Design Document - Iteration 2

Team PA-PI-a

18 March 2018

Table 1: Team

Name	ID Number
Melanie Taing	40009850
Laurie Gagnon	22943433
Wayne Yiel Leung	26586988
Jordan Rutty	27300107
Alice Barkhouse	27486782
Michael Foo	40000225
Pierre-Andre Leger	40004010
Colin Greczkowski	40001600

Contents

List of Figures

1 Introduction

The purpose of this document is to describe and provide details for the design and implementation of the second iteration of the MyMoney application.

The following pages will cover the rational of the architectural design as well as the subsystem interface specifications and details on their implementation. Finally, we will describe three dynamic design scenarios based on use cases specified in the documentation for iteration 1.

2 Architectural Design

The Mymoney application is implemented using a model-view-controller (MVC) architecture. this section will cover the architectural diagram for the MVC as well as the subsystem interface specifications.

2.1 Architectural Diagram

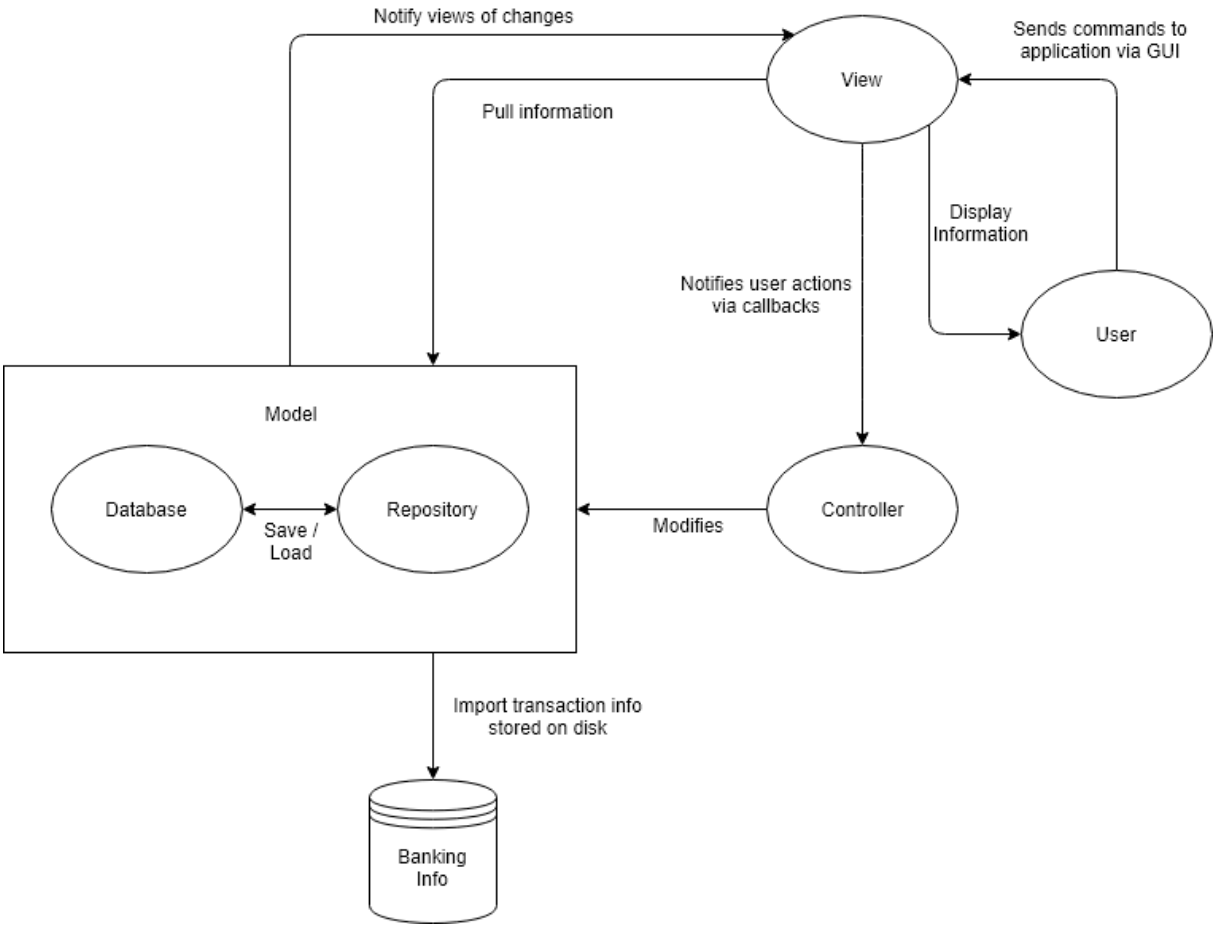

Figure 1: High level structure of MVC architecture

The model contains all the information related to the transactions and the accounts that the user wishes to track. It consists of an SQL database used to serialize and deserialize the information between user sessions and a repository with which the rest of the application interacts. Modifications to the repository are saved on-the-fly to the database while the program is running. The view displays the accounts and transactions loaded into the model (repository) and offers interactive elements that the user can interact with. In essence, it is a GUI. The controller handles user input from the view (GUI) and then acts on the model accordingly by adding, modifying or deleting transactions or accounts.

The main advantage of using an MVC pattern is the separation of concerns. As will be demonstrated in the next section, by enforcing each subsystem to depend strictly on `Interface` types when communicating with each other, we can reduce dependencies and greatly increase modularity.

2.2 Subsystem Interface Specifications

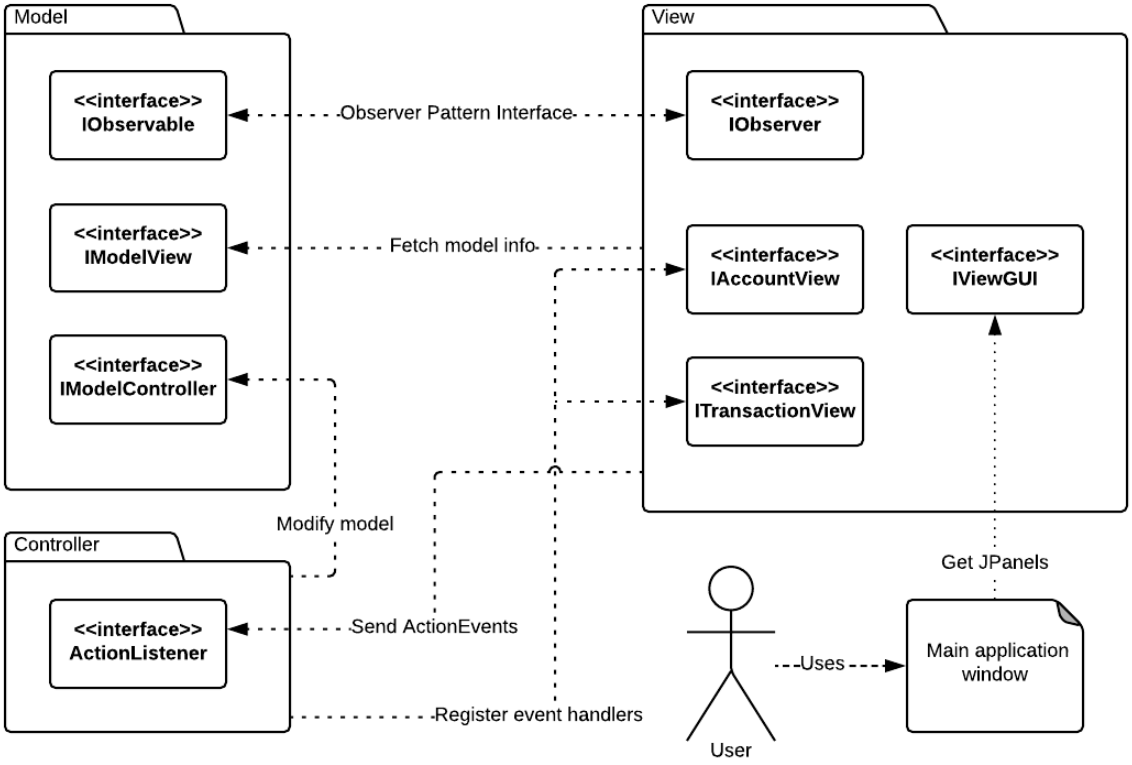

Figure 2: Subsystem specification diagram

2.2.1 Model - View : Observer Pattern

The interfaces `IObservable` and `IObserver` form the observer pattern between the model and the view.

`IObserver`

- `update()` : Called by an `IObservable` object. This should trigger internal logic in the observer to allow it to update its view on the model.

`IObservable`

- `attachObserver(IObserver)` : Attach an observer to this object
- `detachObserver(IObserver)` : Detach an observer from this object
- `notifyObservers()` : Call the `update()` method on all attached observers. Whenever the state of the model changes, it should call this function to allow its attached observers to update their views.

2.2.2 Model : IModelView Interface

The `ImodelView` interface exposes methods to allow the view to fetch information from the the model.

- `getTransactions(Integer accountId)` : Returns a list of `Transaction` objects belonging to the specified accountId. If there are no transactions for the account or the account does not exist, it will return an empty list.
- `getAllAccounts()` : Returns a list of all the `Account` objects for the current user.

2.2.3 Model : IModelController Interface

The `IModelController` interface exposes methods to allow the controller to modify the model.

- `saveTransaction(Transaction)` : Save the given `Transaction` object to the repository and update the SQL database. If the ID of the transaction is 0, create a new entry. Otherwise update the existing one.
- `saveAccount(Account)` : Save the given `Account` object to the repository and update the SQL database. If the ID of the account is 0, create a new entry. Otherwise update the existing one.
- `deleteTransaction(Transaction)` : Delete the specified transaction from both the repository and the SQL database.
- `deleteAccount(Account)` : Delete the specified account from both the repository and the SQL database.
- `importTransactions(String path, Integer accountId)` : Construct and save `Transactions` objects to the repository and SQL databse from a .csv file located at the specified path. The format of the .csv file should be well defined.

2.2.4 View : IAccountView

The `IAccountView` interface exposes methods to allow the controller to register event listeners for user actions (buttons clicks) and have access to the content of the form fields filled by the user.
The callback system uses Java's `ActionEvent` class.

- `registerAddActionCallback(ActionListener, String)` : Attach the specified listener to the GUI element that should trigger the "Add" action and set the event's action command to the specified string (should be "Add").

- `registerUpdateActionCallback(ActionListener, String)` : Attach the specified listener to the GUI element that should trigger the "Update" action and set the event's action command to the specified string (should be "Update").
- `registerDeleteActionCallback(ActionListener, String)` : Attach the specified listener to the GUI element that should trigger the "Delete" action and set the event's action command to the specified string (should be "Delete").
- `getBankInput()` : Return a string consisting of the content of the BankName field in the GUI.
- `getNicknameInput()` : Return a string consisting of the content of the Nickname field in the GUI.
- `getBalanceInput()` : Return an Integer consisting of the content of the Balance field in the GUI.
- `getSelectedAccountId()` : Return a Integer consisting of the id of the account currently selected by the user.
- `setSelection(Integer)` : Overrides the user's current account selection. This method mostly improves user experience (for example, automatically selects a new account when it is created)

2.2.5 View : ITransactionView

The `ITransactionView` interface exposes methods to allow the controller to register event listeners for user actions (buttons clicks) and have access to the content of the form fields filled by the user.
The callback system uses Java's `ActionEvent` class.

- `registerAddActionCallback(ActionListener, String)` : Attach the specified listener to the GUI element that should trigger the "Add" action and set the event's action command to the specified string (should be "Add").
- `registerUpdateActionCallback(ActionListener, String)` : Attach the specified listener to the GUI element that should trigger the "Update" action and set the event's action command to the specified string (should be "Update").
- `registerDeleteActionCallback(ActionListener, String)` : Attach the specified listener to the GUI element that should trigger the "Delete" action and set the event's action command to the specified string (should be "Delete").
- `registerImportActionCallback(ActionListener, String)` : Attach the specified listener to the GUI element that should trigger the "Import" action and set the event's action command to the specified string (should be "Import").

- `getTypeInput()` : Return a string consisting of the content of the Type field in the GUI.
- `getDateInput()` : Return a string consisting of the content of the Date field in the GUI.
- `getDescriptionInput()` : Return a string consisting of the content of the Description field in the GUI.
- `getAmountInput()` : Return an Integer consisting of the content of the Amount field in the GUI.
- `getSelectedAccountId()` : Return a Integer consisting of the id of the account currently selected by the user.
- `getSelectedTransactionId()` : Return a Integer consisting of the id of the transaction currently selected by the user.
- `setSelection(Integer)` : Overrides the user's current transaction selection. This method mostly improves user experience (for example, resetting the current selection when a transaction is deleted)

2.2.6 View : IViewGUI

The `IViewGUI` interface exposes a single method that returns a `JPanel` object. It is only used by the main application window.

- `getPanel()` : Returns the topmost parent `JPanel` of this view. It is meant to be used by the main application window to populate its frame.

2.2.7 Controller : ActionListener

The controller only needs to listen to events triggered by the user's input. From the other subsystems' perspective, it implements a single interface with a single method.

- `actionPerformed(ActionEvent)` : Event handler for `ActionEvent` events created by the view. The controller registers its handlers by using the `IAccountView` or `ITransactionView` interfaces provided by the view(s).

3 Detailed Design

This section will cover the implementation details for each subsystem. Before moving on, a quick overview of the basic types used by the system is in order.

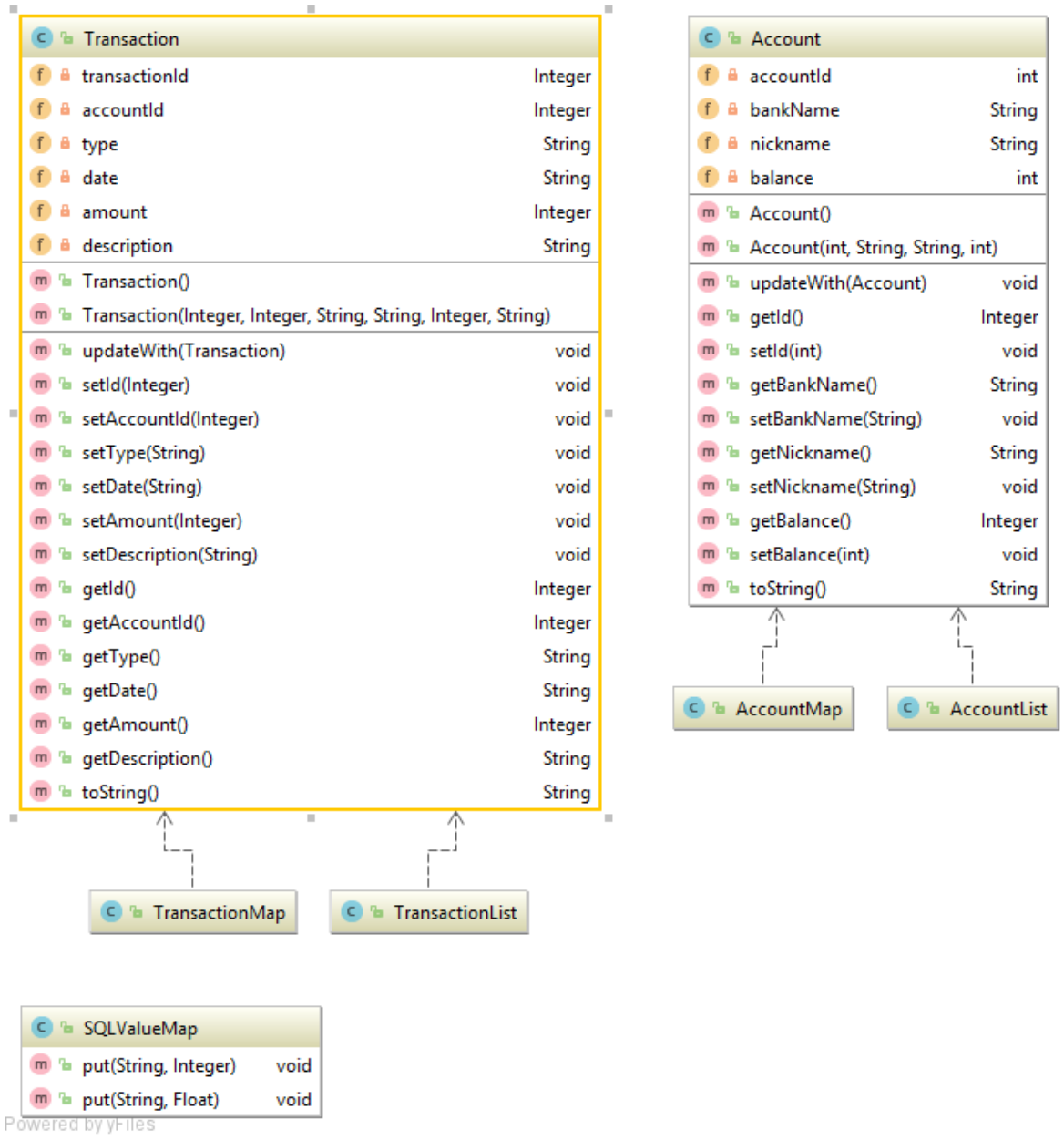

Figure 3: Overview of basic types

The `Transaction` and `Account` are nothing more than plain old data structures with public read and write access. The `SQLValueMap` is a helper class that derives from Java's `HashMap` and overrides the `put()` method to quickly convert the `value` of

the (key,value) pair into a string. `SQLValueMaps` instances are extensively used when building strings that are sent to the SQL database. The remaining `TransactionMap`, `AccountMap`, `TransactionList` and `AccountList` classes are all shorthand "typedef" classes for `HashMap` and `ArrayList` objects.

In order to simplify the following class diagrams, dependencies on these types are to be assumed and have been omitted.

3.1 Model

3.1.1 Design Diagram

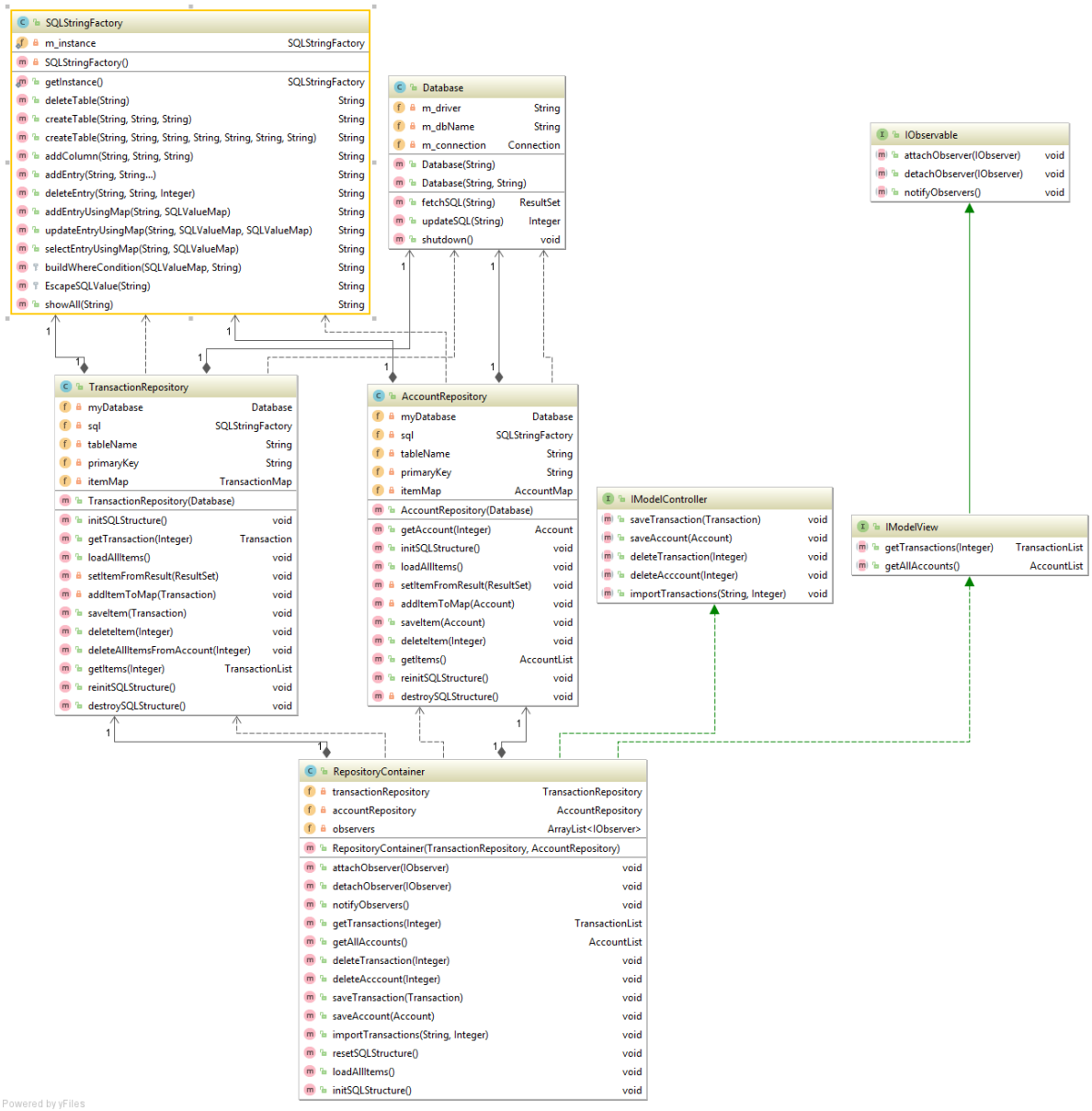

Figure 4: Model subsystem class diagram

The model subsystem consists of two main parts. The first is the `RepositoryContainer` that interfaces with the rest of the application. It owns the `TransactionRepository` and `AccountRepository` instances and passes read or write commands to each as they are received from the controllers. It also notifies the views of any CRUD operations using the observer pattern.

The second part consists of the SQL database. This is where all information is serialized to allow access between user sessions. It is continuously updated by the repositories as modifications are made. SQL queries to the database can easily be created by using the singleton `SQLStringFactory`.

From the template (delete me) — UML class diagram depicting the internal structure of the subsystem, accompanied by a paragraph of text describing the rationale of this design.

3.1.2 Units Description

3.1.2.1 SQLStringFactory.java

Class Name	SQLStringFactory.java			
Inherits				
Description	Builds formatted SQL statements for interaction with databases.			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	private	SQLStringFactory	m_instance	holds the instance of the SQLStringFactory

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	getInstance	SQLStringFactory	getter for the instance of the SQLStringFactory
	private	SQLStringFactory		constructor
	public	deleteTable	String	Creates a drop table SQL statement
	public	createTable	String	Creates a create table SQL statement
	public	addColumn	String	Creates an alter table SQL statement
	public	addEntry	String	Creates an insert into SQL statement
	public	addEntryUsingMap	String	Creates an insert into SQL statement using mapped values
	public	updateEntryUsingMap	String	Creates an update SQL statement using mapped values
	public	selectEntryUsingMap	String	Creates a select SQL statement using mapped values
	protected	buildWhereCondition	String	Generates chunks of SQL where conditions
	protected	EscapeSQLValue	String	"Cleans" SQL statements to prevent injection
	public	showAll	String	Creates a basic select all SQL statement.

3.2 View

3.2.1 Design Diagram

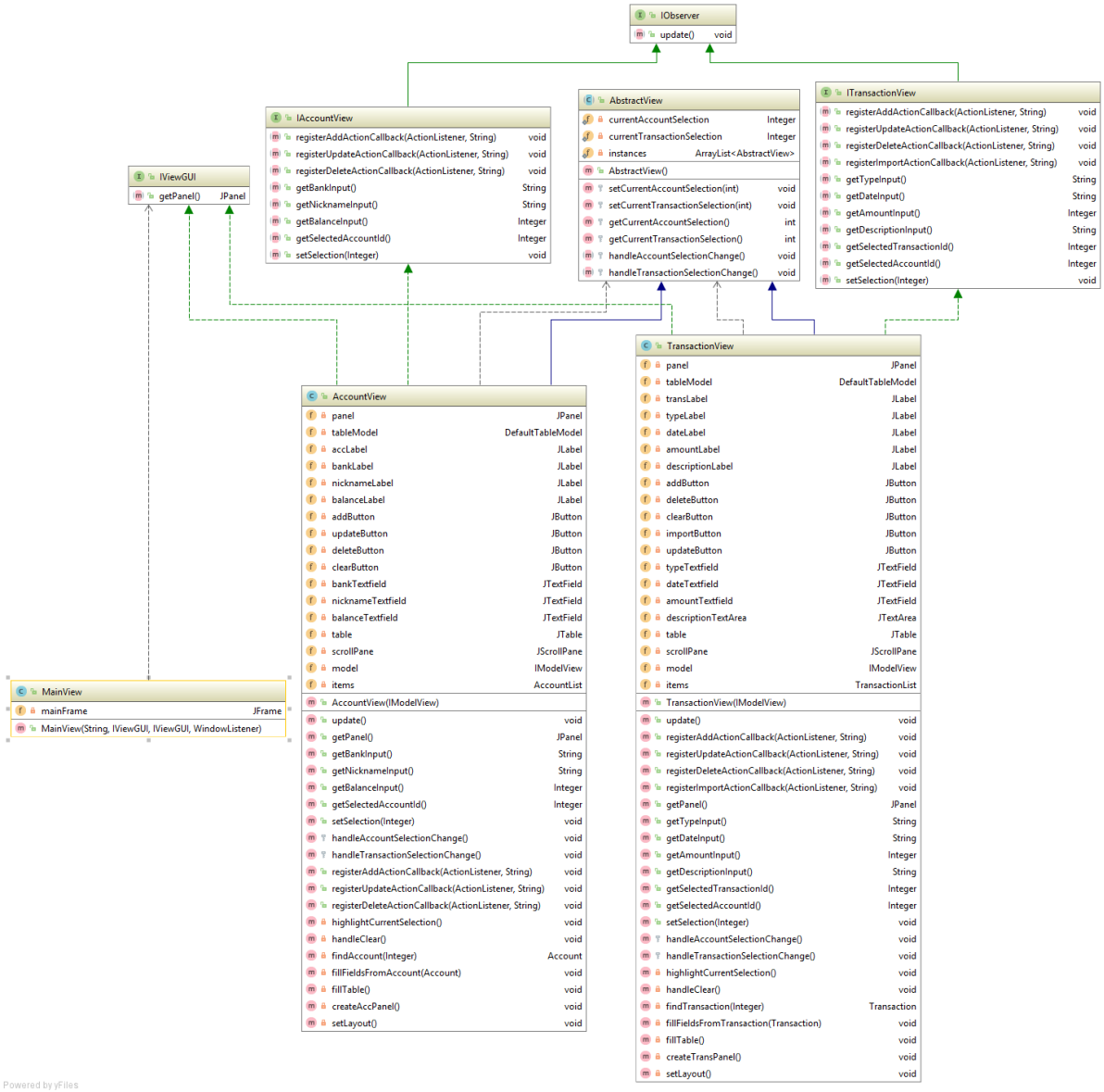

Figure 5: View subsystem class diagram

TODO - Put words here

3.3 Units Description

TODO - Put all classes from the diagram here

3.4 Controller

3.4.1 Design Diagram

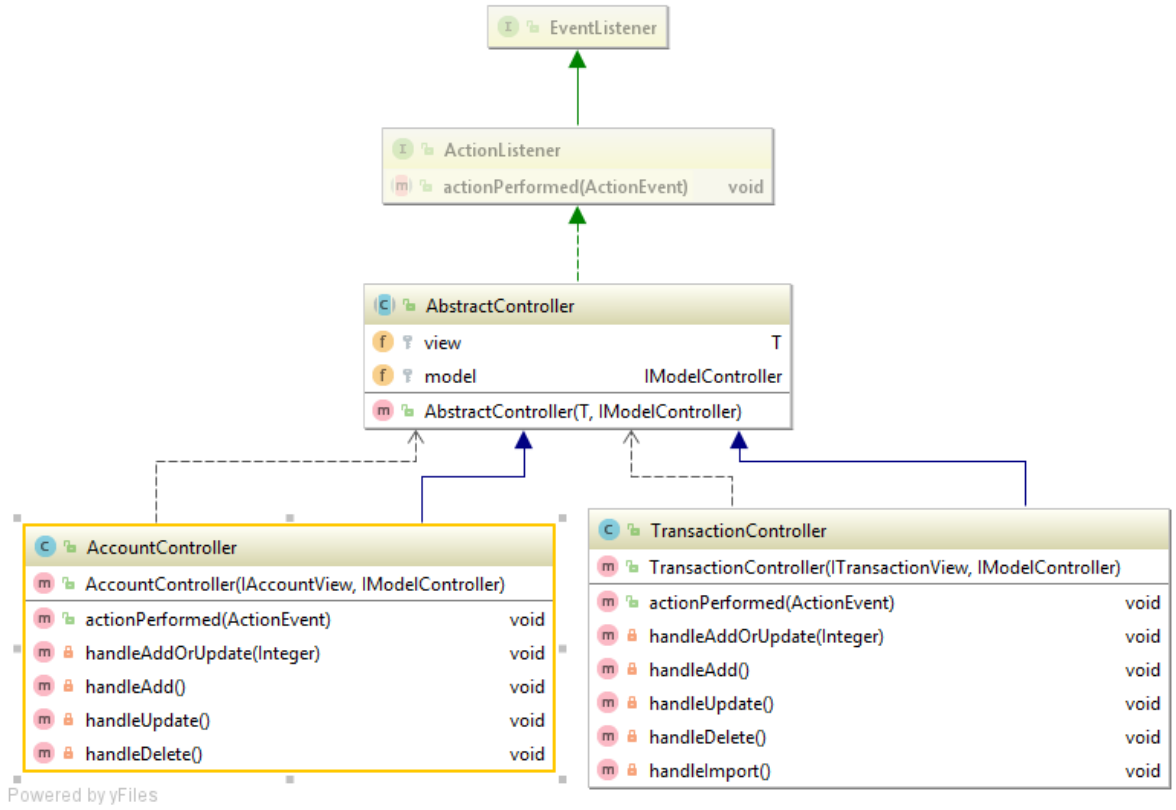

Figure 6: Controller subsystem class diagram

TODO - Put words here

3.4.2 Units Description

TODO - Put all classes from the diagram here

3.4.2.1 AbstractAppController.java

Class Name	AbstractAppController.java			
Inherits				
Description	Abstract App Controller			
Attributes				
Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	AbstractAppController		Constructor
	public	start	void	Abstract start class
	public	shutdown	void	Abstract shutdown class
	public	run	void	Abstract run class

3.4.2.2 AbstractEventListener.java

Class Name	AbstractEventListener.java			
Inherits	java.awt.event.ActionListener			
Description	Abstract Event Listener			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	package	AbstractView	view	
	package	AbstractViewController	controller	

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	AbstractEventListener		Constructor
	public	setView	void	Setter for view
	public	getView	AbstractView	Getter for view
	public	setController	void	Setter for controller
	public	getController	AbstractViewController	Getter for controller
	public	actionPerformed	void	Default message to implement this method in the view controller

3.4.2.3 AbstractModel.java

Class Name	AbstractModel.java			
Inherits				
Description	Abstract class for models			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	package	boolean	boolNew	used to determine if id has been set
	private	HashSet<AbstractView>	m_views	stores the views

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	isNew	boolean	getter for boolNew
	public	setIsNewModel	void	setter for boolNew
	public	setView	void	setter for views
	public	removeView	void	deletes views
	public	notifyViews	void	calls for update on all views

3.4.2.4 AbstractView.java

Class Name	AbstractView.java			
Inherits				
Description	Abstract view class			
Attributes				
Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	package	AbstractView	Constructor	
	public	update	void	abstract update class

3.4.2.5 AbstractViewController.java

Class Name	AbstractViewController.java			
Inherits				
Description	Abstract class for view controller			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	package	AbstractView	view	primary view
	package	AbstractView	secondaryView	secondary view
	private	boolean	controllerInitialized	determines if the controller has been initialized

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	AbstractViewController		constructor
	public	setView	void	Setter for view
	public	getView	AbstractView	Getter for view
	public	setSecondaryView	void	Setter for secondary view
	public	getSecondaryView	AbstractView	Getter for secondary view
	public	setIsInitialized	void	Setter for controllerInitialized
	public	getIsInitialized	boolean	getter for controllerInitialized

3.4.2.6 AccountController.java

Class Name	AccountController.java			
Inherits	AbstractViewController			
Description	Controller for the accounts, initializing all the form elements associated to account			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	private	UserModel	user	user model object

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	protected	AccountController		Constructor
	protected	initController	void	binds event listeners/controls to all account view elements and populates form data
	public	setUser	void	setter for user
	public	getUser	UserModel	getter for user
	private	addButton	void	Behaviour of the "add account" button
	private	updateButton	void	Behaviour of the "update account" button
	private	deleteButton	void	Behaviour of the "delete account" button
	private	clearButton	void	Behaviour of the "clear" button
	protected	getAccountDataFromAddAccountInput	AccountModel	add data inputs to account model
	private	resetAddAccountInput	void	clear the UI account inputs
	public	getAccountDataFromRow	AccountModel	Given a row number, get account data
	protected	updateDataRowFromModel	void	modify account data given UI input
	public	update	void	Update the attached models

3.4.2.7 AccountModel.java

Class Name	AccountModel.java			
Inherits	AbstractModel			
Description	Model for bank accounts			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	private	int	accountId	the Id of the account
	private	String	bankName	the name of the bank the account is held with
	private	String	nickName	the nickname for the account
	private	int	balance	the dollar balance of the account
	package	AccountTransaction-Repository	transactionsRepo	repository that holds account transaction information

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	AccountModel	Constructor	
	public	hasId	boolean	determines if an account has an Id
	public	getId	int	getter for Id
	public	setId	void	setter for Id
	public	hasBankName	boolean	determines if an account has a bank name
	public	getBankName	String	getter for bank name
	public	setBankName	void	setter for bank name
	public	hasNickName	boolean	determines if an account has a nick name
	public	getNickName	String	getter for nick name
	public	setNickName	void	setter for nick name
	public	hasBalance	boolean	determines if an account has a balance
	public	getBalance	int	getter for balance
	public	setBalance	void	setter for balance
	public	toString	String	generates a formatted output of all the account details
	public	setAccount-TransactionRepository	void	Setter for transactionsRepo
	public	getAccount-TransactionRepository	AccountTransaction-Repository	Getter for transactionsRepo
	public	getMapOf-AllTransactions	TransactionMap	gets map of all transactions
	public	getListOfAll-Transactions	TransactionList	gets list of all transactions
	public	saveTransaction	void	Saves a transaction to the repository
	public	deleteTransaction	void	deletes a transaction from the repository

3.4.2.8 AccountRepository.java

Class Name	AccountRepository.java			
Inherits				
Description	Provides functionality to a user's bank accounts database			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	package	Database	myDatabase	Database that stores account information
	package	SQLStringFactory	sql	Builds valid SQL statements
	package	String	tableName	Name of the table
	package	String	primaryKey	Name of the databases' primary key
	package	Boolean	boolAllLoaded	Have all accounts been loaded in database?
	package	AccountMap	itemMap	holds loaded account models

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	AccountRepository		Constructor
	protected	hasItemCached	boolean	checks if item is in itemMap
	public	saveItem	void	Save or update an account
	public	deleteItem	void	Deletes an account
	public	getItem	AccountModel	getter for account item
	public	getMapOfAllItems	AccountMap	getter for map of all items
	public	getListOfAllItems	AccountList	getter for all items
	protected	loadItem	void	load an account
	protected	loadAll	void	load all accounts
	protected	setItemFromResult	void	populate the model with account information
	protected	addItemToMap	void	Add an account to the item map

3.4.2.9 AccountTransactionRepository.java

Class Name	AccountTransactionRepository.java			
Inherits	TransactionRepository			
Description	Contains access to all of the transactions for an account			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	package	AccountModel	account	The account object from which we are accessing transactions

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	AccountTransaction-Repository		Constructor
	public	setAccount	void	Setter for account
	public	getAccount	AccountModel	Getter for account
	public	hasAccount	boolean	Check if account has been initialized
	public	loadAllItems	void	load all transactions for account
	public	saveItem	void	Save a transaction to account

3.4.2.10 AccountView.java

Class Name	AccountView.java			
Inherits	AbstractView			
Description	The view for accounts (accounts UI)			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	private	JPanel	panel	Various account UI elements
	private	DefaultTableModel	model	...
	private	JLabel	accLabel	
	private	JLabel	accountIDLabel	
	private	JLabel	bankLabel	
	private	JLabel	nicknameLabel	
	private	JLabel	balanceLabel	
	private	JButton	addButton	
	private	JButton	updateButton	
	private	JButton	deleteButton	
	private	JButton	clearButton	
	private	JTextField	accountIDTextfield	
	private	JTextField	bankTextfield	
	private	JTextField	nicknameTextfield	
	private	JTextField	balanceTextfield	
	private	JTable	table	
	private	JScrollPane	scrollPane	

	Visibility	Method Name	Return type	Description
Methods	public	getPanel	JPanel	getter for panel
	public	setPanel	void	setter for panel
	public	getTableModel	DefaultTableModel	getter for table model
	public	setTableModel	void	setter for table model
	public	getAccLabel	JLabel	getter for account label
	public	setAccLabel	void	setter for account label
	public	getAccountIDLabel	JLabel	getter for account id label
	public	setAccountIDLabel	void	setter getter account id label
	public	getBankLabel	JLabel	getter for bank label
	public	setBankLabel	void	setter for bank label
	public	getNicknameLabel	JLabel	getter for nickname label
	public	setNickname	void	setter for nickname label
	public	getBalanceLabel	JLabel	getter for balance label
	public	setBalanceLabel	void	setter for balance label
	public	getAccountIDTextfield	JTextField	getter for account id textfield
	public	setAccountIDTextfield	void	setter for account id textfield
	public	getBankTextfield	JTextField	getter for bank textfield
	public	setBankTextfield	void	setter for bank textfield
	public	getNicknameTextfield	JTextField	getter for nickname textfield
	public	setNicknameTextfield	void	setter for nickname textfield
	public	getBalanceTextfield	JTextField	getter for balance textfield
	public	setBalanceTextfield	void	setter for balance textfield
	public	getAddButton	Button	getter for add button
	public	setAddButton	void	setter for add button
	public	getUpdateButton	JButton	getter for update button
	public	setUpdateButton	void	setter for update button
	public	getDeleteButton	JButton	getter for delete button
	public	setDeleteButton	void	setter for delete button
	public	getClearButton	JButton	getter for clear button
	public	setClearButton	void	setter for clear button
	public	getTable	JTable	getter for table
	public	setTable	void	setter for table
	public	getScrollPane	JScrollPane	getter for scroll pane
	public	setScrollPane	void	setter for scroll pane
	public	update	void	updates the Jtable
	private	createAccPanel	void	creates the account UI elements
	private	setLayout	void	sets the visuals and grouping of the UI layout

3.4.2.11 Database.java

Class Name	Database.java			
Inherits				
Description	The database object for storing/retrieving/altering data in the various databases.			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	private	String	m_driver	database driver
	private	String	m_dbName	database name
	private	Connection	m_connection	connection to database

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	Database		Constructor
	public	getConnection	Connection	Getter for connection
	public	fetchSQL	ResultSet	Executes an SQL query
	public	updateSQL	Integer	Executes an SQL update
	public	shutdown	void	Terminates connection

3.4.2.12 DummyAppController.java

Class Name	DummyAppController.java			
Inherits	AbstractAppController			
Description	Controller for a dummy app			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	package	Database	myDatabase	the database where app data is stored
	package	SQLStringFactory	sql	Builds valid SQL statements

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	DummyAppController		Constructor
	public	start	void	starts the app
	public	run	void	initialize and run the dummy app

3.4.2.13 ImportTransaction.java

Class Name	ImportTransaction.java			
Inherits				
Description	Imports transaction data from a CSV file			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	package	String	transactionFilePath	the filepath of the csv that holds new transaction data
	private	AccountTransaction-Repository	accountTransaction-Repository	the repository that holds transaction data

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	setAccount-TransactionRepository	void	setter for accountTransactionRepository
	public	addTransaction	void	Imports transactions from CSV and stores in repository

3.4.2.14 Iteration2AppController.java

Class Name	Iteration2AppController.java			
Inherits				
Description	App controller for current project iteration			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	package	Database	myDatabase	the database where app data is stored
	package	SQLStringFactory	sql	Builds valid SQL statements
	package	AccountRepository	theAccountRespository	the repository that holds account data
	package	TransactionRepository	theTransactionRepository	the repository that holds transaction data

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	Iteration2AppController		Constructor
	public	start	void	starts the app
	protected	devStart	void	starts the app in development mode
	protected	productionStart	void	starts the app in production mode
	protected	InsertFakeAccounts	void	Adds some generic data to accounts repository for development
	public	run	void	initializes and runs the i2 app

3.4.2.15 MainController.java

Class Name	MainController.java			
Inherits	AbstractViewController			
Description	The main controller for the app			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	package	UserModel	user	the user of the app

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	MainController		Constructor
	public	setUser	void	Setter for user
	public	getUser	UserModel	Getter for user
	public	initController	void	initialized the account, transaction and main view

3.4.2.16 MainView.java

Class Name	MainView.java			
Inherits	AbstractView			
Description	The main view for the app			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	protected	JFrame	mainFrame	Swing framework main frame to display the UI
	private	String	title	The title of the main view

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	MainView		constructor
	public	getFrame	JFrame	getter for mainFrame
	public	setFrame	void	setter for mainFrame
	public	update	void	refreshes the main frame
	public	display	void	creates the UI frame
	public	setLayout	void	populates the UI frame with various UI account elements

3.4.2.17 SQLValueMap.java

Class Name	SQLValueMap.java			
Inherits	LinkedHashMap<String,String>			
Description	Shortcut class to shorten the LinkedHashMap setter and eliminate the need to type cast			
Attributes				
Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	put	void	put either float or integers as String

3.4.2.18 TransactionController.java

Class Name	TransactionController.java			
Inherits	AbstractViewController			
Description	Controller for the transactions, initializing all the form elements associated with transactions			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	private	UserModel	user	user model object
	package	int	accountIndex	the index of an account
Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	protected	TransactionController		Constructor
	protected	initController	void	binds event listeners/controls to all transaction view elements and populates form data
	public	setUser	void	Setter for user
	public	getUser	UserModel	Getter for user
	private	addButton	void	Behaviour of the "add transaction" button
	private	deleteButton	void	Behaviour of the "delete transaction" button
	private	clearButton	void	Behaviour of the "clear" button
	private	importTransactionButton	void	Behaviour of the "import transaction" button
	protected	getTransactionDataFromRow	TransactionModel	Returns transaction data based on row number
	protected	getTransactionDataFromInput	TransactionModel	Returns transaction data based on UI input
	protected	updateDataRowFromModel	void	Updates a row in the transaction table based on UI input
	protected	getAccountDataFromRow	AccountModel	Gets the account data for the currently selected row
	public	update	void	Updates the attached models.

3.4.2.19 TransactionModel.java

Class Name	TransactionModel.java			
Inherits	AbstractModel			
Description	Model for the transactions			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	package	Integer	transactionId	id of a transaction
	package	Integer	accountId	id of an account
	package	String	type	type of transaction
	package	String	date	date of transaction
	package	Integer	amount	dollar amount of a transaction
	package	String	description	text description of a transaction

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	TransactionModel		Constructor
	public	setId	void	setter for transactionId
	public	getId	Integer	getter for transactionId
	public	setAccountId	void	setter for accountId
	public	getAccountId	Integer	getter for accountId
	public	setType	void	setter for type
	public	getType	String	getter for type
	public	setDate	void	setter for date
	public	getDate	String	getter for date
	public	setAmount	void	setter for amount
	public	getAmount	Integer	getter for amount
	public	setDescription	void	setter for description
	public	getDescription	String	getter for description
	public	toString	String	generates a formatted output of all the transaction details

3.4.2.20 TransactionRepository.java

Class Name	TransactionRepository.java			
Inherits				
Description	Contains access to all of the transactions on the system			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	package	SQLStringFactory	sql	Builds valid SQL statements
	package	Database	myDatabase	Database that stores transaction information
	package	HashMap<Integer, - TransactionModel>	itemMap	Holds loaded account models

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	TransactionRepository		Constructor
	public	loadItem	void	load a transaction
	public	loadAllItems	void	load all transactions
	public	saveItem	void	Save a transaction to its account
	private	setFromResult	void	populate the model with a transaction result
	private	addToMap	void	Put a transaction in the itemMap

3.4.2.21 TransactionView.java

Class Name	TransactionView.java			
Inherits	AbstractView			
Description	The view for transactions (transaction UI)			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	private	JPanel	panel	Various account UI elements
	private	DefaultTableModel	model	...
	private	JLabel	accountIDLabel	
	private	JLabel	transactionIDLabel	
	private	JLabel	transLabel	
	private	JLabel	typeLabel	
	private	JLabel	dateLabel	
	private	JLabel	amountLabel	
	private	JLabel	descriptionLabel	
	private	JButton	addButton	
	private	JButton	updateButton	
	private	JButton	deleteButton	
	private	JButton	clearButton	
	private	JButton	importButton	
	private	JTextField	accountIDTextfield	
	private	JTextField	transactionIDTextfield	
	private	JTextField	typeTextfield	
	private	JTextField	dateTextfield	
	private	JTextField	amountTextfield	
	private	JTextArea	descriptionTextArea	
	private	JTable	table	
	private	JScrollPane	scrollPane	

	Visibility	Method Name	Return type	Description
Methods	public	TransactionView		Constructor
	public	getPanel	JPanel	getter for panel
	public	setPanel	void	setter for panel
	public	getTableModel	DefaultTableModel	getter for table model
	public	setTableModel	void	setter for table model
	public	getAccountIDLabel	JLabel	getter for account id label
	public	setAccountIDLabel	void	setter getter account id label
	public	setTransactionIDLabel	void	setter for TransactionIDLabel
	public	setTransLabel	void	setter for TransLabel
	public	setTypeLabel	void	setter for TypeLabel
	public	setDateLabel	void	setter for DateLabel
	public	setAmountLabel	void	setter for AmountLabe
	public	setDescriptionLabel	void	setter for DescriptionLabel
	public	setAccountIDTextfield	void	setter for AccountIDTextfield
	public	setTransactionIDTextfield	void	setter for TransactionIDTextfield
	public	setTypeTextfield	void	setter for TypeTextfield
	public	setDateTextfield	void	setter for DateTextfield
	public	setAmountTextfield	void	setter for AmountTextfield
	public	setDescriptionTextArea	void	setter for DescriptionTextArea
	public	getTransactionIDLabel	JLabel	getter for TransactionIDLabel
	public	getTransLabel	JLabel	getter for TransLabel
	public	getTypeLabel	JLabel	getter for TypeLabel
	public	getDateLabel	JLabel	getter for DateLabel
	public	getAmountLabel	JLabel	getter for AmountLabel
	public	getDescriptionLabel	JLabel	getter for DescriptionLabel
	public	getAccountIDTextfield	JTextField	getter for AccountIDTextfield
	public	getTransactionIDTextfield	JTextField	getter for TransactionID-Textfield
	public	getTypeTextfield	JTextField	getter for TypeTextfield
	public	getDateTextfield	JTextField	getter for DateTextfield
	public	getAmountTextfield	JTextField	getter for AmountTextfield
	public	getDescriptionTextArea	JTextArea	getter for DescriptionTextArea
	public	getAddButton	Button	getter for add button
	public	setAddButton	void	setter for add button
	public	getUpdateButton	JButton	getter for update button
	public	setUpdateButton	void	setter for update button
	public	getDeleteButton	JButton	getter for delete button
	public	setDeleteButton	void	setter for delete button
	public	getClearButton	JButton	getter for clear button
	public	setClearButton	void	setter for clear button
	public	getImportButton	JButton	getter for import button
	public	setImportButton	void	setter for import button
	public	getTable	JTable	getter for table
	public	setTable	void	setter for table
	public	getScrollPane	JScrollPane	getter for scroll pane
	public	setScrollPane	void	setter for scroll pane
	public	update	void	updates the Jtable
	private	createTransPanel	void	creates the transaction UI elements
	private	setLayout	void	sets the visuals and grouping of the UI layout

3.4.2.22 UserModel.java

Class Name	UserModel.java			
Inherits				
Description	Model for the user of the app			
Attributes	**Visibility**	**Data type**	**Name**	**Description**
	package	AccountRepository	accountsRepo	the repository that holds account data

Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	UserModel		Constructor
	public	getName	String	gets the user's name
	public	setAccountRepository	void	Setter for accounts Repo
	public	getAccountRepository	AccountRepository	Getter for accounts Repo
	public	getMapOfAllAccounts	AccountMap	gets map of all accounts
	public	getListOfAllAccounts	AccountList	gets list of all accounts
	public	saveAccount	void	Saves an account to the repository
	public	deleteAccount	void	deletes an account from the repository
	public	getAccountAtIndex	AccountModel	gets account based on row index number

3.4.2.23 Util.java

Class Name	Util.java			
Inherits				
Description	Various tools used by the app			
Attributes				
Methods	**Visibility**	**Method Name**	**Return type**	**Description**
	public	isNumeric	boolean	Check if a string is numeric

4 Dynamic Design Scenarios

To illustrate the interactions between the difference classes of our system, we have drawn sequence diagrams for the main features of our program. For simplicity, the model interface is the lowest layer of abstraction in these diagrams. The final section illustrates the internal structure of the model to complement the higher level diagrams.

4.1 Add an account

The first scenario illustrates the addition of an account to the system. The user completes the required text fields and presses the "Add" button, which sends an `ActionEvent` the `AccountController`. The controller then gathers and validates the inputs. If they are valid, the controller constructs an `Account` object and initializes it using the info gathered from the view. Finally, the object is saved by calling the model's `saveAccount()` method. See 4.5.1 for the sequence diagram from the model's perspective once the call to `saveAccount()` is made. The sequence of calls for adding a `Transaction` is fundamentally the same.

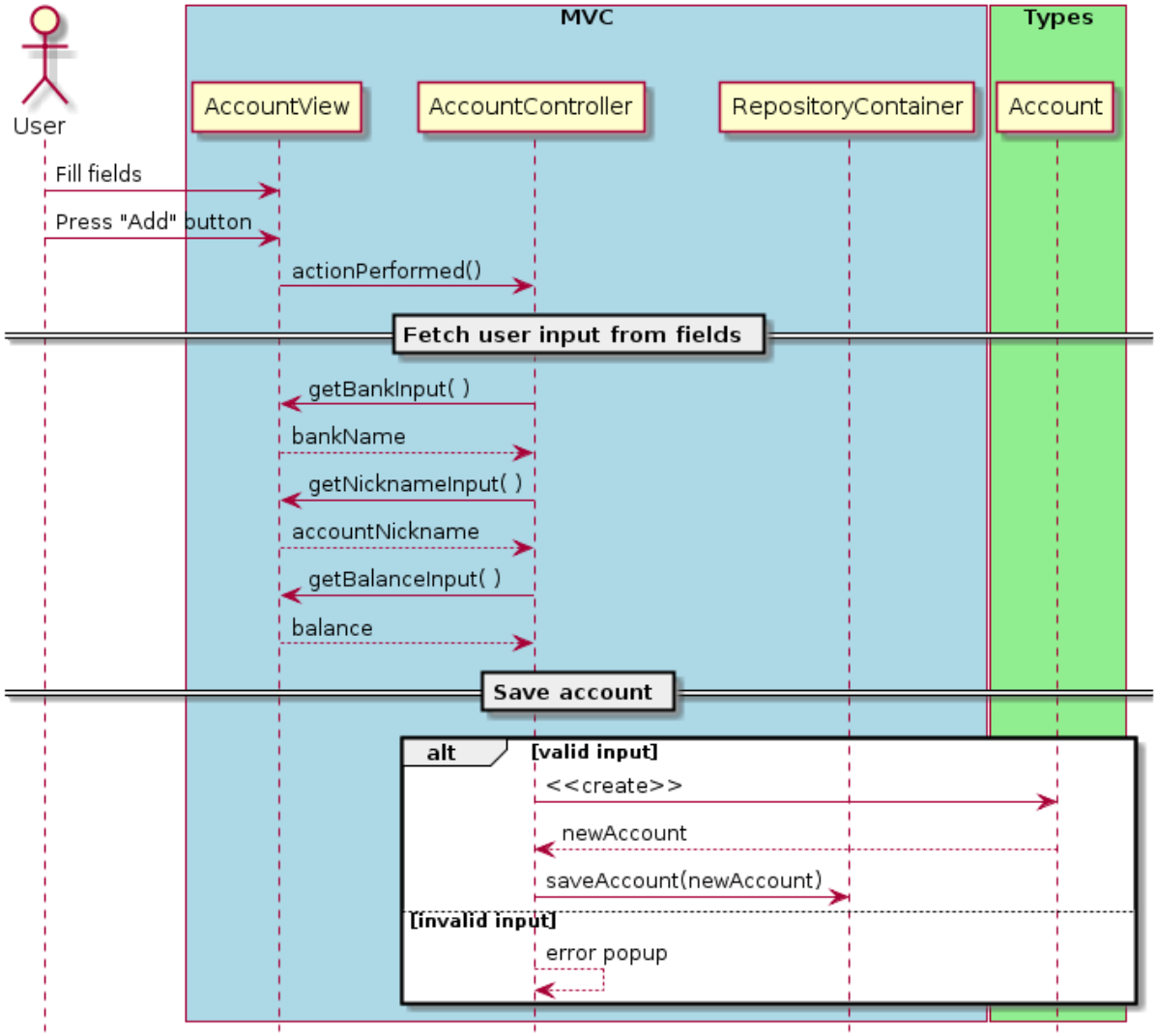

Figure 7: Adding an account

4.2 Update an account

Updating and existing `Account` is very similar to creating a new one. The one difference is that the user must first select an entry from the view. The fields will then update to show the selected `Account`'s information. The user can then modify the fields as desired and press the "Update" button when ready. The flow is then identical to 4.1, with the exception that the `AccountController` will set the created `Account` object's id from the one selected by the user. Again, the sequence of calls for updating a `Transaction` is fundamentally the same.

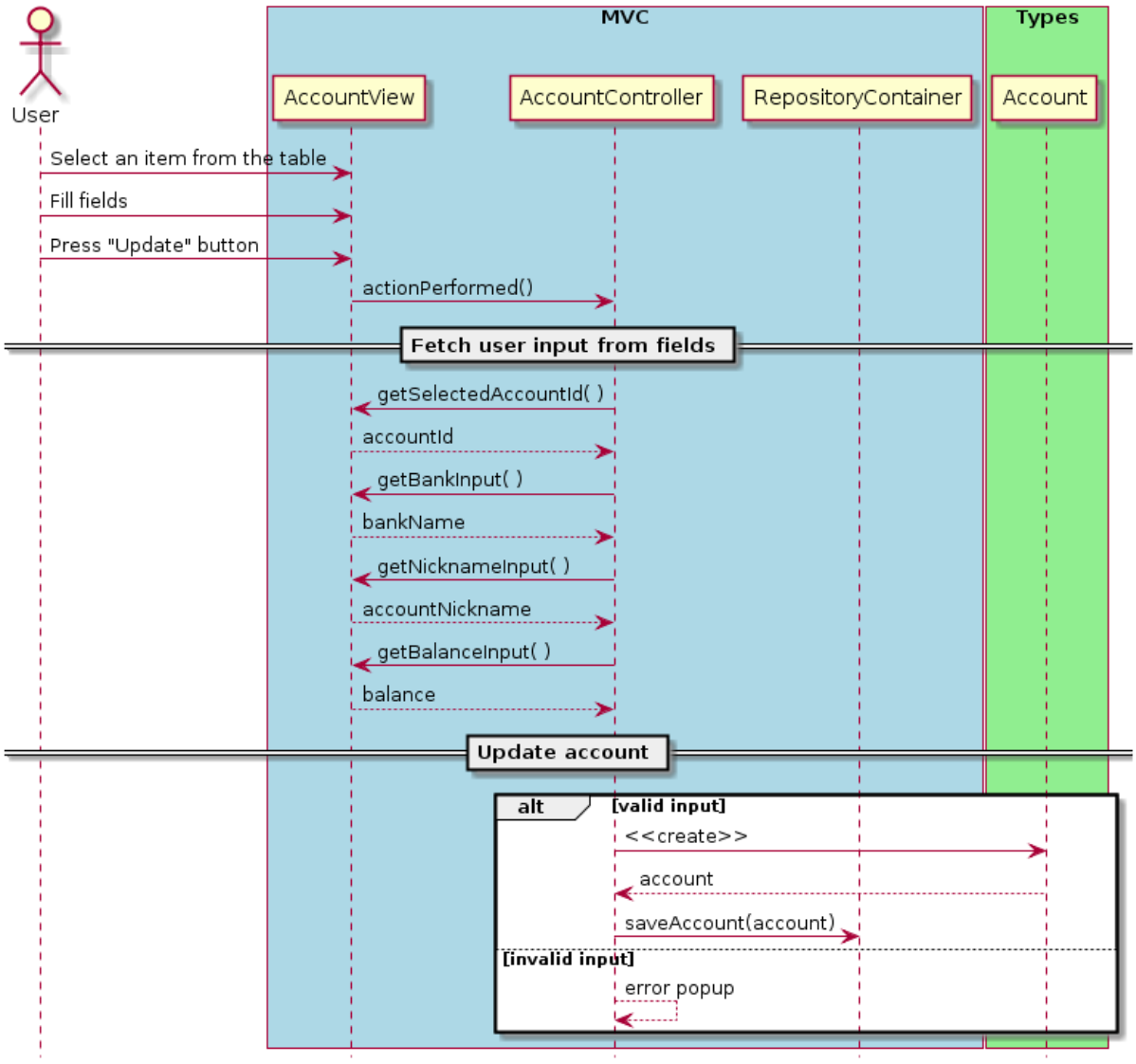

Figure 8: Updating an account

4.3 Delete an account

To delete an account, the user selects an entry in the `AccountView`'s table and clicks the "Delete" button. This sends an `ActionEvent` to the `AccountController`. The registered listener for this event then calls the `deleteAccount()` on the model. `Transaction` deletion is handled similarly.

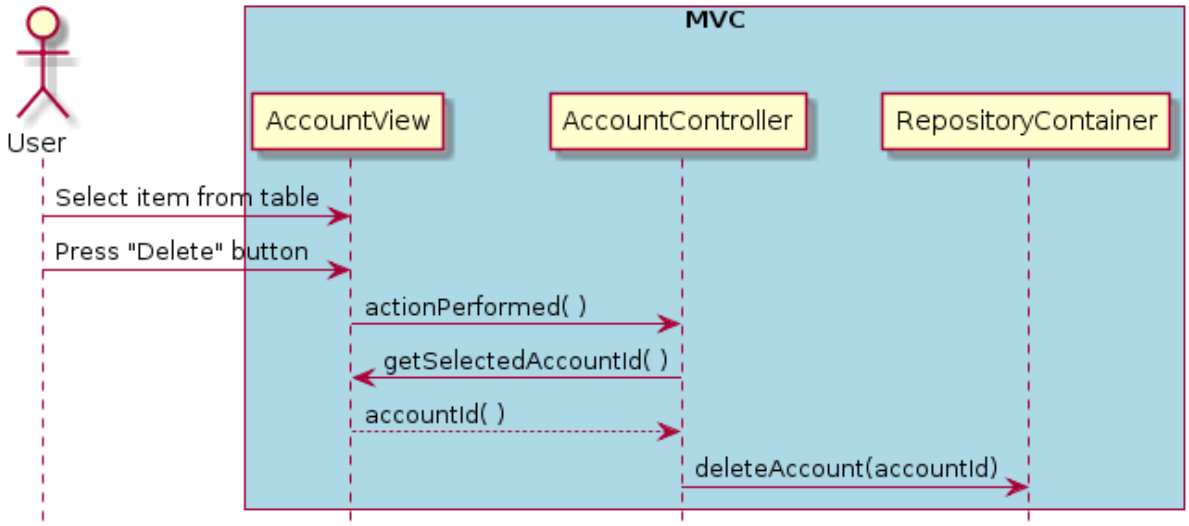

Figure 9: Deleting an account

4.4 Import a transaction list

In this scenario, the users imports a list of transactions from a .csv file and adds them to the model. When the user clicks the "Import" button, the `TransactionController` creates a window with a dialog box. The user then inputs the file path of the transaction csv file. If the file path is valid, both the file path and the currently selected accountId is passed to the model with a call to `importTransactions()`. For details on how the model then handles the import, see 4.5.2.

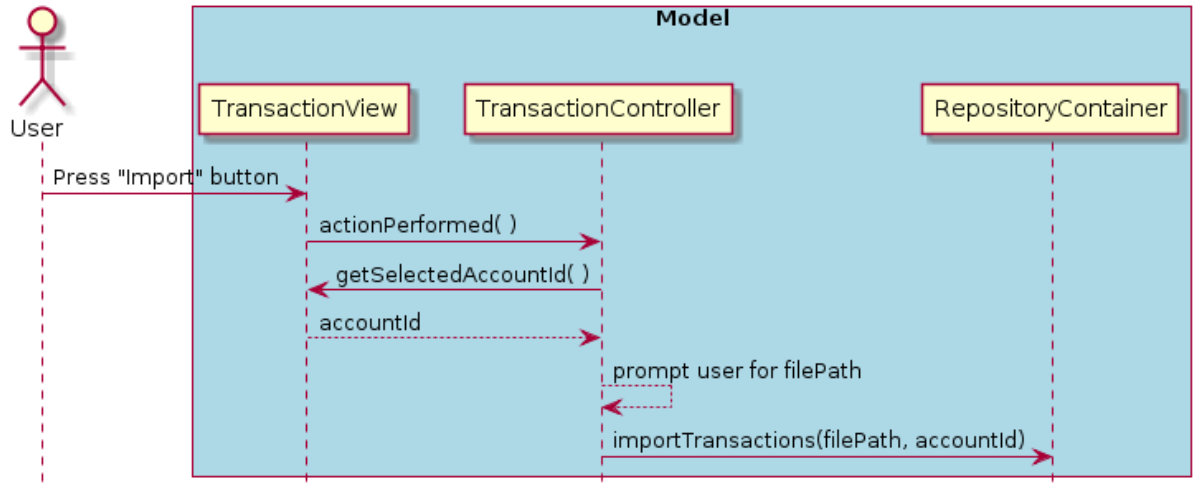

Figure 10: Import a list of transactions from .csv file

4.5 Model implementation details

The next subsections offer a glimpse into the internal logic of the model and how it implements the methods of its interface.

4.5.1 saveAccount()

The `saveAccount(Account)` method provides a simple interface for adding new `Account` objects or updating existing ones. When the specified `Account` object's `accountId` member variable is 0, it is assumed that this is a new entry. Otherwise the call is treated as an update.

After the `saveItem()` method is called, the `AccountRepository` creates an `SQLValueMap` object (linked HashMap with keys and values of type String) to store the column-value mapping.

If the account is new, then the repository uses the `SQLStringFactory` class to build a `String` insert query using the (key,value) pairs in the `SQLValueMap`. It executes the

query by calling `updateSQL` from the Database class. The repository then updates the account's ID using the accountId value returned from updateSQL and proceeds to add the new account to the repository's `AccountMap`.

If the account already exists, then the repository will construct a `SQLValueMap` for the where clause of the query using the account's ID. `SQLStringFactory` is then used to generate an update query, which will be executed by calling the `Database` class's `updateSQL` method. The returned accountId is unused.

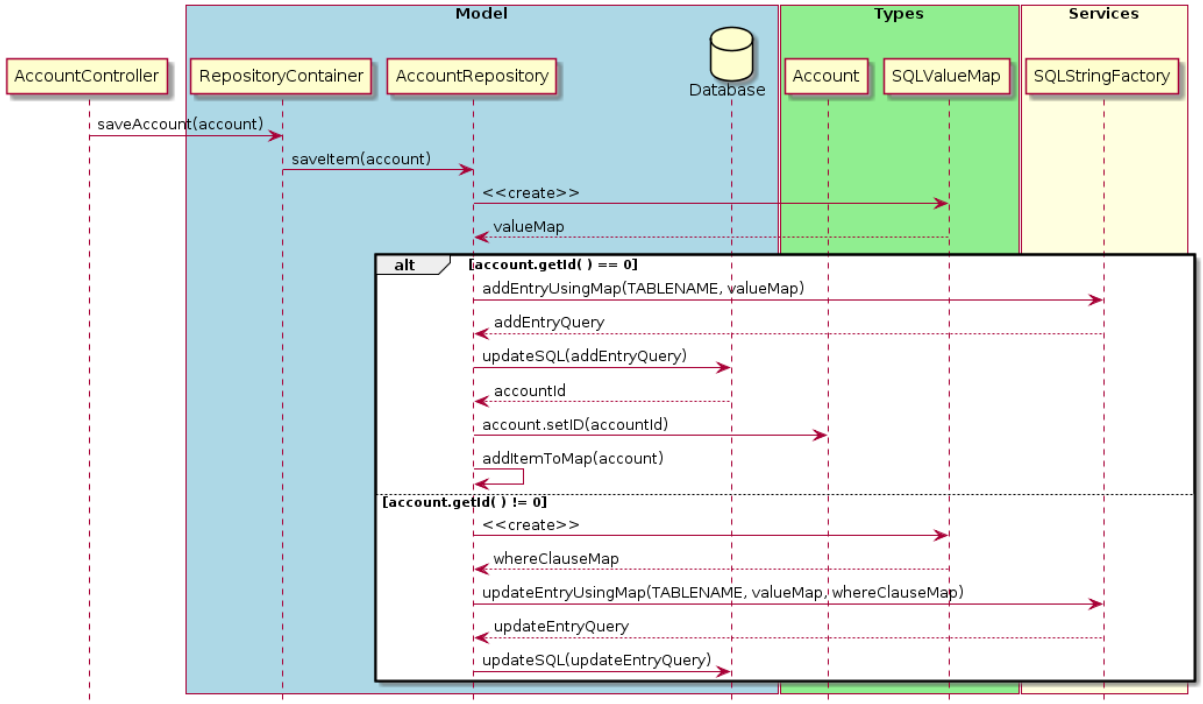

Figure 11: Model - Saving an account

4.5.2 importTransactions()

The `importTransaction()` method first creates a `BufferedReader` using the file path. Then, it iterates over the file, line by line, using the `BufferedReader`'s `readLine()` method.

The line is split into tokens using the `split()` method. Using the returned array of tokens, a `Transaction` object is constructed. The model then saves the new `Transaction` object by calling its own `saveTransaction()` method.

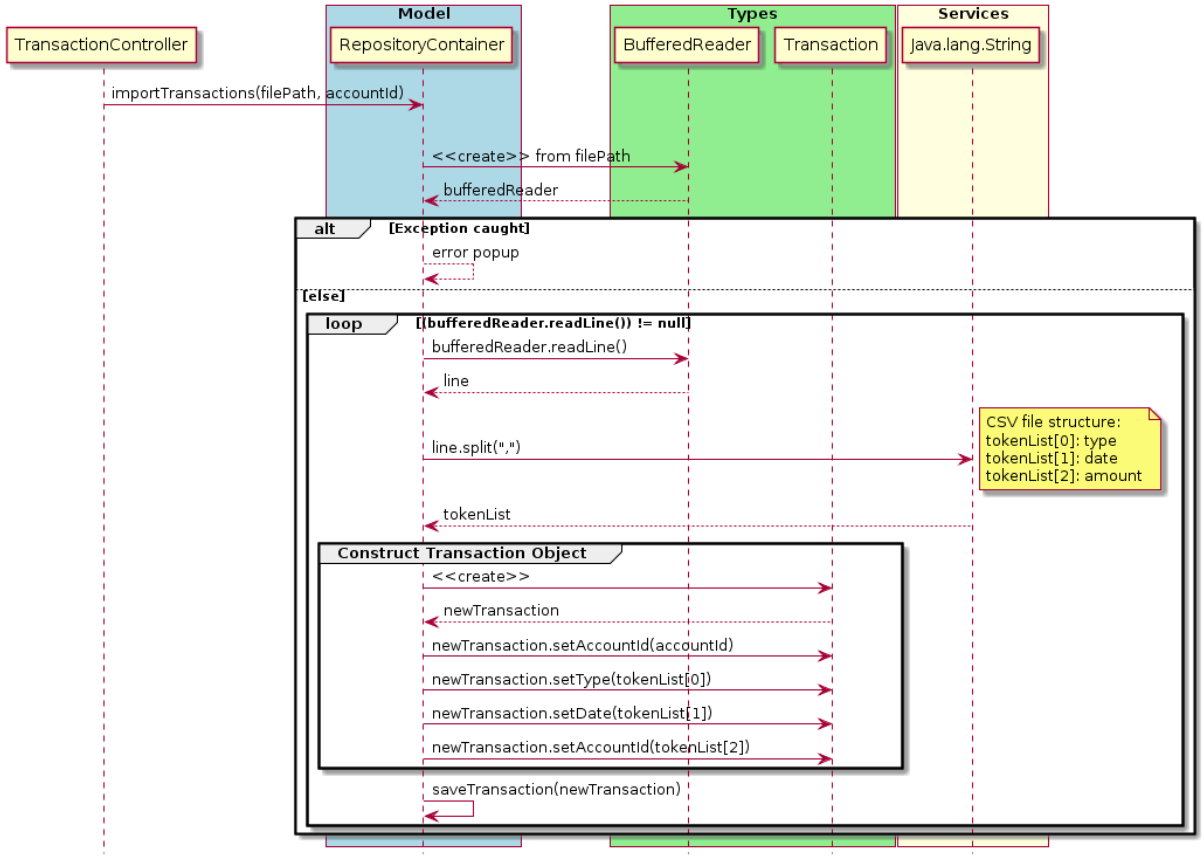

Figure 12: Model - Import list of transactions from .csv file