



informatica

Java – basi del linguaggio



struttura di un programma Java

A. Ferrari

```
/**
 * Classe EsempioProgramma
 * Un esempio di programmazione in Java
 * @author 4A Informatica
 */
public class EsempioProgramma {
    public static void main( String[] args ) {
        System.out.println("Il mio programma Java!");
    }
}
```

- Stile *C*

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

- Stile *Java*

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

- commenti su una sola riga, delimitati da `//`
- commenti su più righe, delimitati da
 - `/*`
 - `*/`
- • commenti *javadoc* delimitati da
 - `/**`
 - `*/`

- o la dimensione dei tipi di dato è *standard*, e non dipende dal sistema operativo

Data Types					
Type Name	Minimum Value	Maximum Value	Default	Size	Literal
byte	-128	127	0	8-bit +/-	_____
short	-32768	32767	0	16-bit +/-	_____
int	-2147483648	2147483647	0	32-bit +/-	3, 077, 0xBAAC
long	-9223372036854775808	9223372036854775807	0	64-bit +/-	3L
float	-1.40239846e-45	3.40282347e+38	0.0	32-bit IEEE float	3.0F, 3.0E2F
double	-4.94065645841246533e-324	1.79769313486231570e+308	0.0	64-bit IEEE float	3.0, 3.0E2, 3.0e2D
boolean	false	true	false	N/A	true, false
char	\u0000	\uffff	\u0000	16-bit Unicode	'3'

- le variabili devono essere *dichiarate*:
int conta;
- prima di utilizzarle devono essere *inizializzate*:
conta = 0;
- è possibile dichiarazione e inizializzazione contemporanea:
int conta = 0;
- è possibile la dichiarazione multipla:
int conta, altezza;

- Java utilizza il codice UNICODE che prevede 2 byte per la memorizzazione di ogni singolo carattere
- *primi 128 codici Unicode == ASCII*

```
char c = 'e';
```



The image displays a portion of the Unicode character set, organized into a grid. The top section shows Korean characters (Hangul) in various combinations. Below that, there are rows of Latin characters with diacritics, Greek characters, and Cyrillic characters. A large, stylized 'UNICODE' logo is centered over the table. The grid is labeled with hexadecimal and decimal values along the top and left sides.

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	(NULL)	32	20	(SPACE)	64	40	@	96	60	`
1	1	(START OF HEADING)	33	21	!	65	41	A	97	61	a
2	2	(START OF TEXT)	34	22	"	66	42	B	98	62	b
3	3	(END OF TEXT)	35	23	#	67	43	C	99	63	c
4	4	(END OF TRANSMISSION)	36	24	\$	68	44	D	100	64	d
5	5	(ENQUIRY)	37	25	%	69	45	E	101	65	e
6	6	(ACKNOWLEDGE)	38	26	&	70	46	F	102	66	f
7	7	(BELL)	39	27	'	71	47	G	103	67	g
8	8	(BACKSPACE)	40	28	(72	48	H	104	68	h
9	9	(HORIZONTAL TAB)	41	29)	73	49	I	105	69	i
10	A	(LINE FEED)	42	2A	*	74	4A	J	106	6A	j
11	B	(VERTICAL TAB)	43	2B	+	75	4B	K	107	6B	k
12	C	(FORM FEED)	44	2C	,	76	4C	L	108	6C	l
13	D	(CARRIAGE RETURN)	45	2D	-	77	4D	M	109	6D	m
14	E	(SHIFT OUT)	46	2E	.	78	4E	N	110	6E	n
15	F	(SHIFT IN)	47	2F	/	79	4F	O	111	6F	o
16	10	(DATA LINK ESCAPE)	48	30	0	80	50	P	112	70	p
17	11	(DEVICE CONTROL 1)	49	31	1	81	51	Q	113	71	q
18	12	(DEVICE CONTROL 2)	50	32	2	82	52	R	114	72	r
19	13	(DEVICE CONTROL 3)	51	33	3	83	53	S	115	73	s
20	14	(DEVICE CONTROL 4)	52	34	4	84	54	T	116	74	t
21	15	(NEGATIVE ACKNOWLEDGE)	53	35	5	85	55	U	117	75	u
22	16	(SYNCHRONOUS IDLE)	54	36	6	86	56	V	118	76	v
23	17	(ENG OF TRANS. BLOCK)	55	37	7	87	57	W	119	77	w
24	18	(CANCEL)	56	38	8	88	58	X	120	78	x
25	19	(END OF MEDIUM)	57	39	9	89	59	Y	121	79	y
26	1A	(SUBSTITUTE)	58	3A	:	90	5A	Z	122	7A	z
27	1B	(ESCAPE)	59	3B	;	91	5B	[123	7B	{
28	1C	(FILE SEPARATOR)	60	3C	<	92	5C	\	124	7C	
29	1D	(GROUP SEPARATOR)	61	3D	=	93	5D]	125	7D	}
30	1E	(RECORD SEPARATOR)	62	3E	>	94	5E	^	126	7E	~
31	1F	(UNIT SEPARATOR)	63	3F	?	95	5F	_	127	7F	(DEL)

- **output**: per scrivere sul terminale si usa il metodo `System.out.println()`, in questo modo:

```
System.out.println("Hello world");
```

```
System.out.println("3 più 5 fa " + (3+5) + ".");
```

- esercizio:
 - realizzare un programma che produce la somma di due numeri

- per dichiarare in Java un array di interi è possibile scrivere:

```
int[] a;
```

dove a è il nome dell'array

- in alternativa è anche possibile scrivere:

```
int a[];
```

- l'istanziamento è la fase in cui viene *allocata* la memoria necessaria per contenere tutti gli elementi di un array
- nella fase di istanziamento viene definita la *dimensione* dell'array
- l'operatore **new** effettua l'istanziamento
`a = new int[8];`
- è anche possibile la dichiarazione e istanziamento con una sola istruzione:
`int[] a = new int[8];`

- per accedere ad un elemento si specifica il suo indice
`a[1] = 3;`
- l'indice parte sempre da **0**
- gli elementi dell'array vengono direttamente *inizializzati* (a 0 se numerici)
- un metodo alternativo per l'istanziamento in fase di dichiarazione è il seguente:
`int[] a = { 1, 3, 5, 7, 9, 0, 0, 0 };`

- una variabile di tipo array può puntare ad array di qualsiasi dimensione
- è possibile anche *riassegnare* un nuovo array a una variabile che prima ne conteneva uno di dimensione diversa
- per esempio, è possibile creare prima un array di 8 elementi, e in seguito uno di 10:

```
int[] a = new int[8]; a = new int[10];
```
- non esiste un modo per ridimensionare un array, la sua dimensione, decisa in fase di definizione, è costante

- uno stesso array può essere associato a variabili diverse

- es:

```
int[] a = new int[8];
```

```
int[] b = a;
```

- ***a*** e ***b*** puntano alla stessa zona di memoria

```
a[5] = 63;
```

```
int i = b[5]; //i vale ora 63
```

- per conoscere la lunghezza di un array, è possibile utilizzare la proprietà `length`
- per esempio:

```
int[] a = new int[10];  
int lung = a.length; //lung vale 10
```
- **non** è possibile accedere a un elemento dell'array oltre i suoi limiti, si otterrebbe un **errore** in esecuzione

```
if (espressione) {  
    //istruzioni...  
} else {  
    //istruzioni...  
}
```

strutture di controllo: switch

```

switch( espressione ) {
  case caso_1:
    //istruzioni
    break;
  case caso_2:
    //istruzioni
    break;
  case caso_n:
    //istruzioni
    break;
  default:
    //istruzioni
    break;
}

```

```

int seme = 1;
switch( seme ) {
  case 1:
    //cuori
    break;
  case 2:
    //quadri
    break;
  case 3:
    //fiori
    break;
  case 4:
    //picche
    break;
  default:
    //caso non previsto
    break;
}

```

L'espressione di controllo può essere solo di tipo **intero** o **carattere**. Non sono supportati valori booleani o reali.

```
int mese = 1;
switch( mese ) {
    case 12:
    case 1:
    case 2:
        //inverno
        break;
    case 3:
    case 4:
    case 5:
        //primavera
        break;
    case 6:
    case 7:
    case 8:
        //estate
        break;
    case 9:
    case 10:
    case 11:
        //autunno
        break;
    default:
        //caso non previsto
        break;
}
```

```
while (espressione) {  
    //istruzioni  
}
```

```
int i = 0;  
while (i<5) {  
    i = i + 1;  
}
```

```
do {  
    //istruzioni  
} while(espressione);
```

```
int i=0;  
do {  
    i = i + 1;  
} while( i<5 );
```

```
for(istr_iniziale;istr_contr;istr_iterazione) {  
    istruzioni  
}
```

```
for(i=0;i<10;i++) {  
    //istruzioni  
}
```

Operatore	Descrizione	Uso	Significato
+	Somma	op1+op2	Somma il valore di op1 a quello di op2
-	Sottrazione	op1-op2	Sottrae al valore di op1 quello di op2
*	Moltiplicazione	op1*op2	Moltiplica il valore di op1 con quello di op2
/	Divisione	op1/op2	Divide il valore di op1 con quello di op2
%	Modulo	op1%op2	Calcola il resto della divisione tra il valore di op1 e quello di op2
-	Negazione aritmetica	-op	Trasforma il valore di op in positivo o negativo

Operatore	Descrizione	Uso	Significato
++	Incrementa l'operando di 1 (prefissa)	++op	Incrementa op di 1; valuta il valore dopo aver incrementato
++	Incrementa l'operando di 1 (postfissa)	op++	Incrementa op di 1; valuta il valore prima di incrementare
--	Decrementa l'operando di 1 (prefissa)	--op	Decrementa op di 1; valuta il valore dopo aver incrementato
--	Decrementa l'operando di 1 (postfissa)	op--	Decrementa op di 1; valuta il valore prima di incrementare

Operatore	Descrizione	Uso	Descrizione Restituisce true se
==	Uguale a	op1 == op2	op1 e op2 sono uguali
!=	Diverso da	op1 != op2	op1 e op2 sono diversi
<	Minore di	op1 < op2	op1 è minore di op2
>	Maggiore di	op1 > op2	op1 è maggiore di op2
<=	Minore o uguale di	op1 <= op2	op1 è minore o uguale di op2
>=	Maggiore o uguale di	op1 >= op2	op1 è maggiore o uguale di op2

Condition	Operator	Example
If one condition AND another condition	&&	int i = 2; int j = 8; ((i < 1) && (j > 6))
If either one condition OR another condition		int i = 2; int j = 8; ((i < 1) (j >= 10))
NOT	!	int i = 2; !(i > 3))

- Java dispone di una completa libreria di gestione della comunicazione delle informazioni da e verso dispositivi esterni alla memoria centrale del computer (*tastiera, video, stampante, disco fisso, dischi USB/Firewire, plotter, joystick, schede di rete ...*)

- un programma Java può produrre un output testuale sul video in due modi diversi:
 - utilizzando lo *standard output*
 - oppure lo *standard error*
- di default entrambi stampano su *console*
- output su standard output:
`System.out.println("Ciao Mondo!");`
- `println()` stampa il valore del parametro seguito da un ritorno a capo
 - per stampare senza andare a capo utilizzare `print()`
- `print` e `println` possono stampare *tutti i tipi di dato* primitivi e le stringhe

- per leggere dati da tastiera si utilizza lo standard input (classe **System.in**)

- per acquisire i dati è possibile utilizzare la classe

```
java.util.Scanner:
```

```
import java.util.Scanner;
```

```
...
```

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

```
System.out.print("Inserire una stringa: ");
```

```
String stringa = tastiera.nextLine();
```

- Scanner ha metodi per acquisire altri tipi di dato

- esempio, per acquisire un numero intero:

```
int num = tastiera.nextInt();
```

- si vuole calcolare il *Massimo Comun Divisore* ed il *minimo comune multiplo* di un insieme di valori interi
 - prima versione: i valori sono inseriti nell'array in fase di dichiarazione/definizione
 - seconda versione i valori sono ricevuti in input (il primo input definisce il numero totale dei valori)
 - terza versione: i valori sono ricevuti in input ma non è conosciuto a priori il loro numero, l'inserimento del valore -1 determina la fine dei valori immessi

- il tipo di dato stringa in Java *non è* un tipo *primitivo*
- il tipo di dato che definisce una stringa è **String**
- dichiarazione:
String nome;
- per inizializzare una stringa è necessario specificarne il valore racchiudendolo tra doppi apici.
String nome = "Jack";
- è anche possibile utilizzare una forma più esplicita:
String nome = new String("Jack");

- il confronto tra due stringhe avviene con il metodo **equals ()** anziché con l'operatore **==**

```
String nome1 = "Jack";
```

```
String nome2 = "Pippo";
```

```
boolean uguali = nome1.equals( nome2 ); //ritorna false
```

- l'operatore **==**, non confronta carattere per carattere, ma verifica se due stringhe puntano alla stessa area di memoria

```
String nome1 = new String("Jack");  
String nome2 = new String("Jack");  
boolean uguali = nome1.equals( nome2 ); //ritorna true  
boolean stesso = (nome1 == nome2)      //ritorna false
```

- *lunghezza* di una stringa \Rightarrow metodo **length()** :

```
String nome = "Jack";
nome.length(); //ritorna 4
```
- è possibile *concatenare* le stringhe utilizzando il metodo **concat()**

```
String a = "Via ";
String b = a.concat("lattea");
//b vale "Via lattea"
```
- oppure utilizzare l'operatore **+**

```
String a = "Via " + "lattea";
```

- per eliminare tutti gli spazi è possibile utilizzare il metodo `trim()` :

```
String a = "Jack    ";  
a.trim() //ritorna "Jack"
```

- le stringhe possono essere anche convertite in caratteri maiuscoli o minuscoli.

```
String a = "Jack";  
a.toLowerCase() //ritorna "jack"  
a.toUpperCase() //ritorna "JACK"
```

- per sostituire un singolo carattere in una stringa si può utilizzare il metodo `replace()`
- `String a = "Roma";`
- `a.replace('R', 't');`
- //ritorna "toma";
- È possibile anche estrarre una parte di una stringa, utilizzando il metodo `substring()`.
- Il metodo `endsWith()` indica se una stringa termina con un determinato suffisso e `startsWith()` se inizia con un certo prefisso
- ...

- le stringhe in Java sono *immutabili*
 - non possono essere modificate
- quando si dichiara una variabile di tipo stringa e le si associa un valore, questo viene inserito in una particolare area di memoria (*string pool*)

```
String a = "Jack" ;
```

- se viene dichiarata una stringa con lo stesso contenuto di una esistente nello string pool, la prima viene riutilizzata
 - non esistono in memoria più copie della stessa identica stringa
- nell'esempio, la stringa **s** punta alla stessa area di memoria di **a**

```
String s = "Jack" ;
```

- se si usa *new*, vengono allocate 2 aree di memoria diverse

```
String nome1 = new String("Pippo") ;
```

```
String nome2 = new String("Pippo") ;
```