Astrazione dei dati

È un processo logico seguito da un progettista (nell'***O***bject ***O***riented ***D***esign) per cui, prescindendo dalle caratteristiche individuali e personali degli oggetti, si individuano nella realtà solo le **proprietà e i comportamenti generali** racchiudendoli nel concetto di **classe**.

Metodo di progettazione OO proposto da Abbot, Booch e Lorensen:

tecnica che suggerisce di evidenziare nel testo gli elementi candidati a diventare, nel progetto, una **classe** (testo evidenziato con bordo), **attributi o proprietà** (testo evidenziato con sottolineatura punteggiata) e il **comportamento** (testo evidenziato con sottolineatura) degli oggetti stessi. Il comportamento è realizzato con metodi, ovvero sottoprogrammi che elaborano i dati memorizzati nelle proprietà per produrre nuove informazioni.

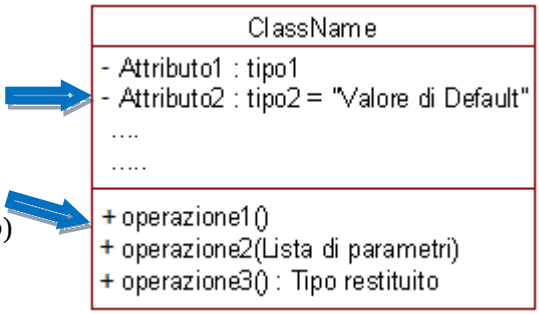
Con uso **UML** si progetta la **classe nomeClasse** con:

- **proprietà private** (incapsulate nella classe)
- **metodi pubblici**:
  - che permettono di accedere alle proprietà private (*metodi di accesso*)
  - che definiscono l'elaborazione (operazioni proprie delle varie *istanze*)

Descrizione con UML **diagramma statico di una classe:**

Il simbolo – specifica che l'accesso è privato (***specificatore di accesso privato***)

Il simbolo + specifica che l'accesso è pubblico (***specificatore di accesso pubblico***)

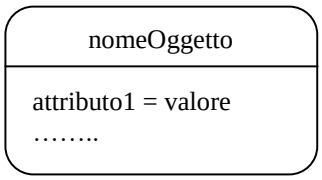


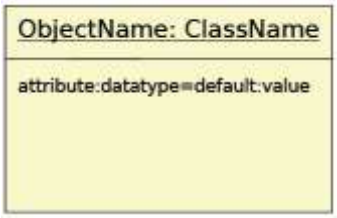
Quindi si progettano gli attributi "***nascosti***": *visibili solo dai metodi della classe*
mentre i metodi, solitamente, si pensano pubblici: *tutti possono vederli/usarli*

*Visibilità: vedremo, in seguito, diversi **modificatori di accesso***

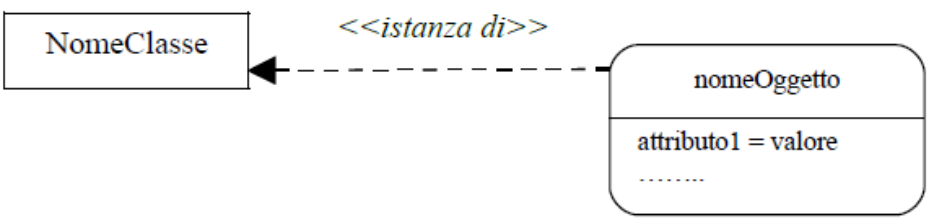
diagramma degli oggetti:

- **nome**
- ***stato*** (valori degli attributi cioè variabili istanza)



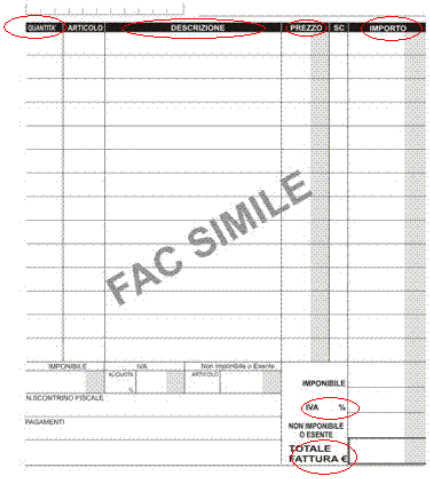


In un **diagramma degli oggetti**, l'**UML** considera un oggetto collegato alla sua classe come una forma di relazione[1] (relazione di **realizzazione** individuata da linea tratteggiata, con una freccia dalla parte della classe e lo stereotipo <<*istanza di*>>)



[1] Per approfondire: le relazioni tra classi, rappresentazione di oggetti e tipi di relazioni. Esercizi.

Una semplice fattura




```
import java.util.*;

class Fattura {                          // inizio della classe

   private String descrizione;   // variabili istanza o attributi
   private int quantita;
   private double prezzoUnitario;
   private double imponibile;
   private double IVA19;
   private double totaleLordo;

   public Fattura()   {         // costruttore di default della classe Fattura per inizializzare le variabili istanza
     System.out.println("Descrizione dell'articolo: ");
     descrizione = inputStr();                  // uso di metodo della classe per leggere una stringa

     System.out.println("Quantità: ");
     quantita = inputInt();                     // uso di metodo della classe per leggere un intero

     System.out.println("Prezzo unitario: ");
     prezzoUnitario = inputDouble();            // uso di metodo della classe per leggere un double

     imponibile = quantita * prezzoUnitario;
     IVA19 = imponibile * 19/100;
     totaleLordo = imponibile + IVA19;
   }
  public String inputStr()  {                   // input di una stringa (oggetto)
    Scanner flussoInput = new Scanner (System.in));  // istanzio Scanner ….. variabile locale (*)
                                                     // senza necessità di gestire eccezioni… migliorabile
    try  {
      String stringa = flussoInput.nextLine();
      return(stringa);
    }
    catch (Exception e)  {         System.out.println("Errore: "+ e +"in input");
                                    System.exit(0);
                                    return("");
                                  }
   }
   public int inputInt()  {                     // input di numero intero
    Scanner flussoInput = new Scanner (System.in));  // istanzio Scanner (*)
    try  {
           String stringa = flussoInput.nextInt();
           return(Integer.parseInt(stringa));                    // analogo a Integer.valueOf(stringa).intValue();
    } catch(Exception e) {         System.out.println ("Errore: "+ e +"in input");
                                    System.exit(0);
                                    return(-1);
                                  }
   }
   public double inputDouble()  {                   // input di numero double
    Scanner flussoInput = new Scanner (System.in));  // istanzio Scanner ….. variabile locale (*)
    try  {
            String stringa = flussoInput.nextDouble() ;
            return(Double.parseDouble(stringa));
    } catch(Exception e) {          System.out.println ("Errore: "+ e +"in input");
                                    System.exit(0);
                                    return(-1.0);
                                  }
   }
```

```
public void stampaFattura()  {              // stampa la fattura
    System.out.println (" ");
    System.out.println ("**********************************");
    System.out.println ("Fornitura di N." + quantita  +" "+ descrizione);
    System.out.println ("Al prezzo unitario di Euro "  + prezzoUnitario);
    System.out.println ("----------------------");
    System.out.println ("Totale imponibile..... "  + imponibile);
    System.out.println ("IVA 19 % ..... "  + IVA19);
    System.out.println ("Totale IVA compresa .. "   + totaleLordo);
    System.out.println ("**********************************");
    System.out.println (" ");
}

public static void main(String arg[])  {   // metodo principale …. per testare la classe

    Fattura f;                  // dichiaro f  oggetto di tipo Fattura
    f = new Fattura();          // alloco l'oggetto in RAM

    f.stampaFattura();
}
} // fine della classe
```

```
Descrizione dell'articolo:
cacciavite
Quantità:
5
Prezzo unitario:
10

**********************************
Fornitura di N.5 cacciavite
Al prezzo unitario di Euro 10.0
----------------------
Totale imponibile..... 50.0
IVA 19 % ..... 9.5
Totale IVA compresa .. 59.5
**********************************
```

Da completare con approfondimenti successivi:

prima illustrando il diagramma di specifica …

… poi completando il diagramma di implementazione esplicitando i tipi

… ed i modificatori di accesso (visibilità)

- **Pubblica (+)**: l'attributo è accessibile da qualsiasi altro oggetto dotato di riferimento all'oggetto che contiene l'attributo in questione;
- **Privata (-)**: l'attributo è accessibile solo all'interno della classe di appartenenza (dichiarante);
- **Protetta (#)**: l'attributo è accessibile da tutte le istanze delle classi che "ereditano" da quella in cui l'attributo è definito;
- **Package (~)**: l'attributo è accessibile da qualsiasi altro oggetto istanza di classi appartenenti allo stesso package o in un altro ad esso annidato a qualsiasi livello.

stessa sottocartella (default)

Esempi con astrazione dei dati e OOD

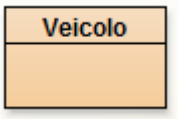
Testo del problema

Un centro di ricerche automobilistico desidera realizzare un software per calcolare l'energia cinetica e lo spazio di frenata di un veicolo su strada (automobile, autocarro ecc..) dati la sua massa, la velocità iniziale (in km/h) ed il coefficiente di attrito gomme-asfalto. Ricordare che il valore della costante ***g*** *di accelerazione di gravità vale 9,8 m/s^2.*

Un ricercatore del centro ricorda ai programmatori che l'energia di movimento (cinetica) e lo spazio di frenata di un veicolo si calcolano con le seguenti relazioni:

- *l'energia cinetica = ½ *massa*velocità^2*
- *spazio di frenata = velocità^2 / (2*g*coefficienteAttrito), ottenuto uguagliando l'energia cinetica con il lavoro svolto dalle forze di attrito ruote-asfalto durante la frenata del veicolo*

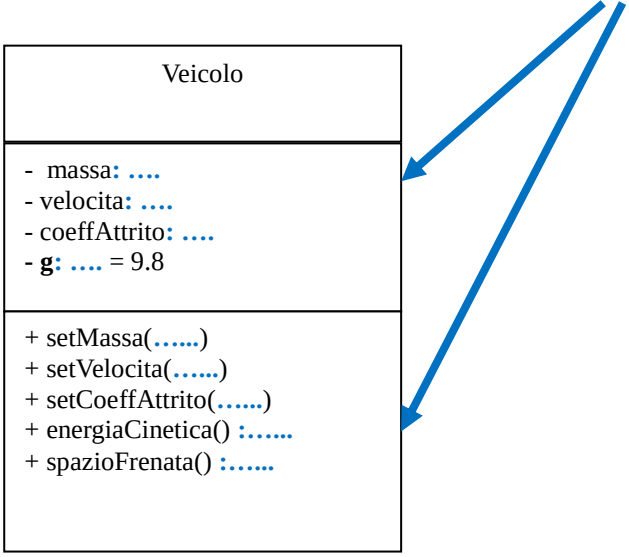
Con uso **UML** si progetta la **classe Veicolo** con:



- **proprietà private** (incapsulate nella classe)
- **metodi pubblici**:
  - che permettono di *accedere* alle proprietà private
  - che definiscono l'*elaborazione*

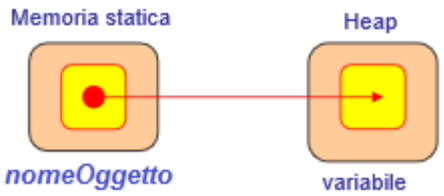
Nella figura seguente, il *diagramma statico della classe senza esplicitare il tipo ma evidenziando i* ***modificatori di accesso (visibilità)***

….. da completare tale diagramma di implementazione esplicitando i tipi

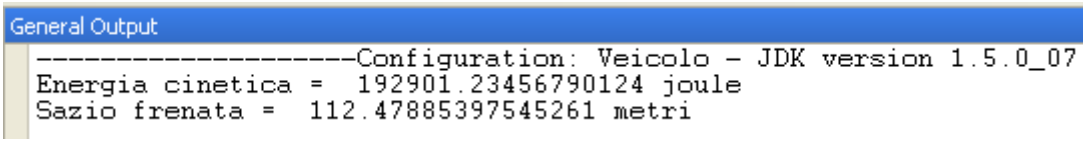


Implementazione come ***applicazione*** Java

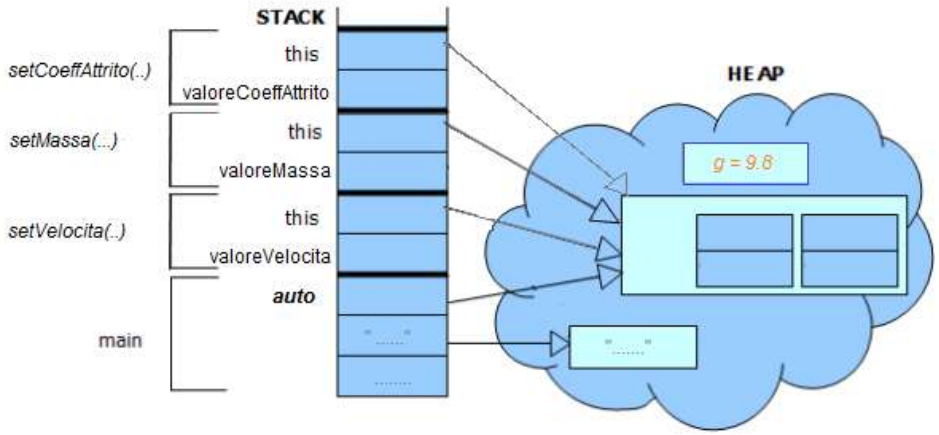
```
public class Veicolo {              // dichiarazione degli elementi comuni a tutti gli esemplari

        private final double g=9.8;                                  // immodificabile
        private double massa, velocita, coeffAttrito;
                                                                     // alcuni metodi di accesso
        public void  setMassa(double valoreMassa) {
                massa= valoreMassa;
        }
        public void  setVelocita (double valoreVelocita) {
                velocita= valoreVelocita*(1000.0/3600.0);            // conversione in m/s
        }
        public void  setCoeffAttrito(double valoreCoeffAttrito) {
                coeffAttrito= valoreCoeffAttrito;
        }
         public double energiaCinetica(){
                return (massa*velocita*velocita/2.0);
        }
        public double spazioFrenata(){
                return (velocita*velocita/(2.0*g*coeffAttrito));
        }
                                                  // metodo che imposta il flusso dell'elaborazione dell'applicazione
        public static void main (String args[]){
                                                   // usa costruttore di default
                Veicolo auto = new Veicolo();      // alloca un riferimento
                                                   // all'oggetto in RAM

                                                   // elaborazione dell'oggetto allocando in heap
                auto.setMassa(500.0);              // 5 quintali
                auto.setVelocita(100.0);           // 100 km/h
                auto.setCoeffAttrito(0.35);
                System.out.println ("Energia cinetica =  "+ auto.energiaCinetica() + " joule");
                System.out.println ("Sazio frenata =  "+ auto.spazioFrenata() + " metri");
         }
}
```



Effetto con uso IDE **JCreator**:



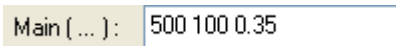
Memoria RAM

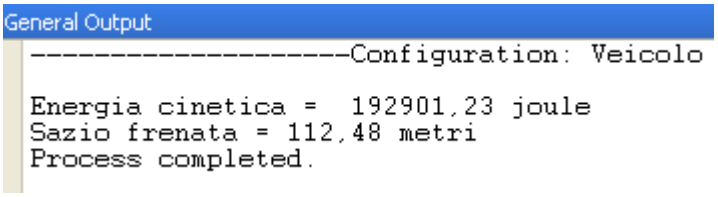


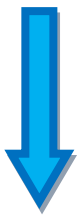
Implementazione come ***applicazione*** Java con **passaggio di argomenti** **da linea di comando**

```
public class Veicolo2 {              // dichiarazione degli elementi comuni a tutti gli esemplari

    private final double g=9.8;                                  // immodificabile
    private double massa, velocita, coeffAttrito;

    public void  setMassa(double valoreMassa) {
            massa= valoreMassa;
    }
    public void  setVelocita (double valoreVelocita) {
            velocita= valoreVelocita*(1000.0/3600.0);            // conversione in m/s
    }
    public void  setCoeffAttrito(double valoreCoeffAttrito) {
            coefficienteAttrito= valoreCoeffAttrito;
    }
    public double energiaCinetica(){
            return (massa*velocita*velocita/2.0);
    }
    public double spazioFrenata(){
           return (velocita*velocita/(2.0*g*coeffAttrito));
    }
                                 // metodo che imposta il flusso dell'elaborazione dell'applicazione

    public static void main (String args[]){
                                                  // metodo costruttore di default
           Veicolo2 auto = new Veicolo2();       // alloca un riferimento all'oggetto in RAM

                                 // elaborazione dell'oggetto allocando in heap

          auto.setMassa(Double.parseDouble(args[0]));                  // da linea di comando
          auto.setVelocita(Double.parseDouble(args[1]));
          auto.setCoeffAttrito(Double.parseDouble(args[2]));
          System.out.println();
          System.out.printf ("Energia cinetica =  %.2f joule ", auto.energiaCinetica());
          System.out.println();
          System.out.printf ("Sazio frenata = %.2f metri",  auto.spazioFrenata() );
     }
}
```

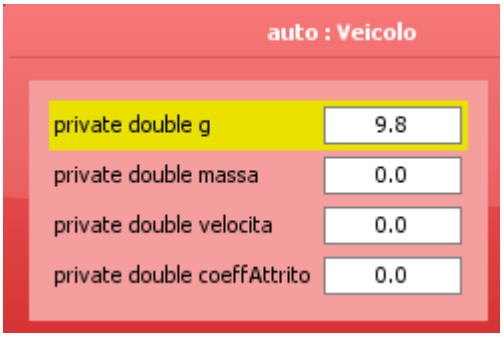
Impostati gli argomenti



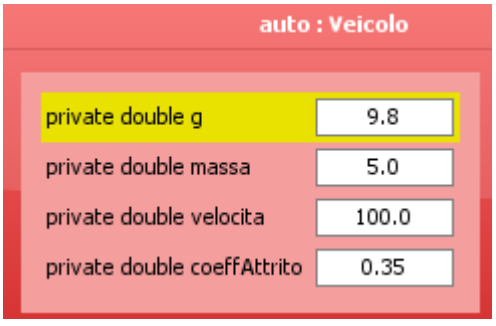




All'inizio: tutti gli attributi inizializzati di *default*

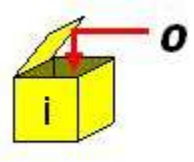


…. al termine lo *stato* dell'oggetto:



Ambiente IDE JCreator

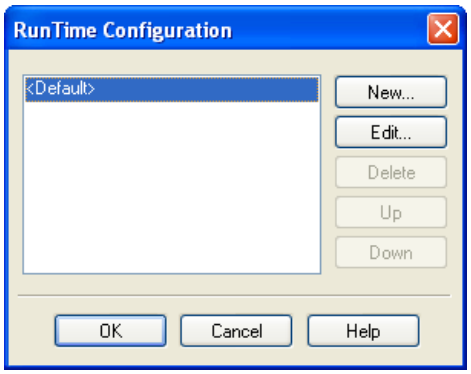
// **passaggio di argomenti** all'applicazione da linea di comando



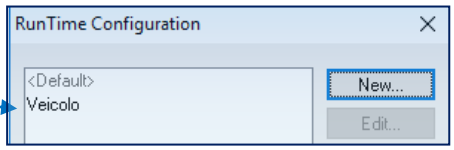

```
class Passa {
        public static void main (String arg[])  {

                for (int i=0; i<arg.length; i++) {   // costrutto iterativo[2]: ripete mentre i<numero parole
                                                     // ad ogni esecuzione, si incrementa i di uno
                                                     // a partire da valore zero

                    System.out.println ("Argomento " +  i  +": " + arg[i]);   // concatenazione tra stringhe
                }
        }
    }
/* Risultato: se si configura l'ambiente di run time digitando prof "paola biasotti", compare su video:
            Argomento 0: prof
            Argomento 1: paola biasotti
*/
```

In ambinte **IDE** JCreator:

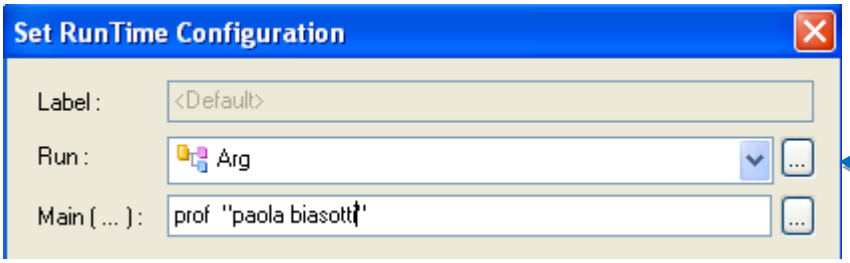
in **Run --> Runtime Configuration** cliccare **<Default>**



oppure **New…**
potendo *aggiungere* con diverse Label



ed **Edit**are le *stringhe*
che si desiderano come ***argomenti*** del main(...)



elenco degli eseguibili di default nel *progetto*

Infine confermare (premendo **OK** in entrambe le finestre)

Testo del problema

Un container, che ha la forma di un parallelepipedo, con una lunghezza di metri 6,10, altezza e larghezza (uguali tra loro) di metri 2,438, contiene al suo interno una determinata sostanza ad un certo livello di altezza X metri, il cui peso specifico è espresso in kg per metri cubi.

---

[2] Scaricabili slides condivise su Drive

Calcolare il peso in tonnellate del contenuto del container ed effettuare una verifica se il peso del materiale potrà essere trasportato, tenendo conto che il peso lordo massimo è di tonnellate 20,32.

nb: peso = pesoSpecifico* (livelloSostanza*lunghezza*larghezza)
dove (livelloSostanza*lunghezza*larghezza) esprime il volume

Specifiche concettuali:

dati input: *lunghezza, larghezza (=altezza) del container; peso specifico e livello di altezza della sostanza contenuta*

risultati: *peso ed esito della verifica*

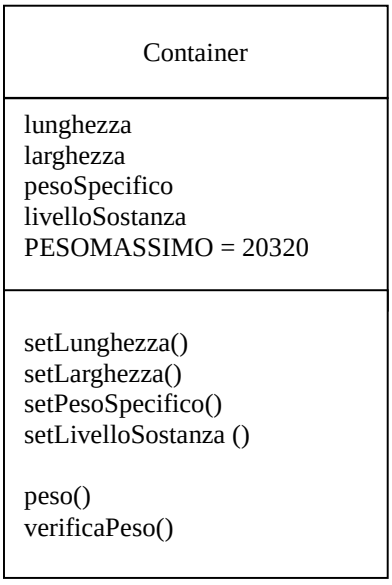
Specifiche tecnologiche:

realizzare come ***applicazione*** **Java** la soluzione del problema

Specifiche per il testing:

prevedere i due casi di peso della sostanza che supera/non supera quello massimo del container

Con uso **UML** si progetta la **classe Container (**in figura il **diagramma di specifica)**



nb: poi completare proponendo il **diagramma di implementazione** esplicitando nel diagramma il ***tipo*** (sia degli attributi sia dei metodi e degli eventuali argomenti passati ai metodi) ed i ***modificatori d'accesso***.

Implementazione in Java con metodo statico **input()** per leggere da tastiera

```
import java.io.*;

class Container {

        private double lunghezza, larghezza, pesoSpecifico, livelloSostanza;
        private final double PESOMASSIMO = 20320.0;

        public void setLunghezza (double valore) { lunghezza = valore;}
        public void setLarghezza (double valore) { larghezza = valore;}
        public void setPesoSpecifico (double valore) { pesoSpecifico = valore;}
        public void setLivelloSostanza (double valore) { livelloSostanza = valore;}
        public double peso(){
                return ( pesoSpecifico* (livelloSostanza*lunghezza*larghezza) );
        }
        public boolean verificaPeso(){

                if ( peso() <= PESOMASSIMO)
                   return true;
                else
                     return false;
        }
        static String input()  {   // metodo per input da tastiera … modificare usando la classe Scanner

         BufferedReader stdin = new BufferedReader( new InputStreamReader(System.in));
        try   {
                String stringa = stdin.readLine();
                return(stringa);
               }
                catch (Exception e)  {
                        System.out.println("Errore: "+ e +"in input");
                        System.exit(1);
                        return("");
                }
        }
        public static void main(String args[]){

                String messaggio1, messaggio2;      // variabili d'uso …. aggiunte
                String input;
                Container oggetto = new Container();

                System.out.print("\nLunghezza in metri = ");
                input = input();
                oggetto.setLunghezza(Double.parseDouble(input));
                System.out.print("\nLarghezza/altezza in metri = ");
                input = input();
                oggetto.setLarghezza(Double.parseDouble(input));
                System.out.print("\nLivello in metri = ");
                input = input();

                oggetto.setLivelloSostanza(Double.parseDouble(input));
                System.out.print("\nPeso specifico in kg/m^3 = ");
                input = input();
                oggetto.setPesoSpecifico(Double.parseDouble(input));

                double risultato = oggetto.peso()/1000.0 ;        // in tonnellate

                risultato= (double) Math.round(risultato*1000)/ 1000.0;  // arrotondo a 3 cifre


                messaggio1="Peso sostanza= " + risultato +" tonnellate";

                if (oggetto.verificaPeso())
```

```
                messaggio2="Peso sostanza non supera peso massimo";
            else
                messaggio2="Peso sostanza supera peso massimo";

            System.out.println("\n" + messaggio1 +"\n" + messaggio2 );
        }
}
```

Effetto della prima prova:

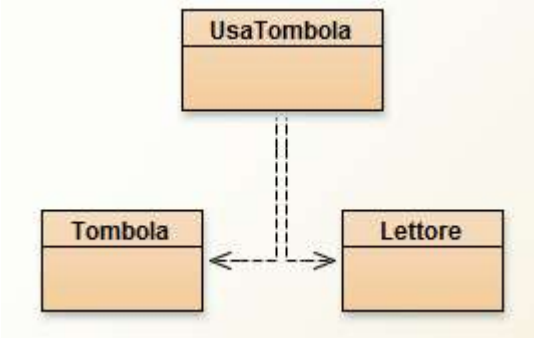
```
General Output
--------------------Configuration: Container

Lunghezza in metri = 6.10

Larghezza/altezza in metri = 2.438

Livello in metri = 1.8

Peso specifico in kg/m^3 = 1200

Peso sostanza= 32.123 tonnellate
Peso sostanza supera peso massimo

Process completed.
```

Effetto della seconda prova con la sola modifica nel livello:

```
General Output
--------------------Configuration: Container

Lunghezza in metri = 6.10

Larghezza/altezza in metri = 2.438

Livello in metri = 1

Peso specifico in kg/m^3 =  1200

Peso sostanza= 17.846 tonnellate
Peso sostanza non supera peso massimo

Process completed.
```

Esercizio da risolvere

progettare e realizzare la classe ***Tombola****,*
la classe ***Lettore*** *e l'applicazione (****UsaTombola****)*
che ne usa istanze per risolvere i seguenti problemi:



Problema 1
Calcolare la somma dei primi 10 valori estratti dalla tombola.
Se viene estratto il numero precedentemente *digitato da tastiera*
(ad esempio 50) il programma deve fermarsi.

Problema 2
Un'urna contiene i 90 numeri della tombola. Simulare l'estrazione di un numero N di numeri precedentemente *digitato da tastiera* (ad esempio 30) e stampare la somma dei valori estratti.

Per approfondire*:*

Ereditarietà, Polimorfismo … e non solo Incapsulamento nella Tecnologia OO (slides)
rileggendo le dispense (OO) ed allenandosi con alcuni esercizi