Definizione di classi

Walter Didimo

Definizione di classi

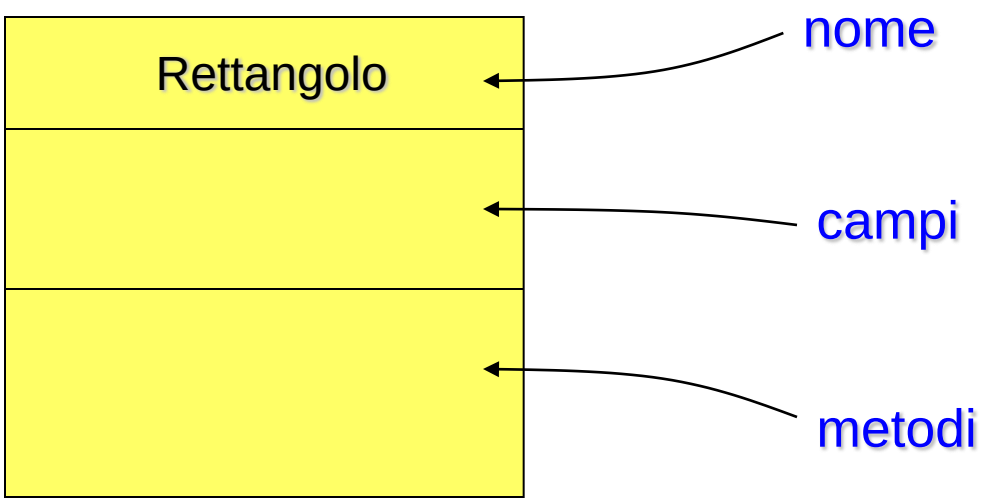
Fino ad ora abbiamo imparato a:

- 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

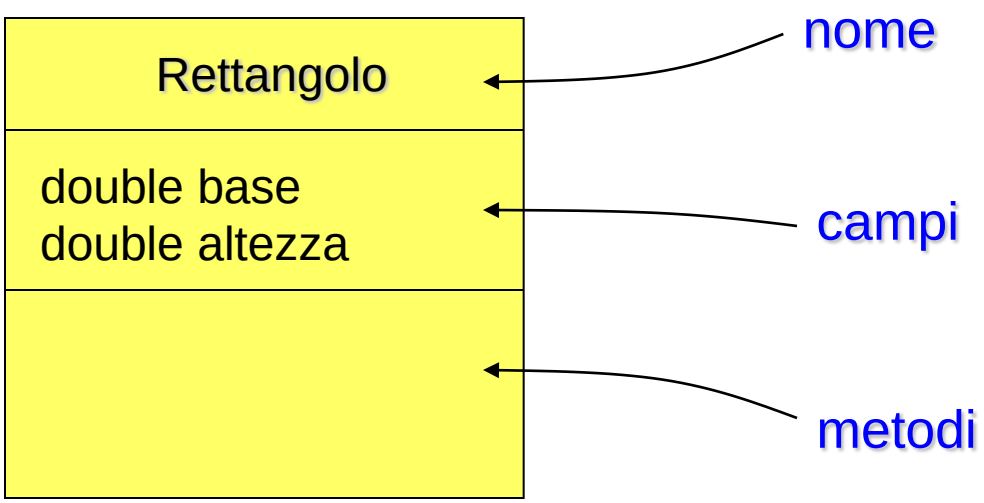
La classe Rettangolo

Inizieremo con un esempio. Vogliamo definire la classe *Rettangolo*, usata in uno dei precedenti esercizi



Oggetti Rettangolo: campi

Un oggetto di tipo *Rettangolo* rappresenta un rettangolo nel piano, ed è definito da una base e da un'altezza, che immaginiamo essere numeri reali

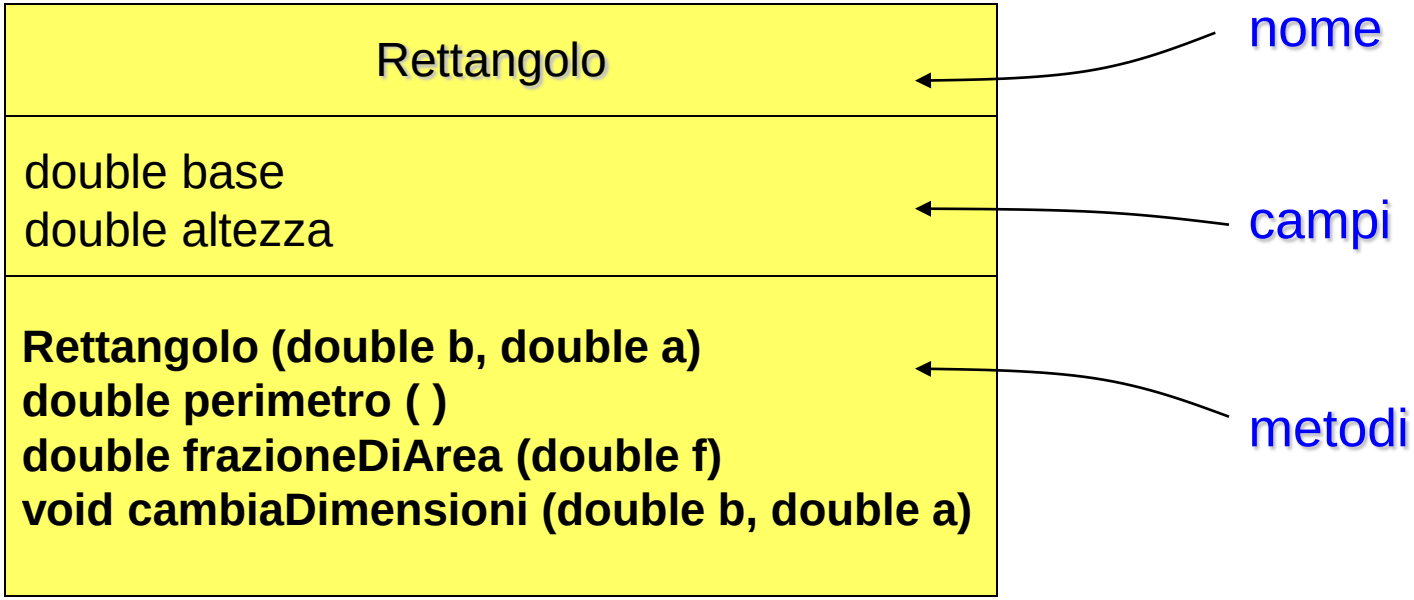


Oggetti Rettangolo: metodi

Un oggetto di tipo *Rettangolo* avrà:

- un costruttore, che permette di specificare base e altezza dell'oggetto creato
- il metodo *double perimetro ()*, che restituisce il perimetro del rettangolo rappresentato dall'oggetto
- il metodo *double frazioneDiArea (double f)*, che restituisce la frazione f dell'area del rettangolo
- il metodo *void cambiaDimensioni (double b, double a)*, che reimposta le dimensioni del rettangolo con b ed a

Oggetti Rettangolo: riassumendo



Campi e variabili

I campi 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, i campi di un oggetto possono essere realizzati attraverso l'uso di opportune variabili, nel nostro caso variabili di tipo *double*

- le variabili usate per definire i campi degli oggetti (istanze) di una classe si chiamano variabili di istanza

Dichiarazione di variabili di istanza

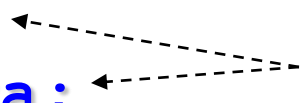
Nella definizione di una classe, le variabili di istanza si dichiarano all'inizio del corpo della classe, prima dei metodi e fuori dei corpi dei metodi

```
class Rettangolo{

   /* base e altezza del rettangolo */
   private double base;
   private double altezza;

   /* costruttori e metodi */
   ……

}
```



La parola private

La dichiarazione di una variabile di istanza avviene come per le variabili viste fino ad ora; si scrive prima il tipo e poi il nome

- tipicamente (ma non sempre) si usa anche inserire una parola speciale all'inizio della dichiarazione, la parola private
- la parola *private* dice che le variabili definite possono essere accedute solo da metodi della classe *Rettangolo*, e non da metodi di altre classi

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!

Semplificazioni sintattiche

Si può anche scrivere

```
class Rettangolo{

   /* base e altezza del rettangolo */
   private double base, altezza;

   /* costruttori e metodi */
   ……

}
```

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

Valore iniziale delle variabili di istanza

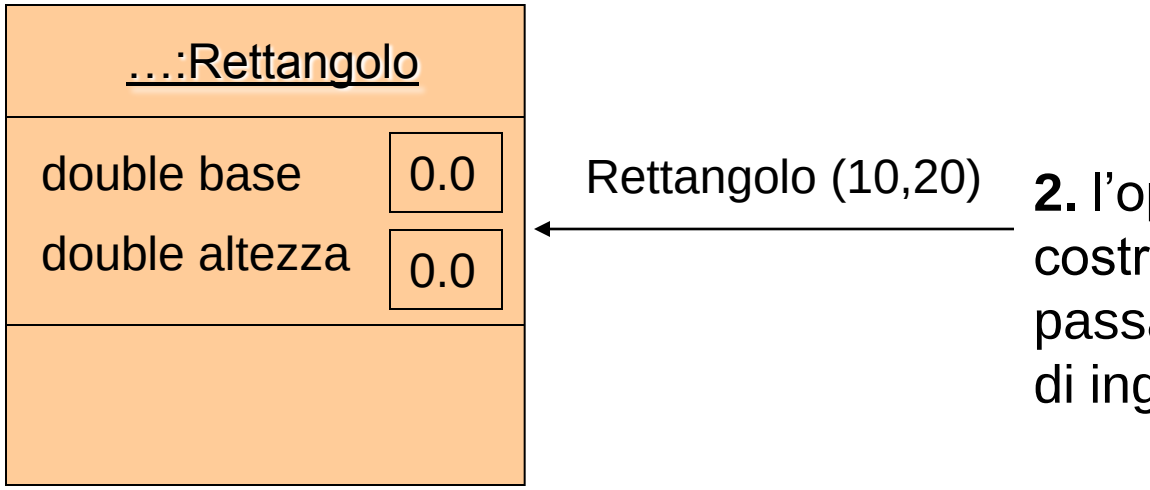
Non appena l'operatore *new* crea un nuovo oggetto *Rettangolo*, che valore hanno le sue variabili di istanza?

- prima che venga eseguito il costruttore, le variabili di istanza hanno un valore di default assegnato dal compilatore (0.0 per variabili di tipo *double*)
- il costruttore ha il compito di ridefinire il valore iniziale delle variabili di istanza, se necessario

Valore iniziale delle variabili di istanza

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

Richiami sui costruttori

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

Definizione di costruttori

Nel corpo di una classe, si usa definire i costruttori dopo le variabili di istanza e prima degli altri metodi

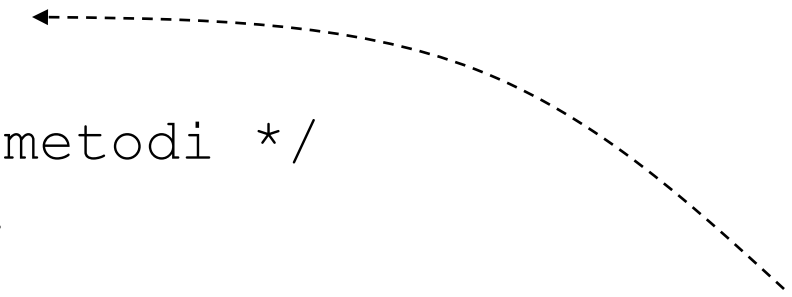
- davanti alla definizione del costruttore si mette di solito la parola speciale *public*
- la parola *public* indica che il costruttore può essere usato per creare oggetti da qualunque altro metodo (per esempio il metodo *main* di una qualche classe)

Definizione di costruttori

```
class Rettangolo{

   /* base e altezza del rettangolo */
   private double base;
   private double altezza;

   /* costruttore che assegna all'oggetto le
      dimensioni specificate */
   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 di *b* e di *a* alle variabili di istanza dell'oggetto sul quale è invocato
- *b* ed *a* si chiamano anche parametri formali
- *b* ed *a* sono variabili usate per contenere i valori che si passeranno al costruttore all'atto della sua invocazione

Parametri attuali

Quando si chiamerà il costruttore, per esempio scrivendo

new Rettangolo (10, 20)

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

- i valori *10* e *20* si chiamano parametri attuali, perché si riferiscono a valori effettivi e non simbolici

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;
}
```

Analizziamo nel seguito le sue istruzioni

La parola chiave “this”

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

Un metodo o un costruttore può riferirsi ad una variabile di istanza dell’oggetto su cui è invocato (oggetto ricevente) 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

Assegnamento a variabili di istanza

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

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*

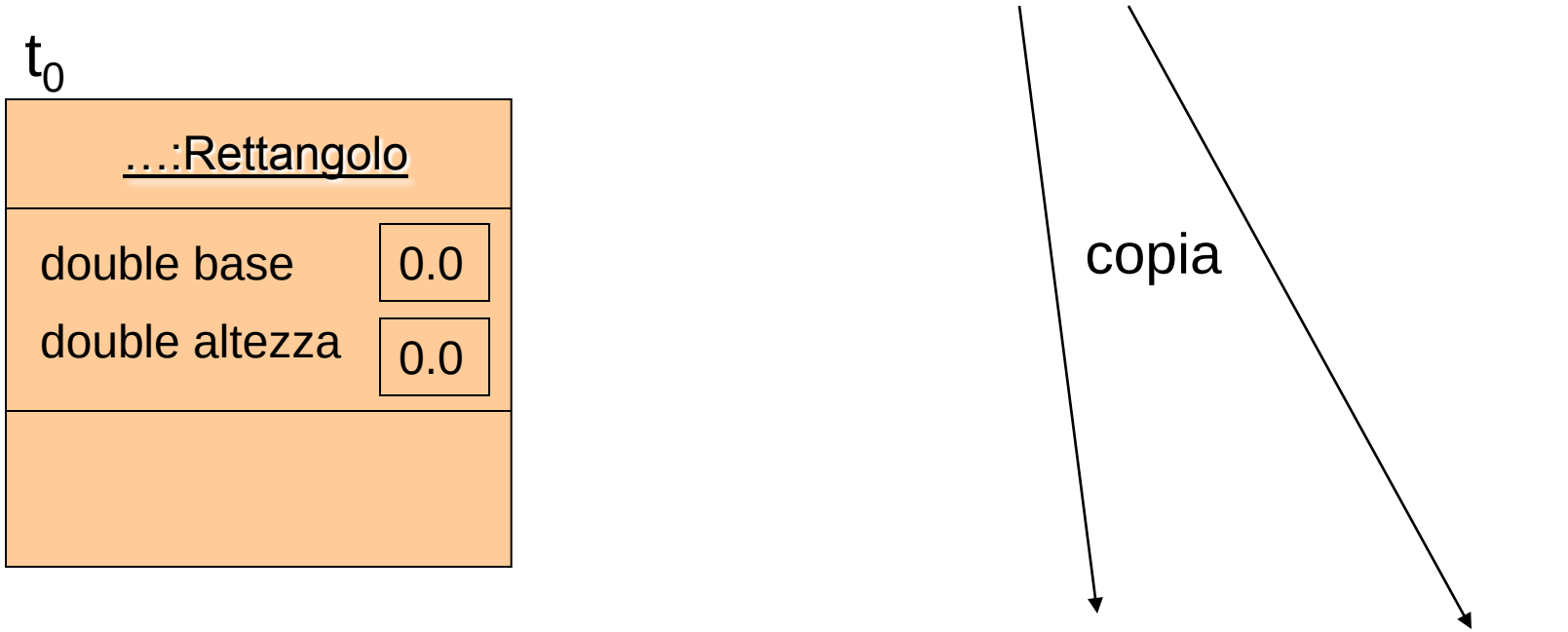
Variabili di istanza e parametri formali

Non c'è alcun legame tra i parametri formali di un metodo e le variabili di istanza di un oggetto:

- ogni metodo di un oggetto può accedere alle variabili di istanza dell'oggetto in qualunque istante
- i parametri formali di un metodo sono solo usati per recepire dei dati passati dall'esterno (cioè all'atto della sua invocazione)
- i parametri formali di un metodo possono solo essere usati da quel metodo – non hanno senso al di fuori del suo corpo, cioè non sono visibili da altri metodi!!

Esecuzione del costruttore

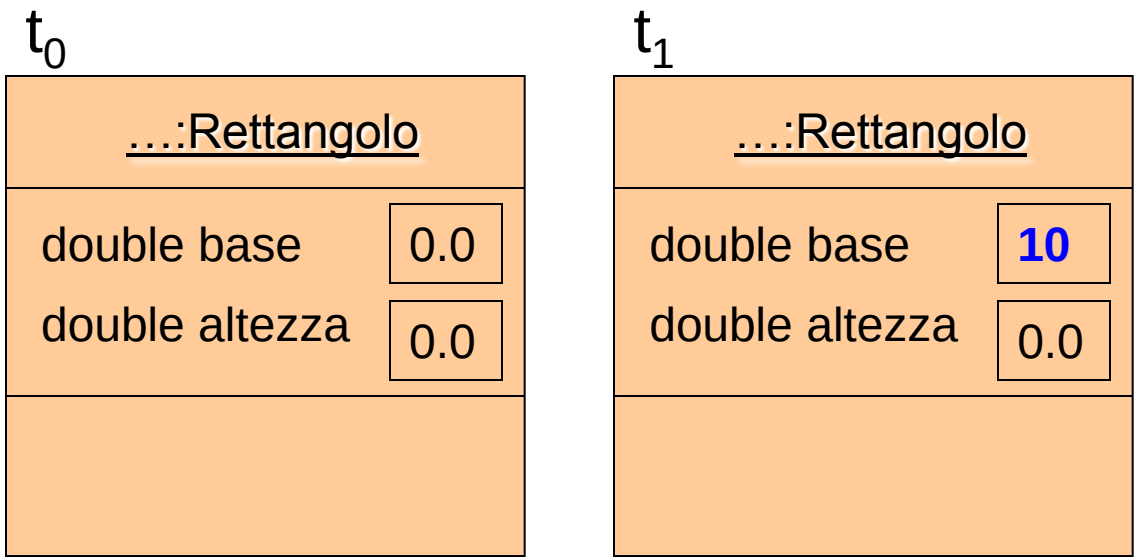
new Rettangolo (10, 20)




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

Esecuzione del costruttore

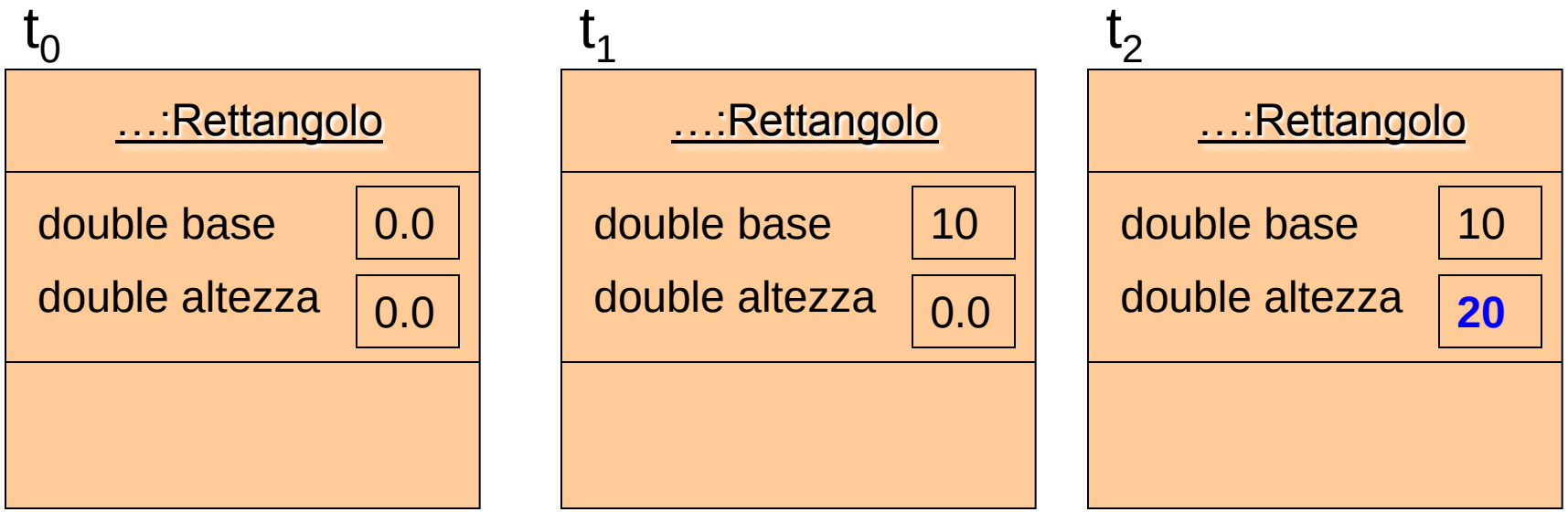
new Rettangolo (10, 20)




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

Esecuzione del costruttore

new Rettangolo (10, 20)




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

Riassumendo

- Il corpo di un metodo viene scritto una volta per tutte dal programmatore!

- L'esecuzione di un metodo varia in funzione dello stato corrente dell'oggetto su cui è invocato e dei parametri attuali ricevuti all'atto dell'invocazione!

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

- le variabili locali sono quelle dichiarate nei metodi e sono visibili solo nei metodi in cui sono dichiarate

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 con 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

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*

Variabili: classificazione

Abbiamo visto che, dal punto di vista del *tipo di dato associato*, le variabili si classificano in:

- variabili riferimento (contengono riferimenti ad oggetti)
- variabili primitive (contengono dati primitivi)

Dal punto di vista di *dove si trovano nel codice*, esse si classificano invece in:

- variabili di istanza (fuori dei metodi)
- variabili locali (dentro ai metodi)

Variabili: classificazione

Le variabili che definiscono i parametri formali di un metodo hanno visibilità solo in quel metodo

Le variabili che definiscono i parametri formali di un metodo sono dunque simili a variabili locali del metodo

Accesso alle variabili locali

Una variabile locale *x* di un metodo *m* è accessibile solo dalle istruzioni di *m*, usando direttamente il suo nome, *x*

- al momento del suo primo accesso, la variabile *x* deve avere già un valore (che potrà eventualmente cambiare)

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

quando si valuta questa espressione, *x* non ha ancora alcun valore – il codice non verrebbe compilato

Accesso alle variabili di istanza

Una variabile di istanza *y* di un oggetto *o* è accessibile da tutti i metodi di *o*, usando il costrutto *this.y*

- se in un metodo di *o* non è presente alcuna variabile locale avente lo stesso nome di *y*, allora si può accedere alla variabile di istanza anche direttamente con il suo nome, *y*

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 e metodi statici

Abbiamo già visto cosa sono e come si usano i metodi statici (anche detti metodi di classe)

Oltre alle variabili di istanza e alle variabili locali, esiste anche il concetto di variabile statica (o variabile di classe)

Vogliamo imparare a definire variabili e metodi statici; partiremo ancora da un esempio concreto di uso

Numero di rettangoli creati

Supponiamo di voler dotare la classe *Rettangolo* di un metodo che mi permetta di sapere (durante l'esecuzione di un programma) quante istanze di tipo *Rettangolo* ho creato fino a quel momento

- questo servizio è offerto direttamente dalla classe, e non da una sua particolare istanza
- il metodo sarà pertanto definito come un metodo statico (cioè di classe) e andrà invocato sulla classe

Metodo che restituisce il numero di rettangoli creati

Sappiamo che un metodo statico va dichiarato con la parola *static* davanti al prototipo

public static int numeroRettangoliCreati ( )

Un metodo statico non può usare variabili di istanza e neanche la parola *this*!!! – l'esecuzione di un metodo statico non è svolta da una istanza ma dalla classe

Il metodo numeroRettangoliCreati

Come fa il metodo *numeroRettangoliCreati* a tenere traccia del numero delle istanze create fino al momento di una sua eventuale invocazione?

- può usare una apposita variabile, chiamiamola *numIstanze*, dichiarata all'atto della definizione della classe e che viene aggiornata durante tutta l'esecuzione del programma
- la variabile *numIstanze* è una variabile gestita dalla classe *Rettangolo*, non da una sua particolare istanza
- essa esiste in unica copia per tutte le istanze

Variabili statiche o di classe

La variabile *numIstanze* viene definita come una variabile statica (o variabile di classe)

- una variabile statica si dichiara come le variabili di istanza (fuori dei metodi)
- la sua dichiarazione è però preceduta dalla parola *static* (come per i metodi statici)

La dichiarazione di numIstanze

```
class Rettangolo{

   /* base e altezza del rettangolo */
   private double base;
   private double altezza;

   /* numero di istanze create */
   private static int numIstanze;

   /* costruttori e metodi */
   ……

}
```

Il codice di numeroRettangoliCreati

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

Una variabile di classe viene acceduta da un metodo usando il nome della classe, un punto, ed il nome della variabile

Rettangolo.numIstanze

Aggiornamento di numIstanze

Ma chi inizializza ed aggiorna il valore di *numIstanze*?

- la variabile dovrebbe avere il valore 0 nel momento in cui la classe è definita
- la variabile va incrementata di una unità ogni volta che viene creato un nuovo oggetto, chiamando il costruttore della classe
  - l'incremento della variabile sarà pertanto eseguito dal costruttore

Inizializzazione di numIstanze

Il valore di default della variabile *numIstanze* è 0 (potremmo evitare di inizializzarla esplicitamente)

è però buona norma farlo comunque – è consentito inizializzare una variabile statica quando si dichiara

```
/* numero di istanze create */
private static int numIstanze = 0;
```

Aggiornamento di numIstanze

Aggiungiamo al corpo del costruttore una istruzione che incrementa il valore di numIstanze

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

Riassumiamo l'intera classe

```
class Rettangolo{

        private double base, altezza;
        private static int numIstanze = 0;

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

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

…… ***(continua)***

Riassumiamo l'intera classe

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

        public void cambiaDimensioni (double nuovaBase,
                                      double nuovaAltezza){
               this.base = nuovaBase;
               this.altezza = nuovaAltezza;
        }

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

}
```

Glossario dei termini principali

Termine	Significato
Variabili di istanza	Variabili usate per implementare i campi degli oggetti di una classe; si dichiarano nel corpo della classe, fuori dei metodi
Variabili statiche (o di classe)	Variabili usate per implementare le proprietà di una classe (non dei suoi oggetti); si dichiarano nel corpo della classe, fuori dei metodi e la dichiarazione è preceduta dalla parola static
private	Parola chiave del linguaggio Java; si mette di solito davanti alla definizione di una variabile di istanza o di classe per indicare che tale variabile è accessibile solo dai metodi della classe; si può anche metter davanti alla dichiarazione di un metodo
public	Parola chiave del linguaggio Java; si mette di solito davanti ai ai costruttori e ai metodi che possono essere invocati pubblicamente (cioè ovunque)
Parametri formali	I parametri di un metodo; usati dal metodo per ottenere dei dati in ingresso
Parametri attuali	Valori passati ad un metodo all'atto della sua invocazione; i parametri attuali vengono copiati nei parametri formali
Passaggio di parametri per valore	Modalità di passaggio di dati in ingresso ad un metodo, in base alla quale i parametri attuali vengono copiati nei parametri formali
this	Parola chiave del linguaggio Java; è usata nei metodi di istanza per denotare l'oggetto su cui il metodo è invocato (quello che esegue il metodo)

Glossario dei termini principali

Termine	Significato
Oggetto ricevente	In una istruzione di invocazione di un metodo di istanza, indica l'oggetto su cui il metodo è invocato
Metodo di accesso	Metodo di istanza che non modifica lo stato dell'oggetto su cui è invocato
Metodo di modifica	Metodo di istanza che può modificare lo stato dell'oggetto su cui è invocato
Variabile locale	Variabile dichiarata all'interno di un metodo; visibile solo nel metodo in cui è dichiarata; i parametri formali sono variabili locali
return	Istruzione Java che serve a restituire un valore; ogni metodo che ha un tipo di ritorno diverso da void deve terminare con una istruzione return