Laboratorio di Programmazione

Laboratorio 20/11/16

---

1 Autostrade, Auto, Pedaggi

Lo scopo dell'esercizio è realizzare un modello semplificato per la gestione di strade a pedaggio. Quindi le classi dovranno esporre metodi per il calcolo del pedaggio, della velocità media, etc. Le **classi** da realizzare sono le seguenti (dettagli nelle sezioni successive):

1. `Autostrada`: strada con pedaggio, introduce il concetto di "veicolo ammesso al transito" (deve avere una certa potenza) e di pedaggio
2. `Biglietto`: biglietto di ingresso, tiene traccia dell'orario di ingresso
3. `Auto`: veicolo

1.1 Specifica delle classi

Le classi (**pubbliche**!) dovranno esporre almeno i metodi e costruttori **pubblici** specificati, più eventuali altri metodi e costruttori se ritenuti opportuni. Gli attributi (campi) delle classi devono essere **privati**. per leggere e modificarne i valori, creare opportunamente, e solo dove necessario, i metodi di accesso (`set` e `get`). Se si usano classi che utilizzano tipi generici, si suggerisce di utilizzarne le versioni opportunamente istanziate (es. `ArrayList<String>` invece di `ArrayList`). Ogni classe deve avere il metodo `toString` che rappresenti lo stato delle istanze.

1.1.1 public class Autostrada

La classe deve prevedere la gestione del pedaggio, la gestione dell'accesso (accedono solo i veicoli sopra una certa potenza) e la gestione degli ingressi (tiene traccia di chi è dentro, calcola se sono stati superati i limiti, etc.).
Deve definire gli attributi: `int limite (Km/h), float lunghezza (Km)`; `float tariffaBase (euro/Km)`, `float potenzaMinimaPerAccedere (kW)` e un "contenitore" (a scelta) per tenere traccia dei veicoli entrati.
Deve definire almeno un costruttore che permetta di impostare gli attributi (adottare per i parametri lo stesso ordine in cui sono elencati sopra).
Deve inoltre definire i seguenti metodi (oltre ai get per gli attributi, NON implementare i set, basta il costruttore):

- `public float orePercorrenzaVelocitaCodice()` restituisce quante ore ci vogliono a percorrere tutta la lunghezza della strada andando alla velocità massima possibile (limite di velocità). Nota: il float restituito rappresenta ore e frazioni di ore, ad esempio 1.20 rappresenta un'ora e 12 minuti.

- `public float velocitaMediaDatoTempoPercorrenzaInSec(float percorrenza)` restituisce la velocità media in Km/h tenuta, dato un tempo di percorrenza effettivo espresso in secondi.
- `public boolean superatoLimiteDatoTempoPercorrenzaInSec(float percorrenza)` restituisce *true* se, dato il tempo di percorrenza (in secondi), si può supporre che il limite di velocità sia stato superato.
- `public float pedaggio(Auto v)` accetta solo auto con potenza superiore a potenzaMinima (se l'auto non ha sufficiente potenza, restituisce -1) e calcola il pedaggio (in euro) in funzione di:
  - tariffa base
  - lunghezza della strada
  - numero di assi del veicolo: fino a 3 assi si usa la tariffa base, oltre i 3 assi si usa 1.5*(tariffa base)
- `public Biglietto ingresso(Auto v)` se l'auto è ammessa (controllo su potenza), la lascia entrare ed emette un biglietto (vedi classe relativa) per il veicolo stesso; restituisce `null` se l'auto non è ammessa. Nota: java.lang.System.currentTimeMillis() permette di avere l'ora esatta espressa in millisecondi.
- `public float uscita(Auto v)` se l'auto non è presente tra i veicoli entrati (considerare il metodo `contains`), restituisce -1; se il tempo di percorrenza (calcolato in base all'ora attuale e all'ora di ingresso) è superiore o uguale a quello "legale" (rispettando il limite di velocità), fa uscire l'auto (la elimina dal "contenitore"), calcola il pedaggio e lo restituisce, altrimenti fa comunque uscire l'auto, ma restituisce il pedaggio incrementato di 100 euro. *Nota*: poiché il tempo di esecuzione del programma è di pochi millisecondi, anche il tempo di permanenza di un'auto in autostrada risulterà di pochi millisecondi, con la conseguenza che tutte le auto supereranno il limite di velocità. Modificate opportunamente il tempo di permanenza di un'auto in modo che sia possibile testare tutte le situazioni. Poiché si prevedete di testare il programma su un'autostrada di 50 km con un limite di velocità di 100 km/h, se l'esecuzione è di circa 10 millisecondi, in questo metodo moltiplicatela per 60.000 (per avere 10 minuti) e sommate 20-30 minuti, ai soli fini di avere tempi ragionevoli nel test.
- `public int quanteAuto()` restituisce il numero di auto attualmente in viaggio.
- `public float potenzaMedia()` calcola e restituisce la potenza media delle auto attualmente in viaggio.

1.1.2 public class Auto

Deve definire gli attributi: `String targa; int assi (numero di); float potenza (in kW)`. Inoltre deve avere un reference a Biglietto (vedi classe relativa) con i metodi set e get relativi. Implementa un costruttore che fissa il numero di assi a 2.

1.1.3 public class Biglietto

Deve definire un attributo `long timestamp` (per registrare l'orario di ingresso) e avere un reference a Auto, coi relativi set e get.
Suggerimento: per il timestamp usare `java.lang.System.currentTimeMillis()`, che restituisce l'ora corrente espressa in millisecondi.

1.1.4 public class Main

Creare un main che realizzi i seguenti:

1. istanzi una Autostrada (attributi: 50 Km lunghezza, 100 Km/h limite vel., 0.20 euro/Km pedaggio, 50 kW potenza minima per accedere)
2. istanzi una serie di auto prendendo i dati da un file di testo (vedi sotto) ridiretto su standard input
3. li immetta uno a uno nell'autostrada
4. stampi la situazione dell'autostrada dopo ogni ingresso
5. calcoli la potenza media dei veicoli che sono effettivamente entrati
6. li faccia uscire uno a uno dall'autostrada, stampando la situazione dopo ogni uscita

Formato del file di testo: TipoVeicolo;Targa;Potenza
Utilizzare il seguente contenuto:

Auto;PV77777;60
Auto;MI656565;200
Auto;MI43432;40

---

2 Raccomandazioni

Implementate costruttori e metodi rispettando tutte le specifiche (nome, tipo, parametri). Implementate i metodi/costruttori che non riuscite a sviluppare con una implementazione fittizia come la seguente:

```
public tipo nomeDelMetodo (listaParametri) {
   return val;
}
```

dove `val` è un valore fittizio del tipo opportuno (es.: `-1, false, ""`).
Si suggerisce quindi di dotare da subito le classi di tutti i metodi richiesti, implementandoli in modo fittizio, e poi di sostituire man mano le implementazioni fittizie con implementazioni che rispettino le specifiche.
