Esercizi con uso di linguaggio JAVA

// Prima **Applicazione** in JAVA

```
class Saluto {                          // nome classe uguale a nome file sorgente:  Saluto.java

    public static void main(String[] args)   {     // static : per chiamare questo metodo direttamente
                                                   // senza  chiederne l'esecuzione come servizio
            System.out.println("Ciao a tutti !");
    }
}
/* Risultato:  compare il messaggio su video
               nella sezione General Output
*/
```

General Output

```
--------------------Configuration: Saluto
Ciao a tutti!

Process completed.
```

Costrutto SEQUENZA con operatori aritmetici e di assegnamento

```
class Cognome {

  public static void main (String arg[])  {

                              // dichiarazione di variabili definendole di tipo reale a doppia precisione
       double  a = 3.5;
       double b = 2;       // … ed operazioni di assegnamento

       System.out.println("La somma dei valori è :  " + (a+b) );
     // si noti la corretta interpretazione dell'operatore polimorfo come concatenazione di
     // stringhe oppure come operatore di somma

       System.out.println("La differenza dei valori è :  " + (a - b));

       System.out.println("Il prodotto dei valori è :  " + ( a * b));

       System.out.println("Ora i dati non sono interpretati come valori in una operazione ma
     accodati : " +  a + b );
  }
}
```

Effetto a video:

```
La somma dei valori è :  5.5                  ← somma 3.5 + 2
La differenza dei valori è :  1.5
Il prodotto dei valori è :  7.0
Ora i dati non sono interpretati come valori in una operazione ma accodati : 3.52.0   ← valore 3.5  seguito da valore 2.0

Process completed.
```

Testo del problema (*tipo da verifica fine anno STA - secondo biennio*):

Nota l'equazione di una retta **y = m*x + q**, progettare un algoritmo che impostati i valori di **m**, di **q** e del valore minimo di **x**, calcoli **numN** punti equispaziati della retta noto il passo di discretizzazione **dx**. Si desidera visualizzare i risultati in forma tabellare: per ogni punto deve essere visualizzato su video il valore di x e il corrispondente valore di y. L'algoritmo deve essere realizzato con ***costrutto iterativo***.
Implementare con uso di linguaggio Java:

1. soluzione non modulare e modulare usando *iterazione post-condizionale*
2. soluzioni non modulari e modulari usando *iterazione pre-condizionale*

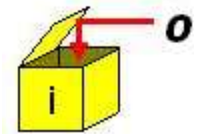
Costrutto Ripetizione con uso do … while

ciclo con controllo in coda o *iterazione post-condizionale* con sintassi:

do
istruzione
while (espressione);

Ripete l'esecuzione dell'istruzione (o il blocco di istruzioni) **mentre** l'espressione è vera.

Listing di soluzione *non modulare* usando *iterazione post-condizionale*

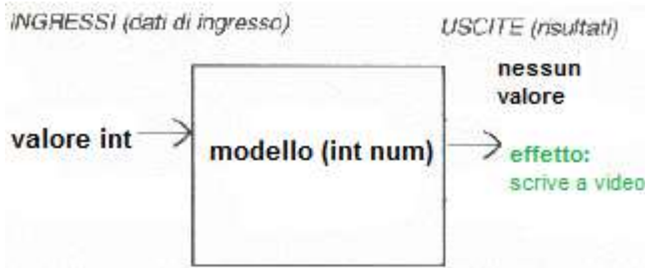



```
class Retta {
  public static void main (String [] args)  {

        int numN = 3;         // numero delle iterazioni cioè dei punti da calcolare
                              // variabile impostabile da programma

        int i = 0;            // contatore per scandire il numero delle iterazioni

        double x = 0.0, dx = 0.1, y, m = 5.0, q = 10.0;
        System.out.println("\nAscisse\t\tOrdinate\n");

        do  {
              y = m*x + q;
              System.out.println("\t" +  x + "\t\t" + y + "\n");
              x = x + dx;
              i++ ;        // incremento il contatore ad ogni iterazione

        }while  (i < numN);     // mentre il contatore è minore del numero massimo delle iterazioni
                                // ripete l'esecuzione delle istruzioni nel corpo

   System.out.println("Termine\n");
  }
}
```

Effetto a video:

```
General Output

Ascisse     Ordinate
      0.0     10.0
      0.1     10.5
      0.2     11.0
Termine

Process completed.
```

Listing di soluzione *modulare* usando *iterazione post-condizionale*



Progettiamo un **metodo** (*visto* come una scatola nera) con nome **modello**
che usa una variabile intera **num** = valore int
passato alla *chiamata*
e non ritorna un valore
ma produce come effetto la stampa a video

Implementazione del metodo** ed esempio di **chiamata diretta[1]:

parametro del metodo

```
public static void modello (int num){          // Called Function
    .....
    do {
          .....
          System.out.println("..");
    }while (i < num);
}

                          {                     // Calling code
                            int numN = 3;
                            modello (numN);
                          }
```

valore 3

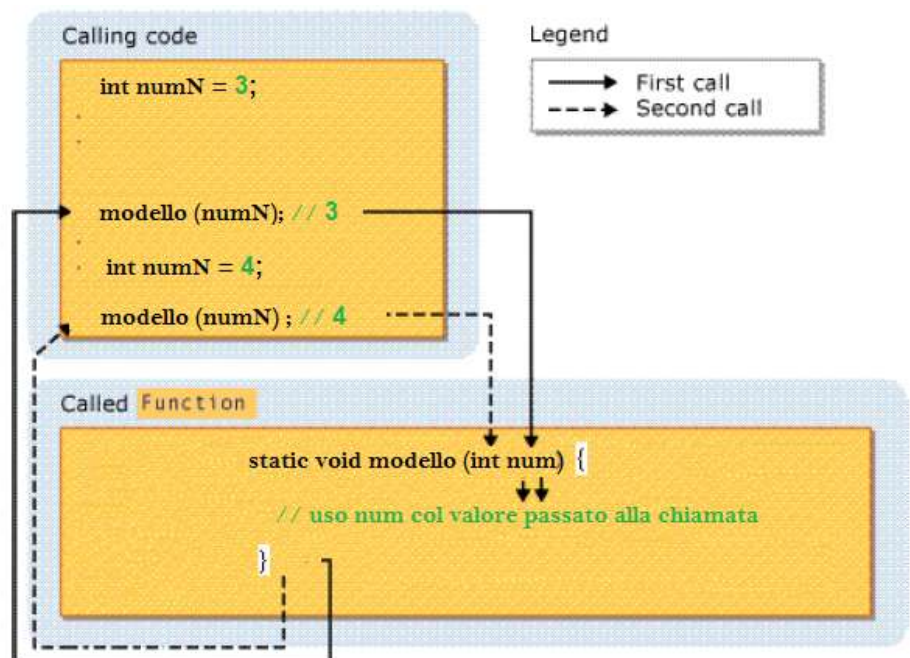
argomento attuale

Quindi si progetta una funzione (*metodo* della classe) che non ritorna un valore (**void**), riceve in ingresso un valore intero ed al suo interno *referenzia* tale valore intero (nell'esempio il **numero 3**) con **num**.

Nb: in Java con **static** si intende che la funzione può essere *chiamata direttamente* (di default è **public** cioè *visibile a tutti*) e la sintassi completa è la seguente:

<*modificatori*> **tipoDiRitornoDelMetodo** nomeDelMetodo(<*parametri*>) { corpoDelMetodo }

Nella figura a fianco è illustrato il caso di due chiamate della funzione:



[1] All'URL http://new345.altervista.org/Dispense/FUNZIONI.pdf (pg. 2-3)

```
class RettaModulare {

  /**
     metodo che calcola i punti e visualizza le coordinate in formato tabellare
     @param  num – numero dei punti da calcolare di tipo intero
  */
  public static void modello (int num){

        int i = 0;              // contatore per scandire il numero delle iterazioni

        double x = 0.0, dx = 0.1, y, m = 5.0, q = 10.0;
        System.out.println("\nAscisse\t\tOrdinate\n");

        do  {
                y = m*x + q;
                System.out.println("\t" +  x + "\t\t" + y + "\n");
                x = x + dx;
                i++ ;          // incremento il contatore ad ogni iterazione

         }while  (i < num);      // mentre il contatore è minore del numero massimo delle iterazioni
                                  // ripete l'esecuzione delle istruzioni nel corpo
  }

  public static void main (String [] args)  {

     int numN = 3;         // numero dei punti da calcolare cioè delle iterazioni
                           // variabile impostabile da programma
                           // per buono stile: nome con iniziale minuscola e formato camel-case
    modello (numN);

    System.out.println("Termine\n");
  }
}
```

(Nota [2] riferita a "@param num – numero dei punti da calcolare di tipo intero")

Si noti come il codice della funzione non venga modificato quando invece di tre punti se ne voglia calcolare un numero diverso: basta digitare nel flusso principale il nuovo valore e **riusare lo stesso metodo**

vedremo in seguito come arrotondare i valori delle ascisse per considerare solo due cifre decimali significative ottenendo la seguente tabella nel caso si imposti numN = 5

```
General Output

      0.1      10.5

      0.2      11.0

      0.3      11.5

      0.4      12.0

 Termine


 Process completed.
```

Buono stile: notazione CamelCase

La **Notazione a Cammello** o in inglese **CamelCase** è la pratica, nata durante gli anni settanta, di scrivere parole composte o frasi unendo tutte le parole tra loro, ma lasciando le loro iniziali maiuscole.

Il nome (letteralmente "*carattere a cammello*") deriva dai "salti" all'interno di una parola, che fanno venire in mente le gobbe di un cammello. La prima lettera può essere sia maiuscola (es. CamelCase), come il **nome delle classi** in Java, che minuscola (es. camelCase), come il nome delle variabili in Java.

[2] Per **lista dei tag di JavaDoc** e guida nell'uso di tale documentazione in **JCreator**

Costrutto Ripetizione con uso while

ciclo con controllo in testa o *iterazione pre-condizionale* con sintassi: ***while (espressione)***

istruzione

Ripete l'esecuzione dell'istruzione (o il blocco di istruzioni) **mentre** l'espressione è vera. Differenza rispetto al costrutto precedente (ciclo con controllo in coda) è che l'istruzione non viene mai eseguita se la condizione è falsa.

Listing di soluzione *non modulare* usando *iterazione pre-condizionale*

```
class RettaAltra {

    public static void main (String [] args)  {

        int numN = 3;                    // variabile impostabile da programma
        int i = 0;
        double x = 0.0, dx = 0.1, y, m = 5.0, q = 10.0;
        System.out.println("\nAscisse\t\tOrdinate\n");

        while  (i < numN)    {
                y = m*x + q;
                System.out.println("\t" + x + "\t\t" + y + "\n");
                x = x + dx;
                i++ ;
        }
        System.out.println("\nTermine\n");
    }
}
```

Listing di soluzione *modulare* usando *iterazione pre-condizionale*

```
class RettaAltraModulare {

    /**
        metodo che calcola i punti e visualizza le coordinate in formato tabellare
        @param  num – numero dei punti da calcolare di tipo intero
    */
    public static void modello (int num){

        int i = 0;          // contatore per scandire il numero delle iterazioni

        double x = 0.0, dx = 0.1, y, m = 5.0, q = 10.0;
        System.out.println("\nAscisse\t\tOrdinate\n");

        while  (i < num)    {
                y = m*x + q;
                System.out.println("\t" + x + "\t\t" + y + "\n");
                x = x + dx;
                i++ ;
        }
    }

    public static void main (String [] args)  {

        int numN = 3;     // numero dei punti da calcolare cioè delle iterazioni
        modello (numN);
        System.out.println("\nTermine\n");
    }
}
```

Costrutto Ripetizione con uso for

ciclo con controllo in testa o *iterazione pre-condizionale* con sintassi:

for (espressione1; espressione2; espressione3)
istruzione

le espressioni contenute nella parentesi hanno il seguente significato:

1) una assegnazione del valore iniziale del contatore
2) una condizione logica o aritmetica che indica la fine del *ciclo*
3) un aggiornamento del contatore (in questo caso un incremento di + 1)

Il corpo del *for* può essere una sola istruzione oppure un blocco di istruzioni { } che viene ***ripetuto mentre*** è verifica la condizione impostata

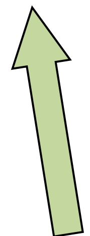
Listing di altra soluzione non modulare usando *iterazione pre-condizionale*

```
class RettaFor{

  public static void main (String [] args)  {

    final int N = 3;          // non si vuole cambiarne il valore : variabile immodificabile o costante

    double x = 0.0, dx = 0.1, y, m = 5.0, q = 10.0;

    System.out.println("\nAscisse\t\tOrdinate\n");

    for (int i = 0; i < N; i++)    {

            y = m*x + q;
            System.out.println("\t" + x + "\t\t" + y + "\n");
            x = x + dx;
    }
    System.out.println("Termine\n");
  }
}
```

Effetto :

```
General Output
Ascisse      Ordinate

    0.0      10.0

    0.1      10.5

    0.2      11.0

Termine
```

```
General Output

Ascisse      Ordinate

   0,00      10,00

   0,10      10,50

   0,20      11,00

Termine
```



nb: è anche possibile visualizzare con modifica del formato (con JDK versione 1.5 o successive).

Per esempio

```
System.out.printf ("%7.2f\t\t%5.2f\n\n", x, y);     // la formattazione è con la virgola
```

Il simbolo % specifica le seguenti formattazioni:

· riserva 7 posizioni con 2 cifre decimali per numero reale x
· riserva 5 posizioni con 2 cifre decimali per numero reale y

(*il risultato è mostrato nella figura a destra*)

/* Un **loop *INFINITO*** si può realizzare con while (*true*) poiché il valore *vero* significa sempre, oppure, come nell'esempio, con un *for infinito*, cioè

for (; ;)

La condizione di uscita viene testata dentro il blocco di istruzioni del ciclo infinito usando un costrutto alternativa con sintassi:

if (espressione)
istruzione_vera

Se tale condizione è vera, si esce dal ciclo con l'istruzione ***break*** che forza l'uscita dal corpo in cui si trova.
*/

```
class Loop{

  public static void main (String [] args)  {

    final int N = 3;
    int i = 0;
    double x = 0.0, dx = 0.1, y, m = 5.0, q = 10.0;
    System.out.println("\nAscisse\t\tOrdinate\n");

    for ( ; ; )   {
              y = m*x + q;
              System.out.println("\t" +  x + "\t\t" + y + "\n");
              x = x + dx;
              i++ ;
              if (i  >=  N)           // uso di operatore relazionale[3] maggiore o uguale
                break;       // per uscire dal corpo del loop
    }
    System.out.println("Termine\n");
  }
}
```

Produce lo **stesso effetto**:

```
General Output
Ascisse     Ordinate

    0.0     10.0

    0.1     10.5

    0.2     11.0

Termine
```

[3] Altri operatori relazionali all'URL: http://new345.altervista.org/Dispense/Operatori_relazionali_confronto.pdf

Arrotondare i valori delle ascisse
per considerare solo due cifre decimali significative

```
import java.lang.Math;

class RettaModulare {

/**
    metodo per arrotondare a due cifre decimali significative
    Sostanzialmente moltiplicare il numero per 100,
    applicare il metodo della classe Math a quest'ultimo, e infine ridividere per 100.

    @param  x – numero da arrotondare di tipo double
    @return  x – numero arrotondato alla seconda cifra decimale significativa
 */
 public static double arrotonda(double x){
         x = Math.round(x*100);
         x = x/100;
        return x;
  }

 /**
     metodo che calcola i punti e visualizza le coordinate in formato tabellare
     @param  num – numero dei punti da calcolare di tipo intero
 */
 public static void modello (int num){

        int i = 0;                // contatore per scandire il numero delle iterazioni

        double x = 0.0, dx = 0.1, y, m = 5.0, q = 10.0;
        System.out.println("\nAscisse\t\tOrdinate\n");

        do   {
                        y = m*x + q;
                        System.out.println("\t" +  x + "\t\t" + y + "\n");
                        x = x + dx;

                        x = arrotonda(x);
                        i++ ;          // incremento il contatore ad ogni iterazione

         }while  (i < num);      // mentre il contatore è minore del numero massimo delle iterazioni
                                 // ripete l'esecuzione delle istruzioni nel corpo
 }

 public static void main (String [] args)  {

    int numN = 5;              // numero dei punti da calcolare cioè delle iterazioni
                               // variabile impostabile da programma

    modello (numN);

    System.out.println("Termine\n");
 }
}
```

Arrotondare i numeri decimali dopo la virgola

Se il nostro problema è **arrotondare i numeri decimali dopo la virgola**, possiamo risolvere creando dei metodi ad hoc che sfruttino quelli che **Java** mette a disposizione per **arrotondare un numero a un intero**.

Java fornisce differenti metodi di **arrotondamento** e **troncamento**: si possono usare i metodi della classe Math, definita nel *package **java.lang,*** elencati di seguito (**Math** contiene solo metodi **static**):

ROUND - int round(int i) oppure **long round(double d)**
questo metodo rappresenta l'arrotondamento forse più usato dall'uomo, il cosiddetto "**arrotondamento matematico**". In pratica arrotonda il numero all'intero più piccolo se la prima cifra decimale eliminata è minore/uguale a zero e all'intero più grande se è maggiore/uguale a 5.

ESEMPI:
double round(4.4) -> restituisce 4.
double round(4.5) -> restituisce 5.
double round(4.6) -> restituisce 5.

FLOOR - double floor(double d)
questo metodo ("*floor*" - *pavimento*) arrotonda il numero alla cifra intera inferiore (troncamento **per difetto**).

ESEMPI:
double floor(4.4) -> restituisce 4.
double floor(4.6) -> restituisce 4.

CEIL - double ceil(double d)
questo metodo ("*ceil*"- *soffitto)* arrotonda il numero alla cifra intera superiore (troncamento **per eccesso**)

ESEMPI:
double ceil(4.4) -> restituisce 5.
double ceil(4.6) -> restituisce 5.

RINT - double rint(double d)
esattamente uguale a ROUND ma arrotonda al **numero pari più vicino se è equidistante da due valori** (cioè qualora fosse 5,5 o 6,5 arrotonda a 6, in caso 3,5 o 4,5 arrotonda a 4).

ESEMPI:
double rint(4.4) -> restituisce 4.
double rint(4.5) -> restituisce 4 (pari più vicino).
double rint(4.6) -> restituisce 5.

double rint(5.4) -> restituisce 5.
double rint(5.5) -> restituisce 6 (pari più vicino).
double rint(5.6) -> restituisce 6.