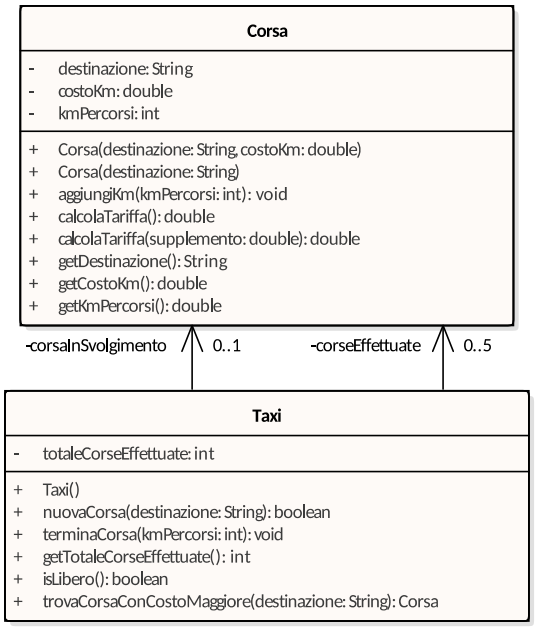
Dipartimento di Informatica Sistemistica e Comunicazione
Corso di Laurea in Informatica

Programmazione 2

3 Luglio 2019 – Recupero Primo Compitino

Testo parte di pratica

Si consideri un programma per gestire le corse che effettua un taxi. Una corsa ha un costo che dipende dal numero di chilometri percorsi e può venir addebitato un supplemento. Ogni corsa conclusa viene archiviata nell'insieme delle corse effettuate. Un taxi può effettuare al massimo 5 corse. Implementare le classi esattamente come rappresentate dal seguente diagramma UML. **Il diagramma include tutti e i soli metodi richiesti, compresi quelli di incapsulamento.**



La classe `TestTaxi` (già fornita e non mostrata nel diagramma delle classi) contiene un insieme di casi di test che devono essere fatti girare di volta in volta in modo da verificare la corretta realizzazione del programma. **Come requisito minimo per ottenere una valutazione positiva, lo studente deve garantire che la sua implementazione non presenti errori di compilazione e superi almeno 3 casi di test fra quelli dati.**

Classe `Corsa`:

- ✓ Rappresenta una corsa. È caratterizzata da una `destinazione`, da un costo al km (`costoKm`) e da un numero di km percorsi (`kmPercorsi`). Gli attributi `destinazione` e `costoKm` sono immutabili, mentre il valore di `kmPercorsi` può essere modificato solo con il metodo `aggiungiKm` (vedi seguito).
- ✓ Definisce un costruttore che inizializza con i valori passati in ingresso la destinazione (`destinazione`) e il costo al km (`costoKm`). I km percorsi (`kmPercorsi`) sono inizializzati a 0. Si assuma che i valori passati in ingresso siano validi (`destinazione` è una stringa diversa da `null` e dalla stringa vuota, e `costoKm` è un `double` non negativo).
- ✓ Definisce un ulteriore costruttore che inizializza la destinazione (`destinazione`) con il valore passato in ingresso, e imposta rispettivamente il costo al km (`costoKm`) pari a 0.5 e i km percorsi (`kmPercorsi`) a 0.
- ✓ Il metodo `aggiungiKm(int kmPercorsi)` aggiorna l'attributo `kmPercorsi` aumentandolo del valore in input, ma solo se il valore in input è positivo.
- ✓ Il metodo `calcolaTariffa()` calcola il costo della corsa moltiplicando il costo al km per i km percorsi.
- ✓ Il metodo `calcolaTariffa(int supplemento)` effettua l'overloding del precedente calcolando il costo con in più il supplemento.

Classe `Taxi`:

- ✓ Un taxi è caratterizzato dall'avere fino a massimo 5 corse svolte (associazione `corseEffettuate`) e dall'avere o meno una corsa in corso di svolgimento (associazione `corsaInSvolgimento`). La classe mantiene anche in un attributo privato il totale delle corse svolte (`totaleCorseEffettuate`). L'attributo `corsaInSvolgimento` viene di volta in volta inizializzato con una nuova corsa nel momento in cui il taxi accetta una nuova corsa e viene impostato a `null` ogni volta che la corsa termina.

  NB: Se l'attributo `corsaInSvolgimento` è diverso da `null`, vuol dire che è in corso una corsa e quindi il taxi non è libero. Se invece è `null`, vuol dire che il taxi non sta facendo alcuna corsa e quindi è libero.
- ✓ Definisce un costruttore di default che inizializza opportunamente tutti gli attributi.
- ✓ Il metodo `nuovaCorsa(String destinazione)` permette di iniziare una nuova corsa se sono verificate le seguenti tre condizioni: 1) il taxi è libero (non sta già svolgendo una corsa, 2) la `destinazione` è una stringa non vuota, e 3) il taxi ha svolto meno delle 5 le corse possibili. Se le tre condizioni non sono verificate, il metodo non fa nulla e restituisce `false`. Se la corsa invece può essere accettata, il taxi non è più libero: il suo attributo `corsaInSvolgimento` viene inizializzato con una nuova corsa con destinazione `destinazione`. Viene quindi restituito `true`.
- ✓ Il metodo `isLibero()` restituisce `true` se il taxi non sta svolgendo alcuna corsa (è libero), `false` se il taxi sta effettuando una corsa (non è libero).
- ✓ Il metodo `getTotaleCorseEffettuate()` restituisce il numero totale di corse già concluse.
- ✓ Il metodo `terminaCorsa(int kmPercorsi)` aggiunge i km percorsi alla corsa in svolgimento, la aggiunge all'array `corseEffettuate`, incrementa il numero di corse concluse e rende il taxi. Se non c'è nessuna corsa in svolgimento, il metodo non fa nulla.
- ✓ Il metodo `trovaCorsaConCostoMaggiore(String destinazione)` cerca all'interno dell'array `totaleCorseEffettuate` e restituisce la corsa che ha costo maggiore tra tutte quelle con la destinazione indicata in input. Nel caso non fosse presente alcuna corsa per la destinazione in input, il metodo restituisce `null`.