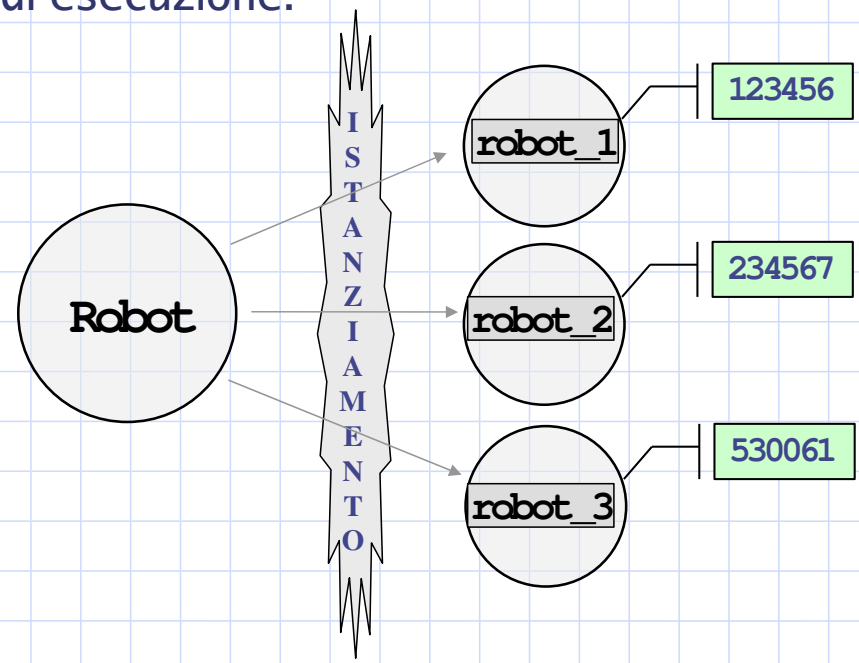
Classi - ulteriori concetti ed esempi

- Gestione della memoria
- Attributi e metodi di classe
- Notazione UML
- Realizzazione in C++ e Java

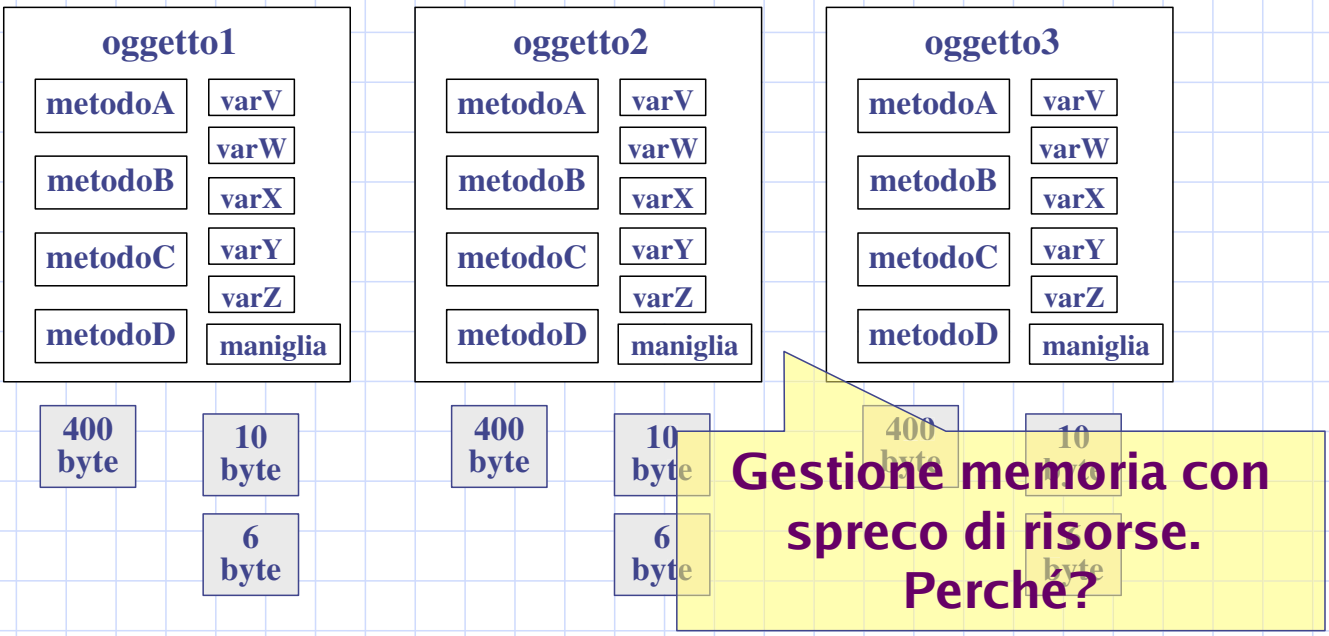
Classi – Oggetti - Maniglie

- Ricordiamo: Una classe è ciò che progettate e programmate.
- Gli oggetti sono ciò che create (a partire da una classe) in fase di esecuzione.



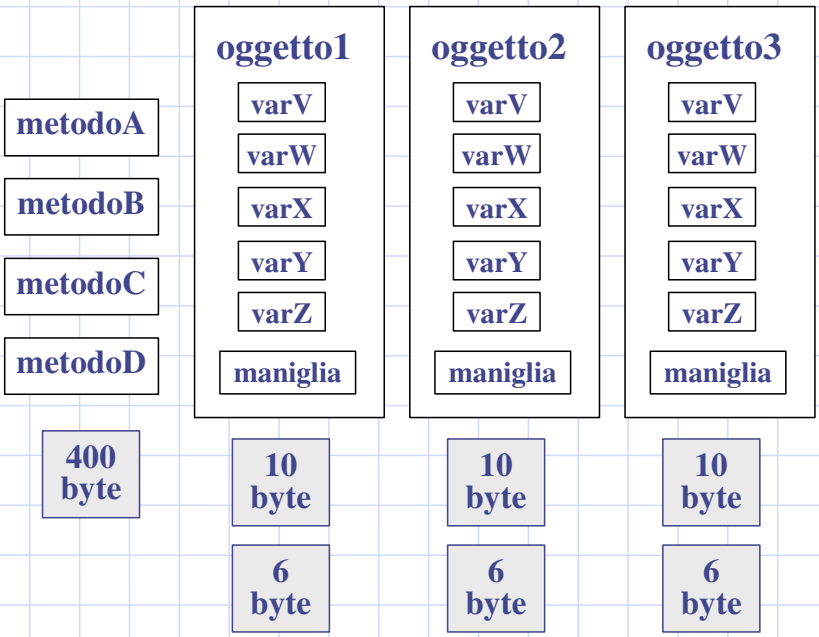
Classe: *gestione memoria (1)*

In linea di principio, l'istanziamento potrebbe produrre tante copie di Metodi ed Attributi quanti sono gli oggetti creati. A livello di memoria sono utilizzati 3*416 = 1248 byte



Classe: *gestione memoria (2)*

Memoria **effettiva** utilizzata da tre oggetti della stessa classe: 400 + 3*16 = 448 byte



Classe: *attributi e metodi di classe*

- Oltre a **attributi** e **metodi di istanza** esistono anche **attributi** e **metodi di classe.**

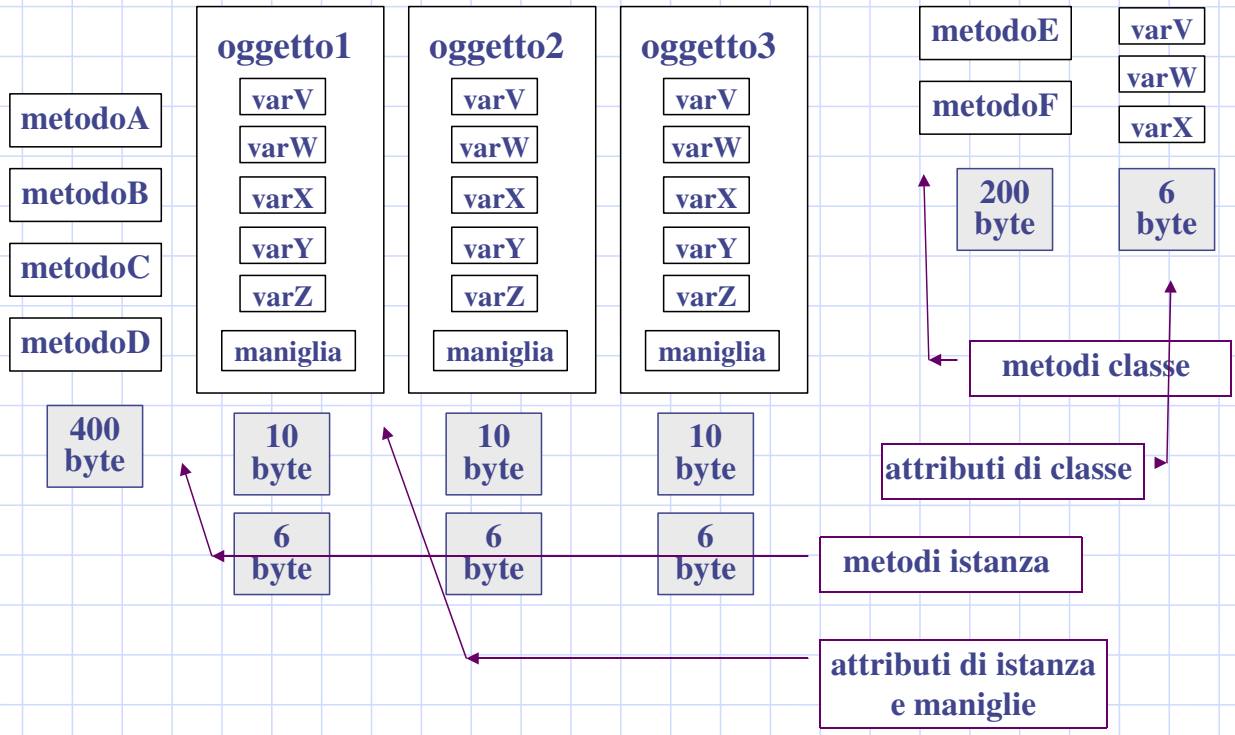
- Le operazioni e gli attributi di classe sono necessari per far fronte alle situazioni che non possono essere responsabilità di un qualunque oggetto singolo.

- Esempi per la classe **`Robot`** :
  - Metodo di classe: **`New()`**
  - Attributo di classe: **`numeroRobotCreati`**

- Esempio per la classe **`Studente`** :
  - Attributo di classe: **`nextMatricola`**

Classe: *gestione memoria (3)*

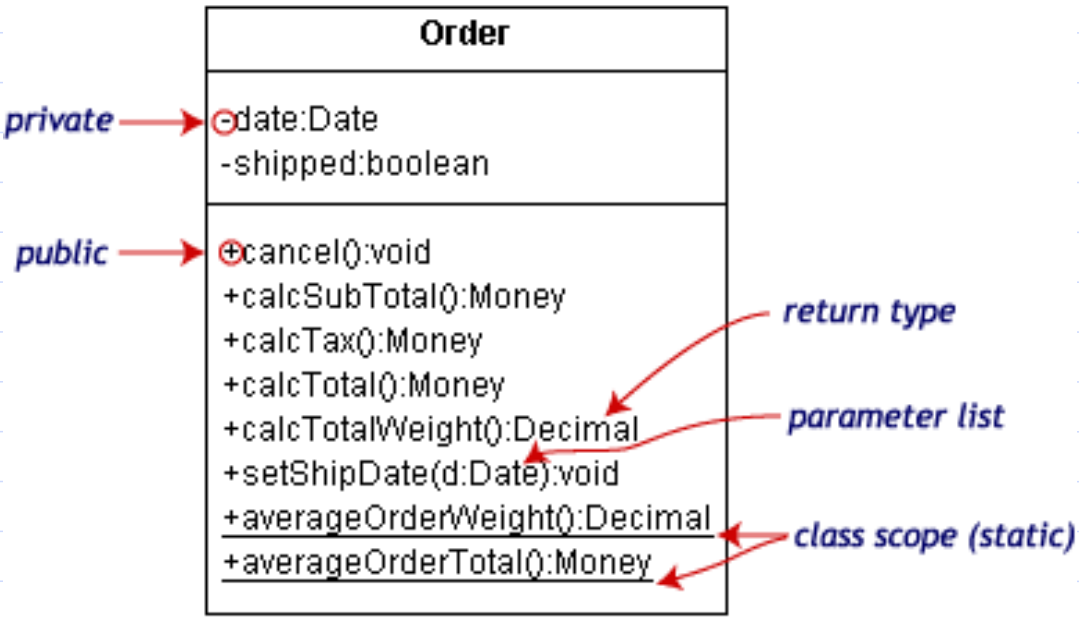
Memoria effettiva utilizzata da tre oggetti e dal "meccanismo di classe":



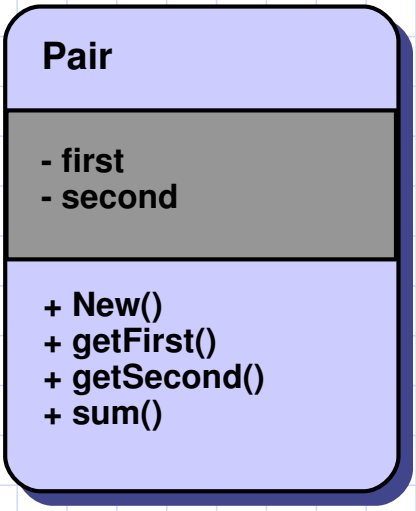
Osservazioni

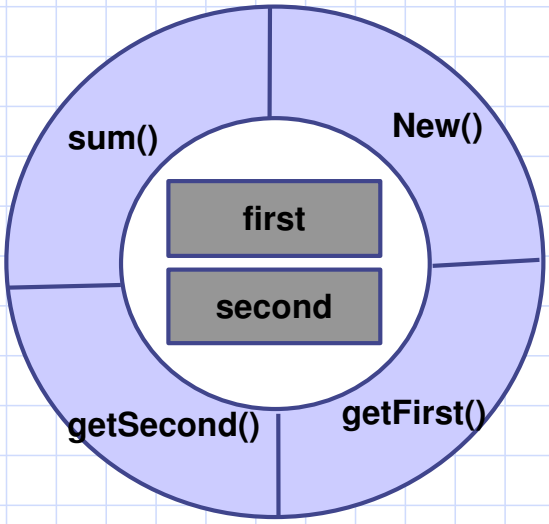
- Classe A senza “attributi di classe”:
  - L’uso della classe A comporta occupazione di memoria solo nel momento in cui il programmatore istanzia il primo oggetto di A
- Classe B con “attributi di classe”
  - L’uso della classe B comporta immediatamente occupazione di memoria; il progettista di B ha dovuto prevedere allocazione di memoria per gli attributi di classe, e questi esistono prima ancora che vengano creati oggetti di B

Classe: notazione UML



La classe “Pair”





Sintassi C++ costrutto “Class”

```
class Pair {
private:
      int first, second;
public:
       Pair( int a, int b ) { … }
   int getFirst() { … }
    int getSecond() { …  }
    Pair sum( Pair c ) { … }
};
```

Nome della classe

attributi

metodi

Implementazione metodi in C++…

```
class Pair {
private:
      int first, second;
public:
        Pair() { first=second=0; }
        Pair( int a ) {first=second=a; }
        Pair( int a, int b ) { first=a; second=b; }
    int getFirst() { return first; }
    int getSecond() { return second;  }
    Pair sum( const Pair & c ) {
     Pair ris;
     ris.first = first + c.first;
     ris.second = second + c.second;
     return ris; }
};
```

nome della classe

attributi

(overloading di) **costruttori**

metodi

Realizzazione

- Realizzare in C++ la classe Pair
- scrivere un programma che crea e manipola oggetti della classe Pair

Sintassi Java costrutto "Class"

```
public class Pair {

    private int first = 0;
    private int second = 0;

    public Pair(int a, int b) {…}

    public int getFirst(){…}
    public int getSecond(){…}
    public Pair sum(Pair c) {…}
}
```

Nome della classe

attributi (inizializzati!)

metodi

Implementazione metodi in Java…

```
public class Pair {
    private int first = 0;
    private int second = 0;

    public Pair() {}
    public Pair(int a) {first=a; second=a;}
    public Pair(int a, int b) {first=a; second=b;}

    public int getFirst(){ return first;}
    public int getSecond(){ return second;}
    public Pair sum(Pair c) {
        Pair ris = new Pair();
        ris.first = first + c.first;
        ris.second = second + c.second;
        return ris;}
}
```

(overloading di) **costruttori**

metodi

TDA "*fraction*"

nome fraction
interfaccia

- *fraction (int, int) : fraction*
  precondizioni e postcondizioni per *fraction(n,m) = n'/m'*
  pre: $m \neq 0$
  post: *n/m= n'/m'*; *m'* > 0; *n' e m'* sono primi fra loro

- *getNum : (fraction) --> int*
  precondizioni e postcondizioni per *getNum(n/m) = n*
  pre: nessuna
  post: nessuna

- *isPos : (fraction) --> bool*
  precondizioni e postcondizioni per *isPos(n/m) = b*
  pre: nessuna
  post: *b=true* se e solo se *n*>0 *(ricorda che, essendo n/m una fraction, allora m>0)*

… continua

- *sum (fraction,fraction) : fraction*
  precondizioni e postcondizioni per *sum(n/m, p/q) = r/s*
  pre: nessuna
  post: *r/s = n/m + p/q*; *r e s* sono primi fra loro

- *sub (fraction,fraction) : fraction*
  precondizioni e postcondizioni per *sub(n/m, p/q) = r/s*
  pre: nessuna
  post: *r/s = n/m - p/q* ; *r e s* sono primi fra loro

- *mul (fraction,fraction) : fraction*
  precondizioni e postcondizioni per *mul(n/m, p/q) = r/s*
  pre: nessuna
  post: *r/s = n/m* $\times$ *p/q* ; *r e s* sono primi fra loro

- *div (fraction,fraction) : fraction*
  precondizioni e postcondizioni per *sub(n/m, p/q) = r/s*
  pre: $p \neq 0$
  post: *r/s = n/m / p/q* ; *r e s* sono primi fra loro

esercizio

- Progettare la classe “fraction” facendo uso del C++
- Rispettare la specifica della classe fornita tramite pre- e post-condizioni
- Inserire altri metodi (pubblici/privati?) se ritenuti necessari
- Suggerimento: è conveniente usare “funzioni di utilità”