[21/22] 38090-MOD2 - PROGRAMMAZIONE AVANZATA

Maffeis Isaac 1041473

Relazione di progetto: Java

1. Introduzione

Per il progetto nel linguaggio di programmazione Java mi sono concentrato nel realizzare un'applicazione che avesse implementato tutti i costrutti studiati a lezione.
In particolare mi sono focalizzato sui concetti di ereditarietà, classi astratte, interfacce, eccezioni, gestione degli array, programmazione generica, visibilità, binding dinamico, sottotipazione, enumerazione, implementazione dei design pattern quali Singleton, Facade e Visitor.

L'applicazione realizzata si occupa di gestire un'agenzia di viaggi, nello specifico in primo luogo permette a un dipendente della agenzia di aggiungere pacchetti di viaggio, in secondo luogo da la possibilità a un cliente di visionarli in base alle proprie esigenze e prenotare quello che trova più adatto.

È presente una parte di test dove si simula il comportamento dei due attori, facendo varie prove e una parte interattiva da console nella quale si possono testare tutti i vari casi.

2. Caratteristiche

2.1 Funzionalità

L'applicazione è realizzata per semplificare il lavoro di un'agenzia di viaggi, lo scopo principale è quello di permettere al cliente di poter trovare velocemente una soluzione di viaggio corrispondente con le proprie esigenze e prenotarla.
Il cliente deve fornire all'applicazione, oltre che le proprie generalità per registrarsi, se desidera fare un viaggio al mare, in montagna oppure in città, il giorno in cui desidera partire, quello in cui vuole tornare e la città di partenza, successivamente gli viene presentata una lista di soluzioni di pacchetti di viaggio, ordinate per prezzo. Il Cliente a questo punto può decidere se procedere con la prenotazione, rifare la procedura con altre scelte oppure terminare la ricerca senza prenotare. In caso sia soddisfatto da una delle scelte proposte può selezionarla, aggiungere eventuali extra e procedere con la prenotazione.

Il manutentore (dipendente dell'agenzia) avvia il programma, caricando la lista delle città disponibili e la lista dei pacchetti che l'agenzia offre e gestisce la lista delle prenotazioni dei clienti.

Il programma deve quindi gestire le interazioni del manutentore e del cliente, creare le liste delle città disponibili e dei pacchetti di viaggio offerti, filtrare queste liste in base ai vincoli del cliente e generare una lista contenente tutte le soluzioni di viaggio da proporre al cliente. Una volta effettuata la prenotazione deve conservarla in una lista delle prenotazioni a cui avrà accesso solo il manutentore.

Funzionalità aggiuntive: l'agenzia di viaggio offre il 20% di sconto per tutti gli studenti, propone in fase di prenotazione il noleggio di un mezzo che varia in base alla destinazione (macchina se in

città, moto d'acqua se al mare, e-bike se in montagna), mette a disposizione su richiesta una guida di viaggio personalizzata per destinazione.

3. Composizione

3.1 Packages

L'applicazione Java è stata realizzata su 6 packages diversi, in particolare:

- collections: classi che si occupano di gestire le istanze degli oggetti città, pacchetto di viaggio, prenotazione e le strutture dati in cui immagazzinarle, lista delle città, lista dei pacchetti di viaggio e lista delle prenotazioni, gestisce inoltre con una classe apposita la comunicazione con oggetti esterni al package.
- exceptions: sottoclassi della superclasse Exception che permettono al programma di lanciare delle eccezioni controllate.
- maintenance: classi che permettono al manutentore di agire sul programma
- support: classi di supporto, sono presenti le classi per ordinare un array e per trovare un elemento in una lista ordinata.
- testing: classi che permettono di effettuare dei test sull'applicazione
- user: classi che gestiscono l'interazione con l'utente

3.2 Strutture dati

Per gestire le città, i pacchetti di viaggio e le prenotazioni sono state scelte le seguenti strutture di dati:

- ArrayList<E>: permette di rappresentare sequenze di oggetti di lunghezza variabile, ciascun oggetto in un'istanza di ArrayList viene identificato da un numero intero, detto indice, che ne indica la posizione. Tramite questa struttura vengono rappresentate le sequenze di pacchetti di viaggio e prenotazione, inoltre anche la lista delle soluzioni di viaggio disponibili in base ai vincoli del cliente viene realizzata tramite un ArrayList, in questo caso al momento della creazione viene scelto il tipo corretto in base alla destinazione (città di mare, città di montagna o città).
  Per avere una lista ordinata si serve della classe generica QuickSort, mentre per ricercare un elemento nella rispettiva lista ordinata utilizza la classe BinarySearch, sempre generica.
- TreeMap<K,V>: rappresenta una successione ordinata di elementi (chiave, valore), dove la chiave individua univocamente il valore associato e deve essere un oggetto confrontabile. Tramite questa struttura viene rappresentata la lista delle città, ordinata in base al codice univoco che ha l'oggetto città e ordinata in ordine alfabetico rispetto al codice città (String).

3.3 Tipi Enumerativi

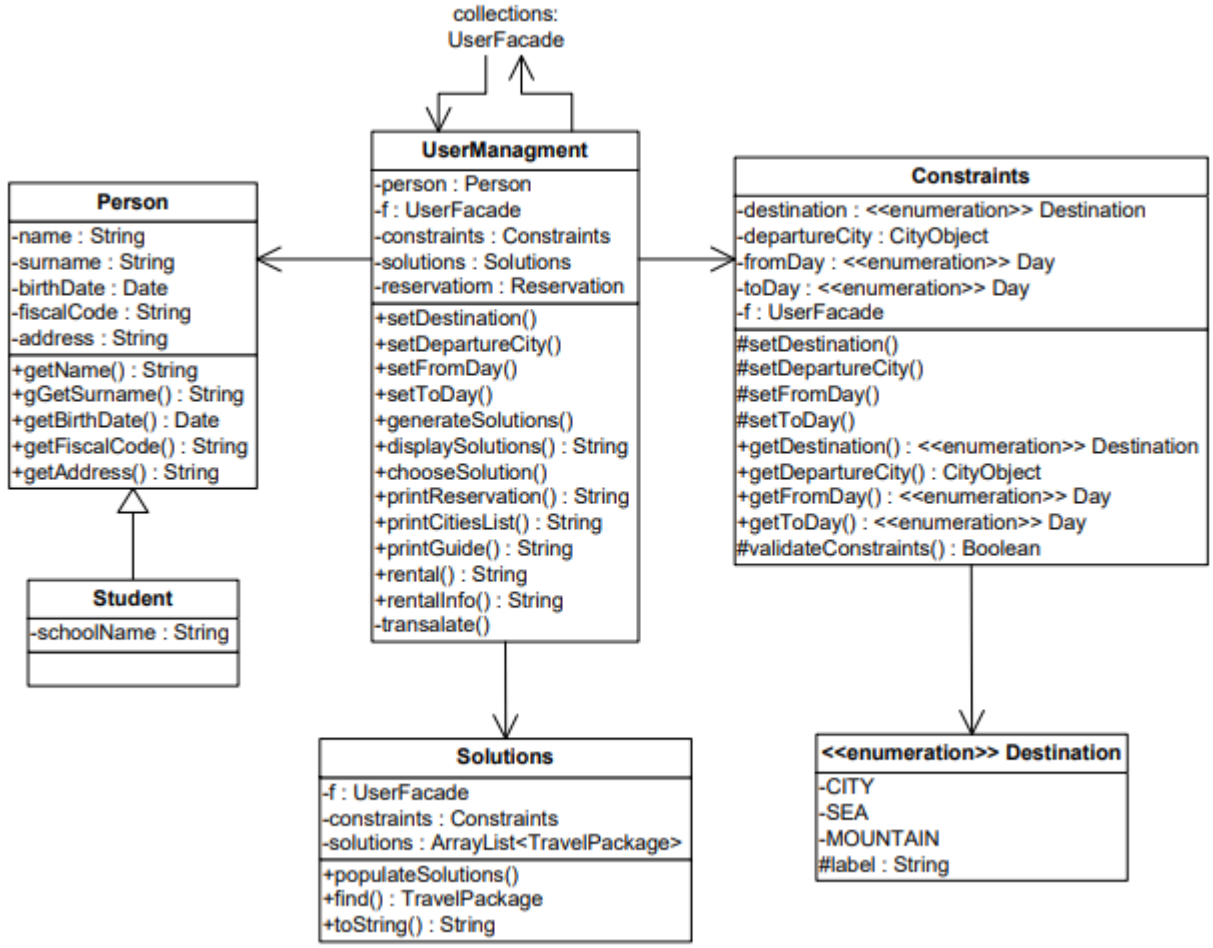
I tipi enumerativi permettono di definire tipi che richiedono un numero limitato di valori specifici. In questa applicazione vengono definiti tre tipi enumerativi:

- Day: giorni della settimana
- Destination: Destinazione della vacanza (città di mare, città di montagna o città).
- Rent: Mezzo da affittare durante la vacanza comprensivo di costo come etichetta.

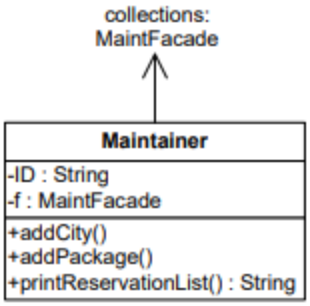
3.4 Diagramma delle Classi

Di seguito in figura è rappresentato il diagramma delle classi realizzato in UML, suddiviso per package:

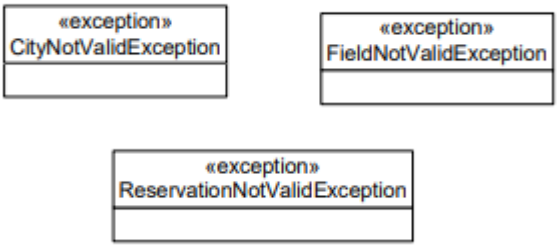
Package user:



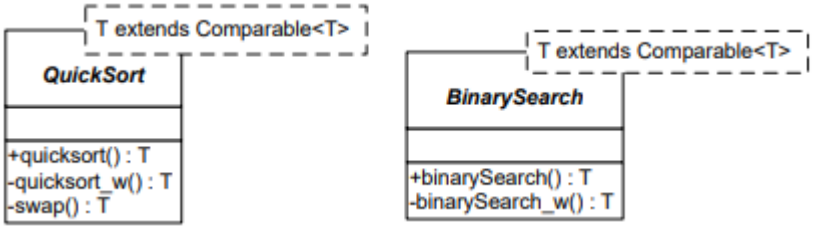
Package maintenance:



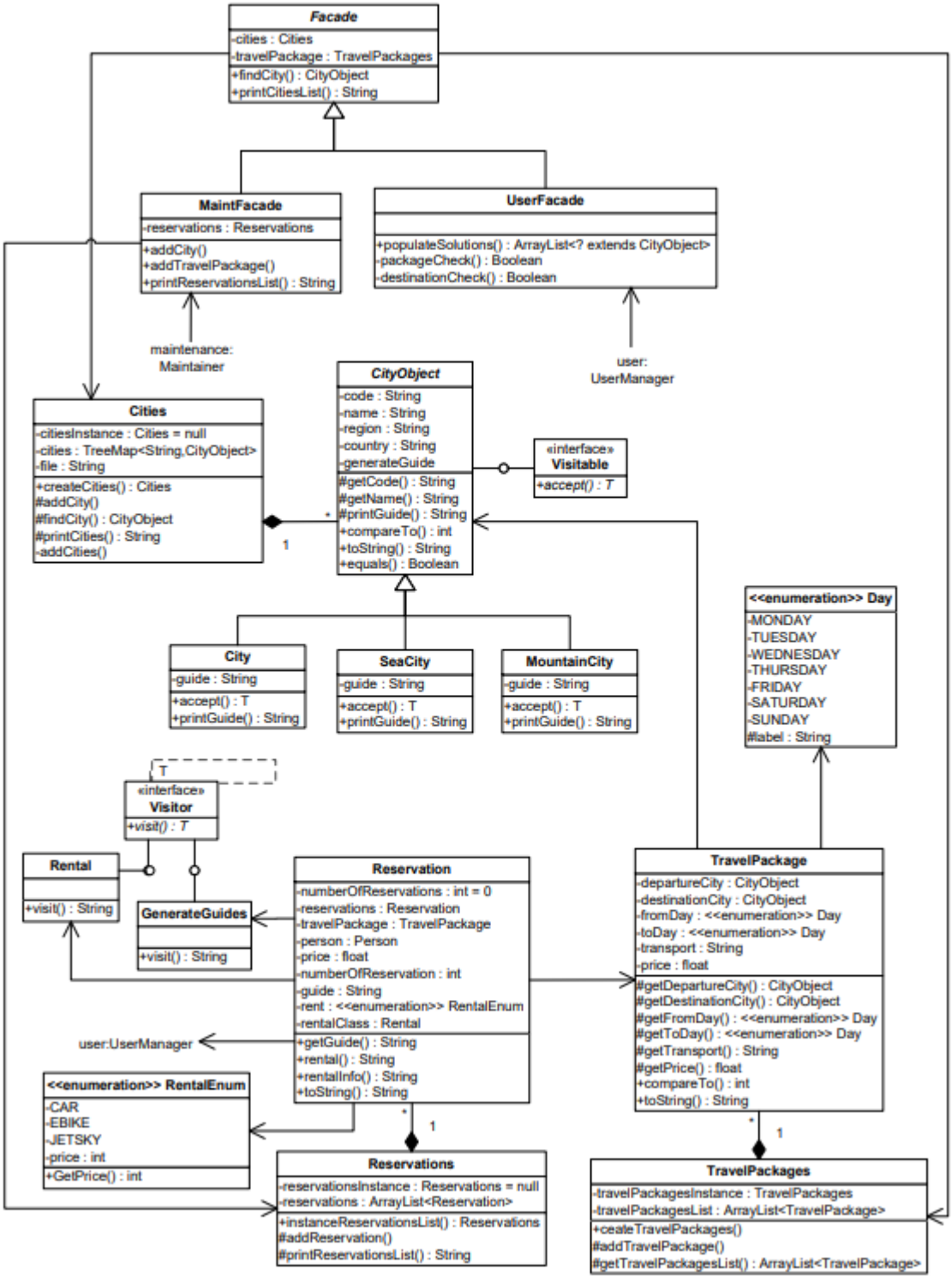
Package exceptions:



Package support:



Package collections:



* nota: il programma utilizzato per comporre il diagramma non dava la possibilità di impostare la visibilità package (~) quindi considerare il simbolo (#) come package invece che protected.

3.5 Design Pattern

Per realizzare il progetto sono stati implementati tre Design Pattern:

- Singleton: pattern creazionale utilizzato in quei casi in cui è necessario esista solamente un'istanza di un determinato oggetto.

Viene applicato per creare le classi che gestiscono la lista delle città, pacchetti viaggio e prenotazioni.

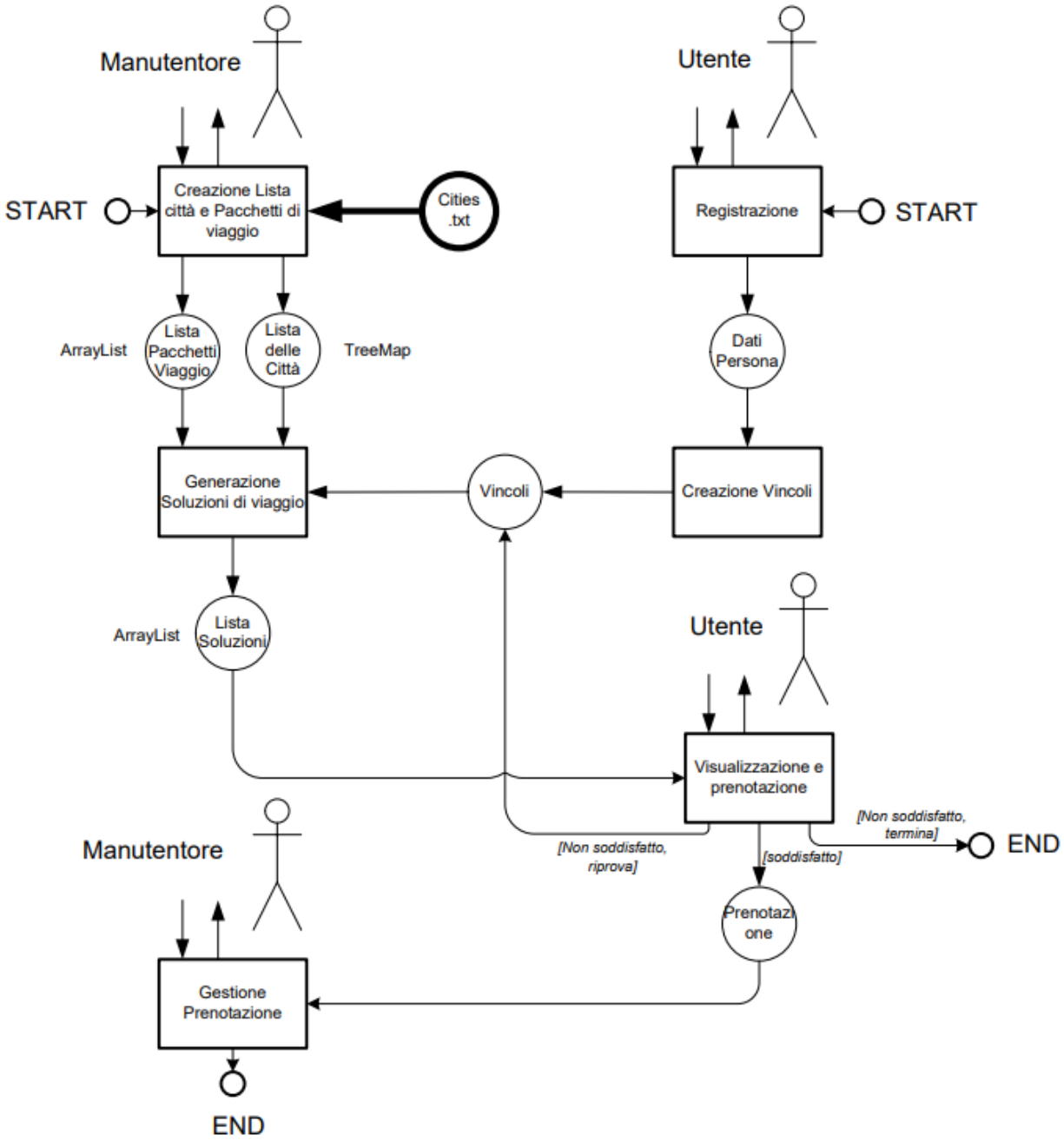
- Facade: pattern strutturale utilizzato per la gestione di oggetti composti da una struttura dati vasta, composta a sua volta da diversi oggetti. Facilità l'accesso tramite un singolo oggetto a un più vasto set di oggetti che lo compongono.
  Nel programma viene utilizzata una superclasse astratta e due sottoclassi con questa logica, una per facilitare l'accesso al manutentore e una per l'utente. In questo modo vengono nascosti oggetti e metodi all'utente finale potenzialmente pericolosi.
- Visitor: pattern ideato per rendere più semplice la manutenzione del codice dedicato alle operazioni da svolgere su determinate classi.
  Nel campo di applicazione di questo programma viene sfruttato per creare metodi per le sottoclassi della superclasse astratta CityObject, in particolare le operazioni di creazione della guida di viaggio personalizzata e il noleggio di un mezzo vengono gestite in classi apposite indipendenti, a cui si accede tramite l'interfaccia visitor e restituiscono un valore diverso in base al tipo di oggetto istanziato che ci accede.

4. Funzionamento

Il funzionamento è descritto tramite la rete di Petri in figura nella pagina successiva, realizzata con un alto livello di astrazione e assumendo che tutte le transizioni siano realizzate in linguaggio Java, tutte sullo stesso server. La risorsa persistente Cities.tex si riferisce alla lista delle città prestabilità.

5. Testing

La fase di test è realizzata sull'apposito package, vengono svolti da prima test funzionali, considerando di creare una persona e procedere con la prenotazione di un pacchetto viaggio, poi si ripete la procedura con uno studente. Per finire un manutentore stampa la lista delle prenotazioni per controllarne la correttezza. Poi si eseguono dei test per verificarne la robustezza, andando a generare eccezioni e valori limite. È previsto anche un test interattivo in cui si simula la procedura del cliente tramite input/output da console.


Manutentore
Utente
START
Creazione Lista città e Pacchetti di viaggio
Cities .txt
Registrazione
START
ArrayList
Lista Pacchetti Viaggio
Lista delle Città
TreeMap
Dati Persona
Generazione Soluzioni di viaggio
Vincoli
Creazione Vincoli
ArrayList
Lista Soluzioni
Utente
Visualizzazione e prenotazione
[Non soddisfatto, termina]
END
Manutentore
[Non soddisfatto, riprova]
[soddisfatto]
Prenotazione
Gestione Prenotazione
END