

# Programmazione I-B 2020-21

Laboratorio T2

Attilio Fiandrotti

[attilio.fiandrotti@unito.it](mailto:attilio.fiandrotti@unito.it)

8 Ottobre 2020

# Outline

- Revisione esercizio 3 settimana scorsa
- Scheletro di una applicazione Java
- Dichiarazione variabili e assegnazione valore
- Salti condizionali if-else
- Iterazioni con il ciclo *while*

# Revisione

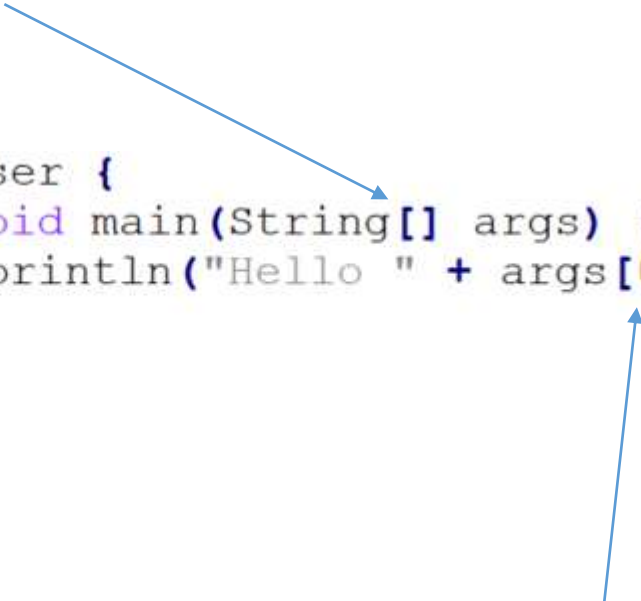
# ESERCIZIO 3 Lezione 1 Ottobre

- Scrivere un file batch *HelloWorld.bat* che
  1. chieda all'utente di inserire il proprio nome e salvi tale informazione in una variabile chiamata *mioNome*
  2. invochi il programma java *HelloUser* passandogli la variabile *mioNome* come argomento in modo che *HelloUser* stampi il nome a schermo
- Come invocare il programma Java dall'interno dello script batch ?
- Come passare il nome utente memorizzato come *mioNome* come argomento da linea di comando di *HelloUser* ?

# ESERCIZIO 3 – HelloUser

I parametri della linea di comando  
sono passati come array di *String*

```
public class HelloUser {  
    public static void main(String[] args) {  
        System.out.println("Hello " + args[0] + " !");  
    }  
}
```



Stampo il primo parametro passato da  
linea di comando (primo elemento di un  
*array* di stringhe di caratteri)

# ESERCIZIO 3 – Soluzione

Nasconde il prompt dei comandi

Memorizza il nome utente  
nella variabile *mioNome*

```
@echo off  
set /p mioNome="Inserisci il tuo nome: "  
java HelloUser %mioNome%
```

Invoca *HelloUser* passando  
*mioNome* come argomento  
della linea di comando

# La programmazione in Java

# ESERCIZIO 1 Lezione 1 Ottobre


- Scriviamo un programma *HelloWorld* che stampi a video la scritta «Hello World!»

```
/* Commento su
 * più linee
 */
public class HelloWorld{
    //Commento su una linea sola
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```



# HelloWorld in Java

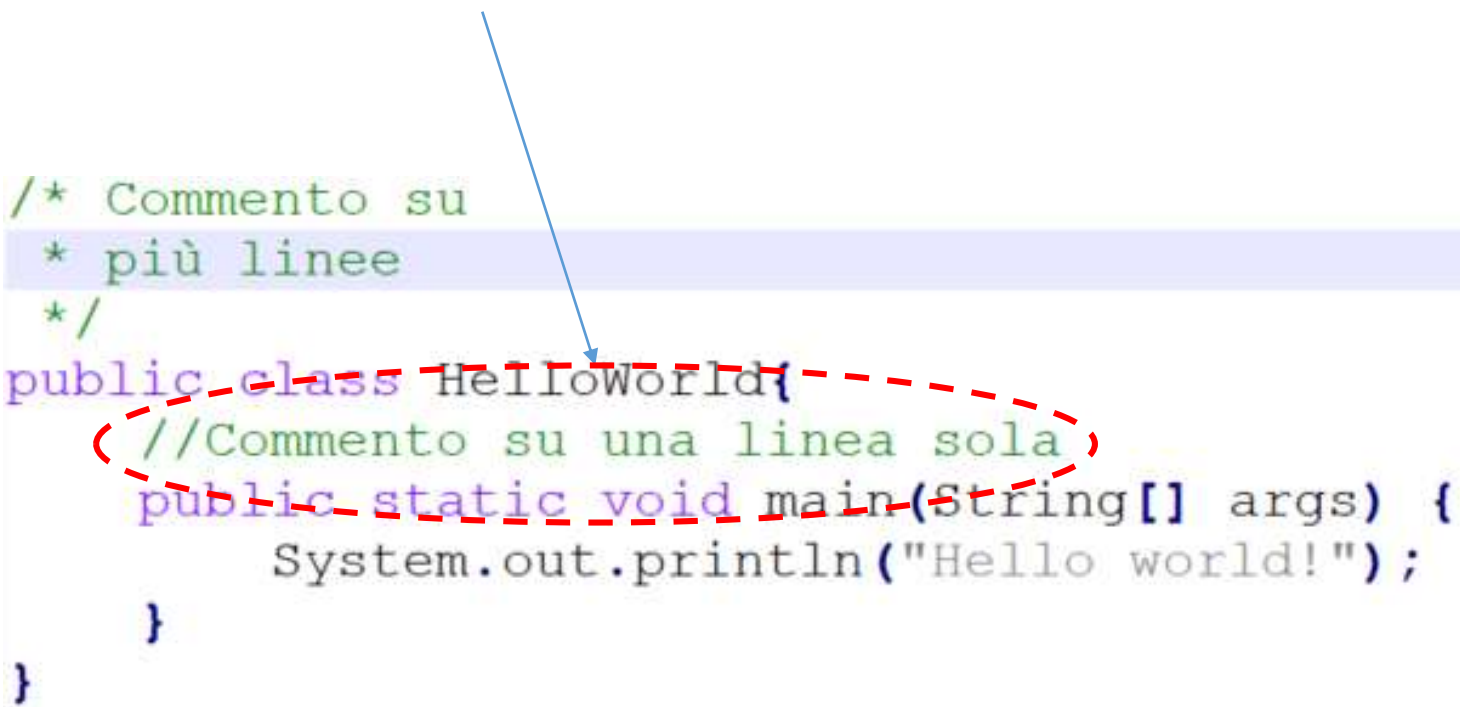
Commento su più righe: tutto ciò che sta fra «/\*» e «\*/» sarà ignorato dal compilatore javac



```
/* Commento su
 * più linee
 */
public class HelloWorld{
    //Commento su una linea sola
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

# HelloWorld in Java

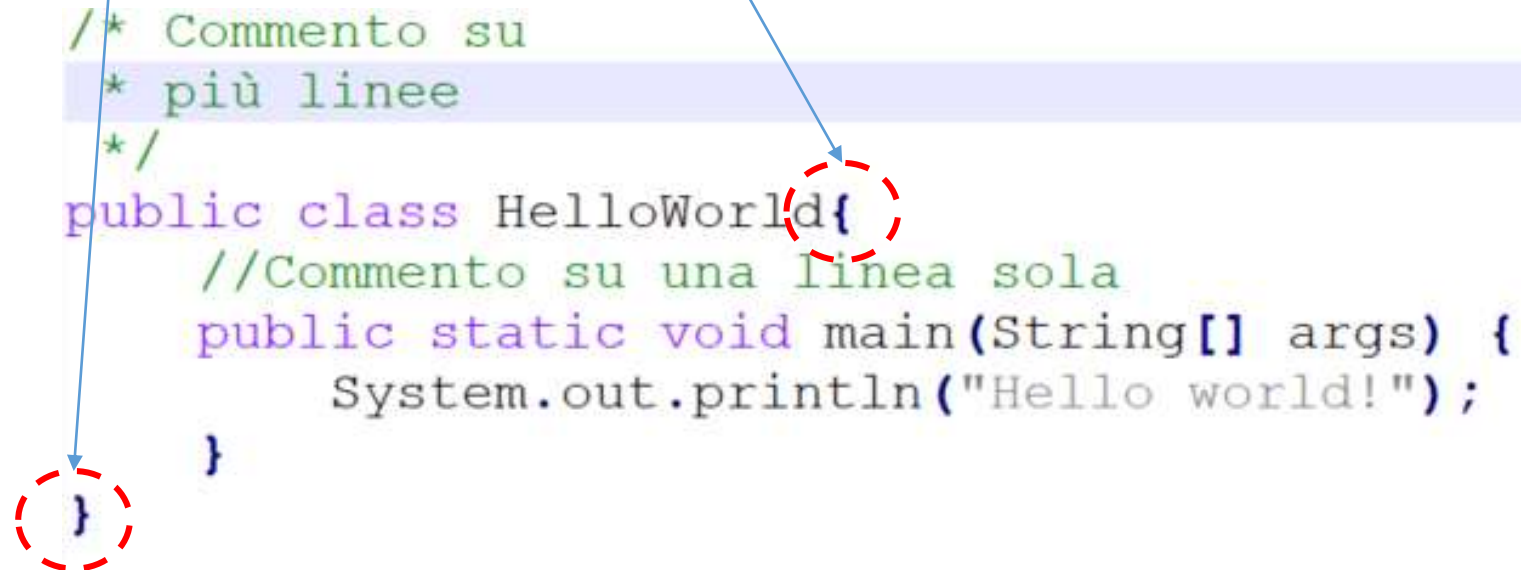
Commento su riga singola: tutto ciò che segue «//» fino alla fine della riga sarà ignorato dal compilatore



```
/* Commento su
 * più linee
 */
public class HelloWorld{
    //Commento su una linea sola
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

# HelloWorld in Java

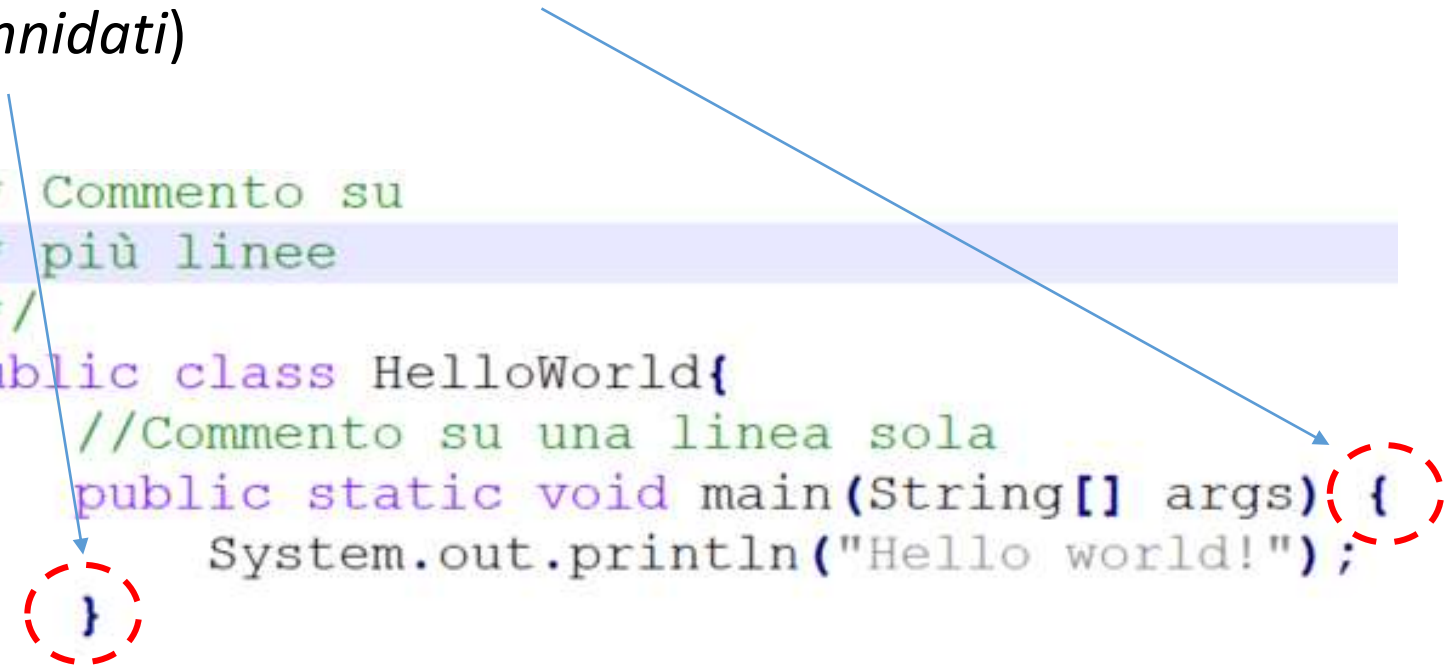
Ciò che é compreso fra «{» e «}»  
é noto come un *blocco di codice*



```
/* Commento su
 * più linee
 */
public class HelloWorld{
    //Commento su una linea sola
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

# HelloWorld in Java

Un blocco di codice può  
contenerne altri al suo interno  
(blocchi *annidati*)



```
/* Commento su
 * più linee
 */
public class HelloWorld{
    //Commento su una linea sola
    public static void main(String[] args){
        System.out.println("Hello world!");
    }
}
```

# HelloWorld in Java

Per convenzione, si *indenta* di un livello ogni blocco di codice (con spazio o tabulazione)

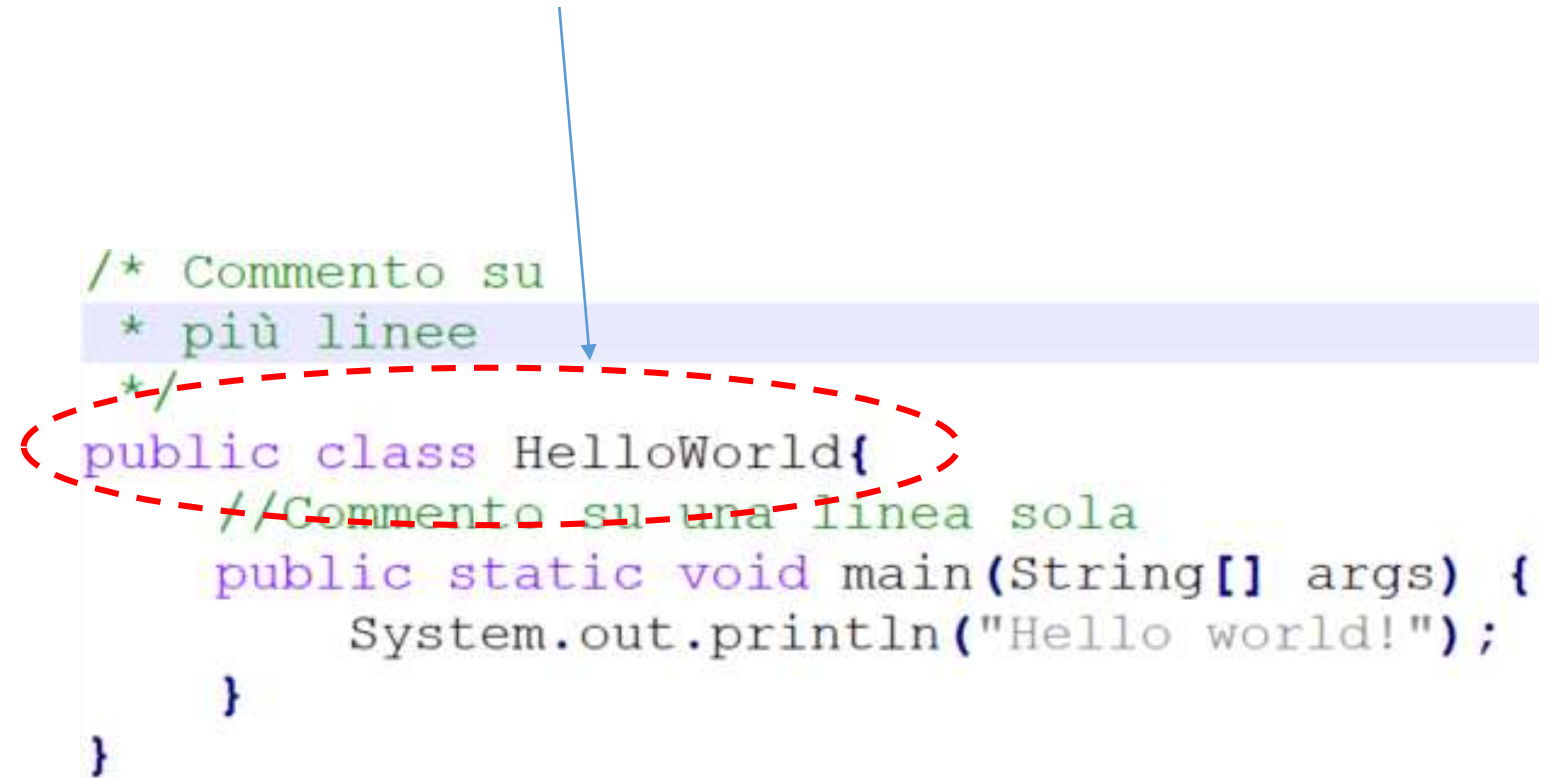
Indentazione di 1 livello

```
/* Commento su
 * più linee
 */
public class HelloWorld{
    //Commento su una linea sola
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Indentazione di 2 livello

# *HelloWorld* in Java

Definizione della classe Java *HelloWorld*



```
/* Commento su
 * più linee
 */
public class HelloWorld{
    //Commento su una linea sola
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

# Le Classi in Linguaggio Java – I

- Java é un linguaggio *ad oggetti* (object-based programming)
  - Molto utile per organizzare il codice
- Iniziano sempre per lettera maiuscola
- Le classi sono *modelli* per concetti astratti e reali
  - Es: per l'esercizio della telefonata creerò una classe *Telefono*
- Le classi includono *variabili* e *metodi* per modellare stato e funzioni
  - Es: lo stato del telefono sarà codificato come *stato = {libero, occupato}*
  - Es: le funzioni del telefono sono modellate dai metodi *chiama()*, *aggancia()*,...
- Una classe Java  $\Leftrightarrow$  un file sorgente con nome identico
  - Es classe *Telefono* definita in *Telefono.java*

# Le Classi in Linguaggio Java – II

- E' necessario creare una classe di Test (*TestCase*)
  - Es: classe *TelefonoTest* per testare *Telefono*
- La classe di test
  - Crea un'istanza della classe da testare
  - Invoca il(i) metodo(i) opportuno(i) (con relativi parametri)
  - Verifica che l'output sia quello atteso
- Es: crea istanza di *Telefono* e invoca il metodo *chiama(011987654)*

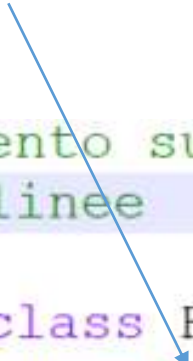


# Le classi in MFN0582

- Tralascieremo l'aspetto orientato alle classi di Java
  - Programmazione strutturata C-like
- Una sola classe in un solo file sorgente (per ora)
- La classe sarà sempre *pubblica*
  - «*public*» anteposto a dichiarazione di classe
- Un metodo `main()` per l'ingresso nel programma
  - Tutto il nostro codice nell'unico metodo *main*

# *HelloWorld* in Java

Definizione del *metodo* principale *main*



```
/* Commento su
 * più linee
 */
public class HelloWorld{
    //Commento su una linea sola
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

# I Metodi

- I metodi rappresentano un blocco di codice invocabile dall'esterno
  - Per noi equivalenti a funzioni
- Iniziano per lettera minuscola
- Possono accettare in input opportuni parametri
- Possono restituire (*ritornare*) dei valori
- Analogia matematica: la funzione trigonometrica *sin()*

The diagram shows the equation  $y = \sin(x)$  in the center. A blue arrow points from the text "y é il valore ritornato" (y is the returned value) to the variable  $y$  on the left side of the equation. Another blue arrow points from the text "x é il parametro" (x is the parameter) to the variable  $x$  inside the parentheses on the right side of the equation.

$y = \sin(x)$

y é il valore ritornato      x é il parametro

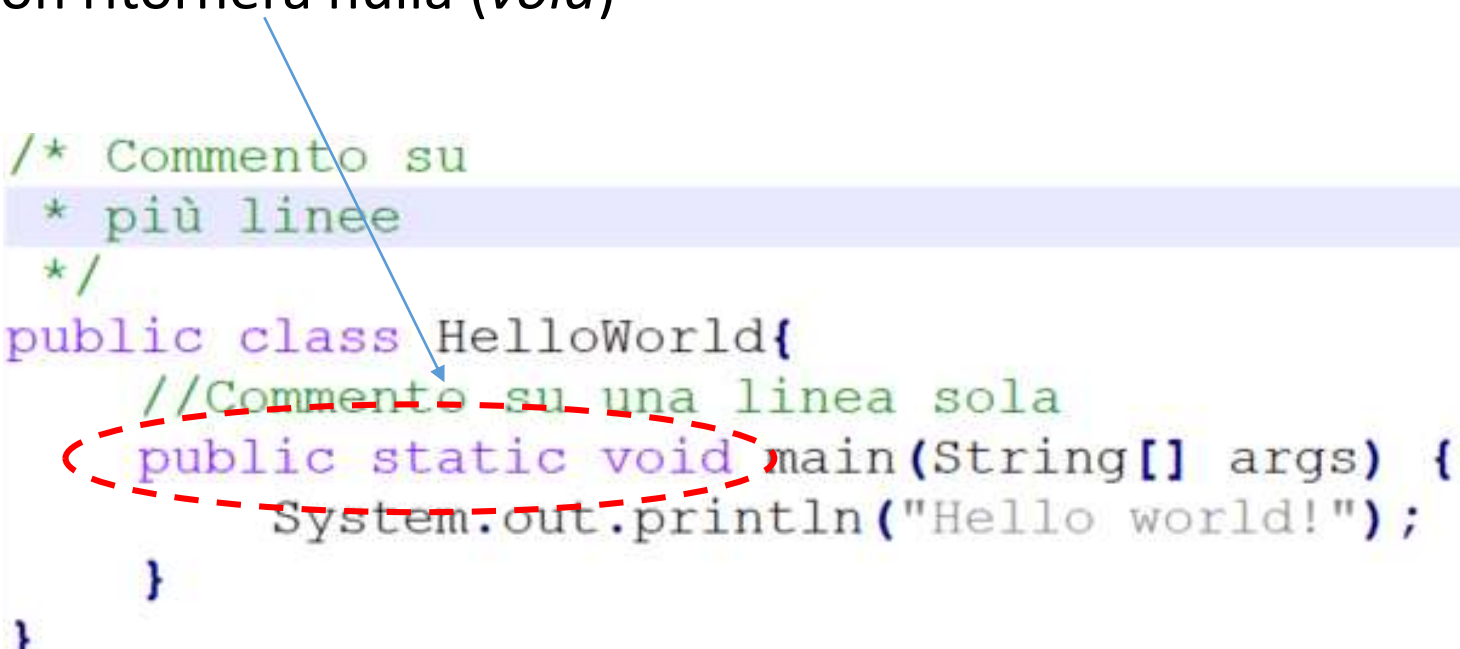
# Il Metodo *main*

- Ve ne deve essere esattamente uno per ogni programma
- E' il punto di ingresso (e uscita) del nostro programma
- Dichiarato obbligatoriamente come *public static void*
- Rende disponibili i parametri passati via linea da comando al programma Java nell'array di stringhe *args*
- Per ora tutto il nostro codice sarà contenuto all'interno del metodo *main*

# HelloWorld in Java

Il *metodo* principale *main* sarà sempre *pubblico* e *statico* e non ritornerà nulla (*void*)

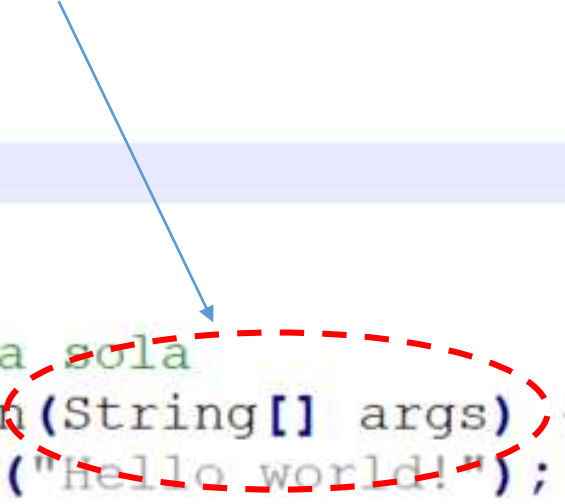
```
/* Commento su
 * più linee
 */
public class HelloWorld{
    //Commento su una linea sola
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```



# *HelloWorld* in Java

Vettore di Stringhe contenenti i parametri passati come argomenti via linea di comando

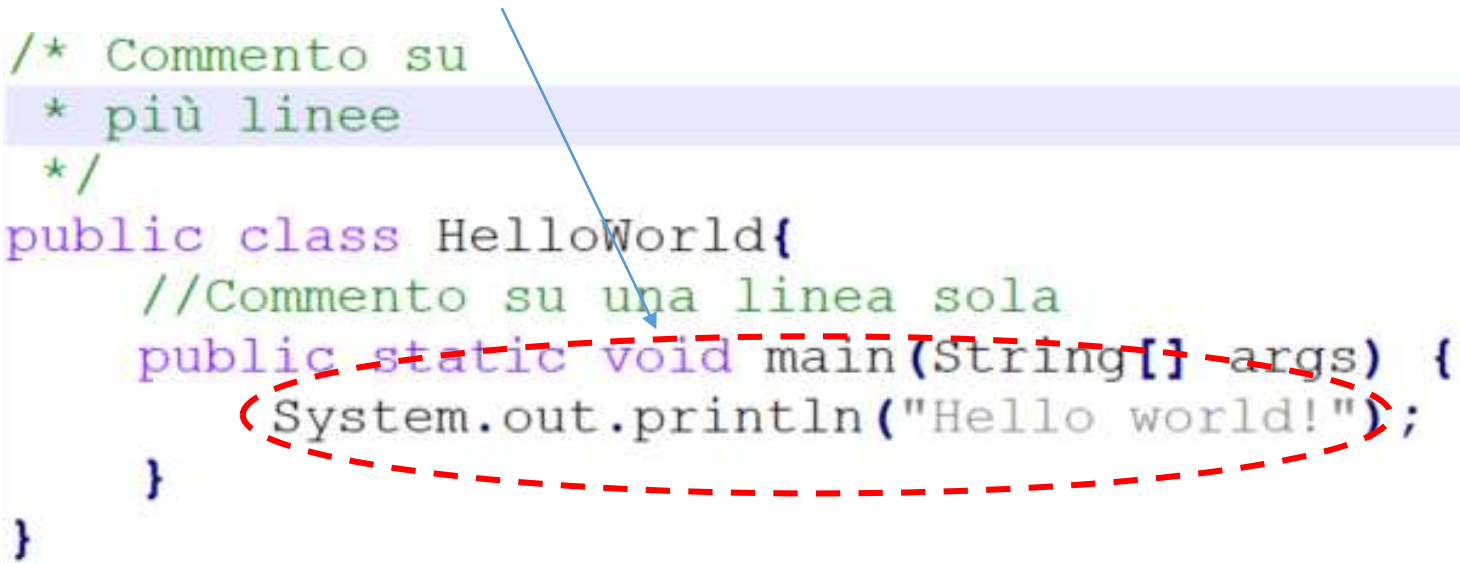
```
/* Commento su
 * più linee
 */
public class HelloWorld{
    //Commento su una linea sola
    public static void main(String[] args){
        System.out.println("Hello world!");
    }
}
```



# *HelloWorld* in Java

Il metodo *println()* del *package* System stampa a schermo una stringa di testo inclusa fra doppi apici

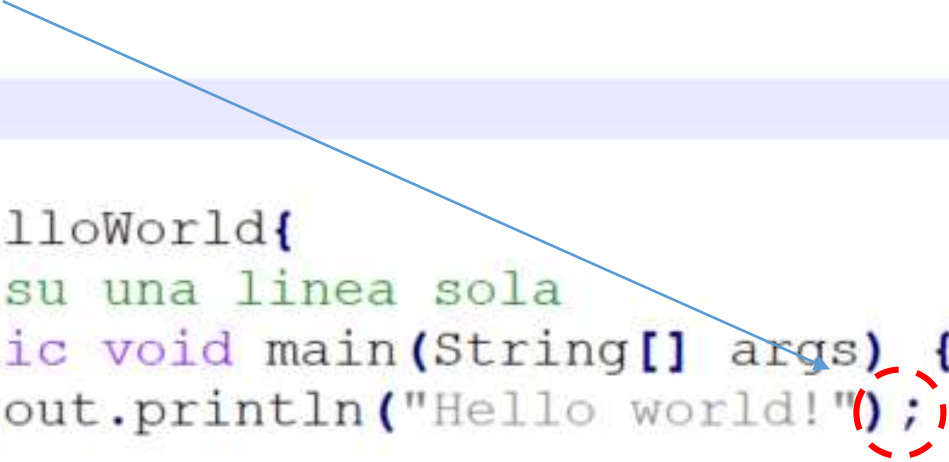
```
/* Commento su
 * più linee
 */
public class HelloWorld{
    //Commento su una linea sola
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```



# *HelloWorld* in Java

Le istruzioni Java sono sempre terminate da «;» !

```
/* Commento su
 * più linee
 */
public class HelloWorld{
    //Commento su una linea sola
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```





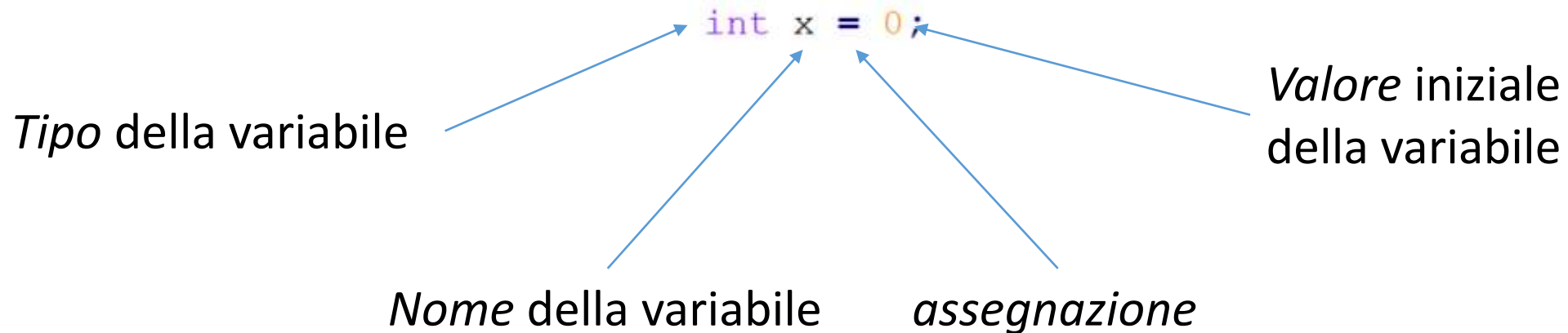
# Le Variabili - 1

- Scriviamo un programma che crei una *variabile* x all'interno del metodo main, assegni a questa un valore a piacere e stampi a schermo il contenuto della variabile

```
public class Variabile {  
    public static void main(String[] args) {  
        int x = 0;  
        System.out.println(x);  
    }  
}
```

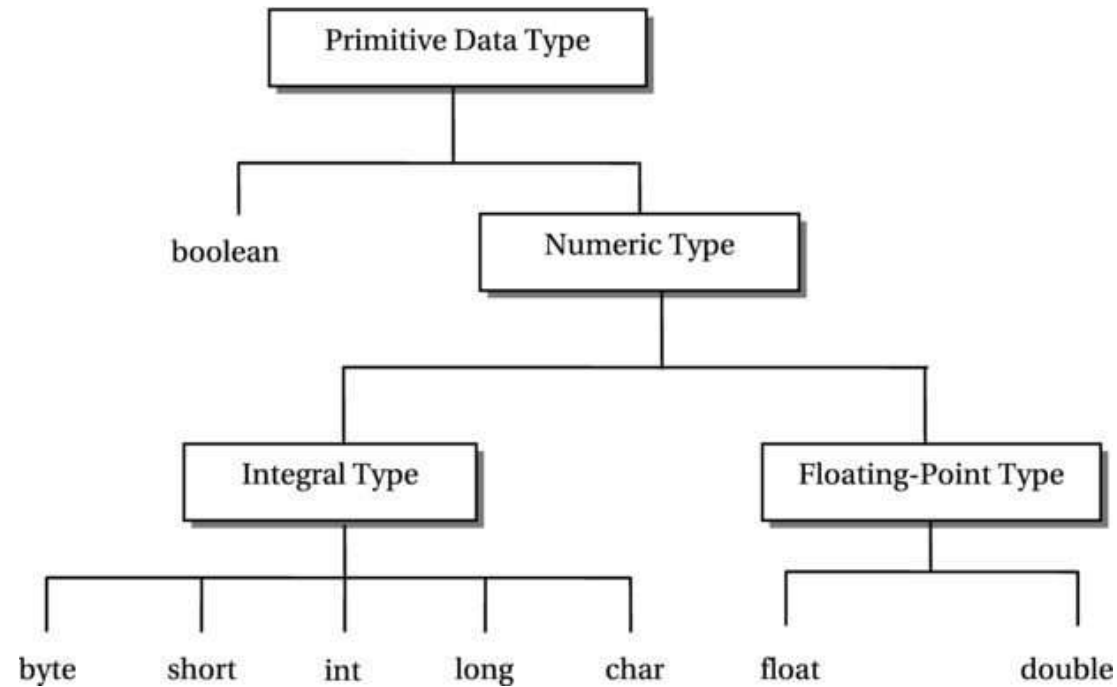
# Definizione di Variabili

- Analogia con le variabili algebriche (per ora)
  - ma solo numeri naturali (per ora)
- *Dichiarazione e definizione* usati equivalentemente
  - abuso di terminologia
- Sintassi per definizione di una variabile



# Tipi Dati Primitivi

- Java é un linguaggio *tipizzato*



# Tipi Dati Primitivi - Java

Table 5.1 · Some basic types

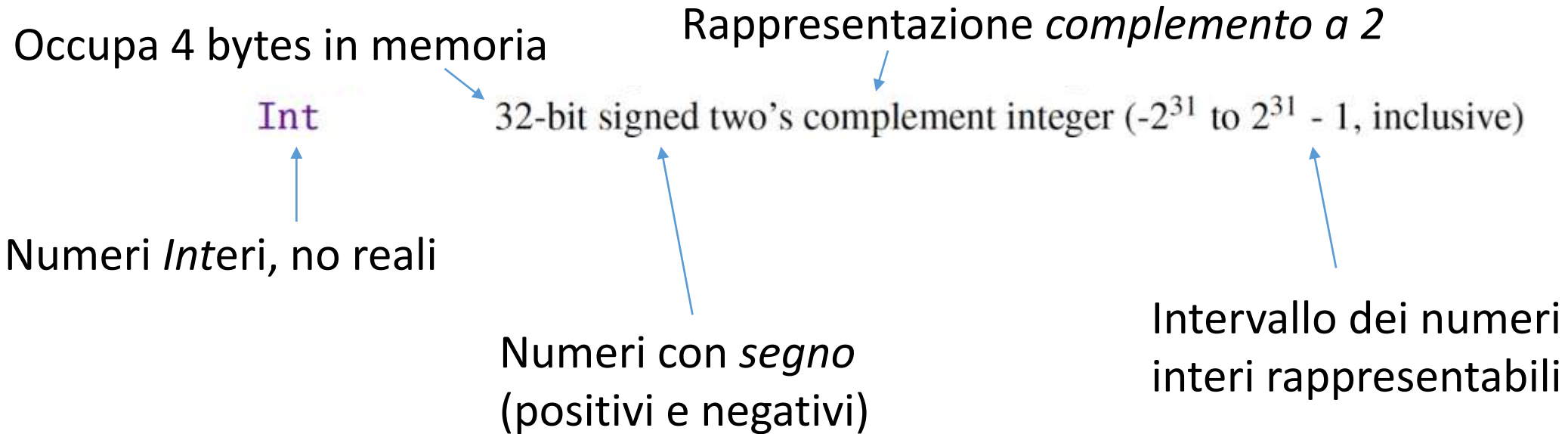
Value type	Range
Byte	8-bit signed two's complement integer ( $-2^7$ to $2^7 - 1$ , inclusive)
Short	16-bit signed two's complement integer ( $-2^{15}$ to $2^{15} - 1$ , inclusive)
Int	32-bit signed two's complement integer ( $-2^{31}$ to $2^{31} - 1$ , inclusive)
Long	64-bit signed two's complement integer ( $-2^{63}$ to $2^{63} - 1$ , inclusive)
Char	16-bit unsigned Unicode character (0 to $2^{16} - 1$ , inclusive)
String	a sequence of Chars
Float	32-bit IEEE 754 single-precision float
Double	64-bit IEEE 754 double-precision float
Boolean	true or false

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

# Tipi Dati Primitivi – Int

Table 5.1 · Some basic types

Value type	Range
------------	-------



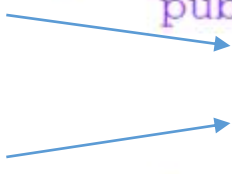
# Le Variabili - 2

- Scriviamo un programma che crei una *variabile* *x* all'interno del metodo main, assegni a questa un valore a piacere e stampi a schermo il contenuto della variabile

Dichiaro la variabile

Assegno un valore

```
public class Variabile {  
    public static void main(String[] args) {  
        int x;  
        x = 0;  
        System.out.println(x);  
    }  
}
```



# Le Variabili - 2

- Scriviamo un programma che crei due *variabili* x e y, assegni loro valori a piacere e stampi a schermo il risultato dell'addizione

Dichiaro le variabili

Uso una variabile temporanea z per l'addizione


```
public class Addizione {  
    public static void main(String[] args) {  
        int x = 2;  
        int y = -6;  
        int z = x + y;  
        System.out.println(z);  
    }  
}
```

L'espressione viene *valutata*

# I salti condizionati - 1

- Scriviamo un programma che crei due variabili *a* e *b*, assegni loro valori a piacere e stampi a schermo un messaggio a piacere se  $a > b$

Costrutto *if()*



```
public class Maggiore {  
    public static void main(String[] args) {  
        int a = 5;  
        int b = -3;  
        if ( a > b ) {  
            System.out.println("a é maggiore di b");  
        }  
    }  
}
```



# Tipi Dati Primitivi – Boolean

Table 5.1 · Some basic types

Value type	Range
Boolean	true or false

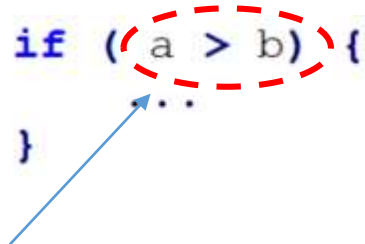
Occupi almeno 1 bit in memoria (tipicamente, 1 byte)

*true* equivale a 1  
*false* equivale a 0

# Il costrutto if

- Permette di eseguire un blocco di codice solo sotto certe condizioni
- L'argomento di *if()* é detto *condizione di salto*
  - La condizione di salto é un'espressione booleana (*vera* oppure *falsa*)

```
if (a > b) {  
    ..  
}
```



*Condizione di salto*

# Il costrutto if

- Permette di eseguire un blocco di codice solo sotto certe condizioni
- L'argomento di *if()* é detto *condizione di salto*
  - La condizione di salto é un'espressione booleana (*vera* oppure *falsa*)
- Se la condizione di salto é verificata
  - Viene eseguito quanto nel blocco di codice del salto

Es: a=5, b=3


```
if ( a > b ) {  
    ...  
}
```

# Il costrutto if

- Permette di eseguire un blocco di codice solo sotto certe condizioni
- L'argomento di *if()* é detto *condizione di salto*
  - La condizione di salto é un'espressione booleana (*vera* oppure *falsa*)
- Se la condizione di salto é verificata
  - Viene eseguito quanto nel blocco di codice del salto
  - L'esecuzione continua dopo il blocco

Es: a=5, b=3

```
if ( a > b ) {  
    ...  
}
```




# Il costrutto if

- Permette di eseguire un blocco di codice solo sotto certe condizioni
- L'argomento di *if()* è detto *condizione di salto*
  - La condizione di salto è un'espressione booleana (*vera* oppure *falsa*)
- Se la condizione di salto è verificata
  - Viene eseguito quanto nel blocco di codice del salto
  - L'esecuzione continua dopo il blocco
- Altrimenti l'esecuzione continua

Es: a=-5, b=3

```
if ( a > b ) {  
    ...  
}  
...
```



# Il costrutto if

- Le tre costruzioni seguenti sono equivalenti
  - La condizione di salto é sempre booleana

```
if ( a > b) {  
    ...  
}  
...
```

```
boolean cond = a > b;  
if (cond) {  
    ...  
}  
...
```

```
boolean cond = (a > b);  
if (cond) {  
    ...  
}  
...
```

# Il costrutto if – *caveat semper*

- La condizione «*a* uguale a *b*» si esprime con «`==`»

```
if (a == b) {  
    ...  
}
```

- L'assegnazione di «*b* in *a* » é sempre verificata!

```
if (a = b) {  
    ...  
}
```



# I salti condizionati - 2

- Scriviamo un programma che crei due variabili  $a$  e  $b$ , assegni loro valori a piacere e stampi a schermo un messaggio a piacere se  $a > b$  e un messaggio diverso *altrimenti*

```
public class Maggiore {  
    public static void main(String[] args) {  
        int a = 5;  
        int b = -3;  
        if ( a > b) {  
            System.out.println("a é maggiore di b");  
        }  
        if ( a < b) {  
            System.out.println("a é minore di b");  
        }  
    }  
}
```



# I salti condizionati - 2

- Scriviamo un programma che crei due variabili  $a$  e  $b$ , assegni loro valori a piacere e stampi a schermo un messaggio a piacere se  $a > b$  e un messaggio diverso *altrimenti*

```
public class Maggiore {  
    public static void main(String[] args) {  
        int a = 5;  
        int b = -3;  
        if ( a > b) {  
            System.out.println("a é maggiore di b");  
        }  
        else {  
            System.out.println("a é minore di b");  
        }  
    }  
}
```

# Il costrutto if-else

- Se la condizione di salto é verificata
  - Viene eseguito quanto nel blocco if() { ... }
- Altrimenti (*in qualsiasi altro caso*)
  - Viene eseguito quanto nel blocco else() { ... }

```
if ( a > b ) {  
    ...  
}  
else {  
    ...  
}
```

Es: a=-5, b=3

# Il costrutto if-else

- Se la condizione di salto é verificata
  - Viene eseguito quanto nel blocco if() { ... }
- Altrimenti (*in qualsiasi altro caso*)
  - Viene eseguito quanto nel blocco else() { ... }


```
if ( a > b ) {  
    ...  
}  
else {  
    ...  
}
```

Es: a=-5, b=3

# Il costrutto if-else

- Se la condizione di salto é verificata
  - Viene eseguito quanto nel blocco `if() { ... }`
- Altrimenti (*in qualsiasi altro caso*)
  - Viene eseguito quanto nel blocco `else() { ... }`
- L'esecuzione continua in ogni caso

```
if ( a > b ) {  
    ...  
}  
else {  
    ...  
}  
...
```



# Le iterazioni/cicli/loop

- I programmi che abbiamo visto ora sono strettamente *sequenziali*
  - Dopo la riga n, sarà sempre eseguita la riga m, con  $m > n$

```
1 public class Maggiore {  
2     public static void main(String[] args) {  
3         int a = 5;  
4         int b = -3;  
5         if ( a > b) {  
6             System.out.println("a é maggiore di b");  
7         }  
8         else {  
9             System.out.println("a é minore di b");  
10        }  
11    }  
12 }
```

# Le iterazioni/cicli/loop

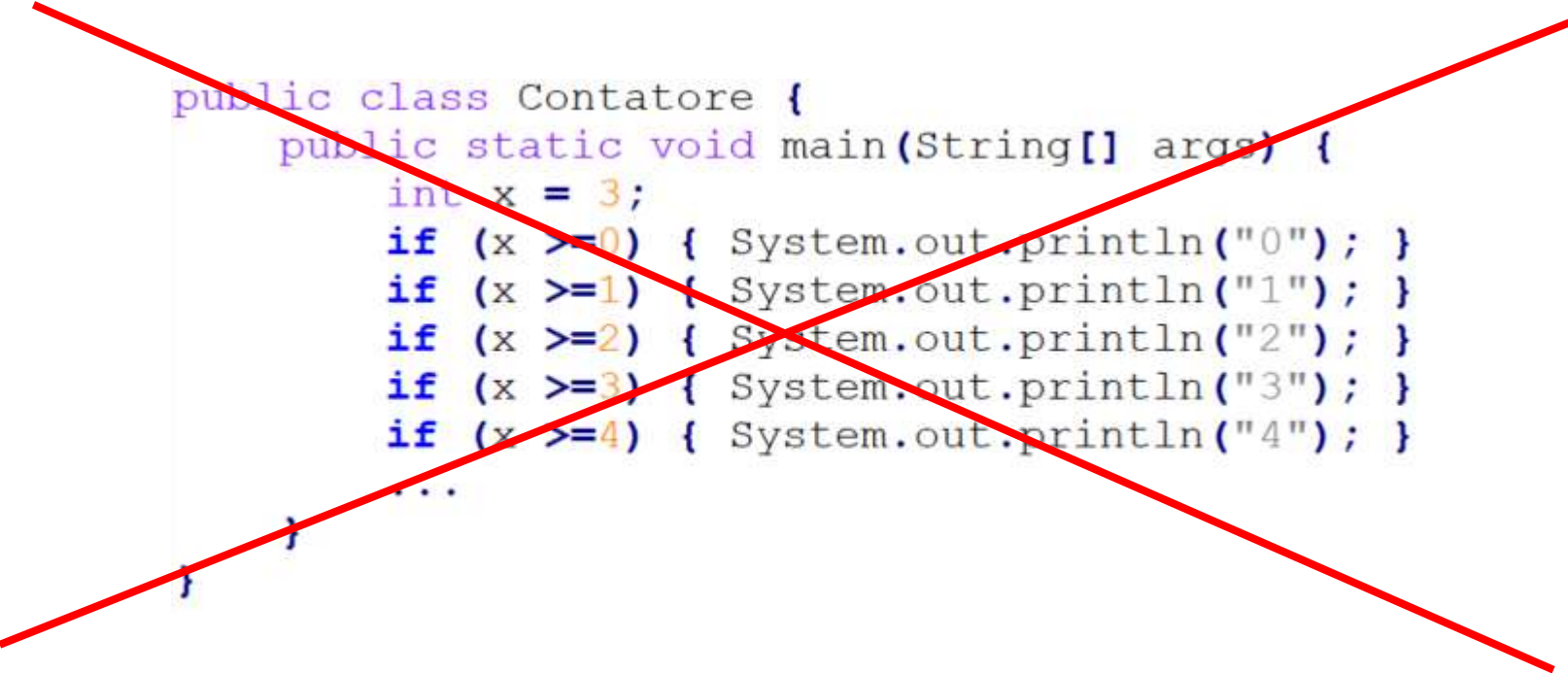
- Scrivere un programma che stampi tutti gli interi da 0 a x compresi

```
public class Contatore {  
    public static void main(String[] args) {  
        int x = 3;  
        if (x >= 0) { System.out.println("0"); }  
        if (x >= 1) { System.out.println("1"); }  
        if (x >= 2) { System.out.println("2"); }  
        if (x >= 3) { System.out.println("3"); }  
        if (x >= 4) { System.out.println("4"); }  
        ...  
    }  
}
```

# Le iterazioni/cicli/loop

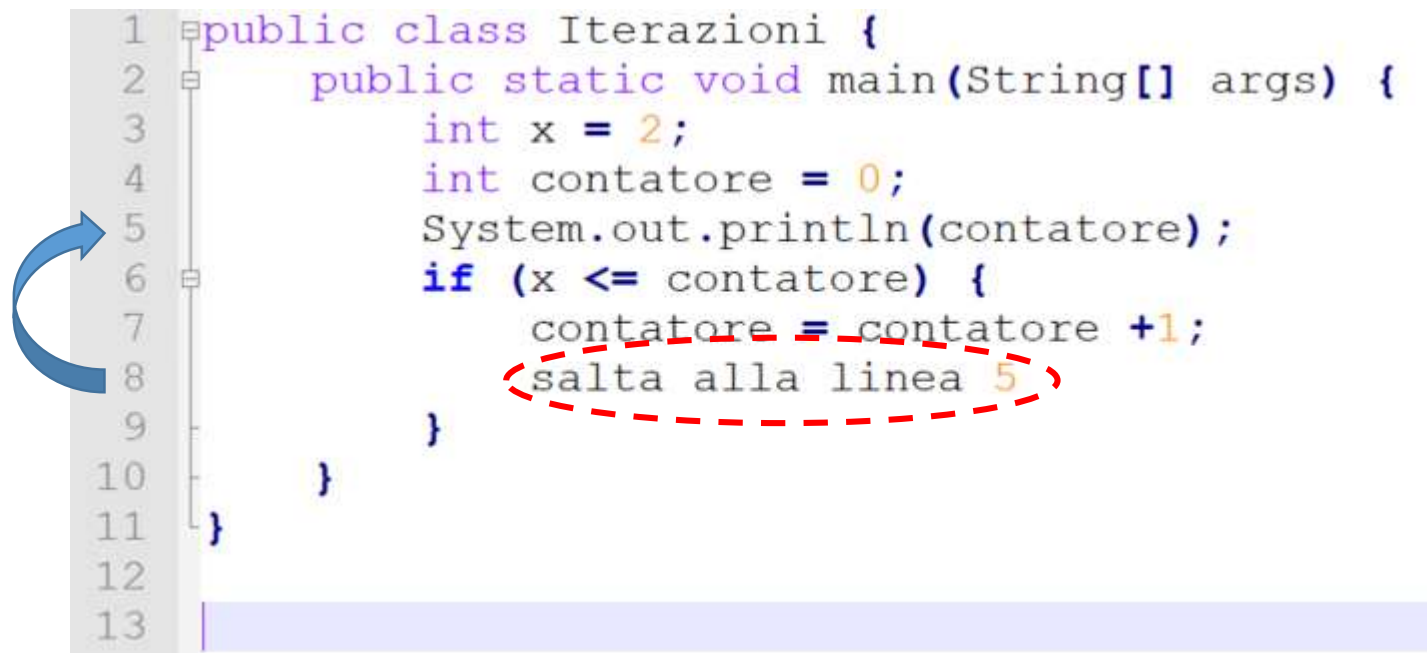
- Scrivere un programma che stampi tutti gli interi da 0 a x compresi
  - Bocciatura garantita all'esame

```
public class Contatore {  
    public static void main(String[] args) {  
        int x = 3;  
        if (x >= 0) { System.out.println("0"); }  
        if (x >= 1) { System.out.println("1"); }  
        if (x >= 2) { System.out.println("2"); }  
        if (x >= 3) { System.out.println("3"); }  
        if (x >= 4) { System.out.println("4"); }  
        ...  
    }  
}
```



# Le iterazioni/cicli/loop

- Ipotesi di eseguire un salto *all'indietro* alla linea di codice 5



The diagram shows a code editor with line numbers 1 through 13 on the left. The code is as follows:

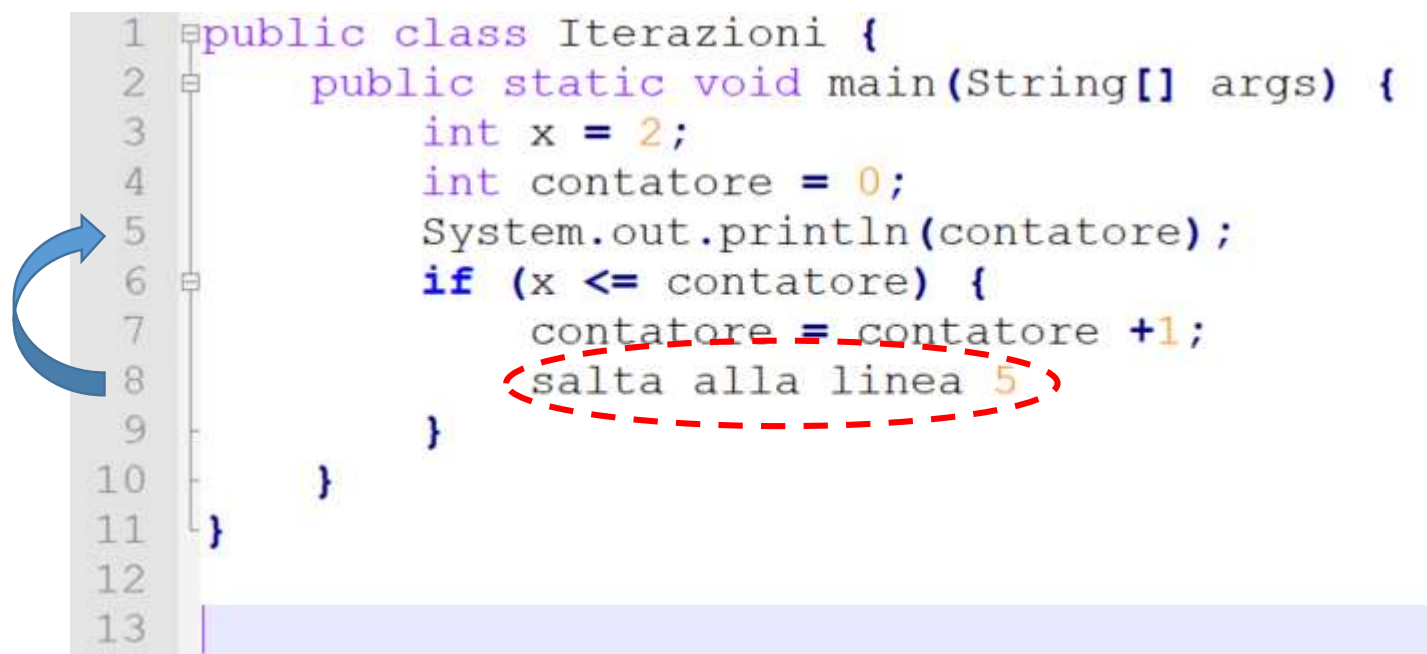
```
1 public class Iterazioni {  
2     public static void main(String[] args) {  
3         int x = 2;  
4         int contatore = 0;  
5         System.out.println(contatore);  
6         if (x <= contatore) {  
7             contatore = contatore + 1;  
8             salta alla linea 5  
9         }  
10    }  
11 }  
12  
13
```

A blue curved arrow points from line 8 back to line 5, indicating a loop. A red dashed oval encircles the text "salta alla linea 5" on line 8.



# Le iterazioni/cicli/loop

- Ipotesi di eseguire un salto *all'indietro* alla linea di codice 5
  - Poco flessibile: linea di salto *hardwired*



The diagram illustrates a code snippet with a jump instruction. A blue curved arrow on the left points from line 8 back to line 5, indicating a loop. The code is as follows:

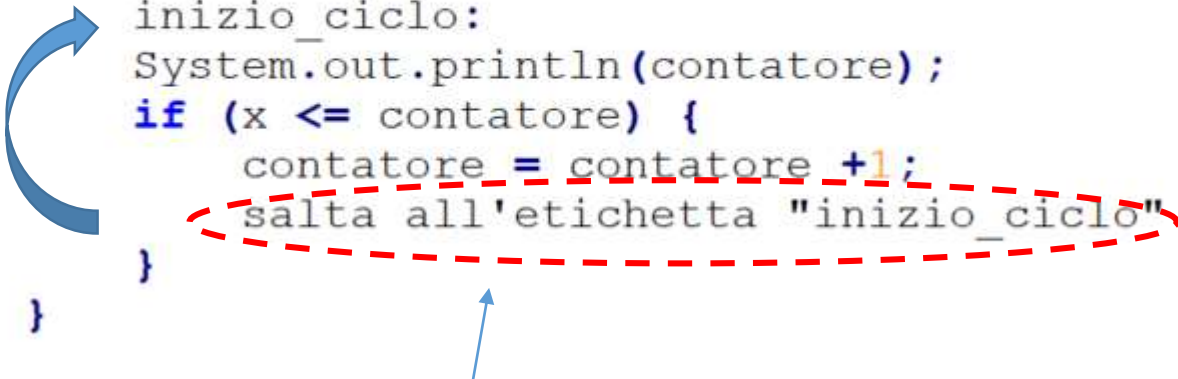
```
1 public class Iterazioni {
2     public static void main(String[] args) {
3         int x = 2;
4         int contatore = 0;
5         System.out.println(contatore);
6         if (x <= contatore) {
7             contatore = contatore + 1;
8             salta alla linea 5
9         }
10    }
11 }
12
13
```

The text "salta alla linea 5" is circled in red, and the number 5 is highlighted in orange.

# Le iterazioni/cicli/loop

- Ipotizzo di eseguire un salto *all'indietro* alla *etichetta inizio\_ciclo*
  - Dipende solo dal posizionamento dell'*etichetta(label)* «inizio\_ciclo»

```
public class Iterazioni {  
    public static void main(String[] args) {  
        int x = 2;  
        int contatore = 0;  
        inizio_ciclo:  
        System.out.println(contatore);  
        if (x <= contatore) {  
            contatore = contatore + 1;  
            salta all'etichetta "inizio_ciclo"  
        }  
    }  
}
```



Salto all'  
indietro esplicito

# Le iterazioni/cicli/loop

- I salti incondizionati all'indietro permettono di interrompere l'ordine sequenziale delle istruzioni
  - Un salto all'indietro é noto come *iterazione* o *ciclo*
- In combinazione con *if()* eseguiamo salti condizionati all'indietro
- Il linguaggio Java non dispone di istruzioni come *salta(label)*
  - Peraltro il processore ne dispone
- Java dispone di costrutti di più alto livello per i salti condizionali
  - *for()*, *while()*

# Il costrutto while

- All'inizio di ogni ciclo si controlla la condizione di *permanenza*
- Se la condizione di *permanenza* é verificata, si avvia un ciclo
  - Viene eseguito quanto nel blocco `while() { ... }`
- Alla fine del ciclo la condizione viene verificata nuovamente
- Il ciclo si interrompe quando la condizione non é più verificata
  - L'interruzione può dipendere dal codice scritto da noi o da un evento esterno

```
while (<condizione>) {  
    ...  
}
```

# Le iterazioni/cicli/loop

- Eseguo un salto all'indietro con il costrutto while
  - Utilizzo variabile ausiliaria *contatore*

```
public class Iterazioni {  
    public static void main(String[] args) {  
        int x = 2;  
        int contatore = 0;  
        while (x >= contatore) {  
            System.out.println(contatore);  
            contatore = contatore + 1;  
        }  
    }  
}
```

Inizializzo variabile  
ausiliaria *contatore*

Check condizione di  
permanenza nel ciclo

Aggiornamento  
variabile ausiliaria

Salto all'  
indietro implicito

# Le iterazioni/cicli/loop

- Eseguo un salto all'indietro con il costrutto while
  - Utilizzo variabile ausiliaria *contatore*

```
public class Iterazioni {  
    public static void main(String[] args) {  
        int x = 2;  
        int contatore = 0;  
        while (x >= contatore) {  
            System.out.println(contatore);  
            contatore = contatore + 1;  
        }  
    }  
}
```

Inizializzo variabile  
ausiliaria *contatore*

Check condizione di  
permanenza nel ciclo

Aggiornamento  
variabile ausiliaria

Salto all'  
indietro implicito

# Le iterazioni/cicli/loop

- Eseguo un salto all'indietro con il costrutto while
  - Utilizzo variabile ausiliaria *contatore*

```
public class Iterazioni {  
    public static void main(String[] args) {  
        int x = 2;  
        int contatore = 0;  
        while (true) {  
            System.out.println(contatore);  
            contatore = contatore + 1;  
            if (x >= contatore)  
                break;  
        }  
    }  
}
```

Check condizione di permanenza nel ciclo sempre verificata

Aggiornamento variabile ausiliaria

Uscita dal ciclo manuale

# ESERCIZIO – Beyond Emersione Massimo

- Ispirandovi all'esercizio *emersione massimo* visto in aula e utilizzando il costrutto *while*, scrivere un programma che date quattro variabili *a,b,c,d* esegua le opportune permutazioni a coppie di variabili adiacenti per cui infine  $a \leq b \leq c \leq d$
- Suggerimento: iterare su *emersione massimo*
  - Quante volte iterare ?
  - Quale variabile ausiliaria ?
  - Come uscire dal ciclo ?