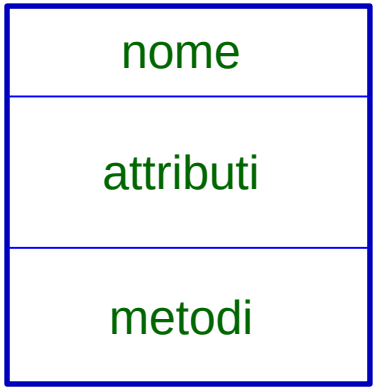
Programmazione Orientata agli Oggetti – Definizioni

Classi e oggetti: introduzione

- Un programma è la definizione di un certo numero di classi. Ogni classe ha un *nome* e descrive le caratteristiche di una tipologia di oggetti, cioè:
  - *come* gli oggetti sono fatti: attributi (o proprietà o campi)
  - *cosa* gli oggetti sanno fare (quali servizi offrono a chi li vuole usare): metodi (l'insieme dei metodi di un oggetto costituisce il suo comportamento)
- Un oggetto è sempre l'istanza di un classe

struttura di una classe



- Lo stato di un oggetto è l'insieme dei valori assunti dai suoi attributi in un determinato istante di tempo (fotografia istantanea)

Nomi di classi, attributi e metodi

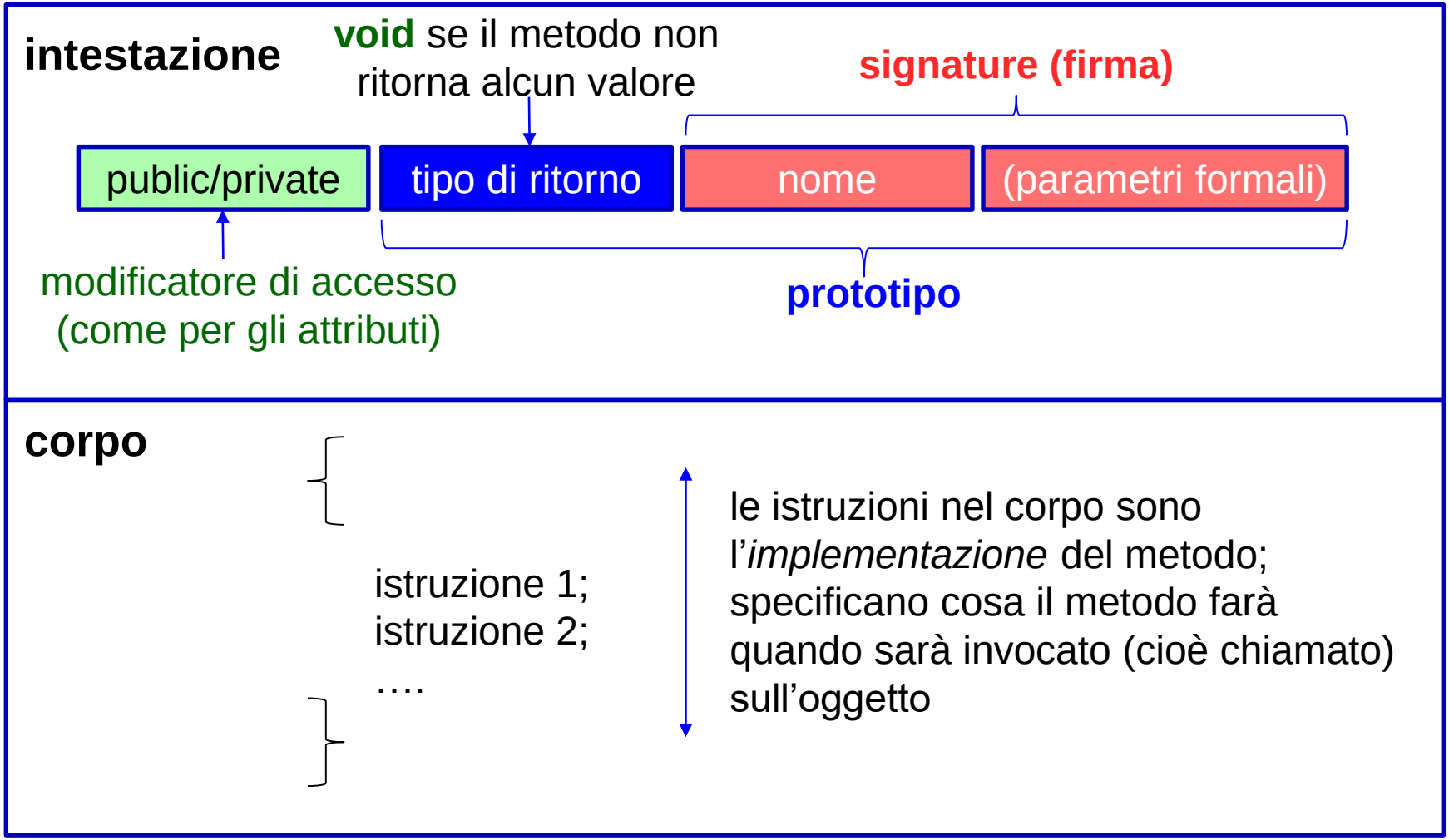
- I nomi di classi, attributi e metodi usano l'alfabeto di simboli Unicode
  - I nomi non possono contenere caratteri speciali (punteggiature, operatori aritmetici, ecc.) e il primo carattere *non* può essere una cifra {0,..,9}
  - per convenzione il primo carattere del nome di una classe è in maiuscolo, mentre quello di attributi e metodi è in minuscolo – inoltre se il nome è formato da più parole, la prima lettera di ogni parola dalla seconda in poi è in maiuscolo

Attributi

- Un attributo è un contenitore (variabile) che può memorizzare valori di un certo tipo:
  - un attributo assume un solo valore alla volta, che può variare molte volte nel tempo, cioè durante il ciclo di vita di un oggetto
  - un attributo è descritto dal tipo di valori (dominio) che può assumere e da un nome che lo identifica
- Esistono due principali categorie di tipi di attributi:
  - primitivi: numeri interi (*byte*, *short*, *int*, *long*), numeri con virgola (*float*, *double*), caratteri (*char*), valori {true,false} (*boolean*)
  - riferimento: per ogni classe di nome *C* esiste un tipo di attributo di nome *C*; un attributo di tipo *C* può memorizzare solo riferimenti (puntatori) ad oggetti di tipo *C*
- In testa alla definizione di un attributo si può specificare un modificatore di accesso:
  - *private*: l'attributo è accessibile solo da codice della classe
  - *public*: l'attributo è accessibile anche da codice esterno

Metodi

- Un metodo è definito da una intestazione e un corpo



Invocazione di metodi

- L'invocazione di un metodo avviene attraverso la sua signature (nome + parametri) – **N.B.** *in una stessa classe non possono esistere due metodi con la stessa signature*
- I parametri formali di un metodo servono a passare al metodo dei valori di input quando lo si invoca; ogni parametro ha un tipo e un nome (come per gli attributi); gli attributi sono separati da virgole
- Se ***ogg*** è il riferimento ad un oggetto si può invocare un metodo di ***ogg*** attraverso la sintassi:

  ogg.<nome-metodo>(<parametri attuali>)

  – I parametri attuali sono una sequenza di valori separati da virgole, un valore per ogni parametro formale del metodo

  esempio di metodo definito in una ipotetica classe *Rettangolo*:
  public void reimpostaDimensioni (int base, int altezza)
  esempio di invocazione del metodo su un oggetto *rett* della classe *Rettangolo*
  rett.reimpostaDimensioni (10,20)

- L'oggetto su cui si invoca un metodo è detto anche oggetto ricevente

Elementi statici e di istanza

- Gli attributi e i metodi di una classe visti sino ad ora si chiamano anche attributi di istanza (o variabili di istanza) e metodi di istanza
- Java (come altri linguaggi di programmazione ad oggetti) permette anche di definire attributi e metodi statici (detti anche di classe)
  - un attributo o un metodo statico si riferisce all'intera classe, *non* ai suoi oggetti (è solo a bordo della classe, non dei suoi oggetti)
  - tutti gli oggetti della classe possono usare attributi o metodi della classe
  - per definire un attributo o un metodo statico occorre mettere la parola chiave ***static*** in testa alla sua definizione, subito dopo il modificatore di accesso
  - se *C* è una classe e ***met(…)*** è un suo metodo statico, si invoca ***met(…)*** con la sintassi ***C.met(…)***

Uso di Classi e Oggetti

Un semplice programma Java

Programma costituito da una sola classe, che scrive «Hello World!» sullo schermo

```
import fond.io.OutputWindow;
public class PrimoProg{
   /* visualizza «Hello World!» */
     public static void main(String[] args){
       OutputWindow out = new OutputWindow();
       out.write("il mio primo programma");
     }
}
```

La parola chiave ***import*** specifica una o più classi che si vogliono utilizzare; la classe ***OutputWindow*** è fornita nel corso, in un ***package*** di nome ***fond.io***. Un suo oggetto rappresenta una finestra grafica

Definizione della classe, tramite la parola chiave ***class***; il nome della classe è PrimoProg; il codice va scritto in un file di nome PrimoProg.java

Crea un oggetto della classe ***OutputWindow*** tramite l'operatore ***new*** e ne assegna il riferimento ad una variabile di nome ***out***

Intestazione del metodo speciale ***main(..)*** – è un metodo pubblico e statico; l'esecuzione di un programma Java inizia sempre con un metodo main (quello della classe di avvio)

commento in Java (verrà ignorato dal compilatore)

Invoca il metodo ***write*** sull'oggetto ***out***. Tale metodo riceve come parametro una stringa (sequenza di caratteri) da visualizzare nella finestra che ***out*** rappresenta

Un secondo programma Java

```
import fond.io.*;

public class Echo{
  /* visualizza una stringa inserita dall'utente */
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    String str = in.readString();
    OutputWindow out = new OutputWindow("Echo");
    out.write(str);
  }
}
```

Il sibolo *****, indica che si vuol poter usare qualsiasi classe nel package ***fond.io***.

La classe ***InputWindow*** è fornita nel corso; un suo oggetto permette di acquisire dall'utente dati immessi tramite la tastiera

Il metodo ***readString()*** degli oggetti ***InputWindow***, apre un pannellino attraverso il quale l'utente può inserire una stringa (ci sono metodi analoghi per acquisire altri tipi di dati – vedi la documentazione del corso); il metodo ritorna il riferimento ad una stringa (oggetto di una classe Java di nome ***String***), che viene memorizzato in una variabile di nome ***str*** **(**e di tipo ***String***)

Il valore della stringa referenziata da ***str*** (cioè la sequenza dei suoi caratteri) viene scritta nella finestra rappresentata dall'oggetto ***out***

Ancora su creazione oggetti

- Gli oggetti in Java vengono creati tramite l'operatore ***new***, seguito da un costruttore:
  - un costruttore di una generica classe C è un «metodo» speciale di C, usato per creare oggetti e inizializzarne lo stato
  - un costruttore ha sempre lo stesso nome della classe in cui è definito
  - un costruttore non restituisce alcun valore (neanche void)
  - una classe può avere più costruttori; poiché tutti hanno lo stesso nome (quello della classe), essi debbono differire per l'elenco dei parametri formali
  - un costruttore che non prende parametri è detto anche costruttore di default; questo perché è implicitamente definito in ogni classe; tuttavia la sua definizione può essere sovrascritta o inibita se si definisce esplicitamente un altro costruttore (vedi più avanti)

Espressioni

Il concetto di espressione

- Una espressione è una istruzione alla quale rimane associato un valore; il tipo di tale valore si chiama tipo dell'espressione; sono esempi di espressione:
  - le invocazione di metodi che restituiscono un valore *non void*
  - le variabili (il cui valore associato è quello che memorizzano); ogni variabile si dichiara con la sintassi ***<tipo> <nome>;***
  - i letterali (es. costanti numeriche, caratteri, boolean, stringhe)
  - Il letterale ***null***, valore che denota un riferimento nullo e che può essere assegnato a qualunque variabile di tipo riferimento
  - espressioni composite, ottenute legando tramite opportuni operatori delle espressioni più elementari; gli operatori ammessi dipendono dal tipo dei dati che formano l'espressione
  - se ***var*** è il nome di una qualche variabile ed ***<esp>*** è una espressione dello stesso tipo di ***var***, si scrive ***var = esp*** per assegnare (memorizzare) a ***var*** il valore di ***esp***

Tipi primitivi in Java

- I tipi primitivi sono tutti e soli quelli elencati in tabella

TIPO PRIMITIVO	DOMINIO VALORI	NUMERO BIT
`byte`	numeri interi nell'intervallo $[-2^7, 2^7 - 1]$	8
`short`	numeri interi nell'intervallo $[-2^{15}, 2^{15} - 1]$	16
`int`	numeri interi nell'intervallo $[-2^{31}, 2^{31} - 1]$	32
`long`	numeri interi nell'intervallo $[-2^{63}, 2^{63} - 1]$	64
`float`	numeri reali tra $\approx 1,4 \cdot 10^{-45}$ e $\approx 3,40 \cdot 10^{38}$	32
`double`	numeri reali tra $\approx 4,9 \cdot 10^{-324}$ e $\approx 1,80 \cdot 10^{308}$	64
`char`	caratteri dell'alfabeto Unicode	16
`boolean`	{true, false}	8

- I letterali numerici interi (es. 10, 200, -3) sono considerati di tipo ***int***
- I letterali numerici con virgola (es. 10.5, -3.4) sono di tipo ***double***
- Se scrivo 20L o 20l, indico il letterale numerico 20 come ***long***
- Se scrivo 10.5F o 10.5f, indico il letterale numero 10.5 come ***float***
- I letterali di tipo char vanno tra apici singoli (es. 'a', 'b', ..)
- I letterali boolean sono le parole chiave ***true*** e ***false***

Espressioni numeriche

- Per comporre espressioni numeriche si possono usare vari operatori, tra cui i classici operatori di addizione (**+**), sottrazione (**-**), moltiplicazione (*****) e divisione (**/**), oltre che parentesi tonde per forzare le precedenze. Se gli operandi sono entrambi interi l'espressione sarà di tipo intero (in tal caso la divisione effettua un troncamento della parte decimale). Se almeno un operando è double (o float), l'espressione sarà un double (o float).
- Se **a** e **b** sono espressioni intere, l'espressione **a%b** fornisce il resto della divisione intera tra **a** e **b** (il segno del resto coincide con quello di **a**)
- Se **a** è una variabile intera, **a++** e **a--** incrementano o decrementano il valore di a di una unità (si può anche usare la notazione **++a** e **--a**)
- Se **x** è una variabile numerica, ed y è una qualunque espressione numerica, scrivere **x += y** equivale a scrivere **x = x+y** (analogamente si può scrivere **x -= y**, **x *= y** e **x /= y**, in luogo di **x = x–y**, **x = x*y** e **x = x/y**
- Gli operatori aritmetici si possono usare anche con operandi ***char*** (gli operandi agiscono sulle posizioni dei ***char*** nell'alfabeto Unicode)

Casting e Overflow

- Sia **a** una variabile di tipo **t1** e sia **b** una espressione di tipo **t2**
  - Se il dominio di valori di **t1** contiene (cioè è più grande di) quello di **t2**, si può sempre effettuare l'istruzione di assegnamento **a = b**;
  - Se invece **t1** è più piccolo del dominio di **t2**, allora l'assegnamento a = b non è consentito (perché potrebbe portare ad una perdita di precisione, un troncamento); si può però scrivere **a = (t1) b,** che equivale a dire che si vuol convertire il valore di **b** nel tipo **t1**, per poter procedere ad assegnarlo ad **a**; tale conversione (detta cast) potrebbe determinare un troncamento del dato; **(t1)** è detto **operatore di cast** verso **t1**.
  - Ad esempio, se a è di tipo **short** e b è di tipo **int**, *non* si può scrivere l'istruzione: **a = b;** si può però scrivere **a = (short)b;**
  - In una assegnazione **a=b** (con o senza cast), se il valore di **b** eccede il massimo valore (in positivo o in negativo) consentito dal dominio del tipo di **a**, si verifica un **overflow**; se **a** è di tipo numerico intero, il valore di **b** viene troncato e viene scritto in **a** un valore errato; se a è un tipo numerico con virgola, viene scritto in a il valore ***Infinity*** o ***-Infinity***

Espressioni boolean

- Ad una espressione ***boolean*** rimane associato il valore ***true*** o il valore ***false.*** Esistono due tipi di operatori per comporre espressioni ***boolean***
  - Operatori logici: si applicano ad operandi ***boolean***
    - **a *&&* b (AND) –** l'espressione è ***true*** se sia **a** che **b** sono ***true***, ed è ***false*** in caso contrario
    - **a || b (OR) –** l'espressione è ***true*** se almeno uno tra **a** e **b** è ***true***, ed è ***false*** in caso contrario
    - **a ^ b (XOR) –** l'espressione è ***true*** se esattamente uno tra **a** e **b** è ***true***, ed è ***false*** in caso contrario
    - **!a b (NOT) –** l'espressione è ***true*** se **a** è ***false*** ed è ***false*** se **a** è ***true***
  - Operatori relazionali: si applicano ad operandi ***numerici***
    - **a > b** è ***true*** se **a** è maggiore di **b**
    - Analogamente abbiamo: **<** (minore), **>=** (maggiore o uguale), **<=** (minore o uguale), **==** (uguale), **!=** (diverso)

Costanti in Java e classe Math

- In Java una costante è una «variabile» alla quale si può assegnare una sola volta il valore (che non potrà dunque più cambiare).
  - si dichiara specificando la parola ***final*** in testa, prima del tipo
  - per convenzione, il nome di una costante è tutto in maiuscolo
  - esesmpio: ***final int PIGRECO = 3.141593***

In Java esiste la classe predefinita ***Math:***

- contiene *solo metodi statici*, ciascuno corrispondente ad una funzione matematica che opera con parametri double e restituisce un double; contiene le costanti ***Math.PI*** (pigrego) e ***Math.E*** (numero di Nepero)

PROTOTIPO DEL METODO	VALORE RESTITUITO
`double abs(double a)`	valore assoluto di `a`
`double sqrt(double a)`	radice quadrata di `a`
`double pow(double a, double b)`	valore della potenza a^b
`double sin(double a)`	seno dell'angolo `a` (in radianti)
`double cos(double a)`	coseno dell'angolo `a` (in radianti)
`double tan(double a)`	tangente dell'angolo `a` (in radianti)
`double log(double a)`	logaritmo in base e di `a`
`double log10(double a)`	logaritmo in base 10 di `a`

// es. calcola area cerchio di raggio r
double area = **Math.pow(r, 2)*Math.PI**

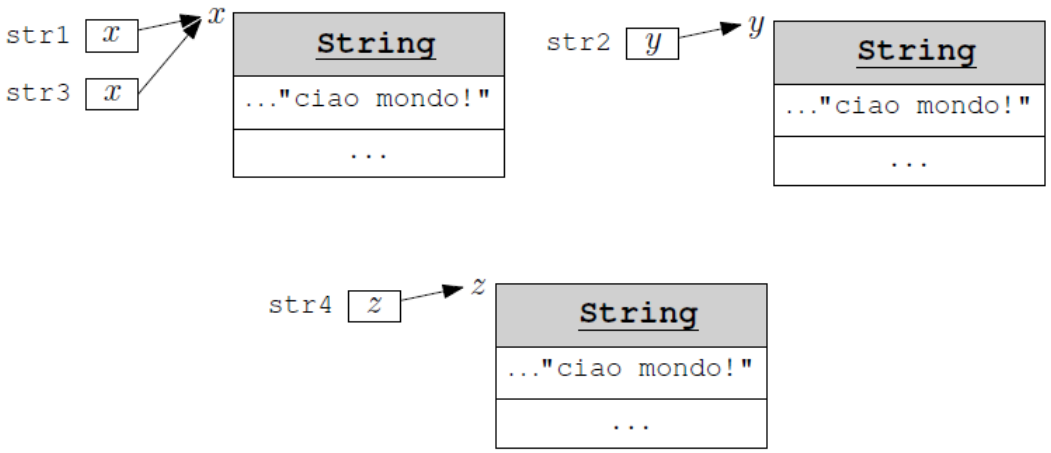
Classe String

- Gli oggetti di tipo *String* rappresentano sequenze di caratteri (cioè sequenze di valori *char*); tale sequenza è detta anche valore dell'oggetto (costituisce lo stato dell'oggetto)
  - gli oggetti *String* sono *immutabili,* cioè il loro valore non può cambiare

```
// esempi di creazione di oggetti String
String str1 = "ciao mondo!";
String str2 = new String("ciao mondo!");
String str3 = "ciao mondo!";
String str4 = new String(str1);
```

letterale String

costruttore della classe String



N.B. due letterali ***String*** dello stesso valore danno luogo allo stesso oggetto; la creazione di una stringa con ***new*** dà sempre luogo ad un nuovo oggetto

Metodi della classe String

La classe ***String*** ha molti metodi di istanza utili; alcuni tra i più usati sono:

- ***int length()*** // restituisce la lunghezza della stringa (numero dei suoi caratteri)
- ***char charAt(int i)*** // restituisce il carattere in posizione i (la prima posizione è 0)
- ***boolean equals (String s)*** // restituisce true se e solo se s ha lo stesso valore dell'oggetto ricevente (quello su cui il metodo equals è invocato)
- ***int compareTo (String s)*** // restituisce un numero negativo, positivo o zero a seconda che il valore dell'oggetto ricevente preceda, segua, o sia uguale a quello di s in base all'ordinamento lessicografico (variante dell'ordinamento alfabetico, derivante dall'ordinamento dei caratteri nell'alfabeto Unicode)
- ***String substring (int i, int j)*** // restituisce la sottostringa dell'oggetto ricevente dalla posizione i (inclusa) alla posizione j (esclusa), o dalla posizione i fino alla fine se il parametro j manca
- ***int indexOf (String s)*** // se s è sottostringa dell'oggetto ricevente, restituisce la posizione di inizio di tale sottostringa, altrimenti restituisce -1
- ***String toUpperCase ()*** // restituisce una stringa equivalente all'oggetto ricevente ma in cui tutte le lettere dell'alfabeto eventualmente presenti sono in maiuscolo
- ***String toLowerCase ()*** // restituisce una stringa equivalente all'oggetto ricevente ma in cui tutte le lettere dell'alfabeto eventualmente presenti sono in minuscolo

Ancora sulla classe String

- L'operatore di concatenazione tra stringhe (simbolo **+**) consente di ottenere un nuovo oggetto ***String***, il cui valore è la concatenazione di quello di altre due. Esempio:

  String s = "ciao " + " mondo!"; *// s ha il valore "ciao mondo!"*

- Si può anche concatenare una stringa ad un valore di tipo primitivo; il risultato è ancora un oggetto ***String*** (il valore primitivo è dapprima «convertito» in stringa). Esempio:

  String s = 10 + " rose"; *// s ha il valore "10 rose"*

- La classe ***String*** ha anche un metodo statico, il metodo ***valueOf***, che consente di ottenere una stringa il cui valore rappresenta un valore di tipo primitivo (passato come parametro). Esempio:

  String s = String.valueOf(100); *// la stringa s vale "100"*

Classi Wrapper

- Per ogni tipo di dato primitivo ***t***, Java offre una classe wrapper (involucro), i cui oggetti «incartano» un valore di tipo ***t,*** che rappresenta il valore dell'oggetto***.*** Le classi wrapper sono: *Byte, Short, Integer, Long, Float, Double, Character, Boolean* che si riferiscono rispettivamente ai tipi primitivi *byte, short, int, long, float, double, char, boolean*

```
Integer ogg = new Integer(100); // crea un oggetto Integer di valore 100
int a = ogg.intValue(); // il metodo intValue() restituisce il valore di ogg
int b = Integer.parseInt("1500") // metodo statico che restituisce un intero
                                 // descritto dal parametro di tipo String
int min = Integer.MIN_VALUE; // costante che equivale al più piccolo int
int max = Integer.MAX_VALUE; // costante che equivale al più grande int
```

- In ogni classe wrapper esistono metodi analoghi a quelli mostrati sopra per la classe ***Integer***. Gli oggetti wrapper sono *immutabili*, come gli oggetto ***String***.

Realizzazione di Classi

Realizzazione di classi

- Nel seguito faremo riferimento alla definizione della seguente 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()`

Attributi e variabili

- Gli attributi di un oggetto *Rettangolo* servono a memorizzare il valore della base e dell'altezza durante tutta la vita dell'oggetto
  - gli attributi di un oggetto vengono realizzati tramite variabili, dette variabili di istanza
- L'attributo di classe *numIstanze* serve a memorizzare il numero di rettangoli creati durante tutta la vita della classe
  - gli attributi di classe vengono realizzati tramite variabili, dette variabili di classe o variabili statiche

```
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 e classe

- Per dichiarare delle variabili di istanza si scrive prima il tipo e poi il nome
  - tipicamente (ma non sempre) la definizione è preceduta dal modificatore d'accesso *private*
  - tale modificatore indica che le variabili possono essere accedute solo da metodi della classe *Rettangolo*
  - in questo modo lo stato degli oggetti può essere modificato solo mediante i metodi pubblici
- 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*

Costruttori

- Un costruttore assegna un valore iniziale alle variabili di istanza
- Un metodo o costruttore può riferirsi ad una variabile di istanza dell'oggetto su cui è invocato usando la parola chiave *this*:
  - *this* indica l'oggetto su cui è invocato il metodo/costruttore
  - *this.base* indica la variabile di istanza *base* di *this*
- Davanti alla definizione del costruttore si mette di solito il modificatore d'accesso *public*
- Se in una classe non si definisce un costruttore, essa viene dotata automaticamente di un costruttore di default senza parametri

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

I valori dei parametri formali *b* ed *a* vengono assegnati alle variabili di istanza dell'oggetto che si sta creando

Si incrementa di uno il valore della variabile *numIstanze*; in queso modo si registra la creazione di un nuovo oggetto

Il metodo 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 delle variabili di istanza

Il valore dell'espressione è poi assegnato alla variabile *p*

- Il metodo è definito *public*; può quindi essere invocato anche all'esterno della classe *Rettangolo*
- La variabile locale *p* di tipo *double* è usata per memorizzare il valore del perimetro
- L'istruzione *return p* termina l'esecuzione del metodo e restituisce il valore *p* a chi lo ha invocato

L’intera classe

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

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

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

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

Istruzioni di Controllo

L'istruzione if-else

- La sintassi dell'istruzione *if-else* è la seguente

  if (⟨*condizione*⟩)

  ⟨*istruzione 1*⟩

  else

  ⟨*istruzione 2*⟩

- ⟨*condizione*⟩ è un'espressione di tipo *boolean* detta condizione dell'*if-else*
- ⟨*istruzione 1*⟩ è una qualunque istruzione Java (anche un blocco) detta corpo dell'*if*
- ⟨*istruzione 2*⟩ è una qualunque istruzione Java (anche un blocco) detta corpo dell'*else*

- Semantica:
  - la ⟨*condizione*⟩ viene valutata
  - se il suo valore è *true* viene eseguito il corpo dell'*if*
  - se il valore della condizione è *false* viene eseguito il corpo dell'*else*

Istruzione if-else: esempio

```
public static int minoreTra(int a, int b){
    int min;
    if (b<a)
          min = b;
    else
          min = a;
    return min;
}
```

- Se *b* è più piccolo di *a*, il valore di *b* viene assegnato a *min*
- Altrimenti a *min* viene assegnato il valore di *a*

Blocchi di istruzioni

- Se nel corpo dell'*if* o nel corpo dell'*else* si ha bisogno di eseguire più istruzioni, è possibile usare un blocco di istruzioni, cioè una sequenza di istruzione racchiuse tra graffe
- Un blocco di istruzioni è una istruzione e può quindi essere utilizzato ovunque ci si aspetti una istruzione
  - ad esempio nel corpo dell'*if* o nel corpo dell'*else*
- È possibile annidare blocchi cioè definire un blocco in un altro blocco
- Le variabili definite in un blocco sono visibili solo in quel blocco e in tutti quelli in esso annidati

L'istruzione while

- La sintassi dell'istruzione *while* è la seguente

  while (⟨*condizione*⟩)
  ⟨*istruzione*⟩

  - ⟨*condizione*⟩ è un'espressione di tipo *boolean* detta condizione del *while*
  - ⟨*istruzione*⟩ è una qualunque istruzione Java (anche un blocco) detta corpo del *while*

- Semantica:
  - la ⟨*condizione*⟩ viene valutata; se è falsa il corpo non viene eseguito e si passa oltre
  - se la ⟨*condizione*⟩ è vera si esegue il corpo del *while*
  - al termine dell'esecuzione del corpo si torna a valutare la ⟨*condizione*⟩ e si itera il comportamento precedente

L'istruzione do-while

- La sintassi dell'istruzione *do-while* è la seguente

  do

  ⟨*istruzione*⟩

  while (⟨*condizione*⟩)

  - ⟨*condizione*⟩ è un'espressione di tipo *boolean* detta condizione del *do-while*
  - ⟨*istruzione*⟩ è una qualunque istruzione Java (anche un blocco) detta corpo del *do-while*

- Semantica:
  - viene eseguito il corpo una prima volta
  - viene poi valutata la ⟨*condizione*⟩
  - se è falsa l'esecuzione del *do-while* termina
  - se invece la ⟨*condizione*⟩ è vera si torna ad eseguire il corpo e si itera il comportamento precedente

L'istruzione for

- La sintassi dell'istruzione *for* è la seguente

  for (⟨*inizializzazione*⟩;⟨*condizione*⟩; ⟨*aggiornamento*⟩)

  ⟨*istruzione*⟩

- Semantica:

1. si esegue l'inizializzazione
2. si valuta la condizione; se essa è falsa l'esecuzione del *for* termina
3. se la condizione è vera viene eseguito il corpo
4. una volta eseguito il corpo viene eseguito l'aggiornamento e si torna al punto 2

- ⟨*inizializzazione*⟩ è un'espressione di qualunque tipo (quasi sempre un'istruzione di assegnazione) detta inizializzazione del *for*
- ⟨*condizione*⟩ è un'espressione di tipo *boolean* detta condizione del *for*
- ⟨*aggiornamento*⟩ è un'espressione di qualunque tipo (quasi sempre un incremento o decremento di una variabile) detta aggiornamento del *for*
- ⟨*istruzione*⟩ è una qualunque istruzione Java (anche un blocco) detta corpo del *for*

Esempi delle istruz. iterative

- I seguenti metodi ricevono come parametro un intero *n* e stampano tutti i numeri pari da *0* a *n*

while

```
public static void stampaPari(int n){
   int i = 0;
   while (i<n+1){
      if (i%2==0)
         System.out.println(i);
      i++;
   }
}
```

do-while

```
public static void stampaPari(int n){
   int i = 0;
   do{
      System.out.println(i);
      i += 2;
   }while (i<n+1);
}
```

for

```
public static void stampaPari(int n){
   int i;
   for(i = 0; i<n+1; i += 2)
      System.out.println(i);
}
```

Array

Gli array

- Un array è una sequenza di variabili:
  - tutte le variabili di un array hanno lo stesso tipo di dato (il tipo dell'array), e si chiamano anche elementi dell'array
  - ogni variabile ha associato un indice (numero intero non negativo)
  - la dimensione (o lunghezza) dell'array è il numero delle sue variabili
- In Java un array è un oggetto, ma
  - non è un'istanza di una classe predefinita
  - non possiede metodi
- Per ogni tipo *T* è automaticamente definito il tipo *T[]* chiamato array di tipo T
- Una variabile di tipo *T[]* può memorizzare un riferimento ad un array il cui tipo è *T*
- Esempio: *String[] s = new String[5];*

crea un array di stringhe di dimensione *5* e memorizza un riferimento ad esso in una variabile *s*

Accesso agli elementi di un array

- L'accesso ad un elemento dell'array avviene specificando un indice compreso tra *0* e *n-1*

  ⟨*riferimento array*⟩ [⟨*espressione*⟩]

String[] s = new String[2]; *s[0] = "alfa";* *s[1] = "beta";*	*int[] a = new int[3];* *a[0] = 100;* *a[1] = 50;* *a[2] = a[0]+a[1];*

- Se tentiamo di accedere ad una posizione non valida, cioè ad un indice minore di *0* o maggiore di *a.length-1* si ottiene un errore del tipo *ArrayIndexOutOfBoundException* in fase di esecuzione

L'attributo length

- Ogni array ha un attributo pubblico di nome *length*, che memorizza la lunghezza dell'array
  - L'attributo è di sola lettura: non è possibile cambiare la dimensione di un array una volta creato
- A cosa serve l'attributo *length*?
  - ad esempio a conoscere la dimensione di un array ricevuto come parametro di un metodo

```
public int sommaElementi(int[] a){
   int somma = 0;
   for (int i = 0; i<a.length; i++)
       somma += a[i];
   return somma;
}
```

Array di array

- Un array i cui elementi sono a loro volta degli array si chiama array di array o array bidimensionale
- La seguente istruzione crea un array bidimensionale di tipo *T* con *m* righe ed *n* colonne

  new T[m][n]
- Il tipo dell'array creato è array di array di tipo T indicato con *T[][]*
- La seguente istruzione crea un array bidimensionale di interi con *m* righe e *n* colonne e ne memorizza il riferimento nella variabile *a*

  int [][] a=new int [m][n];

Accesso agli elementi

- Si accede agli elementi di un array bidimensionale tramite due indici
- Con *a[i][j]* si denota l'elemento di riga *i* e colonna *j* dell'array bidimensionale *a.* Più precisamente:
  - *a[i]* denota l'array monodimensionale corrispondente alla riga *i*
  - *a[i][j]* denota il j-esimo elemento dell'array *a[i]*
- *a[i]* è a tutti gli effetti un riferimento ad un array monodimensionale. Quindi con *a[i].length* si fa riferimento alla sua dimensione

```
public static void stampaElementi(double[][] a){
    for (int i = 0; i<a.length; i++)
      for (int j = 0; j<a[i].length; j++){
        System.out.print("Elem. in posiz. "+"("+i+","+j+"): ");
        System.out.println(a[i][j]);
      }
  }
```

a.length indica la lunghezza dell'array *a*, quindi il numero di righe

a[i].length indica la lunghezza dell'array *a[i]*, quindi il numero di colonne