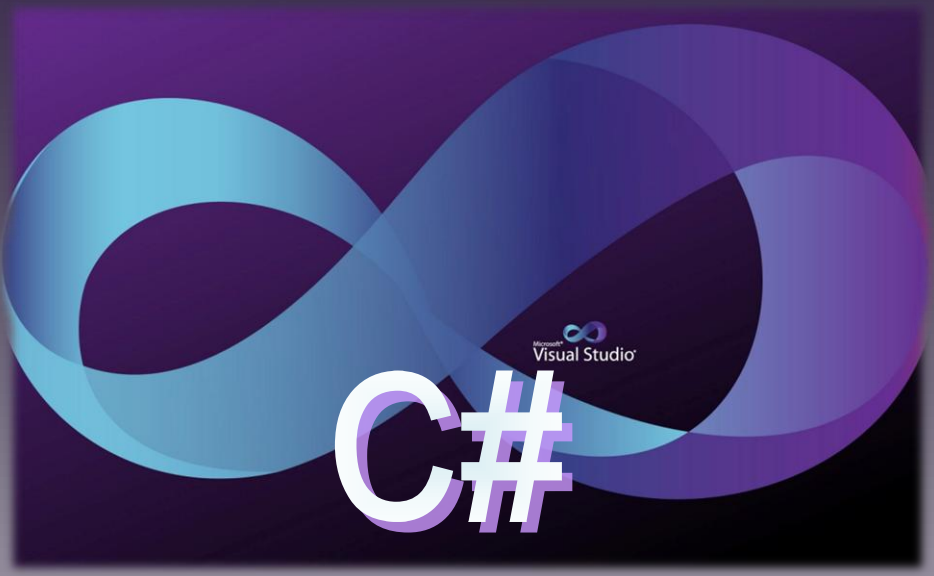
ISTITUTO TECNICO INDUSTRIALE
G. M. ANGIOY
SASSARI



CORSO DI PROGRAMMAZIONE

POLIMORFISMO E CLASSI POLIMORFICHE

DISPENSA 15.05

15-05_OOP_Polimorfismo_[06]

Versione del: 07/11/2015
Revisione numero: 15

Prof. Andrea Zoccheddu
Dipartimento di Informatica

DIPARTIMENTO
INFORMATICA E TELECOMUNICAZIONI

POLIMORFISMO DELLE CLASSI

INTRODUZIONE AL POLIMORFISMO

UN EFFETTO COLLATERALE

LIMITI DEI METODI

Consideriamo un esempio concreto.

VC#
```
public class Mostro {
  public void Verso ()
  {
    MessageBox.Show("grugnito");
  }
}
```

VC#
```
public class Mannaro : Mostro {
  public void Verso ()
  {
    MessageBox.Show("ululato");
  }
}
```

VC#
```
Mostro m = new Mostro ();
m.Verso();
```

VC#
```
Mannaro u = new Mannaro ();
u.Verso();
```

Nell'esempio proposto qui sopra (che è possibile provare al PC con Visual Studio) un'istanza di Mostro grugnisce, mentre un'istanza di Mannaro ulula; e si vede con le finestre di dialogo!

Tuttavia consideriamo di avere a disposizione una collezione (un vettore) di mostri da gestire per un'orda in un videogioco, come questa:

VC#
```
Mostro [ ] avversari = new Mostro [100]; // 100 mostri avversari
```

E supponiamo di generare casualmente i mostri da creare in una zona del videogioco:

VC#
```
Random cauale = new Random();
for (int i = 0; i<100; i++)
  if ( casuale.Next (2) == 0 )
    avversari [i] = new Mostro ();
  else
    avversari [i] = new Mannaro ();
```

Come si vede nell'algoritmo precedente, il vettore è riempito casualmente di Mostri generici e di Mannari. Questa possibilità è garantita dal meccanismo dell'ereditarietà. Questo meccanismo consente di invocare il metodo dei vari mostri direttamente dalle celle del vettore:

VC#
```
for (int i = 0; i<100; i++)
  avversari[i].Verso ();
```

questa invocazione è ammessa poiché il metodo Verso è previsto; tuttavia il funzionamento produce una qualche sorpresa (spero): tutti i Mostri grugniscono! Persino i Mannari!

In effetti le invocazioni del metodo Verso( ) si riferiscono tutte al metodo implementato nella classe Mostro; infatti sebbene le istanze siano create con diversi costruttori il metodo invocato è legato dal compilatore in

base alla dichiarazione del vettore. In altre parole, poiché le celle del vettore avversari sono tutte di tipo Mostro, il compilatore che deve verificare quale metodo di Mostro esista per le invocazioni legate alle celle e deve legare la chiamata in modo definitivo (a tempo di compilazione).

Se poi l'istanza effettiva fosse di tipo discendente (nel nostro caso di tipo Mannaro) questo non importa, poiché il compilatore ha già deciso, prima che il programma sia eseguito, quale sarà il metodo da attuare.

Occorrerebbe che il compilatore rimandasse a tempo di esecuzione la decisione di quale metodo invocare; sarebbe utile cioè che durante l'esecuzione il programma controlli il tipo effettivo dell'istanza associata alla variabile e quindi decida quale metodo invocare run-time.

METODI VIRTUALI

LA PAROLA CHIAVE VIRTUAL

Per dichiarare un metodo con un riferimento ritardato, Visual Studio dispone della parola riservata **virtual**, che induce il compilatore a rimandare la verifica delle invocazioni (sebbene ovviamente il compilatore debba verificare che il nome del metodo esista).

La parola chiave virtual può essere indifferentemente usata prima o dopo public e static.

LA PAROLA CHIAVE OVERRIDE

Le classi discendenti tuttavia devono ridefinire il metodo usando la parola chiave override, che significa scavalcamento. In effetti il metodo della classe derivata scavalca (anticipa) il metodo più generale e viene invocato al suo posto.

La parola chiave override può essere indifferentemente usata prima o dopo public e static.

ESEMPIO DI METODI VIRTUALI

Rifacendoci all'esempio precedente, il metodo andrebbe definito così:

VC#

```
public class Mostro {
  public virtual void Verso ()
  {
    MessageBox.Show("grugnito");
  }
}
```

```
public class Mannaro : Mostro {
  public override void Verso ()
  {
    MessageBox.Show("ululato");
  }
}
```

VC#

```
Mostro m = new Mostro ();
m.Verso();
```

```
Mannaro u = new Mannaro ();
u.Verso();
```

A questo punto la creazione casuale dei Mostri può essere lasciata identica a prima, ma quando si utilizza l'invocazione essa è solo apparentemente uguale alla precedente:

VC#

```
for (int i = 0; i<100; i++)
  avversari[i].Verso ();
```

in realtà il compilatore non lega subito la chiamata al metodo Verso ma la differisce e rimanda al tempo di esecuzione la scelta del metodo da invocare. L'esecuzione produrrà casualmente grugniti e ululati!

RIFERIMENTO RITARDATO

Per comprendere il meccanismo dei metodi virtuali è importante capire la differenza tra tempo di compilazione e tempo di esecuzione.

La compilazione è il momento in cui il programma viene analizzato sintatticamente e tradotto in un codice eseguibile; le invocazioni di metodi, le risoluzioni dei nomi, sono risolte in questo momento. Il compilatore controlla che il programma sia scritto correttamente, ottimizza le istruzioni, e predispone tutto per l'esecuzione; ma non lo esegue. Quando il compilatore ha terminato, il programma non è stato ancora eseguito e quindi non ha lavorato.

Quando il programma viene eseguito, si esegue in realtà il codice eseguibile restituito dal compilatore e non il codice sorgente (quello scritto dall'umano). Alcune istruzioni producono un effetto diverso da quanto ci si aspettava inizialmente, poiché ci sono dati letti dall'esterno o generati casualmente, che possono alterare l'esecuzione.

In generale, le chiamate ai metodi sono risolte a tempo di compilazione. Nel caso dei metodi virtuali tuttavia questo riferimento è ritardato e si rimanda al tempo di esecuzione il controllo del riferimento del metodo da invocare, determinato dal tipo d'istanza effettivamente riscontrato.

POLIMORFISMO

Il riferimento ritardato dei metodi da invocare, in base all'istanza effettivamente legata all'oggetto, consente di adattare dinamicamente l'esecuzione del programma sulla realtà effettuale che si riscontra al momento.

Questo meccanismo di adattamento è detto polimorfismo, ovvero «delle molte forme»; il polimorfismo è quindi un procedimento con cui il comportamento di una locazione di tipo oggetto modifica il suo comportamento conformandosi dinamicamente all'istanza di riferimento.

Gli esempi precedenti sono basati su un vettore di oggetti. In realtà l'adattamento può avvenire in molte occasioni. Per esempio, nell'ipotesi che Verso sia un metodo virtuale di Mostro, ridefinito (override) nelle classi derivate Mannaro e Vampiro:

VC#

```
Mostro m;               // m è una variabile di tipo Mostro
m = new Mostro ();
m.Verso ();             // m ha un comportamento da Mostro
m = new Mannaro ();
m.Verso ();             // m ha un comportamento da Mannaro
m = new Vampiro();
m.Verso ();             // m ha un comportamento da Vampiro
```

Un esempio importante di uso di polimorfismo può essere quello dei parametri di tipo oggetto. Per esempio nel metodo seguente:

VC#

```
public void Comportamento ( Mostro p )
{
  p.Verso ();
}
```

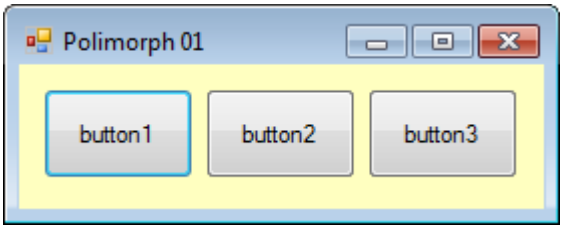
il parametro p è di tipo Mostro, ma potrebbe ricevere un passaggio di un oggetto derivato come nelle seguenti invocazioni:

VC#
```
Mostro m;               // m è una variabile di tipo Mostro
m = new Mostro ();
Comportamento ( m ) ;  // m ha un comportamento da Mostro
m = new Mannaro ();
Comportamento ( m ) ;  // m ha un comportamento da Mannaro
m = new Vampiro();
Comportamento ( m ) ;  // m ha un comportamento da Vampiro
```

ESERCITAZIONE

PROGETTO GUIDATO

- Si deve avviare Visual Studio e si deve preparare il Form1 come nella figura qui illustrata
- Si passi al codice e si dichiarino le seguenti classi:



VC#
```
        public class Uccello
        {
            virtual public void Verso()
            {
                  MessageBox.Show("cip cip" , "Uccello");
            }
        }
        public class Corvo : Uccello
        {
            override public void Verso()
            {
                  MessageBox.Show("cra cra" , "Corvo");
            }
        }
        public class Oca : Uccello
        {
            override public void Verso()
            {
                  MessageBox.Show("qua qua" , "Oca");
            }
        }
        //--- variabile globale
        Uccello u;
```

- Button1

VC#
```
            u = new Uccello();
            u.Verso();
```

- Button2

VC#

```
u = new Corvo();
u.Verso();
```

- Button3
-

VC#

```
u = new Oca();
u.Verso();
```

- Si esegua il progetto.

ESERCIZI

ESERCIZI

ESERCIZI SUL POLIMORFISMO

Esercizio 1) DATA E ORARIO

- Si deve implementare una classe Data che rappresenta una data del calendario Gregoriano; la data ha tre attributi privati: giorno, mese e anno.
- La data ha i seguenti costruttori pubblici:
  a. Senza parametri, crea la data 1/1/1900;
  b. Con tre parametri, crea la data secondo i parametri indicati;
- La data ha i seguenti metodi pubblici **virtuali**:
  c. ToString, che rende la data in forma di unica stringa;
  d. Bisestile, che rende true se l'anno è bisestile, falso altrimenti;
  e. Leggi, imposta la data secondo tre parametri in modo valore;
  f. Scrivi, rende la data secondo tre parametri in modo risultato;
  g. Domani, che incrementa la data e la porta al giorno successivo;
  h. Ieri, che decrementa la data e la porta al giorno precedente;
  i. Giorno, che rende la stringa del giorno della settimana (es. Domenica);
- Si deve implementare una classe DataOra che rappresenta un orario e una data del calendario Gregoriano; la classe è derivata da Data ma dichiara tre attributi privati ore, minuti e secondi;
- La classe deve ridefinire i precedenti costruttori pubblici e aggiungere il seguente:
  j. Con sei parametri, crea la data e l'orario secondo i parametri indicati;
- La classe deve definire l'override dei seguenti metodi pubblici **virtuali**:
  k. ToString, che rende data e orario in forma di unica stringa;
- La classe deve definire l'overload dei seguenti metodi pubblici:
  l. Leggi, imposta data e orario secondo tre parametri in modo valore;
  m. Scrivi, rende la data secondo tre parametri in modo risultato;

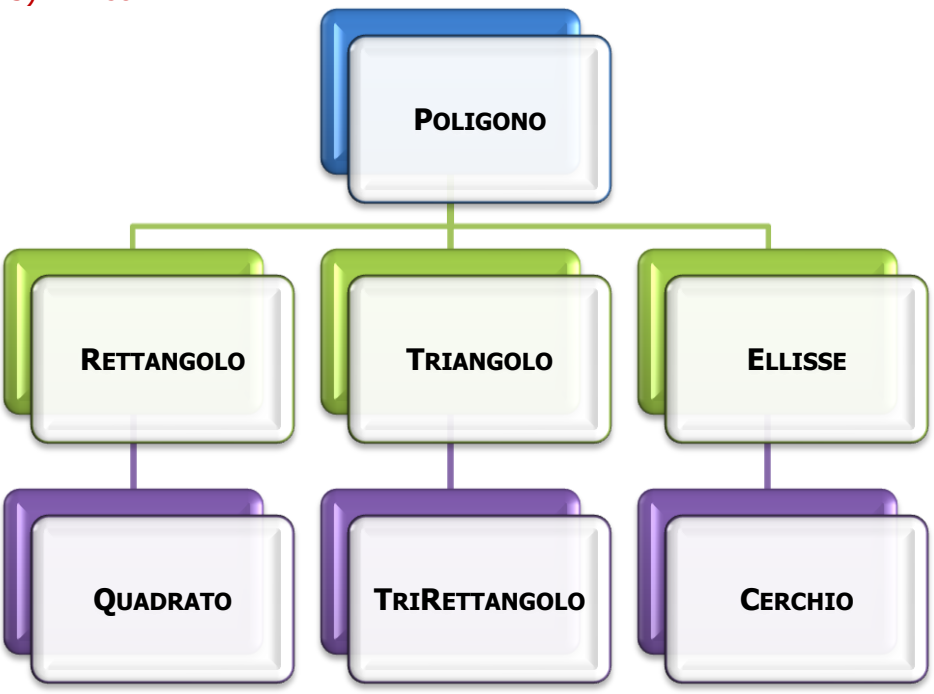
Esercizio 2) CLASSE RECIPIENTE

- Si deve implementare una classe Recipiente che rappresenta un contenitore di liquidi; la classe possiede l'attributo capacità e contenuto, entrambi interi;
- La classe ha i seguenti costruttori pubblici:
  n. Senza parametri, che crea un contenitore vuoto di capacità 1000 ml;
  o. Con un parametro K intero, che crea un contenitore vuoto di capacità di K ml;
- La classe ha i seguenti metodi pubblici virtuali:
  p. ToString, Get e Set, che svolgono i consueti compiti;
  q. Metti, con un parametro K intero, che aggiunge al contenitore K ml, purché non ecceda la capacità del recipiente;
  r. Togli, con un parametro K intero, che toglie K ml dal contenitore, purché esso li contenga (non può diventare meno che vuoto);
- Si deve implementare una classe Bicchiere derivata da Recipiente che rappresenta un contenitore di

liquidi, la cui capacità non può eccedere i 1000 ml;

- La classe deve ridefinire tutti i costruttori pubblici:
  s. Senza parametri, che crea un contenitore vuoto di capacità 100 ml;
  t. Con un parametro K intero, che crea un contenitore vuoto di capacità di K ml; se K eccede 1000 ml, è creato da 1000 ml esatti;
- La classe deve ridefinire i metodi pubblici virtuali:
  u. ToString, Get e Set, che svolgono i consueti compiti;

Esercizio 3) FIGURE PIANE



Lo studente predisponga le classi sopra illustrate e le fornisca dei seguenti attributi privati e metodi pubblici:

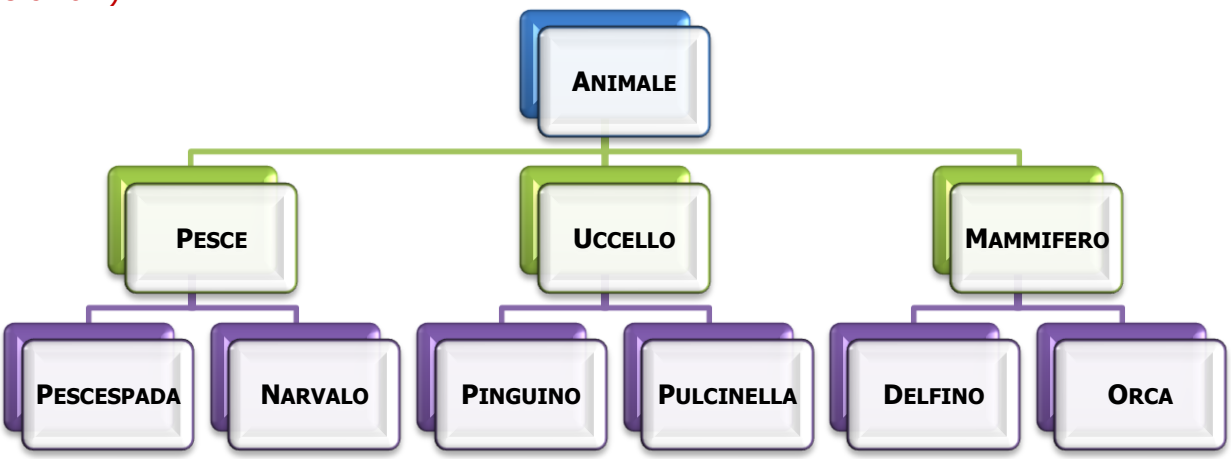
Lati (double) o Raggi (double)
Area(…)
Perimetro(…)
QuantiLati (…)
Metodi Set e Get che impostino gli attributi (tutti)

Lo studente ipotizzi quindi almeno due costruttori per ogni classe di cui:

Uno senza parametri che prepari l'oggetto in modo standard
Uno con parametri che inizializzi tutti gli attributi

Lo studente dichiari quindi una lista tipizzata di Poligoni e visualizzi in una listbox l'elenco delle loro aree:

Esercizio 4) ANIMALI



Lo studente predisponga le classi sopra illustrate e le fornisca dei seguenti attributi privati e metodi pubblici:

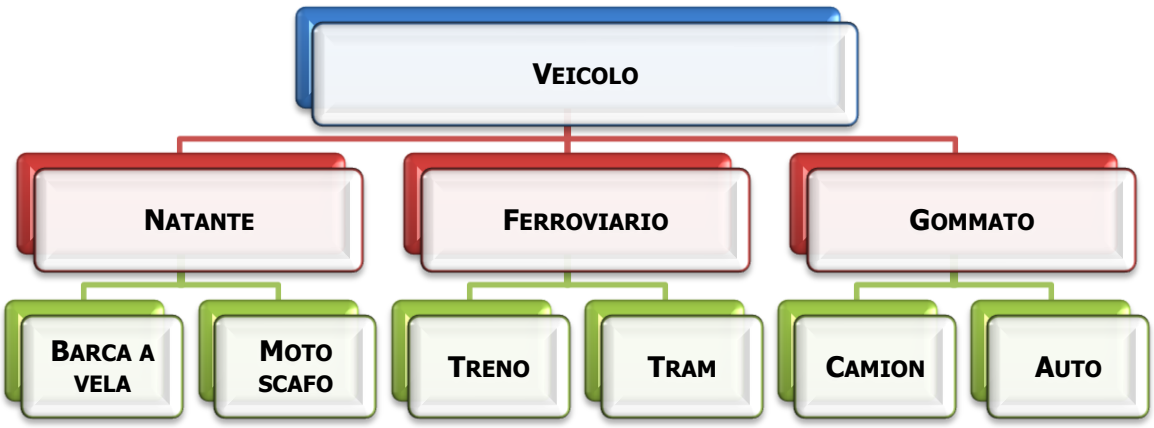
Specie (stringa), Gestazione (in giorni, intero), Riproduzione (in esemplari riprodotti, intero), Verso (stringa),
ToString (che rende la stringa con le caratteristiche dell'animale)
Genera (che rende un numero casuale inferiore alla sua Riproduzione)

Lo studente ipotizzi quindi almeno due costruttori per ogni classe di cui:

Uno con parametri che inizializzi tutti gli attributi
Uno senza parametri che prepari l'oggetto in modo standard

Lo studente dichiari quindi una lista tipizzata di Animali e visualizzi in una listbox l'elenco delle loro caratteristiche:

Esercizio 5) VEICOLI



Lo studente predisponga le classi sopra illustrate e le fornisca dei seguenti attributi privati e metodi pubblici:

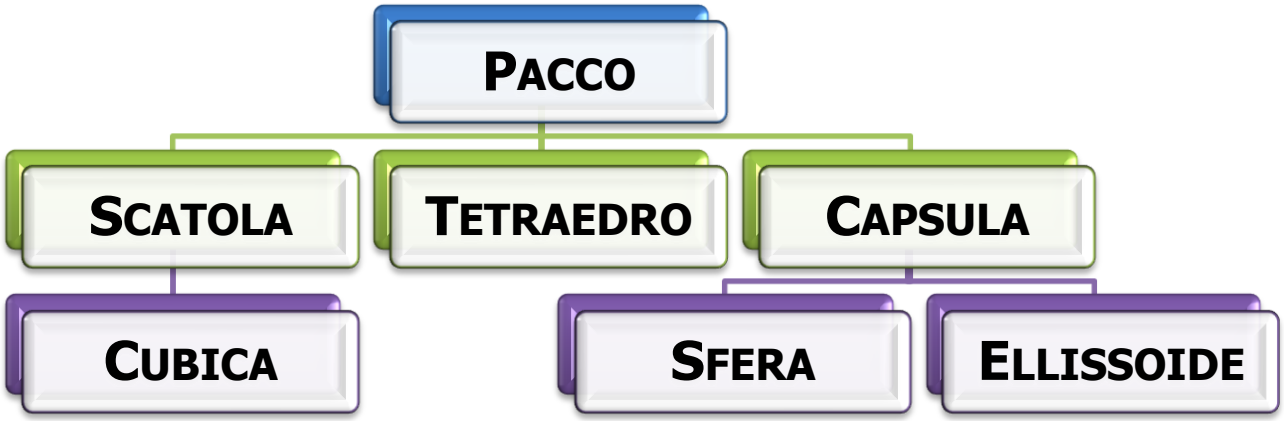
peso, numeroRuote, autonomia, marce, velocitàMax, caricoMax, numPasseggeri
ToString (che rende la stringa con le caratteristiche dell'oggetto)

Lo studente ipotizzi quindi almeno due costruttori per ogni classe di cui:

Uno con parametri che inizializzi tutti gli attributi
Uno senza parametri che prepari l'oggetto in modo standard

Lo studente dichiari quindi una lista tipizzata di Veicoli e visualizzi in una listbox l'elenco delle loro caratteristiche:

Esercizio 6) PACCHI



Lo studente predisponga le classi sopra illustrate e le fornisca dei seguenti attributi privati e metodi pubblici:

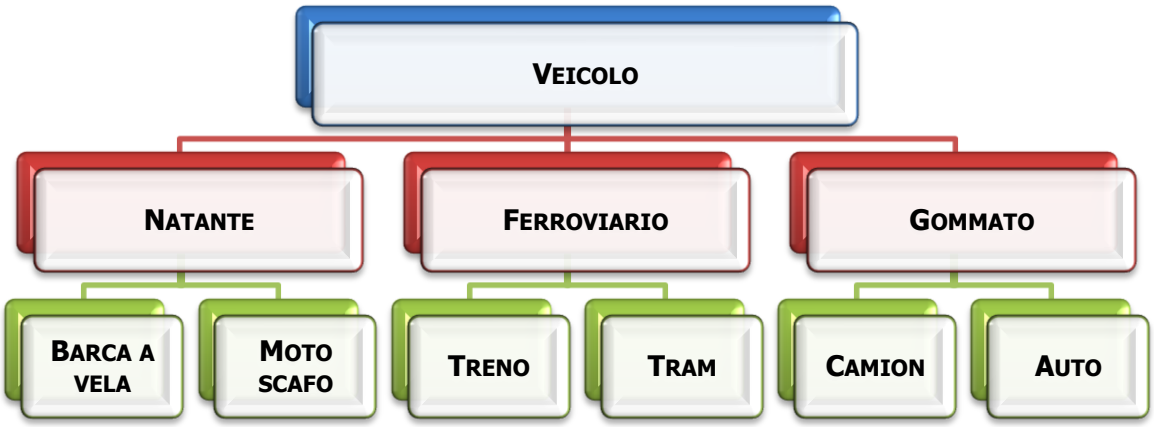
Tipo (nome della classe), misure (int, in cm),
Volume e Superficie laterale
ToString (che rende la stringa con le caratteristiche dell'oggetto, inclusi Volume e Superficie)

Lo studente ipotizzi quindi almeno due costruttori per ogni classe di cui:

Uno con parametri che inizializzi tutti gli attributi
Uno senza parametri che prepari l'oggetto in modo standard

Lo studente dichiari quindi una lista tipizzata di Pacchi e visualizzi in una listbox l'elenco del loro volume ordinato per capacità crescente:

Esercizio 7) VEICOLI



Lo studente predisponga le classi sopra illustrate e le fornisca dei seguenti attributi privati e metodi pubblici:

peso, numeroRuote, autonomia, marce, velocitàMax, caricoMax, numPasseggeri
ToString (che rende la stringa con le caratteristiche dell'oggetto)

Lo studente ipotizzi quindi almeno due costruttori per ogni classe di cui:

Uno con parametri che inizializzi tutti gli attributi
Uno senza parametri che prepari l'oggetto in modo standard

Lo studente dichiari quindi una lista tipizzata di Veicoli e visualizzi in una listbox l'elenco delle loro caratteristiche:

SOMMARIO