

# Java SE

- Operatori
  - Unari, aritmetici, assegnamento
  - Overload per stringhe
  - Relazionali, logici
- Istruzioni condizionali
- Istruzioni di loop
- Progetto di riferimento
  - <https://github.com/egalli64/jse> (*modulo 2*)

# Operatori unari

++ incremento

-- decremento

prefisso: “naturale”

postfisso: ritorna il valore  
prima dell'operazione

+ mantiene il segno corrente

- cambia il segno corrente

```
int value = 1;

result = value;    // value 1, result 1

result = ++value;  // value 2, result 2
result = --value;  // value 1, result 1

result = value++;  // value 2, result 1
result = value--;  // value 1, result 2

result = +value;   // value 1, result 1
result = -value;   // value 1, result -1
```

# Operatori aritmetici e assegnamento

+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione (intera?)
%	Modulo

=	Assegnamento
+=	Aggiungi e assegna
-=	Sottrai e assegna
*=	...
/=	
%=	
...	

```
int a = 10;
int b = 3;
double c = 3.0;

a / b    // 3
a % b    // 1
a / 0    // ArithmeticException

a % c    // 1.0
c - 2.1  // 0.8999999999999999
c / 0    // Infinity
```

```
int alpha = 2;

alpha += 8;    // 10
alpha -= 3;    // 7
alpha *= 2;    // 14
alpha /= 2;    // 7
alpha %= 5;    // 2
```

# Concatenazione di stringhe

- Operatori utilizzabili su stringhe (overload)
  - +** Se almeno un operando è una stringa, genera una nuova stringa che è la concatenazione dei due operandi
  - +=** Se la variabile a sinistra è una stringa, l'espressione a destra viene valutata come stringa. Il risultato della concatenazione viene assegnato alla variabile a sinistra.
- Da **Java 11**, **repeat()** è una specie di moltiplicazione per stringhe

```
System.out.println("Resistance" + " is " + "useless");  
System.out.println("Solution: " + 42);  
System.out.println(true + " or " + false);  
  
System.out.println("Vogons".repeat(3));
```

# Operatori relazionali

<	Minore
<=	Minore o uguale
>	Maggiore
>=	Maggiore o uguale
==	Uguale
!=	Diverso

Reference vs primitivi!

```
int alpha = 12;  
int beta = 21;  
int gamma = 12;
```

```
alpha < beta      // true  
alpha < gamma    // false
```

```
alpha <= gamma   // true
```

```
alpha > beta      // false  
alpha > gamma    // false
```

```
alpha >= gamma   // true
```

```
alpha == beta    // false  
alpha == gamma  // true
```

```
alpha != beta    // true  
alpha != gamma  // false
```

```
String a = "solution";  
String b = "sol";  
String c = "ution";  
String d = b + c;
```

```
a == d // false
```

# Operatori logici (e bitwise)

“shortcut” preferiti

&&	AND
	OR
!	NOT
&	AND
	OR
^	XOR

~	NOT
>>	R SHIFT
>>>	(segno)
<<	L SHIFT

```
boolean alpha = true;  
boolean beta = false;
```

```
alpha && beta    // false  
alpha || beta    // true  
!alpha           // false  
alpha & beta     // false  
alpha | beta     // true  
alpha ^ beta     // true
```

```
int gamma = 0b101; // 5  
int delta = 0b110; // 6
```

```
gamma & delta    // 4 == 0100  
gamma | delta    // 7 == 0111  
gamma ^ delta    // 3 == 0011
```

```
int zeta = ~gamma; // -6  
zeta >> 1          // -3  
zeta >>> 1         // 2147483645  
zeta << 1          // -12
```

0b → rappresentazione binaria

# Condizioni

- Se la condizione è vera, si esegue il blocco associato.
- Altrimenti, se presente si esegue il blocco “else”.

```
if (condition) {  
    // doSomething  
}  
  
// nextStep
```

```
if (condition) {  
    // doSomething  
} else {  
    // doSomethingElse  
}  
  
// nextStep
```

```
if (condition) {  
    // doSomething  
} else if (otherCondition) {  
    // doSomethingElse  
} else {  
    // doSomethingDifferent  
}  
  
// nextStep
```

# switch

Scelta multipla su byte, short, char, int, String, **enum** ➡

```
int value = 42;
// ...

switch (value) {
case 1:
    // ...
    break;
case 2:
    // ...
    break;
default:
    // ...
    break;
}
```

```
String value = "1";
// ...

switch (value) {
case "1":
    // ...
    break;
case "2":
    // ...
    break;
default:
    // ...
    break;
}
```

```
public enum WeekendDay {
    SATURDAY, SUNDAY
}

WeekendDay day = WeekendDay.SATURDAY;
// ...

switch (day) {
case SATURDAY:
    // ...
    break;
case SUNDAY:
    // ...
    break;
}
```



# Costanti simboliche con enum

- **enum** è un tipo speciale di classe i cui elementi sono un insieme di costanti
  - Le istanze della classe sono definite con la definizione dell'enum
    - Non è possibile crearne altre in seguito (il costruttore è privato)
  - Ogni enum estende la classe Enum, che mette a disposizione alcuni metodi
    - Tra cui values() che ritorna tutti gli elementi dell'enum corrente come array
  - Nell'uso base, si fa riferimento solo al nomi delle costanti
    - Che comunque hanno un valore associato (default: 0, 1, ...)
  - Comodo per limitare la scelta in un range noto e limitato
  - Funziona bene in abbinamento allo switch

```
public enum WeekDay { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY }
```

# Loop

```
while (condition) {  
    // ...  
  
    if (something) {  
        condition = false;  
    }  
}
```

```
do {  
    // ...  
  
    if (something) {  
        condition = false;  
    }  
} while (condition);
```

```
for (int i = 0; i < 5; i++) {  
    // ...  
  
    if (i == 2) {  
        continue;  
    }  
  
    // ...  
}
```

```
String[] array = new String[5];  
// ...  
for (String item : array) {  
    System.out.println(item);  
}
```

```
for (;;) {  
    // ...  
  
    if (something) {  
        break;  
    }  
  
    // ...  
}
```

forever

for each