RELAZIONE FINALE "JBUDGET"

Vittorio Rinaldi 100763

Premessa:

L' applicazione è pubblicata dall'autore come "Open Source senza scopo di lucro" (licenza **GPL v3**).

Pattern implementati:

1. **MVC** (Model-View-Controller, strutturale)
2. **Singleton** (creazionale).
3. **Facade**: gestire il sotto-sistema del model tramite un'interfaccia semplice (Ledger) per l'interazione tra l'utente ed il core business (strutturale)

Sviluppato utilizzando le seguenti versioni:

Gradle 6.5

Java 11

JavaFX 14

È presente documentazione **JavaDoc**!

Note dello sviluppatore

Unicamente per questioni legate alle tempistiche di consegna, l'interfaccia utente implementata è in fase di lavorazione: con questo si intende che la filosofia scelta dallo sviluppatore è "Qualità prima della quantità".

In altre parole, a costo di un minor ritmo, ma ben maggiore usabilità, *si è scelto* di NON intraprendere una programmazione imprudente ed affrettata, ma una che prevedesse la cura di ogni funzionalità prima della sua pubblicazione.

Attualmente, tramite interfaccia grafica è possibile gestire perfettamente la creazione di nuovi Account e Tag nell'applicazione, la loro rimozione, nonché visualizzarne le caratteristiche.

Si sono risolti i problemi legati alla precedente consegna e la parte di interfaccia grafica accessibile all'utente in questo momento è garantita come perfettamente funzionante;

si fa notare come lato back-end moltissime altre funzionalità siano curate e già altrettanto pronte, necessitanti unicamente di essere collegate all'utente finale tramite GUI (si parla di: gestione dei movimenti, gestione transazioni, la schedulazione di transazioni, previsione di spesa per uno o più Tag (accantonamenti - budget), report consuntivi che rappresentano il saldo tra entrate ed uscite di un insieme di Transazioni).

➔ Saranno disponibili a breve.

Anche nel lungo termine è in programma un ampliamento del programma:

- Lettura e scrittura dei dati sfruttando la libreria Gson, garantire la sincronizzazione tra dispositivi tramite cloud direttamente dall'applicazione.

I file FXML della GUI sono stati separati dalla cartella del FxController per una questione di sintassi e pulizia;

sono locati nella cartella 'Resources'.

Implementazione test-driven garantita per le funzionalità delle seguenti interfacce:

- Ledger
- Account
- Transaction
- Movement

Per iniziare…

L'applicazione può essere avviata tramite Console digitando il comando "**gradle run**" nella directory del progetto, oppure tramite qualsiasi IDE importando il progetto Gradle.

Package it.unicam.cs.pa.jbudget100763

Qui sono contenute tutte le interfacce e classi del Progetto.

Main:

Classe principale, permette di avviare l'applicazione e la GUI scelta, attualmente JavaFX.

Package it.unicam.cs.pa.jbudget100763.controller

include la classe che ha il compito di ricevere i comandi dell'utente (in genere attraverso la View) e di attuarli modificando lo stato degli altri due componenti del MVC.

Controller:

classe, riceve i comandi dell'utente tramite la View e li attua modificando lo stato del Model.

Gestisce il Ledger, il Budget ed il TagBudgetReport.

Package it.unicam.cs.pa.jbudget100763.view

stabilisce i comportamenti ritenuti fondamentali per visualizzare i dati contenuti nel model ed occuparsi dell'interazione con gli utenti.

View:

Interfaccia, ha la responsabilità di indicare le direttive principali riguardo l'interazione dell'utente con l'applicazione.

Package it.unicam.cs.pa.jbudget100763.view.javafx

include l'implementazione della GUI tramite la piattaforma JavaFX, seguendo le direttive della View.

App:

Applicazione principale dell'implementazione tramite javaFX.

FxController:

classe, gestisce le schermate da avviare e raccoglie tutte le interazioni dell'utente tramite la GUI, inoltrandole al controller dell'applicazione principale.

AccountController:

classe, gestisce la finestra della creazione dei conti.

TagController:

classe, che gestisce la finestra della creazione dei Tag.

Package it.unicam.cs.pa.jbudget100763.model

ha il compito di gestire i dati dell'applicazione, fornisce i metodi per accedere ad essi.

Account:

interfaccia, è implementata dalle classi che hanno la responsabilità di gestire un conto.

Budget:

interfaccia, ha la responsabilità di rappresentare e gestire un particolare budget, ovvero la previsione di spesa/guadagno per uno o più Tag.

Ledger:

interfaccia, è implementata dalle classi che hanno la responsabilità di gestire tutti i dati dell'applicazione.

Movement:

interfaccia, è implementata dalle classi che hanno la responsabilità di gestire un singolo movimento.

ScheduledTransaction:

interfaccia, indica una transazione o una serie di transazioni schedulate (previste) ad una certa data.

Tag:

interfaccia, è implementata dalle classi che hanno la responsabilità di definire una categoria di spesa/guadagno.

TagBudgetReport:

interfaccia, è implementata dalle classi che hanno la responsabilità di gestire un budget (bilancio) accantonato per uno o più tag.

Transaction:

interfaccia, è implementata dalle classi che hanno la responsabilità di gestire una transazione.

AccountImpl:

classe, ha la responsabilità di gestire un conto.

- double getBalance(): aggiorna il saldo attuale, partendo da quello iniziale dell'account e sommando di volta in volta tutti i movimenti avvenuti
- Set<Movement> getMovements(): I movimenti correlati a questo account sono contenuti nelle transazioni (e quindi nel ledger), questi vengono collegati all'account
- Set<Movement> getMovements(Predicate<Movement> condition): ritorna la lista dei movimenti di questo conto che rispettano il predicato

BudgetImpl:

classe, un budget associa ad ogni tag un importo che indica l'ammontare di spesa/guadagno previsto per quel particolare tag.

- double getBalance(Tag t): restituisce la cifra accantonata (prevista) per quel tag
- void setBalance(Tag t, Double expected): aggiunge un nuovo tag e relativo accantonamento previsto
- Set<Tag> tags(Predicate<Transaction> condition): ritorna i tag utilizzati nelle transazioni che rispettano una certa condizione (es: avvenute in un determinato periodo di tempo)
- Predicate<Transaction> after(GregorianCalendar date): lambda expression per definire una transazione avvenuta dopo la certa data
- Predicate<Transaction> before(GregorianCalendar date): lambda expression per definire una transazione avvenuta prima la certa data

LedgerImpl:

classe, ha la responsabilità di gestire tutti i dati dell'applicazione.

- static Ledger getInstance(): Singleton implementation

- Account addAccount(AccountType type, String name, String description, double openingBalance): Crea un nuovo account nell'applicazione
- Tag addTag(String name, String description): Crea un nuovo tag nell'applicazione
- void removeTag(Tag t): tag da rimuovere
- boolean addTransaction(GregorianCalendar date): crea una nuova transazione e la inserisce nella lista dell'applicazione
- Set<Transaction> getTransactions(Predicate<Transaction> condition): ritorna la lista di transazioni che rispettano un certo predicato
- Set<ScheduledTransaction> getScheduled(): ritorna tutte le transazioni schedulate dell'applicazione
- Set<ScheduledTransaction> searchScheduledTransaction(GregorianCalendar d): restituisce le scheduled transaction fissate a quella data
- void addScheduledTransaction(ScheduledTransaction st): Aggiunge l'istanza Sch.Tran. alla lista
- void schedule(ScheduledTransaction st): Sincronizza (aggiunge) tutte le transazioni previste ad una data futura con una scheduled transaction
- boolean scheduleSpecificTransaction(Transaction transaction, ScheduledTransaction st): Combina manualmente una transazione con una Sched.Transaction, devono manifestarsi lo stesso momento

MovementImpl:

implementa la responsabilità di gestire un singolo movimento

- il costruttore del movimento collega automaticamente l'istanza stessa con la transazione associata
- getters & setters dei propri campi: MovementType, amount , date, Tag, description, Account, Transaction

RegistryImpl<T>:

DISCLAIMER: Classe embrionale in via di sviluppo per la persistenza dei dati dell'applicazione. Work in progress.

ScheduledTransactionImpl:

indica una transazione o una serie di transazioni schedulate (previste) ad una certa data.

- boolean isCompleted(): verifica se la serie di transazioni si sia manifestata o meno
- boolean addTransaction(Transaction t): se la transazione da schedulare rispetta i requisiti della scheduledTransaction allora viene inserita nella medesima

TagImpl:

definisce una categoria di spesa/guadagno.

- Getters & setters dei propri campi: nome, descrizione

TagBudgetReportImpl:

mostra il saldo di positivo/negativo di uno o più Tag.

- Map<Tag, Double> getTagBalance(Set<Transaction> transactions): saldo complessivo di ogni distinto tag trovato nelle transizioni in input
- Set<Tag> totalTags(): ritorna tutti i tag creati nell'applicazione
- Map<Tag, Double> report(Predicate<Transaction> condition): ritorna il saldo/bilancio di tutti i tag trovati nelle transazioni che rispettano il Predicato

TransactionImpl:

ha la responsabilità di gestire una transazione.

- Set<Tag> getTags(): mostra la lista di tutti i tag distinti dei movimenti contenuti
- void addTag(Tag t): Tag da inserire a tutti i movimenti della transazione e quindi automaticamente alla transazione stessa
- double getTotalAmount(): somma algebrica del valore di tutti i movimenti contenuti

---

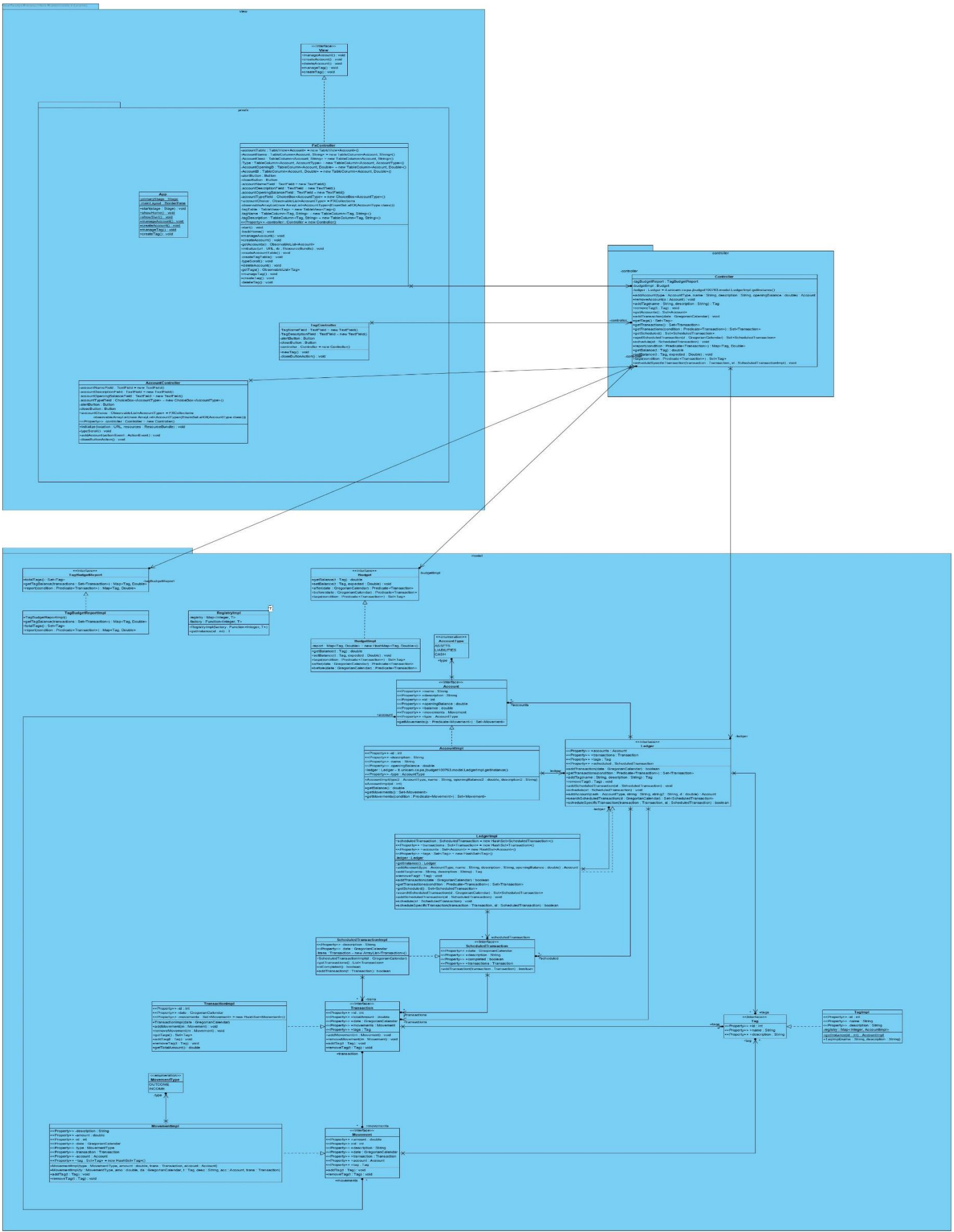
AccountType:

enumerazione che rappresenta le tipologie di conto (ASSETS, LIABILITIES, CASH)

MovementType:

enumerazione che rappresenta la direzione di un movimento su un conto (OUTCOME, INCOME).

[Class Diagram disponibile in alta risoluzione nella directory del progetto]


view
«Interface»
View
+manageAccount() : void
+createAccount() : void
+deleteAccount() : void
+manageTag() : void
+createTag() : void
javafx
App
-primaryStage : Stage
-mainLayout : BorderPane
+start(stage : Stage) : void
+showMain() : void
+showStart() : void
+manageAccount() : void
+createAccount() : void
+manageTag() : void
+createTag() : void
FxController
-accountTable : TableView<Account> = new TableView<Account>()
-AccountName : TableColumn<Account, String> = new TableColumn<Account, String>()
-AccountDesc : TableColumn<Account, String> = new TableColumn<Account, String>()
-Type : TableColumn<Account, AccountType> = new TableColumn<Account, AccountType>()
-AccountOpeningB : TableColumn<Account, Double> = new TableColumn<Account, Double>()
-AccountB : TableColumn<Account, Double> = new TableColumn<Account, Double>()
-alertButton : Button
-closeButton : Button
-accountNameField : TextField = new TextField()
-accountDescriptionField : TextField = new TextField()
-accountOpeningBalanceField : TextField = new TextField()
-accountTypeField : ChoiceBox<AccountType> = new ChoiceBox<AccountType>()
~accountChoice : ObservableList<AccountType> = FXCollections
.observableArrayList(new ArrayList<AccountType>(EnumSet.allOf(AccountType.class)))
-tagTable : TableView<Tag> = new TableView<Tag>()
-tagName : TableColumn<Tag, String> = new TableColumn<Tag, String>()
-tagDescription : TableColumn<Tag, String> = new TableColumn<Tag, String>()
<<Property>> -controller : Controller = new Controller()
+start() : void
-loadHome() : void
+manageAccount() : void
+createAccount() : void
-getAccounts() : ObservableList<Account>
+initialize(url : URL, rb : ResourceBundle) : void
-createAccountTable() : void
-createTagTable() : void
-typeScroll() : void
+deleteAccount() : void
-getTags() : ObservableList<Tag>
+manageTag() : void
+createTag() : void
-deleteTag() : void
TagController
-TagNameField : TextField = new TextField()
-TagDescriptionField : TextField = new TextField()
-alertButton : Button
-closeButton : Button
-controller : Controller = new Controller()
+newTag() : void
-closeButtonAction() : void
AccountController
-accountNameField : TextField = new TextField()
-accountDescriptionField : TextField = new TextField()
-accountOpeningBalanceField : TextField = new TextField()
-accountTypeField : ChoiceBox<AccountType> = new ChoiceBox<AccountType>()
-alertButton : Button
-closeButton : Button
~accountChoice : ObservableList<AccountType> = FXCollections
.observableArrayList(new ArrayList<AccountType>(EnumSet.allOf(AccountType.class)))
<<Property>> -controller : Controller = new Controller()
+initialize(location : URL, resources : ResourceBundle) : void
-typeScroll() : void
+addAccountAction(event : ActionEvent) : void
-closeButtonAction() : void
controller
Controller
-tagBudgetReport : TagBudgetReport
-budgetImpl : Budget
-ledger : Ledger = (Ledger)Class.forName("ca.pa.jbudget100763.model.LedgerImpl").getMethod("getInstance").invoke(null)
+addAccount(type : AccountType, name : String, description : String, openingBalance : double) : Account
+removeAccount(a : Account) : void
+addTag(name : String, description : String) : Tag
+removeTag(t : Tag) : void
+getAccounts() : Set<Account>
+addTransaction(date : GregorianCalendar) : void
+getTags() : Set<Tag>
+getTransactions() : Set<Transaction>
+getTransactions(condition : Predicate<Transaction>) : Set<Transaction>
+getScheduled() : Set<ScheduledTransaction>
+getScheduledTransactions(d : GregorianCalendar) : Set<ScheduledTransaction>
+deleteTag(t : Tag) : void
+deleteScheduledTransaction(transaction : Transaction) : void
+reportCondition : Predicate<Transaction>() : Map<Tag, Double>
+getBalance() : Tag, expected : Double) : void
+getBalance() : Tag, expected : Double) : void
+getTransactions(condition : Predicate<Transaction>) : Set<Tag>
+scheduleSpecificTransaction(transaction : Transaction, st : ScheduledTransactionImpl) : void
model
«Interface»
TagBudgetReport
+initTags() : Set<Tag>
+getTagBalanceTransactions : Set<Transaction>) : Map<Tag, Double>
+report(condition : Predicate<Transaction>) : Map<Tag, Double>
-tagBudgetReport
TagBudgetReportImpl
+TagBudgetReportImpl()
+getTagBalanceTransactions : Set<Transaction>) : Map<Tag, Double>
+initTags() : Set<Tag>
+report(condition : Predicate<Transaction>) : Map<Tag, Double>
RegistryImpl
-registry : Map<Integer, T>
-factory : Function<Integer, T>
-RegistryImpl(factory : Function<Integer, T>)
+getInstance(id : int) : T
T
«Interface»
Budget
+getBalance() : Tag) : double
+setBalance() : Tag, expected : Double) : void
+after(date : GregorianCalendar) : Predicate<Transaction>
+before(date : GregorianCalendar) : Predicate<Transaction>
+tagsCondition : Predicate<Transaction>() : Set<Tag>
-budgetImpl
BudgetImpl
-report : Map<Tag, Double> = new HashMap<Tag, Double>()
+getBalance() : Tag) : double
+setBalance() : Tag, expected : Double) : void
+tagsCondition : Predicate<Transaction>() : Set<Tag>
+after(date : GregorianCalendar) : Predicate<Transaction>
+before(date : GregorianCalendar) : Predicate<Transaction>
«enumeration»
AccountType
ASSETS
LIABILITIES
CASH
-type
«Interface»
Account
<<Property>> +name : String
<<Property>> +description : String
<<Property>> +id : int
<<Property>> +openingBalance : double
<<Property>> +balance : double
<<Property>> +movements : Movement
<<Property>> +type : AccountType
+getMovements(p : Predicate<Movement>) : Set<Movement>
*accounts
-account
AccountImpl
<<Property>> +id : int
<<Property>> +description : String
<<Property>> +name : String
<<Property>> +openingBalance : double
-ledger : Ledger = (Ledger)Class.forName("ca.pa.jbudget100763.model.LedgerImpl").getMethod("getInstance").invoke(null)
<<Property>> +type : AccountType
+AccountImpl(type2 : AccountType, name : String, openingBalance2 : double, description2 : String)
+AccountImpl(id : int)
+getBalance() : double
+getMovements() : Set<Movement>
+getMovements(condition : Predicate<Movement>) : Set<Movement>
ledger
«Interface»
Ledger
<<Property>> +accounts : Account
<<Property>> +transactions : Transaction
<<Property>> +tags : Tag
<<Property>> +scheduled : ScheduledTransaction
+addTransaction(date : GregorianCalendar) : boolean
+getTransactions(condition : Predicate<Transaction>) : Set<Transaction>
+addTag(name : String, description : String) : Tag
+removeTag(t : Tag) : void
+addScheduledTransaction(st : ScheduledTransaction) : void
+schedule(st : ScheduledTransaction) : void
+addAccount(path : AccountType, string : String, string2 : String, d : double) : Account
+searchScheduledTransaction(d : GregorianCalendar) : Set<ScheduledTransaction>
+scheduleSpecificTransaction(transaction : Transaction, st : ScheduledTransaction) : boolean
-ledger
LedgerImpl
-scheduledTransaction : ScheduledTransaction = new HashSet<ScheduledTransaction>()
<<Property>> ~transactions : Set<Transaction> = new HashSet<Transaction>()
<<Property>> ~accounts : Set<Account> = new HashSet<Account>()
<<Property>> ~tags : Set<Tag> = new HashSet<Tag>()
-ledger : Ledger
-getInstance() : Ledger
-addAccount(type : AccountType, name : String, description : String, openingBalance : double) : Account
+addTag(name : String, description : String) : Tag
+removeTag(t : Tag) : void
+addTransaction(date : GregorianCalendar) : boolean
+getTransactions(condition : Predicate<Transaction>) : Set<Transaction>
-getScheduled() : Set<ScheduledTransaction>
+searchScheduledTransaction(d : GregorianCalendar) : Set<ScheduledTransaction>
+addScheduledTransaction(st : ScheduledTransaction) : void
+schedule(st : ScheduledTransaction) : void
+scheduleSpecificTransaction(transaction : Transaction, st : ScheduledTransaction) : boolean
ScheduledTransactionImpl
<<Property>> -description : String
<<Property>> -date : GregorianCalendar
trans : Transaction = new ArrayList<Transaction>()
-ScheduledTransactionImpl(id : GregorianCalendar)
+getTransactions() : List<Transaction>
+isComplete() : boolean
+addTransaction(t : Transaction) : boolean
-scheduledTransaction
«Interface»
ScheduledTransaction
<<Property>> +date : GregorianCalendar
<<Property>> +description : String
<<Property>> +completed : boolean
<<Property>> +transactions : Transaction
+addTransaction(transaction : Transaction) : boolean
+scheduled
TransactionImpl
<<Property>> +id : int
<<Property>> +date : GregorianCalendar
<<Property>> +movements : Set<Movement> = new HashSet<Movement>()
+TransactionImpl(date : GregorianCalendar)
+addMovement(m : Movement) : void
+removeMovement(m : Movement) : void
+getTags() : Set<Tag>
+addTag(t : Tag) : void
+removeTag(t : Tag) : void
+getTotalAmount() : double
«Interface»
Transaction
<<Property>> +id : int
<<Property>> +totalAmount : double
<<Property>> +date : GregorianCalendar
<<Property>> +movements : Movement
<<Property>> +tags : Tag
+addMovement(m : Movement) : void
+removeMovement(m : Movement) : void
+addTag(t : Tag) : void
+removeTag(t : Tag) : void
*transactions
+trans
+transaction
«Interface»
Tag
<<Property>> +id : int
<<Property>> +name : String
<<Property>> +description : String
+tags
TagImpl
<<Property>> -id : int
<<Property>> -name : String
<<Property>> -description : String
-registry : Map<Integer, AccountImpl>
+getInstance(id : int) : AccountImpl
+TagImpl(name : String, description : String)
«enumeration»
MovementType
OUTGOING
INCOMING
-type
MovementImpl
<<Property>> -description : String
<<Property>> -amount : double
<<Property>> -id : int
<<Property>> -date : GregorianCalendar
<<Property>> -type : MovementType
<<Property>> -transaction : Transaction
<<Property>> -account : Account
<<Property>> ~tag : Set<Tag> = new HashSet<Tag>()
+MovementImpl(type : MovementType, amount : double, trans : Transaction, account : Account)
+MovementImpl(ty : MovementType, amo : double, da : GregorianCalendar, t : Tag, desc : String, acc : Account, trans : Transaction)
+addTag(t : Tag) : void
+removeTag(t : Tag) : void
«Interface»
Movement
<<Property>> +amount : double
<<Property>> +id : int
<<Property>> +description : String
<<Property>> +date : GregorianCalendar
<<Property>> +transaction : Transaction
<<Property>> +account : Account
<<Property>> +tag : Tag
+addTag(t : Tag) : void
+removeTag(t : Tag) : void
*movements