

# Introduzione a Java

Modulo 226A

Implementare in base alle classi (senza ereditarietà)



# Storia

Java è un linguaggio di programmazione nato all'inizio degli anni novanta da un gruppo di lavoro della Sun Microsystems.

Concepito per scrivere programmi capaci di essere eseguiti su diversi tipi di processori e che non richiedesse compilatori o interpreti troppo sofisticati.

L'idea fu di introdurre un unico, semplice linguaggio intermedio (chiamato byte-code) per permettere la portabilità dei programmi.

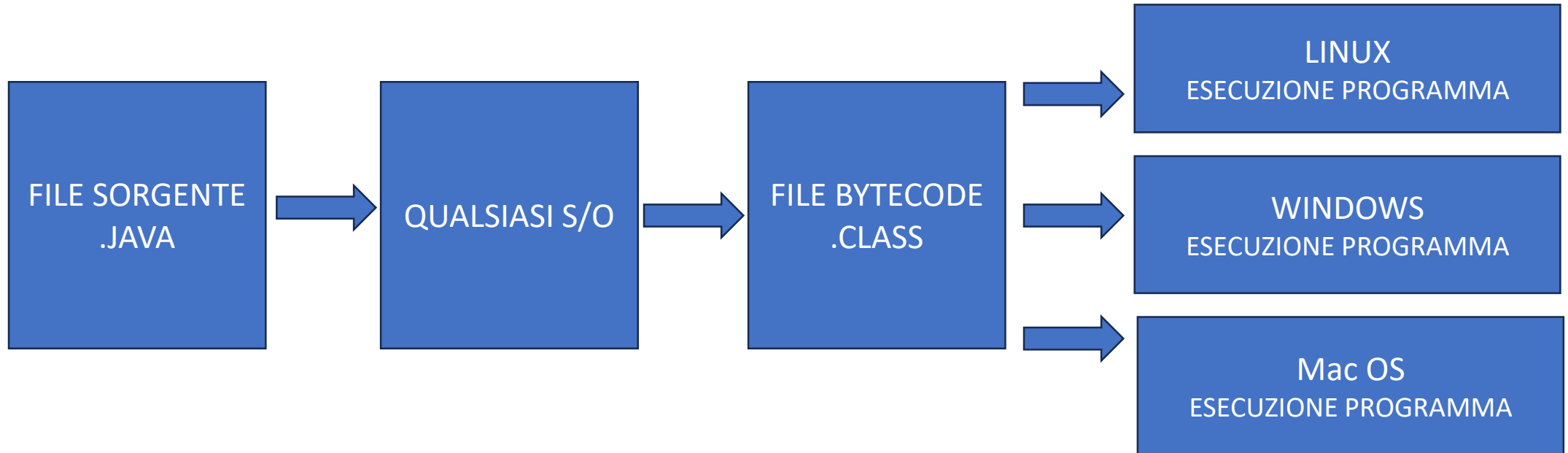
# Byte-code Java

Il linguaggio Java si basa su un approccio che combina compilazione (in byte-code) e interpretazione (del byte-code).

Il byte-code può essere visto come l'assembly di una macchina virtuale, un calcolatore virtuale che ha caratteristiche simili (semplificate) a quelle delle architetture hardware più comuni è un linguaggio di basso livello (come l'assembly) non legato ad una particolare architettura hardware.

L'interprete del byte-code Java è detto **Java Virtual Machine (JVM)**

# Schema Architettura



# Ambienti di sviluppo

Per poter sviluppare un programma Java sono necessari:

- Un Editor o un IDE di sviluppo (Intelj Idea, Code, Eclipse, NetBeans,...)
- Il Java Development Kit (JDK).
- [https://www.w3schools.com/java/java\\_getstarted.asp](https://www.w3schools.com/java/java_getstarted.asp)

# JDK

Java Development Kit (**JDK**) è un ambiente di sviluppo software utilizzato per lo sviluppo di applicazioni e applet Java.

Include Java Runtime Environment (**JRE**), un interprete/caricatore (Java), un compilatore (javac), un archiviatore (jar), un generatore di documentazione (Javadoc) e altri strumenti necessari nello sviluppo Java.

# Anatomia di un programma Java

```
/**
 * La classe HelloWorld è un programma che
 * stampa la stringa "HelloWorld!!" sullo standard output.
 */
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("HelloWorld!!");
    }
}
```

Il programma HelloWorld si compone di tre parti principali:

- **I commenti**
- **La definizione e il corpo della classe HelloWorld**
- **Il prototipo e il corpo del metodo main**

# Compilazione del sorgente

Il comando ***javac*** legge il contenuto del programma, scritto in un file di testo con estensione **.java**

Se non ci sono errori, lo trasforma in un bytecode e lo salva in un file con estensione **.class**

```
C:\HelloWorld>javac HelloWorld.java
```

```
12:41    <DIR>          .
12:41          426 HelloWorld.class
12:40          147 HelloWorld.java
    2 File           573 byte
    1 Directory 239'250'903'040 byte disponibili
```



# Esecuzione del programma

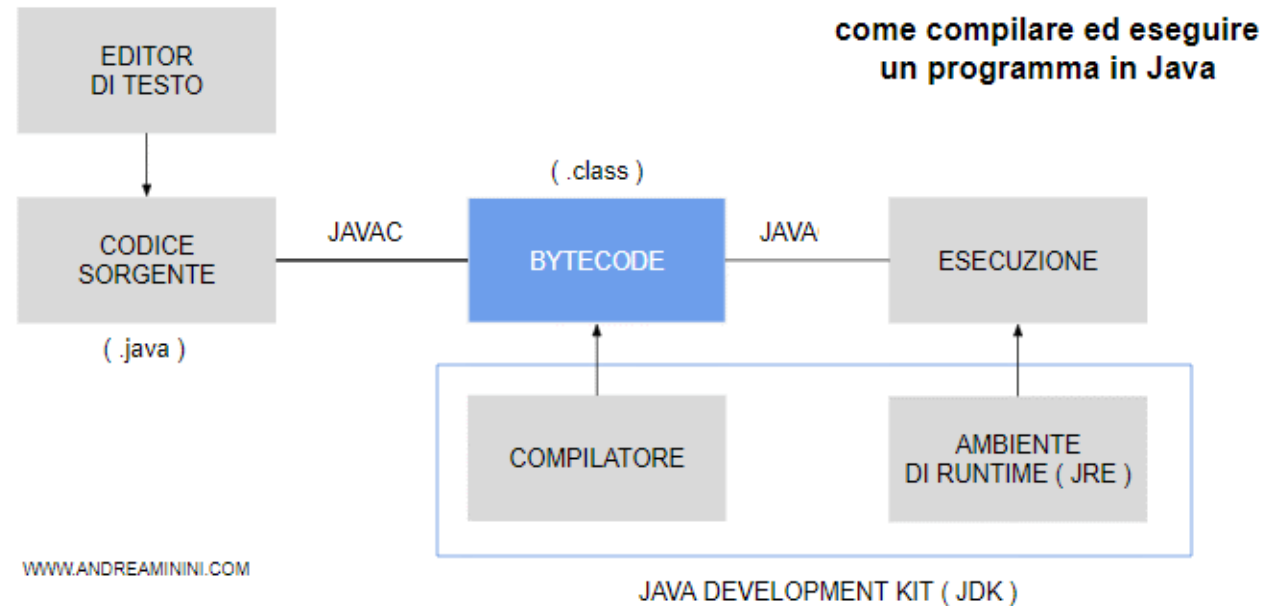
Per eseguire il programma in java bisogna richiamare il bytecode con l'interprete tramite il comando **java**.

```
C:\HelloWorld>java HelloWorld  
Hello world!
```

Il comando java legge il bytecode del programma e lo esegue.

# JRE

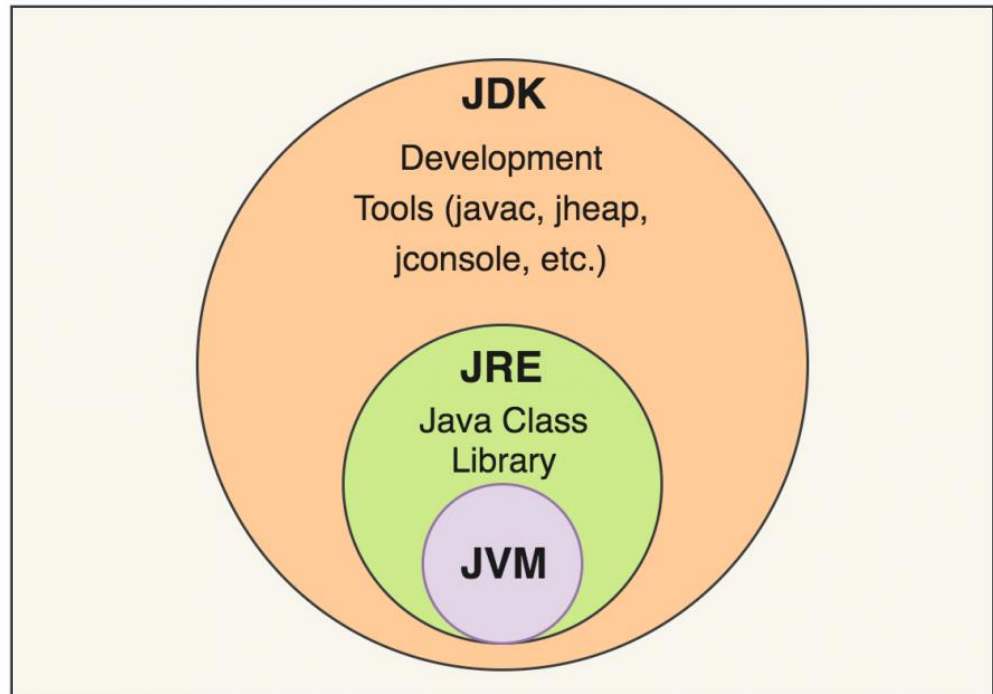
**JRE significa Java Runtime Environment.**



# Differenza tra JVM e JRE

I termini JVM e JRE vengano usati in modo intercambiabile, ma in realtà hanno delle differenze.

- JVM è una macchina virtuale che viene eseguita sul sistema operativo, mentre JRE è l'ambiente di esecuzione del runtime.
- JVM fa parte del JRE.



# Classe

```
public class HelloWorld {  
    ...  
}
```

- Un programma Java è costituito da un insieme di classi (almeno una).
- **public** significa che questa classe è pubblica: può essere utilizzata da qualunque altra classe del programma
- Il contenuto della classe è racchiuso tra parentesi graffe

# Main - 1

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

- Questo metodo è chiamato **main** (principale) ed è un metodo speciale. Viene eseguito all'inizio del programma.
- **public** (modificatore) significa che l'accesso al metodo è permesso anche da altre classi.
- Il corpo del metodo è racchiuso tra le parentesi graffe

# Main - 2

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

- Si può dare all'argomento qualsiasi nome, ma generalmente i programmatori scelgono il nome `args`
- Java è un linguaggio case sensitive
- Il metodo main accetta un singolo argomento: un array di elementi di tipo string.
- Args permette di passare informazioni dal s/o al programma.

# Esercizio in classe

- HelloWorld tramite VsCode
- HelloMondo tramite blocco note e compilazione da CMD.

# Argomenti da linea di comando

Il metodo **main** può ricevere informazioni dal sistema operativo attraverso gli argomenti da linea di comando.

Gli argomenti sono **stringhe** che vengono specificate dall'utente nella fase di esecuzione di un programma.



# Argomenti in Java: il vettore args

Gli argomenti permettono di parametrizzare il funzionamento di programmi per renderli più flessibili ed adattabili alle esigenze degli utenti.

È possibile utilizzare gli argomenti per modificare il comportamento del programma.

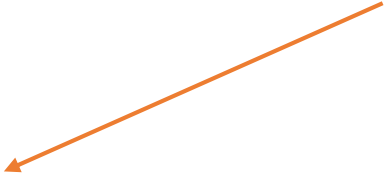
In Java gli argomenti vengono passati al metodo main del programma tramite il vettore di stringhe args.

# Esempio HelloArgs - 1

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

Nell'esempio viene stampata la scritta **Hello** concatenando il primo elemento del vettore args

Primo elemento di args (indice 0)



```
C:\HelloWorld>Java HelloArgs Francesco  
Hello Francesco
```

# Esempio HelloArgs - 2

```
public class HelloArgs {  
    public static void main(String[] args) {  
        System.out.println("Hello " + args[0]);  
        System.out.println("Hello " + args[1]);  
    }  
}
```

```
C:\HelloWorld>Java HelloArgs Simona Roberto  
Hello Simona  
Hello Roberto
```

Prima di avviare il programma bisogna ricompilare (visto che ci sono state modifiche al sorgente).

# Dimensione di args

In Java la dimensione di un vettore/array può essere ottenuta utilizzando la proprietà `length`; se l'espressione `args.length` ha valore uguale a 0 significa che l'utente non ha passato alcun argomento e quindi il vettore `args` è vuoto:

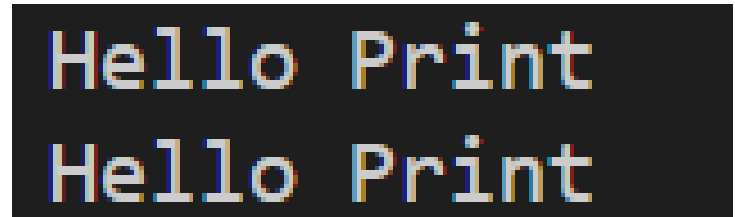
```
public class HelloArgs {  
    public static void main(String[] args) {  
        System.out.println(args.length);  
    }  
}
```

# System.out

System.out è un `PrintStream` che rappresenta lo standard output (per default il terminale da cui è stata lanciata l'applicazione).

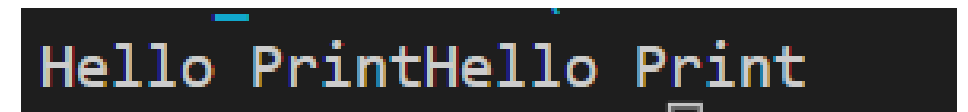
Quando un programma stampa dei caratteri tramite `System.out` essi vengono inviati al terminale.

```
public class HelloArgs {  
    public static void main(String[] args) {  
        System.out.println("Hello Print");  
        System.out.println("Hello Print");  
    }  
}
```



```
Hello Print  
Hello Print
```

```
public class HelloArgs {  
    public static void main(String[] args) {  
        System.out.print("Hello Print");  
        System.out.print("Hello Print");  
    }  
}
```



```
Hello PrintHello Print
```

# Commenti

I commenti sono ignorati dal compilatore ma sono utilizzati dai programmatori per lasciare delle indicazioni nel codice sorgente e per documentare i programmi.

## Commenti su una o più righe

```
/* mio commento. */  
  
/*  
    il mio commento  
    su tre  
    righe  
*/
```

## Commenti su una sola riga

```
System.out.println("HelloWorld!!"); // un altro commento
```

# Esercizi in classe

- Esercizio 1.1
- Esercizio 1.2

The background features two large, decorative, curved lines. One line, in shades of blue and green, curves from the top right towards the center. Another line, in shades of green and blue, curves from the bottom left towards the center. Both lines have a soft, multi-layered appearance.

# Variabili e Tipo di dati



# Dichiarazione

Java appartiene alla categoria dei linguaggi detti **strongly-typed** (fortemente tipizzati).

Una variabile, per essere utilizzata, deve essere prima dichiarata.

Nella dichiarazione della variabile deve essere indicato il tipo e il nome della variabile.

`<tipo> <nome>`

# Tipi di dati fondamentali

Durante la dichiarazione della variabile bisogna obbligatoriamente scegliere il tipo di dato.

Tipo	Descrizione	Bit	Da	A
byte	Intero byte	8	-128	127
short	Intero corto	16	-32'768	32'767
int	Intero normale	32	-2'147'483'648	2'147'483'647
long	Intero lungo	64	-9'223'372'036'854'775'808	9'223'372'036'854'775'807
float	Numero a virgola mobile a precisione singola	32	$\pm 1.4e - 45$	$\pm 3.4028235e38$
double	Numero a virgola mobile a precisione doppia	64	$\pm 4.9e - 324$	$\pm 1.7976931348623157e308$
boolean	Booleano	1 <sup>1</sup>	true	false
char	Carattere Unicode	16	0000	ffff

I tipi di dati fondamentali sono scritti tutti con la lettera minuscola.  
Ad esempio *int* è diverso da **Int**.

```
public class HelloVar {  
    public static void main(String[] args) {  
  
        int decVal = 26; // Il numero 26, in decimale  
        int octVal = 032; // Il numero 26, in ottale  
        int hexVal = 0x1a; // Il numero 26, in esadecimale  
        int binVal = 0b11010; // Il numero 26, in binario  
  
        byte b = 100;  
  
        short s = 10000;  
  
        double d1 = 123.4; //Se un valore viene specificato senza f o d, viene considerato default double.  
        double d2 = 1.234e2; // stesso valore, ma in notazione scientifica  
        double d3 = 1.234e2d; //La lettera d o D indica invece un numero a virgola mobile a 64 bit.  
        float f1 = 123.4f; //La lettera f o F viene utilizzata per indicare numeri a virgola mobile 32 bit  
  
        char c1 = 'C';  
        char c2 = 'i';  
        char c3 = '\u0108';  
  
        boolean result = true;  
  
    }  
}
```

# Le stringhe in Java

In Java le stringhe sono **oggetti** appartenenti alla classe String

```
String s1 = "Michele";
```

La classe String fornisce diversi metodi, di seguito sono rappresentati i più comuni:

- `s1.length()`: restituisce la lunghezza della stringa `s1`
- `s1.charAt(index)`: restituisce un carattere alla posizione prefissata
- `s1.indexOf('c')` ritorna l'indice della prima occorrenza di `c` in `s1` (-1 se non c'è)
- `s1.equals(s2)`: dice se `s1` ed `s2` hanno lo stesso contenuto
- `s1.substring(10,18)`: restituisce la sottostringa che va da 10 a 17 (18-1)
- `s1.replace('E','X')`: restituisce una stringa con tutte le 'E' sostituite con 'X'

```
int a = 5;  
long b = a;  
System.out.println(b);
```

```
long c = 15;  
int d = c;  
System.out.println(d);
```

COSA SUCCEDE?

# Conversioni di tipo

La conversione da un tipo ad un altro può avvenire in due modi; tramite una conversione **implicita** oppure tramite una conversione **esplicita**.

# Conversioni implicite

La conversione implicita avviene **automaticamente** quando il tipo di destinazione è più capiente del tipo di partenza.

Ad esempio la conversione di una variabile da int a long può avvenire in modo implicito:

```
public class HelloCast {  
    public static void main(String[] args) {  
        int i = 5;  
        long l = 9L;  
        l = i; // conversione implicita da int a long (promozione)  
    }  
}
```

# Conversione esplicita numerica

Quando invece il tipo di destinazione è meno capiente del tipo di partenza vi è il rischio di una perdita di precisione e la compilazione del codice produce un errore.

```
public class HelloCast {  
    public static void main(String[] args) {  
        int a = 5;  
        long b = 9L;  
        a = b; // Type mismatch: cannot convert from long to int  
    }  
}
```

In questi casi è necessario applicare una conversione esplicita (o cast esplicito):

```
public class HelloCast {  
    public static void main(String[] args) {  
        int a = 5;  
        long b = 9L;  
        a = (int)b; // casting da long a int  
    }  
}
```



# Esercizio in classe

- Che valore assume la variabile **b** nei due esempi?
- Indica il motivo di tale valore.

```
short a = 328;  
byte b = (byte) a;
```

```
float a = 5.155f;  
int b = (int) a;
```

```
Double a = 5.8676699f;  
Float b = (float) a;
```

# Variazioni e perdita di precisione

Se si effettua un cast esplicito da **intero** ad un altro tipo di **intero con capacità** inferiore, i bit eccedenti di ordine superiore vengono troncati

```
short a = 328;  
byte b = (byte) a; //72
```

Se si effettua una conversione da **intero** ad un tipo a virgola mobile, a dipendenza del valore iniziale, può avvenire una perdita di precisione.

```
float a = 5.8676699f;  
int b = (int) a; //5.86767
```

# Espressioni

Se un'espressione contiene diversi tipi, gli operandi meno capienti o con precisione inferiore vengono promossi implicitamente al tipo più capiente o con maggiore precisione.

```
int a = 1;
int b = 2;
float c = 3.0f;
float d = 4.0f;
c = a * d; // a viene promosso a float
a = c * b; // errore di compilazione. Il risultato è di tipo float
a = (int)c * b; //conversione esplicita di c
a = 1;
b = 2;
c = a/b; // c ora vale 0.0
d = (float)a / b; //d ora vale 0.5
```

# Esercizi in classe

- Esercizio 1.3

# Operatori

# Operatori Binari

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

# Operatori Unari

Operator	Name
+	Positiv
-	Negativ
++	Increment
--	Decrement

# Operatori Relazionali

==	Equal to
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal



# Operatori Logici

Operator	Name
&&	Logical and
	Logical or
!	Logical not
^	XOR

# Bitwise operators

Il linguaggio Java mette a disposizione un insieme di operatori specializzati nella manipolazione dei bit di variabili di tipo intero, detti operatori bit a bit (o bitwise).

**L'uso di questi operatori è solitamente relegato a contesti in cui l'ottimizzazione nella gestione delle risorse quali memoria e cicli di processore ha netta prevalenza sulla leggibilità e manutenibilità del codice.**

Operatore	Simbolo
AND	&
OR	
XOR	^
Complemento a uno	~
Shift a destra	>>
Shift a destra unsigned	>>>
Shift a sinistra	<<

# Precedenza valutazione

Maggiore



Minore

Operator Precedence	
Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	
logical AND	&&
logical OR	
ternary	? :
assignment	= += -= *= /= %= &= ^=  = <<= >>= >>>=

# Esercizio in classe

Esercizio 1.3

Esercizio 1.4

Esercizio 1.5

# Classi Wrapper

Le classi Wrapper sono utili a trattare un tipo **primitivo** come un **oggetto**.

Una classe wrapper (involucro) incapsula una variabile di un tipo primitivo

In qualche modo “trasforma” un **tipo primitivo** in un **oggetto** equivalente

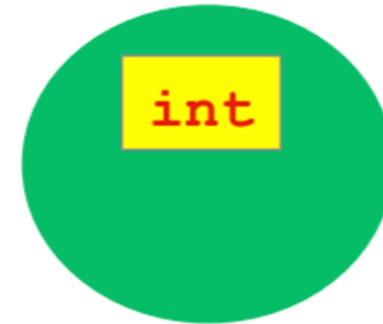
- la classe **Boolean** incapsula un boolean
- la classe **Double** incapsula un double
- la classe **Integer** incapsula un int
- la classe wrapper ha nome (quasi) identico al tipo primitivo che incapsula, ma con l’iniziale maiuscola



# Classi Wrapper

Tipo primitivo	Classe -wrapper“ corrispondente
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

*oggetto Integer*



# Classi Wrapper

Le classi wrapper possono essere utili in diverse situazioni.  
Inizialmente le utilizzeremo per svolgere operazioni di conversione o di parsing.

```
String s = "328";  
int i = Integer.parseInt(s);  
System.out.println(i + 5); // converte la stringa "328" nel valore int 333
```

```
String s = "FF";  
int i = Integer.parseInt(s,16);  
System.out.println(i); // converte la stringa "FF" nel valore int 255
```

# Esercizio in classe

## Esercizio 1.6



# Math

La classe Math mette a disposizione svariati metodi che permettono di effettuare operazioni matematiche.

- `Math.max( x,y )`
- `Math.min( x,y )`
- `Math.sqrt(x)`
- `Math.abs(x)`
- `Math.random()`
- ...

[Math \(Java Platform SE 8 \) \(oracle.com\)](https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html)