

MOȘTENIREA

a) de clasă / implementare

b) de tip / interfață

→ extends → ambele lucruri

(A)

DE CLASĂ

→ Specificatori de acces

- private → vizibil doar în interiorul clasei

+ public → oriunde

protected → în interiorul clasei

→ în subclasele sale

→ în același pachet (orice subpachet)

Constructor

→ prima inst. dată - un constructor →

→ alt apel la alt const.

→ const. superclass

Ordinea inițializării

- (1) Alocare memorie
- (2) Setare pe 0
- (3) Se apelează constructorul (superclasa → subclasa)
- (4) iniț. de la var. de stare
- (5) se execută corpul constructorului

Nu există moștenire a 2 clase simultan

→ implicit extinde clasa Object

Redefinirea → overriding

→ implementare nouă pentru fct. moștenită

→ tipul returnat poate fi modificat

→ specificații pot fi modificate;

(cel puțin la fel de multă vizibilitate ca și met. moștenite)

Metoda final \rightarrow nu poate fi supra scris

Clasa final \rightarrow nu poate fi moștenită

Overloading (semnături diferite)

Overriding (aceleași semnături, implementare diferită)

Hiding

\rightarrow variabile instantate în Subcl. cu același nume cu Supercl.

Point

int x

MyPoint extends Point

double x

super.x
this.x

Main

MyPoint ex = new

ex.x

MyPoint.x

/ ((Point)ex).x

Point.x

versus metode statice ✗
HIDING met. statice ✓

Compunerea de Obiecte

- favorabil în locul moștenirii
 - implementare a clasei pe baza "feature-urilor" altei clase, fără a moșteni toate atributele/metodele
-

Compunerea vs. Moștenire

Comp.	→ relație	HAS - A
Moșt.	→ relație	IS - A

IS - A → se compoartă ca și superclasa

Noțiunea de tip

tip de date (abstract) definește
o mulțime de valori, caracterizată de
operațiile disponibile peste ele
(fără detalii de implementare)

Tipuri vs. Clase

tip \rightarrow interfața obiectului
 \rightarrow set de operații pot să fac

tip + implementare = clasă

extends \rightarrow ambele $\begin{matrix} \text{tip} \\ \text{implementare} \end{matrix}$