Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Università degli Studi dell'Aquila

Modulo di Laboratorio di Algoritmi e Strutture Dati

Java: Introduzione – parte I

Fondamenti di Java

Rif: Introduzione a Java, di M.Bertacca e A.Guidi

Prerequisiti del modulo:

- buona conoscenza dei principi della programmazione
- buona conoscenza del linguaggio di programmazione Java; più in dettaglio:
  - variabili ed espressioni
  - strutture di controllo decisionali ed iterative
  - metodi
  - classi

Variabili

- Ogni variabile deve essere **dichiarata**, prima di essere usata, rispettando la seguente sintassi:

```
[public|protected|private] [static]
[final] Tipo identificatore [= value];
```

- Java ha:
  - **Tipi primitivi**
  - **Tipi reference**

Tipi primitivi

- Logici: `boolean`
- Numerici
  - interi: `byte, short, int, long`
  - floating point: `float, double`
- Caratteri: `char`

Una variabile di tipo primitivo può essere utilizzata direttamente (previa inizializzazione) dopo la sua dichiarazione

Promozioni e casting

Se un valore contenuto in una variabile di un certo tipo viene assegnato ad una variabile più capace, Java esegue una conversione automatica (**promozione**)

L'operazione contraria non è automatica e per ovviare a problemi in fase di compilazione è possibile effettuare un **cast**, ossia forzare una variabile di un certo tipo a diventare di un altro tipo.

```
int i=10;  byte b=(byte) i
```

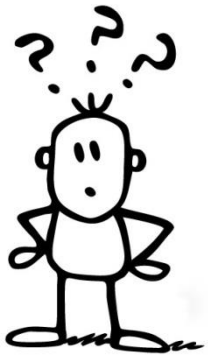
Tipi reference

- Costituiscono dei puntatori o riferimenti (reference) a degli oggetti:
  - Classe

    ```
    String s
    ```
  - Array

    ```
    int[] a1
    String[] s1
    ```
  - **Interfaccia**

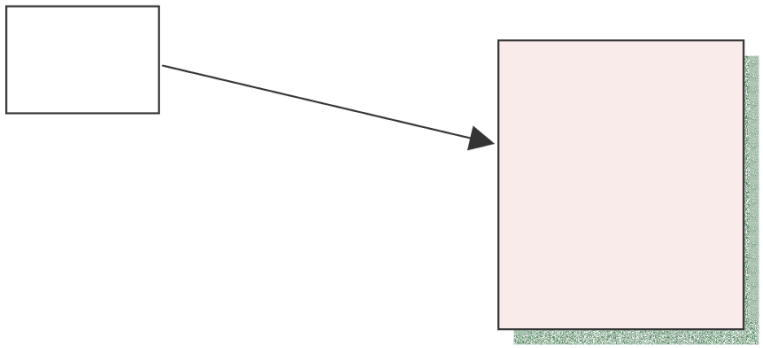
    ```
    Comparable c1
    Comparator c2
    ```



Tipi primitivi e reference

```
int i;
```


```
MiaClasse obj;
```



Array

Un **array** di lunghezza `n` consiste in un insieme di n variabili dello stesso tipo e può essere visto come un tipo derivato

```
<tipo> X[];
// X è una referenza (inizialmente null)
X = new <tipo>[10];
/* new alloca lo spazio necessario, effettua
il collegamento ed inizializza le singole
variabili */
```

Array

- Una variabile array può essere inizializzata in fase di dichiarazione, evitando l'uso dell'operatore `new`

```
// serie di Fibonacci
int X[]= {1,1,2,3,5,8,13,21,34,55};
```

- Una variabile array può essere assegnata ad un'altra del medesimo tipo. Dopo l'assegnamento entrambi gli array permettono l'accesso allo stesso insieme di variabili
- `X.length` consente di conoscere la lunghezza corrente dell'array

Array multidimensionali

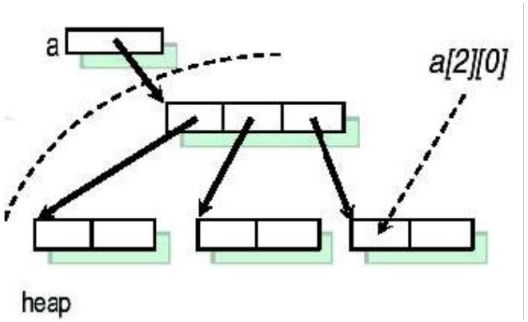
- Il concetto di array (monodimensionale) può essere esteso per rappresentare variabili distinguibili da più di un indice

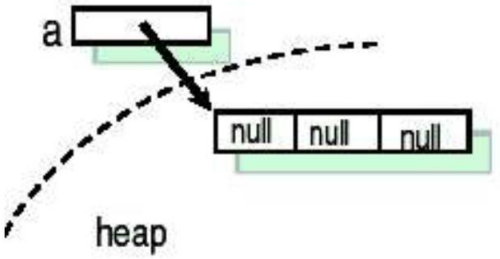
```
int Y[][];
Y=new int[n][m]; //matrice n x m
```

- Y è un array monodimensionale di n elementi che sono a loro volta array di m elementi

Array multidimensionali

- short [ ] [ ] a; /* array di array di short */
- short a[ ] [ ]; /* equivalente */
- a = new short [3][2];
- a = new short [3][ ];





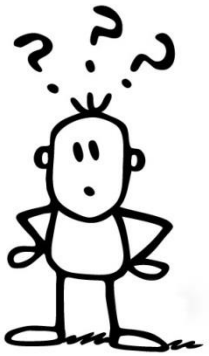
Array incompleti

Esempio:

```
int mmAnno[][]= new int[12][];
mmAnno[0]= new int[31]; // gennaio
mmAnno[1]= new int[29]; // febbraio
mmAnno[2]= new int[31]; // marzo
…
mmAnno[11]= new int[31]; // dicembre
```

Strutture di controllo decisionali

- Costrutto **`if[-else]`**
  - blocchi
  - **`if`** annidati
- Costrutto **`switch`**
- Espressioni condizionate



Costrutto `if[-else]`

- Consente di eseguire `istruzione_1` (o `blocco_1`) o, viceversa, `istruzione_2` (o `blocco_2`), a seconda che condizione logica cui è soggetto risulti o meno verificata

```
if (valore-booleano) istruzione_1;
[else istruzione_2];

if (valore-booleano) { blocco_1 }
[else { blocco_2 }];
```

- È possibile inserire un'istruzione `if[-else]` all'interno di un'altra `if[-else]` (annidamento). In tale caso il ramo `else` si riferisce all'`if` più interno

Costrutto `switch-case`

- Le decisioni a più vie possono essere risolte utilizzando più istruzioni `if-else` in cascata
- In alcuni casi è possibile sostituire le `if-else` in cascata con il più efficiente e leggibile costrutto **`switch-case`**

Sintassi switch-case

```
switch (espressione) {
case costante1:
  [istruzioni1;]
  [break;]
…
case costanteN:
  [istruzioniN;]
  [break;]
[default:
  [istruzioniDefault;]
  [break;]]
}
```

Sintassi `switch-case`

- Il risultato di `(espressione)` deve essere un valore di tipo `byte/char/short/int`
- In fase di esecuzione viene valutata `espressione` ed il risultato viene confrontato con `costante1`;
- Se i valori sono uguali il controllo passa alla prima istruzione del corrispondente `case` e successivamente alle rimanenti istruzioni (incluse quelle del blocco **`default`**), altrimenti si prosegue confrontando il risultato con `costante2`, …
- L'istruzione `break` viene usata per interrompere l'esecuzione del `case` e consente l'uscita immediata

```
switch (mese) {
case 12:
case 1:
case 2: stagione="inverno";
        break;
case 3:
case 4:
case 5: stagione="primavera";
        break;
case 6:
case 7:
case 8: stagione="estate";
        break;
case 9:
case 10:
case 11: stagione="autunno";
         break;
default: //la clausola default è opzionale
        stagione="non identificabile";
}
```

Espressioni condizionate

- Quando si vuole impostare una variabile a uno di due valori è possibile usare l'operatore ternario o condizionale `?` anziché un'istruzione condizionale `if[-else]`
- L'espressione assume la forma:

`espressione_bool ? valore_1 : valore_2`

- Se `espressione_bool` vale `true` viene valutato `valore_1`, altrimenti viene valutato `valore_2` e il risultato diventa il valore prodotto dall'espressione

```
int x = (i<10) ? i*100 : i*10;
```

Strutture di controllo iterative

- Consentono di ripetere una o più istruzioni in base a certe condizioni o per un numero definito di volte:
  - Istruzioni **`while`**
  - Istruzioni do-**`while`**
  - Istruzioni **`for`**
  - Incrementi e decrementi
  - Istruzioni **`break e continue`**

Istruzioni `while` e `do-while`

Sintassi:

```
while (espressione-booleana) istruzione;

do istruzione while (espressione-booleana);
```

- L'istruzione (semplice o composta) viene eseguita fintanto che l'espressione booleana restituisce valore `true`
- Nel caso `do-while` l'istruzione viene eseguita la prima volta prima di valutare l'espressione booleana, per cui viene eseguita almeno una volta.

Istruzione `for`

Sintassi:

```
for (inizializzazione; espressione-bool;
          incremento/decremento) istruzione;
```

```
for (int x=1;x<=10;x=x+1)System.out.println(x);
```

- L'istruzione di `inizializzazione` viene eseguita una sola volta prima di entrare nel ciclo
- La seconda, la condizione di uscita viene valutata prima di ogni iterazione (analogamente al `while`)
- L'istruzione di `incremento/decremento` viene eseguita al termine del corpo del ciclo, prima di una nuova valutazione della condizione d'uscita

Incrementi e decrementi

- `x=x+1;` è equivalente a `++x;` oppure `x++;`
- `x=x-1;` è equivalente a `--x;` oppure `x--;`
- La differenza tra gli operatori preposti e posposti è avvertibile quando sono usati all'interno di espressioni:
  - gli operatori prefissi modificano il valore della variabile cui sono applicati prima che se ne utilizzi il valore.
  - gli operatori post-fissi modificano il valore della variabile dopo l'utilizzo del valore (vecchio) nell'espressione

Incrementi e decrementi: esempio

```
int x=4, y=3;
System.out.println("x = " + x);
System.out.println("y = " + y);
System.out.println("--x + y  vale: " + (--x + y));

System.out.println("x = " + x);
System.out.println("y = " + y);
System.out.println("x++ + y  vale: " + (x++ + y));

System.out.println("x = " + x);
System.out.println("y = " + y);
```

Chiaramente X++ +y è diverso da x+ ++y

Attenzione all'uso degli spazi:
x+++y viene interpretato come x+y !

L'istruzione `break`

- L'istruzione `break` viene usata:
  1. per forzare la terminazione di un'iterazione `while`, `do-while`, `for`, provocando un salto alla prima istruzione successiva al ciclo;
  2. per interrompere l'esecuzione di un'istruzione qualsiasi, purché sia identificata da un nome (uso avanzato, da evitare)

L'istruzione break (con identificatore)

```
char a[]={'a','b','c','?','d','e','f'};
interruzione:
for (int x=0; x<a.length; x++) {
  switch (a[x]) {
      case '.':
      case ',':
      …
      case '?':
         break interruzione;
      default: break;
   }
   System.out.println(a[x]);
}
System.out.println("fine");
```

L'istruzione `continue`

- L'istruzione `continue` usata all'interno di un ciclo provoca il passaggio immediato alla successiva iterazione

```
int ai[]={10,-10,5,-5,7,-7};
int somma=0;
for (int x=0; x<ai.length; x++){
  if (ai[x]<=0) continue;
  somma+=ai[x];
}
System.out.println(somma);
```

I metodi: forma generale

```
<modificatore> <tipo-ritorno>
  identificatore-metodo ([tipo1 par1 [,
  tipo2 par2… [, tipoN parN]]])
{[<istruzioni>]}
```

- I valori in ingresso completi di tipo sono detti **parametri formali.** Si definisce **firma** o **segnatura** del metodo l'insieme "nome più parametri formali"
- La firma identifica univocamente un metodo all'interno di una classe
  - pertanto possono esistere all'interno della stessa classe due metodi con il medesimo nome, a patto che abbiano parametri formali diversi

I metodi: overloading

- Una classe può contenere due o più metodi con:
  - stesso nome
  - diversa segnatura
  - tipo diverso in almeno un parametro se hanno lo stesso numero
  - numero di parametri diversi
  - il tipo di ritorno non è considerato
- Ciascun metodo ridefinito deve avere una lista univoca di tipi degli argomenti
  - sono sufficienti le differenze nell'ordine degli argomenti per distinguere fra due metodi (questo approccio genera codice difficile da gestire)

I metodi: overloading

- Esempio:

```
public void print(int i)
public void print(float f)
public void print(String s)
public void print(String s, int x)
public void print(int x, String s)
```

Variabili locali: visibilità

- Un identificatore (nome simbolico) dichiarato all'interno di un modulo o blocco, detto nome **locale**, ha visibilità estesa dal punto di dichiarazione alla fine del blocco in cui è contenuto
- I parametri formali di un metodo sono da considerarsi variabili locali alla funzione
- Una dichiarazione di variabile in un blocco non può avere lo stesso nome di una variabile locale dichiarata in un blocco più esterno, perché quest'ultima è visibile nel blocco più interno
- Metodi diversi possono avere variabili con lo stesso nome

Visibilità: esempi

```
static void f() {
   int x=1;
    { int x=2; //errore
      System.out.println(x); }
   System.out.println(x);
}

static void g() {
   { int x=2; }
   System.out.println(x); //errore
}
```

Invocazione di un metodo

- Un metodo viene invocato facendo riferimento al nome, e passandogli una lista di parametri (**parametri attuali**) conforme in tipo, numero e ordine alla lista dei parametri formali elencata nella definizione del metodo
- All'interno di un metodo si possono chiamare altri metodi. Quando un metodo invoca se stesso si parla di **ricorsione**

Oggetti

- I programmi non sono costituiti solo da dati primitivi , ma da anche da dati molto più complessi con proprietà e funzioni proprie:
  - questi dati sono rappresentati dagli oggetti
- Un **oggetto** è una entità che il programmatore può manipolare all'interno del programma mediante l'invocazione di metodi
- Quando è invocato un metodo di un oggetto, vengono svolte all'interno dell'oggetto una serie di attività con lo scopo di realizzare la funzione per cui il metodo è stato implementato
- Oggetti differenti forniscono serie diverse di metodi

Esempio

- L'istruzione

  `System.out.println("Hello World!!");`

  chiama il metodo `println` dell'`oggetto System.out`, che è fornito nella libreria standard del JAVA (JDK). Questo metodo stampa a video la stringa che riceve in ingresso.
- Tale metodo però è proprio dell'oggetto `System.out` e se proviamo a chiamare `println()` su di un oggetto diverso, il compilatore ritorna errore

Le classi

"Gruppo di cose o individui caratterizzati da medesime qualità e/o comportamenti"

- Le **classi** sono utilizzate per modellare **nuovi tipi di dato astratto**: il nome della classe rappresenta un nuovo tipo di dato
- Descrivere una classe significa descrivere in modo astratto :
  - le qualità di un insieme di oggetti
  - i comportamenti di un insieme di oggetti senza fare riferimento all'oggetto singolo
  - il livello di protezione (pubblico: visibile anche dall'esterno, privato: visibile solo entro la classe, ecc…)

Classe e oggetti

Ma che differenza c'è tra Classe ed oggetto ?

La stessa differenza che c'è tra tipo di dato e dato.

- **Classe**: descrizione astratta di un oggetto
- **Oggetto**: istanza di una classe
  - un oggetto è una coppia (stato, funzioni)

ES.: Classe Automobile

- Oggetto Berlina
- Oggetto Monovolume

ATTENZIONE: Un'istanza non può esistere senza la sua Classe di appartenenza!!!

Classe e oggetti

Nella classe definiamo:

- **Le variabili/attributi**: consentono di memorizzare le informazioni di ciascuna istanza (cioè lo stato dell'oggetto).
- **I costruttori**: specificano come inizializzare lo stato di una nuova istanza.
- **I metodi**: specificano le operazioni che determinano il comportamento delle istanze.

Classe

```
[ public | abstract | final ]
            class <nome della classe>
                  [extends Tipo]
                  [implements ListaTipi] {


[<dichiarazione di attributi>]

[<dichiarazione dei costruttori>]

[<dichiarazione dei metodi>]


}
```

Corpo della classe

Costruttori

- I metodi costruttori si distinguono dagli altri metodi:
  - il loro nome coincide con quello della classe
  - non dichiarano nessun tipo di dato da restituire.

```
<modificatore> nome-classe ([tipo1 par1 [,
  tipo2 par2… [, tipoN parN]]])
{[<istruzioni>]}
```

```
public class Persona {
     private String nome;
     public Persona(String n) {
       nome = n;
     }
}
```

Costruttori

- I costruttori inizializzano lo stato di un oggetto
  - Memorizzano i valori iniziali negli attributi;
  - A tal fine spesso ricevono parametri dall'esterno.
- Se non si specificano costruttori, viene definito automaticamente un "costruttore di default".
- Una volta definito un costruttore, il compilatore non permette più di creare un'istanza della classe senza passare gli argomenti, a meno di non definire un costruttore senza argomenti.
  - Questo comportamento consente di avere delle classi in cui non è permesso creare un oggetto senza fornire dei parametri al costruttore.

Le sottoclassi

- Un classe può essere composta da più **sottoclassi** riconoscibili dall'uso della parola chiave **`extends`**.
- una sottoclasse è una classe che eredita tutte le proprietà da una superclasse.

Esempio: la classe dei triangoli è composta dalle sottoclassi dei triangoli equilateri, triangoli isosceli e triangoli scaleni.

- Se una classe ha uno o più costruttori, uno di essi deve essere invocato necessariamente anche quando viene creata un'istanza di una sottoclasse.

Modificatori fondamentali

"*un modificatore sta ad un componente di un'applicazione Java come un aggettivo sta ad un sostantivo nel linguaggio umano*"

- Un **modificatore** è una parola chiave capace di cambiare il significato di un componente di un'applicazione Java
- Si possono anteporre alla dichiarazione di un componente di un'applicazione Java anche più modificatori alla volta, senza tener conto dell'ordine in cui vengono anteposti
  - Una variabile dichiarata `static public` avrà le stesse proprietà di una dichiarata `public static`.

Modificatori fondamentali

MODIFICATORE	CLASSE	ATTRIBUTO	METODO	COSTRUTTORE	BLOCCO DI CODICE
public	sì	sì	sì	sì	no
protected	no	sì	sì	sì	no
(default)	sì	sì	sì	sì	sì
private	no	sì	sì	sì	no
abstract	sì	no	sì	no	no
final	sì	sì	sì	no	no
native	no	no	sì	no	no
static	no	sì	sì	no	sì
strictfp	sì	no	sì	no	no
synchronized	no	no	sì	no	no
transient	no	sì	no	no	no

Modificatori d'accesso

I modificatori di accesso regolano la visibilità e l'accesso ad un componente Java:

- **`public:`** Un membro (**attributo** o **metodo**) di una classe dichiarato pubblico sarà accessibile da una qualsiasi classe situata in qualsiasi package.
  - Un **package** permette di raggruppare in un'unica entità complessa classi Java logicamente correlate
- Una **classe** dichiarata pubblica sarà anch'essa visibile da un qualsiasi package.

Modificatori d'accesso

- **`protected`**: Questo modificatore definisce per un membro il grado più accessibile dopo quello definito da **`public`**.
- Un membro protetto sarà infatti accessibile all'interno dello stesso package ed in tutte le sottoclassi della classe in cui è definito, anche se non appartenenti allo stesso package.

Modificatori d'accesso

Attenzione a `protected` !

- Molti programmatori preferiscono ereditare nelle sottoclassi direttamente una variabile `protected` piuttosto che lasciarla `private`, ed ereditarne i metodi "set "e "get" pubblici
- In realtà nella maggior parte dei casi non c'è una vera necessità di utilizzare questo modificatore

Modificatori d'accesso

L'utilizzo del modificatore `protected` implica anche una strana limitazione:

- se una classe A definisce un metodo m() protetto, un'eventuale sottoclasse B appartenente ad un package diverso da quello di A, può accedere al metodo della superclasse mediante il reference **`super`**, ma non può invocare tale metodo su altre istanze di altre classi che non siano di tipo B

```
package package1;

public class Superclasse {
  protected void metodo(){

  }
}
```

```
package package2;

import package1.*;

public class Sottoclasse extends Superclasse {
  protected void metodo(){

  }

  public void chiamaMetodoValido1(){
    super.metodo();
  }

  public void chiamaMetodoValido2(Sottoclasse oggetto){
    oggetto.metodo();
  }

  public void chiamaMetodoNonValido(Superclasse oggetto){
    oggetto.metodo();
  }
}
```

```
metodo() has protected access in package1.Superclasse
    oggetto.metodo();
            ^
1 error
```

Modificatori d'accesso

- **`default`**: Possiamo evitare di usare modificatori sia relativamente ad un membro (attributo o metodo) di una classe, sia relativamente ad una classe stessa.
- Se non anteponiamo modificatori d'accesso ad un membro di una classe, esso sarà accessibile solo da classi appartenenti al package dove è definito.
- Se dichiariamo una classe appartenente ad un package senza anteporre alla sua definizione il modificatore public, la classe stessa sarà visibile solo dalle classi appartenenti allo stesso package.

Modificatori d'accesso

- **`private`**: Questo modificatore restringe la visibilità di un membro di una classe alla classe stessa
  - Osservazione: Due oggetti istanziati dalla stessa classe possono accedere in "modo pubblico" ai rispettivi membri privati.
  - In rif al seguente esempio, nel metodo `getDifferenzaAnni()` si accede direttamente alla variabile `anni` dell'oggetto `altro`, senza usare il metodo `getAnni()`
  - Sebbene il codice seguente sia valido per la compilazione, l'uso del metodo `getAnni()` favorirebbe sicuramente il riuso di codice, e quindi è da considerarsi preferibile. Infatti, `getAnni()` potrebbe evolvere introducendo controlli, che conviene richiamare piuttosto che riscrivere.

Modificatori d'accesso

```
public class Dipendente  {
    private String nome;
    private int anni; //intendiamo età in anni
    . . .
    public String getNome() {
        return nome;
    }
    public void setNome(String n) {
        nome = n;
    }
    public int getAnni() {
        return anni;
    }
    public void setAnni(int n) {
        anni = n;
    }
    public int getDifferenzaAnni(Dipendente altro) {
        return (anni - altro.anni);
    }
}
```

Modificatori di accesso: Riassunto

MODIFICATORE	STESSA CLASSE	STESSO PACKAGE	SOTTOCLASSE	DAPPERTUTTO
public	sì	sì	sì	sì
protected	sì	sì	sì	no
(default)	sì	sì	no	no
private	sì	no	no	no

Il modificatore `final`

- È applicabile a variabili, metodi e classi
- Potremmo tradurre il termine `final` con “finale”, nel senso di “non modificabile”. Infatti:
  - una **variabile** dichiarata `final` diviene una costante
  - un **metodo** dichiarato `final` non può essere riscritto in una sottoclasse (non è possibile applicare l’override)
  - una **classe** dichiarata `final` non può essere estesa
- Il modificatore `final` si può utilizzare anche per variabili locali e parametri locali di metodi. In tali casi, ovviamente, i valori di tali variabili non saranno modificabili localmente.

Il modificatore `Static`

- `static` è forse il più potente modificatore di Java.
- Con `static` la programmazione ad oggetti trova un punto di incontro con quella strutturata ed il suo uso deve essere quindi limitato a situazioni di reale e concreta utilità.
- Potremmo tradurre il termine `static` con "**condiviso da tutte le istanze della classe**", oppure "della classe".

Il modificatore `Static`

- Un membro statico ha la caratteristica di poter essere utilizzato mediante una sintassi del tipo:

```
NomeClasse.nomeMembro
```

- in luogo di:

```
nomeOggetto.nomeMembro
```

- Anche senza istanziare la classe, l'utilizzo di un membro statico provocherà il caricamento in memoria della classe contenente il membro in questione, che quindi, condividerà il ciclo di vita con quello della classe.

Metodi statici

- Un esempio di metodo statico è il metodo `sqrt()` della classe `Math`, che viene chiamato tramite la sintassi:

```
Math.sqrt(numero)
```

- `Math` è quindi il nome della classe e non il nome di un'istanza di quella classe.
- La ragione per cui la classe `Math` dichiara tutti i suoi metodi statici è facilmente comprensibile. Infatti, se istanziassimo due oggetti differenti dalla classe Math, `ogg1` e `ogg2`, i due comandi:

```
ogg1.sqrt(4);   e   ogg2.sqrt(4);
```

produrrebbero esattamente lo stesso risultato (2).

- Effettivamente non ha senso istanziare due oggetti di tipo matematica, che come si sa è unica.

Il metodo main()

- Un particolare metodo **`static`** è il metodo **`main()`**
- È il punto d'accesso di un'applicazione Java.
- Almeno una classe di un applicazione Java deve contenere il metodo **`main`**()

```
public static void main(String args[])
```

Variabili statiche

- Una variabile statica, essendo condivisa da tutte le istanze della classe, assumerà lo stesso valore per ogni oggetto di una classe.
- Di seguito viene presentato un esempio:

Variabili statiche

```
public class ClasseDiEsempio
{
    public static int a = 0;
}
public class ClasseDiEsempioPrincipale
{
    public static void main (String args[])
    {
        System.out.println("a = "+ClasseDiEsempio.a);
        ClasseDiEsempio ogg1 = new ClasseDiEsempio();
        ClasseDiEsempio ogg2 = new ClasseDiEsempio();
        ogg1.a = 10;
        System.out.println("ogg1.a = " + ogg1.a);
        System.out.println("ogg2.a = " + ogg2.a);
        ogg2.a=20;
        System.out.println("ogg1.a = " + ogg1.a);
        System.out.println("ogg2.a = " + ogg2.a);
    }
}
```

Variabili statiche

- L'output sarà:

```
a = 0
ogg1.a = 10
ogg2.a = 10
ogg1.a = 20
ogg2.a = 20
```

- se un'istanza modifica la variabile statica, essa risulterà modificata anche relativamente all'altra istanza. Infatti essa è condivisa dalle due istanze ed in realtà risiede nella classe.
- **Esempio d'uso**:conteggio del numero di oggetti istanziati da una classe (per esempio incrementandola in un costruttore).

Esempio

```
public class Counter {
  private static int counter = 0;
  private int number;
  public Counter() {
    counter++;
    setNumber(counter);
  }
  public void setNumber(int number) {
    this.number = number;
  }
  public int getNumber() {
    return number;
  }
}
```

Esempio

1) `Counter c1 = new Counter();`
2) `Counter c2 = new Counter();`

Inizialmente: `static counter ←0`

1) `static counter ←1`
   `c1.number ←1`
1) `static counter ←2`
   `c2.number ←2`

- Il modificatore `static` prescinde dal concetto di oggetto e lega strettamente le variabili al concetto di classe, che a sua volta si innalza a qualcosa più di un semplice mezzo per definire oggetti
  - `c1.number=1; c1.counter=c1.counter=2;`

Variabili in Java

In Java esistono dunque tre tipi di variabili:

- **variabili di istanza:** definiscono gli attributi e lo stato di un oggetto; persistono per tutto il tempo di vita dell'oggetto
- **variabili di classe:** definiscono gli attributi e lo stato di una classe (si dichiarano **`static`**)
- **variabili locali:** utilizzate all'interno dei metodi per memorizzare informazioni utili alla esecuzione del metodo stesso; sono create nel momento in cui viene creato il metodo e sono distrutte quando il metodo termina.
- ATTENZIONE: variabili di istanza≠ variabili locali!!!

Inizializzatori statici

- Il modificatore `static` può anche essere utilizzato per marcare un semplice blocco di codice, che viene a sua volta ribattezzato **inizializzatore statico.** Questo blocco, come nel caso dei metodi statici, potrà utilizzare variabili definite fuori da esso se e solo se dichiarate statiche.
- In pratica un blocco statico definito all'interno di una classe avrà la caratteristica di essere chiamato al momento del caricamento in memoria della classe stessa, addirittura prima di un eventuale costruttore.

Inizializzatori statici

```
public class EsempioStatico
{
    private static int a = 10;
    public EsempioStatico()
    {
        a += 10;
    }
    static
    {
        System.out.println("valore statico = " + a);
    }
}
```

- Istanziamo con la seguente sintassi:

  `EsempioStatico ogg = new EsempioStatico();`

- Output: `valore statico = 10`

Inizializzatori

- È possibile inserire in una classe anche più di un inizializzatore statico.
- In caso, questi verranno eseguiti in maniera sequenziale “dall’alto in basso”.

Inizializzatori d'istanza

- Esiste anche un'altra tipologia di inizializzatore, ma non statico.
- Si tratta dell'**inizializzatore d'istanza (instance initializer o object initializer)** e si implementa includendo codice in un blocco di parentesi graffe all'interno di una classe.
- La sua caratteristica è l'essere eseguito quando viene istanziato un oggetto, prima del costruttore.

Inizializzatori d'istanza

```
public class InstanceInitializer {
    public InstanceInitializer() {
        System.out.println("Costruttore");
    }
    {
        System.out.println("Inizializzatore");
    }
}
```

- Per esempio, se istanziassimo la precedente classe l'output risultante sarebbe il seguente:

```
Inizializzatore
Costruttore
```