PROGRAMARE AVANSATĂ PE OBIECTE

Conf.univ.dr. Ana Cristina DĂSCĂLESCU

Temtică curs 3

- Extinderea claselor
- Clasa Object
- Polimorfism
- Clase abstracte

EXTINDEREA CLASELOR

- Numita și **derivare**, **moștenirea** este un mecanism prin care se poate defini o clasă care extinde o alta clasă deja existentă, **preluând funcționalitățile sale și adăugând altele noi.**
- **Terminologie:** Clasa care se extinde se numește **superclasă**, iar cea care preia datele și funcționalitățile se numește **subclasă**.

- **Sintaxa:**

```
class Subclasa extends Superclasa
    {
      date și metode membre noi;
    }
```

- **Observații**

✓ În limbajul Java, moștenirea este întotdeauna publică și singulară!!!!

✓ Moștenirea definește o relație între superclasă și subclasa sa de tip **IS_A** și conduce definirea unei ierarhii de clase care are rădăcina în clasa **Object.**

- **Ce se moștenește?**

✓Subclasă moștenește toți membrii publici, protejați și impliciți din superclasă.

✓Membrii privați nu sunt moșteniți, dar pot fi accesați prin metode publice sau protejate din superclasă.

✓Metodele constructor, nu se moștenesc, dar un constructor din subclasă poate apela constructorii din superclasa, folosind expresia **`super([argumente]).`**

Observații

- ✓ La instanțierea unui obiect de tip subclasă se apelează constructorii din ambele clase, mai întâi cel din superclasă și apoi cel din subclasă!

- ✓ Apelul constructorului superclasei, dacă există, trebuie să fie **prima instrucțiune din constructorul subclasei** (un obiect al subclasei este mai întâi de tipul superclasei).

- ✓ Dacă nu se introduce în subclasă apelul explicit al unui constructor al superclasei, atunci compilatorul va încerca să apeleze constructorul fără argumente al superclasei.

- ✓ Cuvântul cheie **super** poate fi folosit și pentru a accesa date membre și metode din superclasă, astfel: **super.metoda(lista arg)** sau super.dată_membră.

Mecanismul de redefinire a unei metode (overriding)

- ✓ Mecanismul prin care o subclasă redefinește o metoda moștenită schimbându-i comportamentul.
- ✓ La executare, în raport cu tipul obiectului se va invoca metoda corespunzătoare.

- **Observații:**

- ✓ O metodă din subclasă care redefinește o metodă din superclasă trebuie să păstreze lista inițială a parametrilor formali.

- ✓ Nu se pot redefini metodele de tip **final**.

- ✓ O metodă din superclasă poate fi redefinită în subclasă, dar poate fi totuși accesată prin `super.metodă([parametrii]).`

EXTINDEREA CLASELOR

- ✓Pentru o metodă redefinită se poate schimba modificatorul de acces, dar fără ca nivelul de acces să scadă.

- ✓Tipul returnat de o metodă din subclasă care redefinește o metodă din superclasă trebuie să fie unul covariant tipului de date inițial, respectând-se astfel **principiul de covarianță** (principiul de substituție Liskov):

```
void ↔ void
tip de date primitiv ↔ același tip de date primitiv
referință de tip superclasă ↔ referință de tip superclasă sau de tip subclasă
```

EXTINDEREA CLASELOR

SUPRAÎNCĂRCARE (OVERLOADING)	REDEFINIRE (OVERRIDING)
• **se poate realiza și doar în cadrul unei singure clase**	• se poate realiza doar într-o subclasă a unei superclase
• **la compilare (legare statică)**	• la rulare (legare dinamică)
• **mai rapidă**	• mai lentă
• **metodele de tip final sau private pot fi supraîncărcate**	• metodele de tip final nu pot fi rescrise
• **nivelul de acces nu contează**	• nivelul de acces nu trebuie să fie mai restrictiv decât cel al metodei din superclasă
• **tipul de date returnat nu contează**	• tipul de date returnat trebuie să respecte principiul de covarianță
• **lista parametrilor formali trebuie să fie diferită**	• lista parametrilor formali trebuie să fie identică

Clasa Object

✓Este definită în pachetul `java.lang` și implementează un comportament comun pentru orice obiect Java.

- **`public final Class getClass()`**
- Este o metodă de tip `final` care returnează un obiect de tip `Class` ce conține detalii despre clasa instanțiată în momentul executării programului.

- Clasa `Class` este definita în `java.lang`, nu are constructor public, astfel încât obiectul este construit implicit de către mașina virtuală Java cu ajutorul unor metode de tip *factory*.

- **public String toString()**

  - Metoda returnează o reprezentare a obiectului sub forma unui obiect de tip String.

  - De regulă, se construiește un șir de caractere care conține valorile câmpurilor.

  - Implicit metoda toString afișează un șir format astfel:

getClass().getName() + '@' + Integer.toHexString(hashCode())

Clasa Object

- **boolean equals(Object obj)**
- În limbajul Java, două obiecte se pot compara în două moduri, folosind:

✓ operatorul **==** care verifică dacă două obiecte sunt egale din punct de vedere al referințelor

```
Persoana p1 = new Persoana("Matei", 23);
Persoana p2 = p1;
System.out.println(p1==p2); //se va afișa true
Persoana p3 = new Persona("Matei", 23);
System.out.println (p1==p3); //se va afișa false
```

✓ metoda **boolean equals()** care, implicit, verifică dacă două obiecte au aceeași referință

```
System.out.println (p1.equals(p3));//se va afișa false
```

`public int hashCode()`

- Codul hash al unui obiect este un număr întreg care este dependent de conținutul său.
- Implicit, codul hash este calculat de către mașina virtuală Java, utilizând un algoritm specific care nu utilizează valorile câmpurilor obiectului respectiv
- Clasa `Objects` conține o metodă cu număr variabil de argumente care calculează codul hash pentru un anumit obiect folosind valorile câmpurilor sale: `Objects.hash(câmp_1, câmp_2,…)`.

- Un obiect din Java poate fi referit prin tipul său sau prin tipul unei superclase

- Implicit se poate realiza conversia unei subclase la o superclasă, mecanism care poartă denumirea de *upcasting*

- Considerăm clasa B extinsă de clasa A. Pot avea loc următoarele instanțieri:

```
B b = new B();
```
//referirea obiectului printr-o referință de tipul clasei

```
A a = new B(...);
```
//referirea obiectului printr-o referință de tipul superclasei

- Obiectul `a` are tipul declarat A și tipul real B!

- *Downcasting*-ul reprezintă accesarea unui obiect de tipul superclasei folosind o referință de tipul unei subclase, necesitând o conversie explicită!

✓**Exemplu:** Considerăm clasa Angajat și două subclase ale sale, Economist și Inginer.

```
Angajat a = new Economist();
Angajat b = new Inginer();
```

Corect, deoarece upcasting-ul se realizează implicit

```
Inginer p = b;
```

eroare la compilare,

```
Inginer p = (Inginer)b;
```

eroare la executare, ClassCastException

Polimorfism

Superclasă	Subclasă

class A
{
int dată_membră = 3;
 	
void metoda1() {
System.out.println("Metoda 1 din clasa A!");
}
 	
static void metoda2(){
System.out.println("Metoda statică 2 din clasa A!");
}
}
	class B extends A
	{
	int dată_membră = 4;
	
	void metoda1() {
	System.out.println("Metoda 1 din clasa B!");
	}
	
	static void metoda2(){
	System.out.println("Metoda statică 2 din clasa B!");
	}
	}


```
A ob = new B();//polimorfism

System.out.println("Datamembră = " + ob.dată_membră);
ob.metoda1();
ob.metoda2();
```

✓ **Output**

```
Data membră = 3 -> din superclasa A
Metoda 1 din subclasa clasa B!
Metoda statică 2 din superclasa A!
```

➢ Observați

- Câmpurile se accesează după tipul declarat, ci nu după tipul real!

- O metodă de instanță este apelată dupa tipul real , respectiv dacă este de tip subclasă, atunci se apelează metoda redefinită!

- Metodele statice nu se redefinesc, deci selecția se realizează după tipul declarat.

➢ Concluzie

- O metodă de instanță este invocată în raport de tipul real, tip care se identifică la executare (*runtime*). Conceptul se mai numește și ***legare dinamică*** **sau** ***legare târzie*** **(late binding).**

➢ Exemplu

```
Scanner sc = new Scanner(System.in);
double d = sc.nextDouble();

A ob;//de tipul superclasa

if (d>0) Ob = new A(...);//de tipul concret ->tipul subclasa
 else ob = new B(...);

ob.met(...);                    //se apelează după tipul real
identificat la executare
```

Clasă abstractă

- Nu are un comportament complet definit
- Se află în vârful ierahiei pentru a modela un comportament comun tuturor claselor din ierahie

```
public abstract class IdClasa {
......
 abstract public tip metodaAbstracta();
 }
```

➢ Observați

- O clasă abstractă nu se poate instanția, deoarece nu se cunoaște integral funcționalitatea sa.

- Dacă o subclasă a unei clase abstracte nu oferă implementări pentru toate metodele abstracte moștenite, atunci subclasa este, de asemenea, abstractă, deci nu poate fi instanțiată!!!

- O clasă abstractă poate să conțină date membre de instanță, constructori și metode publice, astfel încât subclasele sale pot apela constructorul din superclasă, respectiv pot redefini membrii săi.

➢ Exemplu

```
public abstract class Angajat {
double salariu_baza;
......
abstract public double calculSalariu();
}

class Paznic extends Angajat{
        ......
        static double spor_de_noapte = 0.25;
        ......
        public double calculSalariu() {
        returrn salariu_baza + salariu_baza * spor_de_noapte;
}
```