Relazione Progetto Ingegneria Del Software

nome: Bruno **cognome:** Montalto **matricola:** 1000016231
Università degli Studi di Catania

• Introduzione

Si vuole realizzare un software che permetta di simulare il comportamento di un automobile al variare del suo stato (modalità di guida, motore, passeggeri all'interno, velocità, etc…).

Ho suddiviso la creazione di tale software nei seguenti passi:

1. Rappresentazione dei dati
2. Interfaccia utente
3. Code coverage

• Funzionalità Aggiuntive

Oltre alle funzionalità richieste dalla consegna, sono state implementate le seguenti:

- Modalità di guida Elettrica e Normale.
- Menù di creazione ed eliminazione di passeggeri predefiniti.
- Presenza del serbatoio di benzina e della batteria.
- Il peso dei passeggeri influisce sull' accelerazione.

• Scelte di Progettazione

I Design Patterns utilizzati in questo progetto sono i seguenti:

- **Singleton** per permettere la creazione di una sola istanza dell'auto.
- **State** per gestire le diverse modalità di guida.
- **Facade**

1. Rappresentazione dei dati

Sono state realizzate le seguenti classi:

- `Passenger`

  Rappresenta i passeggeri: contiene un nome, rappresentato come `String` e il loro peso, memorizzato come `int`.

- `Car`

  Rappresenta l'auto: contiene lo stato del motore, memorizzato come `boolean`, la velocità, il livello di benzina e di batteria memorizzati come `double`, la modalità di guida, memorizzata come `DriveMode` e la lista dei passeggeri a bordo, memorizzata come `ArrayList<Passenger>`.

  La classe presenta inoltre i seguenti metodi:
  - `public static void getInstance()`
  - `public static void getInstance()`
  - `public void resetState()`
  - `public int toggleEngine()`
  - `public int getTotalWeight()`
  - `public int setMode(DriveModeEnum mode)`
  - `public int canAccelerate()`
  - `public double accelerate(double amount)`
  - `public void brake()`
  - `public boolean isFull()`
  - `public int addPassenger()`
  - `public Passenger removePassenger(int index)`
  - `public boolean removePassenger(Passenger p)`
  - `public boolean refuel()`
  - `public boolean charge()`
  - `public void printPassengers()`
  - `public void printState()`

  Per ognuno dei seguenti metodi ho creato una descrizione nel javadoc.

- `DriveMode`

  È una classe astratta. Rappresenta una modalità di guida generica: contiene un riferimento all'auto che si trova in tale modalità, memorizzato come `Car`, la potenza del motore e la velocità massima di tale modalità, memorizzate come `double`.

La classe presenta inoltre i seguenti metodi:

- `public abstract void accelerate(double acceleration, double amount)`
- `public abstract boolean isLegal()`

Per ognuno dei seguenti metodi ho creato una descrizione nel javadoc.

La classe astratta `DriveMode` è estesa dalle seguenti classi:

- `ComfortableMode`
  Rappresenta la modalità di guida comoda. Permette di raggiungere la velocità massima di 50 Km/h.

- `NormalMode`
  Rappresenta la modalità di guida normale. Permette di raggiungere la velocità massima di 120 Km/h.

- `ElectricMode`
  Rappresenta la modalità di guida elettrica. A differenza delle altre consuma la batteria, oltre a una piccola quantità di benzina. Permette di raggiungere la velocità massima di 90 Km/h. Se tale soglia viene superata, o se il livello di batteria è inferiore al 10%, passerà automaticamente alla modalità di guida normale.

- `SportMode`
  Rappresenta la modalità di guida sportiva. Possibile solo quando vi è un solo passeggero a bordo. Permette di raggiungere la velocità massima di 200 Km/h.

Tutte le modalità di guida elencate si occupano di spegnere il motore dell'auto qualora la benzina dovesse finire.

2. Interfaccia Utente

Per permettere al client di interagire con la simulazione ho creato la classe `SimulationManager`. Quest'ultima si occupa di stampare all'utente un menù di funzionalità ognuna associata a un numero, attendendo poi la scelta dell'utente come input da tastiera, fino alla chiusura del programma.

Le opzioni sono le seguenti:

0: stampa lo stato attuale della macchina (motore, velocità, modalità di guida, benzina, batteria, passeggeri a bordo)

1: Stampa tutti i passeggeri predefiniti, esclusi quelli che si trovano già dentro l'auto.

2: Permette di aggiungere uno dei passeggeri predefiniti a bordo.

3: Permette di rimuovere un passeggero dall'auto.

4: Dà accesso a un menù annidato di creazione ed eliminazione di passeggeri predefiniti.

5: Permette di cambiare modalità di guida.

6: Permette di accendere o spegnere il motore.

7: Permette di accelerare fornendo una percentuale che rappresenta con quanta forza viene premuto l'acceleratore.

8: Permette di frenare.

9: Permette di fare rifornimento di benzina e/o di ricaricare la batteria.

10: Riporta lo stato della macchina a quello iniziale.

3. Code Coverage

Ho realizzato 20 test del software arrivando a una code coverage del 100%, escludendo la classe `SimulationManager` che ha il solo scopo di permettere all'utente di interagire con l'API.

Tra i test più significativi abbiamo:

- `@Test public void accelerationShouldNotChange()`
  Calcola l'accelerazione dell'auto dopo aver aggiunto due passeggeri dai rispettivi pesi di 70 e 50 Kg, dopo di che sostituisce i due passeggeri con altri due la cui somma dei pesi è uguale alla precedente (59 e 61), calcola nuovamente l'accelerazione e verifica che quest'ultima sia uguale alla precedente.

- `@Test public void testModeAcceleration()`
  Per ogni modalità verifica se accelerando con il serbatoio di benzina pieno il motore rimane acceso e,viceversa, se accelerando con il serbatoio praticamente vuoto il motore si spegne. Verifica anche che il valore ritornato dal metodo `accelerate` sia effettivamente l'accelerazione subita.
  Inoltre per la modalità elettrica verifica se superare la velocità massima o arrivare sotto la soglia minima di batteria(10%) comporta il passaggio alla modalità di guida normale.

- `@Test public void weightShouldBeCarWeight()`
  Verifica se la funzione che calcola il peso totale dell'auto, in assenza di passeggeri, ritorna il peso stesso dell'auto.

- `@Test public void shouldNotAddPassenger()`
  Verifica tutti i casi in cui non dovrebbe essere possibile aggiungere un nuovo passeggero. Prima di tutto verifica che non sia possibile aggiungere un nuovo passeggero quando l'auto è al completo. Poi, dopo aver liberato un posto, verifica che non può essere aggiunto un passeggero qualora l'auto fosse in movimento o qualora il passeggero fosse già a bordo.
  Verifica anche il caso in cui viene lanciata un'eccezione

`IllegalStateException` poichè la modalità di guida corrente non permette nuovi passeggeri.

- `@Test public void shouldNotRemovePassenger()`
  Verifica tutti i casi in cui non dovrebbe essere possibile rimuovere un passeggero per entrambi i metodi `RemovePassenger`.
  Verifica che se si prova a rimuovere un passeggero quando l'auto è in movimento, esso sarà ancora dentro l'auto.
  Verifica poi se viene lanciata un'eccezione `IllegalStateException` nel caso in cui si provi a rimuovere il passeggero nella modalità sport, oppure un eccezione `IndexOutOfBoundsException` nel caso in cui l'indice selezionato non esistesse.