JAVA - Elementi del linguaggio

Metodi Avanzati di Programmazione

Laurea Triennale in Informatica

Università degli Studi di Bari Aldo Moro

Docente: Pierpaolo Basile

Sommario

- Variabili
- Tipi primitivi
- Array
- Operatori
- Espressioni, statement, blocchi
- Controllo del flusso

Le variabili

- Hanno un tipo, un nome e memorizzano un valore del tipo specificato
- Le variabili servono a descrivere lo stato di un oggetto
  - in JAVA i termini variabile e field (attributo) sono interscambiabili e possono generare confusione
- Es. lo stato della bicicletta può essere descritto dai seguenti attributi

```
int speed = 0;
int gear = 1;
```

Le variabili in JAVA

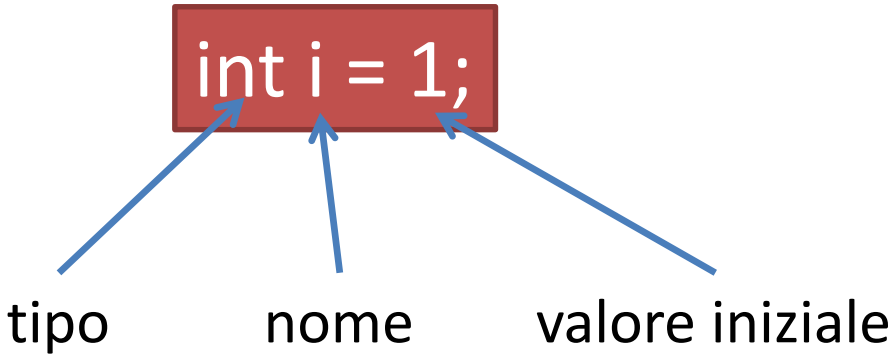
- **Variabili delle istanze (no-static):** ogni istanza di un oggetto conserva il suo stato attraverso i suoi fields (variabili)
  - *speed* è una variabile che dipende dall'istanza, ogni istanza avrà un valore diverso di *speed*
- **Variabili delle classi (static):** la variabile è la stessa per tutte le istanze della classe
- **Variabili locali:** utilizzate dai metodi di una classe, sono accessibili solo all'interno dei metodi
- **Parametri:** sono le variabili che passiamo quando chiamiamo un metodo
  - *String[] args* nel metodo *main* è un parametro

I nomi delle variabili

- Sono case-sensitive: differenza tra maiuscole e minuscole
- Il nome deve iniziare con una lettera, con $ e _
  - per convenzione dovrebbero iniziare sempre con una lettera
- Il nome può contenere anche numeri
  - utilizzare sempre nomi auto-esplicativi
- Convenzioni
  - le costanti e le variabili static vanno scritte in maiuscolo, altrimenti si utilizzano le minuscole
  - se il nome è composto da più parole si utilizza la maiuscola all'inizio di ogni nuova parola, per le costanti il carattere _
    - MAX_GEAR (static), currentSpeed, gear

Dichiarazione

- In JAVA le variabili vanno sempre dichiarate



Tipi di dato primitivi

Tipo	Descrizione
byte	8-bit, intero [-128, 127], valore di default *0*
short	16-bit, intero [-32.768, 32.767], valore di default *0*
int	32-bit, intero [-2^{31},2^{31}-1], valore di default *0*
long	64-bit, intero [-2^{63},2^{63}-1], valore di default *0*
float	32-bit IEEE 754 floating point, valore di default *0*
double	64-bit IEEE 754 floating point, valore di default *0*
boolean	true/false, valore di default *false*
char	16-bit Unicode character, \u0000-\uFFFF, valore di default *\u0000*

Ogni tipo primitivo ha anche un sua rappresentazione ad oggetti: *Integer, Long, Float, Double, Character, Boolean*

Le variabili locali non hanno un valore di default devono essere inizializzate!

Il tipo String

- Ogni sequenza di caratteri tra doppi apici è un'istanza dell'oggetto String: *"this is a string"*
  - JAVA crea automaticamente un'istanza di String
- Gli oggetti String sono immutabili
  - Non cambiano mai valore
- Il tipo stringa **non è un tipo primitivo ma un oggetto JAVA**
- Il suo valore di default è *null*, come tutti gli oggetti JAVA

I letterali

- I letterali (literals) sono i valori che assegniamo alle variabili di tipo primitivo
  - Un caso particolare sono le String

```
boolean r = true;
char c = 'Z';
byte b = 100;
short s = 10000;
int i = 100000;
```

Letterali interi

- Sono sempre di tipo *int* almeno che non venga specificato (l o L)

```
int n=10;
long m=1000L;
```

- Oppure possono essere espressi in esadecimale o binario

```
int h=0xa1;
int b=0b001101;
```

Letterali floating-point

- Sono sempre di tipo *double* almeno che non venga specificato (f o F)

```
float f=3.14f;
double d1=134.54
double d2=1.3454e2 //scientific notation
```

Letterali character e String

- Possono contenere qualsiasi carattere Unicode a 16-bit
  - singoli apici per i caratteri, ‘c’
  - doppi apici per String ‘’questo è un testo’’
- Sequenze di escape speciali
  - \n, \r, \f, \b: line feed, carriage return, form feed, backspace
  - \t: tab
  - \’’, \’, \\: doppio apice, singolo apice, backslash

Array…

- E’ un **oggetto** che contiene un numero finito di oggetti (o tipi primitivi) dello stesso tipo
- La lunghezza dell’array è definita al momento della sua creazione
  - Non può essere cambiata
- *String[] main* era un esempio di array di tipo String

…Array…

- Ogni oggetto nell'array è detto elemento
- Si accede all'elemento attraverso il suo indice nell'array
  - Gli indici partono da 0, es. *al nono elemento si accede con l'indice 8*
- Dichiarazione: *float[] v;*
- Inizializzazione: *float[] v=new float[100];*
- Assegnazione: *v[3]=1.3;*

…Array

- Dichiarazione e inizializzazione: *int[] a = {10, 34, 21};*
- Array multidimensionali: double[][] m;
- Copia di array: la classe *System* mette a disposizione un metodo *arraycopy* per copiare array

Copia di array…

arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

- *src*: array sorgente
- *srcPos*: posizione di inizio in src
- *dest*: array di destinazione
- *destPos*: posizione di inizio in dest
- *length*: numero di elementi da copiare

Copia di array

```
int[] a = {10, 30, 20, 15, 45};
int[] b = new int[3];
System.arraycopy(a, 2, b, 0, 3);
```

```
b[0] = ?
b[1] = ?
b[2] = ?
```

Copia di array

```
int[] a = {10, 30, 20, 15, 45};
int[] b = new int[3];
System.arraycopy(a, 2, b, 0, 3);
```

```
b[0] = 20
b[1] = 15
b[2] = 45
```

Manipolazione di array

La classe Arrays mette a disposizione un serie di metodi statici per manipolare gli array

- http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

- *copyOfRange*: copia di array
- *fill*: riempimento di array
- *equals*: confronto tra due array
- *search:* ricerca di un elemento
- *sort*: ordinamento crescente

Operatori…

- Gli operatori eseguono delle operazioni su uno, due o tre argomenti e restituiscono un valore
- Gli operatori che hanno stessa precedenza sono eseguiti da sinistra a destra, tranne le operazioni di assegnamento

Operatori	Precedenza
postfix	expr++, expr--
unary	++expr, --expr, +expr, -expr, ~, !
multiplicative	*, /, %
additive	+, -
shift	<<, >>, >>>
relational	<, >, <=, >=, instanceof

…Operatori

Operatori	Precedenza
equality	==, !=
bitwise AND	&
bitwise XOR	^
bitwise OR	|
logical AND	&&
logical OR	||
ternary	?, :
assignment	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=

Operatori aritmetici…

- L'operatore più utilizzato è senza dubbio quello di assegnazione =

```
int speed = 0;
int gear = 1;
```

- Operatori aritmetici

+	somma	int s = 3 + 4;
-	differenza	int d = 3 - 2;
*	moltiplicazione	int m = 3 * 2;
/	divisione	int d = 10 / 2;
%	modulo	int r = 11 % 5;

…Operatori aritmetici

- E’ possibile combinare assegnazione ed operatori aritmetici:
  x += 1; è equivalente a x = x + 1;
  x *= 2; è equivalente a x = x * 2;
- L’operatore + può essere utilizzato per concatenare stringhe
  String a=‘’Hello ‘’;
  String b=‘’ world!’’;
  String msg = a + b; (‘’Hello world!’’)

Operatori unari

+	indica i numeri positivi, può essere omesso	int a=+1;
-	indica i numeri negativi	int b=-a;
++	incremento	a++;
--	decremento	b--;
!	complemento logico, inverte il valore di un boolean	boolean b = true; boolean a = !b;

Operatori di uguaglianza e relazionali

== uguale a

!= diverso da

> maggiore di

>= maggiore o uguale a

< minore di

<= minore o uguale a

Restituiscono un boolean come risultato del confronto tra due espressioni

Operatori condizionali

Operatori logici tra due espressioni

- && AND-logico
- || OR-logico

Le espressioni sono valutate solo se necessario

Operatore ternario: *<condition> ? <true> : <false>*

int result = i > 3 ? 1 : 0; *//se i è maggiore di 3 allora a result viene assegnato 1 altrimenti 0*

L’operatore instanceof

- *instanceof* è un particolare operatore per stabile se un oggetto è istanza di una particolare classe

```
String s = ‘’sono una stringa’’;
boolean t = s instanceof String;
boolean f = s instanceof Double;
```

Bitwise e bit shift

- ~ complemento della rappresentazione binaria, trasforma gli 0 in 1 e viceversa
- << sposta i bit a sinistra
- >> sposta i bit a destra
- >>> sposta i bit a destra senza conservare il segno
- operatori binari sui bit:
  - & AND
  - ^ XOR
  - | OR

Le espressioni

- Un’espressione è composta da variabili, operatori e chiamate a metodi
- Un’espressione restituisce un singolo risultato
- Il tipo di risultato restituito dipende dalle variabili e dagli operatori (e dai valori restituiti dai metodi)
- Le parantesi possono essere utilizzate per definire l’ordine di valutazione degli operatori

Gli statement

- Uno statement è una singola istruzione e termina sempre con il **;**
- Gli statement sono di quattro tipi:
  1. Assegnazione
  2. ++ o --
  3. Chiamata ad un metodo
  4. Creazione di un oggetto

I blocchi

- I blocchi sono gruppi di istruzioni racchiusi tra parentesi graffe

```
if (condition) { // begin block 1
    System.out.println("Condition is true.");
} // end block one
else { // begin block 2
    System.out.println("Condition is false.");
} // end block 2
```

JAVA

CONTROLLO DEL FLUSSO

if-then e if-then-else

- Esegue una particolare porzione di codice (blocco) solo se si verifica una condizione

```
if (condizione) {
    \\blocco if
}
```

- Il blocco if viene eseguito solo se la condizione è vera

if-then e if-then-else

- Esegue una particolare porzione di codice (blocco) solo se si verifica una condizione

```
if (condizione) {
    \\blocco if
} else {
    \\blocco else
}
```

- Il blocco if viene eseguito solo se la condizione è vera
- Il blocco else solo se la condizione è falsa

if-else annidati

- E’ possibile annidare più if

```
if (condizione1) {
    \\blocco if condiz-1
} else if (condizione2) {
    \\blocco condiz-2
} else if (condizione3) {
    \\blocco condiz-3
} else {
    \\blocco else
}
```

Esempio if

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

        int testscore = 76;
        char grade;

        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else if (testscore >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
}
```

L’istruzione switch

- Definisce diversi percorsi di esecuzione in base al valore che assume una variabile
- Può essere applicato solo ai tipi primitivi byte, short, int, char, a String, ed enumerativi

```
switch (<variabile>) {
   case <valore1>: …..; break;
   case <valore2>: …..; break;
…
   default: ….; break;
}
```

Esempio switch

```
public static void main(String[] args) {
        int month = 8;
        String monthString;
        switch (month) {
            case 1:  monthString = "January";
                     break;
            case 2:  monthString = "February";
                     break;
            case 3:  monthString = "March";
                     break;
            case 4:  monthString = "April";
                     break;
            case 5:  monthString = "May";
                     break;
            case 6:  monthString = "June";
                     break;
```

Esempio switch

```
            case 7:  monthString = "July";
                     break;
            case 8:  monthString = "August";
                     break;
            case 9:  monthString = "September";
                     break;
            case 10: monthString = "October";
                     break;
            case 11: monthString = "November";
                     break;
            case 12: monthString = "December";
                     break;
            default: monthString = "Invalid month";
                     break;
        }
        System.out.println(monthString);
    }
```

while

- Esegue un blocco di istruzioni finché una condizione è vera

```
while (condizione) {
    //blocco istruzioni while
}
```

Esempio while

```
public static void main(String[] args){
        int count = 1;
        while (count < 11) {
            System.out.println("Count is: " + count);
            count++;
        }
}
```

do-while

- Simile al while ma il controllo della condizione viene fatto alla fine del ciclo

```
do {
    //blocco istruzioni while
} while (condizione);
```

- Il ciclo viene eseguito almeno una volta!

Esempio do-while

```
public static void main(String[] args){
        int count = 1;
        do {
            System.out.println("Count is: " + count);
            count++;
        } while (count < 11);
}
```

for

- L’istruzione **for** permette di iterare un blocco di istruzioni su un intervallo di valori

```
for (initialization; termination; increment) {
    statement(s) //blocco istruzioni
}
```

Esempio for

```
public static void main(String[] args){
        for(int i=1; i<11; i++){
            System.out.println("Count is: " + i);
        }
}
```

for, Array, Collection

- Il for è spesso utilizzato per iterare sugli oggetti di un Array o di una Collection, in questo caso esiste una forma compatta (*for-each*)

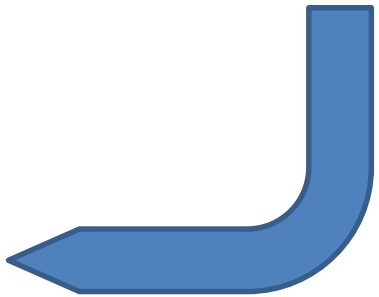
```
public static void main(String[] args){
        int[] numbers = {1,2,3,4,5,6,7,8,9,10};
        for (int item : numbers) {
            System.out.println("Count is: " + item);
        }
    }
```

break statement

- Interrompe un ciclo (quello più innestato rispetto all'istruzione *break*

```
for (i = 0; i < arrayOfInts.length; i++) {
        if (arrayOfInts[i] == searchfor) {
            foundIt = true;
            break;
        }
}
```

Se la condizione è vera il for viene interrotto



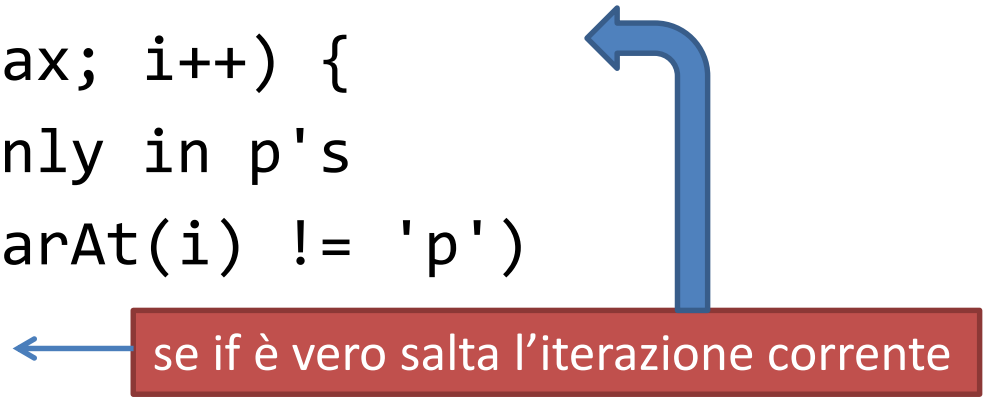
Esempio **break**

```
public static void main(String[] args) {
        int[] arrayOfInts = { 32, 87, 3, 589, 12, 1076, 2000, 8, 622, 127 };
        int searchfor = 12;
        int i;
        boolean foundIt = false;
        for (i = 0; i < arrayOfInts.length; i++) {
            if (arrayOfInts[i] == searchfor) {
                foundIt = true;
                break;
            }
        }
        if (foundIt) {
            System.out.println("Found " + searchfor + " at index " + i);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
```

continue statement

- Salta la corrente iterazione

```
for (int i = 0; i < max; i++) {
      // interested only in p's
      if (searchMe.charAt(i) != 'p')
            continue;
     // process p's
    numPs++;
}
```



Esempio **continue**

```
public static void main(String[] args) {
        String searchMe = "peter piper picked a " + "peck of pickled
peppers";
        int max = searchMe.length();
        int numPs = 0;
        for (int i = 0; i < max; i++) {
            // interested only in p's
            if (searchMe.charAt(i) != 'p')
                continue;
            // process p's
            numPs++;
        }
        System.out.println("Found " + numPs + " p's in the string.");
}
```

L’istruzione **return**

- Termina l’esecuzione di un metodo
  - senza restituire valore: *return;*
  - restituendo un valore: *return x;*

THE END


NOT SURE IF END OF PRESENTATION
OR BEGINNING OF NEW SHOW
memegenerator.net