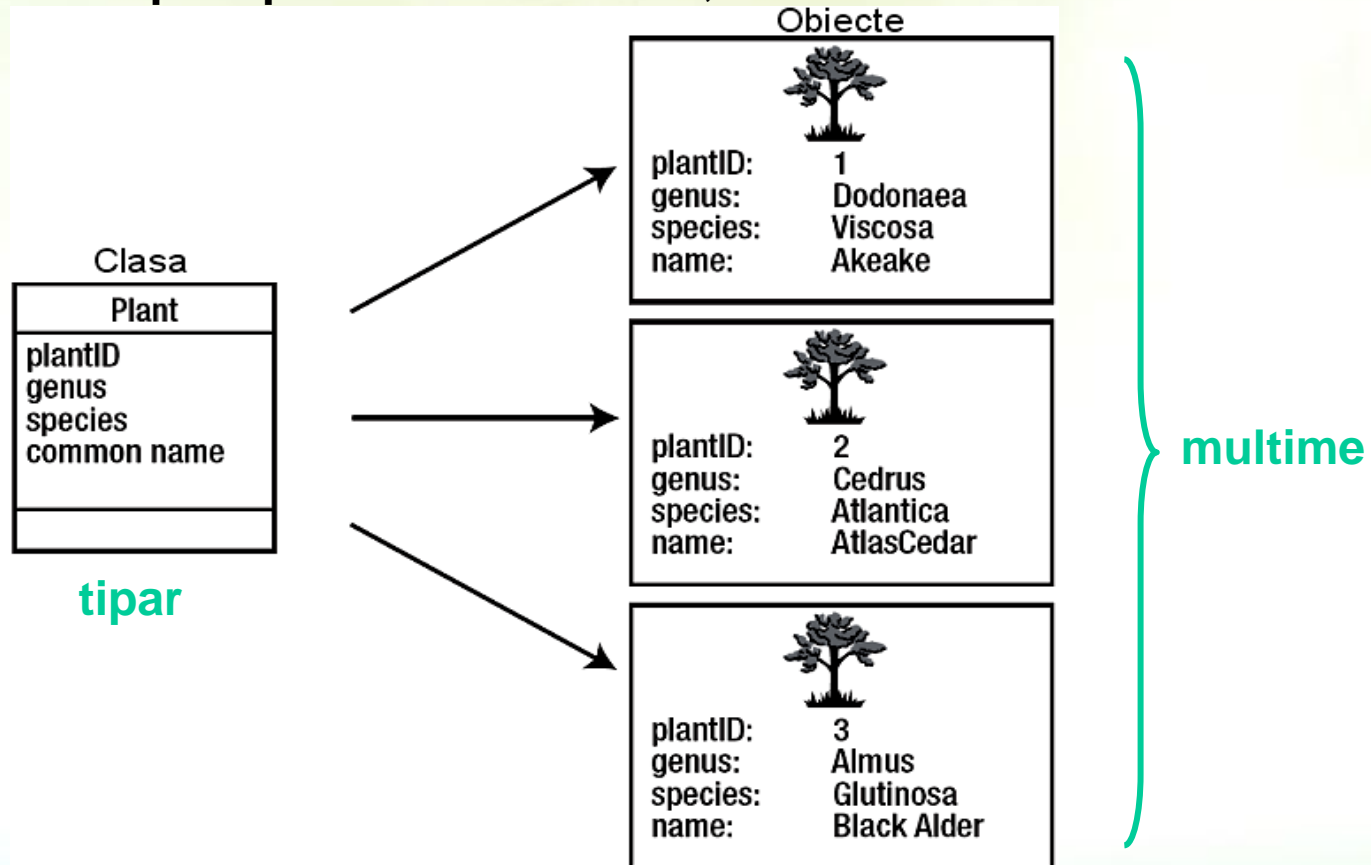


Obiectul

- **reprezentare abstractă** a unor **entități reale sau virtuale**
- caracterizată de:
 - **identitate**, prin care acesta este deosebit de alte obiecte
 - implementata **in Java** ca **variabila referinta la obiect**
 - **comportament accesibil** altor obiecte
 - implementat ca **set de operatii** (metode, functii membru)
 - **stare internă ascunsă**, proprie
 - implementata ca **set de attribute** (campuri, variabile membru)
- **exemplu specific** al unei clase, numit **instanta** a clasei

Clasa si obiectele

Obiectul este **exemplu specific** al unei **clase**, numit **instanta** a clasei



Clasa este

- tipar dupa care sunt construite variabile numite obiecte
- domeniu de definitie (asemanator unei multimi) pentru obiecte

Clasa

Definitia clasei in Java (simplificata)

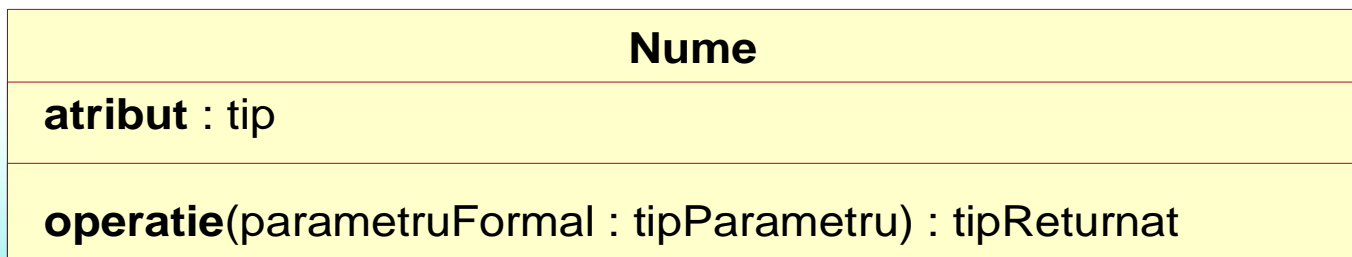
```
public class Nume { // declaratie tip/structura de date

    // declaratie atribut (variabila membru, camp Java)
    tip atribut;

    // semnatura operatie (metoda Java)
    tipReturnat operatie(tipParametru parametruFormal) {

        // corpul functiei membru (metodei)
        // returneaza valoare de tipul tipReturnat
    }
}
```

In UML clasa definita mai sus se reprezinta astfel



Obiectul

Pentru a trata distinct obiectele e necesara utilizarea unor **nume diferite**, care in Java sunt **variabile referinta** la/catre obiecte

```
// declararea variabilei referinta la obiect  
NumeClasa numeObiect;
```

numeObiect null *referinta obiect de tip*
NumeClasa

Prin simpla declarare se creaza **doar spatiul necesar variabilei referinta**, care are **valoarea implicita null** (referinta catre nimic)

In **UML** obiectul declarat mai sus se reprezinta astfel

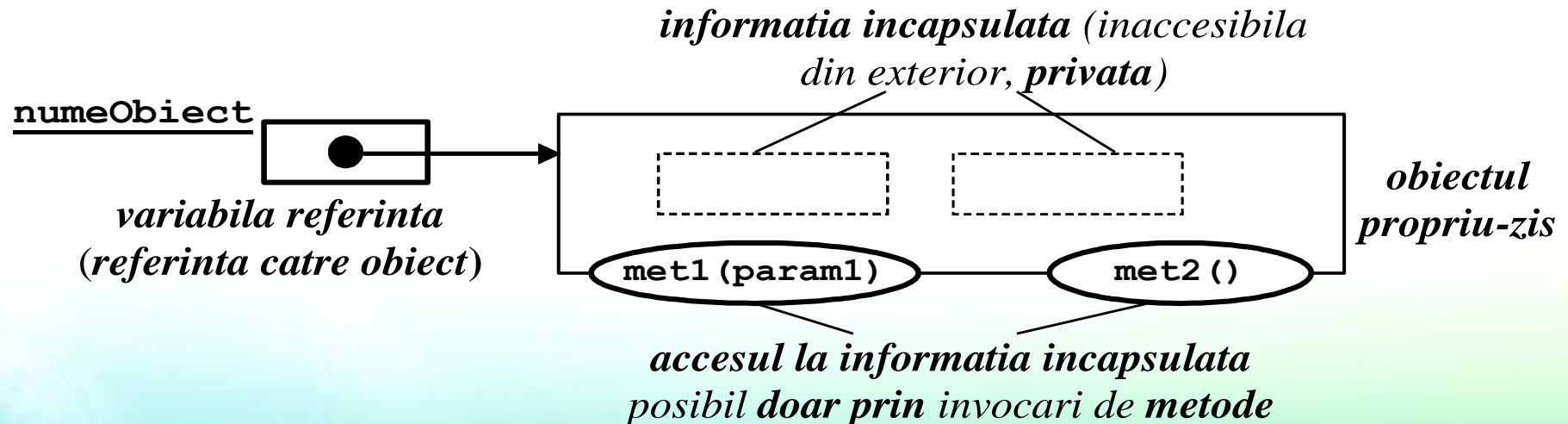
numeObiect : NumeClasa

Obiectul

Alocarea memoriei pentru obiect si atribuirea referintei catre acesta se realizeaza cu ajutorul

- constructorului clasei obiectului si al
- operatorului **new** de alocare dinamica a memoriei

```
// crearea dinamica a obiectului  
numeObiect = new NumeClasa(tipParametru parametruActual);
```



Obiecte si clase

- Constructorii

Clasa si obiectele

Constructorul Java

- **tip special de functie**, care
 - are **acelasi nume** cu numele **clasei** in care este declarat
 - este **utilizat** pentru a **initializa orice nou obiect de acel tip**
 - stabilind valorile campurilor/ atributelor obiectului, in momentul crearii lui dinamice
 - **nu returneaza** nici o valoare
 - are ca si metodele obisnuite aceleasi
 - niveluri de accesibilitate
 - reguli de implementare a corpului
 - reguli de supraincarcare a numelui

```
NumeClasa(tipParametru parametruActual);
```

Clasa si obiectele

Constructorul

- stabileste valorile initiale ale tuturor atributelor unui obiect nou
- ducand astfel obiectul in **starea initiala**

In Java nu este neaparat necesara scrierea unor constructori pentru clase, deoarece

- un constructor implicit este generat automat de sistemul de executie
- **DOAR** pentru o clasa care nu declara explicit constructori

Acest constructor nu face nimic (nici o initializare, implementarea lui continand un bloc de cod vid: { })

```
NumeClasa () {}
```

De aceea

- orice initializare dorita **explicit** impune **scrierea unor constructori**

Obiecte si clase

- Exemple de cod

Exemplu de clasa

```
public class Point {  
    // atribute (variabile membru)  
    private int x;  
    private int y;  
  
    // operatie care initializeaza attributele = constructor Java  
    public Point(int abscisa, int ordonata) {  
        x = abscisa;  
        y = ordonata;  
    }  
  
    // operatii care modifica attributele = metode (functii membru)  
    public void moveTo(int abscisaNoua, int ordonataNoua) {  
        x = abscisaNoua;  
        y = ordonataNoua;  
    }  
    public void moveWith(int deplasareAbsc, int deplasareOrd) {  
        x = x + deplasareAbsc;  
        y = y + deplasareOrd;  
    }  
  
    // operatii prin care se obtin valorile atributelor = metode  
    public int getX() { return x; }  
    public int getY() { return y; }  
}
```

} Declaratii
(specificare)
atribute

} Semnaturi
(declaratii,
specificari)
operatii
+
Implementari
(corpuri)
operatii

Exemplu de clasa utilizator pentru clasa anterioara

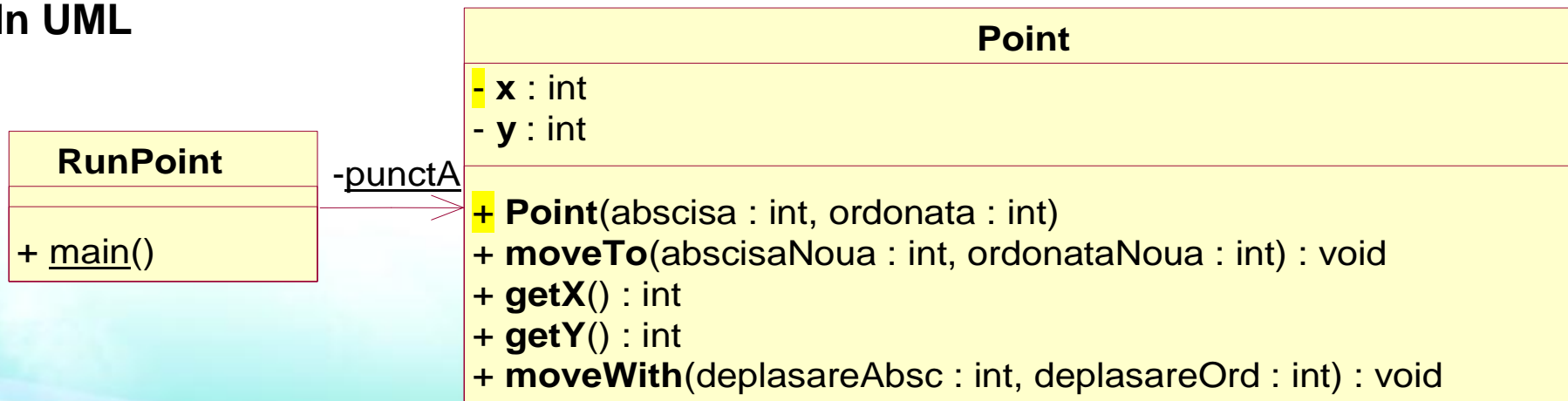
```
// clasa de test pentru clasa Point
public class RunPoint {
    private static Point punctA;           // atribut de tip Point

    public static void main(String[] args) { // declaratie metoda
        // corp metoda
        punctA = new Point(3, 4); // alocare si initializare atribut punctA

        punctA.moveTo(3, 5); // trimitere mesaj moveTo() catre punctA

        punctA.moveWith(3, 5); // trimitere mesaj moveWith() catre punctA
    }
}
```

In UML



Obiecte si clase

- Campurile Java (atributele) si starea obiectului

Exemplu de creare si utilizare a unui obiect

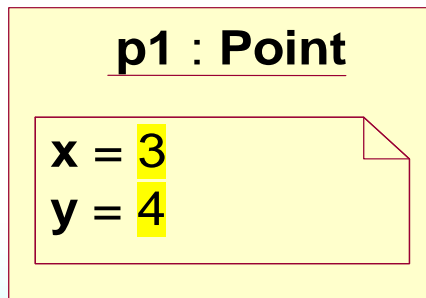
Crearea unui obiect

- numit **p1** de tip **Point**
- ale carui **attribute** au valorile **x=3** si respectiv **y=4**

```
// crearea dinamica si initializarea obiectului  
Point p1 = new Point(3, 4);
```

Obiectul numit p1

- **abstractizeaza** si
- **incapsuleaza informatiile** care privesc un **punct in plan** de coordonate **{3, 4}**



Starea initiala a obiectului **p1** este **perechea de coordonate {3, 4}**

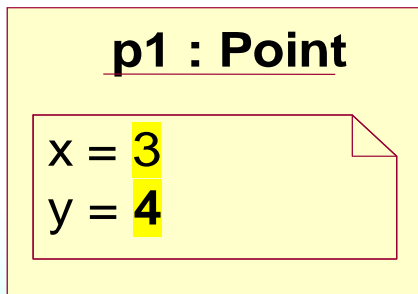
Exemplu de creare si utilizare a unui obiect

Schimbarea starii obiectului p1 din {3, 4} in {3, 5}

- prin **deplasarea ordonatei** (departarea cu 1 de abscisa)

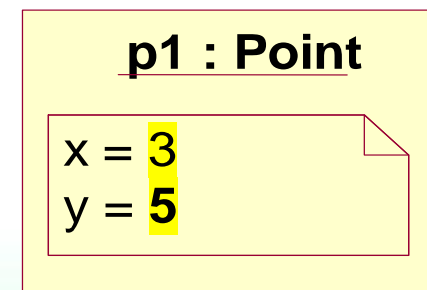
```
// crearea dinamica si initializarea obiectului  
Point p1 = new Point(3, 4);
```

```
// schimbarea starii obiectului  
p1.moveTo(3, 5);
```



dupa operatia
moveTo (3, 5)

-----➔



Obiecte si clase

- Metodele Java (operatiile) si comportamentul obiectului

Operatiile (functiile membru, metodele Java)

Metoda Java (**operatia** in teoria OO), atunci cand este executata:

- realizeaza o **secventa de actiuni (instructiuni)**
 - **asupra** valorilor **campurilor obiectului** caruia ii apartine
 - putand **folosi** valorile acelor campuri, si astfel
- **efectueaza o sarcina** pentru **codul (context) care a apelat-o**

Metoda

- este un **atom de comportament** al obiectului

Comportamentul global al obiectului este obtinut prin

- **inlantuirea apelurilor de metode**

Toate obiectele din aceeasi clasa

- au **aceleasi metode** disponibile

Operatiile (functiile membru, metodele Java)

Dupa invocare (apelare) metodele obiectelor

- **efectueaza sarcini**
 - **utilizand** valorile eventualelor **argumente**
 - **eventual utilizand** valorile **campurilor (atributelor)** obiectului
- sarcini care se pot incheia prin **returnarea unei valori**

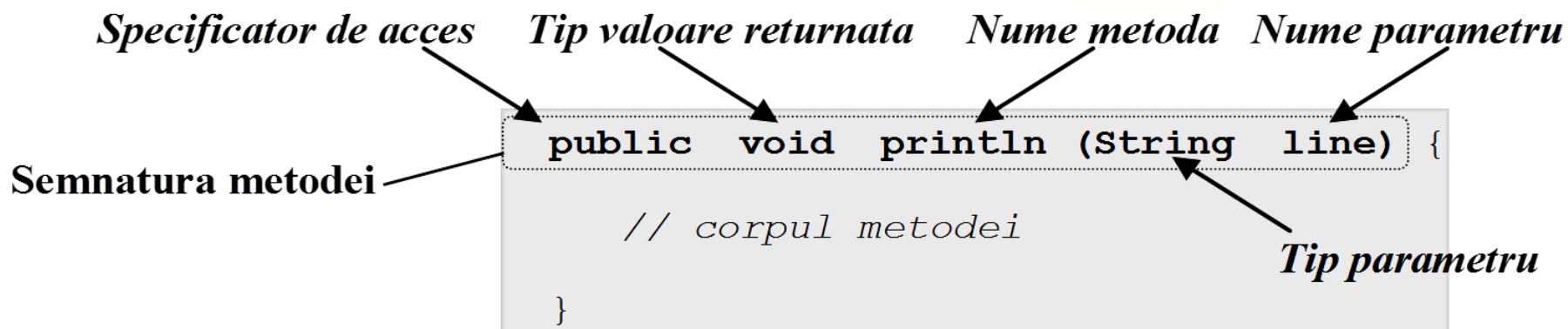
Efectele actiunilor metodei asupra valorilor campurilor (atributelor) obiectului pot fi **combinatii** intre:

- **modificarea valorilor campurilor** obiectului
 - ca in cazul metodelor de tip **setCamp()**
- **obtinerea valorilor campurilor**
 - ca in cazul metodelor de tip **getCamp()**
- **realizarea altor sarcini (“colaterale”)** utilizand aceste valori

Operatiile (functiile membru, metodele Java)

Definitia unei metode contine 2 parti:

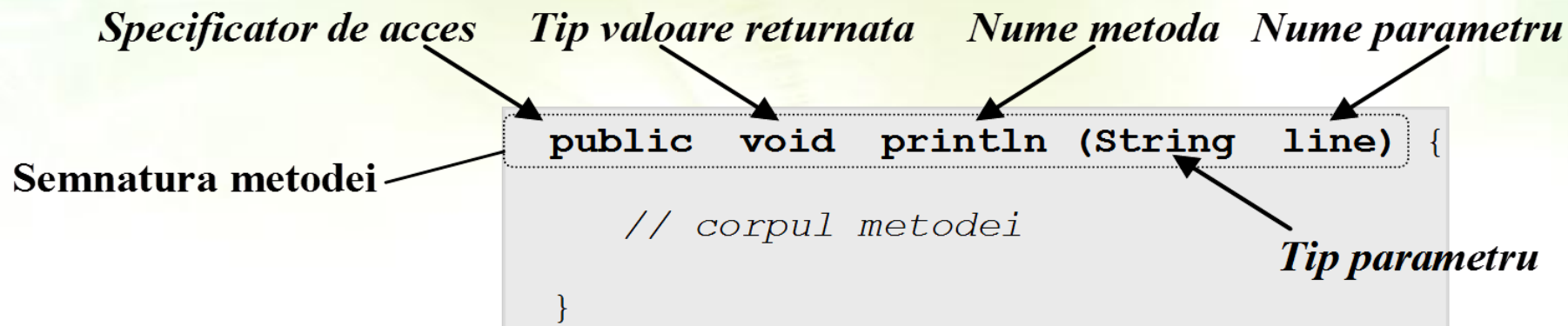
- **semnatura (antetul, declaratia)** si
- **corpul (blocul, segmentul, secventa** de instructiuni a implementarii)



Obiecte si clase

- Semnaturile metodelor

Operatiile (functiile membru, metodele Java)



Semnatura metodei specifica:

- **numele** metodei
- **lista de parametri formali** (numarul, ordinea, tipul si numele lor)
- **tipul valorii returnate** (daca nu returneaza nici o valoare, tipul valorii returnate este declarat **void**)
- **specificatori** ai unor **proprietati explicite** (**modificatori** ai proprietatilor **implicite**)

Operatiile (functiile membru, metodele Java)

Tipul valorii returnate poate fi

- **unul dintre** cele **8 tipuri primitive** Java (byte, short, int, long, float, double, boolean si char), sau
- **unul dintre** cele **3 tipuri referinta** (tablourile, clasele si interfetele Java).

Corpul metodei

- contine **secventa de instructiuni** care
- **specifica pasii** necesari **indeplinirii sarcinilor** (evaluari expresii, atribuire, decizii, iteratii, apeluri metode)

Returnarea valorilor

- este specificata in codul metodelor prin instructiunea **return** urmata de o **expresie**
- care poate fi **evaluata** la o valoare de **tipul declarat in semnatura**

2.1. Obiecte si clase. Metode (operatii) si campuri (attribute)

Operatiile (functiile membru, metodele Java)

Declaratiile (semnaturile) metodelor

- pot include **liste de declaratii de parametri**

Parametrii

- specifica **valorile de intrare** necesare metodelor pentru a fi executate
- sunt **variabile** care au **ca scop intregul corp** al metodei
- declarate ca orice variabila, folosind formatul **tipVariabila numeVar**
- se numesc **parametri formali** sau simplu **parametri**

Apelurile metodelor

- pot include **liste de valori date parametrilor**
- valori care **trebuie sa corespunda ca tip** celor declarate

Valorile pasate metodelor in momentul apelurilor

- se numesc **parametri actuali** sau simplu **argumente**

Obiecte si clase

- Metodele ca mesaje, colaborarea intre obiecte

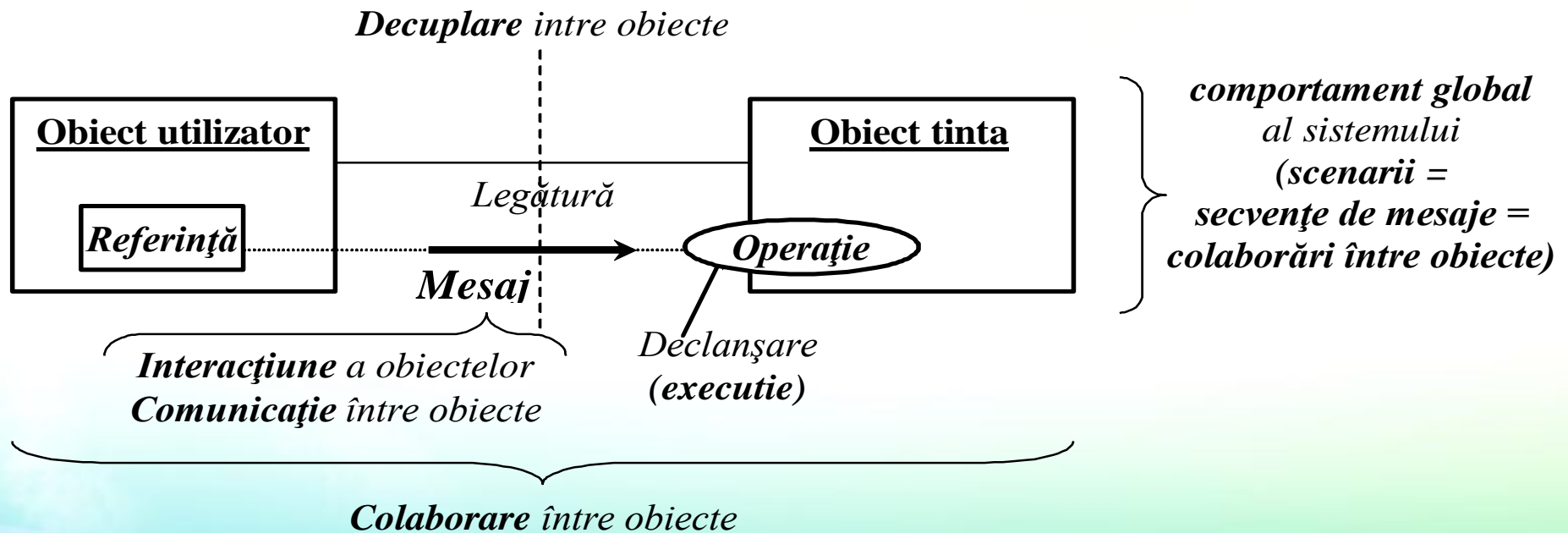
Operatiile vazute ca mesaje schimbate intre obiecte

Obiectele comunica prin mesaje

- mesajul fiind **apelul (invocarea) metodei** altui obiect

Mesajul

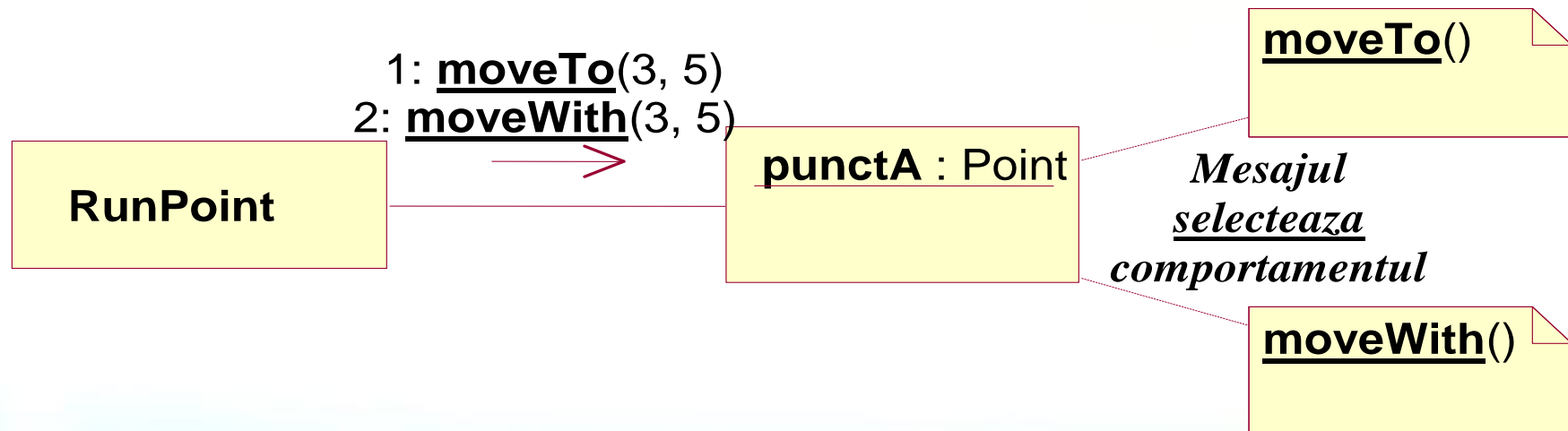
- realizeaza **cuplajul dinamic**
- **intre obiecte** (care sunt **create** la randul lor **dinamic**)



Operatiile vazute ca mesaje schimbate intre obiecte

Mesajul

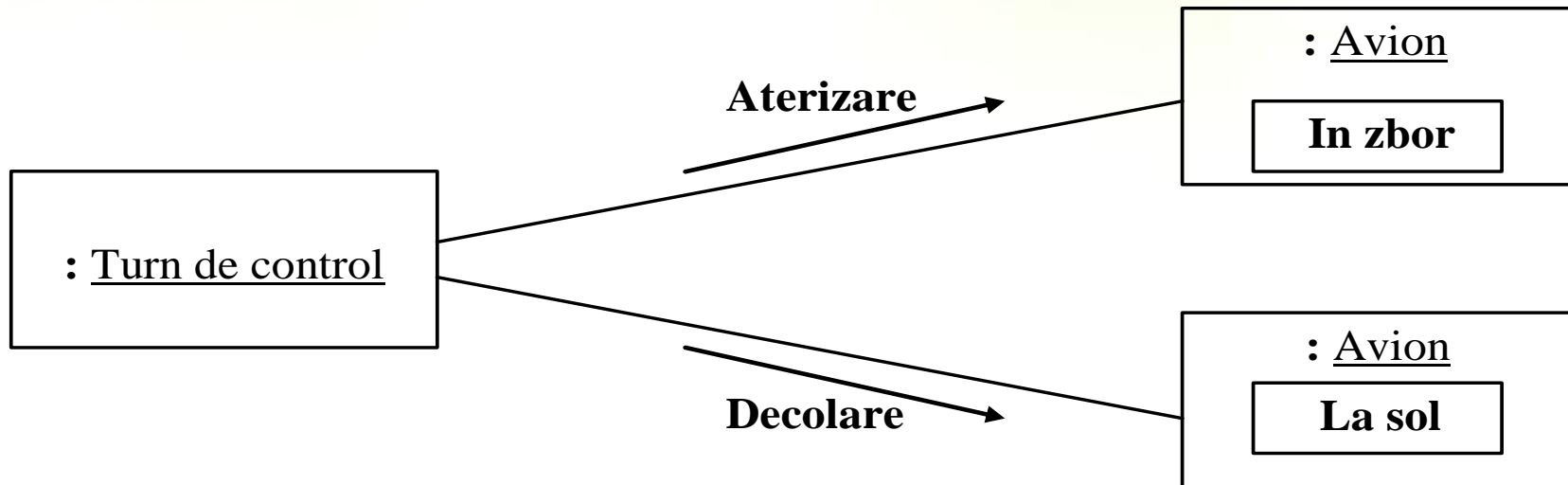
- **selecteaza** operatia si
- **declanseaza** comportamentul (**activeaza** operatia)
 - prin **invocarea / apelul** metodei / functiei membru



Operatiile vazute ca mesaje schimbate intre obiecte

Starea (ansamblul valorilor atributelor) si **comportamentul** (efectul efectuarii operatiilor) **unui obiect** sunt **dependente**

- **comportamentul** la un moment dat **depinde de** starea curenta
- starea poate fi **modificata prin** comportament



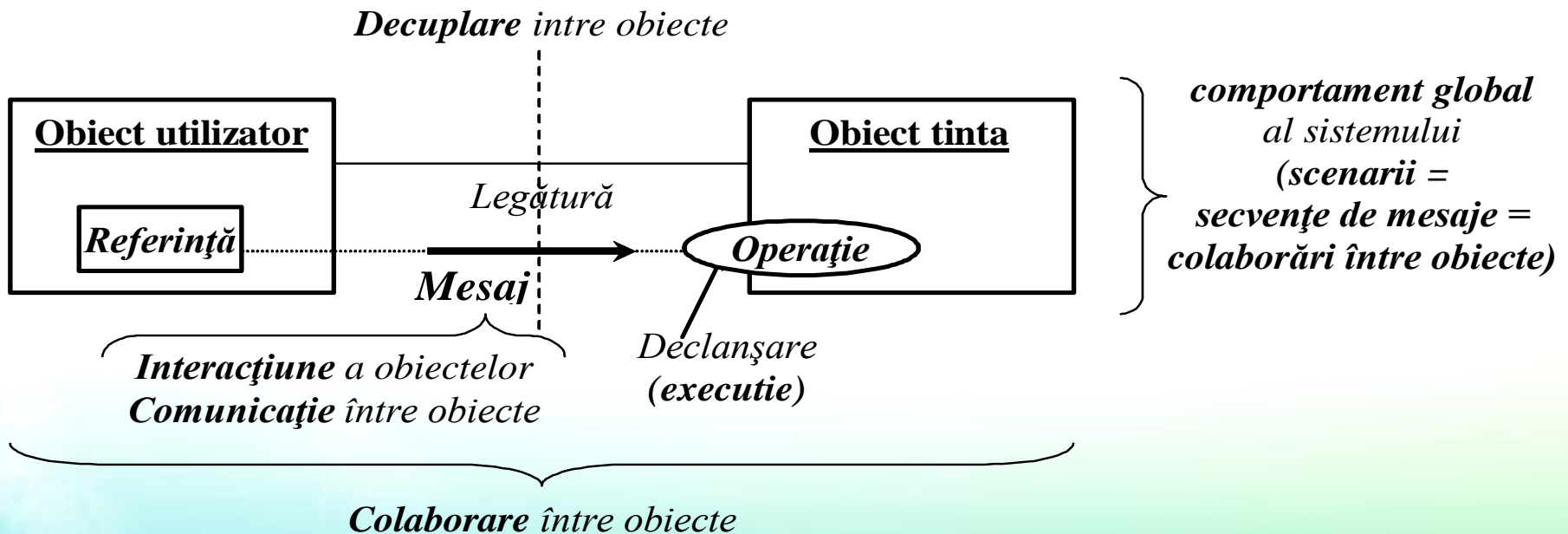
- In starea “**In zbor**” doar comportamentul **Aterizare** e posibil
- El duce la schimbarea starii (in “**La sol**”)
- Dupa aterizare, starea fiind “**La sol**”, operatia **Aterizare** nu mai are sens

Colaborarea între obiecte – efectul schimbului de mesaje

Sistemele software orientate spre obiecte (OO)

- sunt **societăți** de obiecte
- care **colaborează** pentru a realiza funcțiile aplicației

Abordările actuale (orientate spre obiecte - **OO**, bazata pe componente – **CBD**, orientata spre servicii - **SOA**) **imita modelele sociale, colaborative**



Legaturile între obiecte

Între un obiect **utilizator** (din clasa **U**) și un obiect **tinta** (din clasa **S**)

- se realizează o **legatură dinamică**
 - printr-o **referință (r)** către obiectul **tinta** deținută de obiectul **utilizator**
 - și **apelul** unei **metode** a obiectului **tinta** (*a.k.a.* **trimitere de mesaj**)

