



UNIVERSITÀ  
degli STUDI  
di CATANIA

# Introduzione alla programmazione agli oggetti

Corso di programmazione I AA 2019/20

Corso di Laurea Triennale in Informatica

---

Prof. Giovanni Maria Farinella

Web: <http://www.dmi.unict.it/farinella>

Email: [gfarinella@dmi.unict.it](mailto:gfarinella@dmi.unict.it)

Dipartimento di Matematica e Informatica

Gli oggetti sono tipi “non primitivi”.

Non sono definiti dal linguaggio stesso, ma sono definiti dall’utente (*user-defined*).

La libreria standard del C++ permette di instanziare molti oggetti “predefiniti”. ES: string.

<https://en.cppreference.com/w/cpp/header>

## Oggetti

---

Entità software che simulano gli oggetti del mondo reale.

Dotati di una loro propria “individualità”.

Capaci di interagire per **scambio di messaggi**.

Un oggetto si compone di:



- **Stato**, ovvero proprietà che caratterizzano l'oggetto.
- **Comportamento**, che rappresenta l'insieme delle operazioni che è capace di eseguire un oggetto:
  - Cambiare il proprio stato.
  - Inviare messaggi ad altri oggetti

# Oggetti

Rappresentare/modellare una automobile mediante oggetti.

STATO:

- targa
- colore
- accesa 
- velocità 
- livello di benzina
- ...

COMPORTAMENTO:

- Accensione
- Accellera/decella
- Leggere la targa
- Leggere giri del motore
- ...

## STATO

- Insieme di **attributi**
- Un attributo in generale rappresenta una proprietà definita in modo astratto.
- Anche detto “campo” o “variabile” perché spesso **mappato su una variabile**

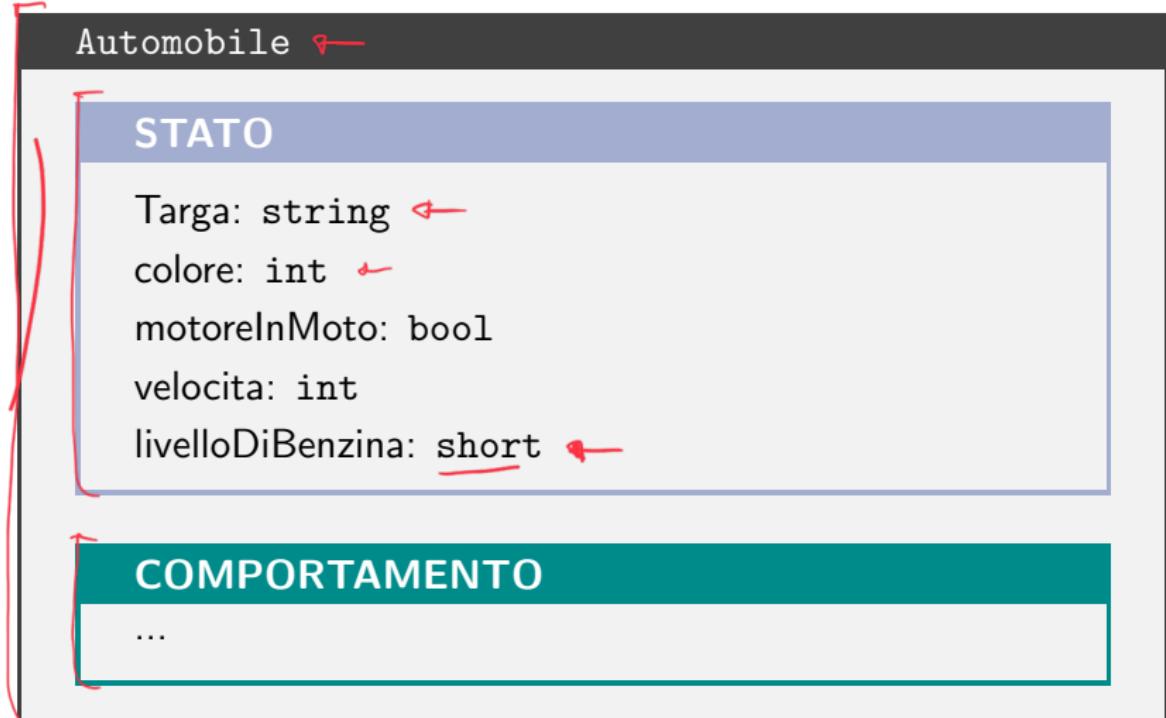
## COMPORTAMENTO

Modellato o descritto da un insieme di **metodi** anche detti **funzioni membro**.

## ESEMPIO

STATO per l'oggetto automobile.

# Oggetti



# Oggetti

OGGETTO di tipo Automobile

## STATO

targa = JK 1234 ↗

colore = 112233 ↗

motoreInMoto = false ↗

velocita = 0 km/h ↗

livelloDiBenzina = 28 lt. ↗

NB: TIPO automobile vs OGGETTO automobile.

---

---

# Oggetti

a1:Automobile

## STATO

targa = JK 1234  
colore = 112233  
motoreInMoto = false  
velocita = 0 km/h  
livelloDiBenzina = 28

a2:Automobile

## STATO

targa = JK 1234  
colore = 112233  
motoreInMoto = false  
velocita = 0 km/h  
livelloDiBenzina = 28  
lt.

### **Principio di identità.**

Ogni oggetto di un determinato tipo ha una propria identità.

Di conseguenza esso è distinguibile da tutti gli oggetti dello stesso tipo (es: automobile).

### **Principio di conservazione dello stato.**

Durante l'esecuzione del programma gli oggetti mantengono le informazioni al proprio interno per un tempo **indefinito**.

## Ciclo di vita di un oggetto.

Un oggetto, nel corso della elaborazione (esecuzione del programma) è soggetto alle seguenti fasi:

- **creazione** (stato iniziale)
- **utilizzo** (cambiamenti di stato e attività)
- **distruzione** (liberazione di memoria)

## Oggetti

**Osservazione.** Due oggetti dello stesso tipo possono avere lo stesso stato?

- Certamente!

**Osservazione.** Due oggetti dello stesso tipo possono avere comportamenti differenti?

- Anche (polimorfismo)

## Classi vs oggetti

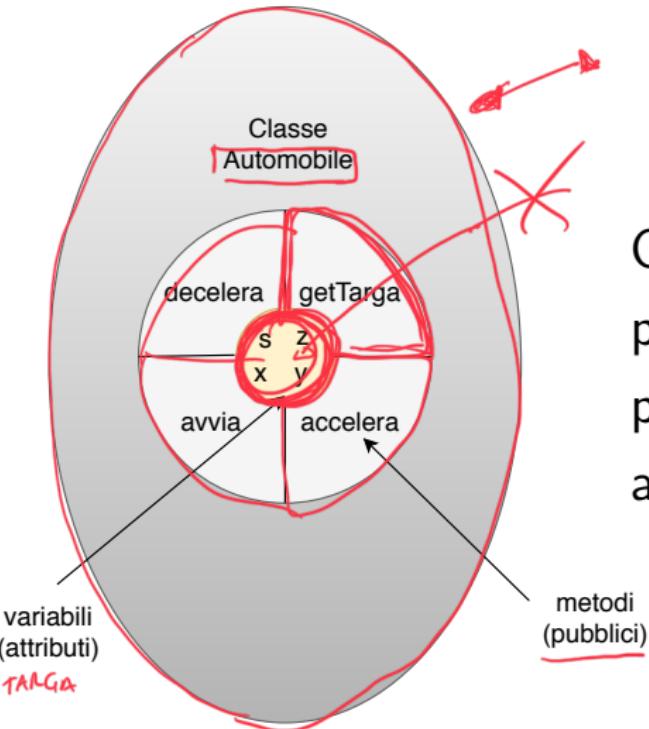
---

Una **classe** costituisce la descrizione “formale” (in codice) del tipo che il programmatore vuole definire.

In altre parole la classe rappresenta la descrizione di come devono essere **costruiti gli oggetti istanza** di essa.

La classe descrive la **struttura** degli oggetti istanza di essa.

# Classi vs oggetti



Oggetto espone i metodi pubblici, che operano prevalentemente sugli attributi.

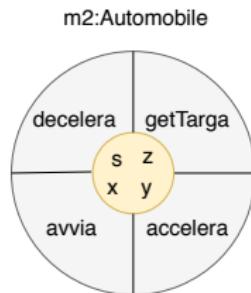
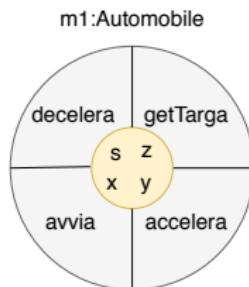
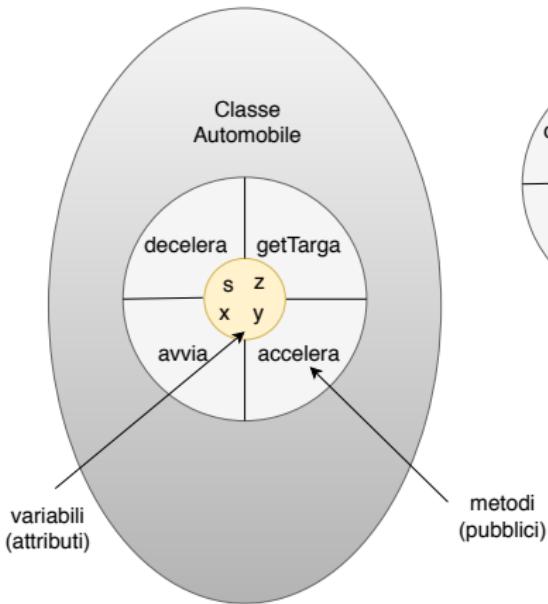
## Classi vs oggetti

Le classi vengono **progettate** in una delle fasi finali della realizzazione del software:

- 
1. Fase di **Analisi** (Object Oriented analysis):
    - Analisi del dominio della applicazione.
    - Analisi dei requisiti.
  2. Fase di progettazione (o **modellazione** o **design**) delle classi (Object Oriented Design).
  3. Fase di **implementazione** (Object Oriented Programming). L'implementazione è costituita dalla codifica delle classi utilizzando un linguaggio di programmazione ad oggetti (ES: C++).

# Classi vs oggetti

Un oggetto di dice **istanza** di una classe, in altre parole un **esemplare** di quella determinata classe.



### Remark.

Un oggetto è un contenitore di:

- **variabili (o attributi)** che descrivono lo stato dell'oggetto o memorizzano dati rilevanti per l'oggetto stesso;
- **metodi** per **elaborare le variabili**
  - ... e/o per **estrarre altre informazioni**
  - ... e/o per **cambiare il valore delle variabili** o per **mostrare all'esterno** dell'oggetto il loro valore.

## Classi vs oggetti

Una **classe** costituisce una **determinata rappresentazione delle caratteristiche salienti** di una entità del mondo reale.

Un oggetto è una **istanza** della classe: la sua struttura ed il suo comportamento sono **conformi** alla descrizione della classe.

Tuttavia ogni oggetto ha la sua **identità**, e generalmente un proprio valore per le variabili di stato o attributi.

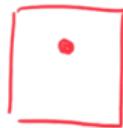
## Classi vs oggetti

In C++ la parola riservata class permette di descrivere le classi.

### Il progettista

- in fase di design deve modellare stato e comportamento degli oggetti di quella classe.
- in fase di implementazione
  - deve mappare il modello dello stato in un insieme di attributi o variabili;
  - deve mappare il modello del comportamento in un insieme di metodi;

# Messaggi



AUTOMOBILE



PERSO~~N~~A

Gli oggetti interagiscono mediante **scambio di messaggi**.

Lo scambio di messaggi avviene tramite la **invocazione dei metodi**.

La invocazione di un metodo può avere finalità differenti:

-  **Prelevare un'informazione** (es: parte dello stato dell'oggetto stesso)
-  Causare un **cambiamento di stato** dell'oggetto stesso.
-  **Avviare un'attività** per il quale l'oggetto è stato progettato.

# Messaggi

AUTOMOBILE mieAuto;

**Sintassi** invocazione del generico metodo.

<obj>.<metodo> ([ p1 ] [ , p2 ] ... );

obj denota l'oggetto destinatario del messaggio.

metodo denota il nome del metodo

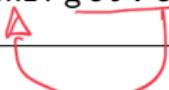
[p1] [, p2] ... etc denota la lista di eventuali parametri che viene per la invocazione del metodo.

## Messaggi

All'invocazione di un metodo segue:

- Eventuale **invio di informazioni** all'oggetto chiamante (valori di ritorno);
- eventuale **cambiamento di stato**;
- avvio di una **certa attività**.

```
1 Automobile m1;
2 m1.setMotore("elettrico"); // cambiamento di stato
3 m1.avviaTergicristallo(); // attività
4 m1.accendiMotore(); // cambiamento di stato
5 m1.accelera(); // attività
6 m1.getVelocita(); // invio di informazioni al chiamante
```



Un metodo è una **funzione associata** ad una ben determinata classe (incapsulamento).

Un metodo rappresenta così parte della implementazione del **comportamento** dello oggetto.

*obj.metodo()*

All'atto della invocazione di un metodo:

- il flusso di controllo **prosegue con** l'esecuzione del **codice di tale metodo**.

- **alla fine della esecuzione** del metodo, ovvero
  - in corrispondenza di una istruzione return
  - oppure dopo l'ultima istruzione del corpo  
del metodo
- ... il flusso di controllo **prosegue con**  
**l'esecuzione della riga di codice successiva**  
alla chiamata;

## Messaggi

Tipi di messaggio: → [S : Myjet -- ()]

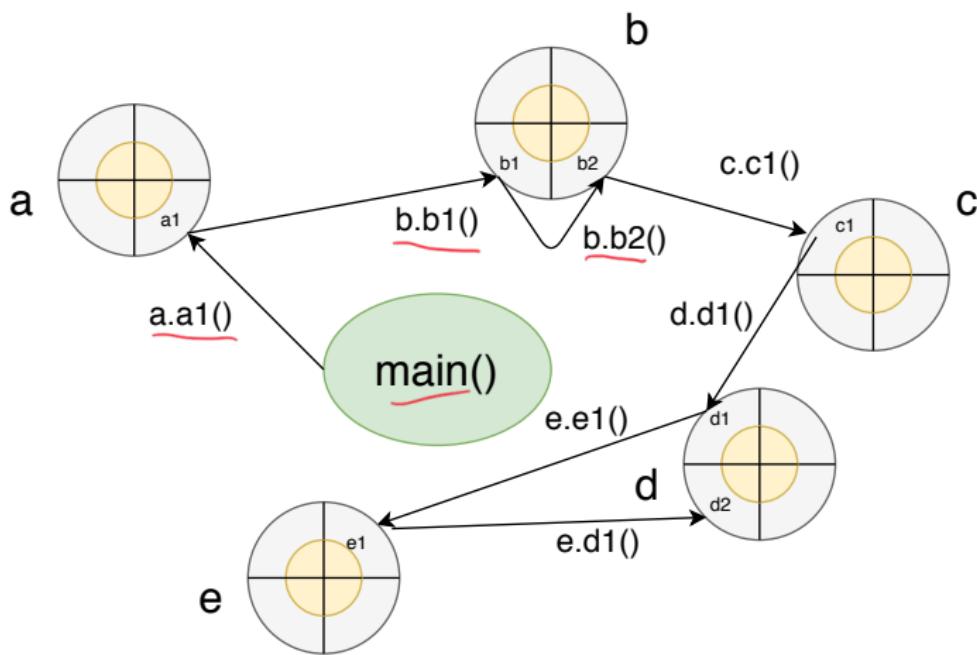
- **informativo:** setTarga(). Informa l'oggetto che qualcosa è cambiato e che deve aggiornare il suo stato;
- **interrogativo:** getTarga(). Chiede informazioni in merito allo stato dell'oggetto;
- **imperativo:** avviaMotore(). Chiede all'oggetto di avviare una o più attività;

Programmazione strutturata/procedurale: una  
**sequenza di invocazioni di funzioni...**

VS

Programma ad oggetti: l'esecuzione di un  
programma ad oggetti flusso di **messaggi** tra  
oggetti, ovvero una **sequenza di invocazione di  
metodi.**

# Messaggi



## Messaggi

```
1 Dado d;  
2 d.effettuaLancio():void; //avvia attivita'  
3 d.getUltimoLancio(); //interrogativo
```

Metodo `effettuaLancio()` non restituisce alcun dato (`void`), ma si occupa di simulare il lancio di un dado.

- ...quindi Imperativo (e il dado cambia stato!)

## Messaggi

```
1 Dado d;  
2 d.effettuaLancio(); //avvia attivita'  
3 d.getUltimoLancio(); //interrogativo
```

Metodo `getUltimoLancio()` interroga l'oggetto sul suo **stato** (la faccia superiore del dado).

## Esempio svolto

A21\_00.cpp, dado.cpp, dado.h

```
$g++ A21_00.cpp dado.cpp
```

# Messaggi

```
1 #define SEQ 3
2 #define MAX_LANCI 10000000
3 #define LANCIO_BUONO 6
4 int i=0, c=0;
5 while ( i++<MAX_LANCI && c<SEQ)
6 {
7     d.effettuaLancio();
8     if (d.getUltimoLancio() == LANCIO_BUONO)
9         c++;
10    else
11        c=0;
12    }
13 cout << (c==SEQ ? "Hai vinto!" \
14      : "Hai perso!") << endl;
```

Ulteriore esempio è costituito dalla classe Moneta.

moneta.cpp, moneta.h

## Esempi svolti

A21\_01.cpp (*lancio coppia di dadi*)

A21\_02.cpp (*lancio della moneta*)

A21\_03.cpp (*lancio di due monete*)

*Anno M1; 3*

Il metodo **costruttore** serve ad **inizializzare lo stato** dell'oggetto appena creato.

La **chiamata** al costruttore è **automatica**, avviene contestualmente alla creazione dello oggetto.

Il compilatore “forza” la chiamata automatica al costruttore.

In questo modo creazione dell'oggetto ed inizializzazione dello stato sono **inseparabili**.

## Classe Dado

### STATO

valoreUltimoLancio: short

### COMPORTAMENTO

Dado();

effettuaLancio();

getUltimoLancio(): bool;

## Classe Moneta

### STATO

→ ultimaFaccia: char;

### COMPORTAMENTO

Moneta();

effettuaLancio():void;

testa(): bool;

croce(): bool;

getFaccia(): char;

## Classe Serbatoio

### STATO

```
capacita: float;  
quantita: float;
```

### COMPORTAMENTO

```
Serbatoio (capacita:float, quantita:float);  
getCapacita():float;  
deposita (float quantita): void;  
preleva (float quantita): float;
```

# Metodi e parametri

## Esempio svolto

A21\_04.cpp, serbatoio.cpp, serbatoio.h

```
$g++ A21_04.cpp serbatoio.cpp -o A21_04
```

- La creazione di un serbatoio da 10 litri contenente 7 litri di benzina (STATO iniziale).
- Prelievo di 3 litri di benzina.
- Svuotamento del serbatoio.
- Deposito di 8 litri di benzina.

**Quale lo stato Finale?**

# Metodi e parametri

Un esempio più complesso. Interazione tra due serbatoi.

## Esempio svolto

A21\_05.cpp, serbatoio.cpp, serbatoio.h

```
$g++ A21_05.cpp serbatoio.cpp -o A21_05
```

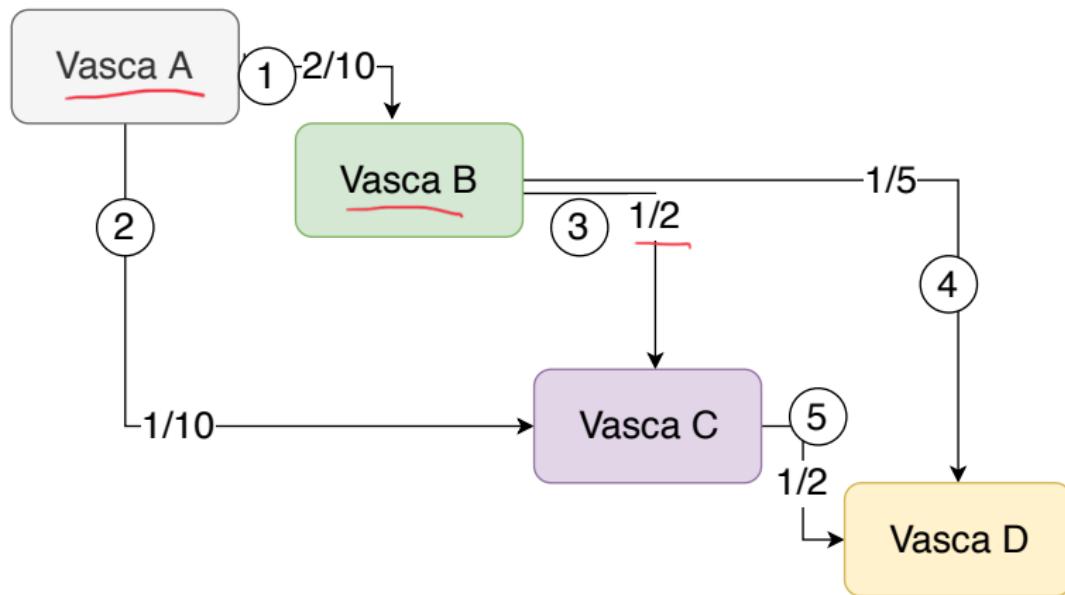
Serbatoio A e serbatoio B.

**Riversamento** del contenuto di un serbatoio in un altro serbatoio.

```
1 Serbatoio a(10,7);
2 Serbatoio b(20);
3 b.deposita(a.svuotaTutto());
```

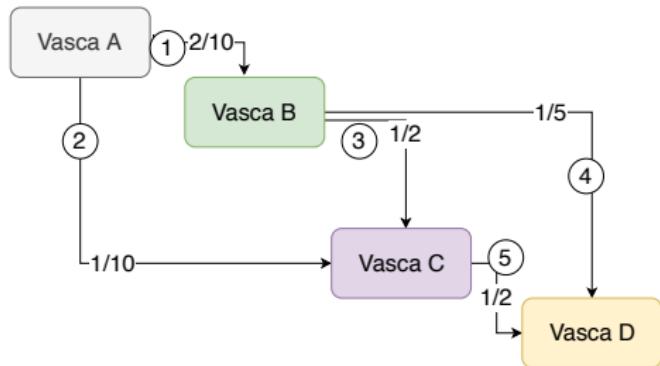
## Metodi e parametri

Ancora un esempio: quattro serbatoi (A,B,C,D) collegati nel seguente modo ([A21\\_06.cpp](#)).



# Metodi e parametri

Ogni due ore avverranno le seguenti transizioni/operazioni:



- L'elettrovalvola **1** fa passare  $1/5$  del contenuto di A in B;
- L'elettrovalvola **2** fa passare  $1/10$  del contenuto di A in C;
- L'elettrovalvola **3** fa passare  $1/2$  del contenuto di B in C;
- L'elettrovalvola **4** fa passare  $1/5$  del contenuto di B in D.
- L'elettrovalvola **5** fa passare  $1/2$  del contenuto di C in D.

# Metodi e parametri

**Parametri stringa.** Esempio: A21\_07.cpp, conto\_corrente.cpp

ContoBancario

## STATO

...

## COMPORTAMENTO

ContoCorrente (nominativo:string, numero:long, primoDeposito:double);

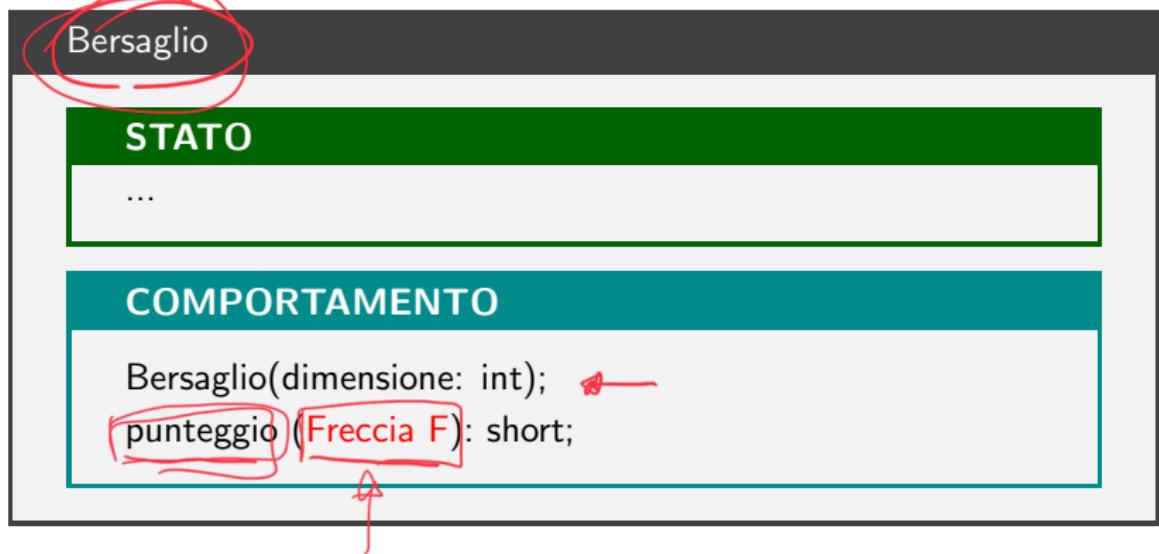
setNominativo (nominativo:string):void

getNominativo():string

...

# Metodi e parametri

Oggetti come parametri A21\_08.cpp, bersaglio.cpp



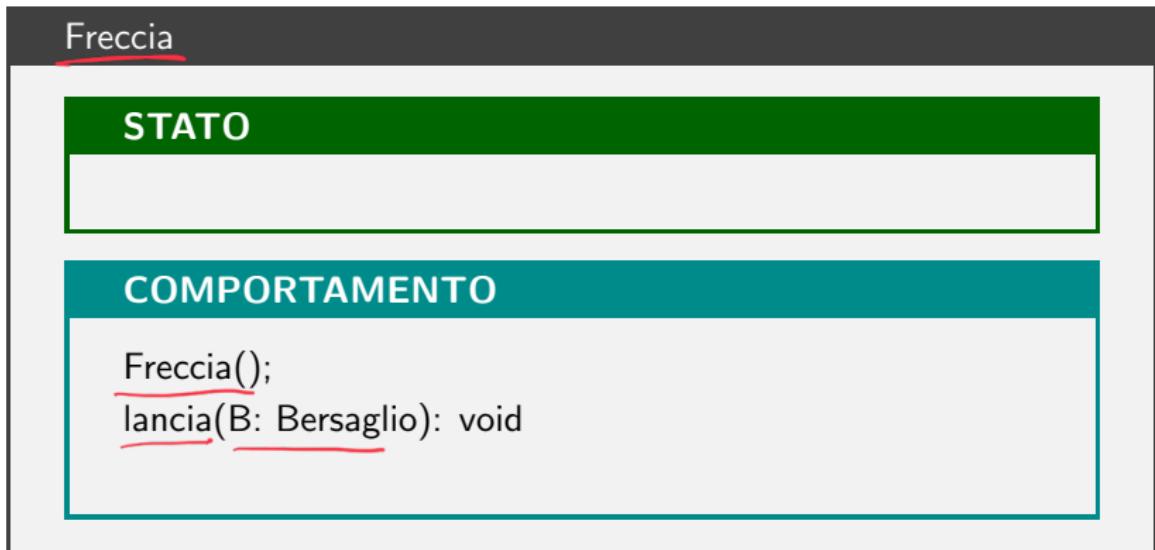
## Metodi e parametri

---

L'oggetto Bersaglio **invia un messaggio** all'oggetto freccia (metodo punteggio()) passato come argomento per ottenere le sue **coordinate** e quindi calcolare il punteggio.

# Metodi e parametri

Oggetti come parametri `A21_08.cpp, bersaglio.cpp`



L'oggetto `Freccia` **invia un messaggio all'oggetto** `Bersaglio`.

# Metodi e parametri

## Lancio della freccia..

Il metodo `lancia()` chiede al bersaglio la misura del lato del quadrato entro il quale si trova il bersaglio ed effettuare un opportuno lancio.

- 1   Bersaglio   b(10);
- 2   Freccia   f;
- 3   f.lancia(b);
- 4   totale += b.punteggio(f);

Perchè non `f.punteggio(b)`?

...il punteggio viene calcolato in base al raggio della circonferenza interna del bersaglio che è un attributo non visibile (private) del bersaglio.

# Metodi e parametri

Metodi che restituiscono oggetti **A21\_09.cpp, frazione.cpp**

Bersaglio

## STATO

...

## COMPORTAMENTO

equals(Frazione f): bool;  
piu(Frazione f):Frazione;  
meno(Frazione f):Frazione;  
per(Frazione f):Frazione;

...

### Esempio: uso della classe Frazione

```
1 Frazione x;  
2 Frazione f1(6,5);  
3 Frazione f2(3,4);  
4  
5 x = f1.meno(f2);
```

## Homework H21.1: La slot machine

### Classe RuotaFortunata

ruota.cpp, ruota.h

A21\_12.cpp

```
1 RuotaFortunata();
2 RuotaFortunata(int num, int min, int max);
3 RuotaFortunata(int num, int *v);
```

Scrivere un programma (non una classe) che simuli una slot Machine che abbia almeno tre ruote usando la classe RuotaFortunata.

## Homework H21.1: La slot machine

---

### Specifiche:

- ad ogni ruota va associata una **posizione** nella slot machine ovvero un numero maggiore o uguale ad 1;
- ogni ruota può contenere uno o più elementi **jolly**;
- il numero di elementi jolly delle differenti ruote può differire, ma **l'insieme dei rimanenti elementi** deve essere **identico** per tutte le ruote. ES:
  - R1: {1,2,3,4,5,  $j_1, j_2$ };
  - R2: {1,2,3,4,5,  $j_1$ };
  - R3: {1,2,3,4,5,  $j_1, j_2, j_3$ };

## Homework H21.1: La slot machine

**Combinazioni** con le quali si vince:

- 10 punti:  $R_1 : i, R_2 : i + 1, \dots, R_n : i + (n - 1)$ . Ovvero le ruote mostrano un insieme di numeri consecutivi, dalla prima all'ultima ruota, dove  $n$  è il numero delle ruote;
- 100 punti: le ruote mostrano una sequenza di elementi jolly  $j_i$ , in cui non tutti gli elementi jolly sono uguali;
- 500 punti: le ruote mostrano una sequenza di  $n$  simboli con lo stesso elemento jolly;

Il programma prende in input il numero  $N$  di "lanci" o tentativi del giocatore, per dare in output il punteggio totale dopo gli  $N$  lanci.

# FINE