Relazione
Progetto Rental

Componenti gruppo

- Alessio Falai
  - Matricola: 6134275
  - E-mail: alessio.falai@stud.unifi.it
  - Prove in itinere superate (Votazione: 28)
  - Anno di iscrizione: 2016
- Leonardo Calbi
  - Matricola: 6155786
  - E-mail: leonardo.calbi@stud.unifi.it
  - Prove in itinere superate (Votazione: 25)
  - Anno di iscrizione: 2016

Esercizio scelto e funzionalità del sistema

L'esercizio scelto è Rental. Questo esercizio chiede di simulare la gestione di un sistema di noleggi. Il sistema è scalabile, in quanto come esempio è stata implementata la funzionalità di noleggio film, ma in un momento successivo sarebbe possibile espandere il sistema per includere anche, per esempio, il noleggio di videogiochi o quant'altro. Il sistema gestisce la parte cliente e amministratore. Ogni cliente registrato possiede una carta fedeltà, che viene utilizzata anche per il pagamento dei noleggi, che puó essere di tipo diverso a seconda del tipo di cliente (studente, anziano, ...). Ad ogni carta è associato un credito, che puó essere ricaricato dal cliente, e un insieme di punti, assegnati al cliente in base al numero di noleggi effettuati. Il numero di punti assegnati per ogni noleggio dipende dal tipo di carta in possesso dal cliente. Un cliente puó effettuare un noleggio solamente se ha credito sufficiente sulla propria carta e non ha ancora raggiunto il numero massimo di elementi noleggiabili contemporaneamente (nel nostro caso 3). Il costo giornaliero del noleggio di un film dipende anche dal tipo di supporto scelto (DVD, Blu-Ray, …), mentre il costo totale del noleggio puó essere determinato da alcune strategie di prezzo, come uno sconto per un noleggio che supera i 10 giorni oppure un sovrapprezzo per il noleggio di un film appena uscito. Le strategie di prezzo non sono mutuamente esclusive, ma possono essere composte per determinare una strategia piú complessa e precisa. Ogni amministratore si occupa invece della gestione degli elementi noleggiabili, della gestione dei clienti e degli altri amministratori. In particolare, un amministratore puó aggiungere e rimuovere clienti e amministratori e gestire l'aggiunta e rimozione di film, con il relativo titolo, regista, genere, data di uscita e altro ancora. Inoltre, per ogni film effettivo possono esserci piú copie fisiche disponibili, magari suddivise in base al tipo di supporto.

Diagramma delle classi (UML)

Il diagramma UML è presente nello zip del progetto, nella cartella "uml", sia come file .mdj sia come .jpg che come .pdf.

Scelte di design

DESIGN PATTERN

- **Strategy**: Il pattern viene utilizzato per la determinazione di una strategia di prezzo per quanto riguarda un noleggio. In particolare le classi che implementano l'interfaccia RentPriceStrategy, come Over10DaysDiscount oppure NewReleaseSurcharge, consentono di determinare il prezzo finale di un noleggio (Rent), applicando, rispettivamente, uno sconto del 10% e un sovrapprezzo del 5%.
- **Composite**: Il pattern viene utilizzato in congiunta con il pattern strategy. In particolare è possibile costruire delle strategie di prezzo composte per quanto riguarda un noleggio (RentPriceStrategyCompound). Il pattern consente di trattare in modo uniforme strategie semplici e complesse.
- **Template**: Il pattern viene utilizzato in diverse sezioni di codice, in quanto consente di specializzare le operazioni svolte da un metodo, in base al tipo effettivo dell'oggetto. Il pattern viene utilizzato, per esempio, nella determinazione del prezzo giornaliero di un film noleggiabile, che cambia in base al tipo di supporto scelto (DVD, BD), oppure nell'aggiunta di punti alla carta fedeltà di un cliente, che dipende invece dal tipo effettivo della carta (Standard, Senior, Student).
- **Iterator**: Il pattern viene utilizzato per separare l'implementazione di un contenitore di oggetti dal suo utilizzo. Questo consente di rendere il sistema scalabile, in quanto future modifiche potrebbero portare a una diversa scelta di struttura dati per memorizzare gli oggetti in questione, ma ció non comporterebbe una ristrutturazione delle classi che accedono a tale contenitore. In particolare, tale pattern viene utilizzato nella classe Rented, che contiene gli elementi già noleggiati da un particolare cliente, e nella classe Cart, che contiene invece gli elementi in procinto di essere noleggiati da un particolare cliente.
- **Singleton**: Il pattern viene utilizzato per la gestione delle repository su cui vengono archiviati i dati del sistema. Tali repository devono infatti consentire la creazione di una sola istanza, per evitare inutili duplicati e semplificare la gestione della memorizzazione dei dati.
- **Factory**: Il pattern viene utilizzato per la creazione della carta associata al tipo effettivo del cliente (Standard, Senior, Student).

PRINCIPI DI PROGETTAZIONE

- **Information hiding e encapsulation**: Tutte le classi implementate dichiarano campi con visibilità privata, membri con visibilità privata dove possibile e limitano la presenza di metodi accessori e mutatori. Inoltre, anche l'utilizzo di pattern come Iterator, consentono di massimizzare il concetto di information hiding.
- **DIP (Dependency Inversion Principle) e ADT (Abstract Data Type)**: Ogni classe concreta dipende da una classe astratta o da un'interfaccia. Ció consente di trattare in modo uniforme i tipi concreti e di nascondere l'implementazione per fornire solamente la funzionalità richiesta. Inoltre, vista la progettazione di un framework, l'astrazione consente di rendere il sistema scalabile e modificabile. In particolare, per esempio, i noleggi concreti (DVD, BD), dipendono da un'astrazione di entrambi che fattorizza le caratteristiche a loro comuni (MovieBox), la quale dipende a sua volta da un'interfaccia implementata dagli elementi noleggiabili (Rentable).

- **Open-Closed**: L'utilizzo congiunto di information hiding e DIP consente di rendere il sistema chiuso alle modifiche, ma aperto alle estensioni. In particolare, per esempio, è possibile aggiungere un nuovo tipo di elementi noleggiabili, come i videogiochi, semplicemente implementando l'interfaccia Rentable. Oppure è possibile aggiungere un nuovo tipo di utente, come KidCustomer e la relativa tessera KidCard, semplicemente estendendo le classi Customer e CustomerCard. Tali modifiche sarebbero locali e non comporterebbero alcuna modifica a cascata.
- **SRP (Single Responsibility Principle)**: Tutte le classi implementate hanno complessità limitata e contengono pochi metodi, gli unici necessari a svolgere il proprio compito, descritto dal nome della classe. In particolare, per esempio, la classe Administrator, oltre a contenere la descrizione dell'amministratore, mette a disposizione due soli metodi, uno per l'aggiunta e l'altro per la rimozione di qualunque tipo di elementi archiviabili (Storable).
- **Inheritance, polimorfismo e binding dinamico**: Il concetto di inheritance viene utilizzato molto per consentire il polimorfismo e garantire efficienza. In particolare, per esempio, le classi concrete StandardCustomer, SeniorCustomer e StudentCustomer sono sottoclassi di Customer, che a sua volta è sottoclasse di User, che a sua volta è sottoclasse di Person. Questo consente di trattare in modo uniforme diversi tipi di dati, come gli Administrator e i Customer, che sono entrambi User.
- **Delegation**: Il concetto di composition e delegation viene utilizzato molto per disaccoppiare le classi. In particolare, ad esempio, la classe Customer contiene un oggetto CustomerCard, al quale vengono delegate tutte le operazioni che il cliente vuole effettuare sulla propria tessera. Oppure, la classe User contiene un oggetto Credentials, al quale vengono delegate le operazioni di controllo degli accessi per quanto riguarda le credenziali (username e password) di un cliente o di un amministratore.

Descrizione implementazione

L'esecuzione del programma prevede la presentazione di vari menú di scelta. Inizialmente è possibile scegliere se effettuare il login come amministratore o come cliente. Se le credenziali inserite sono corrette, ovvero sono presenti nell'archivio, viene presentato un menú diverso a seconda del tipo di utente:

- Amministratore: È possibile effettuare la registrazione o la rimozione di altri amministratori o di clienti, oppure si possono aggiungere o rimuovere elementi noleggiabili, come i MovieBox (DVD e BD), e altri elementi archiviabili, come i film (Movie) e i dati relativi ad ogni film, come il regista (Director) e il genere (Genre).
- Cliente: È possibile effettuare le seguenti operazioni:
    - Noleggio: Se il cliente non ha superato il numero massimo di elementi noleggiati, allora visualizza e sceglie, tramite numero seriale, il film che vuole noleggiare. In base al film scelto, al cliente vengono proposti tutti i supporti disponibili e, dopo aver scelto tramite numero seriale, è possibile inserire il numero di giorni del noleggio. Se il numero di giorni è maggiore di 10, al singolo noleggio viene applicato uno sconto del 10% e se il film scelto è una novità, ovvero è uscito negli ultimi 15 giorni, al singolo noleggio viene applicato un sovrapprezzo del 5%. A questo punto il film puó essere effettivamente noleggiato e viene aggiunto al carrello.

- Restituzione: Se il cliente ha noleggiato almeno un elemento, allora viene visualizzata la lista di elementi noleggiati e, dopo aver scelto tramite numero seriale l'elemento da restituire, tale elemento viene nuovamente marcato come disponibile. Nel caso in cui la restituzione venga effettuata con un ritardo, verranno scalati dei punti dalla carta del cliente.
- Carrello: Se il cliente ha effettuato dei noleggi, allora questi vengono visualizzati, assieme al prezzo totale, ed è possibile rimuovere specifici elementi dal carrello oppure procedere al pagamento. La procedura di pagamento scala il credito necessario al noleggio direttamente dalla carta fedeltà del cliente, solamente se il cliente ha abbastanza fondi sulla carta.
- Carta: Viene visualizzato il saldo della carta e l'ammontare dei punti accumulati. Inoltre, è possibile effettuare una ricarica.

Eccezioni

Il package mp.videorental.exception contiene l'insieme delle eccezioni create appositamente per gestire al meglio gli errori di I/O che potrebbero sorgere in una normale esecuzione del programma.

Testing

Il metodo di test utilizzato è JUnit 4. La stesura del codice di una classe è sempre stata accompagnata dalla stesura dei test relativi a tale classe. In particolare, i test sono stati effettuati per ogni metodo di ogni classe, senza ovviamente contare i metodi accessori e mutatori. Alcune classi, la cui implementazione consiste di soli metodi getter e setter, utilizzate per differenziare precisamente diversi tipi di dati con molte caratteristiche comuni, non sono state testate. In particolare, per esempio, è stato effettuato un unico test per la classe astratta Rent, anche se si specializza in SimpleRent e ComplexRent. L'insieme delle classi di test è contenuto all'interno della cartella "tests", nel package mp.videorental.test. Per poter eseguire tutti i test contemporaneamente è possibile eseguire la classe TestSuite. Inoltre, i test comprendono la gestione del lancio di eccezioni per ogni metodo e la gestione delle repository.

Archiviazione e serializzazione

Il sistema comprende l'implementazione e la gestione di un metodo di archiviazione dei dati. In particolare tutte le classi degli oggetti che possono essere salvate in una repository, ovvero un catalogo, implementano l'interfaccia Storable, che a sua volta estende l'interfaccia Serializable. Invece, tutte le classi degli oggetti che non possono essere salvati direttamente in archivio, ma sono contenuti in altre classi composte che hanno invece questa possibilità, implementano direttamente l'interfaccia Serializable. La classe astratta Repository è una classe generica con placeholder <T extends Storable>, che contiene un catalogo di dati e fornisce varie operazioni su tale catalogo, come l'aggiunta, la rimozione e la ricerca di elementi. La classe Repository viene poi estesa da altre classi, come MovieBoxRepository e CustomerRepository, che consentono di memorizzare nel proprio catalogo solamente oggetti di tipo MovieBox e Customer, rispettivamente. Solamente un Administrator puó effettuare operazioni di aggiunta e rimozione nelle varie repository. In particolare, se per esempio è necessario aggiungere un MovieBox (DVD o BD) al catalogo, verrà chiamato il

metodo add del MovieBox stesso, al quale deve essere passata un'istanza di Administrator. Tale metodo, interno a MovieBox, chiamerà a sua volta il metodo add della repository che gli compete (in questo caso MovieBoxRepository), passandogli se stesso e l'istanza di Administrator ricevuta. A questo punto, il metodo add della repository controllerà se l'amministratore esiste ed è valido e in tal caso aggiungerá l'elemento al catalogo, se non esiste già, altrimenti solleverà un'eccezione. Quando una repository concreta viene istanziata, è necessario leggere dal file corrispondente il contenuto di tale repository. Nel caso in cui il file sia vuoto, ovvero alla prima esecuzione del programma, si crea l'unica istanza della classe, altrimenti tale unica istanza viene acquisita dalla classe StorableHandler, tramite il metodo readObject della classe ObjectInputStream. Ogni volta che viene effettuata un'aggiunta e una rimozione l'intera repository modificata viene scritta su file dalla classe StorableHandler, utilizzando il metodo writeObject della classe ObjectOutputStream. I file di default delle repository (Administrator, Customer, Director, Genre, Movie, MovieBox) sono contenuti nella cartella "data". In particolare, la repository Administrator contiene un amministratore di default, non rimovibile, le cui credenziali di accesso sono "admin" per lo username e "admin" per la password.

Note

In alcuni punti del codice, per facilitare lo sviluppo, sono state utilizzate delle lambda expression. In particolare, per esempio, le lambda sono state utilizzate estensivamente nelle classi di test delle repository, nella fase di teardown, necessaria per fare pulizia e riportare l'ambiente di esecuzione nelle stesse condizioni precedenti all'esecuzione del test, in modo da non influenzare test ed esecuzioni successive del programma. Inoltre, nella stesura della classe "Main", contenuta nel package mp.videorental.main, sono state utilizzate varie lambda expression assieme agli stream associati ai cataloghi delle repository, per reperire insiemi di dati in modo semplice e rapido. Per verificare velocemente un tipico caso d'uso del sistema è possibile eseguire la classe "FastMain" contenuta nel package mp.videorental.main.