OOP: Programmazione ad Oggetti

Prof. Francesco Viglietti
Www.in4matika.altervista.org

Introduzione

La OOP è un'**evoluzione** della programmazione strutturata, nel senso che aggiunge nuovi concetti a quanto di già "collaudato" era stato definito: concetto di variabile, costante, strutture di controllo, dati strutturati, ecc.

L'idea di base della programmazione orientata agli oggetti è quella di strutturare i programmi in modo tale da riflettere l'ordine delle cose del mondo reale.

Nel mondo reale esistono **oggetti semplici**, autonomi, ognuno dei quali ha caratteristiche e funzionalità ben determinate e sui quali è possibile effettuare determinate azioni.

Gli oggetti semplici possono essere combinati tra di loro in modo da formare **oggetti complessi**, con altri ruoli rigorosamente definiti

Esempio

Supponiamo di voler assemblare un PC mettendo insieme singoli componenti indipendenti (SVGA, CPU, MB, tastiera, ...). Ogni singolo componente contribuisce a costituire un sistema più grande il PC appunto. Le varie unità sono autonome e soprattutto sono capaci di interagire con gli altri componenti.(dialogano)

La SVGA ad esempio, può essere prodotta da case diverse purchè sia compatibile con la CPU, con il monitor, …

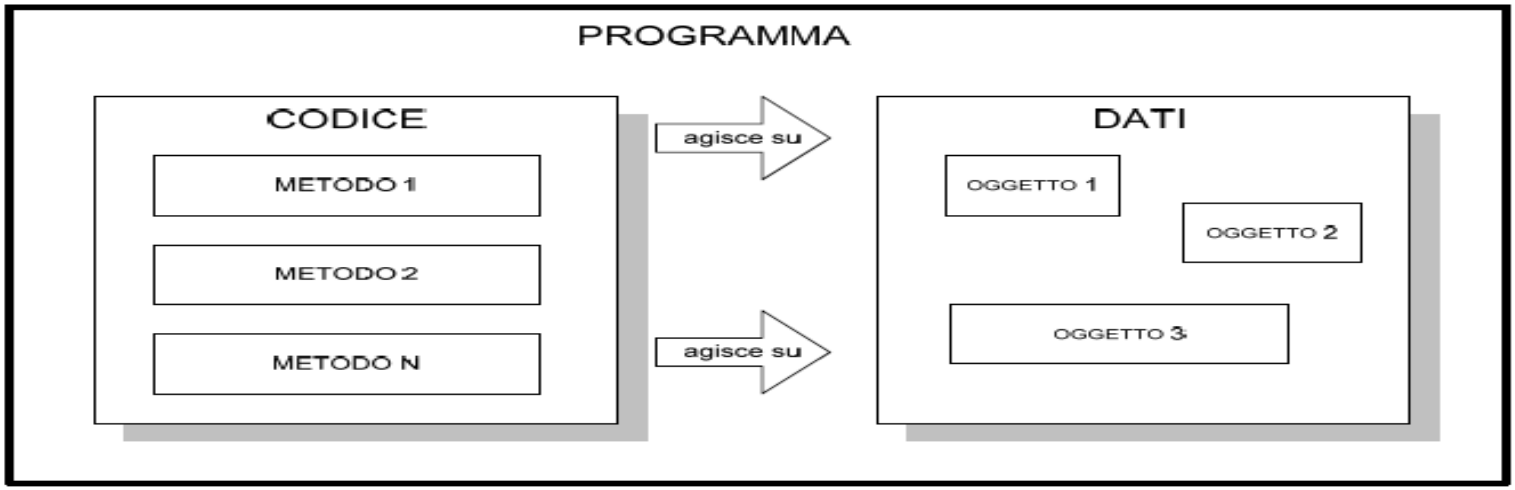
Ogni componente è stato progettato e realizzato da produttori diversi.

Ma affinché il computer funzioni correttamente, ogni suo componente deve "dialogare" con tutti gli altri.

Ciascun produttore non conosce, necessariamente, la struttura interna dei componenti realizzati dagli altri produttori, ma ogni costruttore deve fornire per ogni componente, il modo in cui dialoga con l'esterno, e le sue funzionalità ovvero il ruolo che può svolgere.

OOP vs programmazione procedurale.

La situazione di partenza è la seguente:



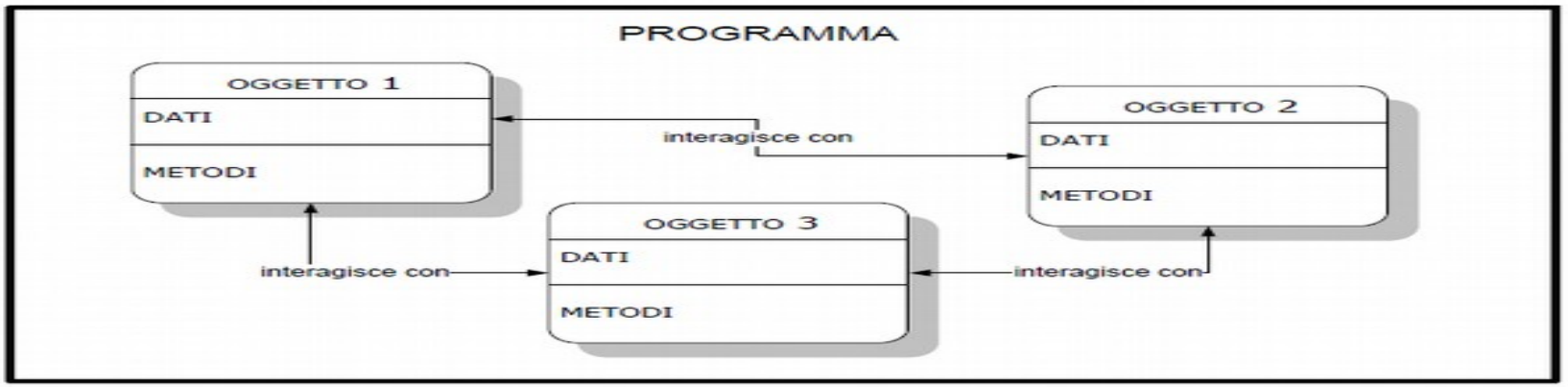
Si può riassumere dicendo che un programma (codice e dati) può essere suddiviso in 2 parti. La parte del codice, che a sua volta comprende vari metodi in cui il problema viene scomposto per la sua risoluzione, e la parte di dati che fa riferimento alla presenza di più oggetti (variabili o altri tipi di dato).
Queste 2 parti sono distinte pur entrando in contatto tra loro durante l'esecuzione del programma, in particolare sarà il codice (insieme di istruzioni) ad agire sui dati per produrre un certo risultato.

OOP vs programmazione procedurale.

In OOP questo aspetto assume una rappresentazione diversa. Definiamo alcuni aspetti della OOP:

- I dati non sono separati dal codice che li elabora
- Cambia il ruolo di un oggetto che non sarà solo un contenitore di dati, ma ***fornisce sia una rappresentazione per i dati, sia l'insieme delle operazioni che possono essere eseguite su di essi.***

Chiariamo il concetto con una rappresentazione grafica:



Definizioni 1

Gli oggetti sono elementi indipendenti tra loro che interagiscono, al loro interno contengono sia dati che metodi.

Un **oggetto** è la realizzazione pratica di un qualcosa descritto in teoria. Esso possiede caratteristiche e funzionalità per utilizzarlo.

Quando un architetto realizza una casa, fa il progetto su carta descrivendone dimensioni, caratteristiche ed eventualmente le tecniche di costruzione. Questo lavoro (Progetto) rappresenta la descrizione teorica di un oggetto (casa) che dovrà essere costruito. In questo esempio, il progetto rappresenta la **classe**, mentre la casa realmente costruita, rappresenta l'**oggetto**. Usando la terminologia ad hoc, si dice che è stato **istanziato l'oggetto** casa cioè è stata creata un'**istanza della classe** (Progetto). In parole povere le classi sono definizioni usate per la creazione di oggetti. Dal progetto dell'architetto, si possono costruire più case.

Definizioni 2

Da una **classe** quindi si possono **istanziare** più **oggetti**, dipende da quanti oggetti dello stesso tipo abbiamo bisogno in una applicazione. Ad esempio:

la **classe** Studente descrive tutte le caratteristiche di uno studente, per es. il nome, il cognome, la provenienza, ecc.

Con una **istanza** della **classe** Studente, che chiameremo **Alunno**, creeremo effettivamente un **oggetto** reale a partire della **classe** e che potremo usare nel nostro programma. E' ovvio che in una aula di una scuola c'è più di un alunno, quindi istanzieremo più **classi** Studente per formare la nostra aula di oggetti **Alunno**.

Riepilogo

Una **classe** è un modello astratto, generico per una famiglia di oggetti con caratteristiche comuni.

Un'**istanza** (**oggetto**), è la rappresentazione concreta e specifica di una **classe.**

Gli **attributi** o proprietà specificano le caratteristiche che tutti gli oggetti della classe devono possedere.

I **metodi** specificano le funzionalità che la classe offre, cioè le operazioni che un oggetto è in grado di compiere. Corrispondono ai comportamenti dell'oggetto e possono essere rivolti alla modifica dello stato dell'oggetto (modifica attributi) o alla comunicazione dello stato all'esterno dell'oggetto stesso (visualizzazione attributi)

Riepilogo


Class
Definition of objects that share structure, properties and behaviours.
Building
class
Dog
class
Computer
class
Instance
Concrete object, created from a certain class.
Empire State
instance of Building
Lassie
instance of Dog
Your computer
instance of Computer

Esempio classe e istanze

La classe *Automobile* definisce:

- le **caratteristiche** che descrivono **come è fatta** un'automobile (ha una marca, ha un modello, ha un motore e così via);
- le **funzionalità** che descrivono **che cosa si può fare** con un oggetto di classe *Automobile*, cioè i suoi possibili comportamenti (è possibile avviarla, fare rifornimento e così via).

Esempi di istanze della classe *Automobile* sono:

- l'auto di Paolo con targa AB 456 TK;
- l'auto di Andrea con targa TF 231 TD.

Esempio attributi e metodi

Nella classe *Motocicletta* possiamo individuare i seguenti attributi e metodi:

Attributi	Possibili valori degli attributi
Marca	Marca1, Marca2, Marca3
Modello	Strada, Enduro, Custom, Corsa
Colore	Rosso, Verde, Bianco, Nero
Cilindrata	125, 250, 500,...
StatoMotore	Vero (acceso), Falso (spento)
MarciaInserita	Folle, Prima, Seconda,...

Rappresentano lo stato dell'oggetto

Metodi

AvviaMotore()
SpegneMotore()
Accelera()
Decelera()
MarciaInferiore()
MarciaSuperiore()
Rifornimento()

Modificano lo stato dell'oggetto

èGuasta()
èInRiserva()

Comunicano lo stato dell'oggetto

File delle classi

Ogni classe in java inizia con una lettera maiuscola!
Ogni classe formerà un file con estensione ***.java***, la classe compilata avrà invece l'estensione ***.class*** .
Tutti i programmi possiedono almeno una classe ed il metodo *main* che rappresenta l'entry point del programma.
Se in un programma vengono definite più classi esse saranno salvate nella directory del programma con il proprio nome, in questo modo non occorre preoccuparsi del posizionamento. Vedremo più avanti come gestire classi in directory differenti.

Dichiarazione di una classe

```
class NomeClasse
{
   // attributi(variabili d'istanza)
   // metodi(metodi d'istanza)
}
```

Dichiarazione di un oggetto

- **dichiarazione di un oggetto:**

```
NomeClasse nomeOggetto;
```

- **creazione dell'istanza**:

```
nomeOggetto = new NomeClasse();
```

Dichiarazione di un attributo

- **dichiarazione di un attributo:**

```
livelloDiVisibilità tipo nomeAttributo;
```

- **Esempi:**

```
private int x, y, z;

public double altezza = 15.76;
```

Livelli di visibilità

- **public**
  l'attributo è accessibile da qualsiasi altra classe
- **private**
  permette di nascondere l'attributo all'interno dell'oggetto
- **protected**
  è visto all'esterno solo dalle classi che appartengono alla stessa libreria oppure dalle sottoclassi della classe in cui è stato dichiarato l'attributo.

Altre caratteristiche

- **static**

  indica un attributo legato alla classe, cioè esiste solo una copia dell'attributo. Un attributo static è quindi condiviso da tutte le istanze della classe, che ne possono leggere e modificare il contenuto comune. Per usarlo non è necessario creare l'oggetto, ma può essere usato con la classe direttamente.

- **final**

  se lo si vuole rendere una costante, in modo che possa assumere un unico valore.

Dichiarazione dei metodi

```
livelloDiVisibilità tipoRestituito nomeMetodo
(parametri){
  // variabili
  // istruzioni
}
```

Esempio: di un metodo che esegue la somma di due numeri interi passati come parametri:

```
public int addizione (int a, int b){
   int sum;
   sum = a+b;
   return sum;
}
```

Valore di ritorno

- Il **tipo del valore di ritorno** indica quale sarà il valore restituito al termine dell'esecuzione del metodo.
- Un metodo può anche non restituire alcun valore: parola chiave **void.**

```
public void setRaggio(double r){
   raggio = r;
}
```

Istruzione return

```
public int max(int a, int b){
   if (a > b)
      return a;
   else
      return b;
}
```

```
public int max(int a, int b){
   int tmp=0;
   if (a > b)
      tmp=a;
   else
      tmp=b;
Return tmp;
}
```

Parametri

- In Java, i parametri possono essere passati ai metodi solo **per valore**.
- Per quanto riguarda i **tipi riferimento**, cioè gli array e gli oggetti, viene creata una copia del riferimento e non dell'oggetto.

Creazione di oggetti

- L'**allocazione dell'oggetto** riserva lo spazio per memorizzare gli attributi dell'oggetto e viene codificata con l'operatore **new** seguito dal nome della classe.

Viene attivato in modo implicito un particolare metodo predefinito, detto **costruttore** della classe, che consente di creare un oggetto.

```
Cerchio tavolo = new Cerchio();
```

Utilizzo degli oggetti

- Accesso agli attributi di un oggetto con l'**operatore punto**:

```
nomeOggetto.attributo
```

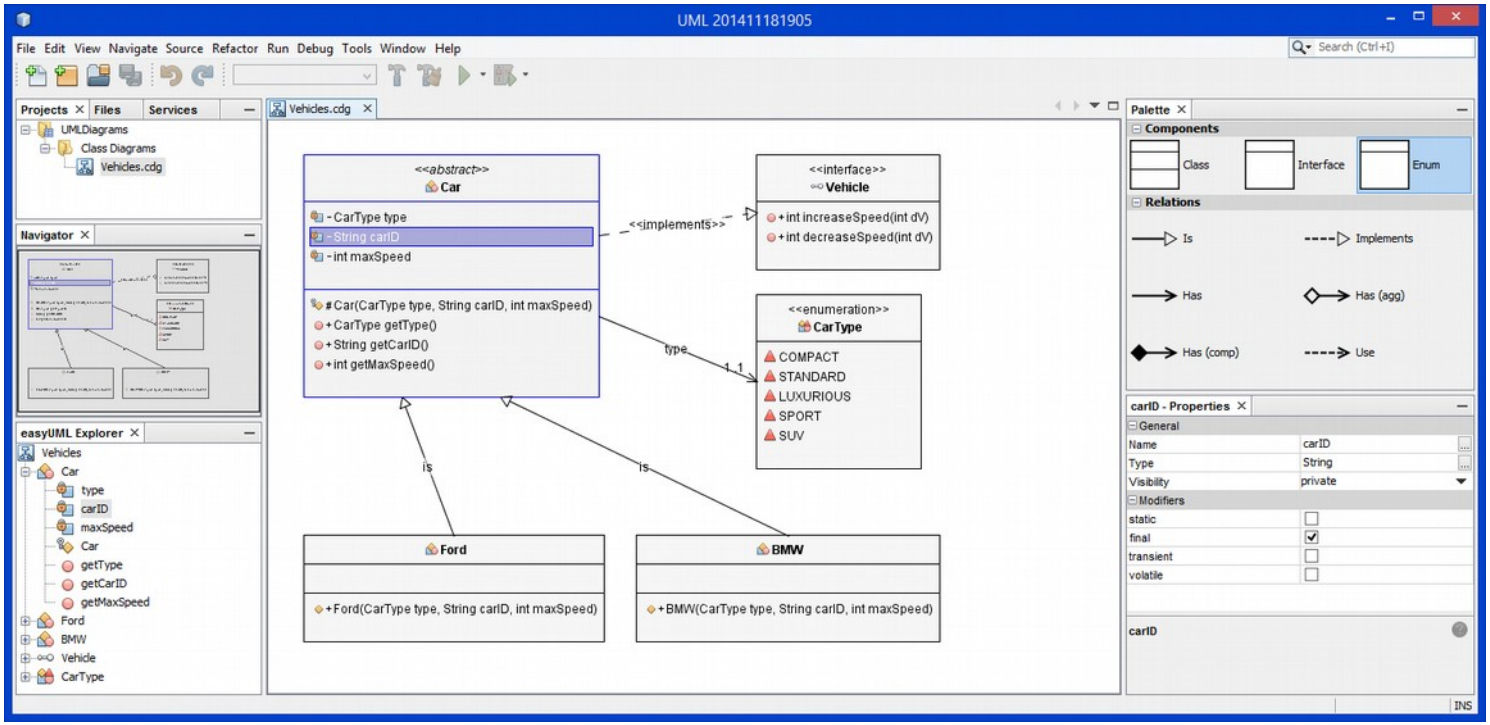
- **Invocazione di un metodo:**

```
nomeOggetto.metodo(parametri)
```

- In ogni classe è implicitamente presente un oggetto speciale identificato dalla parola chiave **this**, per indicare l'oggetto a cui il costruttore si riferisce. In questo caso possono essere usati gli stessi nomi tra parametri e attributi.

Aggiungere plugins easyUML

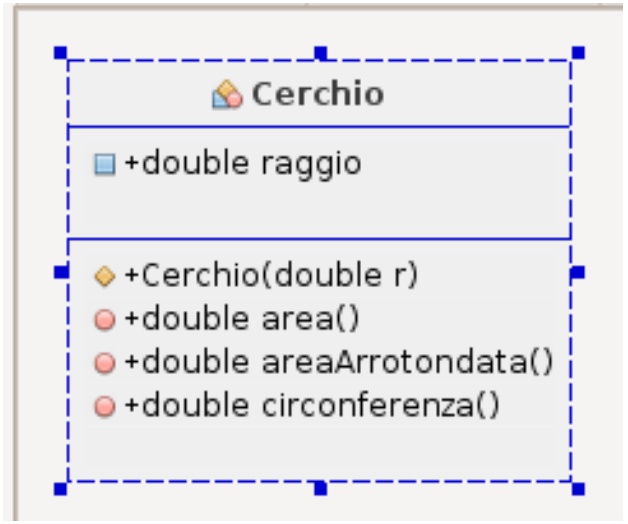
Su Netbeans, andare sul menù tools → plugins cercare easyUML, installare. Quindi andare su nuovo progetto UML, crearlo, quindi aggiungere un nuovo diagramma...



Esempio: UML

Scrivere un programma che calcoli l'area e la circonferenza di 2 cerchi

Diagramma UML
della classe



Esempio: Codice classe

```java
public class Cerchio {
    //attributo
    public double raggio;
    // costruttore
    public Cerchio(double r){
        raggio = r;
    }
    // metodo per calcolare l'area
    public double area(){
        return (raggio * raggio * Math.PI);
    }
    // metodo per calcolare l'area arrotondata
    public double areaArrotondata(){
        double areaTemp;
        areaTemp = area() * 1000;
        areaTemp = Math.round(areaTemp);
        areaTemp = (double) areaTemp / 1000;
        return areaTemp;
    }
    // metodo per calcolare la circonferenza
    public double circonferenza(){
        return (2 * raggio * Math.PI);
    }
}
```

Esempio: main

```
public static void main(String[] args) {

    Scanner in=new Scanner(System.in);
    System.out.print("Inserire il primo raggio: ");
    int RaggioCerchio=in.nextInt();

    Cerchio objCerchio = new Cerchio(RaggioCerchio);
    System.out.println("Primo cerchio.");
    System.out.println("Circonferenza = " + objCerchio.circonferenza());
    System.out.println("Area = " + objCerchio.areaArrotondata() );

    System.out.print("Inserire il secondo raggio: ");
    RaggioCerchio=in.nextInt();
    objCerchio.raggio=RaggioCerchio;
    System.out.println("Secondo cerchio.");
    System.out.println("Circonferenza = " + objCerchio.circonferenza() );
    System.out.println("Area = " + objCerchio.area() );
    System.out.println("Area = " + objCerchio.areaArrotondata() );
}
```