

Fondamenti di Informatica

Java

SUPSI Dipartimento Tecnologie Innovative

Gianni Grasso

16 ottobre 2024

Classe: I1B

Anno scolastico: 2024/2025

Indice

1	Introduzione	3
1.1	Compilazione ed esecuzione	3
1.2	Alcune definizioni	3
2	Variabili	4
3	Costanti	5
4	Identificatori	6
5	Tipi di dato	7
5.1	boolean	7
5.2	Tipi numerici	7
5.2.1	Interi	7
5.2.2	char	7
5.2.3	Decimali	7
6	La classe Math	8
6.1	Generare numeri casuali	8
7	La classe Scanner	9
8	Istruzioni di controllo	10
8.1	Switch	10
8.2	Operatore ternario	11
8.3	Ciclo for	11
8.4	L'istruzione continue e break	11
8.5	L'istruzione do while	12

1 Introduzione

Java è un linguaggio di programmazione **orientato agli oggetti** progettato per essere il più possibile indipendente dalla piattaforma di esecuzione.

1.1 Compilazione ed esecuzione

- **JRE**, Java Runtime Environment, è un ambiente che permette di eseguire applicazioni java, contiene la *Java Virtual Machine (JVM)* e le Java APIs.
- **JDK**, Java Development Kit, una collezione di tools per la programmazione in Java: compilatore, debugger, eccetera. Esso comprende una versione della JRE al suo interno

Nel codice sorgente ogni programma Java è rappresentato da **una classe**. Il nome del file **deve** coincidere con quello della classe.

Ecco la struttura di base di un programma Java:

```
/**
 * Esempio di programma.
 */
public class NomeClasse {
    public static void main(String[] args) {
        //codice del programma
    }
}
```

1.2 Alcune definizioni

- **Letterali**, un letterale è la rappresentazione di un qualsiasi valore di tipo primitivo, stringa o null, in altre parole tutto ciò che potrebbe essere assegnato ad una variabile
- **Espressioni**, un'espressione è un costrutto (porzione di codice) che quando viene elaborato assume un singolo valore. Essa può essere composta da variabili, operatori, letterali, separatori e invocazioni di funzioni. Il suo tipo di dato dipende dagli elementi che la compongono
- **Istruzioni**, l'istruzione è l'unità di esecuzione di Java. Ogni istruzione deve essere completata con il punto e virgola (;), fatta eccezione per le istruzioni di controllo di flusso del codice (if, while, ...)

2 Variabili

Ad ogni variabile è associata una zona di memoria. Le variabili vanno dichiarate fornendo il tipo di dato e il nome. Una variabile è accessibile dal momento in cui viene dichiarata **fino alla fine del blocco in cui è contenuta**. Ecco un esempio:

```
int nomeVariabile; //non inizializzata
int x = 20 + 30; //inizializzata

int y = 1, z; //dichiara e assegna y e dichiara z

double d = z / (y * 1.75); //svolge prima l'espressione e poi l'assegnazione
```

Nota: prima di poter leggere il valore di una variabile, essa deve essere inizializzata, in caso contrario il codice non compila.

```
int b;
if (condizione) {
    b = 0;
}
System.out.println("b = " + b); //non compila. b potrebbe non essere inizializzata
```

3 Costanti

Le costanti in Java si dichiarano **fuori** dalla procedura `main`:

```
public class Sfera {  
    static final double PI = 3.14159265;  
    public static void main(String[] args) {  
        ...  
    }  
}
```

4 Identificatori

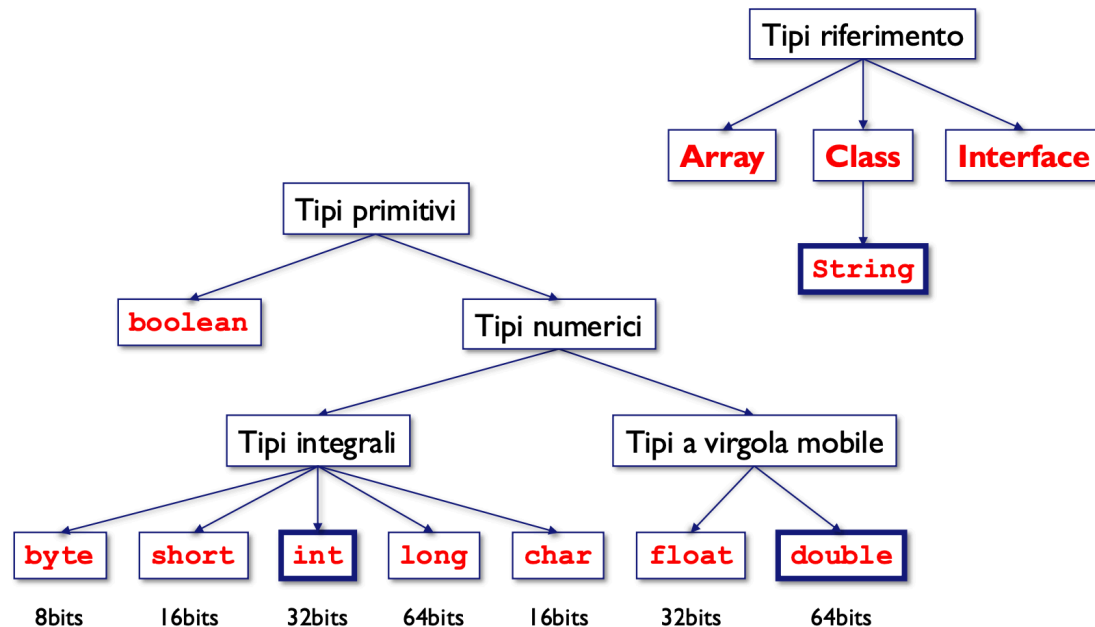
Gli identificatori sono i nomi delle variabili, delle procedure e delle classi. Un identificatore è una sequenza **senza spazi** e di lunghezza illimitata di lettere, cifre e simboli `_` e `$`. Inoltre un identificatore non può essere identico ad una *keyword*, ad un valore booleano (`true` o `false`) o al valore `null`.

- **nomi delle classi**, si utilizza la sintassi *UpperCamelCase*.
- **nomi delle variabili**, si utilizza la sintassi *lowerCamelCase*.

Nota: Java è case sensitive.

5 Tipi di dato

Java è un linguaggio di programmazione fortemente tipizzato, ogni variabile e ogni espressione ha un tipo conosciuto al momento della compilazione. Ecco uno schema con i principali tipi di dato:



5.1 boolean

Il tipo `boolean` può assumere esclusivamente due valori: `true` e `false`. I valori booleani si ottengono anche dalla valutazione di espressioni condizionali, ad esempio:

```
boolean risultato = tasso > 0.05;
```

5.2 Tipi numerici

5.2.1 Interi

I tipi numerici interi sono rispettivamente `byte`, `short`, `int` e `long`. L'unica differenza tra di essi è il range di numeri che è possibile rappresentare. Per valori ancora più grandi è possibile utilizzare il tipo di dato `java.math.BigInteger`.

5.2.2 char

Serve per rappresentare i singoli caratteri. Esso permette l'utilizzo dei caratteri **Unicode**, uno standard per la rappresentazione consistente del testo. Ad ogni carattere è assegnato un valore numerico ed è quindi possibile fare operazioni aritmetiche sui caratteri.

5.2.3 Decimali

È molto raro, ma ci sono alcuni numeri decimali che non possono essere rappresentati. In generale per comparare due valori decimali non utilizziamo `==`:

```
static final double EPSILON = 1e-8; //1 * 10^-8
if(Math.abs(a-b) < EPSILON) { //confronta a e b
    ...
}
```

I tipi decimali utilizzati solitamente sono `float` e `double`, per valori ancora più grandi è possibile utilizzare il tipo di dato `java.math.BigDecimal`.

6 La classe Math

La classe `Math` mette a disposizione le seguenti funzioni:

- `Math.pow(base, esponente)`, elevamento a potenza
- `Math.sqrt(numero)`, radice quadrata
- `Math.log(numero)`, logaritmo naturale
- `Math.log10(numero)`, logaritmo in base 10
- `Math.sin(radiani)`, seno di un angolo in radianti
- `Math.cos(radiani)`, coseno di un angolo in radianti
- `Math.tan(radiani)`, tangente di un angolo in radianti
- `Math.random()`, genera un numero casuale tra 0.0 (compreso) e 1.0 (escluso)

E le seguenti costanti:

- `Math.E`, base del logaritmo naturale
- `Math.PI`, pi greco

6.1 Generare numeri casuali

Per generare numeri casuali con `Math.random()` si utilizza:

```
int randomInt = (int) ((max - min) * Math.random() + min)
```

Dove `max` rappresenta il valore massimo del range (non compreso) e `min` quello minimo (compreso).

Se ad esempio volessi avere un range contenente tutti i numeri da 10 a 50 (entrambi compresi) sarebbe:

```
int randomInt = (int) ((51 - 10) * Math.random() + 10) //il 51 non è compreso
```


7 La classe Scanner

La classe `Scanner` è un semplice scanner di testo che può analizzare tipi primitivi e stringhe. Per utilizzare uno scanner è necessario importare la classe, istanziarlo e **chiuderlo**:

```
import java.util.Scanner; //importa la classe

public class EsempioHasNextInt {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); //istanzia uno scanner

        System.out.print("Inserisci un numero intero: ");
        String x = scanner.next(); //utilizza lo scanner

        scanner.close(); //chiude lo scanner
    }
}
```

Quando utilizziamo uno scanner per leggere un numero, premendo invio lasciamo `\n` nel buffer, in questo modo quando poi proviamo a leggere una stringa, verrà letto soltanto il valore lasciato nel buffer. Per evitare questo ogni volta che leggiamo un valore numerico con uno scanner, chiamiamo successivamente il metodo `nextLine()`, in modo che svuoti il buffer prima della lettura della stringa.

```
Scanner scanner = new Scanner(System.in);

// Lettura di un numero intero
System.out.print("Inserisci un numero intero: ");
int numero = scanner.nextInt(); // Legge il numero, lascia '\n' nel buffer

// Dopo aver letto un numero con nextInt(), rimane il carattere '\n' nel buffer.
// Il metodo nextLine() viene utilizzato qui per consumare quel '\n' ed evitare
// problemi nella successiva lettura di una stringa.
scanner.nextLine();

// Lettura di una stringa
System.out.print("Inserisci una stringa: ");
String stringa = scanner.nextLine();
```

Per verificare che l'utente inserisca un valore del tipo aspettato quando usiamo uno scanner possiamo usare:

```
System.out.print("Inserisci un numero intero: ");

// Ciclo che continua fino a quando non viene inserito un numero intero valido
while (!scanner.hasNextInt()) { // hasNextInt() controlla se il prossimo input
    // è un numero intero, ma non consuma il valore
    System.out.print("Input non valido. Inserisci un numero intero: ");
    scanner.nextLine(); // Scarta l'input non valido
}

// Una volta ottenuto un intero, lo memorizziamo e lo stampiamo
int number = scanner.nextInt();
```

8 Istruzioni di controllo

8.1 Switch

Ci sono due sintassi per l'istruzione `switch` in Java:

- ```
switch (espressione) {
 case valore1:
 ...;
 break;
 case valore2:
 ...;
 break;
 case valore3:
 ...;
 break;
 default:
 ...;
}
```

**Nota:** L'opzione `default` è opzionale. Se non è presente il `break`, il programma continuerà al `case` successivo, anche se la condizione non è soddisfatta.

È anche possibile concatenare più `case` su una sola linea se la condizione è la stessa:

- ```
switch (espressione) {  
    case valore4: case valore6: case valore9:  
        ...;  
        break;  
    case valore2:  
        ...;  
        break;  
    default:  
        ...;  
}
```
- ```
switch (espressione) {
 case valore1 -> {
 ...;
 }
 case valore2 -> {
 ...;
 }
 case default -> {
 ...;
 }
}
```

**Nota:** questa versione non necessita dei `break`, impedisce il `fall-through`

Analogamente all'esempio di prima possiamo fare:

```
switch (espressione) {
 case valore1, valore2, valore3 -> ...;
 case valore4, valore5, valore6 -> ...;
 default -> ...;
}
```

**Nota:** per assegnare un valore quando si utilizzano i blocchi di codice dobbiamo usare la keyword `yield`. Se vogliamo assegnare un valore di default dobbiamo utilizzarlo per forza.

Gli **switch** funzionano con:

- `char`
- `byte`
- `short`
- `int`
- `String`

**Nota:** non funziona con valori decimali come ad esempio `double`.

## 8.2 Operatore ternario

Detto anche `inline if`, restituisce il primo valore se la condizione è vera mentre restituisce il secondo se la condizione è falsa:

```
condizione ? valoreSeVero : valoreSeFalso
```

Ad esempio:

```
if (n % 2 == 0) {
 prossimo = n / 2;
} else {
 prossimo = 3 * n + 1;
}

// sono uguali
prossimo = (n % 2 == 0) ? (n / 2) : (3 * n + 1);
```

## 8.3 Ciclo for

```
for (inizializzazione; condizione; aggiornamento) {
 seqIstruzioni;
}
```

**Nota:** nessuno dei parametri del ciclo `for` (`inizializzazione`; `condizione`; `aggiornamento`) è obbligatorio.

È anche possibile utilizzare più variabili o condizioni:

```
for (double i = 0.5, j = 39; i + j < 40; i++, j--) {
 ...
}
```

## 8.4 L'istruzione `continue` e `break`

Queste due istruzioni vengono utilizzate all'interno di un ciclo:

- `continue` fa continuare il ciclo dall'iterazione successiva
- `break` esce dal ciclo corrente

## 8.5 L'istruzione `do while`

Prima esegue le istruzioni (almeno una volta) poi verifica se la condizione è vera.

```
do {
 seqIstruzioni;
} while (condizione);
```