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 obiect)

Constructori

-> prima instr. dintr-un constructor ->
-> alt apel la alt constr.
-> constr. superclasei

Ordinea inițializărilor

(1) Alocare memorie
(2) Setare pe 0
(3) Se apelează constructorul (superclasă → subclasă)
(4) init. 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
-> specificatorii pot fi modificați
(cel puțin la fel de multă vizibilitate ca și met. moștenită)

Metodă **final** → nu poate fi suprascrisă

Clasă **final** → nu poate fi moștenită

overloading (semnături diferite)

overriding (aceleași semnături, implementare diferită)

Hiding → variabile instanță în Sub cl. cu același nume cu Super cl.

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

Override metode statice ✗

HIDING met. statice ✓

Compunerea de Obiecte

-> favorabil în locul moștenirii

-> implementare a clasei pe baza "features-uri" altei clase, fără a moșteni toate atributele/metodele

---

Compunerea vs. Moștenire

Comp. -> relație HAS-A

Mošt. -> relație IS-A

IS-A -> se comportă 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 -> interfața obiectului
-> set de operații pot să fac

tip + implementare = clasă

extends -> ambele <tip, implementare>