Realizzazione di Classi

Emilio Di Giacomo e Walter Didimo

Definizioni di classi

- Fino ad ora abbiamo imparato a:
  - creare oggetti da classi già pronte
  - usare gli oggetti creati, invocando metodi
  - la creazione e l'uso di oggetti avveniva sempre nel metodo speciale *main*
- Vogliamo ora imparare a definire noi stessi classi complete di proprietà, costruttori e metodi

La classe Rettangolo

- Inizieremo con un esempio. Vogliamo definire la classe *Rettangolo* simile a quella già vista nella lezione D4
- Un oggetto di tale classe rappresenta un rettangolo nel piano descritto tramite *base* e *altezza*
  - questa volta assumeremo tali valori reali

La classe Rettangolo

- La classe *Rettangolo* avrà i seguenti costruttori e metodi:
  - *public Rettangolo (double b, double a)*: costruttore che permette di specificare base e altezza dell'oggetto creato
  - *public double perimetro ()*: restituisce il perimetro del rettangolo rappresentato dall'oggetto
  - *public double frazioneDiArea (double f)*: restituisce la frazione *f* dell'area del rettangolo ($f \in [0,1]$)

La classe Rettangolo

- La classe *Rettangolo* avrà i seguenti costruttori e metodi:
  - *public void cambiaDimensioni (double b, double a)*: cambia le dimensioni del rettangolo con *b* ed *a*
  - *public static int rettCreati()*: restituisce il numero di oggetti della classe *Rettangolo* creati fino a questo momento

Commenti

- Tutti i metodi elencati sono pubblici
  - possono essere utilizzati all'interno della classe *Rettangolo* ma anche all'interno di altre classi
- I metodi *perimetro()*, *frazioneDiArea(double f)* e *cambiaDimensioni(double b, double a)* sono metodi di istanza
  - vanno invocati su un oggetto di tipo *Rettangolo*

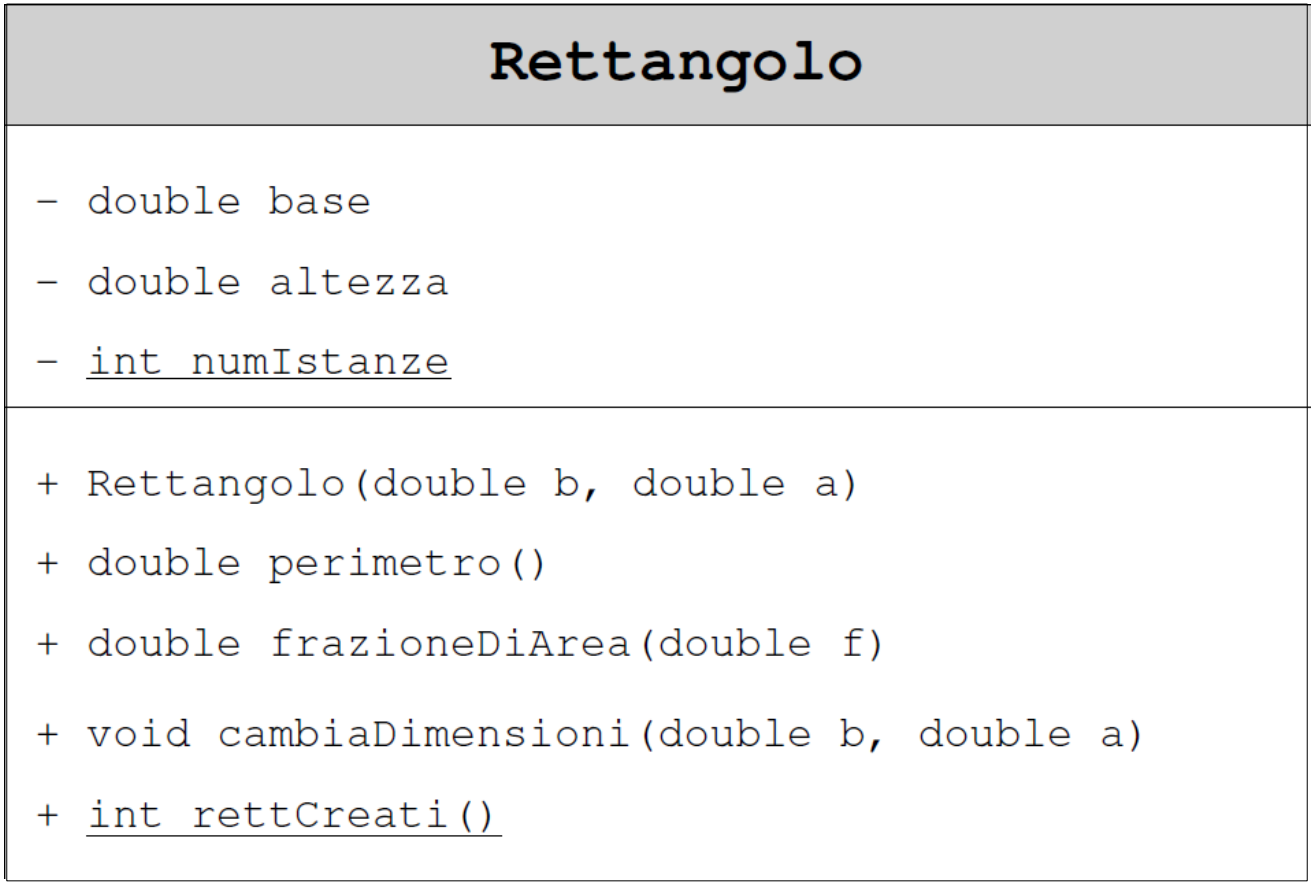
Commenti

- I metodi *perimetro()*, *frazioneDiArea(double f)* non cambiano lo stato dell'oggetto su cui sono invocati
- Il metodo *cambiaDimensioni(double b, double a)* modifica lo stato dell'oggetto su cui è invocato
- Il metodo *rettCreati()* è un metodo di classe
  - va invocato direttamente sulla classe

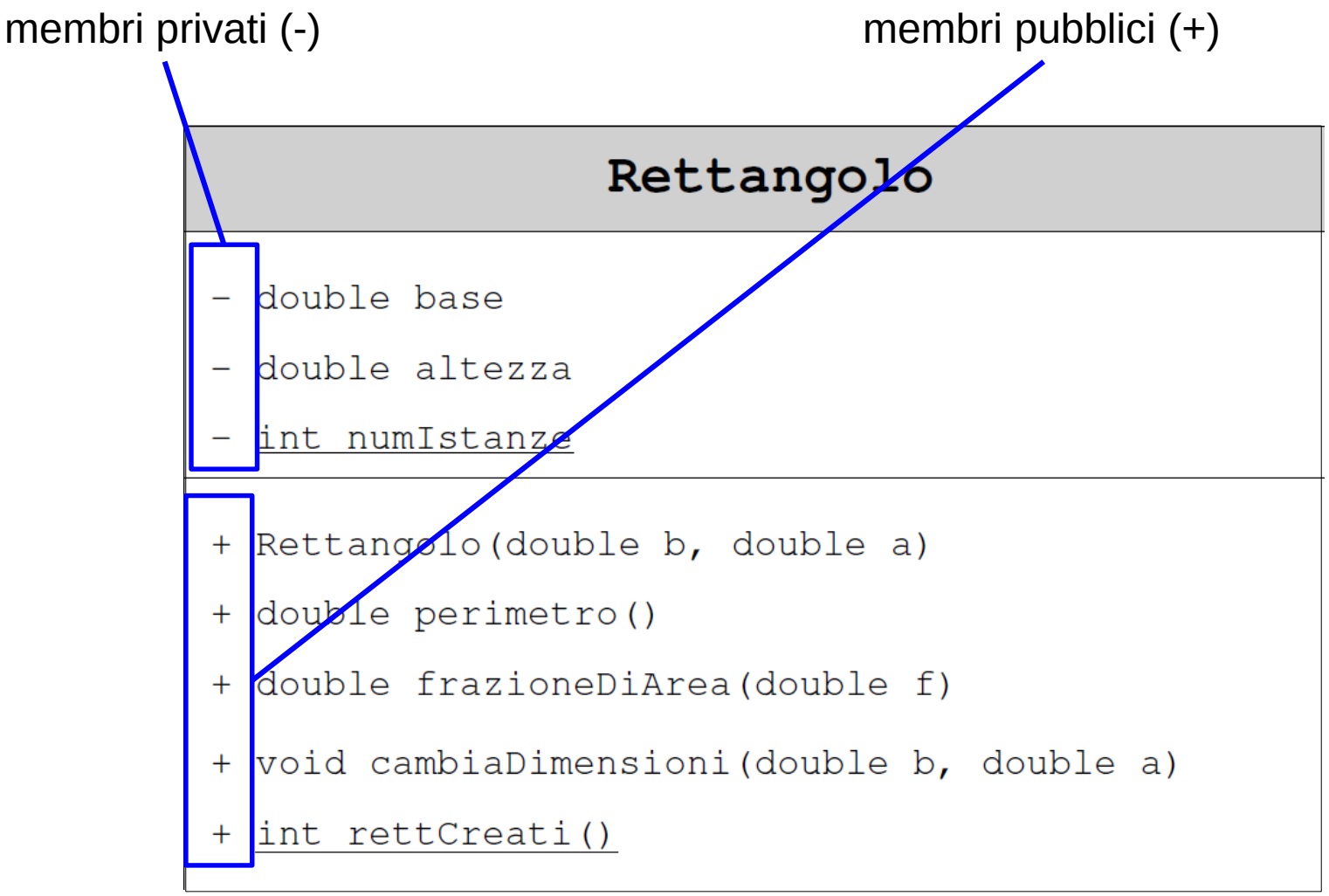
La classe Rettangolo

Rettangolo
double base double altezza int numIstanze
Rettangolo(double b, double a) double perimetro() double frazioneDiArea(double f) void cambiaDimensioni(double b, double a) int rettCreati()

Notazione grafica “arricchita”



Notazione grafica “arricchita”



Attributi

- Gli attributi di istanza della classe Rettangolo sono *base* e *altezza*
  - essi memorizzano rispettivamente la base e l'altezza del rettangolo rappresentato
- L'attributo *numIstanze* è invece un attributo di classe che memorizza il numero di istanze create
  - verrà utilizzato dal metodo *rettCreati()*

Scriviamo la classe Rettangolo

- La struttura della classe *Rettangolo* sarà la seguente

```
public class Rettangolo{
    // attributi
    ...
    // costruttori
    ...
    // metodi
    ...
}
```

Attributi e variabili

- Gli attributi di un oggetto *Rettangolo* debbono servire a memorizzare il valore della sua base e quello della sua altezza durante tutta la vita dell'oggetto
  - a tal fine, gli attributi di un oggetto possono essere realizzati attraverso l'uso di opportune variabili, nel nostro caso variabili di tipo *double*
  - le variabili usate per definire gli attributi degli oggetti (istanze) di una classe si chiamano variabili di istanza

Attributi e variabili

- L'attributo di classe *numIstanze* serve a memorizzare il numero di rettangoli creati durante tutta la vita della classe
  - anche gli attributi di classe possono essere realizzati attraverso l'uso di opportune variabili, nel nostro caso una variabile di tipo *int*
  - le variabili usate per definire gli attributi delle classi si chiamano variabili di classe oppure variabili statiche

Dichiarazione di attributi

- Nella definizione di una classe, le variabili di istanza e di classe si dichiarano all'interno del corpo della classe ma al di fuori dei corpi dei metodi
  - tipicamente all'inizio del corpo della classe, prima dei metodi

```
public class Rettangolo{
    private double base;
    private double altezza;
    private static int numIstanze=0;
   /* costruttori e metodi */
      ...
}
```

variabili di istanza

variabile di classe

Variabili di istanza

- La dichiarazione delle variabili di istanza avviene come per le variabili viste fino ad ora; si scrive prima il tipo e poi il nome
  - tipicamente (ma non sempre) la definizione è preceduta dal modificatore d'accesso *private*
  - il modificatore *private* indica che le variabili definite possono essere accedute solo da metodi della classe *Rettangolo*, e non da metodi di altre classi
  - in questo modo lo stato degli oggetti può essere modificato solo mediante i metodi pubblici

Semplificazioni sintattiche

- Variabili diverse dello stesso tipo possono essere definite mediante un'unica istruzione

```
public class Rettangolo{
    private double base, altezza;
    private static int numIstanze=0;
   /* costruttori e metodi */
      ...
}
```

- In questo modo il compilatore assume che *base* e *altezza* siano entrambe *double* e *private*

Oggetti e variabili di istanza

- Le variabili di istanza rappresentano “la memoria di un oggetto”, cioè il suo stato nel tempo
  - ogni oggetto *Rettangolo* avrà due variabili di istanza, in cui potrà mantenere il valore della sua base e quello della sua altezza per tutta la sua vita
  - il valore di una variabile di istanza può cambiare nel tempo – i metodi dell’oggetto possono modificarlo
  - due oggetti distinti di tipo *Rettangolo* hanno cicli di vita indipendenti – i valori delle loro variabili di istanza variano in modo indipendente!

Valore iniziale delle var. di istanza

- L'inizializzazione delle variabili di istanza di un oggetto, cioè del suo stato, è tipicamente eseguita nel costruttore che viene invocato subito dopo la creazione dell'oggetto
- Tuttavia subito dopo che l'oggetto è stato creato con l'operatore *new* (e prima che il costruttore venga eseguito) la JVM assegna ad ogni variabile di istanza un valore di default

Valore iniziale delle var. di istanza

- Valori di default:
  - per i tipi interi (*byte*, *short*, *int*, *long*): *0*
  - per i tipi in virgola mobile (*float*, *double*): *0.0*
  - per il tipo *char*: *'\u0000'*
  - per il tipo *boolean*: *false*
  - per i tipi riferimento: *null*

Valore iniziale delle var. di istanza

- È anche possibile inizializzare le variabili di istanza contestualmente alla loro definizione
- Ad esempio

```
public class Rettangolo{
    private double base=-1;
    private double altezza=-1;
    private static int numIstanze=0;
   /* costruttori e metodi */
      ...
}
```

Variabili di classe

- Anche la dichiarazione delle variabili di classe avviene scrivendo prima il tipo e poi il nome
  - le variabili di classe vanno definite mediante la clausola *static*
  - anche in questo caso la definizione è tipicamente preceduta dal modificatore d'accesso *private*

Valore iniziale delle var. di classe

- La variabili di classe iniziano ad esistere nel momento in cui la classe è richiamata per la prima volta
- L'inizializzazione di tali variabili avviene di solito contestualmente alla definizione:
  - non esiste un "costruttore di classe" che possa inizializzarle!

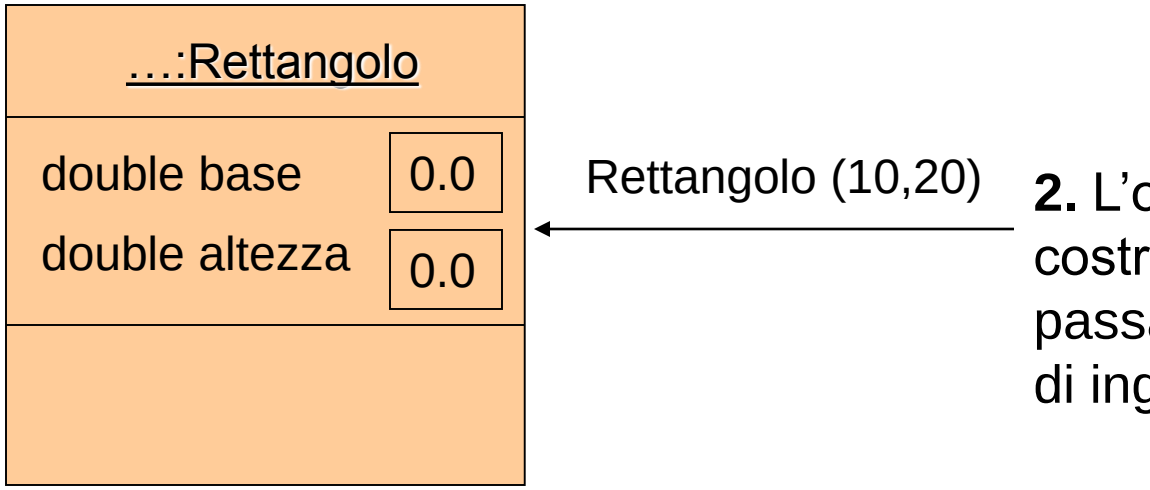
Costruttori

- Un costruttore ha il compito di assegnare un valore iniziale alle variabili di istanza
- Sappiamo che un costruttore è un metodo speciale:
  - non ha tipo di ritorno (un costruttore ha solo una signature e non un prototipo)
  - ha sempre il nome della classe
  - può essere invocato soltanto da *new*, un istante dopo la creazione fisica dell'oggetto
  - un costruttore non può essere invocato in momenti successivi della vita dell'oggetto

Costruttori

new Rettangolo (10, 20)

1. L'operatore *new* crea l'oggetto ed assegna alle sue variabili di istanza un valore di default



2. L'operatore *new* invoca il costruttore sull'oggetto creato, passando ad esso i parametri di ingresso

Costruttori

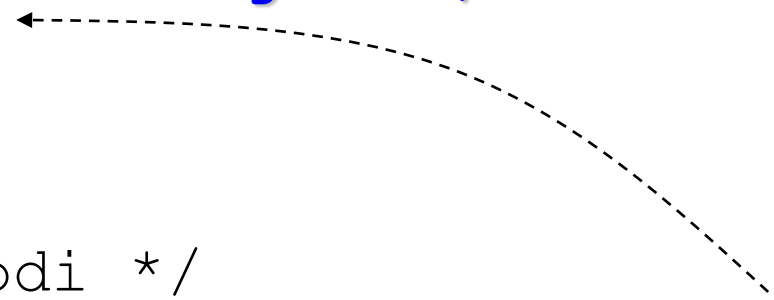
- Se in una classe non si definisce un costruttore, essa viene dotata automaticamente di un costruttore di default
- Tale costruttore non ha parametri e non modifica in alcun modo lo stato dell'oggetto

Definizione di costruttori

- Nel corpo di una classe, si usa definire i costruttori dopo le variabili di istanza e prima degli altri metodi
  - davanti alla definizione del costruttore si mette di solito il modificatore d'accesso *public*
  - il modificatore *public* indica che il costruttore può essere usato per creare oggetti da qualunque altro metodo (per esempio il metodo *main* di una qualche classe)

Definizione di costruttori

```
public class Rettangolo{
  private double base, altezza;
  private static int numIstanze=0;
  /* costruttore:crea un oggetto Rettangolo
  con base b e altezza a */
  public Rettangolo (double b, double a){
      ...
  }
  /* metodi */
  ...
}
```



qui va scritto il corpo del costruttore

Parametri formali

- Prima di scrivere il corpo del costruttore *Rettangolo*, dobbiamo capire meglio il suo ruolo e quello dei suoi parametri di ingresso
- Il costruttore *Rettangolo (double b, double a)* deve assegnare i valori dei parametri formali *b* ed *a* alle variabili di istanza dell'oggetto sul quale è invocato

Parametri formali

- I parametri formali di un metodo o costruttore sono variabili usate per contenere i valori che si passeranno al metodo/costruttore all'atto della sua invocazione
  - Vengono allocate in memoria all'atto dell'invocazione del metodo
  - Vengono rimosse quando l'esecuzione del metodo termina
  - Sono visibili soltanto all'interno del metodo
  - Variabili di questo tipo sono dette variabili locali

Parametri attuali

- Quando si chiamerà il costruttore, per esempio scrivendo

  new Rettangolo (10, 20)

- i parametri attuali *10* e *20* verranno copiati rispettivamente nelle variabili *b* ed *a* del costruttore, e poi verranno eseguite le sue istruzioni

Passaggio di parametri

- Ad ogni invocazione di un metodo (o di un costruttore), i parametri attuali vengono copiati nei parametri formali del metodo
  - questo meccanismo si chiama passaggio di parametri per valore
  - il corpo del metodo va scritto assumendo che nei parametri formali ci saranno i valori passati al metodo dall'esterno (cioè da chi lo invocherà)

Corpo del costruttore Rettangolo

- Il costruttore della classe *Rettangolo* deve copiare i valori dei suoi parametri formali nelle variabili di istanza dell'oggetto su cui è stato invocato – ecco il codice del costruttore

```
public Rettangolo (double b, double a){
    this.base = b;
    this.altezza = a;
    Rettangolo.numIstanze++;
}
```

- Analizziamo nel seguito le sue istruzioni

La parola chiave this

- Un metodo o un costruttore può riferirsi ad una variabile di istanza dell'oggetto su cui è invocato usando la parola chiave *this* seguita da un punto e dal nome della variabile:
  - *this* indica appunto "questo oggetto"
  - *this.base* indica la variabile di istanza *base* di *this*, cioè di questo oggetto

```
public Rettangolo (double b, double a){
    this.base = b;
    this.altezza = a;
    Rettangolo.numIstanze++;
}
```

Assegnamento a variabili di istanza

- L'istruzione *this.base = b* è una istruzione di assegnazione
  - il valore di *b* (espressione a destra di "=") è copiato nella variabile di istanza *this.base*
  - analogamente, *this.altezza = a* copia il valore di *a* nella variabile di istanza *this.altezza*

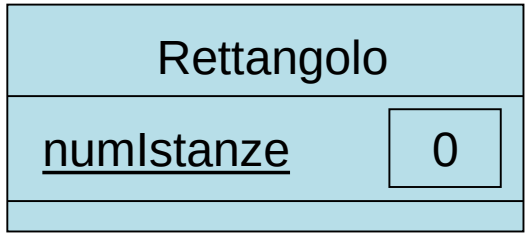
```
public Rettangolo (double b, double a){
    this.base = b;
    this.altezza = a;
    Rettangolo.numIstanze++;
}
```

Conteggio delle istanze create

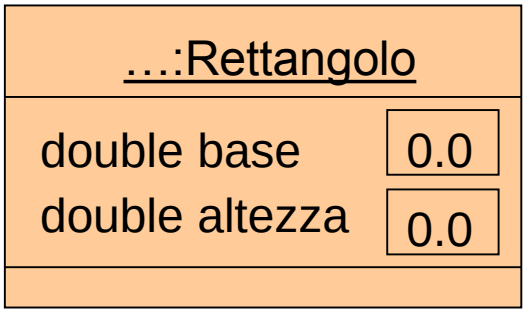
- L'ultima istruzione del costruttore incrementa di uno il valore della variabile *numIstanze*
  - si registra la creazione di un nuovo oggetto
  - poiché il costruttore viene invocato ogni volta che si crea un rettangolo, *numIstanze* memorizza il numero di istanze create
- Poiché *numIstanze* è una variabile di classe, si fa riferimento ad essa tramite il nome della classe seguito da un punto

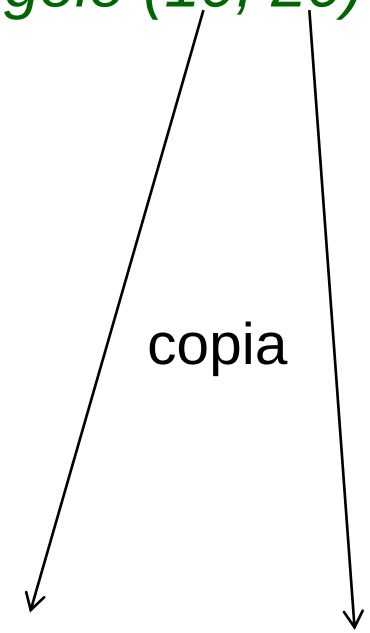
Esecuzione del costruttore

new Rettangolo (10, 20)



t_0

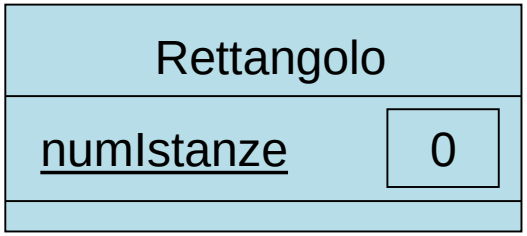




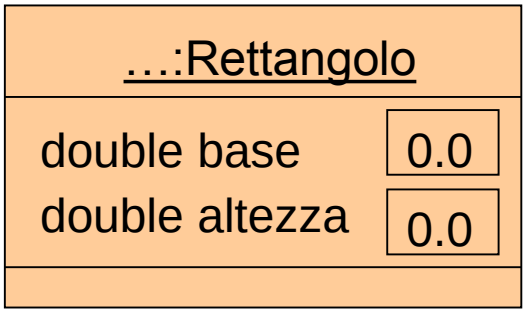

```
public Rettangolo (double b, double a){
        this.base = b;
        this.altezza = a;
        Rettangolo.numIstanze++;
}
```

Esecuzione del costruttore

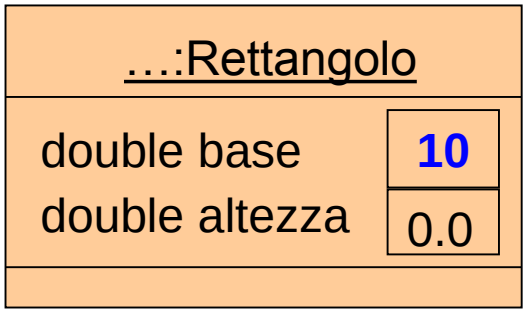
new Rettangolo (10, 20)



t_0



t_1

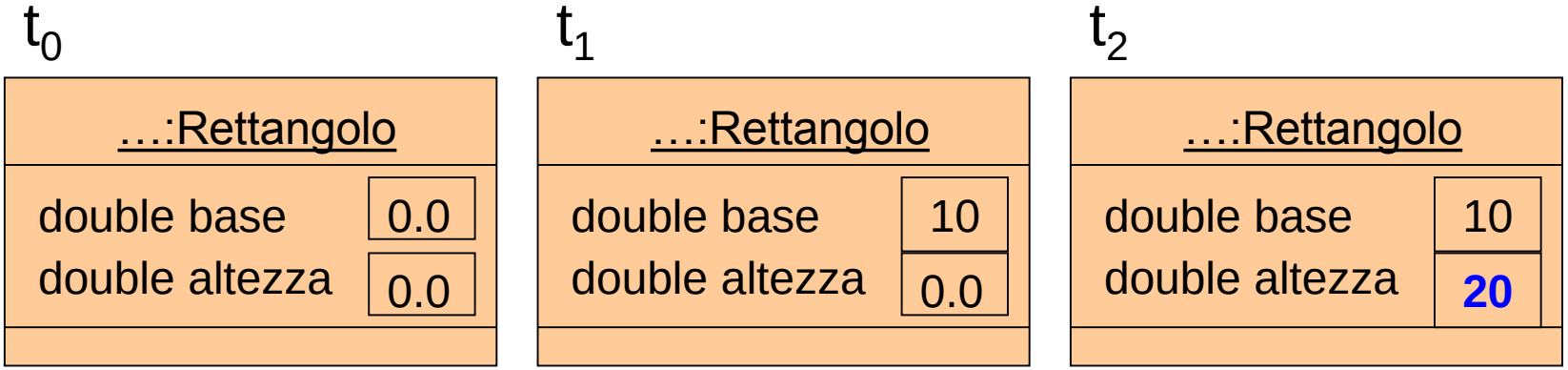



```
public Rettangolo (double b, double a){
        this.base = b;
        this.altezza = a;
        Rettangolo.numIstanze++;
}
```

Esecuzione del costruttore

new Rettangolo (10, 20)

Rettangolo	
numIstanze	0

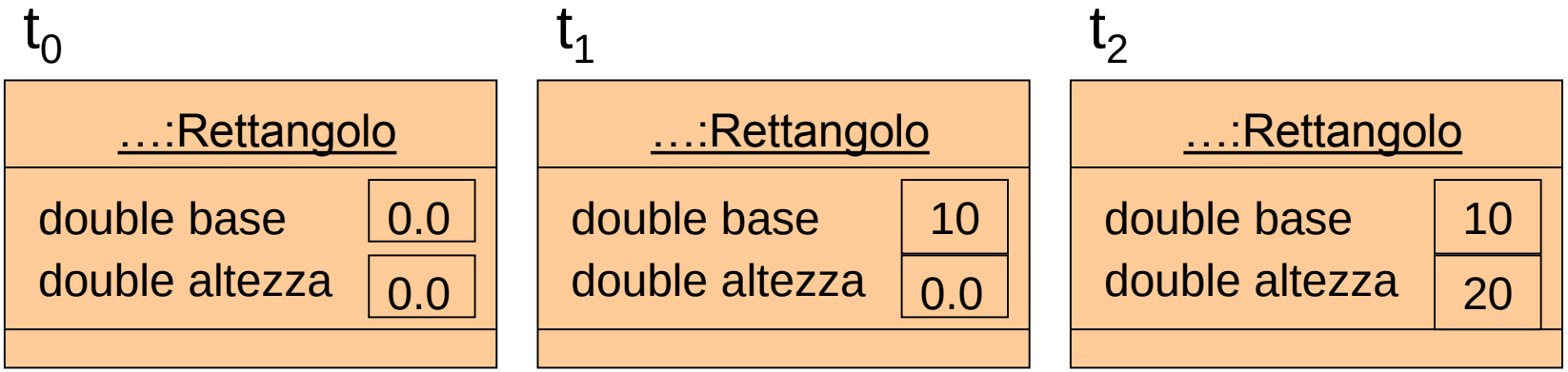



```
public Rettangolo (double b, double a){
        this.base = b;
        this.altezza = a;
        Rettangolo.numIstanze++;
}
```

Esecuzione del costruttore

new Rettangolo (10, 20)

Rettangolo	
numIstanze	1




```
public Rettangolo (double b, double a){
        this.base = b;
        this.altezza = a;
        Rettangolo.numIstanze++;
}
```

Il metodo perimetro

- Dobbiamo ora scrivere gli altri metodi della classe; cominciamo dal metodo che calcola e restituisce il perimetro

```
public double perimetro () {
    double p;
    p = (this.base + this.altezza)*2;
    return p;
}
```

- Discutiamo nel seguito il codice del metodo

Dichiarazione del metodo

```
public double perimetro () {
    double p;
    p = (this.base + this.altezza)*2;
    return p;
}
```

- Come per il costruttore, anche per dichiarare gli altri metodi usiamo la parola *public*; essa indica che i metodi potranno essere invocati da qualunque altro metodo (anche esterno alla classe *Rettangolo*)

Variabili locali

```
public double perimetro () {
    double p;
    p = (this.base + this.altezza)*2;
    return p;
}
```

- L'istruzione *double p* dichiara una variabile di nome *p* e di tipo *double*; essa verrà usata nel metodo per memorizzare il valore del perimetro; la variabile *p* è una *variabile locale*

Valore iniziale per le variabili locali

```
public double perimetro () {
    double p;
    p = (this.base + this.altezza)*2;
    return p;
}
```

- A differenza delle variabili di istanza, una variabile locale appena dichiarata non ha alcun valore di default associato
- Quando si accede ad una variabile locale per la prima volta, ad essa deve essere già stato assegnato un valore, altrimenti il compilatore segnala un errore!

Calcolo del perimetro

```
public double perimetro () {
    double p;
    p = (this.base + this.altezza)*2;
    return p;
}
```

- L'espressione *(this.base + this.altezza)*2* calcola il valore del perimetro, usando i valori contenuti nelle variabili di istanza
- Il valore dell'espressione è poi assegnato alla variabile *p*

Restituzione del perimetro

```
public double perimetro () {
    double p;
    p = (this.base + this.altezza)*2;
    return p;
}
```

- L'istruzione *return p* ha un duplice effetto:
  - restituire all'esterno il valore di *p*;
  - causare la terminazione del metodo;

L’istruzione return

- L’istruzione *return* deve essere usata in ogni metodo che dichiara di restituire un valore:
- il valore restituito deve essere coerente con il tipo di dato di ritorno;
- l’istruzione *return* causa la terminazione immediata del metodo:
  - è buona norma che *return* sia sempre l’ultima istruzione di un metodo che restituisce un valore;
  - se ci fossero istruzioni dopo il *return*, esse non verrebbero eseguite

Ancora sul metodo perimetro

- Il metodo *perimetro* ha le seguenti caratteristiche:
  - non ha parametri formali (non prende dati in ingresso)
  - per calcolare il perimetro dell'oggetto ha solo bisogno di conoscerne lo stato, cioè le sue variabili di istanza
  - non modifica lo stato dell'oggetto, ma lo accede soltanto in lettura
  - il metodo *perimetro* è un metodo di accesso
  - per contro, i metodi che modificano lo stato di un oggetto si chiamano anche metodi di modifica

Semplificazione del codice

- Il codice del metodo perimetro si può anche riscrivere in modo più sintetico come segue

```
public double perimetro () {
    return (this.base + this.altezza)*2;
}
```

- In questo modo verrebbe prima calcolato il valore dell'espressione associata al *return*, e poi tale valore verrebbe restituito

Il metodo frazioneDiArea

- Ecco il codice del metodo *frazioneDiArea*

```
public double frazioneDiArea (double f) {
    return (this.base*this.altezza)*f;
}
```

- Il metodo *frazioneDiArea* combina l'uso del parametro formale a quello delle variabili di istanza per calcolare il risultato – esso è ancora un metodo di accesso, poiché non modifica lo stato dell'oggetto

Il metodo cambiaDimensioni

- Ecco il codice del metodo *cambiaDimensioni*

```
public void cambiaDimensioni(double b,
                             double a){
    this.base = b;
    this.altezza = a;
}
```

- Il metodo non restituisce valori (è di tipo *void*)
- Il metodo agisce similmente al costruttore
- Il metodo cambia lo stato dell'oggetto; è un metodo di modifica

Il metodo rettCreati

- Ecco il codice del metodo *rettCreati*

```
public static int rettCreati(){
    return Rettangolo.numIstanze;
}
```

- Poiché il metodo è di classe nella sua definizione compare la parola chiave *static*
- Il metodo restituisce il valore della variabile *numIstanze*

Test della classe Rettangolo

- Una volta scritto il codice di una classe, è buona prassi verificarne la correttezza
- Una strategia che seguiremo in questo corso è quella di realizzare delle semplici classi di test:
  - una classe di test per una classe *C* ha soltanto il metodo speciale *main*
  - consente di eseguire tutti i metodi della classe *C*

Test della classe Rettangolo

```
import fond.io.*;
public class TestRettangolo{
   public static void main(String[] args){
      InputWindow in = new InputWindow();
      OutputWindow out= new OutputWindow("Test Rettangolo");
      double b = in.readDouble("Base primo rettangolo?");
      double a = in.readDouble("Altezza primo rettangolo?");
      Rettangolo rett1 = new Rettangolo(b, a);
      out.write("Rettangoli creati: ");
      out.writeln(Rettangolo.rettCreati());
      out.write("Perimetro primo rettangolo: ");
      out.writeln(rett1.perimetro());
      ...
   }
}
```

Test della classe Rettangolo

```
import fond.io.*;
public class TestRettangolo{
   public static void main(String[] args){
      ...
      double f=in.readDouble("Fraz. di area desiderata?");
      out.write("Frazione area primo rettangolo: ");
      out.writeln(rett1.frazioneDiArea(f));
      b = in.readDouble("Nuova base primo rettangolo?");
      a = in.readDouble("Nuova altezza primo rettangolo?");
      rett1.cambiaDimensioni(b, a);
      out.write("Perimetro primo rettangolo: ");
      out.writeln(rett1.perimetro());
      b = in.readDouble("Base secondo rettangolo?");
      ...
   }
}
```

Test della classe Rettangolo

```
import fond.io.*;
public class TestRettangolo{
   public static void main(String[] args){
      ...
      a = in.readDouble("Altezza secondo rettangolo?");
      Rettangolo rett2 = new Rettangolo(b, a);
      out.write("Rettangoli creati: ");
      out.writeln(Rettangolo.rettCreati());
      out.write("Perimetro secondo rettangolo: ");
      out.writeln(rett2.perimetro());
      out.write("Area secondo rettangolo: ");
      out.writeln(rett2.frazioneDiArea(1));
   }
}
```

Un altro esempio: Punto

- Scriviamo una classe Punto per rappresentare punti nel piano cartesiano specificati tramite due coordinate reali *x* e *y*
- La classe avrà i seguenti costruttori e metodi:
  - *public Punto(double x, double y)*: crea un oggetto *Punto* che rappresenta il punto (*x*,*y*)
  - *public double getCoordX()*: restituisce la coordinata *x* del punto rappresentato dall'oggetto ricevente
  - *public double getCoordY()*: restituisce la coordinata *y* del punto rappresentato dall'oggetto ricevente

Un altro esempio: Punto

- *public double distanzaDa(Punto p)*: restituisce la distanza euclidea tra il punto rappresentato dall'oggetto ricevente e il punto rappresentato dall'oggetto *p*
- *public Punto puntoTraslato(double dx, double dy)*: restituisce un nuovo oggetto *Punto*, che rappresenta il punto del piano ottenuto traslando il punto rappresentato dall'oggetto ricevente di una quantità *dx* nella direzione x e di una quantità *dy* nella direzione y

Un altro esempio: Punto

- *public boolean equals(Punto p)*: restituisce *true* se e solo se il punto rappresentato dall'oggetto ricevente coincide con il punto rappresentato dall'oggetto *p*
- *public String toString()*: restituisce una descrizione sotto forma di stringa del punto rappresentato dall'oggetto ricevente
- *public static double distanzaTra(Punto p1, Punto p2)*: metodo statico che restituisce la distanza euclidea tra il punto rappresentato da *p1* e quello rappresentato da *p2*

La classe Punto

Punto
- double coordX - double coordY
+ Punto(double x, double y) + double getCoordX() + double getCoordY() + double distanzaDa(Punto p) + Punto puntoTraslato(double dx, double dy) + boolean equals(Punto p) + boolean toString() + double distanzaTra(Punto p1, Punto p2)

La classe Punto

```
public class Punto{
   private double coordX, coordY;
   /* costruttore: crea un oggetto Punto con
   coordinate x e y specificate */
   public Punto(double x, double y){
       this.coordX = x;
       this.coordY = y;
   }
   ...
}
```

- Un punto è rappresentato tramite le due variabili di istanza *coordX* e *coordY*, che rappresentano le coordinate del punto

La classe Punto

```
public class Punto{
   private double coordX, coordY;
   /* costruttore: crea un oggetto Punto con
   coordinate x e y specificate */
   public Punto(double x, double y){
        this.coordX = x;
        this.coordY = y;
   }
   ...
}
```

- Il costruttore inizializza le variabili di istanza con i valori passati tramite i parametri

La classe Punto

```
/* restituisce la coordinate x di questo Punto */
public double getCoordX(){
   return this.coordX;
}

/* restituisce la coordinate y di questo Punto */
public double getCoordY(){
   return this.coordY;
}
```

- I metodi *getCoordX()* e *getCoordY()* restituiscono il valore delle variabili di istanza

La classe Punto

```
/* restituisce la distanza tra questo Punto e p */
public double distanzaDa(Punto p){
   double diffX = this.coordX-p.coordX;
   double diffY = this.coordY-p.coordY;
   double dist = Math.sqrt(diffX*diffX+diffY*diffY);
   return dist;
}
```

- Il metodo *distanzaDa* calcola la distanza tra il punto rappresentato dall'oggetto ricevente e il punto rappresentato da *p*

La classe Punto

```
/* restituisce la distanza tra questo Punto e p */
public double distanzaDa(Punto p){
   double diffX = this.coordX-p.coordX;
   double diffY = this.coordY-p.coordY;
   double dist = Math.sqrt(diffX*diffX+diffY*diffY);
   return dist;
}
```

- Nelle variabili locali *diffX* e *diffY* vengono memorizzate rispettivamente le differenze delle coordinate *x* e delle coordinate *y* dei due punti

La classe Punto

```
/* restituisce la distanza tra questo Punto e p */
public double distanzaDa(Punto p){
   double diffX = this.coordX-p.coordX;
   double diffY = this.coordY-p.coordY;
   double dist = Math.sqrt(diffX*diffX+diffY*diffY);
   return dist;
}
```

- Nella variabile *dist* viene memorizzata la distanza tra i due punti calcolata secondo la formula della distanza euclidea

La classe Punto

```
/* restituisce la distanza tra questo Punto e p */
public double distanzaDa(Punto p){
   double diffX = this.coordX-p.coordX;
   double diffY = this.coordY-p.coordY;
   double dist = Math.sqrt(diffX*diffX+diffY*diffY);
   return dist;
}
```

- La distanza calcolata viene restituita

La classe Punto

```
/* restituisce un nuovo Punto ottenuto traslando
   questo Punto di dx nella direzione dell'asse x
   e di dy nella direzione dell'asse y */
public Punto puntoTraslato(double dx, double dy){
   return new Punto(this.coordX+dx, this.coordY+dy);
}
```

- Il metodo *puntoTraslato* crea e restituisce un oggetto *Punto* le cui coordinate sono *this.coordX+dx* e *this.cordY+dy*

La classe Punto

```
/* restituisce true se questo Punto e il punto p
   hanno le stesse coordinate */
public boolean equals(Punto p){
   return (this.coordX==p.coordX &&
           this.coordY==p.coordY);
}
```

- Il metodo *equals* verifica se l'oggetto ricevente e *p* hanno le stesse coordinate

La classe Punto

```
/* restituisce una descrizione di questo Punto sotto
   forma di stringa */
public String toString(){
   String des = "("+this.coordX+","+this.coordY+")";
   return des;
}
```

- Il metodo *toString* crea una stringa *des* che descrive l'oggetto ricevente nella forma (x,y)

La classe Punto

```
/* restituisce una descrizione di questo Punto sotto
   forma di stringa */
public String toString(){
   String des = "("+this.coordX+","+this.coordY+")";
   return des;
}
```

- Tale stringa viene poi restituita

La classe Punto

```
/* restituisce la distanza tra il Punto p1 e il
   Punto p2 */
public static double distanzaTra(Punto p1,Punto p2){
   double diffX = p1.coordX-p2.coordX;
   double diffY = p1.coordY-p2.coordY;
   double dist = Math.sqrt(diffX*diffX+diffY*diffY);
   return dist;
}
```

- Il metodo è simile al precedente metodo *distanzaDa*
- La differenza è che questo metodo è statico...

La classe Punto

```
/* restituisce la distanza tra il Punto p1 e il
   Punto p2 */
public static double distanzaTra(Punto p1,Punto p2){
   double diffX = p1.coordX-p2.coordX;
   double diffY = p1.coordY-p2.coordY;
   double dist = Math.sqrt(diffX*diffX+diffY*diffY);
   return dist;
}
```

- ...e i due punti sono passati come parametro

La classe Punto

- Una versione alternativa del metodo *distanzaTra* è la seguente:

```
/* restituisce la distanza tra il Punto p1 e il
   Punto p2 */
public static double distanzaTra(Punto p1,Punto p2){
   return p1.distanzaDa(p2);
}
```

Ulteriori osservazioni

- Vogliamo fare nel seguito ulteriori osservazioni sui concetti introdotti; esse riguarderanno:
  - la distinzione tra variabili di istanza, variabili locali e parametri formali
  - l'uso della parola chiave *this*
  - l'uso dei costruttori

Variabili: classificazione

- Abbiamo visto che, dal punto di vista del tipo di dato associato, le variabili si classificano in:
  - *variabili primitive*, se memorizzano dati di tipo primitivo
  - *variabili riferimento*, se memorizzano riferimenti ad oggetti
- Dal punto di vista di dove si trovano nel codice, esse si classificano invece in:
  - *variabili di istanza e di classe*, se definite fuori dei metodi
  - *variabili locali* se definite all'interno dei metodi
  - *parametri formali* se costituiscono i parametri di un metodo

Variabili: classificazione

- Le due classificazioni precedenti sono indipendenti
- Ad esempio, con riferimento alla classe *Rettangolo*, sono variabili primitive
  - le variabili di istanza *base* e *altezza*
  - la variabile di classe *numIstanze*
  - i parametri formali *b*, *a*, *f*,
  - la variabile locale *p* nel metodo *perimetro()* e la variabile locale *frazione* nel metodo *frazioneDiArea(double f)*

Variabili: classificazione

- Le due classificazioni precedenti sono indipendenti
- Ad esempio, con riferimento alla classe *Punto*, sono variabili riferimento
  - il parametro formale *p* del metodo *distanzaDa(Punto p)* e del metodo *equals(Punto p)*
  - la variabile locale *des* nel metodo *toString()*
  - i parametri formali *p1* e *p2* del metodo statico *distanzaTra(Punto p1, Punto p2)*

Variabili: scope

- Data una classe *C* ed una variabile *var* definita nel corpo di *C*, il campo di visibilità di *var*, detto anche scope, indica in quali porzioni del codice di *C* la variabile *var* può essere richiamata (cioè acceduta)
- Il campo di visibilità di una variabile dipende solo da dove essa è definita, non dal suo contenuto

Variabili di istanza

- Ogni variabile di istanza nasce e muore con l'oggetto al quale si riferisce
- È visibile da tutti i metodi che l'oggetto può eseguire, cioè dai metodi di istanza definiti nella classe
- Una variabile di istanza *x* di un oggetto *o* è accessibile da tutti i metodi di *o*, usando il costrutto *this.x*
- Se in un metodo di *o* non è presente alcuna variabile locale avente lo stesso nome di *x*, allora si può accedere alla variabile di istanza anche direttamente con il suo nome, *x*

Esempi di uso del this

```
/* si può omettere il this */
public void cambiaDimensioni(double b,
                             double a){
    base = b;
    altezza = a;
}
```

```
/* non si può omettere il this */
public void cambiaDimensioni(double base,
                             double altezza){
    this.base = base;
    this.altezza = altezza;
}
```

Variabili di classe

- Ogni variabile di classe nasce e muore con la classe alla quale si riferisce
- È visibile da tutti i metodi definiti nella classe, siano essi statici o di istanza
- Una variabile di classe *y* di una classe *C* è accessibile da tutti i metodi di *C*, usando il costrutto *C.y*
- Se in un metodo di *C* non è presente alcuna variabile locale avente lo stesso nome di *y*, allora si può accedere alla variabile di classe anche direttamente con il suo nome, *y*

Variabili locali

- Le variabili locali sono visibili solo all'interno del metodo in cui sono definite
- Nascono e muoiono in ogni ciclo di esecuzione del metodo stesso
- Le variabili locali non vengono inizializzate con valori di default
- Pertanto bisogna assegnare un valore alle variabili locali prima di tentare di leggerle
  - In caso contrario si ha un errore in fase di compilazione

Variabili locali

- Nel seguente codice la variabile locale *x* non è stata inizializzata
- Pertanto l'espressione *x+1* non può essere valutata correttamente
- Il codice non viene compilato

```
public int metodoErrato (){
    int x;
    x = x+1;
    return x;
}
```

Parametri formali

- I parametri formali sono simili a variabili locali:
  - la loro esistenza e visibilità è limitata al metodo in cui sono definiti
- La differenza è che nel momento in cui un metodo viene attivato ogni suo parametro formale ha già un proprio valore assegnato

Ancora sul this

- Se una classe ha più di un costruttore, la parola chiave *this* può essere anche usata per richiamare un costruttore all'interno di un altro costruttore
- Ad esempio, supponiamo di voler dotare la classe *Punto* di un secondo costruttore

  public Punto(Punto p)
- Tale costruttore crea un punto con le stesse coordinate di quello passato per parametro
- Un costruttore di questo tipo è chiamato costruttore di copia

Ancora sul this

- Il costruttore di copia della classe *Punto* può essere implementato
  - richiamando il costruttore precedentemente definito
  - passando ad esso come parametri *x* ed *y* i valori *p.coordX* e *p.coordY*
- Come si può richiamare tale costruttore?
- Con la sintassi *this(⟨parametri⟩)*
- Dove ⟨*parametri*⟩ sono i parametri richiesti dal costruttore che si vuol richiamare

Ancora sul this

- Il costruttore di copia della classe *Punto* può essere implementato come segue

```
public Punto(Punto p){
    this(p.coordX, p.coordY);
}
```

- Ovviamente potremmo realizzare il costruttore di copia in maniera “tradizionale”

```
public Punto(Punto p){
    this.coordX = p.coordX;
    this.coordY = p.coordY;
}
```

Metodi e costruttori privati

- Negli esempi visti sin qui tutti i metodi delle classi implementate sono stati dichiarati *public*
- In molti casi tuttavia è importante definire metodi *private* a supporto di metodi pubblici
- In particolare, quando un metodo pubblico ha una implementazione complessa, può risultare utile spostare ed incapsulare parti del suo codice in metodi privati che vengono poi richiamati nell'ambito del metodo pubblico.

Metodi e costruttori privati

- È anche possibile definire come *private* un costruttore
- Ma in che circostanza ciò può tornare utile?
- Ad esempio per inibire la creazione di oggetti di una certa classe
- Ad esempio la classe *Math* ha un costruttore privato
  - ciò impedisce la creazione di oggetti *Math*
- La classe *Math* ha solo metodi statici
  - ogni servizio viene offerto dalla classe stessa

Metodi e costruttori privati

- **Nota Bene**: se si vuole inibire la creazione di oggetti non basta non definire alcun costruttore
- In tal caso, infatti, la classe verrebbe comunque dotata del costruttore (pubblico) di default