Corso di Laurea In Informatica (L-31)

Università di Camerino

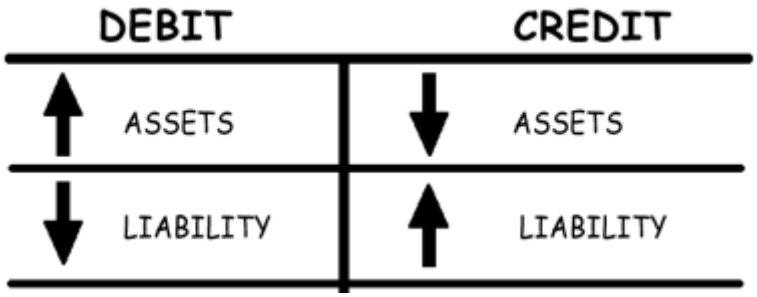
Esame di Programmazione Avanzata – 2° Anno

JBudget : relazione

A seguito di uno studio delle specifiche progettuali, sono stati individuati diversi concetti che ho deciso di implementare all'interno dell'applicativo, di seguito una lista dei concetti con relativa spiegazione.

Conto: rappresenta i movimenti contabili che vengono effettuati, ciascun conto può essere considerato come a sé stante, le transazioni che verranno effettuate su un conto non andranno quindi ad influire sugli altri conti. I conti possono essere di 2 tipi: Assets (che può essere paragonato ad un conto corrente bancario, con entrate ed uscite) e Liabilities (che può essere pensato come un debito). I movimenti che verranno associati ad un conto, avranno quindi diverso significato in base al tipo di conto al quale vengono associati.

Movimento: un movimento è uno spostamento di denaro che viene effettuato, esso può essere di 2 tipi, Credit o Debit, nel seguente schema, possiamo notare come esso va a modificare il balance di un conto, in base al tipo di movimento che viene effettuato in un determinato Account.



Transazione: una transazione è un insieme di movimenti, essa viene utilizzata in quanto è possibile che una transazione sia composta da più movimenti (ad esempio ritirare dei soldi in banca potrebbe essere una transazione composta da 2 movimenti, ossia un movimento che rappresenta i soldi ritirati ed un movimento che rappresenta la commissione che la banca si prende per effettuare tale operazione).

Transazione programmata: una transazione programmata è una transazione che non viene conteggiata all'interno del conto in quanto deve essere ancora eseguita, al raggiungimento di quella data, tale transazione viene spostata diventando così una normale transazione e conseguentemente verrà conteggiata all'interno del conto.

Classi ed interfacce sviluppate:

Tag: questa interfaccia definisce la struttura di un tag contente il nome e la descrizione, serve per definire la categoria del tipo di movimento effettuato.

BasicStructure: definisce una struttura generica per il salvataggio di ID, tag e data e la loro gestione, questa interfaccia viene implementata da tutte le classi che hanno bisogno dell'utilizzo di tali dati, evitando in questo modo codice duplicato

Movement: questa interfaccia definisce la struttura di un movimento, e la gestione dei suoi dati, questa interfaccia viene implementata da tutte le classi che vogliono definire la struttura di un movimento generico, contenente descrizione, tipo di movimento e ammontare del movimento.

Transaction: questa interfaccia definisce la struttura di una transazione, viene implementata da tutte le classi che hanno bisogno di definire una struttura per delle transazioni, le classi che implementano questa interfaccia devono avere una lista di movimenti in quanto al suo interno vi sono metodi per la gestione di movimenti contenuti all'interno di una transazione.

Ledger: questa interfaccia viene implementata da tutte le classi che hanno bisogno della gestione completa dei dati dell'applicativo, è letteralmente il libro mastro in quanto tutte le classi che la implementano, si occupano del salvataggio e gestione dei dati, quali: movimenti, tag, transazioni e transazioni programmate.

Scheduler: questa interfaccia viene implementata dalle classi che si occupano della gestione ed il salvataggio delle transazioni schedulate, le classi che la estendono contengono una lista di Transazioni, la data di queste transazioni è futura. All'interno di tale interfaccia è definito il metodo che si occupa di “spostare” le transazioni che sono state eseguite (transazioni le quali data è precedente od uguale alla data odierna) dallo Scheduler al Ledger.

DataManager: questa interfaccia viene implementata dalle classi che si occupano del caricamento e salvataggio dei dati all’interno di file.

JsonParser: viene implementata questa interfaccia dalle classi che contengono l’implementazione di strutture per effettuare il parse di dati di tipo Json.

Definisce 4 metodi rispettivamente per il parse di: data, tipi JsonPrimitive (es: String, double, int ecc..), parse di liste, deserializzazione di liste.

FileManager: interfaccia implementata da tutte le classi in grado di importare un file di settings ed ottenere quindi il nome del file contente i dati. Le classi che la implementano, permettono quindi il caricamento ed il salvataggio dei dati

GenericController: interfaccia implementata dalle classi che definiscono un controller generico in grado di visualizzare una lista di tag all’interno di una tabella espandibile.

AccountType: enumerazione che definisce i due tipi di account (ASSETS, LIABILITIES).

MovementType: enumerazione che definisce i diversi tipi di movimento (DEBIT, CREDITS), in base a questo tipo, viene definito l’effetto che il movimento avrà su un conto (guardare immagine precedente). In caso di account di tipo ASSETS = DEBIT – CREDITS, Account di tipo LIABILITIES = CREDITS - DEBIT

ButtonType: enumerazione che definisce i diversi tipi di bottoni di un form (ADDMOVEMENT, ADDTAG, SHOWTRANSACTIONS, SHOWMOVEMENTS, SHOWTAG, REMOVETAGMOVEMENT, SHOWACCOUNT), ognuno dei quali ha un path del file FXML associato ed una descrizione di cosa andrà a fare il file fxml associato.

Test effettuati:

MovementTest:

TestGetMovements: effettuati test di aggiunta di un movimento ad una transazione e verifica se tale movimento è presente.

TransactionTest:

testGetTransaction: creo una transazione, la inserisco nel Ledger e la confronto con la transazione corretta.

TestGetAmount: creo una transazione, inserisco due movimenti di due MovementType differenti e confronto il risultato del getBalance

AccountTest:

AccountTestBalance: creo un account di tipo ASSETS, inserisco due movimenti di tipo differente e confronto il risultato.

Rieseguo la stessa operazione per un account di tipo LIABILITIES.

AccountAdapterTest:

testDeserialize : data una stringa json, deserializzo un account, confrontandolo con un oggetto da me creto per confrontare se corrispondono.

TestSerialize: Dato un json effettuo la serializzazione di un oggetto e lo confronto con la stringa json corretta.

JsonParserTest:

Date delle stringhe json corrette, le confronto con il risutato della deserializzazione di JsonParser.

testParseDate: effettuo la deserializzazione di una stringa per ottenere una data

testParsePrimitive: efffettuo la deserializzazione di una stringa per ottenere diversi tipi quali: String, int e double, confrontando quindi con i risultati corretti.

testParseList: data una stringa json contente i dati di 2 Tag, effettuo la deserializzazione e confronto i risultati con gli oggetti creati.

Estendibilità:

Per rendere l'applicativo più aperto all'implementazione di nuove funzionalità, il parse dei dati, date delle stringhe json, è stato effettuato con tipi generici, in modo che possa essere riutilizzato indipendentemente dal tipo che ci può servire in quel momento, i controller sono stati suddivisi in più classi, questo fa sì che un controller possa essere riutilizzato in maniera più semplice, ad esempio il controller TagController viene riutilizzato sia all'interno di una tabella espandibile durante la visualizzazione delle transazioni, sia per la sola visualizzazione dei Tag.

È stata creata una classe generica che definisce la struttura di ID, lista di Tag e data, questa struttura permette al programma di essere più espandibile in quanto potrebbe essere riutilizzata in maniera molto semplice, nello stesso modo in cui viene utilizzata dalle classi Movement e Transaction.

La classe FileManager effettua il caricamento di un file di settings, il quale contiene il path del file contente i dati, in questo modo il file contente i dati può essere spostato o cambiare il nome, senza avere conseguenze sul risultato. La stessa classe implementa metodi di lettura e scrittura su file molto generici in modo che possano essere riutilizzati da qualsiasi classe per il salvataggio di altri file contenti dati.