Corso di Laurea in Ingegneria e Scienze Informatiche – Università di Bologna

Corso di Programmazione ad Oggetti – a.a. 2013/2014

Relazione relativa allo sviluppo dell'applicazione "Cars Manager"

A cura di Cianci Canio, Modugno Alberto

1 - Analisi del problema

L'applicazione sviluppata permette la gestione delle auto di un utente, tenendone sotto controllo i principali costi. "Cars Manager" permette quindi di inserire la propria auto, specificandone marca, modello, targa, anno di immatricolazione e, facoltativamente, una foto.
Per ogni auto è possibile aggiungere i rispettivi costi, suddivisi in costi di amministrazione, carburante e manutenzione. Con i costi così inseriti si possono osservare delle statistiche su quanto una determinata auto influisca sul portafogli dell'utente, mostrando il costo medio settimanale, mensile e annuale.
Ulteriore funzionalità è la possibilità di registrare il luogo in cui è stato parcheggiato il veicolo per poi successivamente avviare il navigatore verso la meta salvata in precedenza.

L'applicazione è di interesse per l'utente a cui piace tenere sotto controllo i costi che quotidianamente vengono affrontati, ma anche per chi per motivi di "archiviazione" voglia conservare uno storico dei costi per confrontare tra i mezzi in suo possesso qual è il più costoso o il più utilizzato.

Funzionalità offerte dall'applicazione:

- **Aggiunta di un'auto:** inserimento di un'auto specificandone i dati principali.
- **Modifica ed eliminazione auto:** modifica di tutti i dati principali di un'auto oppure eliminare completamente il veicolo dal database locale.
- **Aggiunta di un costo:** inserimento di un costo relativo ad un auto esistente, specificandone i dettagli principali.
- **Visualizzazione costi:** interfaccia a interattività limitata che mostra i costi già registrati per un'auto.
- **Eliminazione di un costo:** attraverso la stessa interfaccia appena descritta si può eliminare un costo.
- **Visualizzazione statistiche:** interfaccia che permette di visualizzare i costi di un'auto separati per categoria, e mostrati in costo medio settimanale/mensile/annuo.
- **Registrazione di un parcheggio, e successiva navigazione:** aggiunta della posizione in cui è stata parcheggiata l'auto, e navigazione all'ultimo punto di parcheggio salvato.

2 - Progettazione architetturale

2.1 - Organizzazione dei package

Il progetto è stato suddiviso in modo da avere due macro-package:

- model
- view

così come da tradizione Android. Questa suddivisione infatti separa il codice destinato alle interfacce e, in generale, a ciò che percepisce l'utente (posti all'interno del package "view"), dalla parte di codice che ha come scopo l'elaborazione e la modifica dei dati (all'interno del package "model").

All'interno del package model troveremo diversi packages, uno per ogni categoria di classe che sono state necessarie sviluppare durante lo sviluppo dell'applicazione.

2.2 - Gestione degli oggetti

L'applicazione gestisce tre tipi principali di dati: veicoli, costi e parcheggio. Per ognuno di questi oggetti sono state effettuate scelte implementative simili tra di loro. Veicoli, costi e parcheggi vengono infatti salvati su un file di testo diverso in base alla tipologia. Successivamente accedendo a questi file vengono generate le rispettive "ListView" attraverso adpter specifici,

L'oggetto più importante è sicuramente l'oggetto di tipo veicolo. Quest'ultimo infatti, essendo il punto cardine dell'applicazione, contiene quelle che sono le informazioni principali, tra cui la targa. La targa di un veicolo è gestita come identificatore univoco, e, associato ad ogni costo parcheggio permette di selezionare e suddividere questi ultimi in base al veicolo stesso.

È possibile eliminare veicoli e costi. Nel caso in cui ad essere eliminato sia un veicolo allora, attraverso la sua targa, si andranno ad eliminare tutti i costi e i parcheggi associati per "pulire" i file da informazioni superflue.

I dettagli di un veicolo possono essere modificati per correggere errori dell'utente durante un primo inserimento. Nel caso però in cui a cambiare sia proprio la targa, allora tale informazione verrà cambiata anche nei file di testo dei costi e dei parcheggi per evitare di avere informazioni inutilizzabili.

2.3 - Pattern

Singleton: pattern singleton utilizzato nell'implementazione di ogni FileManager.

Adapter: pattern adapter utilizzato nella compilazione di ogni ListView dinamica.

3 - Suddivisione del lavoro

Il progetto è stato eseguito con le seguenti suddivisioni tra i due membri, sebbene in linea di massima sia stato svolto attraverso una solida cooperazione, soprattutto nelle scelte implementative:

- **Cianci Canio:** gestione dei veicoli, comprese interfacce per l'inserimenti delle auto, interfaccia per la visualizzazione in una ListView, classi per la modellazione dei dettagli e per il salvataggio e la lettura da file, nonché tutto il necessario per modificare le informazioni dell'oggetto stesso. Modellazione delle informazioni riguardo i costi per generare le relative statistiche in base a periodi di visualizzazione e tipologia di costo.
- **Modugno Alberto:** gestione dei costi, comprese interfacce per visualizzarne i dettagli separati in base al tipo, classi per la modellazione dei dettagli e per il salvataggio e la lettura da file. Sviluppo e cura delle interfacce.
- **Parte in comune:** salvataggio e lettura parcheggio di un veicolo da file, sviluppo scheletro dell'applicazione.

3.1 - Progettazione - Cianci Canio

Vehicle, VehicleImpl: interfaccia e relativa implementazione per la modellazione di un veicolo. Consiste, oltre ai metodi "getter" per ogni campo dell'oggetto, di due costruttori. Questo perché l'utente a propria descrizione può scegliere se scattare una foto del proprio veicolo o se farlo più tardi, e in base a questa scelta verrà generato un veicolo dall'uno o dall'altro costruttore.

FileManagerVehicle: classe che permette di salvare su file le informazioni di un veicolo. Estende e specializza la classe "FileManager" che contiene i metodi in comune tra le tre classi che salvano su file. Nello specifico con l'ausilio di questa classe si può salvare su file l'oggetto veicolo come insieme di stringhe, leggere le stringhe già presenti su file, modificarle tali valori e eliminarle insieme all'immagine. Nonché ottenere un singolo veicolo in base alla targa desiderata e controllare se un veicolo che si vuole aggiungere non abbia come targa una già presente sul file.

VehicleAdapter: personalizzazione del pattern adapter per rappresentare liste di veicoli in una ListView.

Statistic: classe che genera le statistiche in base ai costi contenuti nel file di testo di competenza, selezionando solo quelli relativi all'auto in questione, al tipo di costo di interesse e calcolate in base ad un periodo di competenza che si vuole osservare.

Activity relative alle funzionalità: interfacce Android delle funzionalità sopra descritte che richiedevano interazione da parte dell'utente.

3.2 - Progettazione – Modugno Alberto

Cost, CostImpl: interfaccia e relativa implementazione per la modellazione di un costo. Consiste, oltre ai metodi "getter" per ogni campo dell'oggetto, di più costruttori. Questo perché l'utente può inserire diversi tipi di costi, e in base al tipo verrà generato un costo dall'uno o dall'altro costruttore.

FileManagerCost: classe che permette di salvare su file le informazioni di un costo. Estende e specializza la classe "FileManager" che contiene i metodi in comune tra le tre classi che salvano su file. Nello specifico con l'ausilio di questa classe si può salvare su file l'oggetto costo come insieme di stringhe, leggere le stringhe già presenti su file e modificare tali valori. Nonché ottenere un singolo costo in base alla targa desiderata.

CostAdapter, FuelCostAdapter: personalizzazione del pattern adapter per rappresentare liste di costi nelle tre ListView relative ai costi.

Principali Activities: interfacce Android delle funzionalità sopra descritte che richiedevano interazione da parte dell'utente.

3.3 - Progettazione comune

Scheletro dell'applicazione: insieme abbiamo visto tutte le parti dell'applicazione necessarie a far funzionare le parti sviluppate in singolo, nonché la struttura grafica e funzionale dell'applicazione.

Parking: classe che gestisce il parcheggio.

FileManagerParking: classe che permette di salvare su file le informazioni di un parcheggio. Estende e specializza la classe "FileManager" che contiene i metodi in comune tra le tre classi che salvano su file. Nello specifico con l'ausilio di questa classe si può salvare su file l'oggetto parcheggio come insieme di stringhe, leggere le stringhe già presenti su file e modificare tali valori.

4 – Note

4.1 - Testing

L' applicazione è stata testata per l'intero sviluppo su un LG Nexus 5 (Lollipop 5.1.1) e un Samsung Galaxy Note 3 (Lollipop 5.0.1).

Api minime richieste: 16

4.2 – Librerie Esterne

Nello sviluppo dell'applicazione sono state usate librerie esterne quali:

- ***MaterialDrawer***: per realizzare il drawer a scorrimento laterale
- ***FloatingActionButton***: per realizzare i tasti di aggiunta auto e aggiunta costi flottanti
- ***Picasso***: per gestire le immagini circolari e non, nell'applicazione

4.3 – Importazione su Eclipse

L'applicazione è stata interamente sviluppata sull'IDE Android Studio, ciò rende difficoltosa l'importazione su Eclipse.

Per importarlo quindi abbiamo scaricato un tool dal marketplace di Eclipse (Gradle Ide Pack 3.6.x+0.17), tramite il menu File->Import si seleziona 'Gradle Project', il percorso del progetto e poi si clicca su 'Build Model', si seleziona la root del progetto e tramite il bottone Finish la procedura è completata.

4.4 - Problemi

Dopo svariati tentativi con ogni genere di tool, non siamo riusciti a esportare i diagrammi delle classi UML a causa dell'errore "resource does not exists" persistente.