OOP

Attributi e metodi di classe

ATTRIBUTI DI ISTANZA

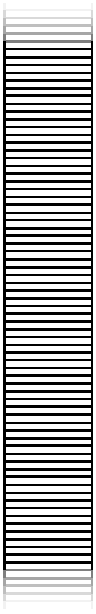
Consideriamo una classe qualsiasi

Sappiamo che ogni volta che istanziamo un oggetto si riserva in memoria lo spazio per gli attributi

```
// dichiaro e istanzio una data
Data d1;
d1 = new Data(19,4,1976);
```

Data	
- int	giorno
- int	mese
- int	anno
...	

d1: Data	
giorno	19
mese	4
anno	1976



ATTRIBUTI DI ISTANZA

Consideriamo una classe qualsiasi

Sappiamo che ogni volta che istanziamo un oggetto si riserva in memoria lo spazio per gli attributi

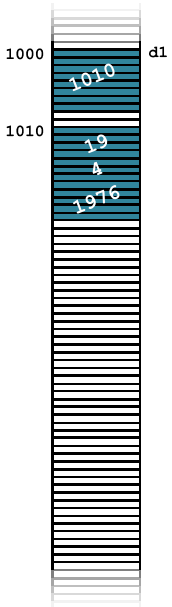
```
// dichiaro e istanzio una data
Data d1;
d1 = new Data(19,4,1976);

// dichiaro e istanzio una data
Data d2;
d2 = new Data(7,2,1952);
```

Data	
- int	giorno
- int	mese
- int	anno
...	

d1: Data	
giorno	19
mese	4
anno	1976

d2: Data	
giorno	7
mese	2
anno	1952



ALESSANDRO URSOMANDO

ATTRIBUTI DI ISTANZA

Consideriamo una classe qualsiasi

Sappiamo che ogni volta che istanziamo un oggetto si riserva in memoria lo spazio per gli attributi

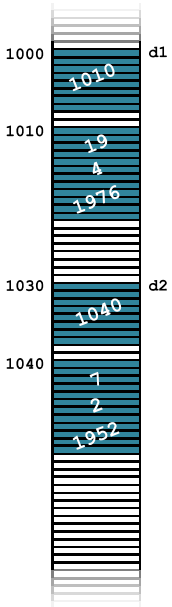
```
// dichiaro e istanzio una data
Data d1;
d1 = new Data(19,4,1976);

// dichiaro e istanzio una data
Data d2;
d1 = new Data(7,2,1952);
```

Data	
- int	giorno
- int	mese
- int	anno
...	

d1: Data	
giorno	19
mese	4
anno	1976

d2: Data	
giorno	7
mese	2
anno	1952



Ecco perché questo tipo di attributi si chiamano anche **attributi di istanza**!
Perché ogni istanza ha le sue!

ALESSANDRO URSOMANDO

ATTRIBUTI DI CLASSE

In contrapposizione a quanto appena detto presentiamo gli **attributi di classe**.

Per definizione gli attributi di classe (anche detti **attributi statici**) sono quelli che non sono replicati in tutte le istanze ma che si conservano altrove una sola volta e valgono per tutti gli oggetti della classe

Pensiamo ai vettori costanti **NOMI_COLORI_DISPONIBILI** e **CODICI_COLORI_DISPONIBILI** della classe **Colore** del nostro esercizio 11

ATTRIBUTI DI CLASSE

Colore		
-	String	nome
-	String	hex
+	String[]	NOMI_COLORI_DISPONIBILI
+	String[]	CODICI_COLORI_DISPONIBILI
	...	

0	"nero"
1	"rosso"
2	"verde"
3	"blu"
4	"bianco"
5	"giallo"
6	"viola"

0	"000000"
1	"FF0000"
2	"00FF00"
3	"0000FF"
4	"FFFFFF"
5	"FFFF00"
6	"FF00FF"

```
Colore x = new Colore("rosso");
```

x: Colore	
nome	"rosso"
hex	"FF0000"
NOMI_COLORI_DISPONIBILI	↷
CODICI_COLORI_DISPONIBILI	↷

Riserviamo spazio in memoria per i due attributi di tipo stringa (**nome** ed **hex**) ma anche per i due vettori **NOMI_COLORI_DISPONIBILI** e **CODICI_COLORI_DISPONIBILI**

ATTRIBUTI DI CLASSE

Colore	
- String	nome
- String	hex
+ String[]	NOMI_COLORI_DISPONIBILI
+ String[]	CODICI_COLORI_DISPONIBILI
...	

0	"nero"
1	"rosso"
2	"verde"
3	"blu"
4	"bianco"
5	"giallo"
6	"viola"

0	"000000"
1	"FF0000"
2	"00FF00"
3	"0000FF"
4	"FFFFFF"
5	"FFFF00"
6	"FF00FF"

```
Colore x = new Colore("rosso");
Colore y = new Colore();
```

x: Colore

y: Colore	
nome	"nero"
hex	"000000"
NOMI_COLORI_DISPONIBILI	↷
CODICI_COLORI_DISPONIBILI	↷

La stessa quantità di spazio riservata poco fa alla variabile x deve essere riservata a y che ha la sua copia dei vettori **NOMI_COLORI_DISPONIBILI** e **CODICI_COLORI_DISPONIBILI**

ATTRIBUTI DI CLASSE

Colore	
- String	nome
- String	hex
+ String[]	NOMI_COLORI_DISPONIBILI
+ String[]	CODICI_COLORI_DISPONIBILI
...	

0	"nero"
1	"rosso"
2	"verde"
3	"blu"
4	"bianco"
5	"giallo"
6	"viola"

0	"000000"
1	"FF0000"
2	"00FF00"
3	"0000FF"
4	"FFFFFF"
5	"FFFF00"
6	"FF00FF"

```
Colore x = new Colore("rosso");
Colore y = new Colore();
Colore z = new Colore("blu");
```

x: Colore

y: Colore

z: Colore	
nome	"blu"
hex	"0000FF"
NOMI_COLORI_DISPONIBILI	↷
CODICI_COLORI_DISPONIBILI	↷

Anche z avrà la sua copia (inutile) di quei due vettori.

ATTRIBUTI DI CLASSE

Colore			
-		String	nome
-		String	hex
+	$	String[]	NOMI_COLORI_DISPONIBILI
+	$	String[]	CODICI_COLORI_DISPONIBILI
		...	

In casi come questo
invece che definire i due vettori come
attributi di istanza
ci conviene definirli come
attributi di classe (o **statici**)

Il sistema provvederà a istanziarne una sola copia
e ciascuna istanza farà riferimento sempre alla stessa copia della risorsa.

Nei diagrammi UML le variabili di classe
sono precedute dal **simbolo del dollaro** ($)

In Java un attributo di classe
ha lo specificatore **static**

```
static private String[] NOMI_COLORI_DISPONIBILI = {…};
static private String[] CODICI_COLORI_DISPONIBILI = {…};
```

ATTRIBUTI DI CLASSE

Colore			
-		String	nome
-		String	hex
+	$	String[]	NOMI_COLORI_DISPONIBILI
+	$	String[]	CODICI_COLORI_DISPONIBILI
		...	

Ma gli **attributi di classe**
sono sempre **pubblici** e **costanti**?

Assolutamente no!

Vediamo un esempio con un attributo **privato** e **variabile**.

ATTRIBUTI DI CLASSE

Supponiamo di avere tanti oggetti di una classe Poliziotto in esecuzione tutti insieme.
E supponiamo che questi poliziotti girino per la città
fino a quando non abbiano sgominato una banda di **dieci** malviventi.

E' chiaro che nella classe Poliziotto ci deve essere un attributo statico
(comune a tutte le istanze) che conti quanti elementi della banda sono stati catturati

Possiamo immaginare che
un'istanza di Poliziotto
passi il tempo ciclando su questa variabile

E che ad ogni ciclo
il poliziotto esamini una persona a caso in città,
comunicando agli altri quando ha trovato un malvivente

ATTRIBUTI DI CLASSE

```
public class TestPoliziotto {
    public static void main () {
        (new Poliziotto("Jack")).start();
        (new Poliziotto("Al")).start();
        (new Poliziotto("Mickey")).start();
        (new Poliziotto("Bob")).start();
        (new Poliziotto("Gary")).start();
}   }
```

```
public class Poliziotto extends Thread {
    static private int totCattivi;

    public Poliziotto(String nome) {
        super(nome);
    }

    public void run() {
        while (totCattivi < 10) {
            int x = (int)(Math.random()*5000000);
            if (x == 0) {
                System.out.println(getName() + " ha trovato un cattivo.");
                totCattivi++;
}   }   }   }
```

ATTRIBUTI DI CLASSE

```
public class TestPoliziotto {
    public static void main () {
        (new Poliziotto("Jack")).start();
        (new Poliziotto("Al")).start();
        (new Poliziotto("Mickey")).start();
        (new Poliziotto("Bob")).start();
        (new Poliziotto("Gary")).start();
}   }
```

```
public class Poliziotto extends Thread {
    static private int totCattivi;

    public Poliziotto(String nome) {
        super(nome);
    }

    public void run() {
        while (totCattivi < 10) {
            int x = (int)(Math.random()*5000000);
            if (x == 0) {
                System.out.println(getName() + " ha trovato un cattivo.");
                totCattivi++;
}   }   }   }
```

Una eventuale inizializzazione in questo punto sarebbe eseguita solo la prima volta.

METODI DI CLASSE

Allo stesso modo
durante la realizzazione di una classe
quando scriviamo un metodo che non è relativo alle istanze della classe
ma alla classe stessa
definiremo quel metodo come **metodo statico** (o **metodo di classe**)

Abbiamo già usato (inconsapevolemente) metodi statici

```
public class TestData {
    public static void main() {
        // ...
    }
}
```

Per esempio abbiamo
sempre dichiarato come statico
ogni nostro metodo **main**

Che infatti veniva invocato
senza istanziare nessuna classe

METODI DI CLASSE

Allo stesso modo
durante la realizzazione di una classe
quando scriviamo un metodo che non è relativo alle istanze della classe
ma alla classe stessa
definiremo quel metodo come **metodo statico** (o **metodo di classe**)

Abbiamo già usato (inconsapevolemente) metodi statici

```
public class TestData {
    public static void main() {
        // ...
    }
}
```

```
// ...
int m2 = (int)(Math.random()*12)+1;
// ...
```

Un altro esempio
è l'uso del metodo **random**
della classe **Math**

Lo abbiamo sempre invocato
senza istanziare la classe
ma semplicemente indicando il nome
della classe che lo contiene

METODI DI CLASSE

Ma **noi** quand'è che sentiremo l'esigenza di realizzare un metodo statico?

Data
...
...
+ boolean isDataValida(int, int, int)
...

```
public class TestData {
    public static void main() {
        // ...
    }
}
```

```
// ...
int m2 = (int)(Math.random()*12)+1;
// ...
```

```
{
   ...

   int g = casuale();
   int m = casuale();
   int a = casuale();

   Data d = new Data();

   if (d.isDataValida(g,m,a)) {
      System.out.println ("si");
   } else {
      System.out.println ("no");
   }

   ...
}
```

METODI DI CLASSE

Ma **noi** quand'è che sentiremo l'esigenza di realizzare un metodo statico?

Data
...
...
+ boolean isDataValida(int, int, int)
...

Non è bello dover
istanziare un oggetto di tipo Data
per poter verificare se una terna di valori
rappresenta una data valida

Data
...
...
+ $ boolean isDataValida(int, int, int)
...

```
{
   ...

   int g = casuale();
   int m = casuale();
   int a = casuale();

   Data d = new Data();

   if (d.dataValida(g,m,a)) {
      System.out.println ("si");
   } else {
      System.out.println ("no");
   }

   ...
}
```

```
   if (Data.dataValida(g,m,a)) {
```


RIASSUMENDO

Un membro **statico** è **relativo alla classe** e non alle sue istanze.

Per dichiarare statico un certo membro
bisogna usare lo specificatore **static**

Un membro statico in un diagramma UML
si riconosce mediante l'uso del **$**

L'accesso a un membro statico è regolato
con gli specificatori di visibilità **public** e **private** così come
per tutti gli altri membri

L'accesso a un membro statico dichiarato pubblico
dall'esterno della classe che lo contiene avviene indicando **il nome della classe**