Polimorfisme i interfaces

JAVA

Introducció

- És una característica que proporciona una nova dimensió en la separació interfície / implementació (el “què” del “com”).
- Permet que varis “tipus” (tots derivats del mateix tipus base) es tractin com si fóssin un sol tipus.
  - Es pot escriure codi que treballi amb tots els tipus esmentats de la mateixa manera.
- Aquesta característica també es coneix amb el nom de “late binding”, “dynamic binding” o “runtime binding”.

Upcasting

- Fer un “upcast” consisteix en convertir una variable de tipus A en una de tipus B (sempre que B és una superclasse de A).
- Veure exemple:
  - Classe Instrument. Té el mètode “play()”
    - Classes derivades: Wind, Stringed, Brass
- Suposem que volem escriure un mètode “void tune(Instrument i) { }”
  - Aquest mètode funcionarà amb qualsevol instrument.
- És innecessari escriure un mètode “tune()” per a cada instrument.
- El mètode “tune” cridarà al mètode “play”, però aquest mètode depèn del tipus final de l’instrument (no és el mateix “play” de Brass, que “play” de Wind.

Binding

- La connexió de la cridada a mètode amb el mètode adeqüat es diu “binding”.
- Quan aquest procés es fa quan es compila el programa, s’anomena “early binding”. Per exemple, en el llenguatge “C” sempre es fa així.
- En “late binding” tenim que el compilador no pot saber quin mètode es cridarà, sinó que només es sap quan s’executa el programa.
- Veure exemple (Shape):
  - Shape s = new Circle();
  - s.draw(); // Cridada polimòrfica
- En aquest darrer exemple podem veure que la funció de “Shape” és la de proveir una interfície comú a totes les figures.

Extensibilitat

- En l’exemple “Instrument”, podem veure que és fàcil extendre el programa simplement afegint nous instruments, heredant de les classes ja existents.
- El codi que ja està preparat per manejar “instruments” funcionarà igual amb les noves classes, fins i tot sense recompilar.
- Exemples:
  - Classes Brass i Woodwind són subclasses de Wind
  - tune(Instrument i)
  - tuneAll(Instrument[] ins)

Pitfalls

- Alerta: els mètodes “private” no es poden fer overrides. Per tant, no funciona el polimorfisme amb aquests (simplement es defineixen nous mètodes a sobre).
- Només els mètodes són polimòrfics. Els atributs (o “camps”, o “variables de classe”) no es comporten de manera polimòrfica. Veure exemple.
- Els mètodes “static” tampoc es resolen en temps d’execució. Si hi ha dos mètodes amb el mateix nom a dues classes (amb herència), el nom del mètode sempre es resol en temps de compilació.

Polimorfisme i els constructors

- Quan es crea un objecte i es crida al constructor:
  - Primer es crida al constructor de la classe base. Aquesta passa es repeteix de manera recursiva de manera que s'executa el constructor de l'arrel primer de tot, seguit del següent dins la jerarquia, i el següent, i així...
  - Els inicialitzadors de les variables membres es criden en l'ordre que s'han declarat
  - Finalment s'executa el codi del constructor de la sub-classe.

Polimorfisme i disseny

- Mètode “ideal” o “pur”: les sub-classes fan un override dels mètodes descrits a la super-classe.
  - Això possibilita escriure codi que funcionarà amb qualsevol tipus derivat
- Però hi ha vegades (la majoria) que descobrim que el millor és que les sub-classes **afegeixin** mètodes nous no definits a la classe base.
  - Això té el problema que si fem un “upcast” perdem l’accés a aquesta nova part de la interfície, perquè no existeix a la classe base.
  - Per tornar un objecte al seu tipus original després d’un “upcast”, es pot fer un “downcast”.
  - Podem produir una “ClassCastException” (error).

Interfaces i classes abstractes

- Són mecanismes que ens permeten una manera millor de separar la interfície de la implementació.
- Classe abstracta:
  - Es tracta d'una classe que no s'empra mai directament, sinó que el seu únic propòsit és expressar una interfície a una família d'objectes (exemple: Instrument).
  - Pensem bé: un objecte "Instrument" pur no existeix. Existeixen flautes, tambors i pianos, però no "instruments purs".
  - En JAVA, podem declarar una classe abstracta amb el keyword "abstract".
  - Una classe abstracta pot tenir mètodes abstractes (es tracta de mètodes sense cos), també declarats amb la keyword "abstract".
  - Una classe "normal" no pot tenir mètodes "abstract", però pot ésser derivada d'una classe abstracta.

Interfaces i classes abstractes

- En JAVA, tenim una segona opció per expressar una classe abstracta: emprar la keyword “interface”.
- Es tracta de produir una classe totalment abstracta, sense atributs i amb tots els seus mètodes abstractes (sense cos).
- És a dir, una “interface” és una plantilla en el sentit més pur, el seu propòsit és que es faci servir per crear noves classes.
- En JAVA, una classe pot “implementar” vàries “interfaces”. D’aquesta manera tenim una espècie d’herència múltiple restringida.

Interfaces

- Una “interface” permet que es pugui aplicar un mètode (troç de codi) a objectes que en principi no formen part de la mateixa jerarquia.
  - Exemple:
    - Class CD extends Disk...
    - Class DVD extends Disk...
    - Class Flash extends SRam…
    - Void read(???) { ... }
  - Solució:
    - Interface Reader...
    - Class CD extends Disk implements Reader
    - Class DVD extends Disk implements Reader
    - Class Flash extends SRam implements Reader
    - Void read(Reader r) { … }