Programmazione orientata agli oggetti

IMPORTAZIONE

- Una classe che si trova nello stesso package della classe principale può non essere importata. Per usare delle sue funzioni basterà anteporre il nome della classe stessa

  es. int x = Math.sqrt(27);

  La classe java.lang è intrinsecamente appartenente a tutti i package creati. Dunque non deve essere importata.

  Una classe appartenente ad un package esterno a quello della classe principale deve essere SEMPRE importata.

  es. import java.util.*;
  Scanner sc = new Scanner(System.in);

  Tuttavia, una classe statica può essere importata tramite il comando static e tutti suoi metodi possono essere di conseguenza utilizzati senza anteporre il nome della classe.

  es. import static java.lang.Math,*;
  int x = sqrt(27);

PASSAGGIO PER VALORE E RIFERIMENTO

In C++ una variabile può essere passata ad un metodo per

- Valore: viene fatta una copia della variabile e questa è passata al metodo. Modifiche al valore passato però non influenzano la variabile nel codice
- Riferimento: viene passato l'indirizzo della variabile che ne permette dunque la modifica effettiva

In Java le variabili associate a tipi primitivi sono passate a un metodo come valore. Dunque, qualsiasi modifica ad una variabile di tipo primitivo non ha ripercussioni al di fuori del metodo.

Gli oggetti invece vengono passati ai metodi per riferimento dunque è possibile modificarli e tali modifiche sono effettive anche al di fuori del metodo stesso. Se si vuole passare un oggetto e, per chiarezza nel caso di tipi primitivi o per non riassegnare un oggetto e quindi perdere quello di partenza, si possono definire final i parametri formali

es. public int m(final int x, final Punto p) {
Punto d = new Punto(5,6);
p = d; } //ERRORE -> un oggetto final NON può essere riassegnato ma ne posso modificare i campi

Inoltre:

- Variabili locali: variabili di supporto create localmente all'interno del metodo e che vengono automaticamente cancellate quando termina
- Parametro formale: parametro con nome generico utilizzato come argomento quando si scrive un metodo
- Parametro attuale: parametro reali passato come argomento quando si invoca un metodo

COSTRUTTORE

In una classe in cui non viene proggettato un costruttore, Java lo fornisce di default ed esso si occuperà ad inizializzare tutte le variabili in questo modo:

- Boolean: false
- int: 0

- Oggetto: null

MASCHERAMENTO

Una variabile può essere "mascherata" da un'altra variabile presente in un blocco più interno e con lo stesso nome. Per risolvere questa sovrapposizione, nel blocco interno si usa il nome per riferirsi alla variabile locale mentre this.nome per riferirsi alla variabile più globale

```
es. class C {
        int x = 5;
        …

        void m (int x) {
                int z = x + this.x;
        } // x fa riferimento al parametro formale mentre this.x alla variabile sopra istanziata
```

VISIBILITA' DI UNA CLASSE

Gli attributi di una classe (campi o metodi) possono essere definiti come:

- **public:** visibilità estesa a tutte le classi che la importano
- **private:** visibilità solo nella classe in cui viene dichiarato
- **protected:** visibili solo alle classi eredi e alle classi dello stesso package
- **(nulla):** visibilili solo all'interno dello stesso package
- **final:** un metodo final non può essere ridefinito in una classe erede

Un metodo ridefinito può avere una visibilità maggiore ma non minore rispetto al metodo originario (es. da protected a public ma non viceversa).

I modificatori di una classe sono:

- **public:** visibilità estesa a tutte le classi che la importano
- **(nulla):** visibilili solo all'interno dello stesso package
- **final:** non può essere ridefinito nelle classi eredi
- **abstract:** prevede almeno un metodo astratto che deve essere implementato in una classe erede

CLASSI ASTRATTE

Una classe astratta prevede almeno un metodo abstract che deve essere implementato in una classe erede e può anche avere metodi e istanze concrete. Una classe erede che implementa tutti i metodi astratti è detta concreta. Una classe erede che implementa solo alcuni metodi astratti è astratta a sua volta e gli ulteriori metodi astratti rimansti devono essere implementate da ulteriori classi eredi.
Una classe erede ESTENDE una classe astratta:

```
es. public abstract class Sortable { … }:
    public class Intero extends Sortable { … };
```

INTERFACCIA

Una interfaccia è una classe in cui vengono scritte le intestazioni dei metodi che devono essere implementati in una classe concreta e tali segnature sono astratte anche senza il modificatore abstract. Una classe che implementa una interfaccia deve implementare tutti i metodi. Una classe può implementare zero o più interfacce.
Una classe erede IMPLEMENTA una interfaccia:

es. public interface Comparable { … };
public class Razionale implements Comparable { … };

In un'interfaccia è possibile specificare per un metodo la sua implementazione, tramite la keyword default, definendo il comportamento di default. Le classi che implementano l'interfaccia possono scegliere di definire una propria implementazione del metodo o usare quella fornita dall'interfaccia.

VARARG

Un metodo in Java può ricevere un numero indefinito di parametri attraverso l'espressione dei tre punti. Essi vengono inseriti in un array e resi disponibili al metodo.

es. public static int max(int… x) // il metodo max può ricevere un numero variabile di argomenti

TIPO STATICO E TIPO DINAMICO

Quando si hanno due classi, una erede dell'altra, si dice che la prima ha

- Tipo statico: classe (la superclasse) che definisce cosa si può fare con la sottoclasse
- Tipo dinamico: classe (la sottoclasse a cui appartiene) che definisce quali sono gli effettivi metodi mandati in esecuzione

DYNAMIC BINDING

Il dyanmic binding si ha quando su un oggetto di una classe generale può essere applicato o il metodo della superclasse o della classe erede in base al tipo dinamico (vedi sopra) dell'oggetto su cui si invoca

POLIFORMISMO

Un oggetto erede di una classe risulterà istanza sia della superclasse che della classe erede.
Un oggetto della superclasse, se non castizzato al tipo dell'erede, è SOLO istanza della superclasse

OVERLOADING E OVERRIDING

Overload: riscrittura di un metodo con modifica di parametri e/o intestazione
Overriding: modifica solo del corpo del metodo che mantiene la stessa intestazione e gli stessi parametri
Nel caso di dynamic binding e poliformismo, si verificano correttamente solo se vi è l'overriding dei metodi

HASHCODE

L'hashcode restituisce un intero che identifica univocamente un oggetto. Due oggetti uguali secondo l'equals hanno stesso hashcode mentre due oggetti non uguali possono comunque avere lo stesso hashcode. Il metodo usa una tecnica canonica per cui si combinano gli hashcode degli elementi che lo compongono e utilizzando un fattore di shuffling quale un numero primo

```
es. public int hashCode() { ì
        final int M=83;
        int h=cognome.hashCode()*M+nome.hashCode();
        return h;
    }//hashCode
```

ENUMERAZIONE

Una enumerazione definisce una classe implicita che elenca tutti e soli i valori che gli oggetti di quella classe possono assumere. Poichè una enumerazione è una classe a tutti gli effetti, per ogni valore che l'oggetto

della classe può assumere, posso associare degli attributi. Per fare ciò li istanzio come variabili d'istanza, scrivo un costruttore e volendo anche i metodi get e set. Sono presenti dei metodi di default

SINGLETON

Il singleton ha come scopo quello di garantire che di una determinata classe venga creata una ed una sola istanza, e di fornire un punto di accesso globale a tale istanza. Il modo più semplice per implementare una classe singleton richiede di rendere privato il costruttore della classe e implementare un metodo statico (detto factory) che istanzia e restituisce un oggetto della classe.

METODO FACTORY

Indirizza il problema della creazione di oggetti senza specificarne l'esatta classe. Questo pattern raggiunge il suo scopo fornendo un'interfaccia per creare un oggetto, ma lascia che le sottoclassi decidano quale oggetto istanziare