



OBIETTIVI:

1. Abilità nella produzione di codice sorgente sintatticamente e logicamente corretto;
2. Gestione efficace degli ambienti di sviluppo e debug

CU9 : “JAVA BASE”

- JAVA base: Cos'è Java

- * **Linguaggio di programmazione:**

- * Object Oriented
 - * Portabile e sicuro

- * **Tre piattaforme:**

- * J SE - Java Standard Edition
 - * J EE - Java Enterprise Edition
 - * J ME - Java Micro Edition

- JAVA base: La Storia - un linguaggio di ultima generazione

1 Dall'Assembler
a JAVA
passando da Cobol, C

2 Dalla sintassi del C
alla filosofia ad oggetti di C++



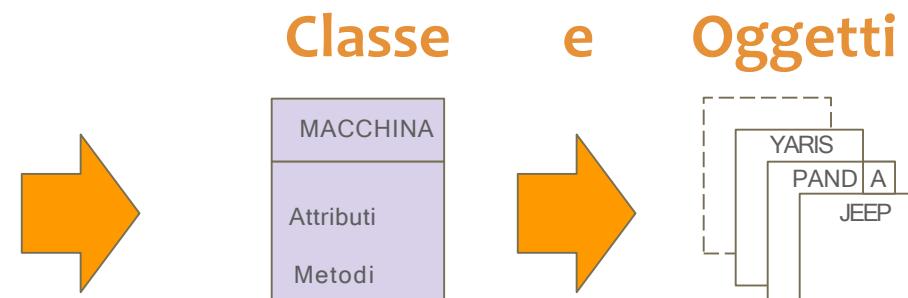
3 Dalla Programmazione
procedurale all'Object Oriented
Programming (OOP)

4 JVM
Java Virtual Machine
(esegue Bytecode)

• JAVA base: OOP

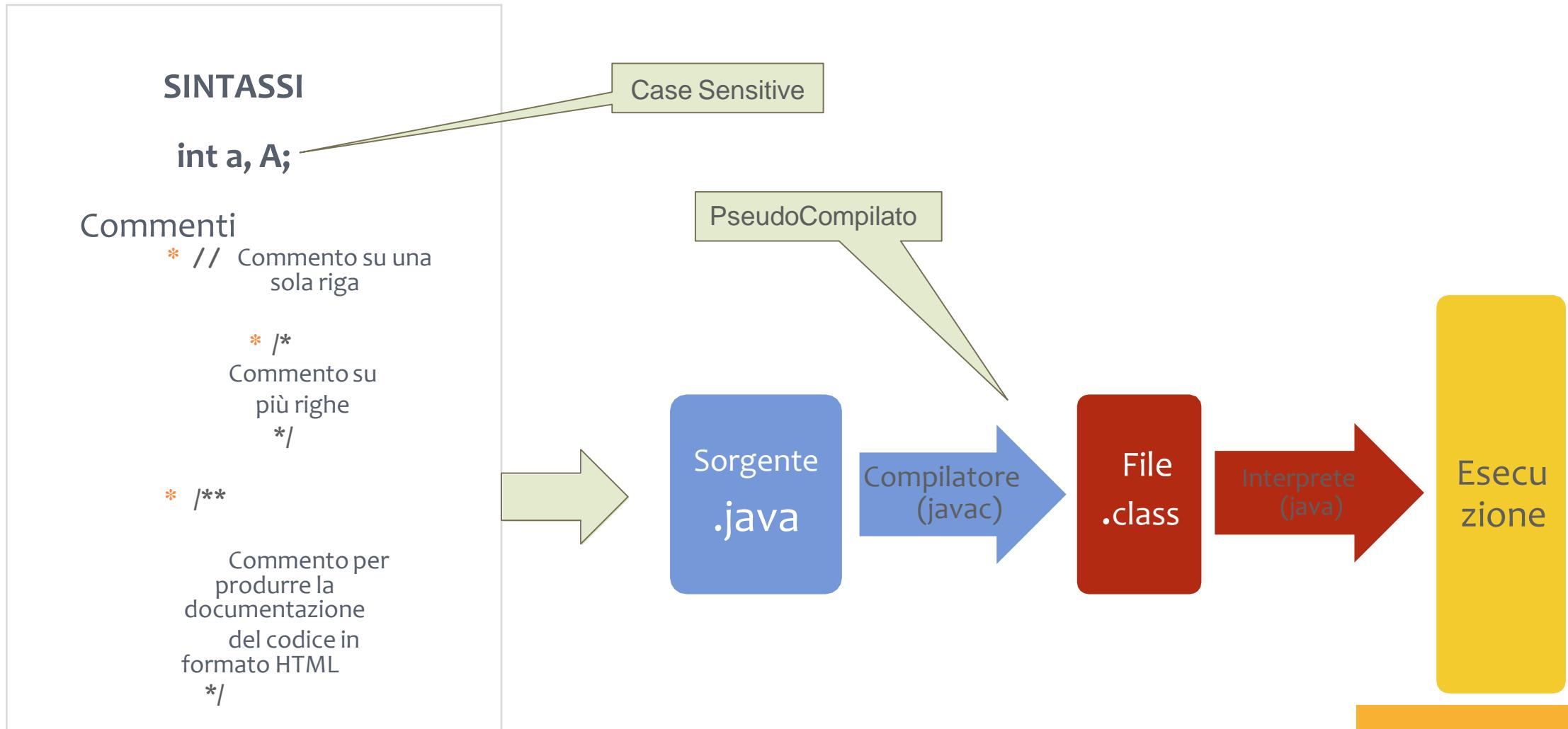
```
/*
 * La classe Cerchio definisce un cerchio di raggio r
 */
public class Cerchio
{
    //variabili di istanza (sezione privata)
    private double raggio;
    //costruttori (sezione pubblica)
    /**
     * Crea un cerchio di raggio 0
     */
    public Cerchio()
    {
        raggio = 0;
    }
    /**
     * Crea un cerchio di raggio dato
     * @param r il raggio dato
     */
    public Cerchio(double r)
    {
        raggio = r;
    }
    //metodi (sezione pubblica)
    /**
     * Calcola la lunghezza della circonferenza
     * @return la circonferenza del cerchio
     */
    public double calcolaCrf()
    {
        return 2 * Math.PI * raggio;
    }
}
```

**Tutte le istruzioni Java
saranno contenute
all'interno di una CLASSE**



- * Principi:
 - * Incapsulamento
 - * Ereditarietà
 - * Polimorfismo

• Caratteristiche del linguaggio: alcuni elementi



• Caratteristiche del linguaggio: alcuni elementi

**Gli IDENTIFICATORI
(nomi di variabili e costanti)
sono sottoposti a :**

- **Regole (generano errori)**
Es. non usare caratteri speciali e spazi per il nome di variabili
- **Convenzioni (standard da seguire)**
Es. Regola del Camel Case

Esempi di valori (literals)

10	(int)
10.5	(float o double)
true/false	(boolean)
'a'	(char)
"ciao"	(String)

SEPARATORI

Symbol	Name	Purpose
()	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a for statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

• Caratteristiche del linguaggio: parole chiave e librerie

Java Reserved Keywords				
abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Libreria da importare

```

input.java ✘
1 package input;
2
3 import java.util.Scanner;
4
5 public class input {
6
7     public static void main(String[] args) {
8         int a,b,c;
9         Scanner input = new Scanner(System.in);
10        System.out.print("Inserisci il valore in A: ");
11        a=input.nextInt();
12        System.out.print("Inserisci il valore in B: ");
13        b=input.nextInt();
14        c=a+b;
15        System.out.print("La somma tra A e B fa: "+c);
16    }
17
18 }
```

Libreria già incorporata:
Java.lang

- Caratteristiche del linguaggio: Un semplice Esempio

```
4o /*
5 Questo è un semplice programma Java.
6 Dovrai chiamare questo file obbligatoriamente "BenvenutoInJava.java".
7 */
8
9 public class BenvenutoInJava {
10
11 //Il tuo programma inizia con la chiamata al metodo main()
12
13o     public static void main(String[] args) {
14
15         System.out.println("Benvenuto in Java!!!");}
16     }
17 }
18
```

- Caratteristiche del linguaggio: VARIABILI e COSTANTI

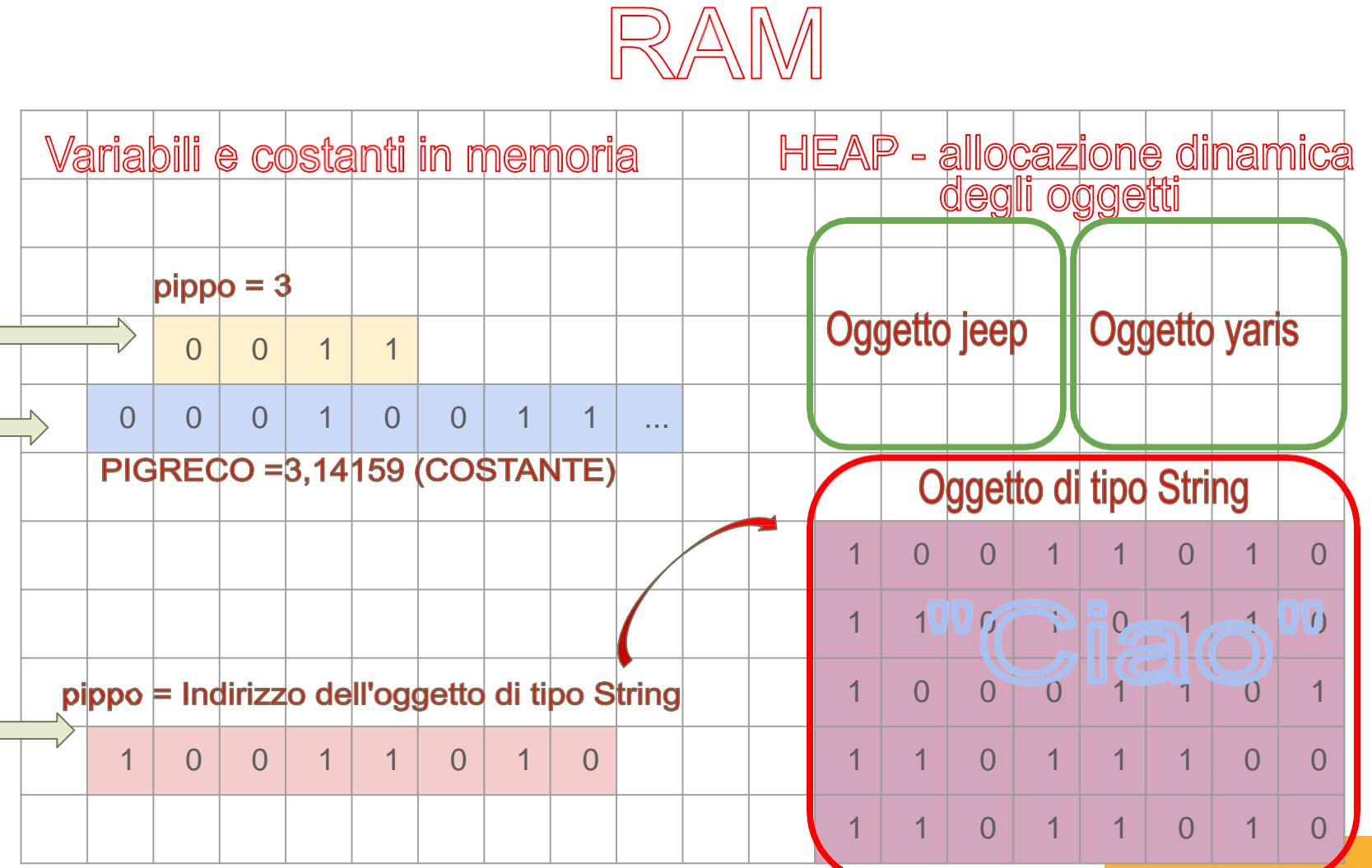
VARIABLE
area di memoria contenente un valore

```
int pippo=3;
```

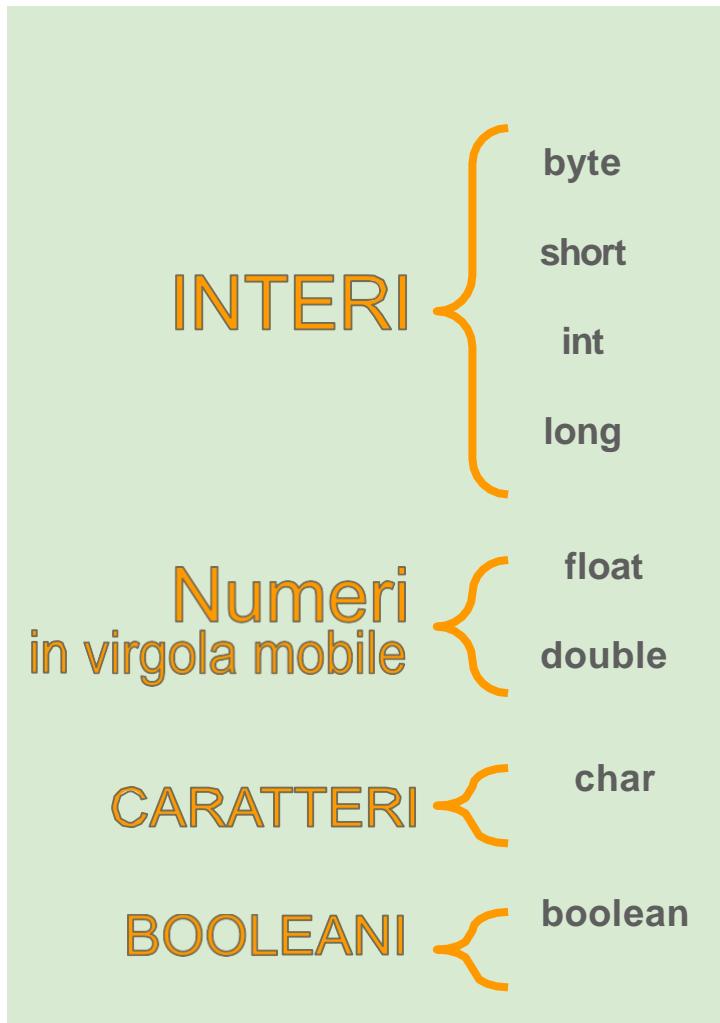
VARIABLE DI RIFERIMENTO
area di memoria il cui valore è un riferimento ad un oggetto

```
String pippo= new String("Ciao");
```

```
Macchina jeep= new Macchina();
```



- Caratteristiche del linguaggio: tipi di dato - primitivi



Dichiarazioni e inizializzazioni

```

byte pippo=1;
short pippo=30000;
int pippo=2100000000;
long pippo=340000....;
float pippo=30,43198;
double pippo=43,989898...;
char pippo='S';
boolean pippo=true;

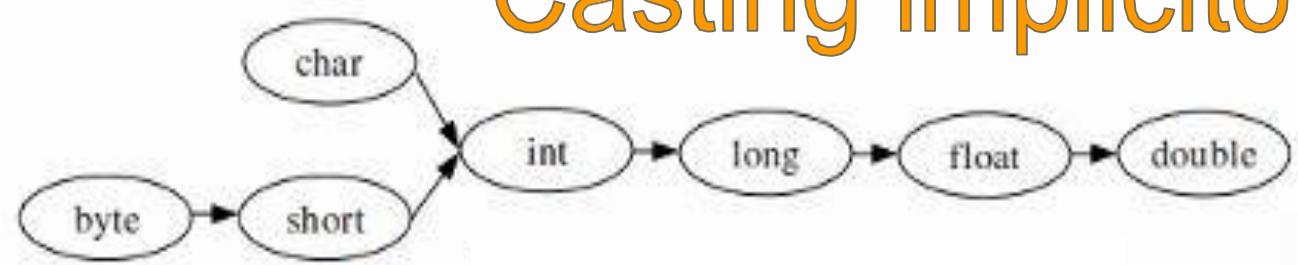
```



Tipo	Requisiti di memorizzazione	Intervallo
byte	1 byte	da -128 a 127
short	2 byte	da -32768 a 32767
int	4 byte	da -2147483648 a 2147483647
long	8 byte	da -9 miliardi di miliardi a 9 miliardi di miliardi
float	4 byte	6-7 cifre decimali significative
double	8 byte	15 cifre decimali significative

- Caratteristiche del linguaggio: Casting implicito ed esplicito

Casting implicito



Casting esplicito

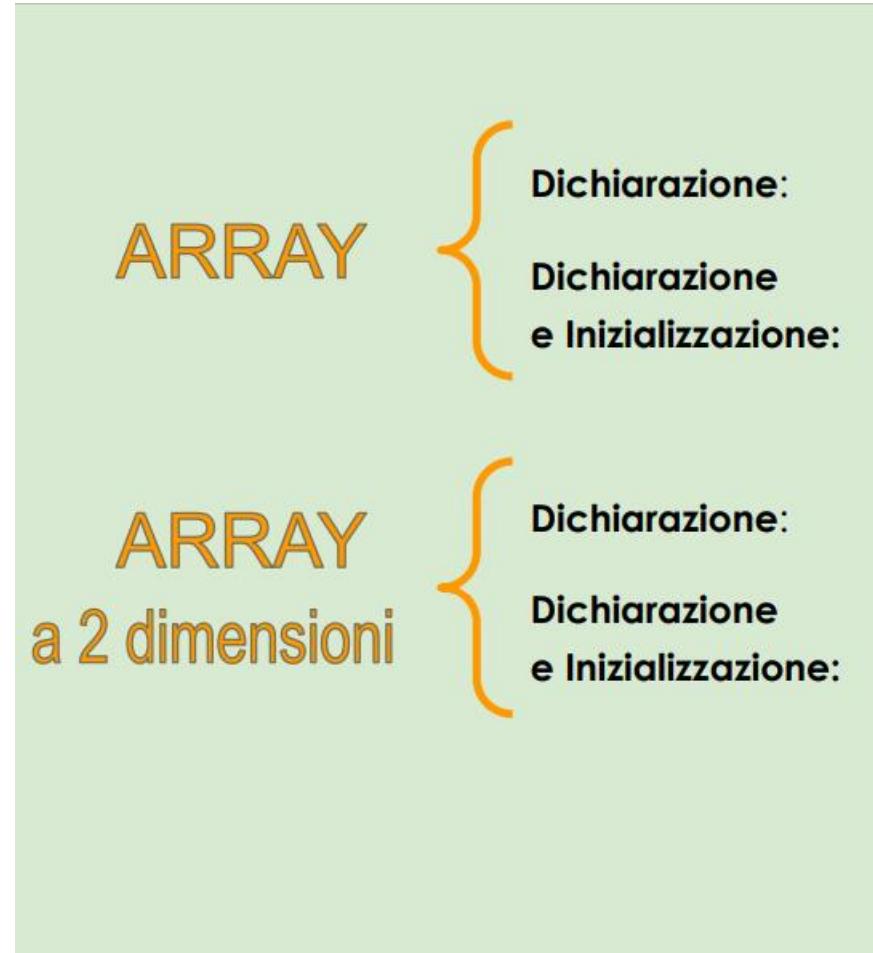
- E' necessario quando un tipo memorizzato su un numero piu' grande di bit e' assegnato ad un tipo memorizzato su un numero piu' basso di bit
- Sintassi:

(tipo)nomeVariabile

- Esempio:

```
int x;  
float y=5.7;  
x = (int)y; // in x e' memorizzato il valore 5
```
- Osservazione: il cast esplicito effettua un troncamento!

- Caratteristiche del linguaggio: tipi di dato - ARRAY



• Caratteristiche del linguaggio: tipi di dato - STRINGA

Le stringhe sono oggetti



- * **Creazione esplicita :** Es. **String** pippo= **new String**("parola");
- * **Creazione implicita:** Es. **String** pippo = "parola";

Metodi Classe String

nome_stringa.charAt(indice)

Restituisce il carattere che si trova alla posizione *indice* della stringa corrente *nome_stringa* (this). Gli indici sono numerati a partire da 0.

nome_stringa.compareTo(altra_stringa)

Confronta la stringa corrente *nome_stringa* (this) con *altra_stringa* per individuare quale viene prima in ordine lessicografico. L'ordine lessicografico corrisponde all'ordine alfabetico quando entrambe le stringhe sono costituite solo da lettere maiuscole o solo da lettere minuscole. Il metodo restituisce un valore intero negativo se la stringa corrente viene prima, 0 (zero) se sono uguali o un numero positivo se la stringa corrente viene dopo.

nome_stringa.concat(altra_stringa)

Restituisce una nuova stringa che presenta gli stessi caratteri della stringa corrente *nome_stringa* (this) concatenati con quelli in *altra_stringa*. Invece di **concat** può essere utilizzato l'operatore +.

nome_stringa.equals(altra_stringa)

Restituisce true se la stringa corrente *nome_stringa* (this) e *altra_stringa* sono uguali. Altrimenti restituisce false.

nome_stringa.equalsIgnoreCase(altra_stringa)

Si comporta come il metodo **equals**, ma considera uguali le lettere maiuscole e le lettere minuscole della stringa.

nome_stringa.indexOf(altra_stringa)

Restituisce l'indice della prima occorrenza della sottostringa *altra_stringa* nella stringa corrente *nome_stringa* (this). Restituisce -1 se la sottostringa *altra_stringa* non compare. Gli indici sono numerati a partire da 0.

nome_stringa.lastIndexOf(altra_stringa)

Restituisce l'indice dell'ultima occorrenza della sottostringa *altra_stringa* all'interno della stringa corrente *nome_stringa* (this). Restituisce -1 se la sottostringa *altra_stringa* non compare. Gli indici sono numerati a partire da 0.

nome_stringa.length()

Restituisce la lunghezza della stringa corrente *nome_stringa* (this).

nome_stringa.toLowerCase()

Restituisce una nuova stringa che presenta gli stessi caratteri della stringa corrente *nome_stringa* (this), ma in cui tutte le lettere maiuscole sono state sostituite con le minuscole corrispondenti.

nome_stringa.toUpperCase()

Restituisce una nuova stringa che presenta gli stessi caratteri della stringa corrente *nome_stringa* (this), ma in cui tutte le lettere minuscole sono state sostituite con le corrispondenti lettere maiuscole.

nome_stringa.replace(vecchio_carattere, nuovo_carattere)

Restituisce una nuova stringa che presenta gli stessi caratteri della stringa corrente *nome_stringa* (this), ma in cui tutte le occorrenze del carattere *vecchio_carattere* sono state sostituite dal carattere *nuovo_carattere*.

nome_stringa.substring(inizio)

Restituisce una nuova stringa che presenta gli stessi caratteri della sottostringa che inizia all'indice *inizio* della stringa corrente *nome_stringa* (this) fino alla fine della stringa. Gli indici sono numerati a partire da 0.

nome_stringa.substring(inizio, fine)

Restituisce una nuova stringa che presenta gli stessi caratteri della sottostringa che inizia all'indice *inizio* della stringa corrente *nome_stringa* (this) fino all'indice *fine* escluso. Gli indici sono numerati a partire da 0.

nome_stringa.trim()

Restituisce una nuova stringa che presenta gli stessi caratteri della stringa corrente *nome_stringa* (this), ma in cui sono stati rimossi i caratteri di spaziatura in testa e in coda alla stringa.

• Caratteristiche del linguaggio: operatori aritmetici

Simbolo	Operazione
+	Addizione / Concatenazione
-	Sottrazione
*	Moltiplicazione
/	Divisione
%	Divisione con modulo (solo per interi)
++	Incremento unitario
--	Decremento unitario

```
i++;           i = i+1;
i--;           i = i-1;
n%2 == 0;      if (n%2 == 0) {
                then (numero pari)
                else (numero dispari)
}

somma = a + 5;
differenza = b-a;
prodotto = a * 5;
quoziente = b / a;
```

• Caratteristiche del linguaggio: operatori DI CONFRONTO

Simbolo	Operazione	Utilizzo
<code>==</code>	Uguale a	<code>a == b</code>
<code>!=</code>	Diverso da	<code>a != b</code>
<code><</code>	Minore di	<code>a < b</code>
<code>></code>	Maggiore di	<code>a > b</code>
<code>>=</code>	Maggiore o uguale a	<code>a >= b</code>
<code><=</code>	Minore o uguale a	<code>a <= b</code>

ATTENZIONE

Confronto
`if a == b`

Assegnazione
`a = b`

• Caratteristiche del linguaggio: operatori LOGICI

Simbolo	Operazione	Utilizzo
<code>&&</code>	AND logico	<code>(a && b)</code>
<code> </code>	OR logico	<code>(a b)</code>
<code>!</code>	NOT logico	<code>! (a && b)</code>

■ Applicazione della “*valutazione cortocircuitata*”

- `(a && b)`
 - Se a è falsa allora b non viene valutata
- `(a || b)`
 - Se a è vera allora b non viene valutata

- Caratteristiche del linguaggio: costrutti di controllo: SCELTA

Costrutto decisionale *if – else*

```
if( condizioneBooleana ) {
    istruzione;
    ...
}
else {
    istruzione;
}
```



```
1. public class HelloWorld{
2.
3.     public static void main(String []args){
4.         int a=10;
5.         int b=5;
6.         int max;
7.         if (a>b) {
8.             max=a;
9.         } else {
10.            max=b;
11.        }
12.        System.out.println("max = " + max);
13.    }
14. }
```

WWW.ANDREAMININI.COM

- Caratteristiche del linguaggio: costrutti di controllo: SWITCH CASE

Costrutto decisionale **switch – case**

```
switch (espressione) {
    case val1:
        istruzione_a;
        break;
    case val2:
        istruzione_b;
        break;
    [default:
        istruzione_default;
        break;]
}
```



```
1  public class HelloWorld{
2
3      public static void main(String []args){
4          String mese = "02";
5          String nome;
6
7          switch(mese) {
8              case "01":
9                  nome="Gennaio";
10                 break;
11             case "02":
12                 nome="Febbraio";
13                 break;
14             case "03":
15                 nome="Marzo";
16                 break;
17             default:
18                 nome="Altri mesi";
19         }
20     }
21 }
22 }
```

WWW.ANDREAMININI.COM

• Caratteristiche del linguaggio: costrutti di controllo: ITERAZIONE

Ciclo condizionale **while**

```
while(condizioneBooleana) {
    istruzione1;
    istruzione2;
    ...
}
```

Ciclo condizionale **do- while**

```
do {
    istruzione1;
    istruzione2;
    ...
} while(condizioneBooleana);
```

```
1- public class HelloWorld{
2
3-     public static void main(String []args){
4
5         int conta=0;
6
7         while(conta<11) {
8
8             System.out.println(conta);
9             conta++;
10
11     }
12
13 }
14
15 }
```

WWW.ANDREA

```
reader = new BufferedReader(new FileReader(
        "/Users/pankaj/Downloads/myfile.txt"));
String line = reader.readLine();
while (line != null) {
    System.out.println(line);
    // read next line
    line = reader.readLine();
}
```

```
1- public class HelloWorld{
2
3-     public static void main(String []args){
4
5         int conta=1;
6
7         do {
8
8             System.out.println(conta);
9             conta++;
10
11     } while(conta<11);
12
13 }
14
15 }
16 }
```

WWW.ANDREAININI.COM

- Caratteristiche del linguaggio: costrutti di controllo: FOR

Ciclo con contatore **for**

```
for(init_statement ; conditional_expr ; iteration_stmt) {
    istruzione1;
    istruzione2;
    ....
}
```

Ciclo condizionale **foreach**

```
for(tipo var-itr: collection) {
    istruzione1;
    istruzione2;
    ....
}
```



```
1 public class HelloWorld{
2
3     public static void main(String []args){
4
5         int conta=0;
6
7         for(conta=0; conta<10; conta++) {
8             System.out.println(conta);
9         }
10
11     }
12 }
```

WWW.ANDREAMININI.COM

```
1 public class WriteForEachLoops {
2
3
4
5     public static void main(String[] args) {
6         String [] names = { "Regina", "Stefano", "Flavia"};
7         System.out.println("For each loop output:");
8         for (String name : names) {
9             System.out.println(name);
10        }
11
12    }
13
14 }
```

- Caratteristiche del linguaggio: costrutti di controllo: istruzioni di SALTO

* break

* continue

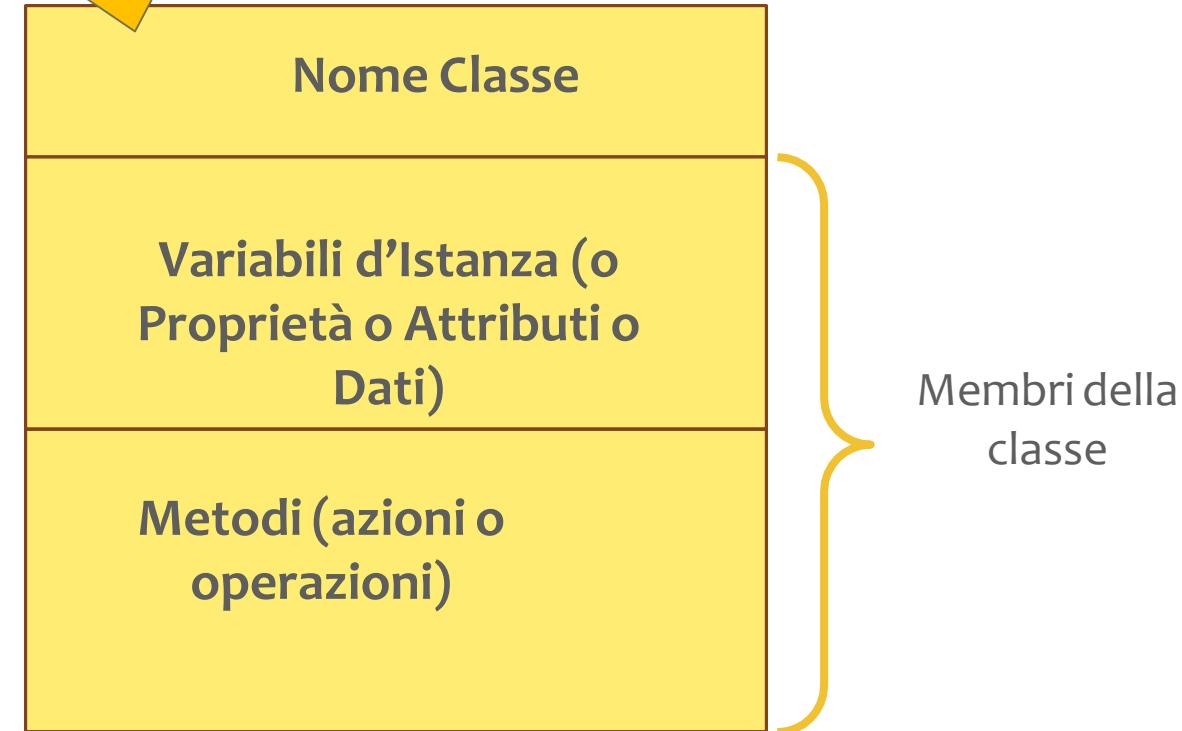


Esempi

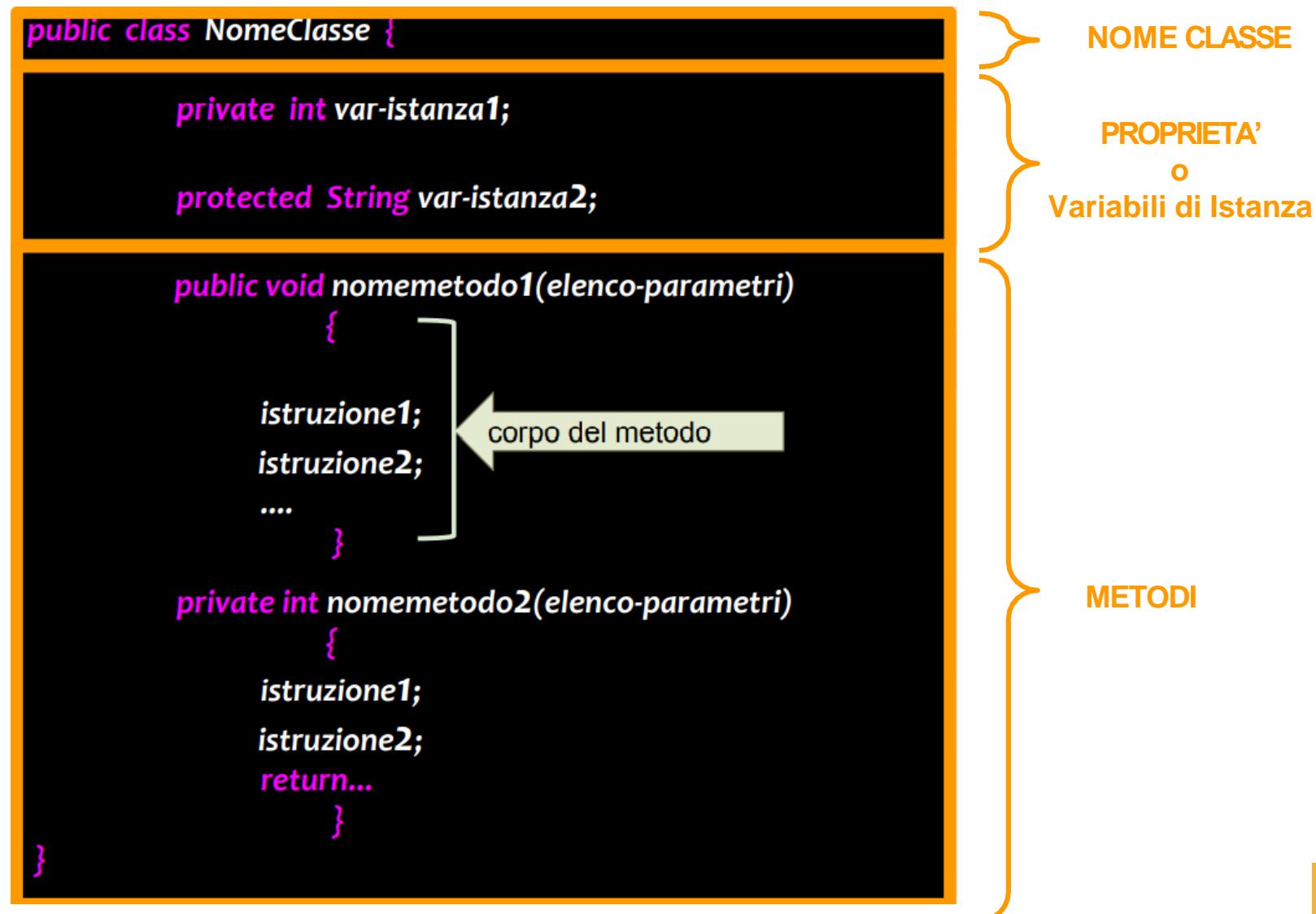
```
for (int i = 0; i < 100; i++) {  
    if (i == 15) break; ←  
    if (i % 5 != 0) continue; ←  
    System.out.println (i);  
}  
  
int i = 0;  
while (true) {  
    i++;  
    int j = i * 4;  
    if (j == 40) break; ←  
    if (i % 10 == 0) continue; ←  
    System.out.println (i);  
}
```

• Caratteristiche del linguaggio: CLASSI e OGGETTI

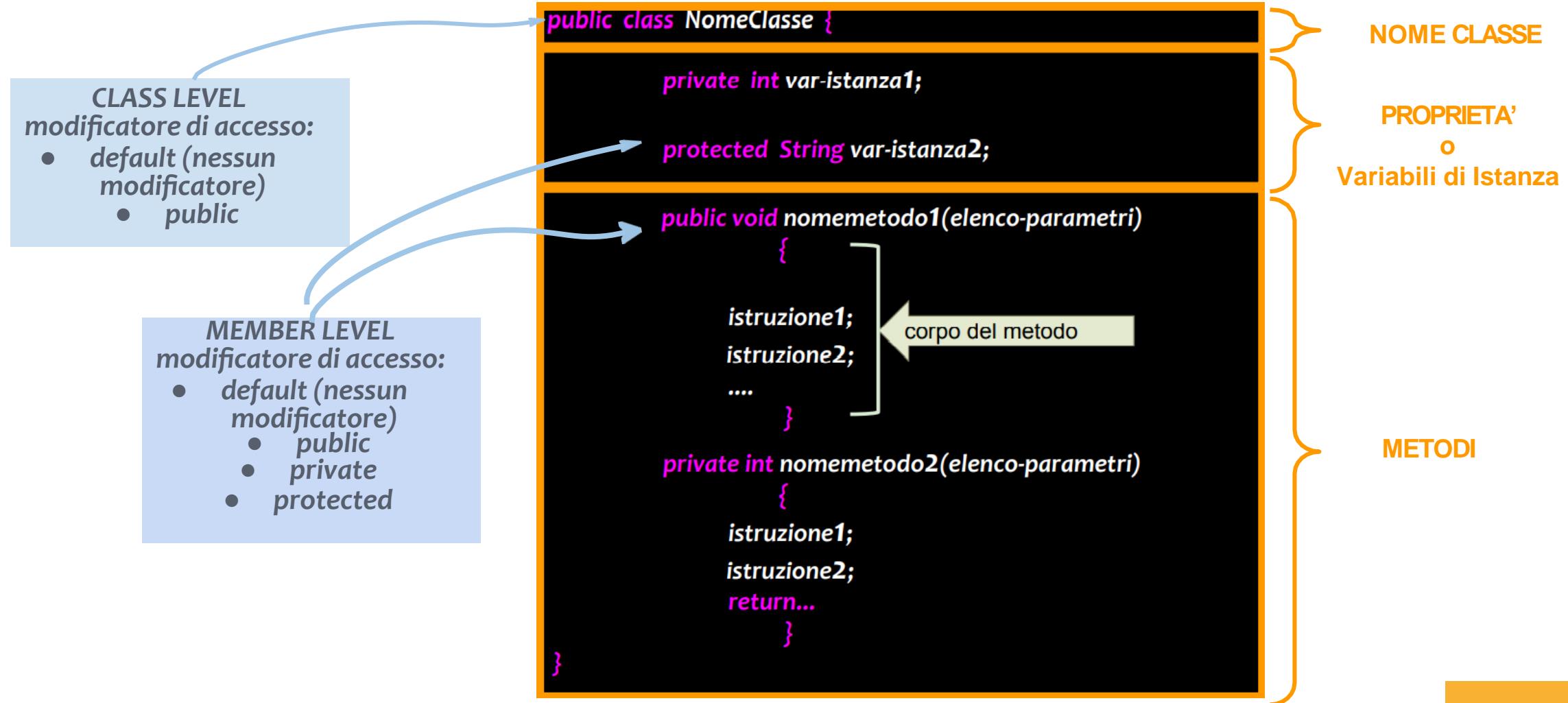
- * Classe: una classe è un'astrazione che rappresenta un insieme di oggetti che condividono le stesse caratteristiche e le stesse funzionalità
- * Oggetto: un oggetto è un'istanza o più precisamente una realizzazione concreta di una classe



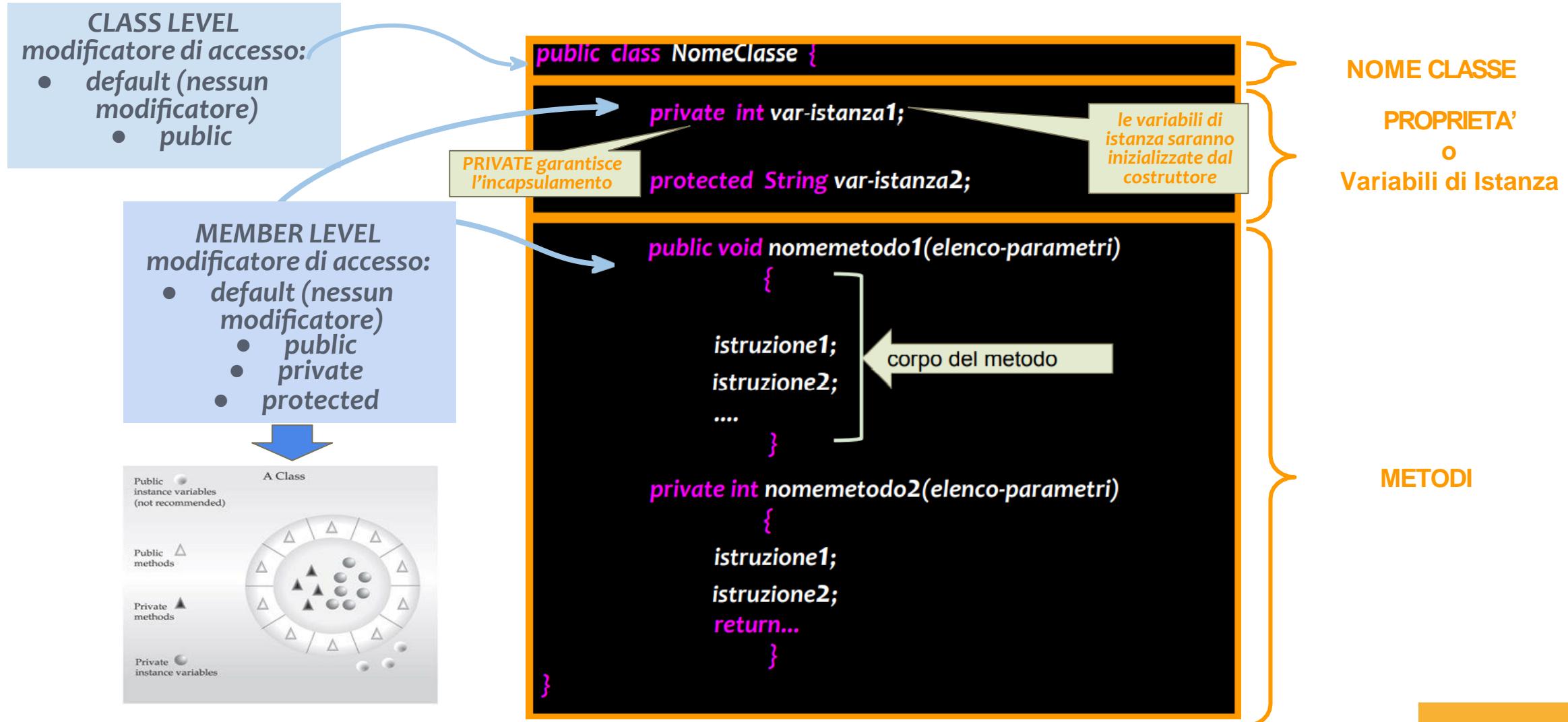
- Caratteristiche del linguaggio: forma generale di una CLASSE



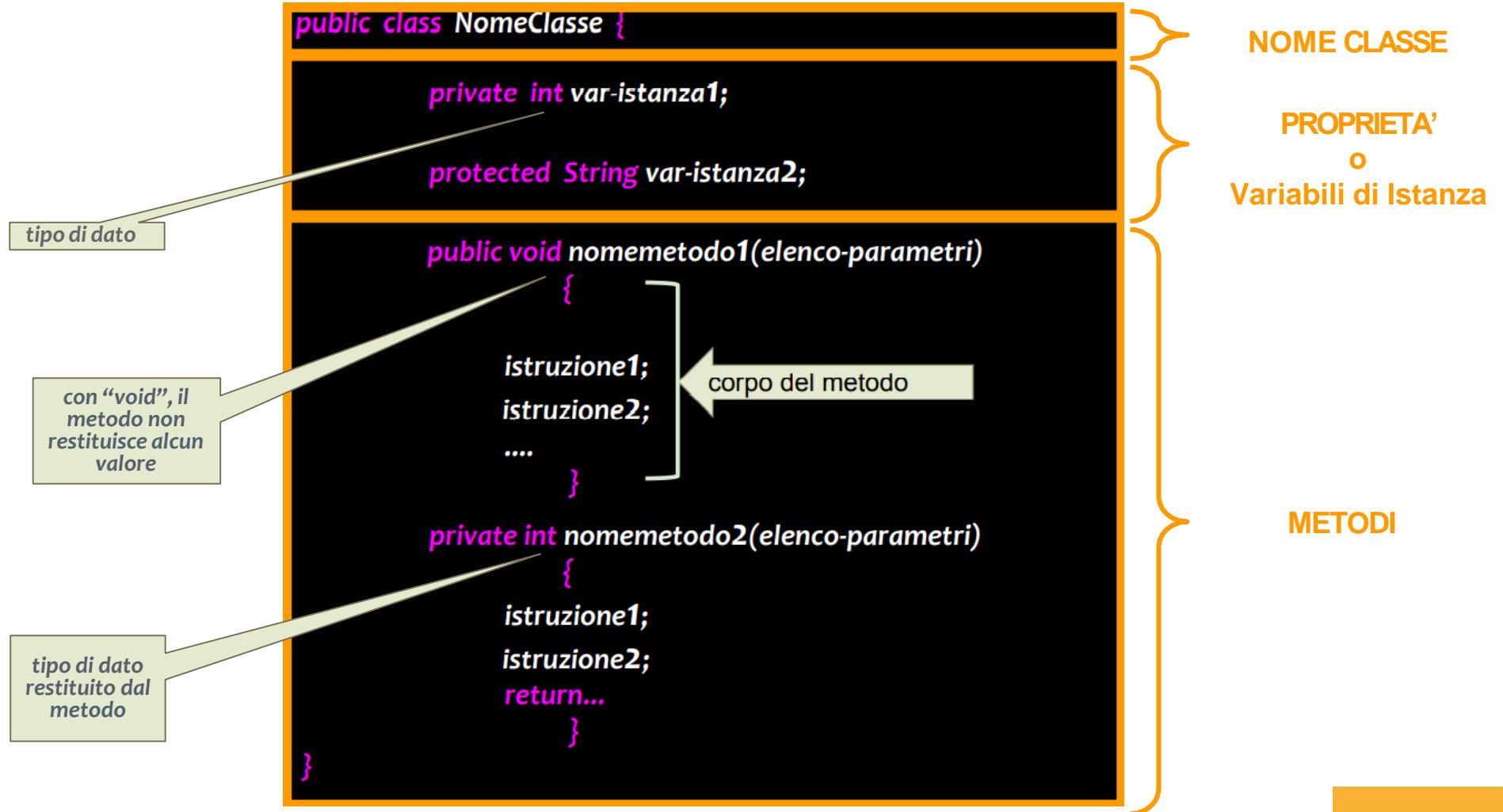
- Caratteristiche del linguaggio: forma generale di una CLASSE



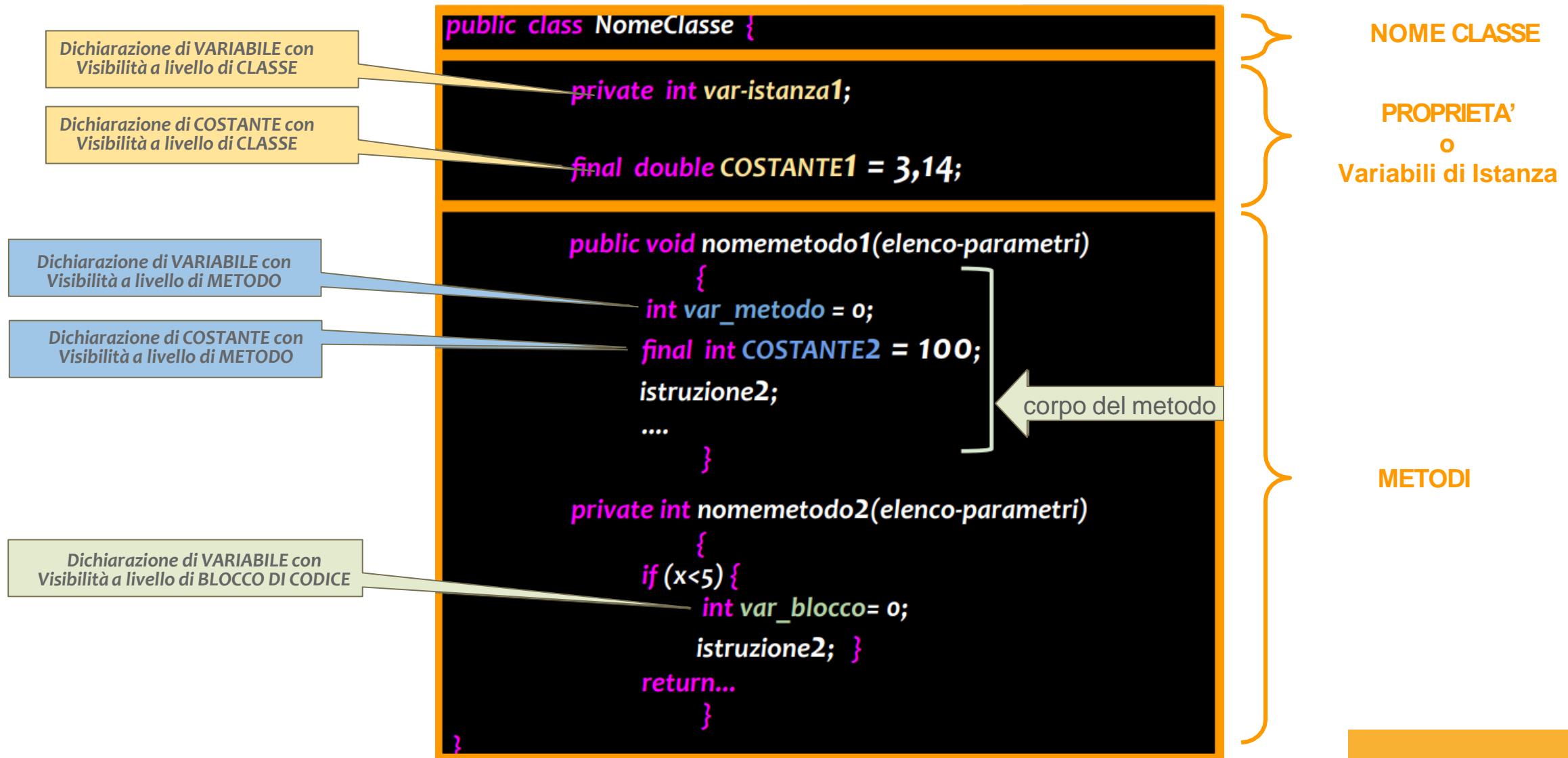
• Caratteristiche del linguaggio: forma generale di una CLASSE



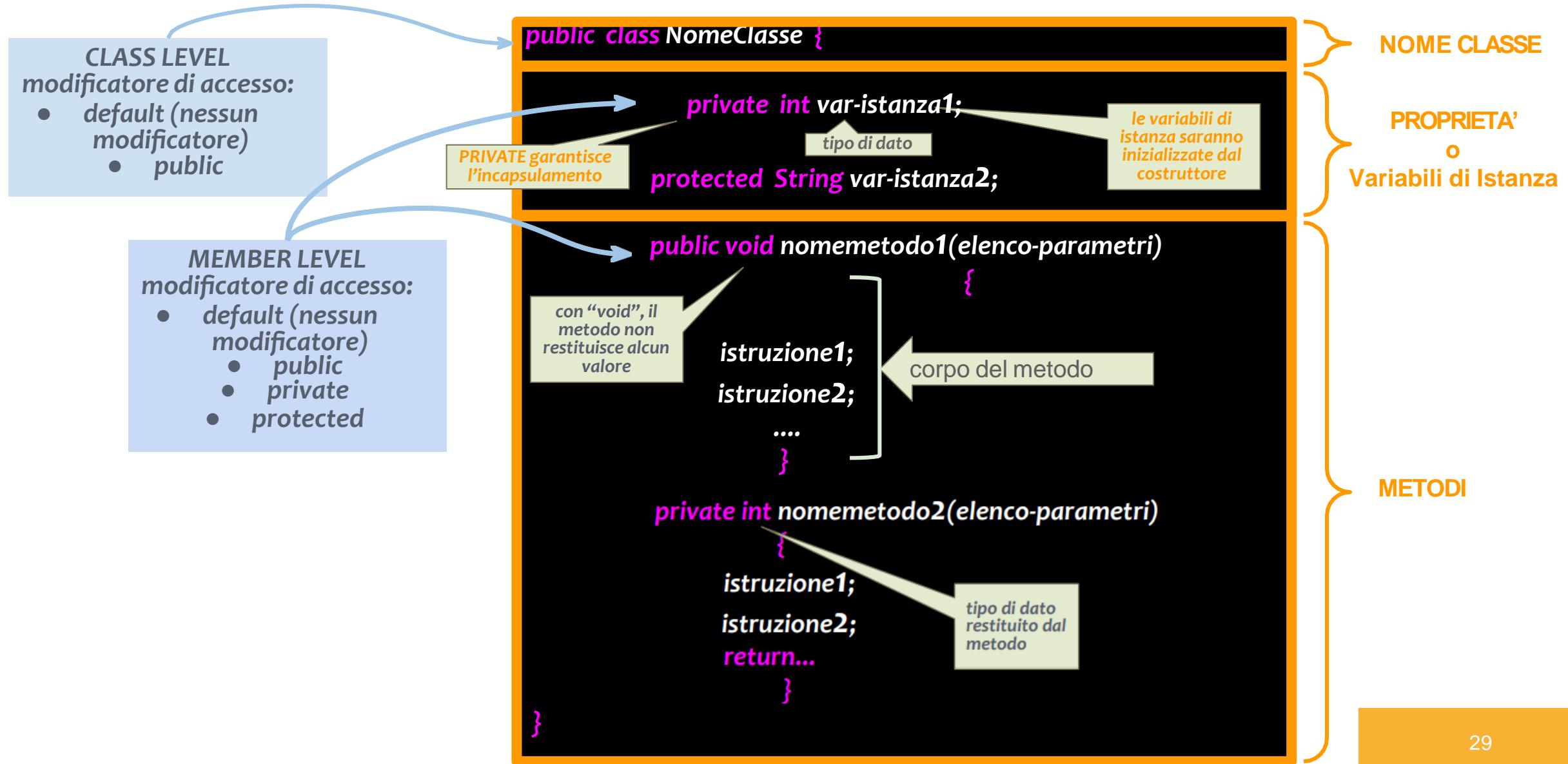
- Caratteristiche del linguaggio: forma generale di una CLASSE



• Caratteristiche del linguaggio: Variabili e Costanti - dichiarazione e SCOPE



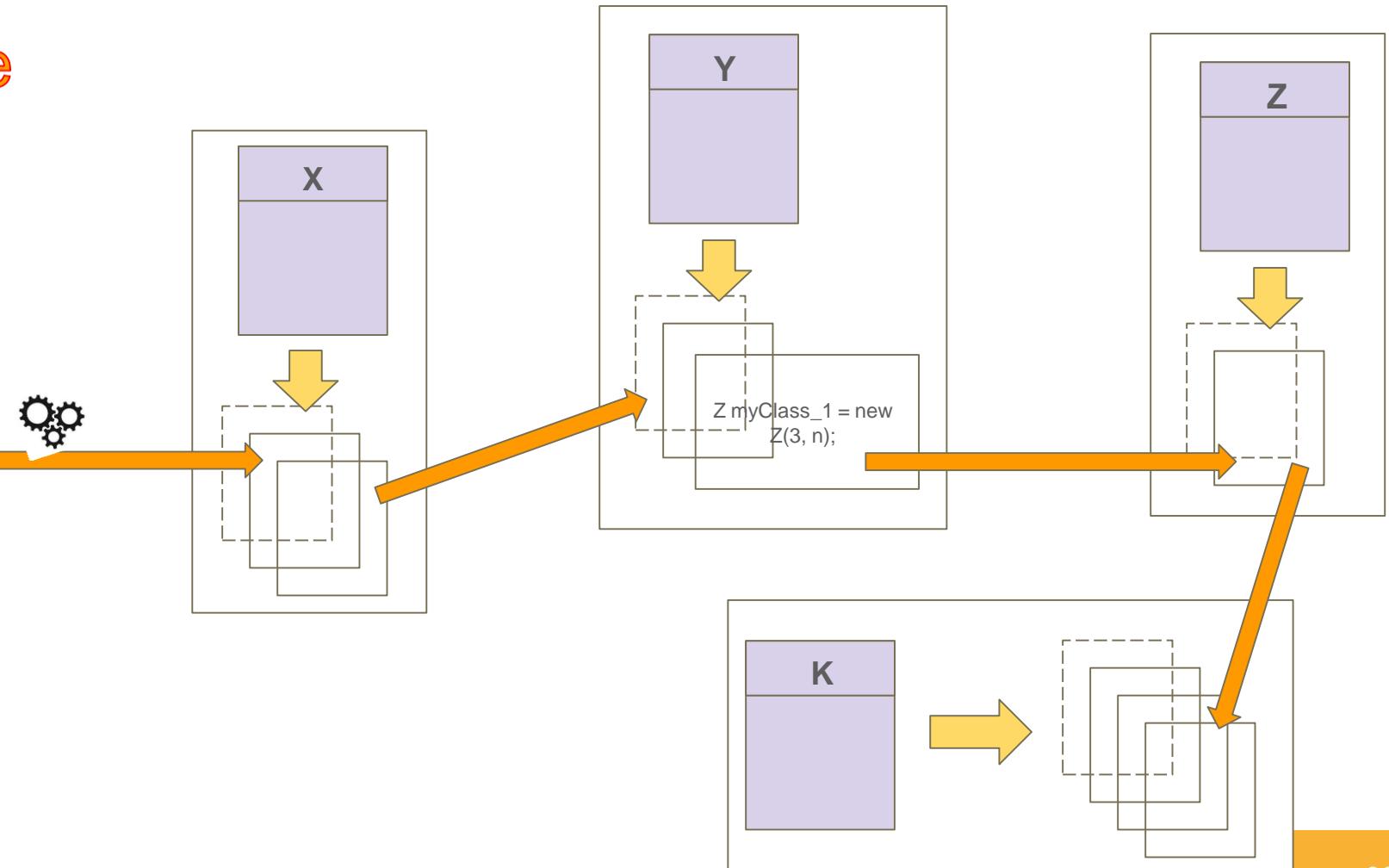
• Caratteristiche del linguaggio: forma generale di una CLASSE



- Caratteristiche del linguaggio: OGGETTO

Classe Principale metodo main

```
class Classe_Princente {
    public static void main (String [] args)
        X variableX = new X(40, 50);
    }
}
```



• Caratteristiche del linguaggio: OGGETTO

```
class MyClass {
    // proprietà
    private int a;
    private int b;

    // costruttore
    public MyClass() {
        a = 2;
        b = 5;
    }

    // costruttore
    public MyClass(int x, int y) {
        a = x;
        b = y;
    }

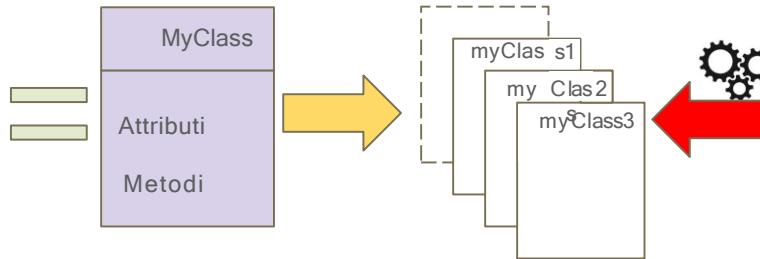
    // metodi
    public void sayHello() {
        System.out.println("Hello! " + a + " " + b);
    }
}
```

Il COSTRUTTORE è un metodo che ha lo stesso nome della classe (esempio di POLIMORFISMO OVERLOADING)

In assenza della dichiarazione di un COSTRUTTORE:

- la classe genera in automatico un costruttore di default che inizializzerà le variabili di istanza in base al tipo di dato

Classe e Oggetti



GARBAGE COLLECTION

- routine automatica che intercetta oggetti istanziati ma senza più riferimenti e li dealloca dalla memoria finalize()
- utilizzare il metodo finalize() in una classe consente di eseguire delle istruzioni prima che il Garbage Collector deallochi l'oggetto

```
class Classe_Principale {
```

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

// STEP 1 : creazione delle istanze

```
MyClass variabile_1 = new MyClass(40, 50);
MyClass variabile_2 = new MyClass();
```

istanzia l'oggetto con passaggio di parametri, chiamando il metodo costruttore

```
MyClass variabile_3;
variabile_3 = new MyClass("a", "b");
// restituisce un "errore tipo di dato parametri"
```

```
MyClass variabile_4 = NULL;
```

inizializzazione di una variabile di riferimento

// STEP 2 : utilizzo metodi pubblici della classe

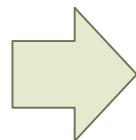
```
variabile_1.sayHello();
// Stamperà "Hello! 40 50"
```

```
variabile_2.sayHello();
// Stamperà "Hello! 2 5"
```

```
variabile_4.sayHello();
// Errore, oggetto non istanziato
```

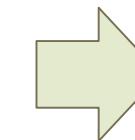
- Caratteristiche del linguaggio: I METODI ... più da vicino

DEFINIZIONE
blocchi di istruzioni che permettono di agire sulle proprietà della classe



DICHIARAZIONE

```
public int myMetodo(int x, int, y) {  
    a = x;  
    b = y;  
    return a+b;  
}
```



CHIAMATA
int somma = myMetodo(10,20);

SIGNATURE (FIRMA):

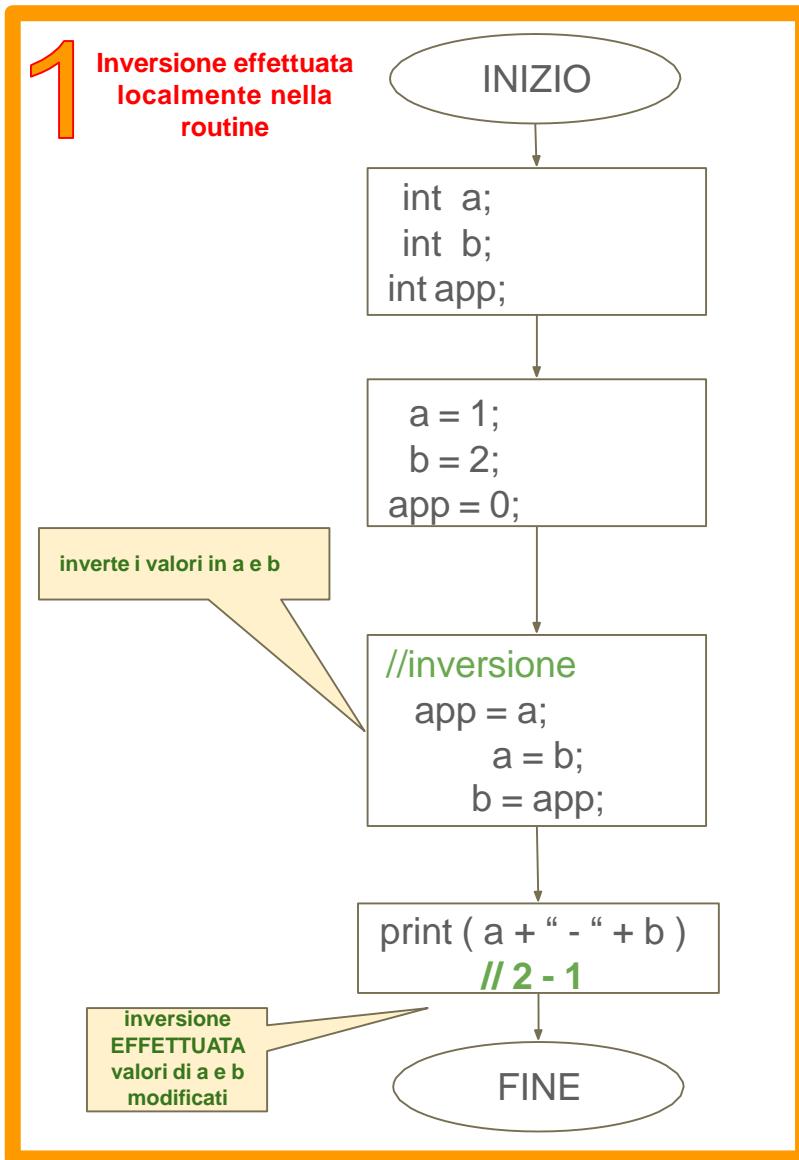
- nome del metodo
- eventuali parametri

myMetodo(int x, int, y)

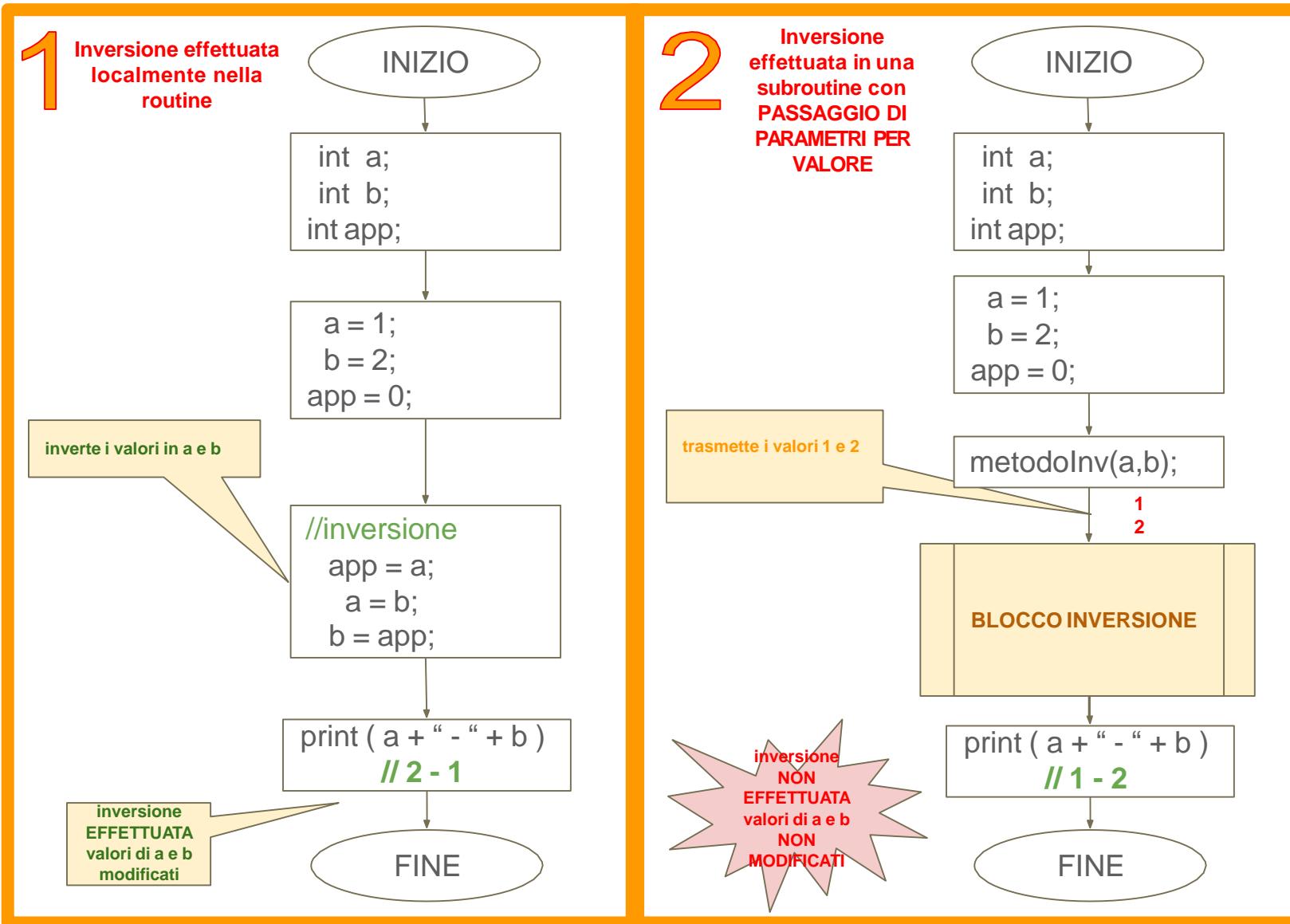
RETURN

esce dal metodo e restituisce un valore del tipo specificato nella dichiarazione

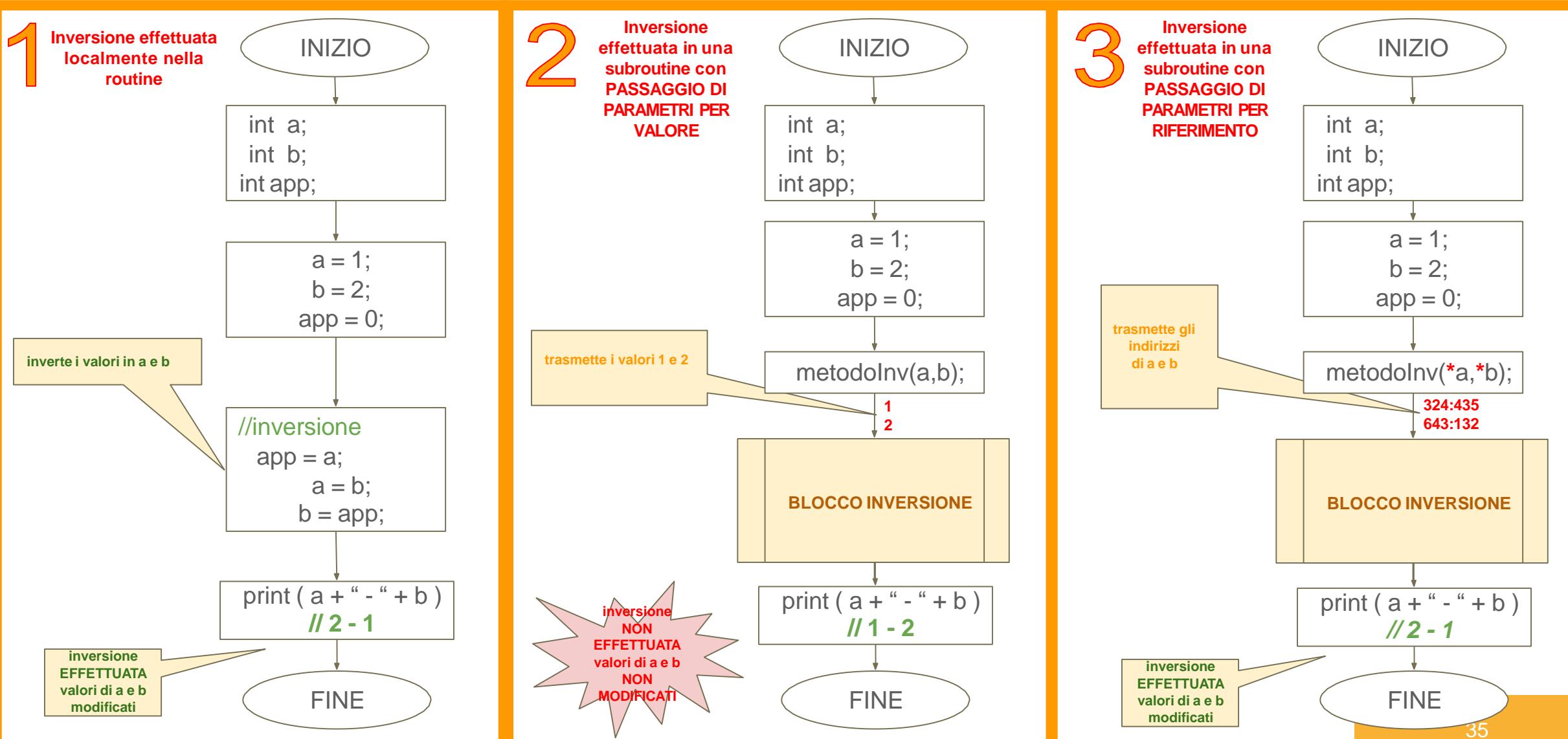
• Caratteristiche del linguaggio: I METODI - passaggio di parametri



• Caratteristiche del linguaggio: I METODI - passaggio di parametri

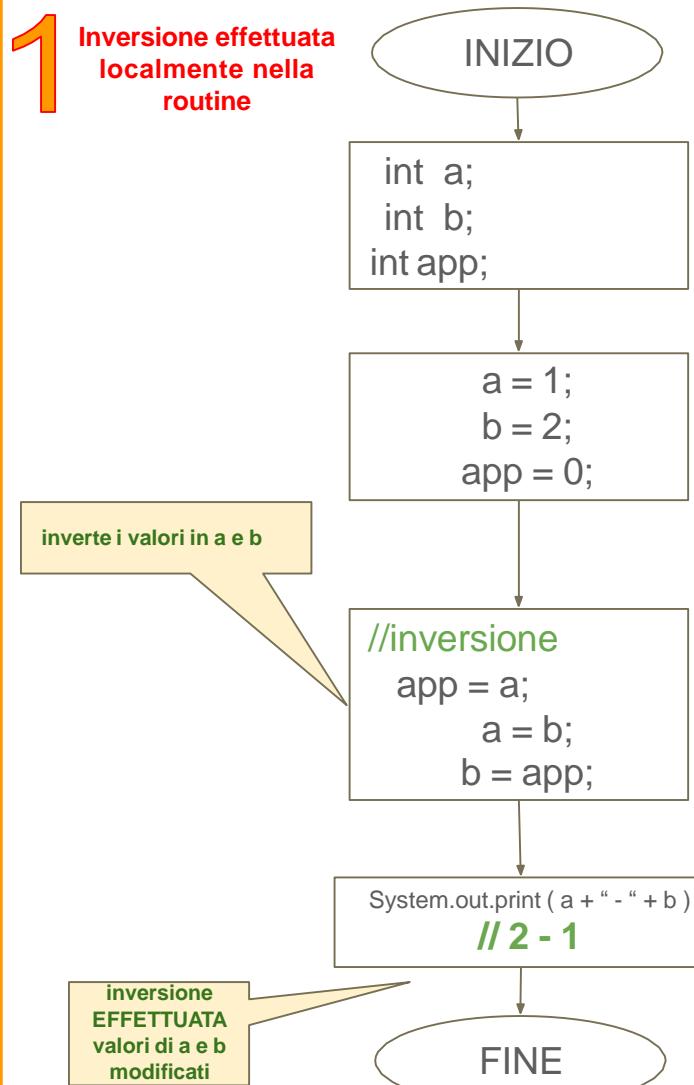


• Caratteristiche del linguaggio: I METODI - passaggio di parametri

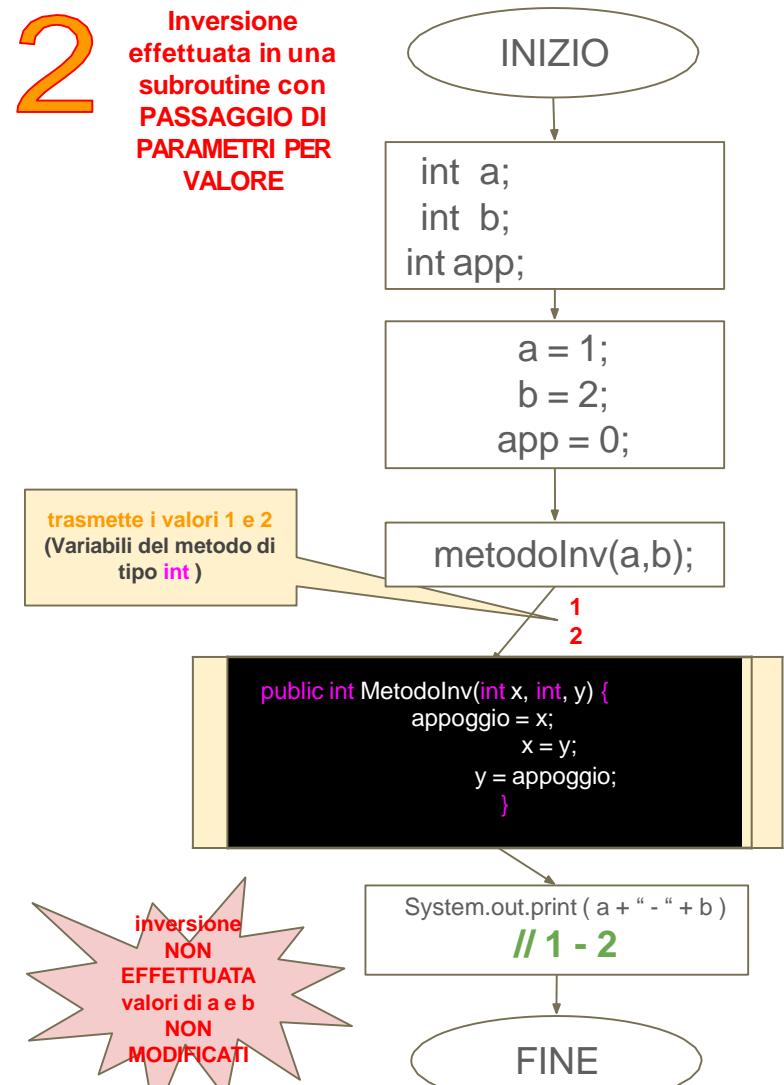


• Caratteristiche del linguaggio: I METODI - passaggio di parametri

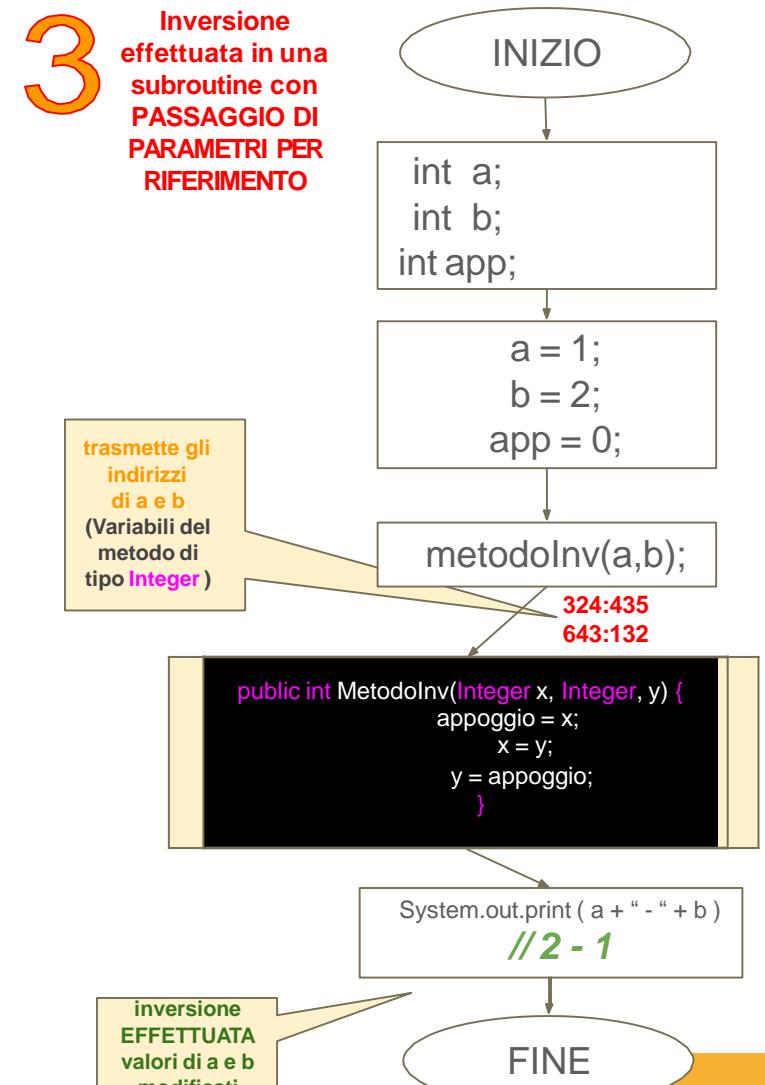
1 Inversione effettuata localmente nella routine



2 Inversione effettuata in una subroutine con PASSAGGIO DI PARAMETRI PER VALORE



3 Inversione effettuata in una subroutine con PASSAGGIO DI PARAMETRI PER RIFERIMENTO



• Caratteristiche del linguaggio: I METODI - setter e getter

```
class MyClass {
    // proprietà
    private int a;
    private int b;

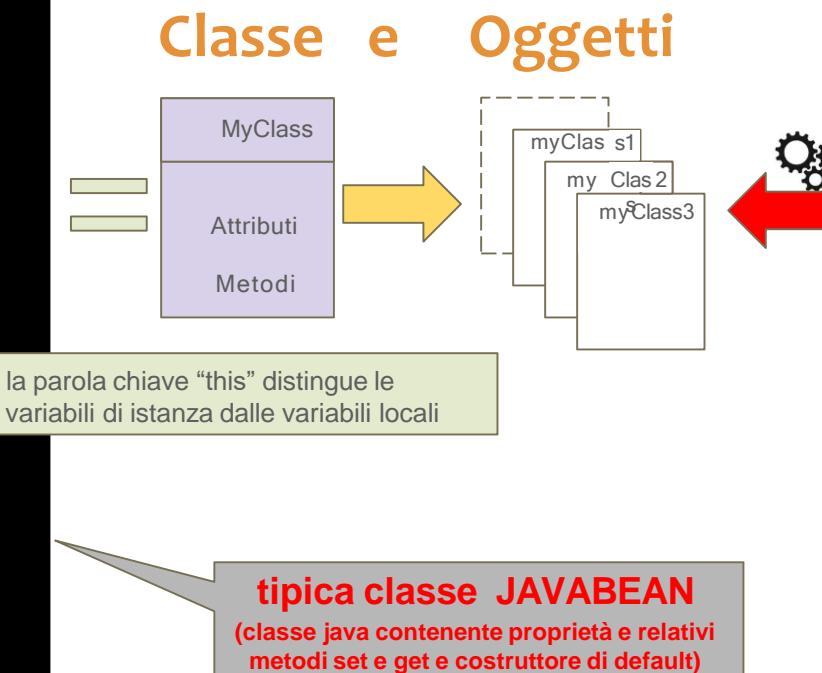
    // metodo set della variabile a
    public void setA(int x) {
        a = x;
    }

    // metodo set della variabile b
    public void setB(int b) {
        this.b = b;
    }

    // metodo get della variabile a
    public int getA() {
        return a;
    }

    // metodo get della variabile b
    public int getB() {
        return b;
    }
}
```

- In assenza della dichiarazione di un COSTRUTTORE:**
- la classe genera in automatico un costruttore di default che inizializzerà le variabili di istanza in base al tipo di dato



```
class Classe_Principale {
    public static void main (String [] args) {
        // STEP 1 : creazione delle istanze
        MyClass variabile_1 = new MyClass();

        // STEP 2 : utilizzo metodi pubblici della classe
        variabile_1.setA( 34 );
        // valorizza la variabile a con il valore 34
        variabile_1.setB( 105 );
        // valorizza la variabile b con il valore 105
        int valoreA = variabile_1.getA();
        System.out.print( "valori : " + valoreA + " - " + variabile_1.getB());
        // Stamperà "valori : 34 - 105"
    }
}
```

istanzia l'oggetto SENZA passaggio di parametri, chiamando il metodo costruttore di default

// STEP 1 : creazione delle istanze

MyClass variabile_1 = new MyClass();

// STEP 2 : utilizzo metodi pubblici della classe

variabile_1.setA(34);
// valorizza la variabile a con il valore 34

variabile_1.setB(105);
// valorizza la variabile b con il valore 105

int valoreA = variabile_1.getA();

System.out.print("valori : " + valoreA + " - " + variabile_1.getB());

// Stamperà "valori : 34 - 105"

chiamate ai metodi setter

chiamate ai metodi getter

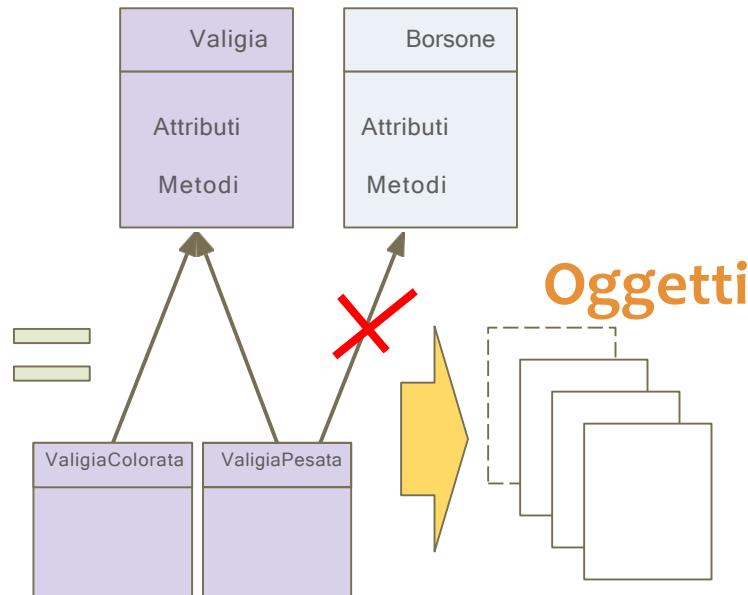
• Caratteristiche del linguaggio: Il principio dell'ereditarietà nel codice

```
class Valigia {
    // proprietà
    private int lunghezza;
    private int altezza;
    private int profondita;
    // metodo costruttore
    public Valigia(int l, int a, int p) {
        lunghezza = l;
        altezza = a;
        profondita = p;
    }
    // metodo calcolaVolume
    public int calcolaVolume() {
        return lunghezza*altezza*profondita;
    }
}
```

```
class ValigiaColorata extends Valigia {
    // proprietà
    private String colore;
    // metodo costruttore
    public ValigiaColorata(int l, int a, int p, String colore) {
        super(l,a,p);
        this.colore = colore;
    }
}
```

```
class ValigiaPesata extends Valigia {
    // proprietà
    private double peso;
    // metodo costruttore
    public ValigiaPesata(int l, int a, int p, double peso) {
        super(l,a,p);
        this.peso = peso;
    }
}
```

Gerarchia delle Classi



Una sottoclasse può estendere una sola classe padre

```
class Classe_Principale {
```

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

// STEP 1 : creazione delle istanze

```
Valigia val = new Valigia(2,4,1);
```

```
ValigiaColorata valCol= new ValigiaColorata(2,4,1, «rosso»);
```

```
ValigiaPesata valPeso= new ValigiaPesata(2,4,1, 3.4);
```

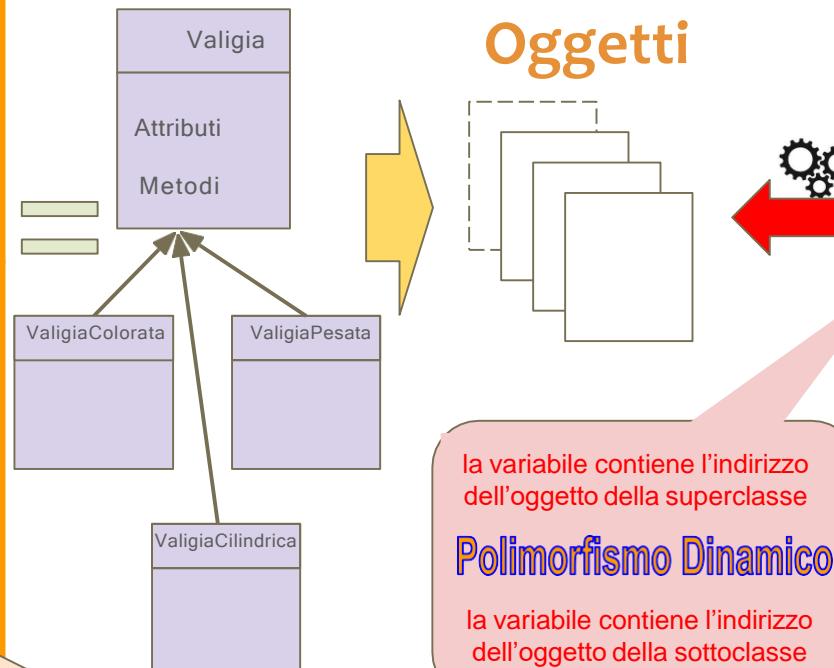
```
}
```

• Caratteristiche del linguaggio: OVERRIDING (polimorfismo)

```
class Valigia {
    // proprietà
    private int lunghezza;
    private int altezza;
    private int profondita;
    // metodo costruttore
    public Valigia(int l, int a, int p) {
        lunghezza = l;
        altezza = a;
        profondita = p;
    }
    // metodo calcolaVolume
    public int calcolaVolume() {
        return lunghezza*altezza*profondita;
    }
}

class ValigiaCilindrica extends Valigia {
    // proprietà
    private int raggio;
    final double PIGRECO = 3.14159;
    // metodo costruttore
    public ValigiaCilindrica(int l, int a, int p, int raggio) {
        super(l,a,p);
        this.raggio = raggio;
    }
    // OVERRIDE metodo calcolaVolume
    public int calcolaVolume() {
        return raggio*raggio*altezza*PIGRECO;
    }
}
```

Gerarchia delle Classi



- OVERRIDE del metodo calcolaVolume
- stessa dichiarazione
 - corpo diverso

```
class Classe_Principale {
```

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

// STEP 1 : dichiarazione variabile di riferimento della superclasse

```
Valigia varRiferimentoValiglia;
```

// STEP 2 : creazione istanza (oggetto) della superclasse

```
varRiferimentoValiglia = new Valigia(30,50,20);
```

// 30.000

// STEP 3 : creazione istanza (oggetto) della sottoclasse

```
varRiferimentoValiglia = new ValigiaCilindrica(0,50,0,30);
```

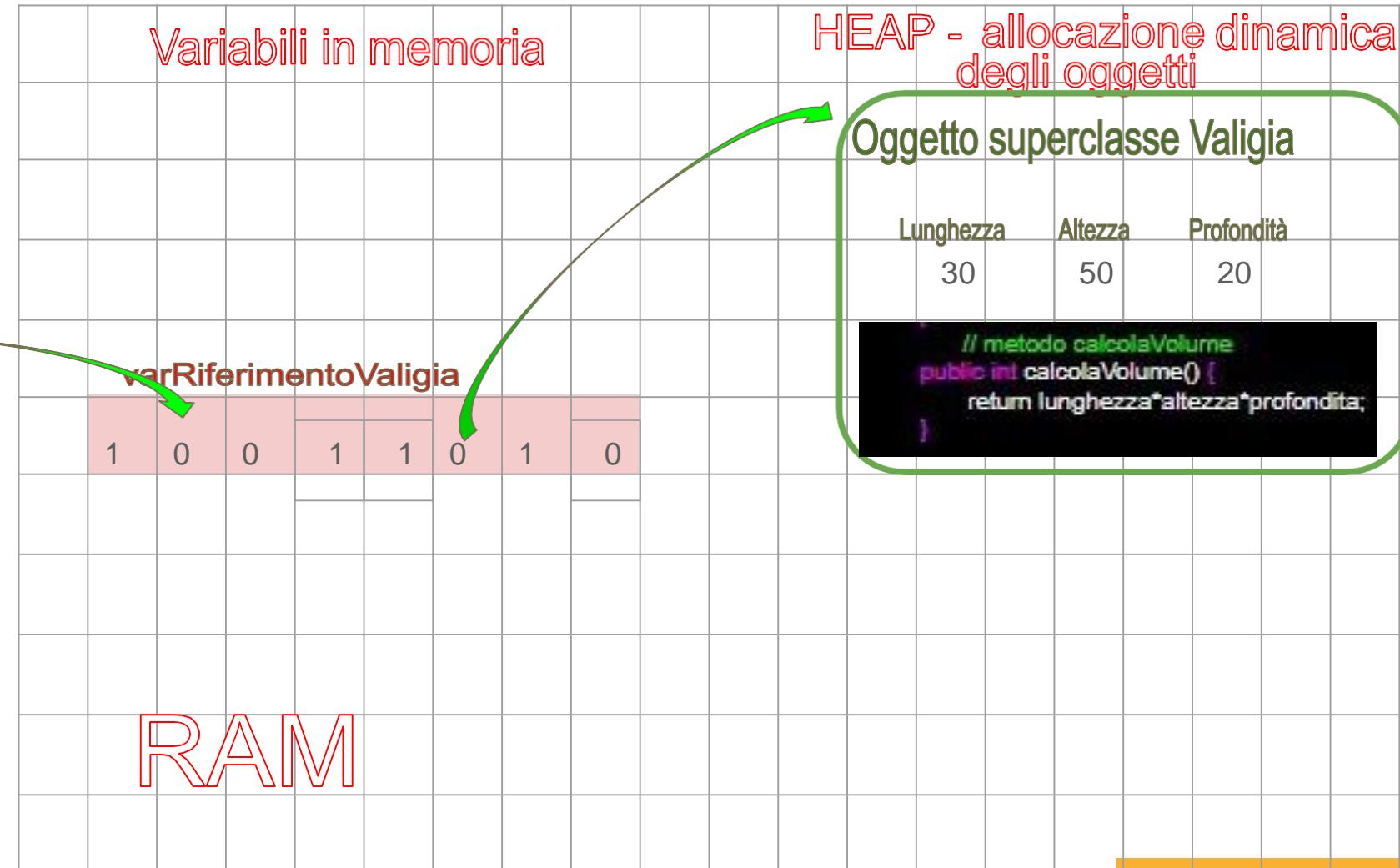
// 141.300

```
}
```

- Caratteristiche del linguaggio: Polimorfismo dinamico 1

VARIABLE DI RIFERIMENTO
area di memoria il cui valore è un riferimento ad un oggetto

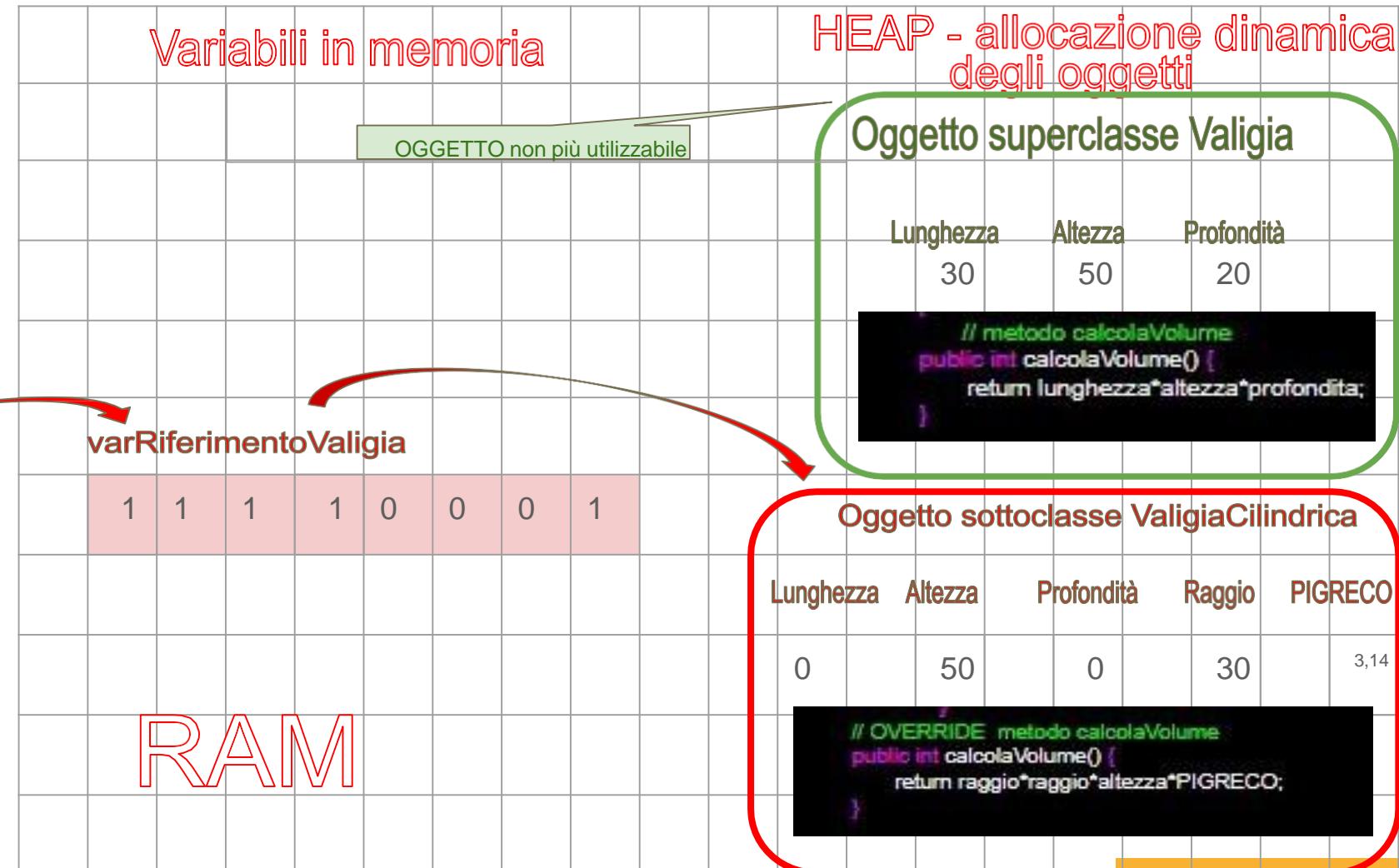
```
Valigia varRiferimentoValigia;
varRiferimentoValigia = new Valigia(30,50,20);
int volume = varRiferimentoValigia.calcolaVolume();
```



- Caratteristiche del linguaggio: Polimorfismo dinamico 2

VARIABILE DI RIFERIMENTO
area di memoria il cui valore è un riferimento ad un oggetto

```
Valigia varRiferimentoValigia;  
  
varRiferimentoValigia = new Valigia(30,50,20);  
int volume = varRiferimentoValigia.calcolaVolume()  
  
varRiferimentoValigia = new ValigiaCilindrica(0,50,0,30);  
int volume = varRiferimentoValigia.calcolaVolume();
```

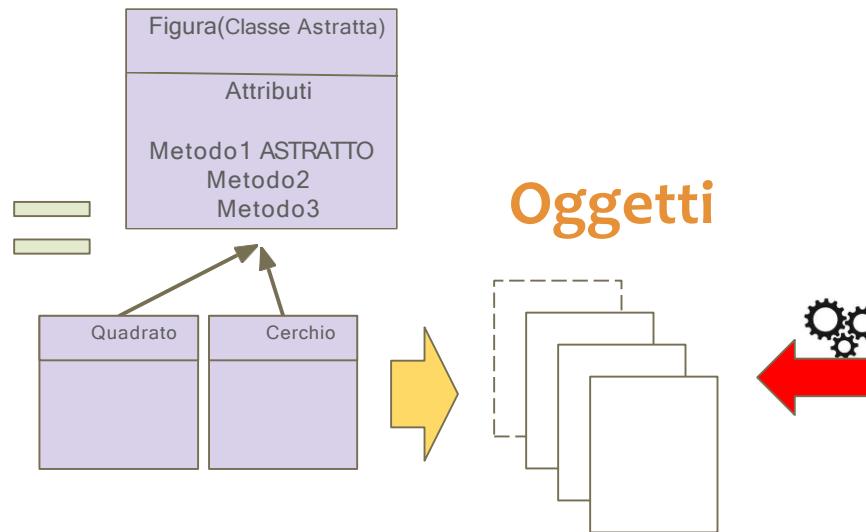


• Caratteristiche del linguaggio: Classi Astratte

```
abstract class Figura{
    // proprietà
    private String colore;
    // metodo astratto calcolaArea
    abstract public double calcolaArea();
}
```

```
class Quadrato extends Figura {
    // proprietà
    private double lato;
    // metodo costruttore
    public Quadrato(String c, double l) {
        super(c);
        lato = l;
    }
    // OVERRIDE metodo calcolaArea
    public double calcolaArea() {
        return lato*lato;
    }
}
```

```
class Cerchio extends Figura {
    // proprietà
    private double raggio;
    final double PIGRECO = 3,14159;
    // metodo costruttore
    public Cerchio(String c, double r) {
        super(c);
        raggio = r;
    }
    // OVERRIDE metodo calcolaArea
    public double calcolaArea() {
        return raggio*raggio*PIGRECO ;
    }
}
```



- deve contenere almeno un metodo astratto (dichiarazione del metodo senza corpo)
 - non può essere istanziata
- le sottoclassi DEVONO ridefinire i metodi astratti (OVERRIDE)
- sarà possibile comunque dichiarare una variabile di riferimento cui assegnare il riferimento ad oggetti delle sue sottoclassi (POLIMORFISMO DINAMICO)

```
class Classe_Principale {
    public static void main (String [] args) {
```

// STEP 1 : dichiarazione variabile di riferimento della superclasse

```
Figura varRiferimentoFigura;
```

// STEP 2 : creazione istanza (oggetto) della superclasse

```
varRiferimentoFigura = new Figura();
```

// STEP 3 : creazione istanza (oggetto) della sottoclasse

```
varRiferimentoFigura = new Quadrato("rosso",2);
double area = varRiferimentoFigura.calcolaArea();
// 4
```

```
varRiferimentoFigura = new Cerchio("verde",2);
double area = varRiferimentoFigura.calcolaArea();
// 12,56
}
```

• Caratteristiche del linguaggio: Parola chiave FINAL

Applicata alle classi

```
final class Valigia {

    // proprietà
    private int lunghezza;
    private int altezza;
    private int profondita;
    // metodo costruttore
    public Valigia(int l, int a, int p) {
        lunghezza = l;
        altezza = a;
        profondita = p;
    }
    // metodo calcolaVolume
    public int calcolaVolume() {
        return lunghezza*altezza*profondita;
    }
}
```

```
class ValigiaCilindrica extends Valigia {

    // proprietà
    private int raggio;
    final double PIGRECO = 3,14159;
    // metodo costruttore
    public ValigiaCilindrica(int l, int a, int p, int raggio) {
        super(l,a,p);
        this.raggio = raggio;
    }
    // OVERRIDE metodo calcolaVolume
    public int calcolaVolume() {
        return raggio*raggio*altezza*PIGRECO;
    }
}
```

VIETATO creare sottoclassi

Applicata ai metodi

```
class Valigia {

    // proprietà
    private int lunghezza;
    private int altezza;
    private int profondita;
    // metodo costruttore
    public Valigia(int l, int a, int p) {
        lunghezza = l;
        altezza = a;
        profondita = p;
    }
    // metodo calcolaVolume
    final public int calcolaVolume() {
        return lunghezza*altezza*profondita;
    }
}
```

```
class ValigiaCilindrica extends Valigia {

    // proprietà
    private int raggio;
    final double PIGRECO = 3,14159;
    // metodo costruttore
    public ValigiaCilindrica(int l, int a, int p, int raggio) {
        super(l,a,p);
        this.raggio = raggio;
    }
    // OVERRIDE metodo calcolaVolume
    public int calcolaVolume() {
        return raggio*raggio*altezza*PIGRECO;
    }
}
```

VIETATO ridefinire il metodo

Applicata alle VARIABILI di istanza

```
class Cerchio extends Figura {

    // proprietà
    private double raggio;
    final double PIGRECO = 3,14159;
    // metodo costruttore
    public Cerchio(double r) {
        raggio = r;
    }
    // OVERRIDE metodo calcolaArea
    public double calcolaArea() {
        return raggio*raggio*PIGRECO;
    }
}
```

VIETATO cambiare il valore della costante

```
class Classe_Principale {
    public static void main (String [] args) {

```

// STEP 1 : creazione di un'istanza (oggetto)

```
Cerchio varRiferimentoCerchio = new Cerchio(20);
```

```
varRiferimentoCerchio.PIGRECO = 6,23;
```

• Caratteristiche del linguaggio: Parola chiave STATIC

METODI STATIC O DI CLASSE

- Il metodo statico non richiede di istanziare oggetti per essere utilizzato;
- Si tratta di un metodo che generalmente prende in input uno o più parametri e NON usa variabili di istanza nel proprio corpo;
- potrà essere usato in tutto il programma che accede alla classe che lo contiene;
- può a sua volta richiamare altri metodi statici definiti nella stessa classe;
- può accedere solo a variabili statiche definite nella stessa classe di solito trova utilizzo in classi generiche che contengono un certo numero di metodi statici di utilità (Es. classe Math, classe System)

```

class Utility{

    // metodo somma 2 numeri
    public static int somma(int a, int b) {
        return a + b;
    }

    class Calcolatrice {
        public static void main(String args[ ]) {
            int risultato = Utility.somma(5,8);
            System.out.println("Il risultato è: " + risultato);
        }
    }
}

```

UTILIZZO DI UNA
VARIABILE STATICÀ
NomeClasse.variabile

CHIAMATA DI UN
METODO STATICO
NomeClasse.metodo()

VARIABILI STATIC O DI CLASSE

- Le variabili static rappresentano proprietà comuni a tutte le istanze della classe (il loro valore è condiviso);
- se un oggetto modifica una variabile static il suo nuovo valore sarà visibile a tutti gli oggetti, istanze di quella classe;
- le variabili statici potranno essere dichiarate PUBLIC O PRIVATE (con PUBLIC, la variabile sarà accessibile anche al di fuori della classe)

```

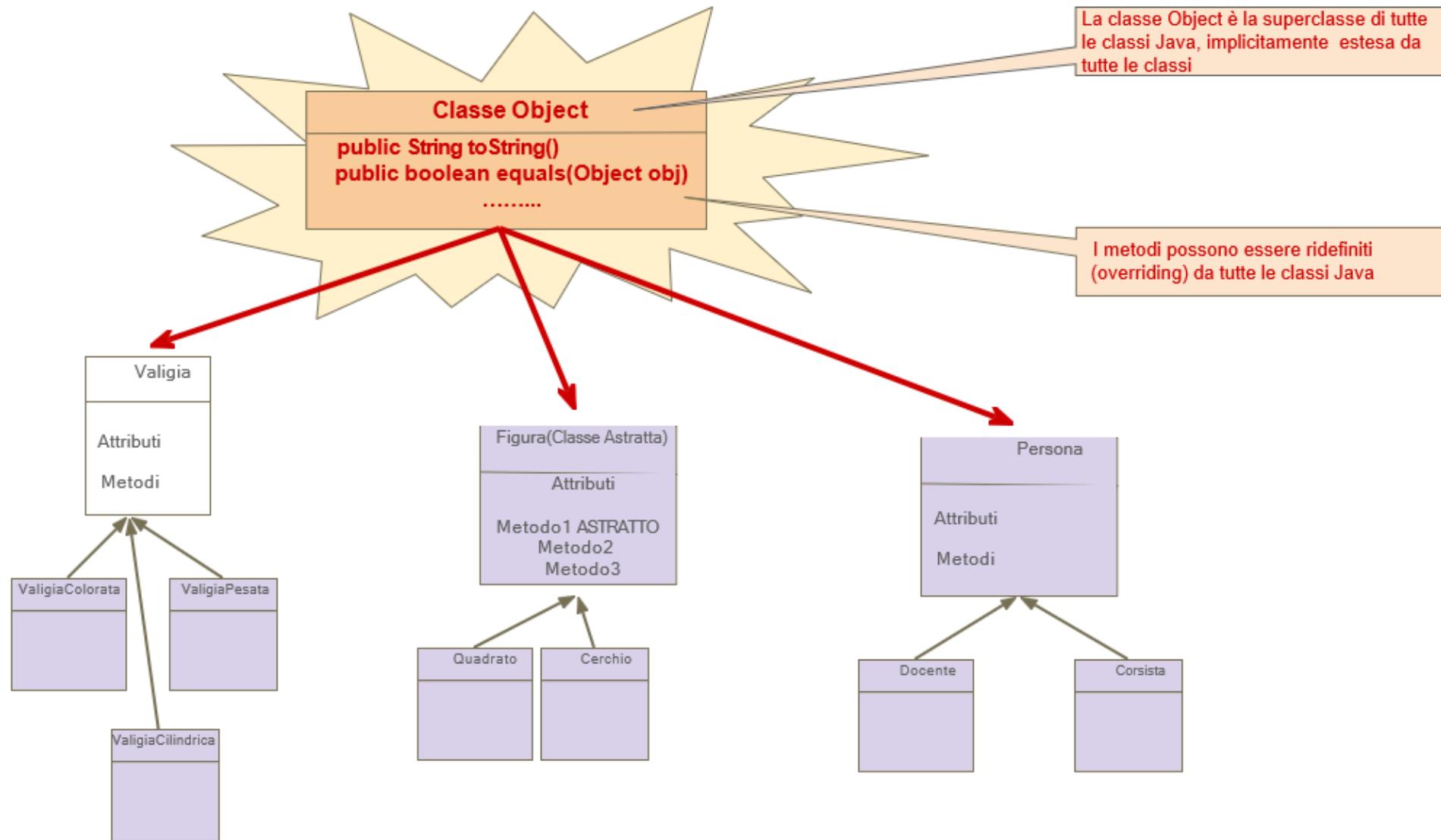
public class Articolo {
    public static int numeroArticoli=0;

    public Articolo() {
        numeroArticoli= numeroArticoli+1;
    }

    public class UsaArticolo {
        public static void main(String[] args) {
            Articolo art1= new Articolo();
            Articolo art2= new Articolo();
            Articolo art3= new Articolo();
            System.out.println("il numero di articoli creato è :" +
                Articolo.numeroArticoli);
            // stamperà 3
        }
    }
}

```

• Caratteristiche del linguaggio: Classe Object



- Caratteristiche del linguaggio: metodi `toString()` ed `equals(Object obj)`

```

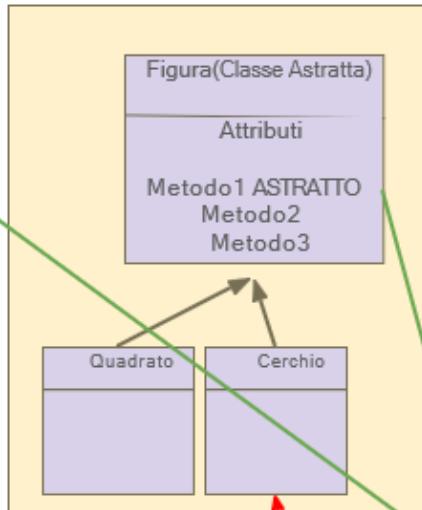
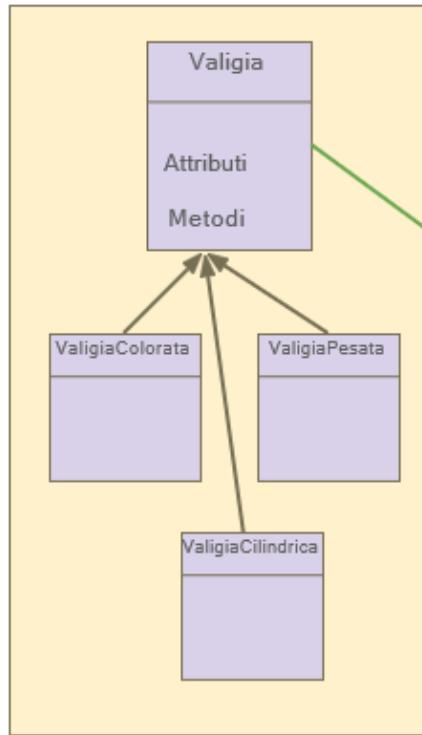
class Valigia {

    // proprietà
    private int lunghezza;
    private int altezza;
    private int profondita;
    // metodo costruttore
    public Valigia(int l, int a, int p) {
        lunghezza = l;
        altezza = a;
        profondita = p;
    }
    // metodo calcolaVolume
    public int calcolaVolume() {
        return lunghezza*altezza*profondita;
    }
    // metodo toString
    public String toString(){
        return "altezza: " + altezza + "lunghezza: " + lunghezza+
            "profondita" + profondita;
    }
    // metodo equals
    public boolean equals (Object obj) {
        Valigia val= (Valigia) obj;
        If (this.altezza == val.altezza && this.lunghezza == val.lunghezza &&
        this.profondita== val.profondita) {
            return true;
        }
        return false;
    }
}

```

Quando verrà chiamato questo metodo nella classe principale, stamperà direttamente i valori delle variabili di istanza

• Caratteristiche del linguaggio: INTERFACCE

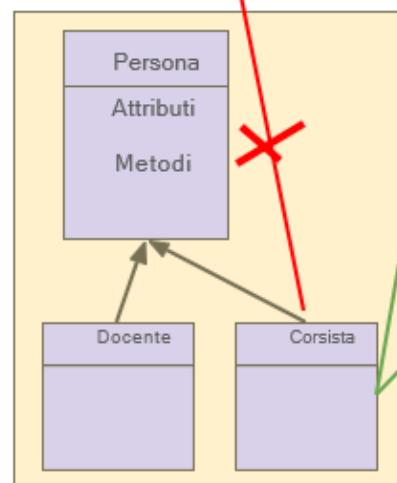


```

class Corsista extends Persona implements
    interfaccia1, interfaccia2 {

    private int n_presenze;

    public void trasla (from_x, from_y, to_x, to_y) {
        .....
        .....
    }
}
  
```



Le interfacce stabiliscono una convenzione tra le classi e il mondo esterno:

1. Le interfacce contengono costanti e/o metodi astratti ;
2. le interfacce non si possono istanziare;
3. l'interfaccia consente di parallelizzare la scrittura del codice delle classi e la scrittura del codice del main;
4. sarà possibile dichiarare una variabile di riferimento cui assegnare il riferimento ad oggetti delle classi che implementano l'interfaccia (POLIMORFISMO DINAMICO) ;
5. con java8 è possibile implementare il codice del metodo anche nell'interfaccia;
6. nell'interfaccia NON sarà presente:
 - a. costruttore
 - b. variabili di istanza



```

class Classe_Principale {
    public static void main (String [] args) {
        Interfaccia1 var_rif_interfaccia1;
        var_rif_interfaccia1 = new
            ValigiaColorata ();
        var_rif_interfaccia1.trasla(2,3,5,7);

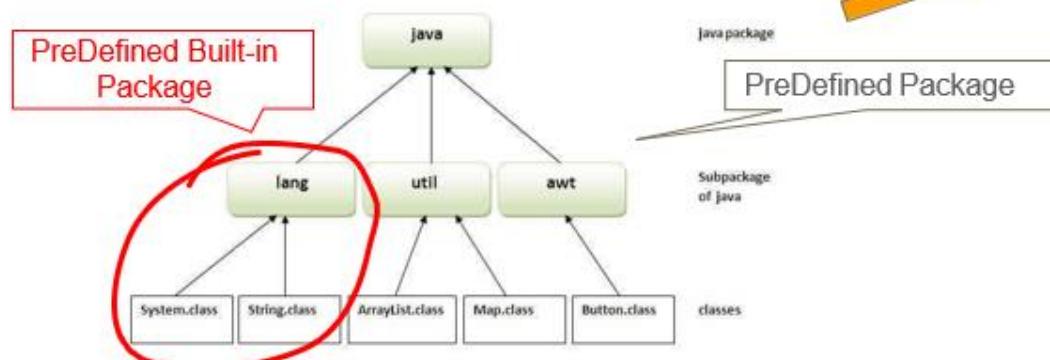
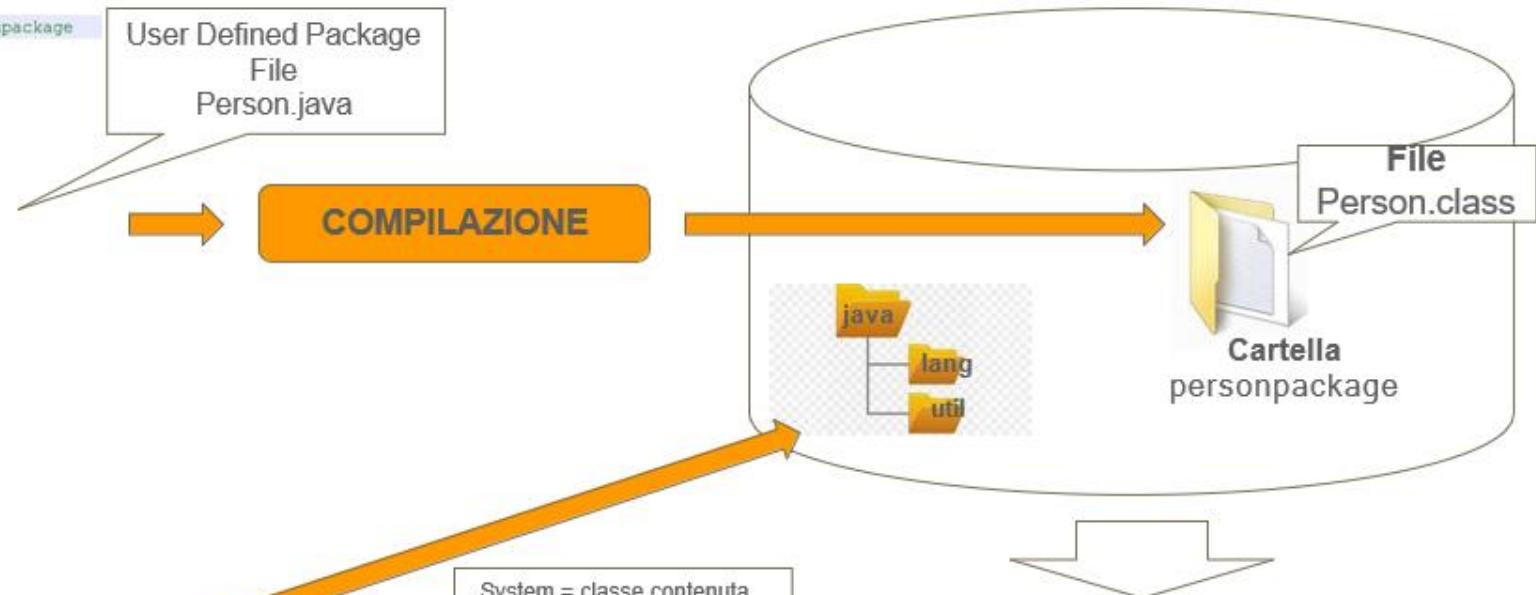
        var_rif_interfaccia1 = new
            Docente();
        var_rif_interfaccia1.trasla(3,7,8,17);

        var_rif_interfaccia1 = new Cerchio();
        var_rif_interfaccia1.trasla(1,6,2,3);
    }
}
  
```

• Caratteristiche del linguaggio: PACKAGE = LIBRERIE

```

1 // Indicate to compiler that the Person class belongs in the personpackage
2 package personpackage;
3 public class Person {
4     // Private variables
5     private String firstName;
6     private String lastName;
7
8     // Getters and setters for variables
9     public String getFirstName() {
10         return firstName;
11     }
12     public void setFirstName (String firstName) {
13         this.firstName=firstName;
14     }
15     public String getLastName() {
16         return lastName;
17     }
18     public void setLastName (String lastName) {
19         this.lastName=lastName;
20     }
21 }
```



System = classe contenuta nel package lang, predefinito e già incorporato

Scanner = classe contenuta nel package util, predefinito e da importare

setFirstName = metodo della classe contenuta nel package definito dall'utente "personpackage"

```

import java.util.Scanner;
import personpackage.*;
class Classe_Principale {
    public static void main (String [] args) {
        Person persona = new Person();
        System.out.println("Inserisci il nome: ");
        Scanner tastiera = new Scanner(System.in);
        String nome = tastiera.nextLine();
        persona.setFirstName(nome);
    }
}
```

• Caratteristiche del linguaggio: le eccezioni - dove accadono



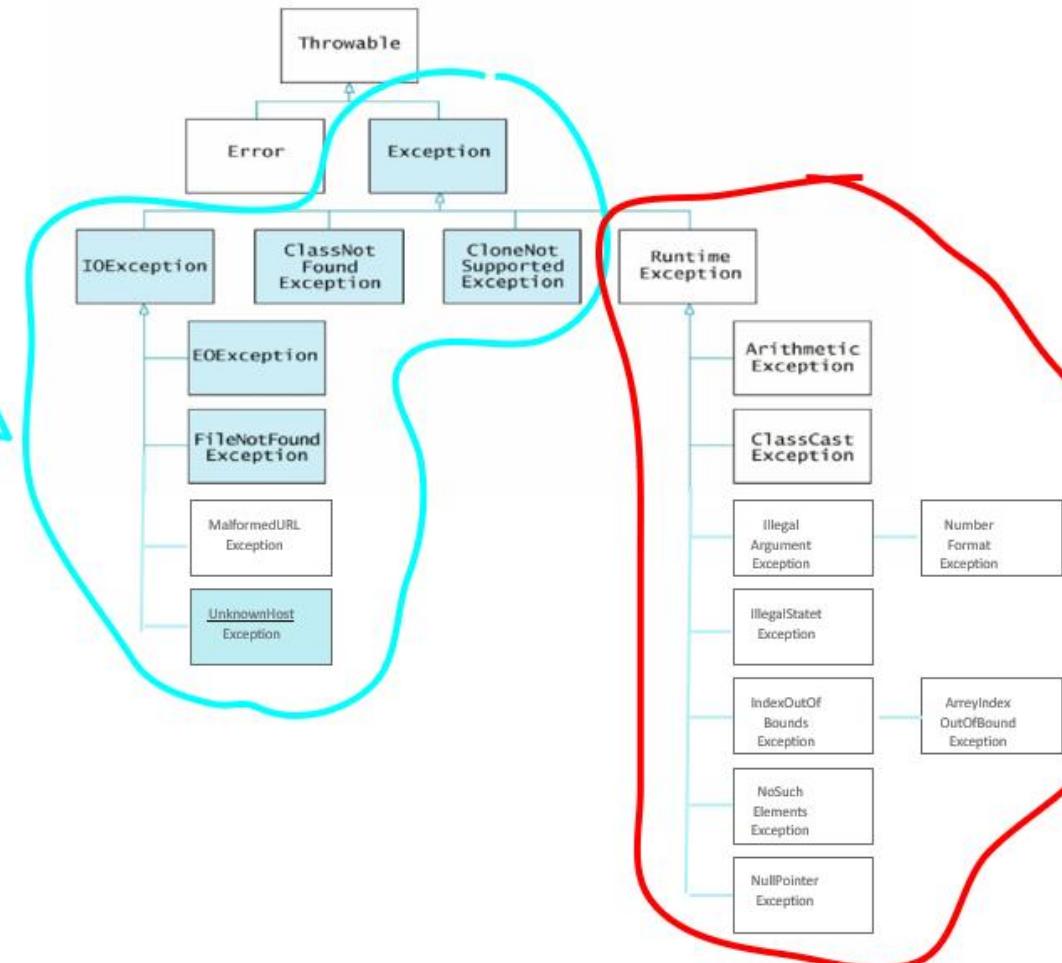
Un'**ECCEZIONE** è un evento che interrompe il normale flusso di esecuzione del **RUN** del programma, infatti esse si verificano durante l'esecuzione del programma (box giallo n° 3)

• Caratteristiche del linguaggio: le eccezioni - due scenari

le "Eccezioni controllate" dal compilatore, richiedono obbligatoriamente il TRY-CATCH

o il THROWS (altrimenti il compilatore restituisce un errore ed interrompe la compilazione) e.....nel caso in cui si verificano,

il programma proseguirà l'elaborazione eseguendo le istruzioni contenute nel CATCH



le "Eccezioni NON controllate" dal compilatore, non richiedono obbligatoriamente il TRY-CATCH e il THROWS ma.....nel caso in cui si verificano,

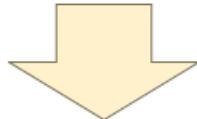
il programma andrà in crash, interrompendo l'esecuzione



NB è comunque consigliato gestire anche le eccezioni non controllate con un try-catch

• Caratteristiche del linguaggio: eccezioni non controllate

Possibile eccezione non controllata dal compilatore e non gestita dal Try-Catch



```
class ClasseMedia {
    // metodo calcolaMedia
    public double calcolaMedia(int [] arrayDiNumeri) {
        int somma = 0;
        for (int x = 0;x < arrayDiNumeri.length;x++)
            somma = somma + arrayDiNumeri[x];
        return somma/arrayDiNumeri.length;
    }
}
```

in assenza di errore/ eccezione,
l'elaborazione prosegue regolarmente

1

in presenza di
errore/eccezione
Arithmetic Exception,
l'elaborazione verrà
interrotta

2

in presenza di errore/eccezione, gestita
dal Try-Catch
l'elaborazione prosegue regolarmente

3

```
class Media {
    public static void main (String [] args) {

        int n[ ] = {3,4,5,3};
        ClasseMedia m = new ClasseMedia();
        double risultato = m.calcolaMedia(n);
        System.out.println(" La media è : " + risultato);
        // stamperà : La media è : 3,75
    }
}
```

Array non valorizzato

```
int n[ ] = { };
ClasseMedia m = new ClasseMedia();
double risultato = m.calcolaMedia(n);
System.out.println(" La media è : " + risultato);
// stamperà :

Exception in thread "main" java.lang.ArithmetricException: / by zero
at ClasseMedia.CalcolaMedia(ClasseMedia.java:7)
at Media.main(Media.java:7)
```

```
int n[ ] = {};
ClasseMedia m = new ClasseMedia();
try {
    double risultato = m.calcolaMedia(n);
    System.out.println(" La media è : " + risultato);
} catch ( ArithmetricException a ) {
    System.out.println(" Impossibile calcolare la media");
} finally {
    System.out.println(" - Fine Elaborazione - ");
}
// stamperà : Impossibile calcolare la media
// - Fine elaborazione -
}
```

• Caratteristiche del linguaggio: eccezioni controllate

```
import java.io.FileReader
public class EsempioEccControllate {
    // metodo leggiFile
    public void leggiFile() {
        FileReader f = new FileReader("prova");
    }
}
```

1

Errore in compilazione
 error: unreported exception FileNotFoundException;
 must be caught or declared to be thrown
 FileReader f = new FileReader ("prova");
 a causa di possibile eccezione
 controllata e assenza del Try.Catch o
 del Throws

```
import java.io.FileReader
public class EsempioEccControllate {
    // metodo leggiFile
    public void leggiFile() throws Exception {
        FileReader f = new FileReader("prova");
    }
}
```

2

CLASSE COMPILATA CON SUCCESSO
 la gestione della possibile eccezione,
 utilizzando Throws nella dichiarazione
 del metodo, è rimandata al metodo
 chiamante che dovrà gestirlo
 obbligatoriamente con try catch

```
import java.io.FileReader
public class EsempioEccControllate {
    // metodo leggiFile
    public void leggiFile() {
        try {
            FileReader f = new FileReader("prova");
        } catch ( Exception e) {
            System.out.println("File non trovato")
        }
    }
}
```

3

CLASSE COMPILATA CON SUCCESSO
 la possibile eccezione viene gestita ,
 utilizzando Try-Catch nel corpo del
 metodo chiamato

```
public class MainEsempioEccControllate{
```

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

in presenza di
 errore/eccezione gestita
 con Try-Catch :
 l'elaborazione prosegue
 regolarmente

```
        EsempioEccControllate ec = new EsempioEccControllate();
```

```
        try {
```

```
            ec.leggiFile(); }
```

```
        catch(Exception e){
```

```
            System.out.println("File non trovato"); }
```

```
        finally { System.out.println("Fine gestione File") }
```

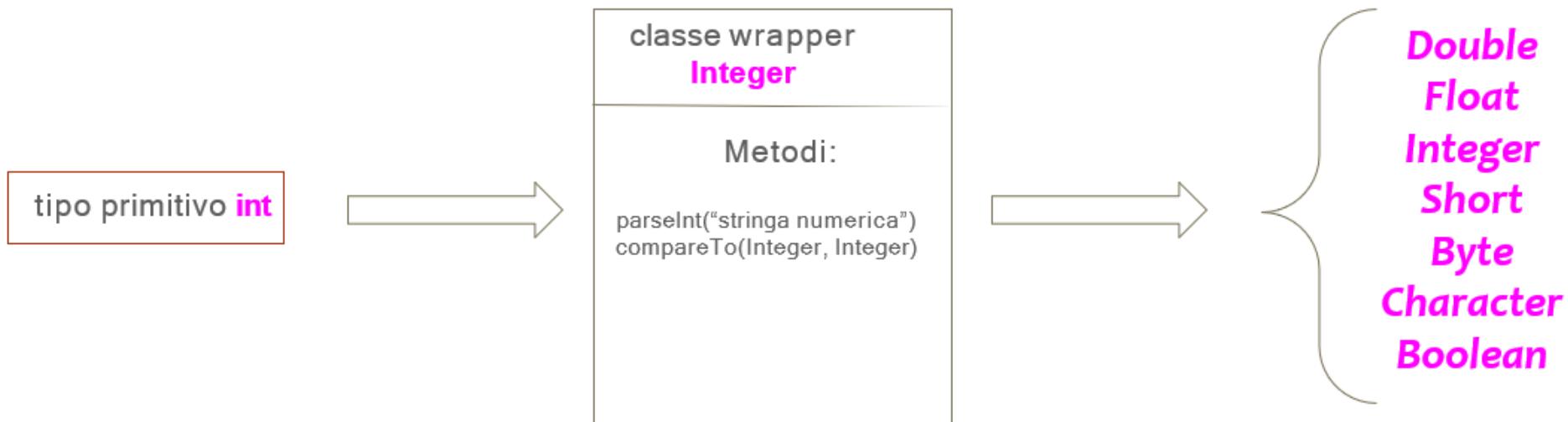
Il finally verrà
 eseguito in ogni caso

```
        EsempioEccControllate ec1 = new EsempioEccControllate();
```

```
        ec1.leggiFile();
```

```
    }
```

• Caratteristiche del linguaggio: Wrapper Classes



```
int a = 5; // OK
int b = NULL; // ERRORE
```

Un tipo primitivo non può contenere un NULL ma solo i valori previsti nel range specifico di quel tipo

```
Integer a = new Integer (5); // OK
Integer a = 5; // OK Autoboxing
Integer b = NULL; // OK
```

il valore 5 viene automaticamente incorporato in un oggetto

E' OK perchè b è una variabile di riferimento di un oggetto

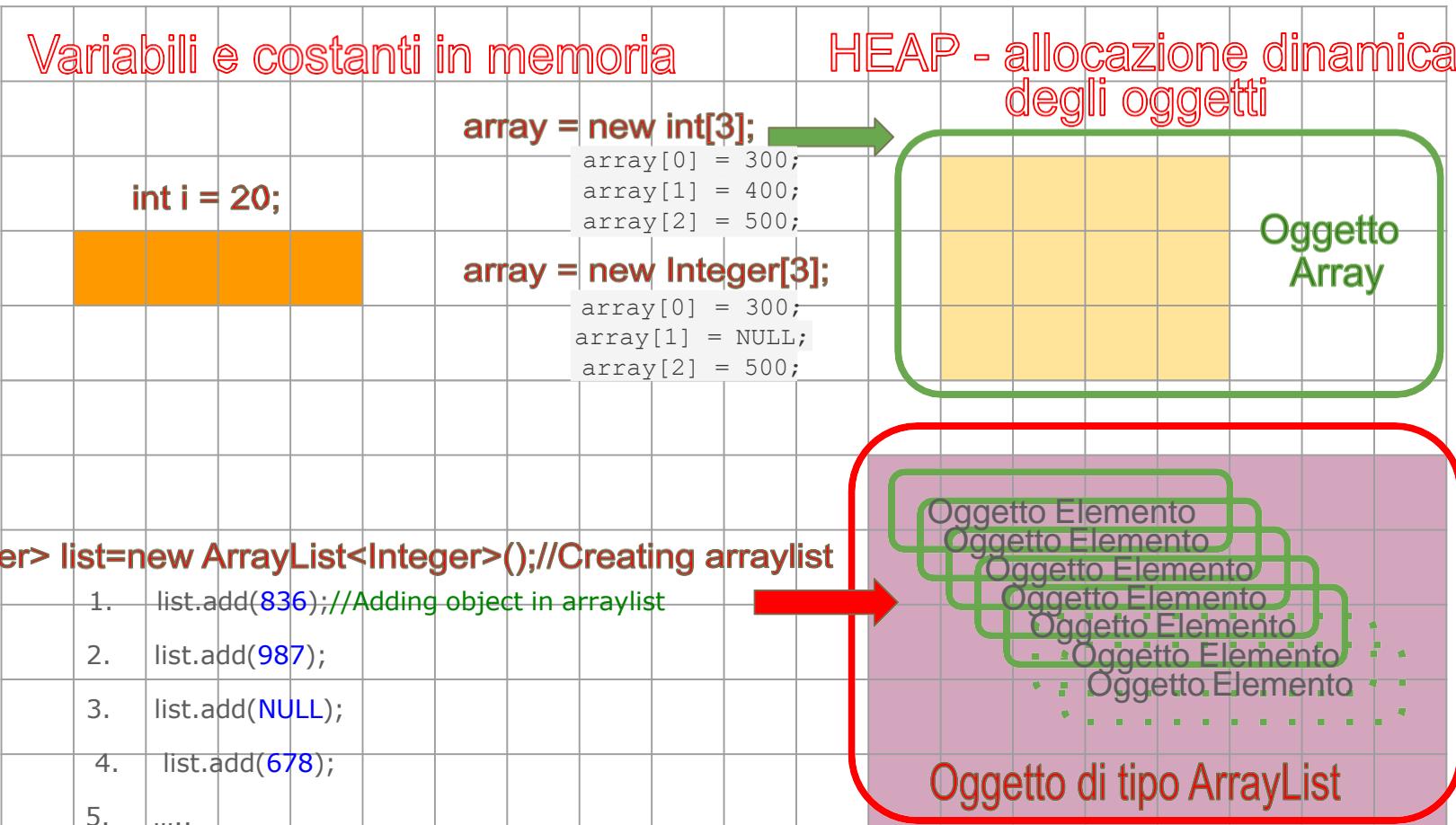
• Caratteristiche del linguaggio: COLLECTION

RAM

ArrayList<Integer> list=new ArrayList<Integer>(); //Creating arraylist

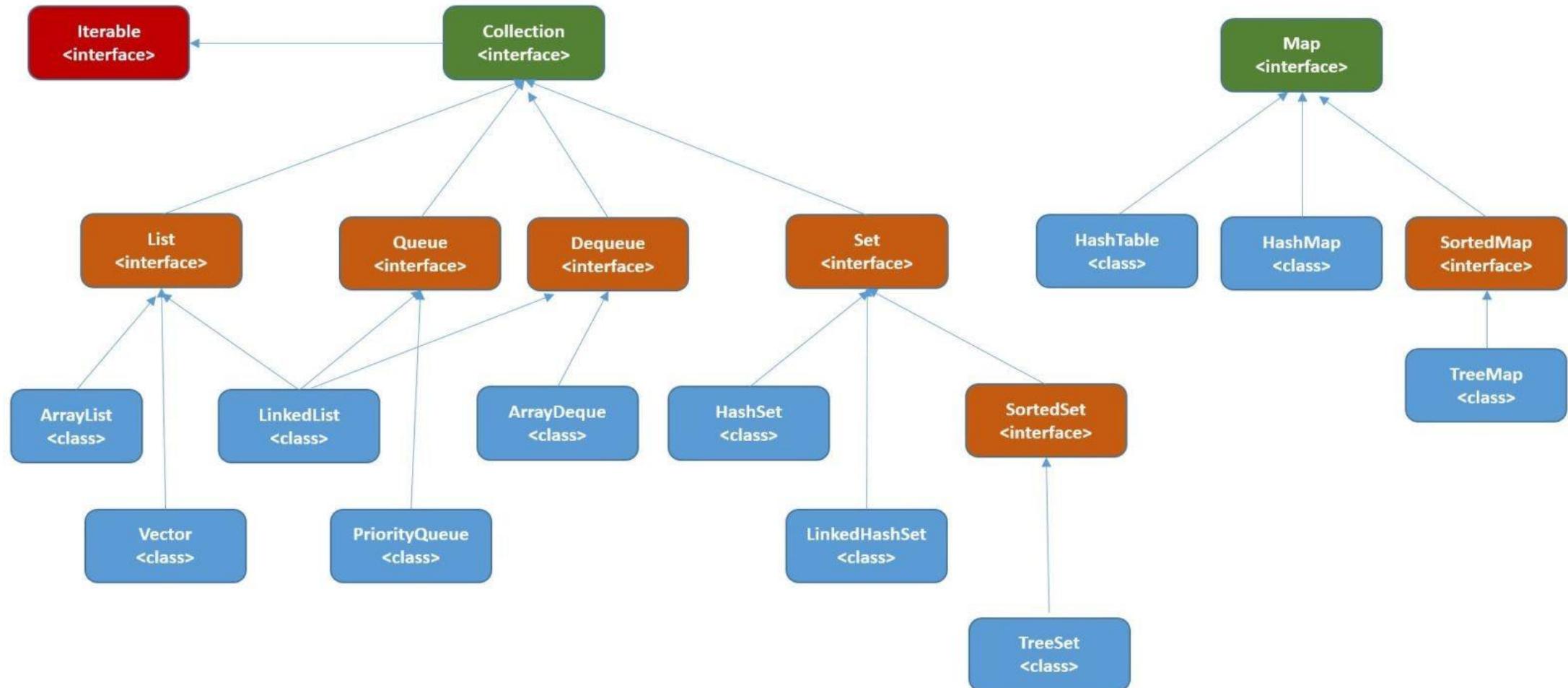
1. list.add(836); //Adding object in arraylist
2. list.add(987);
3. list.add(NULL);
4. list.add(678);
5.

GENERICs = il compilatore si assicura che le tue istruzioni usino la collection con un tipo di dato compatibile



- Caratteristiche del linguaggio: collection framework

Collection Framework Hierarchy



• Caratteristiche del linguaggio: Collection - Iteratore

Scorrimento con Iteratore

DEFINIZIONE:

La classe iterator è il modo standard per scorrere gli elementi di una qualunque collection.

Contiene 3 metodi:

1. `hasNext`
2. `next`
3. `remove`

Il metodo remove elimina dalla collection l'ultimo elemento restituito

```
// crea un array list
ArrayList<String> al = new ArrayList<String>();

// aggiungi elementi all'array list
al.add("ciao");
al.add("a");
al.add("tutti");

// scorro per visualizzare il contenuto dell'array
Iterator<String> altr = al.iterator();
while(altr.hasNext())
    System.out.print(altr.next()+" ");
```

il metodo hasNext () restituisce TRUE se ci sono ancora elementi da scorrere

il metodo next () restituisce l'elemento successivo

Scorrimento senza Iteratore

```
// crea un array list
ArrayList<String> al = new ArrayList<String>();
```

```
// aggiungi elementi all'array list
al.add("ciao");
al.add("a");
al.add("tutti");
```

```
// scorro per visualizzare il contenuto dell'array
for(String a : al){
    System.out.println(a);
}
```

