OOP

Gli oggetti e la memoria

COSA ACCADE IN MEMORIA

```
Contatore c = new Contatore();
```

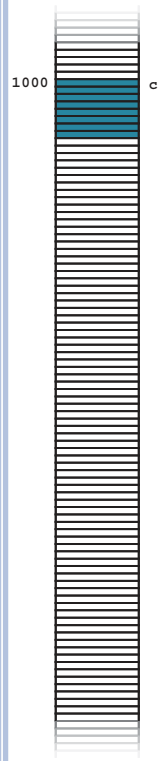
```
Contatore c;          //dichiaraz
c = new Contatore(); //istanziaz
```

Contatore	
int	valore
void	incrementa()
void	decrementa()
void	reset()

La **dichiarazione** di un oggetto riserva in memoria 8 byte che servono per mantenere l'indirizzo della cella di memoria che conserverà l'oggetto.

ALESSANDRO URSOMANDO

COSA ACCADE IN MEMORIA




```
Contatore c = new Contatore();
```

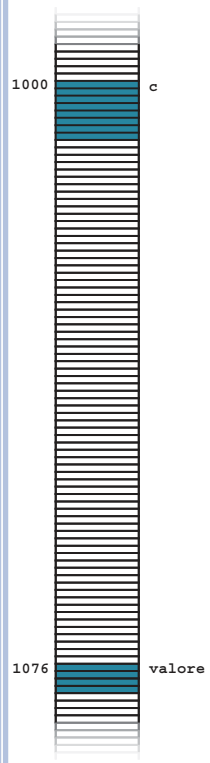
```
Contatore c;          //dichiaraz
c = new Contatore(); //istanziaz
```

Contatore	
int	valore
void	incrementa()
void	decrementa()
void	reset()

La **dichiarazione** di un oggetto riserva in memoria 8 byte che servono per mantenere l'indirizzo della cella di memoria che conserverà l'oggetto.

ALESSANDRO URSOMANDO

COSA ACCADE IN MEMORIA




```
Contatore c = new Contatore();
```

```
Contatore c;          //dichiaraz
c = new Contatore(); //istanziaz
```

Contatore	
int	valore
void	incrementa()
void	decrementa()
void	reset()

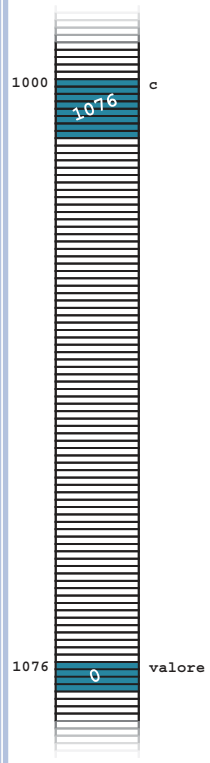
La **dichiarazione** di un oggetto riserva in memoria 8 byte che servono per mantenere l'indirizzo della cella di memoria che conserverà l'oggetto.

Con l'operatore **new** riservo in memoria lo spazio necessario a contenere tutti (e solo) gli attributi dell'oggetto.

L'operatore **new** restituisce l'indirizzo della cella di memoria allocata.

ALESSANDRO URSOMANDO

COSA ACCADE IN MEMORIA




```
Contatore c = new Contatore();
```

```
Contatore c;          //dichiaraz
c = new Contatore(); //istanziaz
```

Contatore	
int	valore
void	incrementa()
void	decrementa()
void	reset()

La **dichiarazione** di un oggetto riserva in memoria 8 byte che servono per mantenere l'indirizzo della cella di memoria che conserverà l'oggetto.

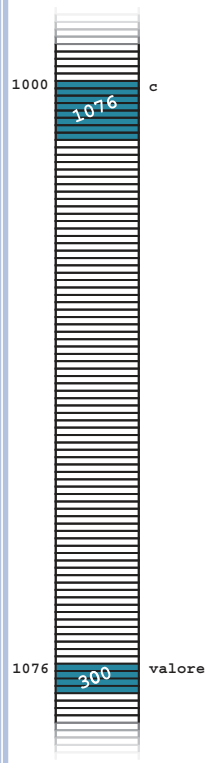
Con l'operatore **new** riservo in memoria lo spazio necessario a contenere tutti (e solo) gli attributi dell'oggetto.

L'operatore **new** restituisce l'indirizzo della cella di memoria allocata.

Infine, si scrive il valore **zero** in ogni attributo.

ALESSANDRO URSOMANDO

COSA ACCADE IN MEMORIA




```
Contatore c = new Contatore();
```

```
Contatore c;          //dichiaraz
c = new Contatore(); //istanziaz
```

```
c.valore = 300;
```

Contatore	
int	valore
void	incrementa()
void	decrementa()
void	reset()

La **dichiarazione** di un oggetto riserva in memoria 8 byte che servono per mantenere l'indirizzo della cella di memoria che conserverà l'oggetto.

Con l'operatore **new** riservo in memoria lo spazio necessario a contenere tutti (e solo) gli attributi dell'oggetto.

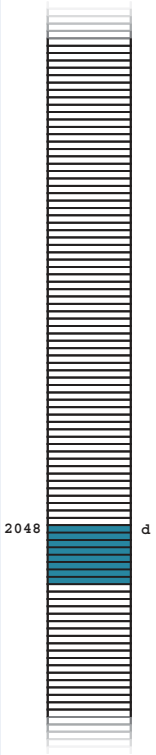
L'operatore **new** restituisce l'indirizzo della cella di memoria allocata.

Infine, si scrive il valore **zero** in ogni attributo.

Questa istruzione significa vai dove punta c e cerca «valore»

ALESSANDRO URSOMANDO

FACCIAMO UN ALTRO ESEMPIO

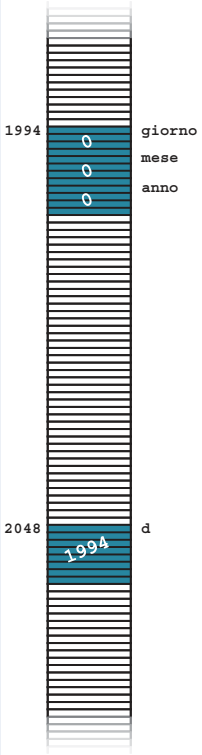



```
Data d;          //dichiarazione
```

Data	
int	giorno
int	mese
int	anno
...	

ALESSANDRO URSOMANDO

FACCIAMO UN ALTRO ESEMPIO

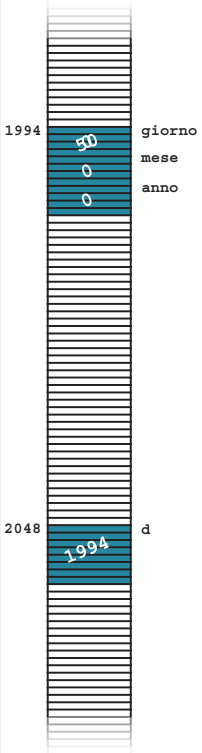



```
Data d;          //dichiarazione
d = new Data(); //istanziazione
```

Data	
int	giorno
int	mese
int	anno
...	

ALESSANDRO URSOMANDO

FACCIAMO UN ALTRO ESEMPIO

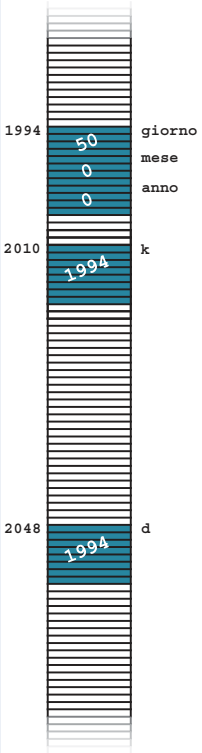



```
Data d;          //dichiarazione
d = new Data(); //istanziazione
d.giorno = 50;  //accesso
```

Data	
int	giorno
int	mese
int	anno
...	

ALESSANDRO URSOMANDO

FACCIAMO UN ALTRO ESEMPIO

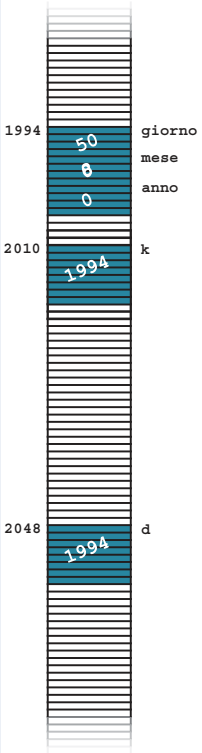



```
Data d;          //dichiarazione
d = new Data(); //istanziazione
d.giorno = 50;  //accesso
Data k = d;      //dichiarazione
```

Data	
int	giorno
int	mese
int	anno
...	

ALESSANDRO URSOMANDO

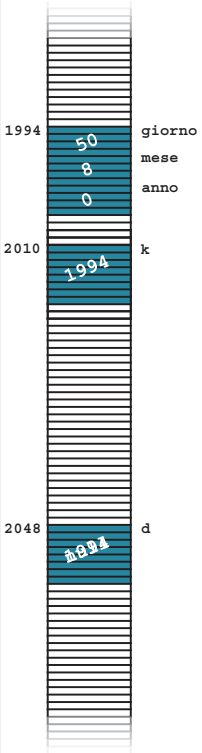
FACCIAMO UN ALTRO ESEMPIO




```
Data d;          //dichiarazione
d = new Data(); //istanziazione
d.giorno = 50;  //accesso
Data k = d;     //dichiarazione
k.mese = 8;     //accesso
```

Data	
int	giorno
int	mese
int	anno
...	

FACCIAMO UN ALTRO ESEMPIO




```
Data d;          //dichiarazione
d = new Data(); //istanziazione
d.giorno = 50;  //accesso
Data k = d;     //dichiarazione
k.mese = 8;     //accesso
d = null;       //sovrascrittura
```

Data	
int	giorno
int	mese
int	anno
...	

OOP

Il costruttore

IL COSTRUTTORE IMPLICITO

```
Contatore c = new Contatore();
```

```
Data d = new Data();
```

Abbiamo appena visto che istruzioni come queste – tra le altre cose – provvedono a mettere il valore 0 in ogni attributo.

Questo accade perché vogliamo che **l'oggetto sia in ogni istante utilizzabile**.

Se non ci fosse questo accorgimento le seguenti righe di codice darebbero l'errore: «la var potrebbe non essere inizializzata»

```
Contatore c = new Contatore();
c.incrementa();
```

```
Data d = new Data();
d.aggiungiUnGiorno();
```

Questo meccanismo si chiama **costruttore implicito**.

Attenzione! Mentre nel caso di sinistra il costruttore implicito dando il valore 0 all'attributo **valore** prepara un oggetto di tipo **Contatore** valido, non si può dire lo stesso nel caso di destra per la **Data**!

ALESSANDRO URSOMANDO

IL COSTRUTTORE PROPRIO

Ci viene data la possibilità di creare un costruttore personalizzato.

Il costruttore ha sempre lo stesso nome della classe,
non restituisce niente (niente non signfica **void**, significa niente!)
e non può essere chiamato esplicitamente.

```
public class Data {
    // attributi
    int giorno, mese, anno;

    // il costruttore




    // qui ci sono gli altri metodi
}
```


IL COSTRUTTORE PROPRIO

Ci viene data la possibilità di creare un costruttore personalizzato.

Il costruttore ha sempre lo stesso nome della classe,
non restituisce niente (niente non signfica **void**, significa niente!)
e non può essere chiamato esplicitamente.

```
public class Data {
    // attributi
    int giorno, mese, anno;

    // il costruttore
    Data() {
        giorno = 19;
        mese = 4;
        anno = 1976;
    }

    // qui ci sono gli altri metodi
}
```

Adesso
il seguente blocco di codice
non va in errore
ed ha anche senso
perché l'oggetto di tipo **Data**
ha un contenuto coerente con se stesso
anche appena istanziato.

```
Data d = new Data();
d.aggiungiUnGiorno();
```

IL COSTRUTTORE CON PARAMETRI

Abbiamo già visto che spesso
prima di cominciare ad usare un oggetto
ci tocca andare a riempire i suoi attributi.

Ha senso dunque pensare a un
costruttore parametrizzato.

```
// dichiaro la data
Data d = new Data();

// riempio la data
d.giorno = 19;
d.mese = 4;
d.anno = 1976;
```

```
public class Data {
    // attributi
    int giorno, mese, anno;

    // il costruttore
    Data(int g, int m, int a) {
        giorno = g;
        mese = m;
        anno = a;
    }

    // qui ci sono gli altri metodi
}
```

```
// dichiaro la data e
//  contestualmente la
//  inizializzo

Data d = new Data(12, 12, 1969);
```


IL COSTRUTTORE CON PARAMETRI

Qui il nome del parametro ed il nome dell'attributo sono diversi

ma cosa accadrebbe se fossero uguali?

che questa istruzione sarebbe ambigua

```
public class Data {
    // attributi
    int giorno, mese, anno;

    // il costruttore
    Data(int g, int m, int a) {
        giorno = g;
        mese = m;
        anno = a;
    }

    // qui ci sono gli altri metodi
}
```

```
public class Data {
    // attributi
    int giorno, mese, anno;

    // il costruttore
    Data(int giorno, int m, int a) {
        giorno = giorno;
        mese = m;
        anno = a;
    }

    // qui ci sono gli altri metodi
}
```

IL RIFERIMENTO THIS

In questo caso si usa il riferimento **this** per disambiguare l'attributo dal parametro.

```
public class Data {
    // attributi
    int giorno, mese, anno;

    // il costruttore
    Data(int giorno, int m, int a) {
        this.giorno = giorno;
        mese = m;
        anno = a;
    }

    // qui ci sono gli altri metodi
}
```

```
public class Data {
    // attributi
    int giorno, mese, anno;

    // il costruttore
    Data(int giorno, int m, int a) {
        giorno = giorno;
        mese = m;
        anno = a;
    }

    // qui ci sono gli altri metodi
}
```


PIÙ DI UN COSTRUTTORE

È ragionevole pensare a dei casi in cui abbia senso predisporre più di un costruttore.

```
public class Orario {
    // attributi
    int ore, minuti, secondi;









}
```

Più di un Costruttore

È ragionevole pensare a dei casi in cui abbia senso predisporre più di un costruttore.

```
public class Orario {
    // attributi
    int ore, minuti, secondi;

    // costruttore senza parametri
    Orario() {
        ore = minuti = secondi = 0;
    }









}
```


Più di un Costruttore

È ragionevole pensare a dei casi in cui abbia senso predisporre più di un costruttore.

```
public class Orario {
    // attributi
    int ore, minuti, secondi;

    // costruttore senza parametri
    Orario() {
        ore = minuti = secondi = 0;
    }
    // costruttore con due parametri
    Orario(int o, int m) {
        ore = o;
        minuti = m;
        secondi = 0;
    }









}
```

PIÙ DI UN COSTRUTTORE

È ragionevole pensare a dei casi in cui abbia senso predisporre più di un costruttore.

```
public class Orario {
    // attributi
    int ore, minuti, secondi;

    // costruttore senza parametri
    Orario() {
        ore = minuti = secondi = 0;
    }
    // costruttore con due parametri
    Orario(int o, int m) {
        ore = o;
        minuti = m;
        secondi = 0;
    }
    // costruttore con tre parametri
    Orario(int o, int m, int s) {
        ore = o;
        minuti = m;
        secondi = s;
    }
}
```


PIÙ DI UN COSTRUTTORE

Nel caso di costruttori multipli,
i costruttori possono richiamarsi l'un l'altro mediante il riferimento **this**

```
public class Orario {
    // attributi
    int ore, minuti, secondi;

    // costruttore con tre parametri
    Orario(int o, int m, int s) {
        ore = o;
        minuti = m;
        secondi = s;
    }









}
```

PIÙ DI UN COSTRUTTORE

Nel caso di costruttori multipli,
i costruttori possono richiamarsi l'un l'altro mediante il riferimento **this**

```
public class Orario {
    // attributi
    int ore, minuti, secondi;

    // costruttore con tre parametri
    Orario(int o, int m, int s) {
        ore = o;
        minuti = m;
        secondi = s;
    }
    // costruttore senza parametri
    Orario() {
        this(0,0,0);
    }




}
```


PIÙ DI UN COSTRUTTORE

Nel caso di costruttori multipli,
i costruttori possono richiamarsi l'un l'altro mediante il riferimento **this**

```
public class Orario {
    // attributi
    int ore, minuti, secondi;

    // costruttore con tre parametri
    Orario(int o, int m, int s) {
        ore = o;
        minuti = m;
        secondi = s;
    }
    // costruttore senza parametri
    Orario() {
        this(0,0,0);
    }
    // costruttore con due parametri
    Orario(int o, int m) {
        this(o,m,0);
    }
}
```