# Il Metodo Costruttore

Specifica le operazioni di inizializzazione (attributi, etc.) che vogliamo vengano eseguite su ogni oggetto della classe appena viene creato

Tale metodo ha

- Lo **stesso nome** della classe
- Tipo **non** specificato

Non possono esistere attributi non inizializzati

- Gli attributi vengono inizializzati comunque con valori di **default**

Se non viene dichiarato un costruttore, ne viene creato uno di default vuoto e senza parametri

Spesso si usa l'**overloading** definendo diversi costruttori

La distruzione di oggetti (garbage-collection) non è a carico del programmatore

---

### Il costrutto new

- Crea una nuova istanza della classe specificata, allocandone la memoria
- Restituisce il riferimento all'oggetto creato
- Chiama il costruttore del nuovo oggetto

```
Automobile a = new Automobile ();
Motorcycle m = new Motorcycle ();
String s = new String ("ABC");
```

### Per "gestire" una classe occorre

- Accedere ai metodi della classe
- Accedere agli attributi della classe

---

## Messaggi

- L'invio di un messaggio provoca l'esecuzione del metodo

### Inviare un messaggio ad un oggetto

- Usare la notazione "puntata" oggetto.messaggio(parametri)
- Sintassi analoga alla chiamata di funzioni in altri linguaggi
- I metodi definiscono l'implementazione delle operazioni
- I messaggi che un oggetto può accettare coincidono con i nomi dei metodi
- p.es mettiInMoto(), vernicia(), etc.

- Spesso i messaggi includono uno o più parametri
- .vernicia("Rosso")

### Esempi

```
Automobile a = new Automobile();
a.mettiInMoto();
a.vernicia("Blu");
```

---

### All'interno della classe

- I metodi che devono inviare messaggi allo stesso oggetto cui appartengono
- non devono obbligatoriamente utilizzare la notazione puntata: è sottinteso il riferimento

```
public class Libro {
    int nPagine;
    public void leggiPagina (int nPagina) {...}
    public void leggiTutto () {
        for (int i=0; i<nPagine; i++)
            leggiPagina (i);
    }
}
```

---

## Attributi

- Stessa notazione "puntata" dei messaggi `oggetto.attributo`
- Il riferimento viene usato come una qualunque variabile

```
Automobile a=new Automobile();
a.colore = "Blu";
boolean x = a.accesa;
```

I metodi che fanno riferimento ad attributi dello stesso oggetto possono tralasciare il rif-oggetto

```
public class Automobile {
    String colore;
    void vernicia(){
        colore = "Verde";// colore si riferisce all'oggetto corrente
    }
}
```

---

- Esempio (messaggi e attributi)

```
public class Automobile {
    String colore;
    public void vernicia () {
        colore = "bianco";
    }
    public void vernicia (String nuovoCol) {
        colore = nuovoCol;
    }
}

Automobile a1, a2;
a1 = new Automobile ();
a1.vernicia ("verde");
a2 = new Automobile ();
```

---

Esempio (costruttori con overloading)

```
Class Finestra {
    String titolo;
    String colore;
    // Finestra senza titolo nè colore
    Finestra () {
    ...
    }
    // Finestra con titolo senza colore
    Finestra (String t) {
    ...
    titolo = t;
    }
    // Finestra con titolo e colore
    Finestra (String t, String c) {
    ...
    titolo = t; colore = c;
    }
}
```

---

## Operatore `this` (Puntatore Auto-referenziante)

La parola riservata `this` e' utilizzata quale puntatore auto-referenziante

- `this` riferisce l'oggetto (e.g., classe) corrente

Utilizzato per:

- Referenziare la classe appena istanziata
- Evitare il conflitto tra nomi

```
class Automobile{
String colore;
...
...
void vernicia (String colore) {
this.colore = colore;
}
}
. . .
Automobile a2, a1 = new Automobile;
a1.vernicia("bianco"); // a1 == this
a2.vernicia("rosso");
// this == a2
```

---


```
class Automobile{
String colore;
...
...
void vernicia (String colore) {
this.colore = colore;
}
}
. . .
Automobile a2, a1 = new Automobile;
a1.vernicia("bianco"); // a1 == this
a2.vernicia("rosso");
// this == a2
```