

Appunti estratti da <http://java.html.it/guide/leggi/22/guida-java/>

Le variabili sono dei valori modificabili, ovvero sono dei nomi che rappresentano un valore di un certo tipo, il valore associato al nome può essere variato. Ad esempio se dico che X è una variabile di tipo intero, e poi dico che X ha valore 10, se scrivo l'espressione $5 + X$, è come se avessi scritto l'espressione $5 + 10$. Credo che comunque tutti voi abbiate dimestichezza con le variabili, in quanto sono la cosa principale di ogni linguaggio di programmazione, detto ciò vediamo come Java tratta le variabili.

Innanzitutto, prima di essere usata, una variabile deve essere dichiarata attraverso degli identificatori.

Gli identificatori possono essere uno o più di uno, separati da virgole, e sono i nomi che avranno le variabili. Ad esempio la dichiarazione di X dell'esempio precedente sarà:

```
int X;
```

ma posso anche dichiararne più di uno, ad esempio è valida la seguente dichiarazione:

```
int X,Y,Z;  
double W=3.12;
```

Come si vede dall'esempio è possibile anche inizializzare una variabile al momento della sua creazione, semplicemente aggiungendo dopo il nome l'uguale ed il valore che prende.

Dopo la dichiarazione della variabile, e la sua inizializzazione, ovunque venga scritto l'identificatore della variabile si intende il suo valore attuale, tranne che in un caso, nell'assegnamento di un nuovo valore. Ad esempio se io scrivo $X = 10$ intendo per X la variabile e non il suo valore, e con quell'uguale 10 dico che il nuovo valore che ha la variabile è 10.

Se dopo scrivo:

```
X = X+1;
```

Con la X a sinistra dell'uguale intendo dire la variabile chiamata X, mentre con la X a destra intendo dire il valore attuale della variabile X, quell'espressione $X+1$ sarà quindi in questo caso

$10 + 1$ e quindi 11, ovvero per $X = X + 1$; ho detto $X = 11$;

Gli identificatori Java devono cominciare con una lettera (oppure con `_` o `$`) seguita da una sequenza lunga a piacere di caratteri e cifre, che essendo codificate Unicode sono molto più numerose delle lettere e delle cifre ASCII usate dagli altri linguaggi. Sono spiacente di non avere una mappa completa dei caratteri unicode da darvi, ma posso dirvi che tra i caratteri sono compresi i caratteri greci, cirillici, arabi, eccetera.

Vediamo adesso come dichiarare e inizializzare un array, la dichiarazione avviene mettendo un paio di parentesi quadre dopo il tipo:

```
int [] vettoreDiInteri;
```

Gli array in Java sono quasi degli oggetti, solo che essi non possono essere estesi per aggiungervi nuovi metodi. Quello dichiarato sopra è un riferimento ad un array, che però il linguaggio non ha allocato, se voglio inizializzare l'array elemento per elemento, come si fa con gli altri linguaggi, devo dichiararne le dimensioni:

```
int [] vettoreDiInteri = new int[100];
```

Da questo momento in poi posso iniziare a mettere elementi nel vettore nelle posizioni da 0 a 99, se vado fuori avrò una eccezione di out of range.

Al momento della creazione posso anche inizializzare un array, basta scrivere:

```
int [] vettoreDiInteri = { 1,2,3,10,100,555,0, .altri valori...};
```

Posso anche creare dei vettori i cui elementi sono dei vettori, e così via, questo si fa così:

```
double[] [] matriceIdentitaDiOrdine3 = { { 1.0 , 0.0 , 0.0 } ,  
{ 0.0 , 1.0 , 0.0 } ,  
{ 0.0 , 0.0 , 1.0 } };
```

Operatori

Gli operatori del linguaggio Java sono gli stessi degli altri linguaggi.

Ci sono i soliti operatori aritmetici: +, -, *, / e %.

Da notare che le operazioni sono anch'esse tipate, ovvero se sommo due interi ho un valore intero, così per tutte le operazioni, in modo che data una espressione è sempre possibile definirne il tipo del risultato. Java fa un forte controllo sui tipi, infatti non è possibile assegnare un carattere ad un intero, cosa possibile nel linguaggio C, ma è anche impossibile assegnare un double ad un float, senza un esplicito casting. Il casting consiste nel forzare un valore di un tipo ad assumere un tipo diverso. Se scrivo 5 io intendo un intero, mentre se ne faccio un cast a float intenderò la versione di 5 reale, per farlo scriverò (float) 5.

facendo così posso fare i famosi assegnamenti di interi a reali e di reali double a reali float, si pensi ad esempio a :

```
double v1=10.0;
float v2;
int v3=5;
```

e si pensi agli assegnamenti errati seguenti:

```
v2=v1;
v1=v3;
```

Se effettuiamo dei cast, diventano leciti anche per Java:

```
v2=(float) v1;
v1=(double) v3;
```

Da notare che la divisione tra interi, a differenza che in altri linguaggi da un intero, quindi se si vuole il numero reale derivato dalla divisione dei due numeri interi x e y, bisogna prima trasformarli in reali:

```
float risultato= (float ) x / (float) y;
```

Per le stringhe esiste un operatore di concatenamento:

```
String a="Pietro " ;
String b= a + "Castellu";
String b+="cci"
```

Il risultato sarà Pietro Castellucci. L'operatore è il +. Abbiamo ad un certo punto usato un assegnamento particolare, ovvero +=, questa è una abbreviazione, se devo scrivere a=a+1; posso abbreviare e scrivere a+=1, che è lo stesso. Vale lo stesso per ogni operatore binario.

L'operatore + appende ad una stringa anche caratteri e numeri. Sembra strano questo, anche in considerazione del fatto che abbiamo detto che le espressioni Java sono ben tipate a differenza del C, questo avviene però perché il sistema effettua delle conversioni implicite di tipi primitivi e oggetti in stringhe.

Java inoltre offre altri quattro operatori che sono delle abbreviazioni, due di incremento di variabili e due di decremento, sia X una variabile, io posso scrivere:

```
X++
++X
X--
--X
```

che stanno rispettivamente ad indicare valuta prima e poi incrementa X, incrementa prima e poi valuta X, valuta e decrementa X ed infine decrementa e valuta X, questo vuol dire che l'espressione X++ è un comando di assegnamento, ma anche una espressione che restituisce un risultato, risultato che in questo caso è X prima di essere incrementata.

Ad esempio:

```
X=10;
Y=X++;
```

mi darà come risultato X=11 e Y=10.

Altri operatori sono quelli di confronto, ovvero:

> maggiore di, x>y restituisce true se x è maggiore stretto di y, false altrimenti

>= maggiore o uguale di, x>=y restituisce true se x è maggiore o uguale di y, false altrimenti

< minore di, x<y restituisce true se x è minore stretto di y, false altrimenti

<= minore o uguale di, x<=y restituisce true se x è minore o uguale di y, false altrimenti

== uguale, x==y restituisce true se x uguale a y, false altrimenti

!= diverso, x!=y restituisce true se x diverso da y, false altrimenti

Vi sono gli operatori logici &&, || e !, che rappresentano l'and, l'or e il not:

Date x e y valori booleani, x && y sarà vera se x e y sono entrambe vere, sarà falsa altrimenti (se tutte e due false o se una vera e l'altra falsa).

x || y sarà vera se x oppure y o entrambe sono vere (sarà falsa solo se sono ambedue false).

!x sarà vera se x è falsa, mentre sarà falsa se x è vera.

Queste operazioni binarie sono molto utili nella costruzione di maschere.

L' ultimo operatore da vedere è l'operatore condizionale, che permette di definire espressioni con due differenti risultati a seconda di un valore booleano.

Si consideri ad esempio l'espressione seguente:

```
int ValoreAssoluto = ( a<0 ? -a : a );
```

ValoreAssoluto assumerà il valore -a se a è negativo oppure a se è positivo o nullo.

A questo punto siamo in grado di scrivere tutte le espressioni che vogliamo nel linguaggio Java.

Istruzione condizionale

if (EXPBOOL) IST

Questa istruzione esegue l'istruzione denominata IST, la quale può essere una qualsiasi istruzione del linguaggio, solo se l'espressione booleana EXPBOOL è vera, altrimenti viene saltata. Una variante è quella in cui se invece EXPBOOL è falsa viene eseguita un'altra istruzione, essa è:

```
if (EXPBOOL) IST
    else
        IST2
```

Da notare che IST e IST2 sono istruzioni generiche, quindi anche blocchi o altre istruzioni condizionali, in questo secondo caso si parla di if annidati.

Istruzione switch

```
Switch (EXP)
{
case CST11: .... :case CST1n: IST1
case CST21: .... : case CST2n: IST2
....
case CSTm1: .... : case CSTmn: ISTm
default ISTm+1;
};
```

Dove EXP è una espressione, CSTij sono delle costanti dello stesso tipo dell'espressione EXP e ISTi sono delle istruzioni.

L'istruzione valuta EXP, e confronta il risultato con le costanti dei case, se ne trova una uguale esegue l'istruzione corrispondente. Se invece non trova nessuna costante uguale esegue l'istruzione dopo il default, la quale è opzionale, nel caso in cui è assente e nessun case ha la costante uguale al valore di EXP si salta lo switch.

Si consideri ad esempio il seguente comando:

```
switch (5+1)
{
case 6 : System.out.println ("Bravo");
default : System.out.println ("Asino, è sei");
};
```

Istruzione for

for (exp di inizializzazione; exb booleana; exp di incremento) IStr

esegue l'istruzione IStr un numero di volte pari ai valori contenuti in un intervallo. Tipicamente l'istruzione di inizializzazione inizializza una variabile ad un valore, variabile che viene incrementata (o decrementata) dalla istruzione di incremento e nell' espressione booleana viene controllato che la variabile assuma i valori voluti, infatti appena l'espressione booleana è falsa si esce dal ciclo for.

Faccio un esempio, voglio inizializzare i valori di un vettore di 100 interi tutti a zero, invece di scrivere 100 assegnamenti farò:

```
int v = new int[100];  
for (int i = 0 ; i<100; i++) v[i] = 0 ;
```

Come si vede la variabile si può anche dichiarare nell'istruzione di inizializzazione, così si vedrà solo all'interno del for.

Istruzione while

while (EXPBOOL) IST

Esegue IST fintanto che EXPBOOL è vera, ad esempio l'inizializzazione di prima si può fare con il while così:

```
int v = new int[100];  
int i=0;  
while (i<100) {  
v[i]=0;  
i+=1;  
};
```

oppure in forma più compatta, sfruttando l'incremento in forma compatta:

```
int v = new int[100];  
int i=0;  
while (i<100) v[i++]=0;
```

Istruzione do-while

do IST while (EXPBOOL);

E' come il while, solo che prima esegue l'istruzione e poi controlla l'EXPBOOL, di nuovo l'inizializzazione precedente diviene:

```
int v = new int[100];  
int i=0;  
do {  
v[i]=0;  
i+=1;  
} while (i<100);
```

oppure ancora in modo più compatto:

```
int v = new int[100];  
int i=0;  
do v[i++] = 0 while (i<100);
```

Etichette E' possibile etichettare delle istruzioni, come in codice assembler, questo è utile in combinazione con istruzioni di break e di continue:

etichetta: istruzione;

Istruzione di break L'istruzione break viene utilizzata per uscire da un blocco o da uno switch, quando si incontra un break si esce dal blocco più interno, mentre se si mette una etichetta, si esce fino a dove è dichiarata l'etichetta. Esempio:

```
{
while ( COND )
{
while (COND2)
{
if (QUALCOSA) break;
}
}
}
```

Si esce in questo caso dal while interno, mentre:

```
{
pippo: while ( COND )
{
while (COND2)
{
if (QUALCOSA) break pippo;
}
}
}
```

si esce fuori da tutti e due i while. **Istruzione di continue** In un ciclo salta tutte le istruzioni che lo seguono e va a valutare direttamente l'espressione booleana, ad esempio:

```
while (Condizione)
{
istruzioni;
if (ColpoDiScena) continue;
altre istruzioni;
};
```

Si entra nel ciclo , si eseguono istruzioni, se ColpoDiScena è true salta altre istruzioni e riva a valutare Condizione, se è ancora vera continua il ciclo eseguendo istruzioni , ecc..

Istruzione di return Provoca la conclusione dell'esecuzione del metodo in cui si trova e il ritorno al chiamante. Se il metodo è void basta scrivere return; , altrimenti deve contenere una espressione dello stesso tipo del metodo, ad esempio

```
int pippo()
{
return 10;
}
```

con l'istruzione return è possibile anche bloccare l'esecuzione di un costruttore o di un metodo static.

Cosa sono i Package di Java

I package sono delle collezioni di classi, racchiuse in una collezione che le accomuna. Sono in pratica delle librerie a cui l'utente può accedere e che offrono varie funzionalità.

I package possono anche essere creati dall'utente, ad esempio racchiudendovi tutte le classi che ha definito per svolgere alcune funzioni che poi userà in vari programmi, ma questo a noi non interessa, perché ci interesserà vedere i package sicuramente più interessanti definiti in Java.

E' questa la vera potenza odierna di Java, il numero di classi già definite che svolgono i più svariati compiti, ed è anche la parte che continua sempre di più a crescere e ad aggiornarsi con le nuove versioni di Java (JDK 1.3 ne ha ben 18Mb). I package di Java sono così tanti che ne descriverò per sommi capi alcuni in un intero capitolo (il capitolo 3) del corso, e ne vedremo più o meno in dettaglio un paio, ovvero quelli che ci serviranno per definire interfacce grafiche e applets. Ovviamente tutti i package del linguaggio sono descritti nella mastodontica guida in linea del JDK scaricabile insieme al pacchetto del compilatore (La JDK Documentation 1.3 beta è di 105 Megabyte zippata è di una trentina di Mega), la quale è anch'essa gratuita, e che se avete voglia di passare un paio d'ore a scaricarla vi consiglio di prenderla insieme al compilatore, comunque è a vostra disposizione anche in linea (c'è il link nella pagina www.javasoft.com o www.java.sun.com, è lo stesso, sono parenti stretti).

Il nucleo del linguaggio Java contiene solo le parole chiave per la costruzione delle classi, i commenti, i normali costrutti if, switch, while, do-while, for, etichette, break, continue e return (manca il goto), che vedremo in dettaglio nel prossimo capitolo, tutto il resto è implementato nei package del linguaggio, comprese le normali primitive di Input e di Output. In questo paragrafo vedremo la lista dei packages di Java e ne vedremo uno in particolare, quello che ci interessa vedere per scrivere la nostra prima applicazione Java ancor prima di avere visto i costrutti, ovvero quello in cui possiamo trovare le istruzioni di input e output.

La lista completa dei packages di Java 1.2.1 in ordine alfabetico è la seguente :

com.sun.image.codec.jpeg, com.sun.java.swing.plaf.windows, com.sun.java.swing.plaf.motif, java.applet, java.awt, java.awt.color, java.awt.datatransfer, java.awt.dnd, java.awt.event, java.awt.font, java.awt.geom, java.awt.im, java.awt.image, java.awt.image.renderable, java.awt.print, java.beans, java.beans.beancontext, java.io, java.lang, java.lang.ref, java.lang.reflect, java.math, java.net, java.rmi, java.rmi.activation, java.rmi.dgc, java.rmi.registry, java.rmi.server, java.security, java.security.acl, java.security.cert, java.security.interfaces, java.security.spec, java.sql, java.text, java.util, java.util.jar, java.util.zip, javax.accessibility, javax.swing, javax.swing.border, javax.swing.colorchooser, javax.swing.event, javax.swing.filechooser, javax.swing.plaf, javax.swing.plaf.basic, javax.swing.plaf.metal, javax.swing.plaf.multi, javax.swing.table, javax.swing.text, javax.swing.text.html, javax.swing.text.html.parser, javax.swing.text.rtf, javax.swing.tree, javax.swing.undo, org.omg.CORBA, org.omg.CORBA.DynAnyPackage, org.omg.CORBA.ORBPackage, org.omg.CORBA.portable, org.omg.CORBA.TypeCodePackage, org.omg.CosNaming, org.omg.CosNaming.NamingContextPackage, sun.tools.ttydebug, sunw.io, sunw.util.

Sembrano pochini in effetti, sembra quasi che io abbia detto un stupidaggine in riferimento alla potenza di Java, ma se pensate che solo il package java.io comprende 50 classi e 10 interfacce capite bene che quelli di sopra sono una collezione di classi corposa.

A questo punto vorremmo fare l'input e output da console, per fare questo dobbiamo usare il package `java.lang`.

Per usare un package in una nostra classe, prima della definizione della classe dobbiamo inserire l'istruzione `import`, ad esempio volendo usare il package `java.awt` dobbiamo inserire all'inizio del nostro file:

```
import java.awt.*;
```

La `*` sta ad indicare che vogliamo usarne tutte le classi, se invece vogliamo usarne solo una classe possiamo specificarlo, ad esempio, `import java.awt.Frame`; potremo usare solo la classe `Frame` dell'`awt`. Nel caso nostro, in cui vogliamo fare una operazione di output, dovremmo dichiarare all'inizio `import java.lang.*`;

oppure, sapendo che la classe del package `java.lang` che contiene i metodi per fare questo è `System` potremmo scrivere

```
import java.lang.System;
```

La classe `System` a sua volta avrà al suo interno una `import java.io` (che è il package per l'input e l'output) per accedere alle classi dell'input e dell'output, e le userà per mandare quello che vogliamo su schermo.

Abbiamo visto un esempio di package che al suo interno invoca un altro package per fargli svolgere dei compiti, in queste librerie Java accade spesso questo, ed è proprio questo fenomeno che a molti fa apparire Java un linguaggio ostico, però superato questo blocco psicologico di fronte a quest'aspetto del linguaggio, la programmazione diviene semplice ed immediata.

Ho introdotto il package `java.lang`, vi dico anche che questo è il più importante di Java, in questo sono racchiuse le classi fondamentali del linguaggio, tanto che non serve dichiarare l'`import`, perché `java` lo importa automaticamente.

Inoltre vorrei soffermarmi su un aspetto fondamentale dell'importare i package, se io importo nel mio file il package `java.lang`, anche se esso importerà il package `java.io`, io dal mio file non potrò usare le classi di `java.io`, per farlo devo importarlo esplicitamente. Questo succede anche se programmo una applicazione in più file (con più classi), in ogni file devo importare i package che mi occorrono per la classe che sto definendo, non basta importarli in una sola. Vediamo le classi che contiene quindi questo package `java.lang` tanto importante per il linguaggio:

Boolean

abbiamo detto che i tipi di dato possono essere anche rappresentati da oggetti detti;

contenitori

questa è la classe che genera i contenitori per i tipi di dato booleani (vero, falso).

Byte

questa è la classe che genera i contenitori per i tipi di dato interi corti (di un byte).

Character

questa è la classe che genera i contenitori per i tipi di dato caratteri.

Character.Subset

questa è la classe che genera i contenitori per i tipi di dato caratteri con codifica unicode.

Character.UnicodeBlock

questa è la classe che genera i contenitori per i tipi di dato caratteri con codifica unicode 2.0.

Class

rappresenta classi e interfacce a runtime (in esecuzione), in Java è possibile a tempo di esecuzione creare, mandare in esecuzione e compilare classi, il compilatore e l'esecutore sono mischiati in modo strano, ma a noi non serviranno queste classi "particolari", perché scriveremo delle applicazioni tranquille.

ClassLoader

caricatore di classi a tempo di esecuzione.

Compiler

compilatore di classi a tempo di esecuzione.

Double

questa è la classe che genera i contenitori per i tipi di dato reali in virgola mobile di 64 bit.

Float

questa è la classe che genera i contenitori per i tipi di dato reali di 32 bit in virgola mobile.

Integer

questa è la classe che genera i contenitori per i tipi di dato interi.

Long

questa è la classe che genera i contenitori per i tipi di dato interi doppi.

Math

contiene metodi che emulano funzioni matematiche.

Number

classe di oggetti contenitori numeri generici.

Object

questa è la classe da cui derivano tutte le classi del linguaggio Java, ovvero ogni classe è sottoclasse (non per forza diretta) di questa.

Package

contiene metodi per estrapolare informazioni sui package di Java.

Process

Java è un linguaggio che permette di gestire Thread, ovvero programmini che girano in Parallelo, questa classe si occupa di loro, come le altre classi di java.lang: Runtime, RuntimePermission, Thread, ThreadGroup, ThreadLocal, Throwable

SecurityManager

per la sicurezza

Short

questa è la classe che genera i contenitori per i tipi di dato interi corti.

String

questa è la classe che genera i contenitori per i tipi di dato stringhe di caratteri.

StringBuffer

questa è la classe che genera i contenitori per i tipi di dato stringhe di caratteri modificabili.

System

è la classe che interagisce con il sistema, essa contiene molti metodi utili, e dei riferimenti ad altre classi (detti class field), è quella che useremo ad esempio noi per creare un output su schermo a caratteri.

Void

questa è la classe che genera i contenitori per i tipi di dato void, ovvero senza tipo. Questo sembra un tipo di dato inutile, ma vedremo che non lo è affatto, anzi è utilissimo, infatti è utile quando si vuole definire un metodo-procedura (un metodo che non ha valore di ritorno), in Java esistono solo funzioni, le quali sono obbligate a ritornare un valore del tipo che dichiarano. Dichiarando un metodo come void, si emula la procedura, ovvero non vi è la necessità del return.