Domenico Valle

Appunti JAVA

testo di riferimento:
Cay S. Horstmann – Concetti di informatica e fondamenti di Java 2 – seconda edizione Apogeo [2002]

Indice generale

CAPITOLO 1 – Elementi di programmazione....4
1. Algoritmo....4
2. Analisi....4
3. Programmazione....5
4. Costanti e variabili....5
5. Simboli utilizzati nei diagrammi a blocchi....6

CAPITOLO 2 – Il linguaggio Java....7
1. Introduzione....7
2. Perché Java? 7
3. Anatomia di un programma Java....7
4. Il processo di compilazione/esecuzione di un programma Java....8

CAPITOLO 3 – Classi e oggetti....9
1. Classe e oggetto....9
2. Variabile Oggetto....9
3. Definire una classe....10
4. Collaudare una classe....10
5. Attributi (variabili istanza)....10
6. Costruttori....10
7. Interfaccia pubblica di una classe....10
8. Commenti - Commentare l'interfaccia pubblica (javadoc)....10
9. Progettare e realizzare una classe....11
10. Tipi di variabili....11
11. Parametri espliciti e impliciti nei metodi....11

CAPITOLO 4 - Dati....12
1. Tipi di dati....12
2. Assegnazioni....13
3. Costanti....13
4. Conversione dei tipi di dati....13
5. Formattazione di numeri....14
6. Leggere dati in ingresso....15
7. Leggere dati in ingresso da console....16

CAPITOLO 5 – Selezione 17
1. Selezione....17
2. Selezione multipla....17

CAPITOLO 6 - Iterazione....18
1. Iterazione....18
2. Enunciati break e continue....18
3. Suggerimenti per controllare i cicli....18
4. Consigli pratici per realizzare cicli....19

CAPITOLO 7 – Classi, metodi, variabili....20
1. Scegliere le classi....20

2. Accoppiamento di classi....20
3. Metodi accessori e metodi modificatori....20
4. Effetti collaterali di un metodo....20
5. Chiamata per valore e chiamata per riferimento dei metodi....20
6. Metodi statici....20
7. Variabili statiche....20
8. Visibilità....21
9. Pacchetti (package)....21

CAPITOLO 8 – Interfacce, ereditarietà, errori....23
1. Interfacce....23
2. Ereditarietà....24
3. Gestione degli errori di esecuzione....25

CAPITOLO 9 – Vettori e Array....28
1. Introduzione....28
2. Array....28
3. Array a due dimensioni (matrici)....30
4. Passaggio di un vettore a un metodo e ritorno di un vettore da un metodo....30
5. Tabelle....32

CAPITOLO 10 - File....34
1. Flussi, lettori e scrittori....34
2. Leggere e scrivere file di testo....34
3. Flussi di oggetti....35

CAPITOLO 1 – Elementi di programmazione

1. Algoritmo

Per risolvere un problema utilizzando un computer occorre seguire una ben precisa sequenza di operazioni che possiamo raggruppare sotto le due fasi:

- **Analisi** : definizione del problema e stesura dell'algoritmo;
- **Programmazione** : scrittura del programma;

Scopo dell'**analisi** è quello di definire un **algoritmo**: successione di istruzioni che definiscono le operazioni da eseguire su dei dati per ottenere dei risultati (la parola algoritmo deriva dal nome di un matematico arabo, Muhammad ibn Mūsā al-Khwārizmī, vissuto nel nono secolo D.C.). Possiamo dire anche che un algoritmo è un elenco di istruzioni che permette di risolvere una classe di problemi; non è detto però che ogni elenco di istruzioni sia un algoritmo: esistono dei precisi requisiti che devono essere soddisfatti affinché un elenco di istruzioni possa essere considerato un algoritmo; essi sono :

a) **Finitezza** : deve essere composto da un numero finito di passi e richiedere una quantità finita di dati in ingresso.

b) **Terminazione** : deve avere termine dopo un tempo finito.

c) **Generalità** : deve valere per tutti i problemi di una determinata classe.

d) **Determinismo** : una volta fatto un passo, quello successivo può essere individuato dall'esecutore in maniera univoca, anche se ci sono alternative.

e) **Non ambiguità** : qualunque sia l'esecutore ogni istruzione deve essere interpretata sempre allo stesso modo.

Scopo della **programmazione** è quello di definire un **programma**: descrizione dell'algoritmo in una forma comprensibile e quindi eseguibile da parte dell'elaboratore.

2. Analisi

Definizione del problema

In questa fase è necessario definire, in modo il più completo possibile, alcune specifiche relative al problema che si vuole risolvere :

- Definizione delle uscite (output);
- Definizione degli input necessari al programma per ottenere gli output desiderati;
- Definizione di come gli output devono essere generati dagli input (funzione di trasferimento se si vuol vedere il programma come un sistema, algoritmo in termini prettamente informatici).

Stesura dell'algoritmo : diagramma a blocchi

Vi sono diversi modi per schematizzare l'algoritmo formulato in fase di definizione del problema. Uno dei più utilizzati è il diagramma a blocchi.

Il diagramma a blocchi o flow-chart è una rappresentazione grafica dell'algoritmo; esso indica il flusso (cioè la sequenza) delle operazioni da eseguire per realizzare la trasformazione, descritta nell'algoritmo, dei dati iniziali per ottenere i risultati finali.
Un particolare simbolo grafico, detto **blocco elementare** o più semplicemente **blocco,** è associato ad ogni tipo di operazione. I blocchi sono collegati tra loro da linee munite di frecce che indicano la sequenza delle azioni.
Un diagramma a blocchi è un insieme di blocchi elementari costituito da:
- un blocco iniziale
- un blocco finale
- un numero finito n (n>=1) di blocchi di azione e/o blocchi di lettura/scrittura
- un numero finito m (m>=0) di blocchi di controllo.

Un **algoritmo** (e relativo diagramma a blocchi) è **strutturato** se può essere visto come una composizione delle sole strutture fondamentali :
- **sequenza**
- **selezione**
- **iterazione**

(Teorema di Iacopini-Bohm : qualsiasi diagramma a blocchi può essere trasformato in un altro che esegue la stessa funzione e che utilizza soltanto le strutture fondamentali).

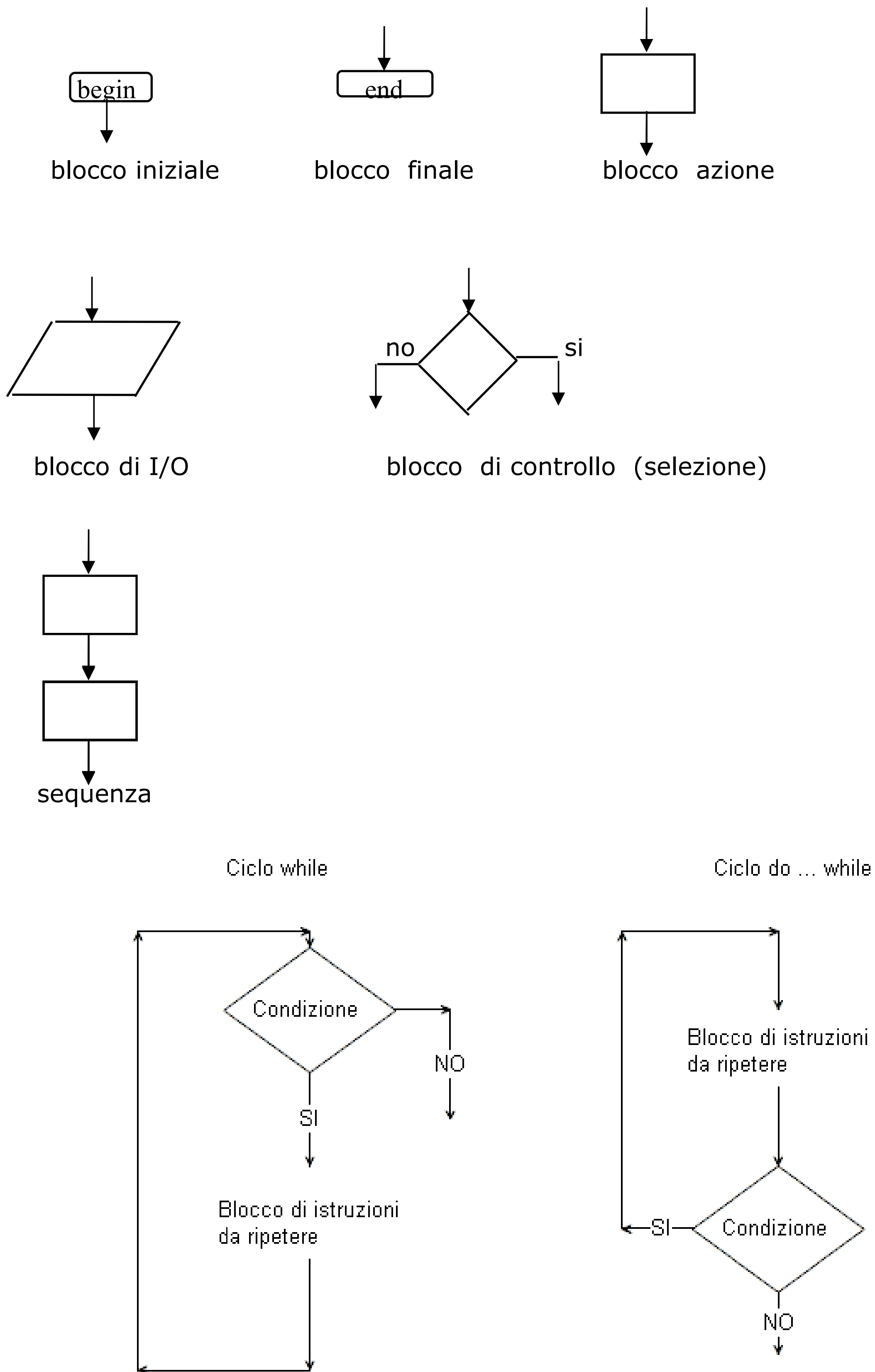
3. Programmazione

Definito l'algoritmo, esso deve essere messo in una forma che l'elaboratore (che è l'esecutore delle nostre istruzioni) possa capire per poterlo eseguire: ciò viene fatto descrivendo le azioni individuate nell'algoritmo tramite un linguaggio di programmazione, che pertanto può essere definito come un formalismo per descrivere all'elaboratore un algoritmo.
Un certo tempo, percentualmente non indifferente, della fase di programmazione deve essere dedicato al cosiddetto **"debugging"**, cioè alla prova del programma.

4. Costanti e variabili

Esistono due tipi di dati che possono essere usati in un programma : le costanti e le variabili.
La differenza è costituita dal fatto che mentre una costante mantiene lo stesso valore per l'intera durata del programma, una variabile può cambiare il suo contenuto durante l'elaborazione, ad esempio per effetto di un calcolo.

5. Simboli utilizzati nei diagrammi a blocchi



CAPITOLO 2 – Il linguaggio Java

1. Introduzione

Java viene creato nel 1991 in Sun Microsystem, da un gruppo di ricerca chiamato "Green project", come linguaggio per controllare apparecchiature elettroniche di largo consumo (televisione via cavo, computer tascabili); il primo linguaggio si chiamava Oak (quercia) ed è indipendente dalle varie piattaforme su cui può essere utilizzato.
Nel 1995 il nome venne cambiato in Java e i ricercatori capirono che il loro linguaggio, indipendente dalla piattaforma, era ideale per il Web. Le pagine Web che fino ad allora includevano solo testo e immagini fisse, potevano essere rese interattive includendo programmi Java di piccole dimensioni (applet).

2. Perché Java?

- è orientato agli oggetti : oggi è il paradigma dominante nella progettazione del software;
- è sicuro : rispetto al C/C++ gli errori fatti dal programmatore vengono segnalati piuttosto che ignorati e di conseguenza si evitano comportamenti misteriosi e oscuri del programma (pensiamo al supero della capacità in un vettore, uso dei puntatori);
- è semplice se paragonato al C++; forse la difficoltà maggiore sta nell'adattarsi all'utilizzo del paradigma orientato agli oggetti soprattutto per chi arriva dalla programmazione procedurale;
- è nato per il Web.

Difetti come linguaggio per i principianti

- non è stato curato per scrivere programmi elementari (ad es. non sono di immmediata comprensione le "istruzioni" per l'input dei dati)
- per scrivere un programma anche semplice bisogna comunque ricorrere a librerie di programmi che anche i programmatori più esperti non conoscono. compiutamente

3. Anatomia di un programma Java

- Disposizione a formato libero : non ci sono regole sugli spazi e le interruzioni di riga (si possono scrivere più istruzioni su una riga come si può scrivere ogni parola/simbolo di una istruzione su una riga diversa);
- Java distingue fra maiuscolo e minuscolo;
- Ogni programma Java si compone di una o più classi;
- Ogni classe può contenere delle definizioni di metodi;
- Ogni metodo contiene una sequenza di istruzioni;
- Un programma Java può essere un applet o una applicazione;
- Un'applicazione contiene una classe che ha il metodo main;

File Hello.java

```
public class  Hello
        {
                public static void main(String[] args)
                        {
                                System.out.println("Hello");
                        }
```

}

Ogni file sorgente conterrà al massimo una classe pubblica il cui nome deve corrispondere al nome del file che contiene la classe

Spiegazione delle istruzioni:

public	pubblico
String[] args	argomenti della riga di comando
static	il metodo *main* non agisce su un oggetto; deve essere sempre *static* perché la sua esecuzione inizia prima che il programma possa creare oggetti

Commenti: *//* (solo per commenti limitati a una riga) e *⁄* .... */*

Istruzioni(enunciati): ogni istruzione termina con un *;* un blocco di istruzioni deve essere racchiuso tra una coppia di parentesi graffe:*{* …..... *}.*

System.out.println:

- *System:* classe che contiene oggetti e metodi per accedere alle risorse del sistema;
- *out:* oggetto che identifica la periferica standard d'uscita (finestra terminale);
- *println:* metodo della classe *PrintStream.*

Stringa: sequenza di caratteri racchiusa tra virgolette (")
Sequenza di escape: \ seguita da altri simboli (la barra rovesciata non rappresenta se stessa ma viene utilizzata per codificare altri caratteri difficilmente codificabili in altro modo); ad es.:

\"	per rappresentare le virgolette (")
\n	per indicare a capo
\u00E9	carattere corrispondente al codice unicode 00E9 (é)
\\	per rappresentare proprio la \

4. Il processo di compilazione/esecuzione di un programma Java

Il compilatore Java (*javac*) traduce il codice sorgente in istruzioni per la macchina virtuale Java (**JVM**) (bytecode)
L'interprete Java (*java*) esegue il programma caricando il bytecode e i file di libreria necessari

Codice sorgente (.java) → COMPILATORE (javac) → File di bytecode (.class)

File di bytecode (.class) + Librerie → INTERPRETE(java) → programma in esecuzione

Il programma compilato (in bytecode) viene eseguito non dal computer reale ma da una macchibna virtuale: la **JVM** (Java Virtual Machine); questo rende un programma Java indipendente dalla piattaforma (il file *.class* è eseguibile su qualsiasi computer).
JRE (Java Run time Environment: ambiente Java al tempo di esecuzione) realizza la JVM per una particolare piattaforma.

CAPITOLO 3 – Classi e oggetti

1. Classe e oggetto

Un'istanza(oggetto): è un'entità concreta che esiste nel tempo (viene costruita e poi distrutta) e nello spazio (occupa memoria).
Una classe: è un'astrazione che rappresenta le proprietà comuni (struttura e comportamento) di un insieme di oggetti concreti (istanze): descrive il comportamento degli oggetti che da essa si possono creare.
Le classi sono fabbriche di oggetti.
Una classe può essere definita, in modo più stringato, come un insieme di metodi e dati.

Nei programmi non si manipolano soltanto numeri e stringhe ma anche dati più complessi (oggetti).

Interfaccia pubblica: specifica cosa si può fare con gli oggetti; in concreto è un elenco dei metodi (ed eventualmente degli attributi) che possono essere chiamati.

Implementazione nascosta: codice e dati che sono necessari per far funzionare i metodi; nella implementazione nascosta sono descritte come si svolgono le azioni indicate dai metodi.

Operatore new : creazione di un oggetto; per creare un nuovo oggetto

- usare l'operatore new
- dare il nome della classe
- fornire i parametri di costruzione (se richiesti)

2. Variabile Oggetto

Nelle variabili oggetto vengono memorizzati gli indirizzi degli oggetti (posizione dell'oggetto in memoria); la variabile oggetto contiene il riferimento a un oggetto di una determinata classe.

Rectangle a; variabile oggetto della classe Rectangle (di tipo Rectangle) non inizializzata: non fa riferimento ad alcun oggetto.

Rectangle a = new Rectangle(0,0,10,10); variabile oggetto inizializzata che contiene il riferimento all'oggetto (posizione dell'oggetto in memoria).

I programmi usano gli oggetti in questo modo :

- costruiscono un oggetto con l'operatore *new;*
- memorizzano il riferimento all'oggetto in una variabile oggetto;
- invocano metodi tramite la variabile oggetto;

package (pacchetto) : raccolta di classi che hanno finalità simili;
Nei programmi, tranne java.lang, bisogna importare tutti gli altri package.

3. Definire una classe

Una classe contiene

- gli attributi dei suoi oggetti
- la definizione dei metodi che si possono applicare ai suoi oggetti.

Definire un metodo

1) specificatore di accesso (*public, private, protected*)
2) tipo del dato restituito dal metodo
3) nome del metodo
4) elenco dei parametri del metodo racchiusi tra parentesi (tipoParametro1 nomeParametro1,tipoParametro2 nomeParametro2,....)
5) corpo del metodo : blocco di istruzioni racchiuse tra parentesi graffe.

4. Collaudare una classe

Scrivere un'applicazione contenente il metodo *main* che utilizzi oggetti della classe che si vuole collaudare.

5. Attributi (variabili istanza)

Le variabili istanza, o attributi, memorizzano lo stato di un oggetto; sono i dati di cui l'oggetto ha bisogno per eseguire i suoi metodi; ogni oggetto ha le sue variabili istanza.

Per dichiarare una variabile istanza è necessario indicare

- lo specificatore di accesso (*private, public, protected*: di solito *private*)
- il tipo
- il nome

Incapsulamento: processo che nasconde i dati di un oggetto; è possibile accedere a questi dati tramite metodi

6. Costruttori

I costruttori contengono istruzioni per inizializzare gli oggetti.

Il nome del costruttore è sempre uguale al nome della classe.

I costruttori possono essere richiamati solo in combinazione con l'operatore *new*; essi non sono metodi e quindi l' istruzione <u>nomeoggetto.costruttore</u> è errata.

7. Interfaccia pubblica di una classe

Costruttori e metodi della classe: per definirli è necessario capire come si comporteranno gli oggetti della classe e come vogliamo costruire un oggetto

Sovraccarico (overloaded) : lo stesso nome viene utilizzato per più di un costruttore o metodo (cambia il tipo/numero dei parametri); non conta il tipo del valore restituito.

8. Commenti - Commentare l'interfaccia pubblica (javadoc)

Un commento che si estende su più righe deve essere racchiuso tra

```
/*

*/
```

Nell'ambito di una riga tutto ciò che segue // (doppia barra) è considerato commento.

I commenti possono esere utilizzati per generare la documentazione: un commento di documentazione è racchiuso tra

```
/**
```

*/
la prima frase (sequenza di parole che termina con il . (punto)) viene copiata in una tabella riassuntiva perciò è opportuno che sia una frase di senso compiuto.
Commentare ogni classe, ogni metodo, ogni parametro e ogni valore di ritorno; in particolare nei metodi commentare ogni parametro di ingresso:

@param nome_parametro   commento

e il valore di ritorno

@return nome_valore_ritorno   commento

Scrivere il commento prima di scrivere il codice consente di verificare la reale comprensione di ciò che si sta programmando.

9. Progettare e realizzare una classe

Per progettare e realizzare una classe è opportuno seguire il seguente percorso :

1) capire cosa si deve fare con un oggetto della classe
2) assegnare i nomi ai metodi
3) scrivere la documentazione per l'interfaccia pubblica
4) identificare le variabili istanza
5) identificare i costruttori
6) realizzare i metodi
7) collaudare la classe

10. Tipi di variabili

1) variabili istanza
2) variabili locali
3) variabili parametro

tempo di vita : le variabili istanza appartengono a un oggetto quindi hanno la stessa vita dell'oggetto (rimangono in vita finché qualche metodo usa l'oggetto) invece le variabili locali e le variabili parametro appartengono al metodo: prendono vita quando il metodo viene richiamato, muoiono quando il metodo termina.

inizializzazione : le variabili istanza se non vengono inizializzate esplicitamente in un costruttore sono inizializzate con un valore predefinito (i numeri a 0, gli oggetti a *null*, i boolean a *false*); le variabili locali devono essere inizializzate esplicitamente; le variabili parametro sono inizializzate con i valori passati al metodo al momento del suo richiamo.

Errore : dimenticarsi di inizializzare le variabili ad es.

```
Rectangle a;
a.translate(15,25); // a non è stata inizializzata quindi non può essere
                    // nemmeno spostata.
```

11. Parametri espliciti e impliciti nei metodi

Parametri espliciti sono i parametri che compaiono nella testata del metodo.
Parametro implicito: oggetto che ha invocato il metodo: vine indicato con la parola *this;* quando un metodo usa una variabile istanza questa variabile istanza si riferisce all'oggetto che ha invocato il metodo (il parametro implicito appunto).

CAPITOLO 4 - Dati

1. Tipi di dati

Tipo di dato	Byte	Intervallo
byte	1	-128 a +127
short	2	-32768 +32767
int	4	-2.147.483.648 +2.147.483.647 (2 miliardi)
long	8	9 * 10E18
char	2	Numeri interi senza segno a 16 bit rappresentanti i simboli del codice alfanumerico Unicode
float	4	+- 3.40 * 10E-39 +- 3.40 *10e+38 (7 cifre decimali significative)
double	8	+- 1.79 * 10E-309 +- 1.79 *10e+308 (15 cifre decimali significative)
boolean		True e false

Classe String
String x, y;
int a;
char c;
Le posizioni dei caratteri in una stringa sono numerate a partire da 0

Metodi per manipolare le stringhe

Metodo	Significato
a = x.length()	a contiene la lunghezza della stringa x
y = x.substring(inizio, fine)	y contiene la sottostringa di x, estratta a partire dal carattere di posizione 'inizio' fino al carattere di posizione 'fine'-1
c = x.charAt(i)	c contiene il carattere della stringa x nella posizione i
x.equals(y)	Confronta le 2 stringhe x e y e restituisce *true* se le due stringhe sono uguali *false* nel caso contrario
x.equalsIgnoreCase(y)	Come sopra ignorando maiuscolo/minuscolo
y = x.toUpperCase() y = x.toLowerCase()	y contiene la stringa x convertita in maiuscolo (*toUpperCase*) o minuscolo (*toLowerCase)*
x.indexOf (carattere| sottostringa) x.lastIndexOf (carattere|sottostringa)	Ricerca, la prima o l'ultima (last) occorrenza del carattere o sottostringa
x.replace(cCercato,cSostituito)	Sostituisce tutti i caratteri ‘cCercato’ con 'cSostituito’
x.trim()	Elimina tutti gli spazi iniziali e finali in una stringa

Classe Math

Metodo	**Funzione**
Math.pow(x,y)	x elevato y
Math.sqrt(x)	Radice quadrata di x
Math.random()	Genera un numero (pseudo) casuale >=0 e <1
Math.ceil(x)	Numero intero più piccolo >= x (di tipo double)
Math.floor(x)	Numero intero più grande <= x (di tipo double)
Math.round(x)	Trasforma il numero reale x in un intero troncando la parte frazionaria dopo averla arrotondata (restituisce un intero long)
Math.PI	Pigreco (3.14....)
Math.E	Numero di Nepero (base dei logaritmi naturali : 2.718...)

Ovviamente ci sono anche i metodi per le funzioni logaritmo e trigonometriche. Normalmente gli argomenti e i valori restituiti sono in virgola mobile (float o double)

Esempio arrotondamento:

Per x = 3.141 Math.ceil(x) valore restituito 4.0
Math.floor(x) valore restituito 3.0
Math.round(x) valore restituito 3

2. Assegnazioni

a = 10;
a += 10; significa *a = a+10;*
a++; significa *a = a+1;*
a--; significa *a= a-1;*

3. Costanti

Sono identificate con la parola chiave *final*.
Una costante non può mai essere modificata una volta che le è stato assegnato un valore.
Per convenzione il nome di una costante è scritto tutto in maiuscolo.

Es.
final double EURO = 1936,27;
Oppure
final double EURO*;*
....
EURO = 1936,27; // solo una volta si può assegnare il valore
// a una costante;
// si può fare l'assegnazione anche da input.

Normalmente le costanti sono utilizzate all'interno di un metodo; per poterle utilizzare in più metodi, dichiararle, al di fuori ogni metodo, come *static*

private static final double EURO = 1936.27;
oppure
public static final double EURO = 1936.27;

4. Conversione dei tipi di dati

E’ ammesso assegnare un *int* a un *double* ma non il contrario.
La divisione fra numeri interi dà sempre un numero intero: per avere un numero decimale almeno uno degli operandi deve essere o deve essere forzato (*cast*) a decimale.

Il *cast* forza il tipo di una variabile.
es. :

```
double prezzo = 23.78;
int lire = (int)prezzo;   // la variabile prezzo, in questa assegnazione,
                          //viene forzata a int ( cast di prezzo a int) e
                          //quindi il valore contenuto in prezzo viene
                          //assegnato alla variabile lire.Forzando prezzo a
                          // int vengono persi i decimali: il valore assegnato
                          //a lire è 23 non 23.78.
```

5. Formattazione di numeri

Per formattare i numeri si può usare la classe *NumberFormat* del pacchetto *java.text*

1) creare un oggetto di tipo *NumberFormat* tramite il metoto statico *getNumberInstance()* (metodo statico è un metodo di classe cioè un metodo non richiamato da un oggetto ma direttamente indicando la classe in cui è stato definito;l' esempio più comune sono i metodi della classe *Math*).

   NumberFormat formatter = *NumberFormat.getNumberInstance();*

   per stampare valori in valuta invece di *getNumberInstance()* usare *getCurrencyInstance()*

2) Impostare il numero massimo delle cifre decimali (ad es. a 2)

   formatter.*setMaximumFractionDigits(*2*);*

3) Impostare il numero minimo delle cifre decimali: permette di stampare gli eventuali 0 dopo il punto decimale

   formatter.*setMinimumFractionDigits(*2*);*

   0.3 viene stampato come 0.30 e non 0.3

4) usare metodo format per avere il valore formattato

   formatter.*format(*valore_da_formattare*);*

Esempio di formattazione

```
public static void main(String[] args)
{
        double valore1 = 123456.12;
        double valore2 = 7890123.10;
        double valuta = 456789.78;

        NumberFormat formattatoreNumero = NumberFormat.getNumberInstance();
        NumberFormat formattatoreValuta = NumberFormat.getCurrencyInstance();

        formattatoreNumero.setMaximumFractionDigits(3);
        formattatoreNumero.setMinimumFractionDigits(3);

        formattatoreValuta.setMaximumFractionDigits(2);
        formattatoreValuta.setMinimumFractionDigits(2);

      System.out.println("numeri formattati: " + formattatoreNumero.format(valore1)
                          +" -  " +formattatoreNumero.format(valore2));
    System.out.println("valuta formattata: " + formattatoreValuta.format(valuta));
}
```

il risultato è

```
        numeri formattati: 123.456,120 -  7.890.123,100
        valuta formattata: € 456.789,78
```

6. Leggere dati in ingresso

La classe *JOptionPane* del pacchetto *javax.swing* permette tramite il metodo statico *showInputDialog*, in una finestra grafica, l'input di dati di tipo stringa

JOptionPane.showInputDialog("   ")

Nel caso si debbano inserire dei numeri, occorre trasformare la stringa di input utilizzando i metodi

Byte.parseByte(...)
Short.parseShort(...)
Integer.parseInt(...)
Long.parseLong(...)
Float.parseFloat(...)
Double.parseDouble(...)

Per terminare un programma che ha un'interfaccia utente grafica bisogna inserire alla fine del metodo *main*

System.exit(0);

Il metodo *showInputDialog* provoca l'avvio di un thread (flusso di esecuzione) dell'interfaccia utente per la gestione dei dati in ingresso; quando il metodo main termina il thread è ancora in esecuzione e il programma non termina in modo automatico perciò bisogna forzare la terminazione inserendo *System.exit(0);*

7. Leggere dati in ingresso da console

Fare l' import di *java.io.*;*

I dati in ingresso vengono letti dall'oggetto *System.in* che, a differenza di *System.out* che ha la capacità di scrivere numeri e stringhe, può leggere solo byte.

Per ottenere un lettore di caratteri bisogna trasformare *System.in* in un oggetto di tipo *InputStreamReader*

InputStreamReader a = *new InputStreamReader(System.in);*

ma per leggere un'intera stringa per volta bisogna trasformare l'oggetto *InputStreamReader* in un oggetto *BufferedReader*

BufferedReader c = *new BufferedReader(a);*

oppure direttamente

BufferedReader a = *new BufferedReader(new InputStreamReader(System.in));*

ora si può usare il metodo *readLine()* per leggere un' intera riga.

Il metodo *readLine*() può generare un'eccezione (*IOException*) a controllo obbligatorio: in questo caso è necessario fare una delle seguenti azioni

1) gestire l'eccezione
2) annunciare che non avete intenzione di gestire l'eccezione

Public static void main(String[] args) ***throws IOException***

CAPITOLO 5 – Selezione

1. Selezione

```
if (condizione)
        {
          ............
          istruzioni del ramo SI
          .............
        }
else
        {
         ............
          eventuali istruzioni del ramo NO
          .............
         }
```

2. Selezione multipla

```
switch  variabile_intera
        {
                case costante1 :
                        ...........;
                break;

                case costante2 :
                        ...........;
                break;

                case costante3 :
                        ...........;
                break;

                case costante4 :
                        ...........;
                break;

                default :
                        ...........  ;
        }
```

CAPITOLO 6 - Iterazione

1. Iterazione

- do.......while

```
do
{
        ...........
        istruzioni da ripetere

}  while (condizione );
```

- while

```
while( condizione )
{
        ...........
        istruzioni da ripetere
}
```

- for

```
for (valore_iniziale_indice;condizione;variazione_indice)
{
        ...........
        istruzioni da ripetere
}
```

2. Enunciati *break* e *continue*

break può essere usato per uscire

- da un *case* del blocco *switch* per andare alla prima istruzione che segue lo *switch*
- per uscire da un ciclo *while*, *for*, *do .... while*

è possibile usare l'enunciato *break* etichetta1, per far proseguire il programma a partire dall'istruzione con etichetta uguale a etichetta1.
E' possibile etichettare qualsiasi istruzione:

etichetta1: istruzione;

continue viene usato all'interno di un' iterazione e fa proseguire il programma dalla fine dell'iterazione attuale, passa, cioè, all'iterazione successiva.

3. Suggerimenti per controllare i cicli

- limiti simmetrici:operatore di confronto <= in entrambi i lati: 1<= i <= n)
- limiti asimmetrici:l'operatore <= è presente solo da una parte: 0<= i < n)

contare le iterazioni :

- for (i=a; i<b ; i++) : il ciclo viene eseguito (b-a) volte
- for (i=0; i<=10 ; i++) : il ciclo viene eseguito 11 volte : rispetto alla situazione precedente abbiamo <= nella condizione: bisogna contare, in

questo caso, anche l'ultima iterazione: per scoprire che i cicli eseguiti sono 11 (potrebbe essere una situazione di errore se si vogliono eseguire solo 10 cicli); per capire ciò potrebbe essere utile ricorrere all'esempio dei pali (|) e delle sezioni (=) di una staccionata. Se la staccionata ha dieci sezioni quanti pali deve avere ? [errore del palo di staccionata]

|=|=|=|=|=|=|=|=|=|=|

ci sono 11 pali perché ciascuna sezione ha un palo sulla sinistra e c'è un palo in più dopo l'ultima; per poter avere solo 10 pali bisognerebbe escludere l'ultimo (per l'appunto < 10 e non <=10).

In ogni caso, indicando con c l'incremento (normalmente unitario), il conteggio dei cicli è

- (b – a) / c per cicli asimmetrici
- ((b – a) / c) +1 per cicli simmetrici ad es. for(i=10; i<=40;i+=5) verrà eseguito per ((40-10)/5) +1 pari a 7 volte

4. Consigli pratici per realizzare cicli

1) elencare passo per passo ciò che deve essere fatto nel corpo del ciclo;
2) determinare quante volte viene ripetuto il ciclo;
3) individuare la condizione di fine-ciclo;
4) scrivere gli enunciati del corpo del ciclo;
5) esaminare attentamente l'inizializzazione delle variabili;
6) controllare eventuali errori "per scarto di uno" : inizializzazione non corretta di alcune variabili ad es. una variabile che dovrebbe essere inizializzata a 0 viene inizializzata 1 : verificare con una simulazione.

CAPITOLO 7 – Classi, metodi, variabili

1. Scegliere le classi

Le classi non sono funzioni, le funzioni descrivono azioni come i metodi. Le classi invece sono raccolte di oggetti e gli oggetti non sono azioni ma entità.
In un problema per individuare gli oggetti e i metodi, in prima approssimazione, è bene tenere presente la seguente regola:

Oggetti: sostantivi
Metodi: verbi

2. Accoppiamento di classi

Accoppiamento: due classi sono accoppiate quando una dipende dall'altra cioè usa istanze di tale classe; l'accoppiamento dovrebbe essere ridotto al minimo.

3. Metodi accessori e metodi modificatori

- **metodi accessori** : non modificano lo stato di un oggetto;
- **modificatori** : modificano lo stato dell'oggetto;
- **classi immutabili** : hanno solo metodi accessori; ad es. la classe String: una volta costruita una stringa nessun metodo della classe String può modificare il suo contenuto.

4. Effetti collaterali di un metodo

Per effetto collaterale di un metodo si intende qualsiasi comportamento osservabile all'esterno del parametro implicito (oggetto medesimo).

- modifica di un altro oggetto
- visualizzazione di dati in uscita (System.out)

Gli effetti collaterali sono da evitare

5. Chiamata per valore e chiamata per riferimento dei metodi

Si ha sempre una chiamata per valore; le variabili di tipo primitivo (numeri e boolean) non possono essere modificate; per gli oggetti si può aggiornare lo stato di un oggetto (il valore degli attributi dell'oggetto) ma non si può modificare il contenuto del riferimento all' oggetto (la variabile oggetto passata).

6. Metodi statici

Sono metodi che possono essere richiamati senza far ricorso ad alcun oggetto

NomeClasse.metodoStatico

A differenza dei metodi di istanza, i metodi *static* possono

- richiamare soltanto altri metodi statici
- possono usare solo variabili locali e *static* ma non variabili di istanza

7. Variabili statiche

Sono dichiarate all'interno di una classe con la parola *static*

private static int a ;

esse non appartengono a nessuno oggetto bensì alla classe: a differenza delle variabili istanza non ne esiste una copia per ogni oggetto ma una sola copia.

Ogni metodo della classe può accedere alle variabili *static*
Inizializzazione delle variabili statiche

- no nel costruttore altrimenti verrebbero inizializzate ogni volta che si costruisce un nuovo oggetto.
- non fare nulla: viene inizializzata automaticamente (numero a 0, boolean a *false*, oggetto a *null*).
- usare un inizializzatore esplicito (*private static int* a = 0;)

8. Visibilità

- **Visibilità delle variabili locali**: blocco in cui è dichiarata; il blocco è delimitato dalle { }; caso tipico è la dichiarazione nella testata del for :

  for (int i =0 ; i< 10; i++) {…………..}

  la visibilità di *i* termina con la fine del ciclo

- **Visibilità di membri (attributi/campi e metodi) di classe :** all'interno di un metodo si ha l'accesso a tutti i campi e metodi della classe stessa. Al di fuori della classe bisogna usare il nome qualificato :

  per i metodi e campi *static* pubblici
  nome_classe.metodo
  nome_classe.campo

  per i metodi e campi istanza pubblici
  nome_oggetto.metodo
  nome_oggetto.campo

- **Visibilità sovrapposte** ad es. variabile locale e variabile istanza con lo stesso nome: prevale il nome della variabile locale che mette in ombra la variabile istanza. Ci si può riferire alla variabile istanza usando l'operatore *this*

9. Pacchetti (package)

Un pacchetto è un insieme di classi correlate. Tra i pacchetti più significativi della libreria java

java.lang	supporto al linguaggio
java.util	utility
java.io	Input/output
java.awt	Abstract Windowing Toolkit (interfaccia grafica old)
javax.swing	Interfaccia grafica (new)
java.applet	Applet
java.net	Connessione di rete
java.sql	Accesso a DB tramite sql

Per importare una classe in un programma

import nomePacchetto.nomeclasse;

oppure

import nomePacchetto.*; // tutte le classi del pacchetto

Non è necessario importare *java.lang.*

nomi di pacchetto: scegliere un nome univoco ad es. potrebbe essere il nome del dominio internet scritto all'incontrario oppure il nome della casella postale; le parole che compongono il nome devono essere separate da "**.**"

localizzazione delle classi: un pacchetto si trova in una sottocartella il cui nome corrisponde a quello del pacchetto; le singole parole separate da **.** rappresentano le cartelle annidate in successione; ad esempio il pacchetto

com.horstmann.bigjava

si troverà nella sottocartella

com\horstmann\bigjava

CAPITOLO 8 – Interfacce, ereditarietà, errori

1. Interfacce

Un' interfaccia è un tipo di dato dichiarato con il costrutto *interface* che deve essere interamente implementata. Il suo compito è quello di dichiarare un modello (template)

```
public  interface nome_interfaccia
        {
          public  static final tipoDato nome_costante = valore costante
          public abstract tipo dato restituito nomeMetodo (lista parametri)
         …................
        }
```

L' interfaccia serve per

1) stabilire similitudini tra tipi di dato differenti
2) dichiarare metodi che altri programmatori devono implementare
3) definire un insieme di costanti di utilità generale

Es. Cilindro, cono e sfera hanno in comune la costante **PIGRECO** e i metodi **superficie()** e **volume()** si può pertanto definire l'interfaccia

```
public interface  SolidoRotazione
        {
                public static final PIGRECO = 3.14;
                public abstract double superficie();
                public abstract double volume();
        }
```

e una delle classi che implementa l'interfaccia

```
public class Cilindro  implements SolidoRotazione
        {
           private double raggio, altezza;

           public Cilindro(raggio,altezza)
               {
                 this.raggio = raggio;
                 this.altezza = altezza;
               }

                public double superficie()
              {
                        return (2.0*PIGRECO*raggio*(altezza+raggio));
               }
```

```
                public double volume()
                {
                        return (PIGRECO*raggio *raggio*altezza);
                }
        }

public class Test_cilindro
        public static void main (..............)
        {
                Cilindro a = new Cilindro(2,2);
                System.out.println (a .superficie() + a.volume());
        }
```

Caratteristiche dell'interfaccia

1) tutti gli attributi sono delle costanti pubbliche statiche; non ci possono essere variabili di istanza;
2) tutti i imetodi sono pubblici e dichiarati con il modificatore abstract;
3) non possiede costruttori;
4) è possibile dichiarare variabili reference di un tipo interfaccia ma non creare oggetti.

metodo astratto : sottoprogramma che non contiene il blocco di istruzioni (cfr prototipo delle funzioni in C).

classe astratta : classe che contiene metodi normali e metodi astratti.

L'interfaccia va salvata in un file .java come una classe normale.

Nelle dichiarazioni, in una interfaccia, si possono evitare le parole chiave

public static final per gli attributi
public abstract per i metodi

La classe che implementa l'interfaccia :

public class nome_classe *implements* nome_interfaccia1, nome_interfaccia2,........

- può' accedere alle costanti dichiarate nel corpo dell'interfaccia
- deve implementare tutti i metodi dichiarati nell'interfaccia
- può avere suoi metodi e sue variabili di istanza

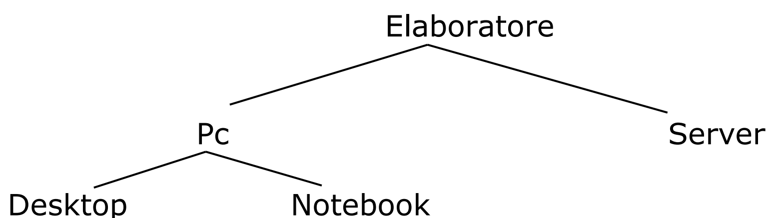
2. Ereditarietà

E' possibile solo implementare l'ereditarietà singola e non quella multipla: cioè una classe derivata (child, discendente, sottoclasse) può ereditare attributi e metodi solo da una classe genitrice (parent, superclasse) :

```
public class nome_sottoclasse extends nome_ superclasse
        {
           ................
        }
```

nell'ereditarietà multipla : una sottoclasse può ereditare da più di una superclasse

gerarchia di classi Es.



Accesso agli elementi di una classe : modificatori di accesso

Modificatore di accesso dell'elemento	Elemento visibile nella stessa classe	Elemento visibile nelle sottoclassi	Elemento visibile nelle classi esterne
.... public	Si	Si	Si
.... protected	Si	Si	No
.... private	Si	No	No

Sottoclasse e superclasse
La terminologia sopra/sotto deriva dalla teoria degli insiemi la superclasse definisce attributi e metodi comuni a tutti gli oggetti delle varie sottoclassi; la sottoclasse definisce un tipo di dati più specializzato rispetto all'insieme formato dagli oggetti della superclasse perciò

- è possibile assegnare un oggetto della sottoclasse a un reference della superclasse (è possibile assegnare un oggetto Desktop a una variabile reference Elaboratore)
- non è possibile il contrario (oggetto superclasse a reference sottoclasse)
- è possibile convertire un reference a oggetto di una sottoclasse in un reference a oggetto di una superclasse (semplice assegnazione) :

```
Desktop c = new Desktop();
Elaboratore e = new Elaboratore();
```

È possibile

```
e = c;
```

- il contrario si può facendo il cast

```
c = (Desktop)e;
```

3. Gestione degli errori di esecuzione

Quando si verifica un errore di runtime l'ambiente di esecuzione crea un oggetto eccezione; questo oggetto viene "lanciato" (throw) dal metodo che ha provocato il malfunzionamento; se l'eccezione non viene catturata il programma si interrompe e compaiono una serie di messaggi indicanti

- l'eccezione
- stack delle chiamate ai metodi del programma attivi prima che si verificasse l'eccezione; normalmente ai fini dell'individuazione del nostro metodo e della riga di programma che ha generato l'errore sono significative la prima e l'ultima riga dell'elenco

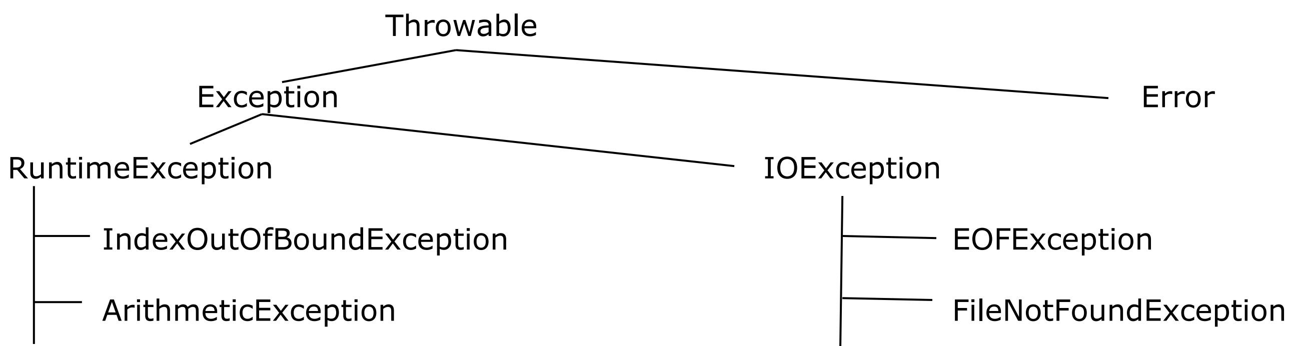
Per gestire gli errori si utilizza il costrutto

```
try
        {
                .........
                .........

        }

catch  (Classe_eccezione, oggetto_eccezione)
        {
                .........
        }
catch  (Classe_eccezione, oggetto_eccezione)
        {
                .........
        }
catch  (Classe_eccezione, oggetto_eccezione)
        {
                .........
        }
finally      // opzionale vengono sempre eseguite sia che si verifichi o non si verifichi  un'eccezione
        {
        .........
        }
```

un oggetto eccezione di una superclasse cattura anche le eccezioni di una sottoclasse perciò i *catch* delle superclassi dovrebbero essere posti in fondo; si deve tener conto della seguente gerarchia delle classi eccezioni



Rilancio di un'eccezione
Un' eccezione può essere rilanciata al metodo chiamante (e dal *main* direttamente all'interprete Java) tramite

- l'istruzione *throw*

throw new ClasseEccezione1();

- dichiarare, nell' intestazione del metodo, tramite la parola *throws,* le eccezioni rilanciate

  .... nomeMetodo(.............)*throws* ClasseEccezione1, ClasseEccezione2, ...

CAPITOLO 9 – Vettori e Array

1. Introduzione

Vettori (*ArrayList):* insieme di oggetti disposti in sequenza.
Array: sequenza di lunghezza prefissata di valori omogenei(valori dello stesso tipo: può essere un tipo primitivo(numero o boolean) o un classe).
Differenza tra vettori e array :

- un array ha lunghezza prefissata mentre un vettore inizia con lunghezza 0 e aumenta la sua dimensione man mano che viene aggiunto un nuovo elemento; diminuisce di dimensione quando un elemento viene tolto;
- gli elementi di un array sono di un tipo specifico mentre un vettore memorizza un insieme di riferimenti a Object.

I numeri non sono oggetti quindi non possono essere memorizzati in un *ArrayList* (vettore). Si potrebbero memorizzare in un vettore passando attraverso le classi involucro *Integer , Double, Boolean* che incorporano numeri e booleani in oggetti es.

```
ArrayList data = new ArrayList();
Double x = 29,89;
Double involucro = new Double(x);
Data.add(involucro);
```

2. Array

L'array è una sequenza prefissata di valori dello stesso tipo (primitivo o classe).

Es. costruzione di un array di dieci valori di tipo double

```
new double[10]
```

per memorizzare il riferimento all'array in una variabile

```
double[] a
```

è possibile dichiarare gli array anche nella forma tradizionale

```
double a[]
```

le due dichiarazioni si possono leggere come :

- *double[]* a   un array di double chiamato a
- *double* a*[]*   un array a di double

E' possibile mettere insieme dichiarazione variabile riferimento + creazione oggetto

```
double[] a = new double[10];
```

a :   oggetto array *double[]*

Nel momento in cui viene creato, l'array viene inizializzato a

- 0 per un array di numeri,
- false per array di boolean

- null per array di oggetti

per accedere agli elementi dell'array

a[n.elemento]

- le posizioni negli array sono numerate a partire da 0;
- il supero dei limiti viene segnalato come errore;
- la costante pubblica di istanza *length* (senza parentesi perché non è un metodo) indica le dimensioni dell'array.

Per inizializzare un array :

```
int[]  a = new int[5];
a[0] = 3;
a[1] = 15;
a[2] = 27;
a[3] = 32;
a[4] = 9;
```

oppure

```
int k[] = new int[] {0, 1, 2, 3, 4 };

int[]  a = { 3, 15, 8, 11, 56 };

int k = new int[5] {0,1,2,3,4};     // ------ errata  ----------
```

array anonimo : può essere utile quando si deve passare come parametro a un metodo che si aspetta un array

```
new int[]   { 15, 24, 52, 35};
```

Copiare array

Le variabili array funzionano esattamente come le variabili oggetto: contengono un riferimento all'array :

```
int[]  a = new int[10];
int[]  b = a;       // non è stato creato un nuovo oggetto ma semplicemente un nuovo riferimento
```

b non è una copia dell'array ma una copia del riferimento; sia a che B puntano allo stesso oggetto array di interi di 10 elementi

per copiare l'array dobbiamo creare un nuovo oggetto e fare la copia elemento per elemento

```
int[]  a = new int[10];
int[]  b = new int[10];
for (int i=0 ; i< a.length; i++ ) b[i]= a[i];
```

Esiste un metodo della classe *System* che permette di copiare degli elementi di un array in un altro

```
System.arraycopy (from, fromStart, to, toStart, count);
```

from: array di partenza
fromStart: posizione di partenza
to: array di destinazione (può coincidere con quello di partenza)
toStart: posizione di arrivo
count: n.elementi

Esempio :

*System.arraycopy (*data, i, data, i+1, data.length-i-1 *);*

sposta verso il basso, nel vettore data, tutti gli elementi a partire dalla posizione i

*System.arraycopy (*data, i+1, data, i, data.length-i-1 *);*

sposta verso l' alto, nel vettore data, tutti gli elementi a partire dalla posizione i

3. Array a due dimensioni (matrici)

int[][] a = *new int[*5*][*5*];*

per utilizzare *length* in una matrice

a.*length*	n.righe
a[i].*length*	n.colonne della riga i-esima

Non tutte le righe devono avere lo stesso numero di colonne.

4. Passaggio di un vettore a un metodo e ritorno di un vettore da un metodo

```
public class  Mainsort {
        public static void main {
            int[] a = new int[5];
            Vettore v = new Vettore();
            a = v.carica(a.length);
            v.stampa(a);
            v.ordina(a);
            v.stampa(a);
            }
      }

      /**
         la classe contiene dei metodi per gestire un array
         carica, stampa ordina
      */
public class  Vettore {
       /**
       Metodo per caricare un array
             @ param lunghezza dell'array
             @return x: array caricato in modo random
       */
        public int[] carica ( int lunghezza)
```

```
        {
           int x [] = new int[lunghezza];
           for (int i = 0; i< x.length; i++)
                x[i] = (int) (Math.random()*100);
           return x;
        }


 /**
        Metodo per ordinare un array
        @ param vettore da stampare
 */
public void ordina ( int [] x)
        {
           for (int i = 0; i< x.length-1; i++)
                 for (int j = (i+1); j< x.length-1; j++)
                        if ( x[i] > x[j])
                        {
                           int c = x[i]
                           x[i] =  x[j];
                           x[j] =  c;
                        }

        }

 /**
        Metodo per stampare un array
        @ param vettore da stampare
 */
public void stampa ( int[] x)
        {
           for (int i = 0; i< x.length; i++)
                System.out.println(x[i] );

        }

} // chiude la classe
```

La classe *Arrays* del package java.util fornisce dei metodi statici per lavorare con gli array (di numeri, caratteri, oggetti)

- *Arrays.sort (*varray*)*: ordina varray in ordine crescente;
- *Arrays.binarySearch(*varray, datoricercato*)*: restituisce la posizione del datoricercato; il vettore deve essere ordinato
- *Arrays.fill(*varray, valore*)* : inizializza il vettore con il valore indicato

Con *Arrays.sort* e *Arrays.binarySearch()* si possono ordinare e fare ricerche per oggetti di qualsiasi classe che implementi l'interfaccia *Comparable*

```
public interface Comparable
        {
              int compareTo(object  otherObject)
```

```
            }
```

l' invocazione *a.compareTo(b)* deve restituire un numero negativo se a precede b, 0 se a=b e un numero positivo se a segue b.
Se non è possibile modificare la classe in modo che realizzi l'interfaccia Comparable o non si vuole modificare quella già implementata si può definire una classe che implementi l'interfaccia Comparator

```
        public interface  Comparator
            {
                public int compare (Object firstObject, Object secondObject)
            }
```

se *comp* è un oggetto *Comparator* l' invocazione *comp.compare(a,b)* deve restituire un numero negativo se a precede b, 0 se a=b e un numero positivo se a segue b.

Es.
Coin è una classe che permette di definire oggetti "monete"

```
public class  CoinComparator  implements Comparator
    {
        public int compare (Object firstObject,  Object secondObject)
            {
              Coin first = (Coin) firstObject;
              Coin second = (Coin) secondObject;
             /*
                 get Value()  metodo della classe Coin
                 che restituisce il valore di una moneta
             */
              if first.getValue() <  second.getValue()    return  -1;
              if first.getValue()  =  second.getValue()    return  0;
              if first.getValue()  >  second.getValue()    return  1;
            }
    }
```

allora si può ordinare un array di Coin

```
                    Coin[] a = ..........;
                    Comparator comp = new CoinComparator();
                    Arrays.sort(a, comp);
```

5. Tabelle

```
public class Studente
    {
        public String cognome, nome, classe;
    }

public class Test_stud
    {
        public static void main(String[] args)
            {
               Studente[] tabella  = new Studente[10];
```

```
        for( int i = 0; i< tabella.length; i++)
            {
               tabella[i]  =  new Studente() ;
               tabella[i].cognome = "pippo";
               ..........................
            }
}
```

CAPITOLO 10 - File

1. Flussi, lettori e scrittori

Flussi (stream) : accedono a sequenze di byte : classi *InputStream* e *OutputStream* e loro sottoclassi

Lettori (reader) e scrittori (writer) accedono a sequenze di caratteri : classi *Reader* e *Writer* e loro sottoclassi

Per leggere e scrivere file su disco dobbiamo creare oggetti di tipo

FileReader e *FileWriter* per dati testuali

FileInputStream e *FileOutputStream* per dati binari

Tutte queste classi si trovano nel package *java.io*

I flussi , i lettrori, gli scrittori elaborano solo singoli byte o caratteri; per elaborare linee di testo o interi oggetti bisogna combinarli con altre classi.

2. Leggere e scrivere file di testo

Scrivere

Apertura del file (associazione file logico – file fisico)

FileWriter nome_logico_file = *new FileWriter (*percorso+nome_fisico_file)

Per aprire un file già esistente per accodargli i dati

FileWriter nome_logico_file = *new FileWriter* (percorso+nome_fisico_file*,* ***true***)

L'oggetto cosi creato viene utilizzato per creare un oggetto bufferizzato

BufferedWriter out = *new BufferedWriter(*nome_logico_file)

Si utilizzano quindi i metodi

*write(*stringa):	per scrivere una stringa
newLine():	per inserire un carattere di fine linea

alla fine chiudere con il metodo *close().*

Leggere

Apertura del file (associazione file logico – file fisico)

FileReader nome_logico_file = *new FileReader* (percorso+nome_fisico_file)

L'oggetto cosi creato viene utilizzato per creare un oggetto bufferizzato

BufferedReader in = *new BufferedReader(* nome_logico_file)

Si utilizza quindi il metodo

ReadLine() : per leggere un'intera riga

Questo metodo restituisce null quando si è alla fine del file.

alla fine chiudere con il metodo close().

3. Flussi di oggetti

E' possibili scrivere e leggere interi oggetti. Gli oggetti vengono salvati in formato binario perciò si usano i flussi :

ObjectOutputStream : per scrivere
ObjectInputStream : per leggere

Per poter salvare gli oggetti in un flusso tali oggetti devono essere istanze di una classe che implementa l'interfaccia Serializable

```
public class  nome_classe   implements Serializable

        {
                .............
        }
```

l' interfaccia *Serializable* non ha metodi.

Il processo di memorizzazione di un oggetto in un flusso si chiama serializzazione perchè ogni oggetto riceve un numero di serie nel flusso : se lo stesso oggeto viene salvato 2 volte la seconda volta viene scritto solo il suo numero di serie (numeri di serie ripetuti fanno riferimento allo stesso ogggetto).

Perchè non tutte le classi implementano l'interfaccia Serializable? per motivi di sicurezza : una volta che una classe è serializzabile chiunque può scriverla su disco e analizzare il file.

Si può non voler memorizzare alcuni attributi : devono essere dichiarati *transient*

private transient tipo nome_attributo.