Dipartimento di Informatica Sistemistica e Comunicazione
Corso di Laurea in Informatica

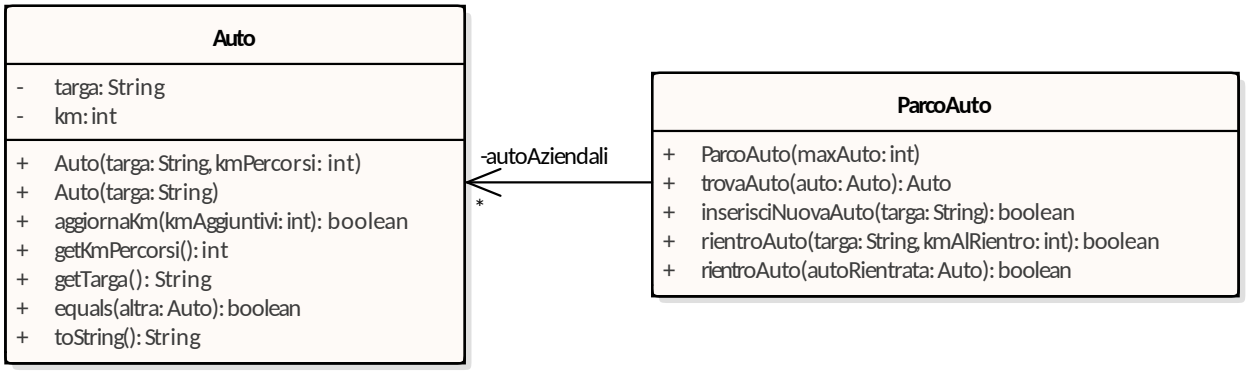
Programmazione 2

29 Aprile 2019 – Primo Compitino

Testo parte di pratica

Si consideri un programma per gestire il parco auto che un'azienda mette a disposizione dei dipendenti. Il sistema memorizza e gestisce targa e chilometri per ogni auto.

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 `TestParcoAuto` (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 `Auto`:

- ✓ Rappresenta un'auto aziendale. È caratterizzata dalla `targa` e dal numero di chilometri (`km`) percorsi, entrambi accessibili in lettura con metodi *get*. L'attributo `targa` è immutabile.
- ✓ Definisce un costruttore che inizializza entrambi gli attributi. Si assuma che i valori passati in ingresso siano validi (`targa` è una stringa diversa da `null` e `km` è un intero non negativo)
- ✓ Definisce un ulteriore costruttore che inizializza la targa con il valore passato in ingresso, e imposta i chilometri a zero.
- ✓ Il metodo `aggiornaKm(int kmAggiuntivi)` controlla che il parametro passato `kmAggiuntivi` sia un numero non negativo e, se è così, aggiorna i chilometri percorsi incrementandoli corrispondentemente e quindi ritorna `true`. Altrimenti, se il parametro `kmAggiuntivi` è negativo, mantiene invariato il valore dei chilometri percorsi e restituisce `false`.
- ✓ Due oggetti `Auto` sono uguali se hanno la medesima `targa`.
- ✓ il metodo `toString()` restituisce una stringa con i dati di targa e chilometri percorsi

Classe `ParcoAuto`:

- ✓ Rappresenta l'insieme delle auto disponibili per i dipendenti (associazione `autoAziendali`).
- ✓ Definisce un costruttore che inizializza il numero massimo di auto che potranno essere inserite nel sistema. Si assuma che il valore passato in ingresso sia valido (`maxAuto` è un intero maggiore di zero)
- ✓ Il metodo `trovaAuto(Auto auto)` cerca se all'interno del parco auto esiste un'auto uguale a quella passata in ingresso. Se esiste, la restituisce (quella nel parco auto uguale ad `auto`). Se invece l'oggetto in ingresso `auto` è `null`, o se non esiste un'auto uguale, il metodo restituisce `null`.
- ✓ Il metodo `inserisciNuovaAuto(String targa)` inserisce una nuovo oggetto di tipo `Auto` con la targa specificata e chilometri impostati a zero, ma solo se nel parco auto non è già presente un oggetto uguale (ovvero un'auto con la medesima targa), e se c'è uno spazio libero in cui aggiungerlo. Restituisce

`true` in caso di inserimento, `false` in caso contrario. Nel caso `targa` sia `null`, il metodo restituisce `false`.

- ✓ Il metodo `rientroAuto(Auto autoRientrata)` aggiorna il chilometraggio di un'auto al rientro dopo un'uscita. Se `autoRientrata` corrisponde a una delle auto presenti nel parco auto `autoAziendali`, il metodo aggiorna il chilometraggio dell'auto presente nel parco auto aggiungendo i km percorsi (la differenza fra i chilometri al rientro e il numero di chilometri attualmente assegnati all'auto). Se l'aggiornamento restituisce `true`, il metodo `rientroAuto` restituisce anch'esso `true`, altrimenti restituisce `false`.
- ✓ Il metodo `rientroAuto(String targa, int kmAlRientro)` effettua l'overloading del precedente, aggiornando il chilometraggio dell'auto come il metodo che overlodato, ma usando la `targa` e i `kmAlRientro` come dati in input.