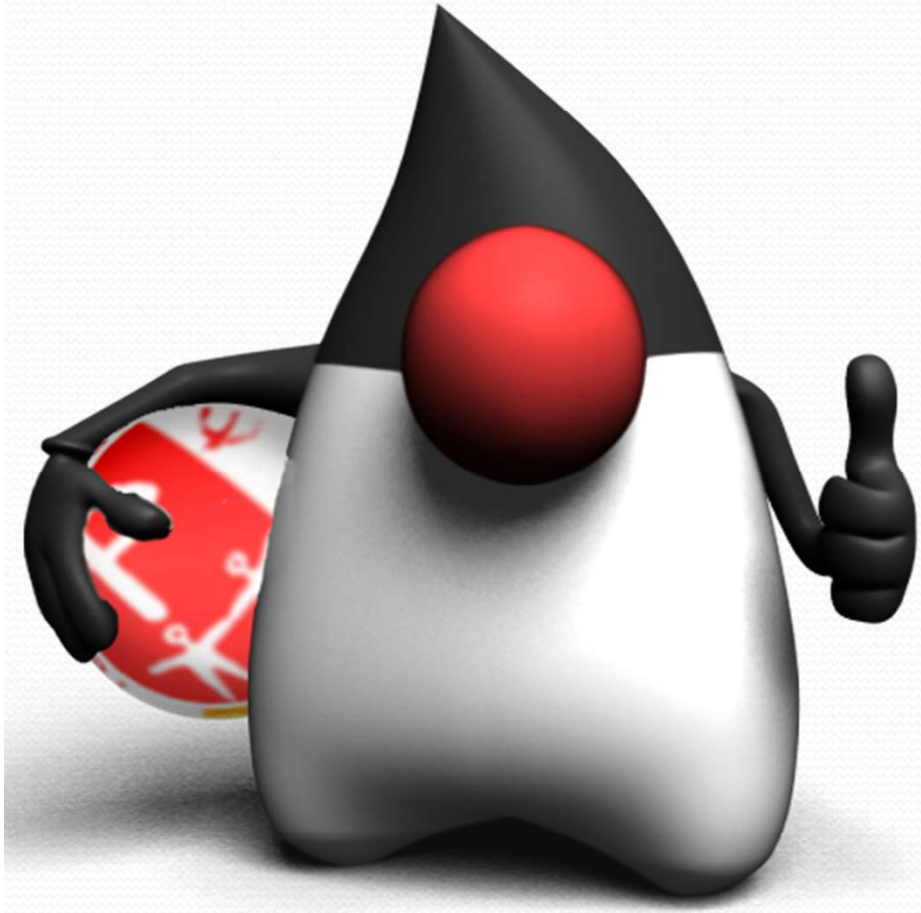

Java

Herència i polimorfisme

Herència

Ens podem trobar que tinguem una classe dissenyada i ens adonem que en necessitem una altre de molt semblant, igual que la que tenim amb uns quants mètodes o atributs més, o amb alguns mètodes que necessiten tenir un comportament diferent del que ja tenim.

Podem crear aquesta classe nova des del principi, però, a part de la feina que comporta, tindrem dues classes totalment deslligades i que fàcilment poden perdre aquesta semblança que haurien de tenir.

Una altra opció és utilitzar el **concepte d'herència: Crear la nova classe a partir de l'existent de forma que qualsevol canvi que es faci a l'original es reflecteixi a la nova.** Només hi haurem d'afegir aquells membres que necessitem i que no té l'original.

Herència

La idea de l'herència és molt simple però molt potent: Si tenim una classe A semblant a una altra B podem crear A a partir de B de forma que tingui tots els membres que té B.

- Superclasse: la classe de la que es deriven altres classes.
- Subclasse, classe derivada o classe filla: classe creada a partir d'una altra.

Una classe pot tenir *n* subclasses, **però una subclasse només pot tenir una superclasse.** En Java no existeix l'herència múltiple.

Una classe pot tenir una superclasse, que pot tenir una superclasse, que pot ...

Totes les classes en Java descendeixen d'Object.

Per tant, les classes formen una jerarquia en forma d'arbre, on l'arrel és Object.

Superclasse

Qualsevol classe pot tenir subclasses, **no hem de fer res especial per marcar-la com a tal.**

Superclasse només és un concepte, no implica cap comanda ni res especial. Si tenim una classe A, en crear una subclasse seva B direm que A és la superclasse de B, però el codi d'A seguirà sent el mateix, no l'hem de modificar per res.

Podem dir que hi ha dues maneres de trobar una relació d'herència entre classes quan dissenyem una aplicació:

- **Per especialització**: Tenim una classe i veiem que en necessitem definir casos especialitzats d'aquesta classe. Per exemple tenim la classe Vehicle i ens adonem que necessitem tractar de forma separada camions i autocars, però volem aprofitar el codi de Vehicle i que es mantingui una relació entre ells.
- **Per generalització**: A la nostra aplicació tenim una classe Camió i una altre Autocar i ens adonem que comparteixen gran part del codi i que a més tenim processos que han de tractar-los indistintament. Ens podem plantejar crear la classe Vehicle.

Subclasse

Per crear una subclasse ho feim de la següent manera:

```
public class Camio extends Vehicle{
```

La paraula clau **extends** indica que és una subclasse i ***Vehicle*** ens diu quina és la seva superclasse.

A més dels modificadors d'accés vistos (públics i privats), tenim un altre modificador d'accés intermedi (a nivell de restricció): **protected**. Si declarem un atributo como protected, **aquest significa que les subclasses sí poden tenir accés al camp o al mètode.**

Una subclasse hereta tots els membres ***public*** **i** ***protected*** de la superclasse i si es troben al mateix paquet, els que no tenen qualificador d'accés.

- Els atributs heretats es poden utilitzar directament.
- Podem declarar atributs nous o amb el mateix nom que un de la superclasse (ocultar-lo).

Subclasse

- Els mètodes heretats es poden utilitzar directament.
- Podem escriure nous mètodes:
  - Totalment nous.
  - Amb la mateixa signatura que un d'instància de la superclasse: sobreescriptura.
  - Amb la mateixa signatura que un estàtic de la superclasse: ocultació.
  - Nous constructors. Criden al de la superclasse implícitament o explícita (super).

Els membres privats de la superclasse no són heretats ni accessibles directament, a no ser a través d'altres membres accessibles.

Els constructors tampoc s'hereten. El constructor de la subclasse ha de cridar un constructor de la superclasse. Si no ho fem explícitament amb *super,* el compilador intentarà cridar el constructor sense arguments. Si la superclasse no el té donarà error.

Sobreescriptura i ocultació

Sobreescriptura: un mètode d'una subclasse en sobreescriu un de la seva superclasse si té la mateixa signatura i tipus de retorn. Encara que tractem l'objecte de la subclasse com un de la superclasse, s'executarà el de la subclasse.

Ocultació:

- Si a una subclasse declarem un mètode estàtic amb la mateixa signatura que un mètode estàtic de la superclasse, l'oculta. Si convertim l'objecte de la subclasse a la superclasse fent un càsting, llavors s'executarà el mètode de la superclasse.

- Si a una subclasse declarem un atribut amb el mateix nom que un de la superclasse l'oculta, és a dir, a la subclasse només podreu utilitzar el de la subclasse, només tindrem accés al de la superclasse utilitzant super. Si fem un càsting d'un objecte de la classe derivada cap a la superclasse, llavors podrem accedir a l'atribut de la superclasse i no al de la subclasse.

Polimorfisme

Un objecte és de la classe que s'ha instanciat, però també és un objecte de totes les seves superclasses i es pot utilitzar per tot on s'esperi un objecte d'aquestes classes.

Per exemple, tenim Objecte -> Fruita -> Poma. A qualsevol lloc on s'esperi un Object o una Fruita hi podem passar un objecte Poma.

Polimorfisme: La capacitat d'un objecte per comportar-se de formes diferents.

Un objecte d'una subclasse pot ser considerat com un objecte de qualsevol de les seves superclasses fent un càsting implícit o explícit. Llavors no podrem executar cap mètode ni utilitzar cap atribut que hagi estat afegit a la subclasse.

Per altre banda, si tenim dos objectes d'una superclasse tampoc tenen perquè comportar-se de la mateixa manera. Si un s'ha creat com a objecte de la superclasse i l'altre com un objecte de la subclasse, en executar els mètodes sobreescrits tindrem comportaments diferents.

super, getClass() i instanceof

super

Permet invocar un mètode sobreescrit o un camp ocult de la classe pare. S'utilitza quan el mètode de la subclasse ha de realitzar les mateixes accions que les del mètode que sobreescriu i unes quantes més. S'utilitza molt en els constructors, on la crida al super ha de ser la primera instrucció.

```
super.mètode();
x=super.atribut;
```

```
public NomClasse(...){
  super(...);
```

getClass()

És un mètode de la classe Object, per tant el tenen tots els objectes que podem crear. Ens diu de quina classe és realment l'objecte, amb quina classe feren el *new*, encara que s'hagin fet càstings.

```
System.out.println(fruita.getClass());
```

instanceof

És un operador del llenguatge. Permet comprovar si un objecte és una instància d'una determinada classe. Un objecte serà instància de la classe a partir de la qual va ser creat i de totes les seves superclasses.

```
if (fruita instanceof Poma) System.out.println("És una poma!");
```

Final i abstract

final

Un mètode es pot declarar *final* de forma que les subclasses no el puguin sobreescriure.

Una classe es pot declara *final* de forma que no es poden crear subclasses a partir d'ella.

abstract

Dins una classe podem declarar un mètode sense implementar-lo, de forma que ho facin les subclasses d'aquesta classe. **Si definim un mètode abstracte, la classe sencera ho serà. No es poden crear objectes d'una classe abstracta.** Les subclasses han d'implementar tots els mètodes abstractes de la superclasse o convertir-se elles mateixes en abstractes.

```
public abstract class Meva{

private int num;

public abstract void mostra();

....
```