Gerarchia di sottoclassi

Per come è strutturato il linguaggio Java, la sottoclasse può ereditare tutti gli **attributi** e i **metodi** di una classe madre, detta *superclasse*. Ciò può tranquillamente avvenire a patto che una classe **erediti da una e una sola superclasse**, tenendo presente però che può valere il contrario ovvero due classi figlie diverse possono ereditare da un'unica *superclasse*. Questa proprietà è chiamata **ereditarietà semplice**.

Nel caso in cui non si espliciti la classe di appartenenza, tramite il comando " extends " allora di default la classe interessata erediterà dalla classe **Object**, una classe predefinita del linguaggio Java. Il vantaggio di estendere dalla classe Object in una situazione di default è che quest'ultima al suo interno contiene metodi molto utili come:

- public boolean **equals** (Object)
- protected void **finalize** ()
- public String **toString** ()

Overriding

Non è sbagliato vedere il processo dell'ereditarietà come una vera e propria *estensione* della superclasse. A questo proposito è necessario mettere in evidenza potenziali nuovi scenari di ampliamento di una classe, partendo con l'**overriding**.

Tramite l'**overriding** vengono **ridefiniti** determinati **metodi** della superclasse ponendo attenzione al fatto che **non cambia la firma del metodo ridefinito** bensì rimane la stessa. Come nel caso del costruttore, all'interno di un metodo ridefinito è sempre possibile **richiamare il metodo originale della superclasse** con il comando " this.<nomeMetodo>(lista parametri) ". Un primo diretto esempio di questa pratica è l'overriding del metodo *toString* avente come superclasse la classe Object, al fine di adattarlo come meglio possibile alle nuove classi figlie.

Overloading

La pratica dell'**overloading** si pone invece come il contrario dell'overriding: in una classe possono esserci metodi con stesso nome ma è **fondamentale che cambi il numero o il tipo di parametro passati**, si tenga presente del fatto che non è sufficiente cambiare solo il tipo ritornato, deve **cambiare proprio la firma**. Si tenga presente che nel caso in cui si inseriscono due metodi con firma uguale ma tipo di ritorno diverso, nell'istante della chiamata nel main di uno dei due genera un ERRORE A COMPILATION TIME perché il compilatore non è in grado di capire quale metodo in realtà si voglia chiamare (non lo deduce dal numero dei parametri perchè è invariato).

Information Hiding per sottoclassi

	visibilità			
modificatore	classe	package	sottoclasse	mondo
`private`	Y	N	N	N
"package"	Y	Y	N	N
`protected`	Y	Y	Y	N
`public`	Y	Y	Y	Y

Come nel caso di classi singole comunicanti tra di loro a livello di package, esiste anche un criterio di *information hiding* per metodi e attributi tra superclasse e classe figlia in relazione all'ambiente globale:

- **private** : permette l'accesso a tutti i componenti nella classe e basta.
- **package (nessun modificatore)** : implementazione di private, permette l'accesso in aggiunta anche a componenti del package.
- **protected** : implementa package e private e permette la visualizzazione anche alle sottoclassi.
- **public** : rappresenta la massima apertura della disponibilità di informazioni, rendendo metodi e attributi visibili a tutte le componenti del programma, anche al di fuori del package.

Ereditarietà e costruttori

Di base i costruttori non vengono ereditati ma Java mette a disposizione la possibilità di richiamare i costruttori della classe **super** in modo da risparmiare linee di codice superfluo, ciò può avvenire solo se **l'istruzione super è la prima ad essere scritta** all'interno del nuovo costruttore.

Come nel caso delle classi isolate, se una sottoclasse non contiene un costruttore allora il compilatore ne crea uno di default contenente **super()**. Altrimenti se nella sottoclasse è già presente un costruttore ma non è presente un richiamo ad un costruttore della superclasse, automaticamente il compilatore aggiunge **super()** in prima riga.

Tutto ciò può avvenire solo se il costruttore a zero parametri della superclasse esiste, altrimenti si verifica un ERRORE A COMPILATION TIME, caso un cui la superclasse avrebbe solamente un costruttore a più parametri.

Classi e metodi astratti

Nel caso in cui si volesse aumentare ulteriormente l'astrazione del codice, Java mette a disposizione due strumenti in grado di rispondere a queste esigenze: **classi e metodi astratti**. Si definisce un **metodo astratto** un metodo la cui implementazione non è specificata, parallelamente si definisce **classe astratta** una classe che contiene almeno un metodo astratto. Per identificare questi nuovi strumenti, prima della denominazione del metodo o della classe aggiungiamo il costrutto " abstract ". Una restrizione importante è il fatto che **non è possibile creare istanze di una classe astratta** ma è necessario creare una sua sottoclasse che ne implementa i metodi.