Object Oriented Programming

Programmazione Orientata agli Oggetti

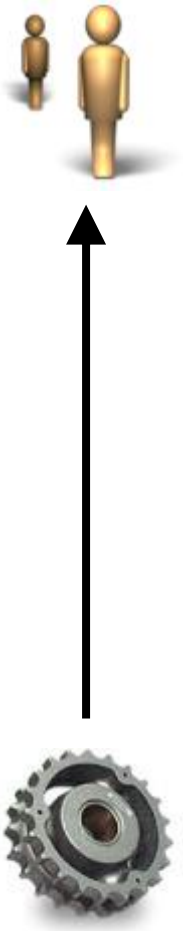
- la **programmazione orientata agli oggetti *(Object Oriented Programming)*** ha l'obiettivo di formalizzare gli ***oggetti*** del mondo reale e di costruire con questi un ***mondo virtuale***
- questa parte di mondo che viene ricostruita in modo virtuale è detta ***dominio applicativo***

- quotidianamente interagiamo con oggetti del mondo che ci circonda
- oggetti:
  - animali
  - piante
  - tutti gli oggetti inanimati del mondo reale
  - un pensiero, una filosofia o più in generale un'entità astratta
- un esempio di oggetto astratto: il voto

- esempio: un bicchiere
- ne sappiamo definire le ***caratteristiche*** e conosciamo anche quali ***azioni*** si possono fare con esso
- possiamo definirne la forma, il colore, il materiale di cui è fatto e possiamo dire se è pieno o vuoto
- sappiamo anche che si può riempire e svuotare
- abbiamo definito un oggetto attraverso
  - le sue ***caratteristiche***
  - le ***operazioni*** che può compiere

livello di astrazione

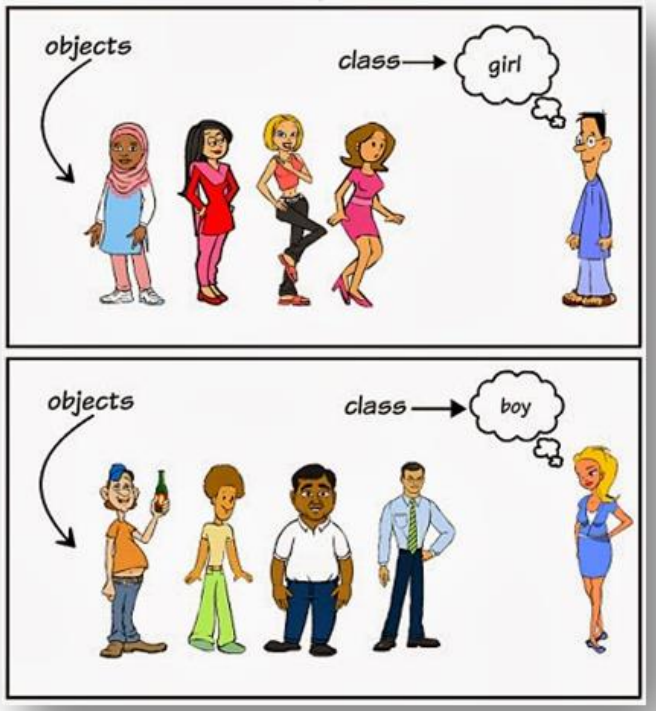
- ***evoluzione*** dei linguaggi di programmazione
  - i codici sorgenti sono sempre più astratti rispetto al codice macchina
- nella OOP non ci si vuole più porre i problemi dal punto di vista del calcolatore, ma si vogliono risolvere facendo ***interagire oggetti*** del dominio applicativo come fossero oggetti del mondo reale
- obiettivo: formalizzare soluzioni ai problemi, pensando come una persona senza doversi sforzare a pensare come una macchina



- per popolare il dominio applicativo utilizzato dall'applicazione è necessario ***creare gli oggetti***, e per fare questo è necessario definire le ***classi***
- una classe è lo strumento con cui si identifica e si crea un oggetto

una classe è un modello per la creazione di oggetti

- la classe è paragonabile allo stampo
- gli oggetti sono i biscotti ottenuti con quello stampo



- una classe è a tutti gli effetti un ***tipo di dato*** (come gli interi e le stringhe e ogni altro tipo già definito)
- un tipo di dato è definito dall'insieme di ***valori*** e dall'insieme delle ***operazioni*** che si possono effettuare su questi valori
- nella programmazione orientata agli oggetti, è quindi possibile sia utilizzare tipi di dato esistenti, sia definirne di nuovi tramite le classi
- i nuovi tipi di dato si definiscono ***ADT*** *(Abstract Data Type)*

incapsulamento *(information hiding)*

- nascondere il funzionamento interno (la struttura interna)
- fornire un'interfaccia esterna che permetta l'utilizzo senza conoscere la struttura interna

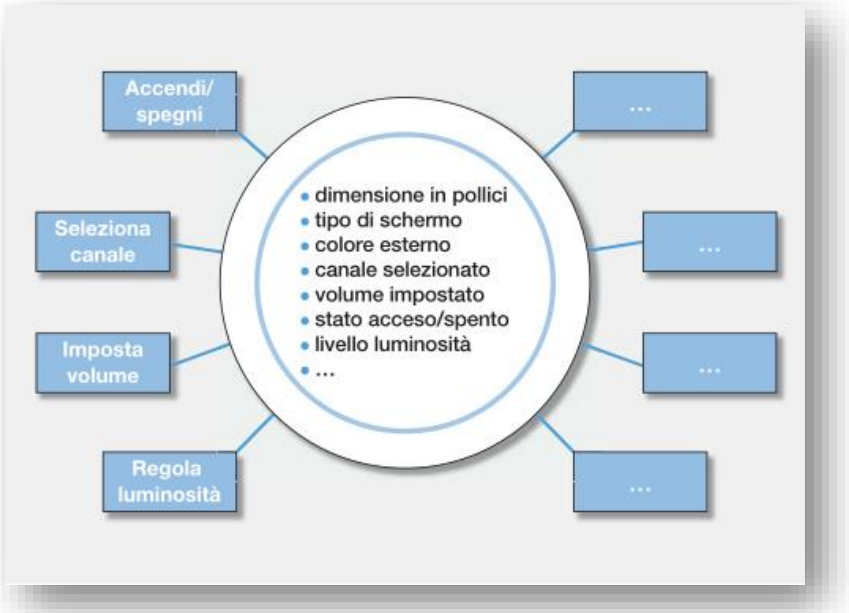
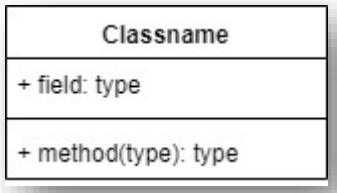
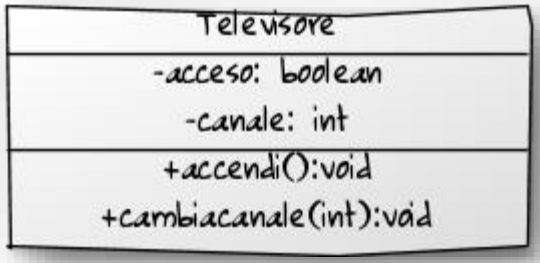


diagramma delle classi

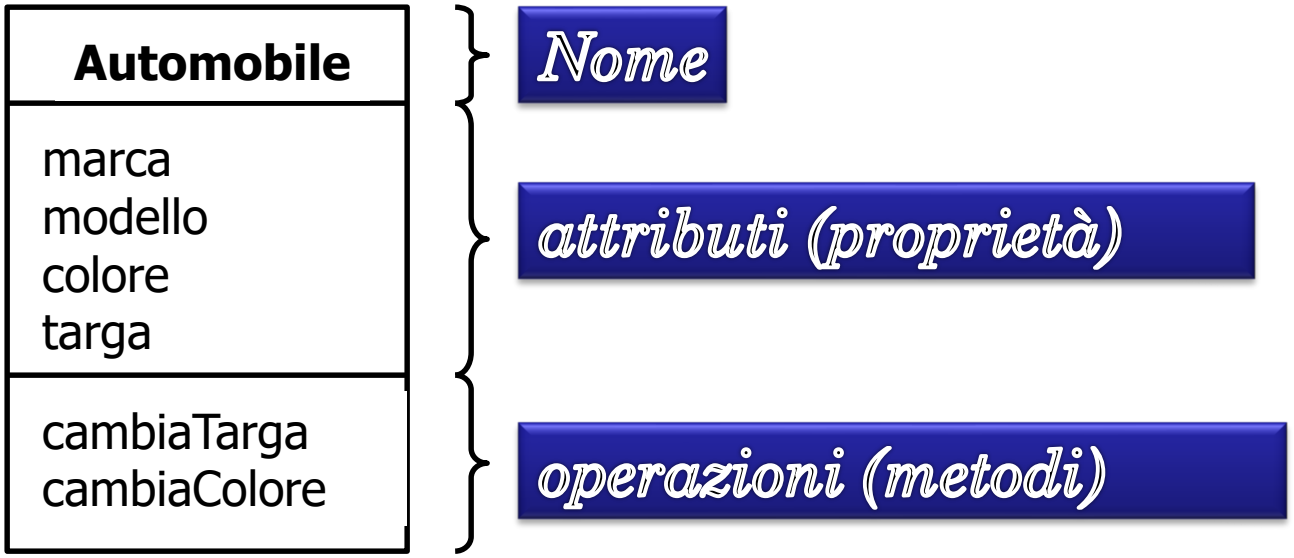




http://www.draw.io

- la prima sezione contiene il ***nome*** della classe
- la seconda sezione definisce i suoi ***attributi***
- la terza i ***metodi***, le operazioni che si possono compiere sull'oggetto

un esempio

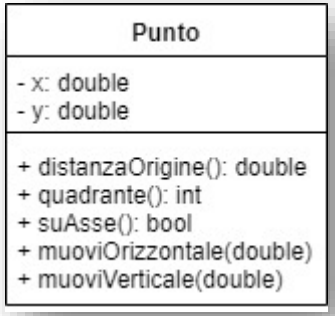

Automobile
marca
modello
colore
targa
cambiaTarga
cambiaColore
Nome
attributi (proprietà)
operazioni (metodi)

esempio: punto sul piano cartesiano

A. Ferrari

- definire mediante class diagram la classe ***Punto*** che permette di istanziare oggetti che rappresentano punti sul piano cartesiano.
- ***proprietà*** (caratteristiche) (attributi) (fields)
  - ***x*** che rappresenta l'ascissa
  - ***y*** che rappresenta l'ordinata
- ***funzionalità*** (metodi) (operazioni)
  - ***distanzaOrigine*** per ottenere la distanza del punto dall'origine
  - ***quadrante*** per ottenere il quadrante in cui è posizionato il punto
  - ***muoviOrizzontale*** e ***muoviVerticale*** che spostano il punto di un valore stabilito

Punto: class diagram

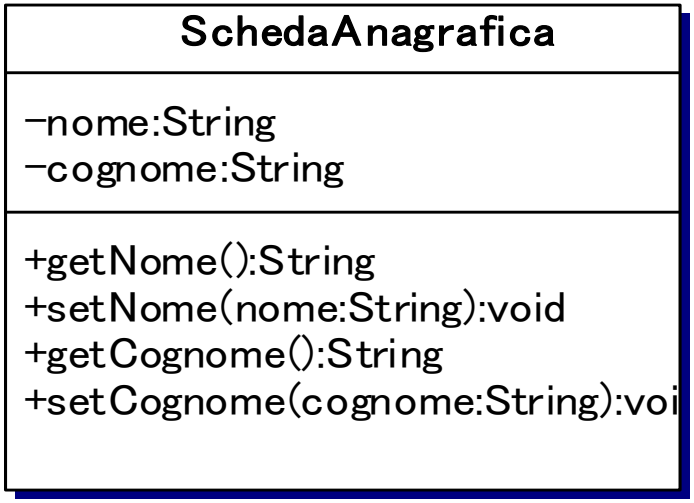


- di ogni automobile interessa il ***modello*** (es. Audi e-tron Quattro), la ***cilindrata*** (es. 2000), il tipo di ***motore*** (b per benzina, d per diesel, e per elettrico, m per metano), la norma ***Euro*** (es. 4), il ***numero di marce*** (es. 6) lo ***stato*** del motore (acceso, spento) e la ***marcia attuale*** (0 per folle, 1,2 …)
- sono da realizzare le funzionalità:
  - ***accendi***, ***spegni***, ***aumentaMarcia***, ***scalaMarcia***, ***cambiaMarcia*** che richiede come parametro la nuova marcia, ***puoCircolare*** che restituisce true se in base alle normative vigenti può circolare nelle giornate ecologiche
- realizzare il class diagram (www.draw.io) e salvare il file automobile.xml e automobile.jpg

```
[modificatore] class [nome della classe]{
  [attributi]
  [metodi]
}
```

```
class MiaClasse {
  String mioAttributo;
  void mioMetodo() {
  }
}
```

```
public class SchedaAnagrafica {

    private String nome;

    private String cognome;

    public String getNome() {
      return nome;
    }
    public void setNome(String nome) {
           this.nome = nome;
    }
    public String getCognome() {
      return cognome;
    }
    public void setCognome(String cognome) {
      this.cognome = cognome;
    }
}
```



gli oggetti

- gli oggetti sono le ***entità*** di un programma che ***interagiscono*** tra loro per raggiungere un obiettivo
- vengono ***creati*** (istanziati) in fase di esecuzione
- ognuno di essi fa parte di una categoria (una ***classe***)
- ogni classe può creare ***più oggetti***, ognuno dei quali, pur essendo dello stesso tipo, è ***distinto*** dagli altri
- un oggetto è una ***istanza*** di una classe

- anche se due oggetti dello stesso tipo hanno tutti gli attributi con gli stessi valori, non sono uguali, ma sono oggetti distinti
- sarebbe come dire che due gemelli, solamente perché identici fisicamente, siano la stessa persona: ovviamente è scorretto

esempio di classe

- se vogliamo catalogare i cd musicali in nostro possesso, abbiamo bisogno di implementare un programma nel cui dominio applicativo è presente la classe CD
- i metodi della classe CD servono per impostare e recuperare i valori degli attributi

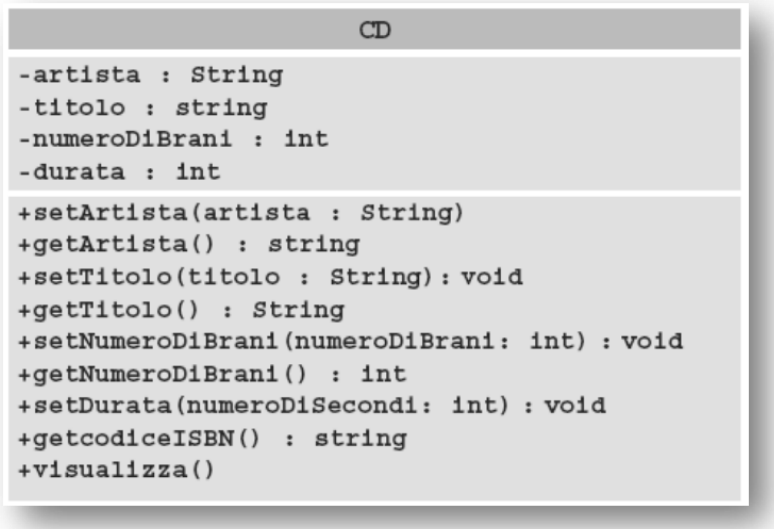
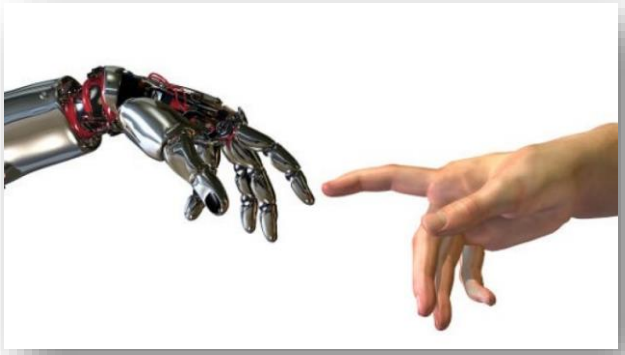


diagramma degli oggetti

- i diagrammi che rappresentano gli oggetti ***(Object Diagram** in **UML)*** mettono in luce i ***valori*** che assumono gli attributi

- l'insieme dei valori degli attributi di un oggetto è chiamato stato dell'oggetto e generalmente può variare in funzione del tempo

- per creare un oggetto si effettua una ***istanziazione*** di una classe
- in questa fase viene riservato uno spazio di memoria per conservare i valori degli attributi dell'oggetto che si sta creando (per mantenere memorizzato lo stato dell'oggetto)



istanziare un oggetto in Java

- a seconda del linguaggio utilizzato si impiegano diversi costrutti di programmazione per creare un oggetto
- in Java la creazione di un oggetto si effettua mediante l'istruzione **new**
- esempio:

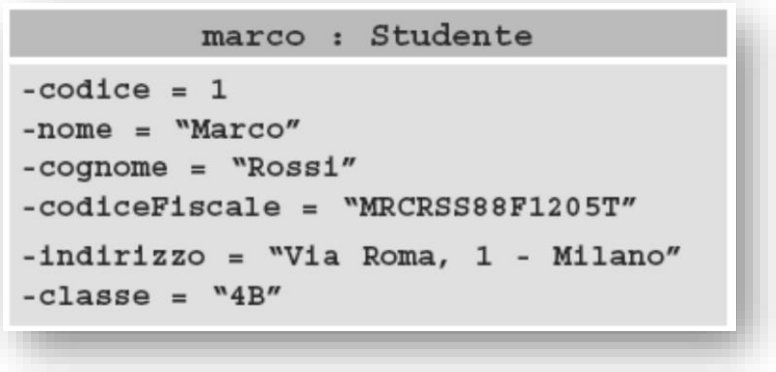
```
Bicchiere calice;
calice = new Bicchiere();
```

- oppure:

```
Bicchiere calice = new Bicchiere();
```

gli attributi di istanza

- gli attributi di istanza sono quelli posseduti da un oggetto, chiamati anche più semplicemente **attributi**
- un attributo di un oggetto è una variabile che ne descrive una caratteristica o ***proprietà***



- un attributo ***costante*** è un attributo il cui valore resta invariato nel tempo
- in Java per dichiarare una costante si utilizza il modificatore ***final***

```
public class Calendario {
   public final int numeroDeiMesi = 12;
   // Metodi
}
```

- il valore di **`numeroDeiMesi`** non può essere modificato, ma resta invariato nel corso dell'esecuzione del codice

un ***attributo di classe*** è un attributo ***condiviso*** da tutte le istanze della classe, cioè da tutti gli oggetti creati con questa

- in Java per dichiarare un attributo di classe si utilizza il modificatore **`static`**.

```
public class Gatto {
  public static int numeroDiGatti = 0;
  public Gatto() {
    numeroDiGatti ++;
  }
}
```

- ogni volta che viene creato un oggetto di tipo **`Gatto`**, il contatore **`numeroDiGatti`** è automaticamente incrementato di uno
- La sintassi per accedere ad un attributo di classe è:
  `<NomeClasse>.<NomeAttributo>`
  per esempio
  `System.out.print(Gatto.numeroDiGatti);`

metodi:
le azioni degli oggetti

- un metodo è un'azione che l'oggetto può eseguire
- in Java la dichiarazione di un metodo è composta da:
  - Modificatore
  - Nome del metodo
  - Tipo di dato da ritornare
  - Tipo e nome dei parametri di ingresso
  - Eventuali eccezioni sollevate
- tutto questo è detto **firma del metodo**

- un metodo di istanza è un metodo che, per essere utilizzato, ha bisogno della creazione di un oggetto della classe a cui appartiene su cui essere invocato
- un metodo di istanza è anche chiamato semplicemente **metodo**

esempio di firma

- `public int studia(String testo)throws HoStudiatoTroppoException`
- **`public`** è il modificatore
- **`int`** è il tipo del metodo
- **`studia`** è il nome del metodo
- **`String testo`** è il tipo e nome del parametro
- **`HoStudiatoTroppoException`** è la possibile eccezione sollevata

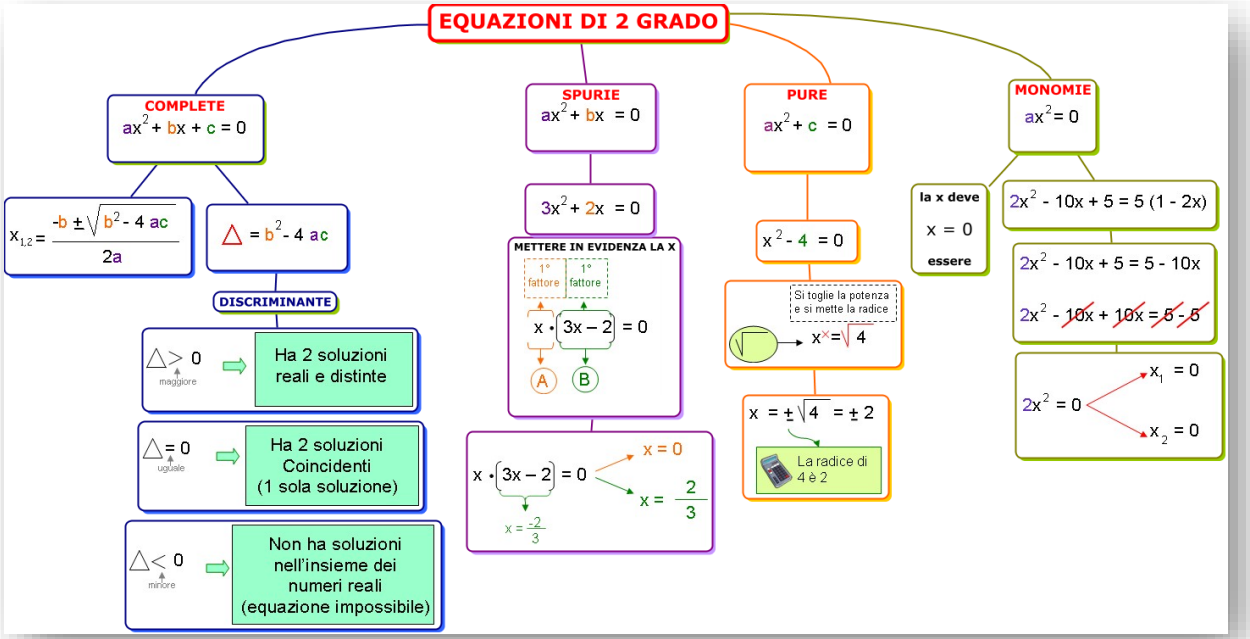
formalizzare i metodi

A. Ferrari

```
public void mangia(int[] portate) {

  for(int i; i < portate.length; i++) {

    calorie += portate[i];
  }
}

public void bevi(Bicchiere bicchiere) {

  bicchiere.svuota();
  liquidi += bicchiere.capienza;
}

public int corri(int kilometri) {

  calorie -= kilometri * peso * 0.9;
  acidoLattico += kilometri * 10;

  return calorie;
}

public void defaticare() {

  acidoLattico -= 50;
}
```

un esempio: attributi

- si vuole realizzare una classe che permetta di gestire e risolvere equazioni di secondo grado
- in una equazione individuiamo tre **attributi**: **`a, b, c`** che rappresentano i coefficienti di **x^2**, di **`x`** ed il termine noto
- l'equazione **$3x^2-2x+1=0$** avrà come attributi i valori 3, −2 e 1



un esempio: metodi

- definiamo un insieme di metodi che ci permetta di:
  - modificare i valori dei coefficienti
  - ottenere i valori dei coefficienti
  - conoscere il tipo di equazione
  - ottenere la prima soluzione
  - ottenere la seconda soluzione

diagramma UML della classe

Equazione
-a : double -b : double -b : double
+setA(in v : double) +getA() : double +setB(in v : double) +getB() : double +setC(in v : double) +getC() : double -delta() : double +pura() : boolean(idl) +spuria() : boolean(idl) +complessa() : boolean(idl) +soluzione1() : double +soluzione2() : double

- implementare in Java la classe Equazione
- istanziare due equazioni:
  - $5x^2-3x+2=0$
  - $2x^2-4=0$

eq1 : Equazione
a : double = 5
b : double = -3
b : double = 2

eq2 : Equazione
a : double = 2
b : double = 0
b : double = -4

- un metodo di classe è un metodo ***invocabile sulla classe*** stessa senza dovere necessariamente istanziare un oggetto
- i metodi di classe sono principalmente utilizzati per inglobare al loro interno algoritmi, o in generale operazioni che non cambiano lo stato di un oggetto

metodi di classe, quando?

- quando si devono modificare o leggere attributi di classe riguardanti informazioni inerenti a tutti gli oggetti della classe
- quando non ha senso creare oggetti di una certa classe, in quanto questa possiede solo metodi di utilità

- in Java i metodi di classe si implementano utilizzando il modificatore **static**

```
public class Matematica {
  public static int somma(int addendo1,
  int addendo2) {
    return addendo1 + addendo2;
  }
}
```

- per invocare un metodo static si utilizza la tradizionale notazione puntata, ma al posto del nome dell'oggetto si inserisce il nome della classe:

```
int risultato = Matematica.somma(3, 5);
```

il metodo main

- il metodo **`static main`** è il il primo metodo dell'applicazione che viene eseguito
- questo metodo è invocato automaticamente quando si esegue una classe
- se si tenta di eseguire una classe priva di un metodo main si ottiene un errore
- main è il metodo all'interno del quale in genere si istanziano i primi oggetti che si fanno interagire tra loro

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

- in alcuni casi è utile avere un metodo che possa essere chiamato sia con parametri, sia senza, oppure con numero e tipo di parametri differenti
- nel caso di due o più metodi con lo stesso nome ma con parametri differenti si parla di **overloading**

```
public int somma(int addendo1, int addendo2) {
  return addendo1 + addendo2;
}
public float somma(float addendo1, float addendo2) {
  return addendo1 + addendo2;
}
```

- l'overloading consente di **sovraccaricare** il metodo con più di un significato

- il **costruttore** è un metodo particolare che viene invocato alla creazione dell'oggetto e che contiene tutte le istruzioni da eseguire per la sua inizializzazione

- in Java i metodi costruttore:
  - devono avere lo stesso nome della classe a cui appartengono
  - possono anche essere vuoti o non essere definiti. In questi casi, sull'oggetto creato non sarà effettuata alcuna operazione di inizializzazione
    - viene utilizzato il costruttore di default della JVM
  - possono avere parametri di input che serviranno per effettuare le operazioni di inizializzazione alla creazione dell'oggetto
  - possono esistere più costruttori con lo stesso nome, ma con numero e tipo di parametri differenti
  - è possibile creare un oggetto invocando uno dei costruttori

```
public class Bicchiere {

  public String forma;
  public String materiale;
  public boolean pieno;

  public Bicchiere() {
    pieno = false;
  }

  public Bicchiere(String nuovaForma, String nuovoMateriale)
{
    forma = nuovaForma;
    materiale = nuovoMateriale;
    pieno = false;
  }
  …
}
```

- **public**
  - consente a qualunque classe o oggetto di qualsiasi tipo di avere accesso all'attributo o al metodo a cui è applicato
- **protected**
  - consente l'accesso solo alle classi e agli oggetti il cui tipo è una sottoclasse di quella in cui è utilizzato
- **private**
  - consente l'accesso solo agli oggetti della classe in cui è utilizzato
- visibilità di default (senza alcun modificatore)
  - consente a tutte le classi appartenenti allo stesso package di accedere all'attributo o al metodo

Tabella 1 I differenti livelli di visibilità				
Livello di visibilità	**Tutte le classi**	**Package**	**Sottoclassi**	**Classe**
public	*	*	*	*
default		*		*
protected			*	*
private				*

- un esempio:

```
int a, b;
a = 3
b = a;
a = 5;
System.out.print(b);
```

- viene visualizzato il valore 3
- le variabili di un tipo base contengono un valore

- un altro esempio:

```
Bicchiere biccUno, biccDue;
biccUno = new Bicchiere(“calice”, “vetro”);
biccDue = biccUno;
biccUno.forma = “coppa”;
System.out.print(bicc2.forma);
```

- viene visualizzato “coppa”
- gli oggetti sono un **riferimento** ad una zona di memoria
- in questo caso **`biccUno`** e **`biccDue`** sono due riferimenti allo stesso oggetto

incapsulamento

- l'**incapsulamento (***information hiding)* è un concetto fondamentale dell'ingegneria del software
- questo principio prevede che si possa **accedere** alle informazioni di un oggetto **unicamente attraverso i suoi metodi**
- in Java l'incapsulamento si avvale dei **modificatori di visibilità** per nascondere gli attributi di un oggetto
- mettere in atto questa tecnica significa **non avere** mai **attributi** di un oggetto di tipo **public**, salvo eccezioni particolari per costanti o attributi di classe da gestire in base al caso specifico

- per accedere dall'esterno agli attributi, si inseriscono metodi **public** che possono essere chiamati da chiunque per impostare o richiedere il valore dell'attributo
- i metodi hanno di solito un nome particolare:
  - set (seguito dal nome dell'attributo) per modificarne (settare) il valore
  - get (seguito dal nome dell'attributo) per recuperare (get) il valore

```
private int codice;

public void setCodice(int nuovoCodice) {
   codice = nuovoCodice;
}

public int getCodice() {
   return codice;
}
```

incapsulamento: perché?

- potrebbe sembrare che non vi sia alcuna differenza rispetto ad accedere direttamente agli attributi
- sembra che questa tecnica serva solo a rendere più complessa la loro gestione
- le motivazioni sono:
  - controllo sulle operazioni effettuate sugli attributi, limitando l'utilizzo improprio (sicurezza)
  - possibilità di nascondere il modo in cui i dati sono memorizzati negli attributi (hiding)

controllo sui valori inseriti

```
public void setCodice(int codice) throws
   CodiceErratoException {
  if( (codice >= 100) && (codice <= 1000000) ) {
    this.codice = codice;
  } else {
    throw new CodiceErratoException();
  }
}
```

convenzioni
nomi delle classi

- il **nome** di **classe** dovrebbe iniziare sempre con la lettera **maiuscola**
- nel caso di nomi composti, si utilizzano le maiuscole per le iniziali di ogni parola che compone il nome
- nel caso di acronimi, il nome sarà interamente maiuscolo
  - `Persona`
  - `IndirizzoDiCasa`
  - `HTTPMessage`

- i **nomi** di **attributi** e **metodi** dovrebbero iniziare con lettera **minuscola** *(nomi composti camelCase)*
  - `nome`
  - `codiceFiscale`
  - `hTTPHeader`
  - `esegui()`
  - `scriviSuFile()`
- le **costanti** devono essere scritte in maiuscolo.
  i nomi composti devono avere le parti del nome separate da _
  - `PI_GRECO`
  - `RADICE_QUADRATA_DI DUE`

- ogni attributo è definito **private**
- ogni attributo ha una coppia di metodi **public** per impostarne e richiederne il valore.
- il nome di questi metodi è composto dai prefissi **get** e **set**, a cui va aggiunto il nome dell'attributo.

```
private String indirizzo;
public void setIndirizzo(String indirizzo) {
  this.indirizzo = indirizzo;
}
public String getIndirizzo() {
  return indirizzo;
}
```

- per comunicare gli oggetti utilizzerano i metodi, scambiandosi messaggi l'uno con l'altro
- quando un oggetto invoca un metodo di un altro, quest'ultimo reagisce eseguendo il metodo opportuno
- l'invocazione dei metodi può richiedere parametri di qualsiasi tipo, compresi quindi oggetti
- un oggetto può passarne un altro attraverso un metodo, o addirittura potrà passare se stesso
- un messaggio ha la seguente sintassi:

```
<nomeOggetto>.<nomeMetodo>(<parametri>)
```

- in alcuni casi un oggetto ha la necessità di riferirsi a se stesso, per esempio all'interno di un suo metodo o nel metodo costruttore
- questo può accadere perché l'oggetto deve riferirsi a un suo membro (attributo o metodo) oppure deve passare se stesso come parametro durante l'invocazione di un metodo di un altro oggetto
- in Java, per effettuare questa operazione, un oggetto può utilizzare la parola chiave ***this***

```
public Bicchiere(String forma, String materiale) {
  this.forma = forma;
  this.materiale = materiale;
  pieno = false;
}
```