Composició i Herència

JAVA

Reutilitzant el codi

- Els progamadors sempre han reutilitzat seccions de codi (plantilles, funcions, procediments…).
- Volem estalviar temps i eliminar redundàncies, emprant recursos que altres programadors ja han implementat.
- La forma més senzilla de reutilitzar codi és copiar-lo i canviar-lo (adaptant-lo a les nostres necessitats).
- Estudiarem com podem enfocar aquest problema en JAVA
  - Composició: Creem objectes de classes existents dins la nova classe.
  - Herència: Formem un nou sub-tipus (nova classe) d'una classe ja existent.

Composició. Sintaxi.

- La idea és simple: la nova classe tindrà referències (com a variables membre) a objectes d'altres classes.
- Veure exemple (Job - Person).
- Les referències a objectes per defecte són "null". Tenim diverses opcions:
  - Inicialitzar les referències on els objectes són definits.
  - Inicialitzar dins el constructor.
  - Inicialització d'instància (bloc de codi anònim dins la classe)

Herència

- El concepte d'herència és integral dins JAVA (ho podem extendre a tots els llenguatges Orientats a Objecte).
- Distintes classes d'objectes tenen moltes vegades certs atributs en comú. Per exemple: els objectes cotxes, camions i tractors tenen les característiques de vehicles de quatre rodes amb un volant.
- En programació Orientada a Objectes, podem crear classes que "hereden" els atributs i el comportament d'altres classes.
- Les classes més "generals" es diuen super-classes.

Herència

- Dins l'entorn Java, una classe només pot tenir una sola super-classe, però cada super-classe pot tenir moltes sub-classes.
- En JAVA, sempre fem ús de l'herència quan creem una nova classe, perquè tots els objectes hereden de la classe stàndard "Object".
- La sintaxi en Java per definir que una classe és sub-classe d'una altra classe, és mitjançant el keyword "extends" (Veure exemple).

Herència

- Una sub-classe “hereda” els mètodes i els atributs de la super-classe. Per exemple, “cotxe” heredaria de “vehicle” l’atribut “motor” i el procediment “start”.
- Una sub-classe no podrà accedir als mètodes “private” de la super-classe, però sí als públics i els “protected”. Dins el mateix paquet, també podrà accedir als membres sense modificador d’accés.
- Una sub-classe pot modificar un mètode de la super-classe. Si també vol cridar al codi de la super-classe ho pot fer mitjançant el keyword **super**.

Herència

- Evidentment, la sub-classe pot definir nous mètodes, no està limitada a modificar només els mètodes de la super-classe.
- Si el constructor de la super-classe accepta arguments, dins el constructor de la sub-classe haurem de cridar esplícitament al constructor superior.
  - Això es fa amb la següent sintaxi: super(args_del_constructor_superior)
  - Si la super-classe té un constructor sense arguments, Java insertarà una cridada al constructor superior automàticament dins el constructor de la sub-classe.

Combinant Herència i Composició

- Moltes vegades és útil emprar l'herència i la composició a la vegada. Veure exemple.
- Per emprar aquestes tècniques no fa falta tenir el codi font de les classes emprades.
- Però en tot cas, si no disposem del codi font, recordem que hem d'importar el paquet on les classes emprades estan definides.
- Recordem que podem emprar el modificador "private" pels atributs i els mètodes que no s'han d'emprar fora de la classe.

Ocultació de noms en herència

- En Java, si una classe base té un mètode que és sobreescrit vàries vegades, redefinir el mètode a una classe derivada no amaga cap de les versions de la classe base.
- En altres llenguatges Orientats a Objectes com C++, els mètodes de la classe derivada amaguen els de la classe base.
- En Java SE5 es va afegir la notació “@Override”, per fer que el compilador ens avisi quan realment volem fer un “override” i no un “overload”.
- Veure exemple...

Composició vs. Herència

- Tant la tècnica de composició com la tècnica d'herència permeten introduir nous objectes dins una nova classe.
- Quan hem de triar un o l'altre?
- La composició s'empra generalment quan volem la funcionalitat d'una classe existent dins la nova classe. L'usuari de la nova classe no té el perquè saber que a dins hi ha una altra classe.
  - Podem escollir fer "private" la classe composada, o no.
- Quan emprem herència, agafem una classe existent i fem una versió especial d'ella. Alerta amb les relacions "és-un" (herència) o "té-un" (composició).

El keyword “protected”

- En un mon ideal, el modificador “private” seria suficient, però en projectes reals, hi ha vegades que volem fer un mètode o atribut privat per tot el mon, però que les sub-classes hi tinguin accés.
- Això precisament és el que aconseguim amb **protected**
- Generalment, només es fan “protected” alguns mètodes.

Upcasting / Downcasting

- Si A és una super-classe i B és una sub-classe de A:
  - B té el tipus A
- Tots els mètodes públics de A funcionaran en B.
- Convertir un objecte de tipus B en un objecte de tipus A es diu “Upcasting”
  - Aquesta operació sempre es pot realitzar i és “segura”.
- Convertir un objecte de tipus A en un objecte de tipus B es diu “Downcasting”
  - No sempre és possible, i es presenten una sèrie de dificultats que s’examinaran més endavant.

El keyword **final**

- En general, “final” és un modificador que expressa que “això no es pot canviar”. És semblant a la noció de constant en altres llenguatges.
- Si es tracta d’un atribut:
  - Constant de “temps de compilació”, mai canviarà.
  - Valor inicialitzat al moment de l’execució que no volem que pugui canviar.
- Les primitives “final” mantenen inalterable el seu valor.
- Les referències “final” a altres objectes no mantenen inalterable l’objecte al qual apunten. Només mantenen constant la referència (veure exemple).

El keyword **final**

- Blank final: atributs que són declarats “final” però sense valor inicial. Aquests atributs s’han d’inicialitzar dins el constructor, i a partir d’aleshores no podran ésser canviats.
- Arguments final: A un mètode, si declarem els seus arguments “final”, aquests no podran ésser canviats dins el cos de la funció.
- Si declarem un mètode “final” a una super-classe, una sub-classe no podrà fer un “override” d’aquest mètode.
- Si fem una classe “final”, aleshores no podran existir sub-classes d’aquesta. Estem prohibint explícitament que una classe heredi de la nostra classe.