CS2103 Project Final Report

V0.2

t13-4j

Wong Jing Ping	A0086581W
Pang Kang Wei, Joshua	A0087809M
Hao Wei	U080159R
Liu Bohua	A0091879J
Lin Zhiyuan	A0091859M

Table of Contents

Project Scope **3**
Baseline Requirements 3
Requirements Analysis 3
Target User 3
Proposed Features 3

Design Model **4**
Domain Diagram 4
Use Cases 6
Architecture 8
Class Diagram 9

Tool Infrastructure **14**
Tools 14

Implementation **15**
Coding Standards 15

Testing **16**
Test Plan 16

Project Management **21**
Timeline 21
Roles and Responsibilities 21

Appendix **22**
User Guide 22
Class Diagram 30
Algorithms 33
Cyclomatic Complexity Diagram 36
Unit Testing 37
Bug List 40
Glossary 41

In our report, we have put the focus on showcasing the project's life cycle from the team's development perspective. As such, the bulk of the report is essentially a developer's guide, with the user guide at the end (in the appendix).

Developer's Guide

Project Scope

Baseline Requirements

EzXpns, a personalized expense management system, is conceptualized as a tool that can support the users to record all the expense items and do some simple management of the budget and expense. According to the project specifications, it should

- Support recording of various expense items
- Track user's expenses
- Generate basic reports and alerts
- Support adding, editing, and deleting of payment categories
- Allow user to undo
- Perform a simple search
- Have a simple interface as a desktop utility

Requirements Analysis

In addition to the given requirements, our team did a preliminary requirement analysis by first identifying a group of users, and then finding out what features would be useful to such a group.

Target User

For this project, we have selected a single generic working-class professional as our target user. With the busy schedule that such working adults have nowadays, coupled with the profusion of financial transactions that their lives revolve upon, it is often difficult to keep track clearly of every debit or credit entry for one's finances. Moreover, with the abundance of details that users often have to swim through to find out what they find, users can be put off by software that merely serves as a secondary memory storage for their assorted transaction. As such our group has decided to develop a simple software that is specially tailored to address the above pain points, and in the process, make expense tracking a meaningful and delightful experience.

Proposed Features

With the specific needs of the working-class adult in mind, we have decided to

- Design an inviting and user-friendly graphical user interface
- Allow users to import/export their transaction history from/to Excel

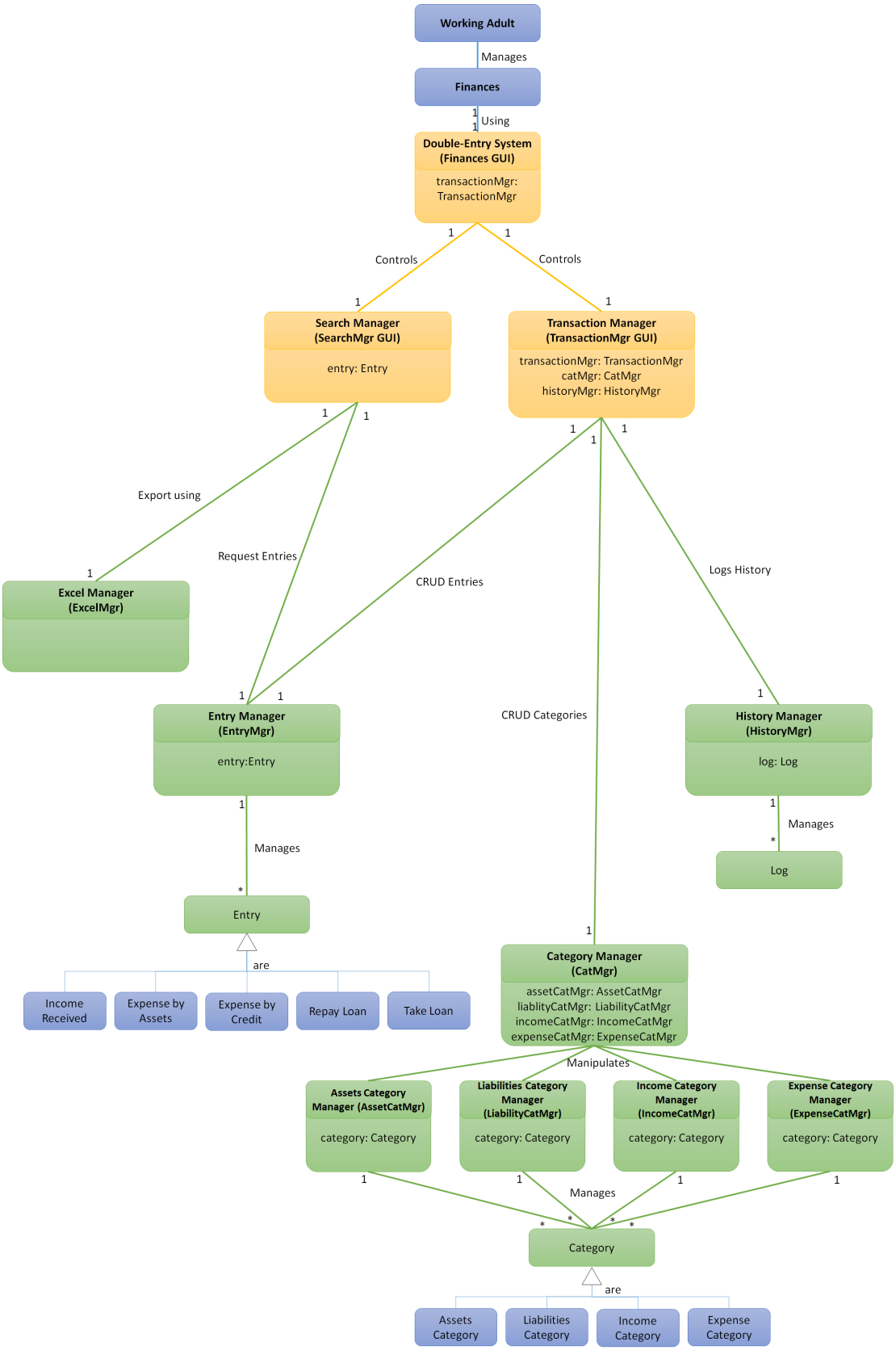
Firstly, we plan to have a sleek graphical user interface, comprising charts to show the user the state of his finances, as well as a news-feed-style scrolling-pane on the right that displays a history of his transactions in chronological order. By allowing the user to have such a 'Dashboard' view of his finances, we will inevitably remove the hassle and frustration out of the whole expense tracking experience that is commonly associated to complex expense managers.

Design Model

Domain Diagram

After establishing our target user and nailing down the features that we will be building in our project conceptually, we drafted a domain model that helped us to visualize the context of our problem.

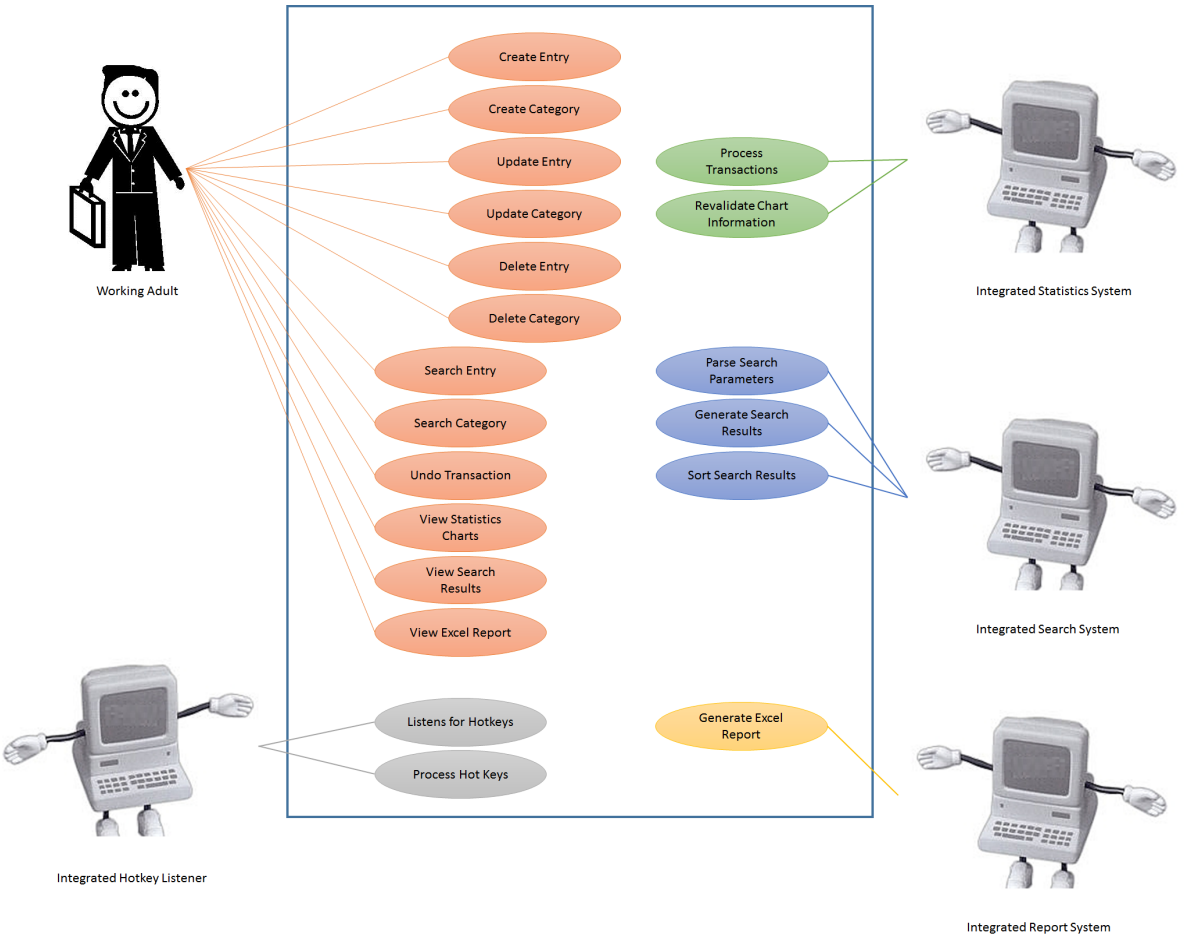
Due to the space constraint, we have put the diagram on the next page



From the domain diagram, we realized that a working-class professional typically handles 7 different types of transactions involving different accounts. To be able to capture all these diverse transactions, our team came up with the concept of a personal account assistant, that not only keeps track of the user's expenses, but also other vital information to the user, such as income, assets, and liabilities.

Use Cases

To better understand the needs of the user, and how the user will interact with our software, we sketched out a simple use case diagram to visualize the possible ways that the user could interact with our software. To further identify the system interface, we have identified 3 main use cases, and have used them to gain a better understanding of the functions (colored in blue) that needs to be implemented.



System: EZXPNS
Use Case: Create Entry (Income Received)
Actor: Working Adult
Pre-condition: -
Post-condition: Successfully adds the data of the Entry into the system
MSS:
1) Working adult chooses to add in an income entry by selecting the "Add an Entry" button in the main Finances Window
2) EZXPNS's Transaction Manager appears and requests for type of input transaction
3) Working adult chooses "Income Received" transaction type
4) EZXPNS prompts for input details of income

5) Working adult keys in the necessary details and confirms by clicking the "Add Entry" button
6) EZXPNS creates Entry into the EntryList.txt and updates the total amounts in AssetCategory.txt and IncomeCategory.txt
7) EZXPNS closes the Transaction Manager and refreshes the charts and transaction list with the updated entry
Extensions:
5a) Working adult keys in an invalid input
5a1) Transaction Manager prompts user to give a valid input and goes back to step 4)

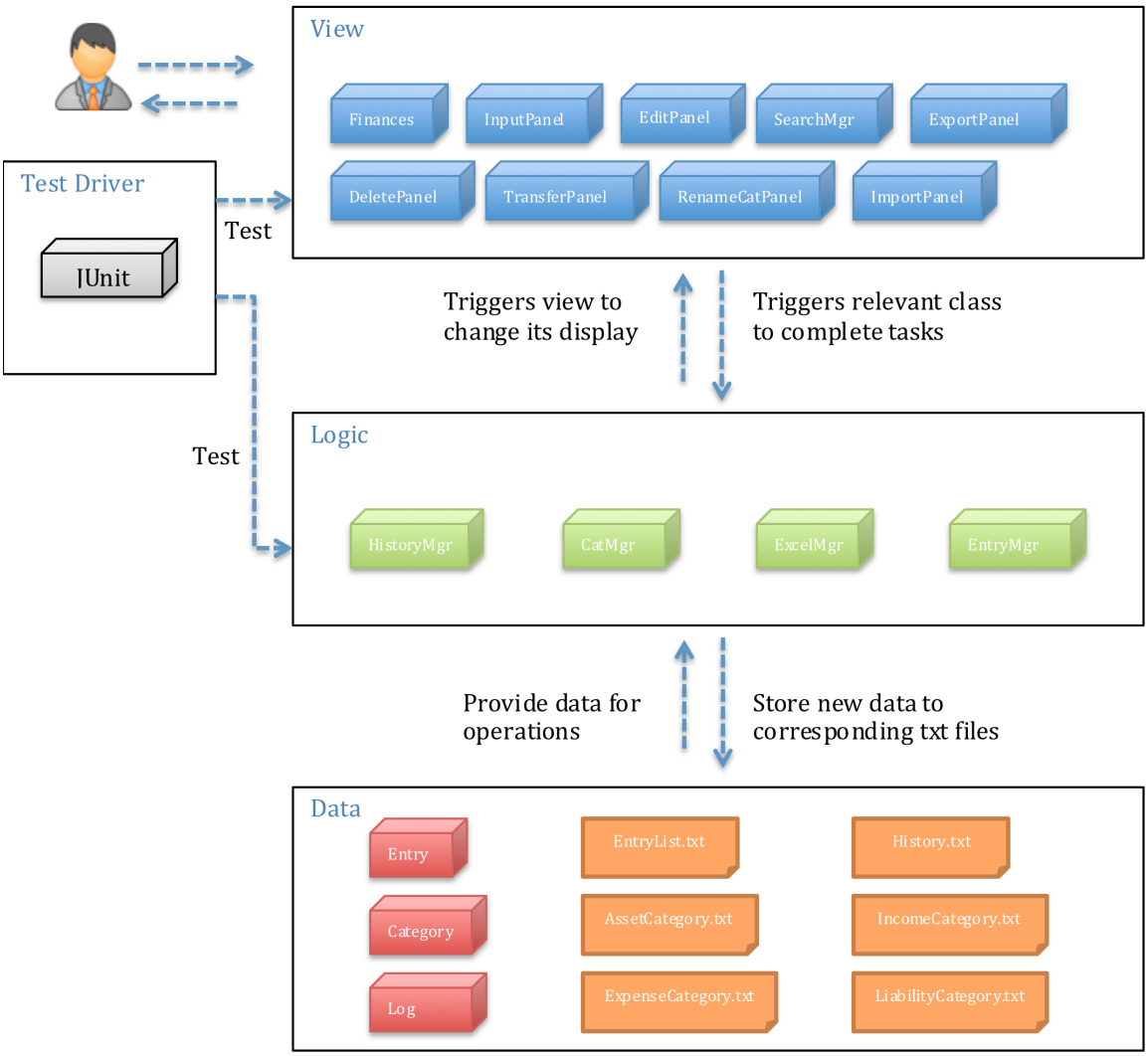
System: EZXPNS
Use Case: Updating an Entry
Actor: Working Adult
Pre-condition: User must know the transaction ID
Post-condition: Successfully updates the data of the Entry into the system
MSS:
1) Working adult chooses to update that entry by selecting "Edit an Entry" button in the Finances Window
2) EZXPNS requests for the transaction ID of the Entry to be edited in the form of a drop down menu
3) Working adult chooses the particular transaction ID
4) EZXPNS then displays the fields for that Entry and allows user to edit the fields
5) Working adult then edits the details and confirms by clicking the "Edit Entry" button
Extensions:
*a) Working adult does not know the transaction ID of the Entry which is needed to be updated
*a1) Working adult views latest transaction records in the Transaction Log Panel in the Finances Window and takes note of the transaction ID of the incorrect data

System: EZXPNS
Use Case: Searching all entries over $1,000
Actor: Working Adult
Pre-condition: -
Post-condition: Successfully returns the correct results in SearchMgr's jTable
MSS:
1) Working adult chooses to search for Entries by selecting "Search" button
2) EZXPNS's Search Manager requests for search details
3) Working adult inputs the search parameters and confirms by pressing the "Search Now" button
4) EZXPNS generates and displays the list of searched items in table form
Extensions:
3a) Working adult keys in an invalid input
3a1) Transaction Manager prompts user to give a valid input and goes back to step 2)

Given these functional requirements derived from our use cases, we proceeded to design our software's architecture.

Architecture

As our program involves a graphical user interface (GUI), we decided to adopt a 3-tier architecture.



The View layer contains the classes used for the graphical user interface. View managers are used to communicate with the user through the front-end display, by rendering the appropriate information for the user to progress through the various use cases. This will be coordinated through a central 'funneling-class', which is our Transaction Manager. Each view manager is not independent, but will communicate with the logic managers after registering user input.

Logic managers contain methods that process information from the user and the database, either for display or storage. They are in-charge of performing different transaction operations and writing information into the database, which is stored as .txt files in our case. Through data objects, which encapsulate the information in our .txt files, logic managers extract relevant information that is to be displayed on frontend display.

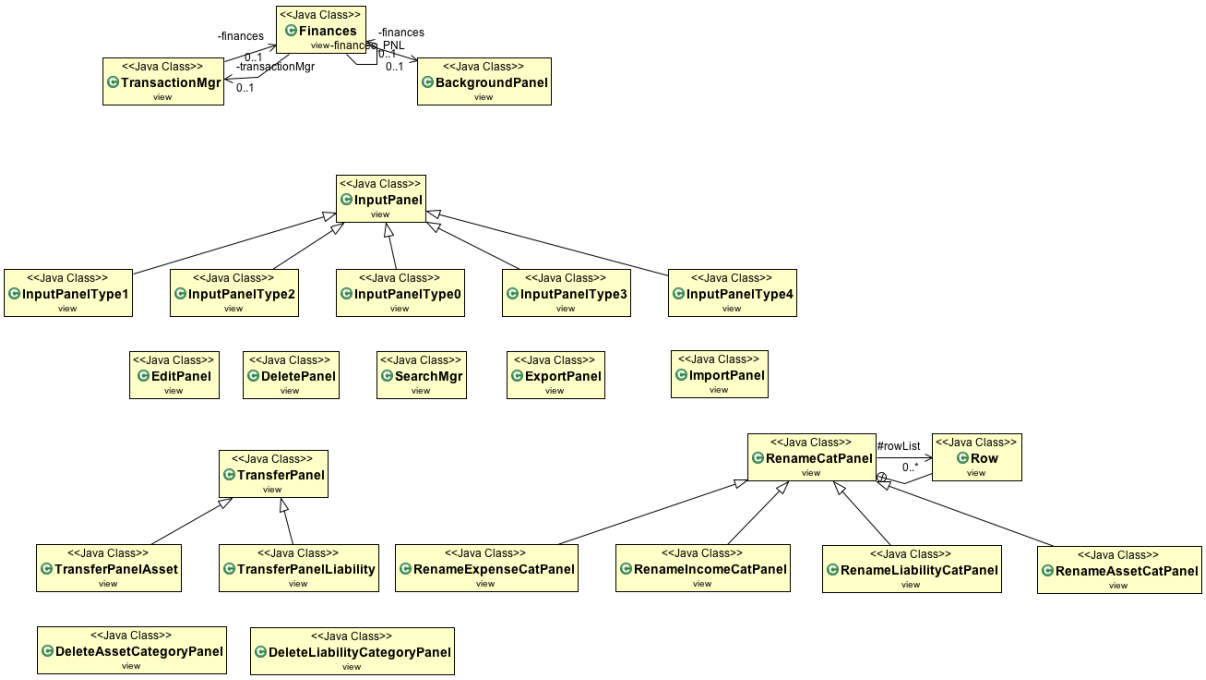
Data objects are essentially objects that encapsulate the representation of data in 3 main forms, which are category, entry and log. They allow logic managers to extract relevant information in a systematic way, thus making the functions of logic managers more focused on logical computations. Finally, we have our txt files, which contain all the transaction and history data used in the program.

Class Diagram

After designing the architecture of our software, our team proceeded to design the classes in each layer. To provide a logical organization or our code, we have packaged our code in 4 main packages, namely, view, logic, data, and test. The following packages will be elaborated in detail in the same order.

View

The user interface is designed to interact with users, as well as to display information that is relevant to users' needs.



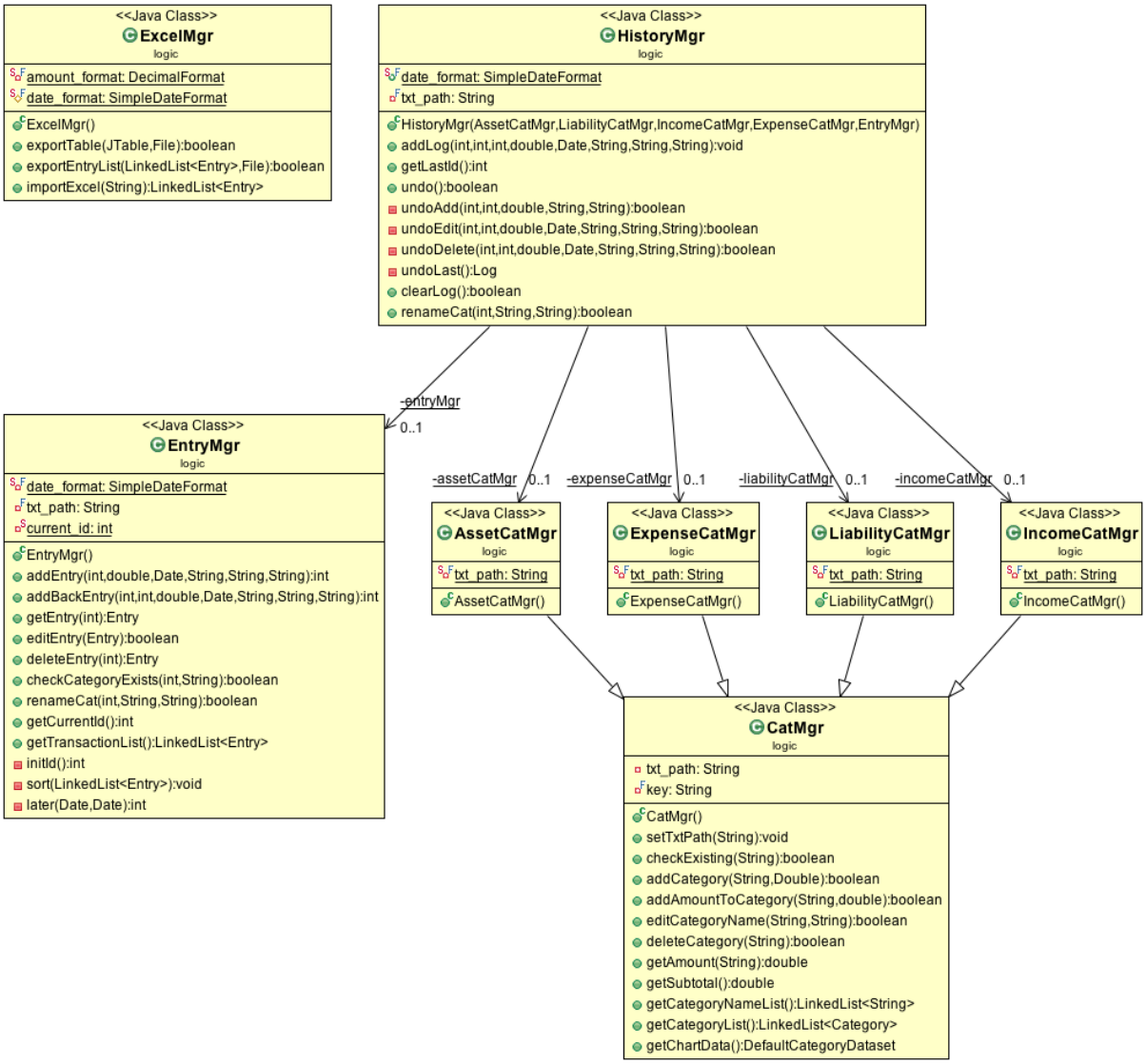
From the diagram[1], we have the following

- Finances:
  - Main GUI class that displays (contains) the main window.
- BackgroundPanel:
  - Modified version of the JPanel to support a customized background for the main window of our program.
- TransactionMgr:
  - Class that handles all the requests from the user, and directs the requests to the relevant logic classes.
  - Aided by other panel classes, which abstract the pop-up rendering and user input processing.
- XXXPanel (class name that ends in 'Panel'):
  - 18 different classes that serve to render individual operation panels.
  - Used for various user operations, ranging from creating a new transaction to exporting a list of transactions.
  - To increase the reusability and modularity of our code, as well as to apply object-oriented concepts in our program, we have designed a few parent classes such as InputPanel, TransferPanel and RenameCatPanel. This was not the case initially, where we coded out each panel class individually. However, after we had coded out the panels, we realized many similarities (in terms of attributes and functions) between the various

[1] Due to space constraints, we have left out the attributes and functions from this diagram, but included a comprehensive diagram in the appendix.

panel classes. Hence, we decided to refactor our code to reduce the clutter, as well as organize it in logical sections to facilitate integration.

Logic



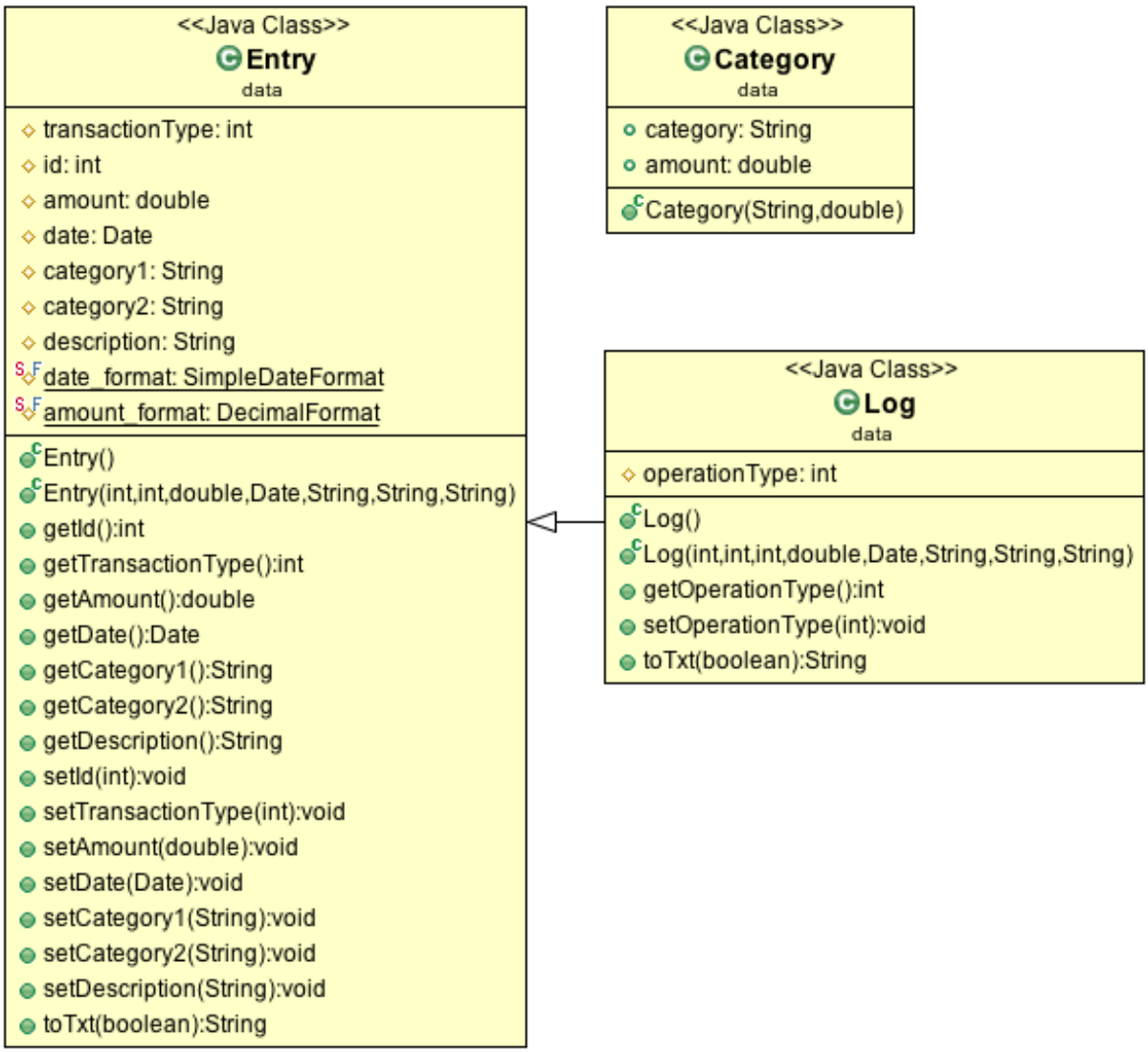
There are 4 Logic classes in EzXpns. They are the CatMgr, EntryMgr, HistoryMgr and ExcelMgr. CatMgr extends to 4 sub-classes that are used to handle the 4 main types (Assets, Liabilities, Income, and Expenses).

- CatMgr
  - Parent class that extends to 4 sub-types.
  - Manages information related to a type (Asset, Liability, Income, Expense).
  - Accesses data from *Category.txt files using Category object in Data package.
  - Alive throughout run-time.
- EntryMgr
  - Class that manages list of transactions from the user.
  - Accesses data from EntryList.txt using Entry object in Data package.
  - Alive throughout run-time.
- HistoryMgr
  - Class that manages history of operations from the user.

- Handles undo feature.
- Instantiated with other logic classes to update the relevant data when performing 'undo' function.
- Alive throughout run-time.

- ExcelMgr
  - Class that handles exporting and importing of data to and from Excel.
  - Instantiated only upon functional activation to reduce run-time memory usage.

Data



To support data encapsulation, we have the following data classes:

- Entry
  - Class that handles encapsulates data related to a given entry.

    - Instantiated when reading/writing information from EntryList.txt or user input.
    - Instantiated only upon functional activation to reduce run-time memory usage.
- Log
    - Class that encapsulates data related to a log in History.txt
    - Extends Entry due to similar fields.
    - Instantiated when reading/writing information from/to History.txt
- Category
    - Class the encapsulates data related to a category in *Category.txt
    - Instantiated when reading information from *Category.txt

Tool Infrastructure

Tools

The following table shows the types of tools that we will be using to develop this software. Our project will be coded in Java, and we will be using Eclipse as our main development tool. To enhance our GUI, we will be importing libraries from jFreeChart for chart rendering, jExcelApi to handle data transfer with Excel files, and MigLayout which will be the standard API we use for our GUI formatting.

Development Tool	Eclipse Juno & Windows Builder
Third-party Library	jFreeChart JExcelApi MigLayout
Versioning Tool	TortoiseHG & Google repository
Team communication & documentation tool	Google groups & Google docs

Implementation

Coding Standards

Naming Conventions

In GUI classes, we have standardized a naming convention. All GUI elements will be named in this way: [classname][what we want to call it]_[shortform of the type]. For example, a JPanel containing asset information in the Finance class will be named financeAsset_PNL.
The shortforms will be as such:

Shortform	Description
PNL	Panel
LBL	Label
SCP	Scrollable Panel
FRM	Frame
CB	Combo box (drop down menu)
BTN	Button
TF	Text Field

Code Organization and Documentation

In all classes, the methods are ordered as follows:

1) Class Constructor
2) Public setters and getters
3) Public class-specific methods
4) Private methods

In each class, every method is javadoc-commented, to allow easy generation of javadocs (software API) for project documentation.

Design of Algorithms

A detailed description of the main algorithms used is located in the appendix. For the full API, it is accessible on the google code repository, at
https://code.google.com/p/cs2103jan13-t13-4j/source/browse/#hg%2Fdoc

Testing

Test Plan

Testing will consist of three phases that will build on each other. To minimize the effort in creating extra stubs for testing, our team has used the **bottom up approach** in testing the various portions. From the domain diagram[2], one could see that the top classes, which are the GUI managers (in yellow), requires the bottom classes, non-GUI (in green), in order to work. Thus, we decided to ensure the lower level managers were tested properly before moving upwards.

Each of these phases and their objectives are represented in the following flow-chart:

Phase 1

Tools: JUnit test drivers + individual classes
Purpose: Identify logic errors in the backend classes
Identify design problems in the frontend (GUI) classes

In the 1st phase, we will be unit testing for each of the classes in our program. We begin with JUnit tests to test individual classes in the logic and data package, followed by some manual testing for the GUI classes in the view package. As the classes in the logic package will require the use of the classes in the data package, we will be testing the data classes first to minimize the need for extra stubs during testing.

First, we had to ensure that Entries, Categories and Logs could be read and manipulated. As creating, reading, updating and deleting these data are of utmost importance in such an application, we had simulated dummy Entries, Categories and Logs, to check whether the Entry Manager, sub Category Managers and History Manager could handle their functionalities properly.

At this level, a significant portion of our tests deal with boundary testing of inputs, and we have given a few examples of the boundaries we evaluated.

Equivalence partitioning, Boundary testing (Where user inputs are required):
Amount in creation and editing of a transaction:

1) amount < 0 (disallowed negative value)
2) **0 ≤ amount ≤ 21,474,836.47 (max integer value / 100) (acceptable values)**
3) amount > 21,474,836.47 (defined if statement if user keys in more than such an amount)
4) NaN (not a number, catch exception)

[2] Page 5

Date in creation and editing of a transaction:
Let x be the amount to be keyed into the transaction

1) date (day or month or year) < 1 (invalid date, catch exception)
2) **date (day or month or year) is a valid date (acceptable values)**
3) date (day or month or year) has no such date (invalid date, catch exception)
4) date (day or month or year) > current date (disallow future transactions)

Renaming of a Category:

1) name is an Integer (disallowed)
2) **name is a String with at least 1 character (acceptable values)**
3) name is a String of symbols only (disallowed)

In addition to evaluating the boundaries of user inputs, we also plan to test the logic sequences in our program, to ensure that all possible paths in the program have been covered at least once. To do so, we first evaluated the cyclomatic complexity for a Basis Path Test for creating any transaction.
Cyclomatic Complexity: c = 43 – 33 + 2 = 12[3]

As we realized that we would not have time during the semester to finish testing all the possible paths, we have only attempted only the paths that deal with the main success and a few important alternate scenarios in our use cases, and omitted the remaining paths in our testing[4]. We recommend any developer who will be taking on the project to conduct a full-scale white-box testing with all the paths, to ensure the robustness of the shipped software.

During this phase, we identified approximately 80% of our total logical errors in our backend classes, and rectified the problems in the respective classes accordingly. To maximize our efficiency in the process, we applied the Pareto principle, which meant that we did not document every bug issue that we found, but only the 20% of bugs that caused a significant bulk of the problems[5]. This was to prevent information clutter, and to raise the awareness of key issues that each of the developers would face as they proceeded with their coding/testing.

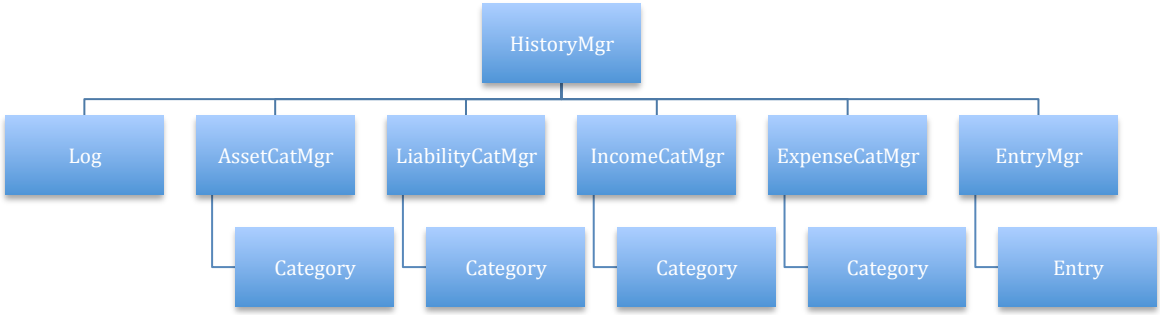
In addition, by using JUnit test drivers, we were able to identify the source of our errors very quickly, and make the appropriate edits in a timely manner. This prevented us from digging our code extensively during the next phase of testing, thus saving precious time downstream.

[3] Due to space constraints, we have included the control flow diagram in the Appendix.
[4] We have included an example of our unit test in the Appendix.
[5] The list can be found in the appendix, after the user guide.

Phase 2

Tools: JUnit test drivers + groups of classes
Purpose: Identify logic errors in the backend classes
Identify integration issues when connecting classes together

At this stage, we used grey box testing, which is a mixture of black box and white box testing, in order to encounter a wider range of problems we would encounter, thus allowing a better debugging of the software application. As we did not test every single possible path in the previous phase, the value of a full white-box test in this phase would have limited benefits. In the interest of time as well, we tested out the main paths involved for the main success scenarios and the few standard alternate scenarios, and left out the rest for future testing. We continued to use JUnit test drivers, but each time, testing logical sets of classes that depended on each other. And example would be the testing of the HistoryMgr:



Phase 3

Tools Eclipse debugger + whole program
Purpose: Identify logic errors in the backend classes
Identify layout/graphic problems in the frontend (GUI) classes
Identify

In the 3rd phase, our development team will be running and testing out our program as a whole to verify that the functions, which we initially planned to build, are functioning properly. We will be using the use cases identified in our requirements analysis to perform the testing for this phase.

This testing phase helped us identify a few minor functional mismatches between the steps, such as classes not communicating with the same data, and showed up a few more logic errors related to the editing and deleting of transactions.

An example of how we tested our first use case (adding a new transaction) is as follows:

Start program

Program should display all the icons, charts, and the transaction list on the right

Click on the 'Add Transaction Button'

Button should respond with a new pop up window requesting for transaction type	Main window should be inactive (not click-able)

Select transaction type

The 5 different transaction types should be rendered in the combo box

Click 'Select'

The appropriate fields for each transaction type should be rendered (Asset/ Liability/ Income/ Expense)	The InputPanel should have the matching colour for selected transaction type	The date field should be auto-populated with today's date

Fill in the fields

The amount, description, date fields should be editable	The edit-able and delete-able categories should be populated in the drop down menu

Click 'Submit'

The relevant errors entered should show up in red, and the pop up window should not close	If successful, the pop up window should close, the main window reactivates, and the charts and transaction list updates accordingly

Phase 4

Tool: Exported program
Purpose: Product validation

In the 3rd phase, we evaluated the features of the program against the original set of requirements once more, to ensure that we have met the necessary acceptance criteria. Our evaluation is as follows:

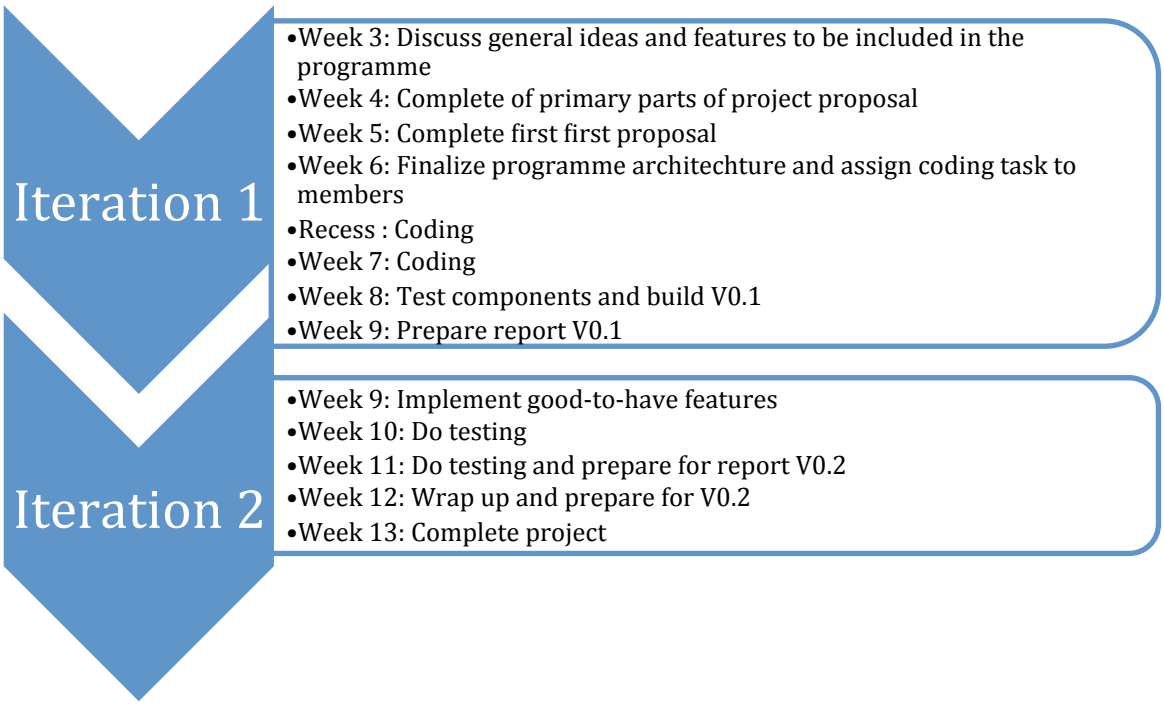
1) Support for recording of various expense items
✓ Supports recording different types of categories, not only expense, but also income, asset and liability categories. Also supports creating of sub categories in each of these 4 main categories.
2) Track expense, where is the money being spent
✓ EzXpns stores the user's data such as expenses. When the user needs to track an expense, he/she is able to use the visual representations in the application to receive timely feedback. Additionally, the user can not only view his/her expenses, he/she will be able to view related information of his/her current assets, liabilities, and sources of income, which gives a more holistic picture of the user's financial status.
3) Basic reports and alerts
✓ Bar charts are constantly refreshed and updated at the instance of a new transaction being made.
✓ Excel reports also can be generated and exported.
✓ A pop up informs the user when he/she has a negative net balance.
4) Adding, Editing and Deleting of Categories and Entries
✓ EzXpns allows the adding, editing and deleting of transactions. During a transaction, Entries and Categories are constantly created, read, updated and deleted according.
5) Undo Operations
✓ EzXpns allows multiple undo operations of the transaction made in the current session of run time, which include add, edit and delete operations.
6) Simple Search
✓ A search manager in GUI form has been created for easy use of searching for particular information by allowing search parameters. These search results could also be generated into an Excel Report.

If we have time in the future, we will be rolling out our program to prospective users of our identified demographics, which are working adults in general, and solicit feedback from them during their use. This phase is synonymous with user acceptance testing, which is to ensure that our program meets the expectations of the prospective users, and to demonstrate the capabilities that we have initially set out to build.

Project Management

Timeline

Our project development life cycle generally followed a waterfall model, where we freeze the previous part before proceeding to the next. This worked out well at times for us, but not at the start, as when we realized that our software did not meet some of the project requirements, we had to begin from scratch. Fortunately for us, we had enough time to catch up with our coding, and managed to stick to our project timeline.



Roles and Responsibilities

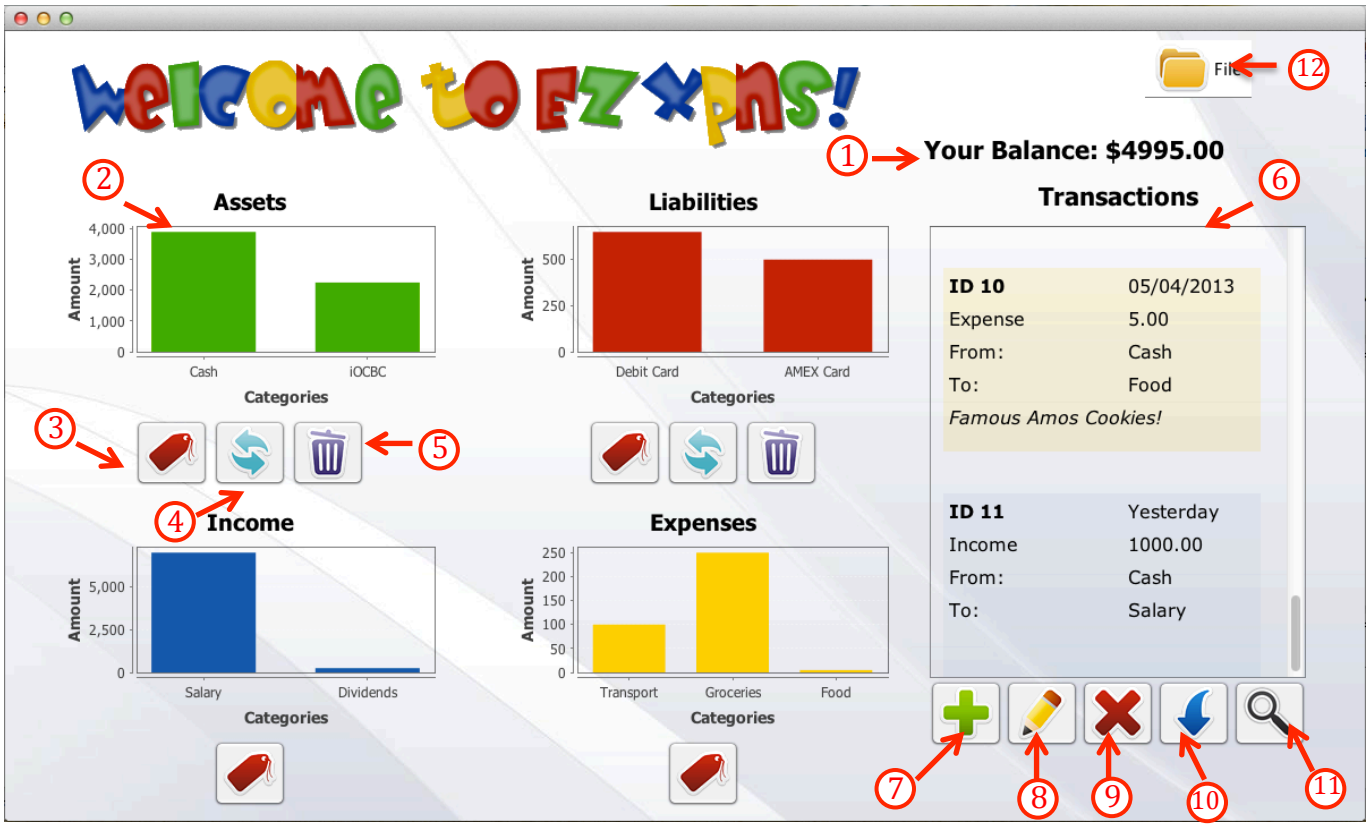
Team Member	Role
Liu Bohua	Timekeeper (ensure we meet our deadlines) and program tester
Lin Zhiyuan	Program tester
Pang Kang Wei, Joshua	Program code
Wong Jing Ping	GUI designer
Hao Wei	Documentations

Appendix

User Guide

Overview of Main Panel

EzXpns has a very interactive and easily understood Graphic User Interface (GUI). Users can get all the information at a glance from the main user panel. Various transaction operations are to be done in a separate pop-up window. Pressing the respective button can easily activate these pop-up windows. The diagram below is a screenshot of the main user panel and its important information can be found in the table after the diagram.

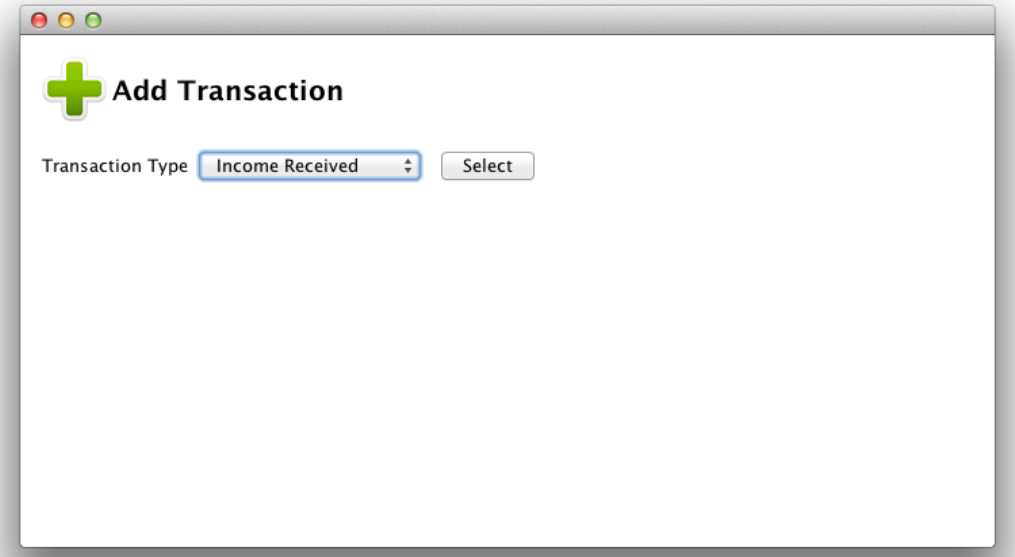


The functions related to the respective buttons are elaborated in a table in the next page.

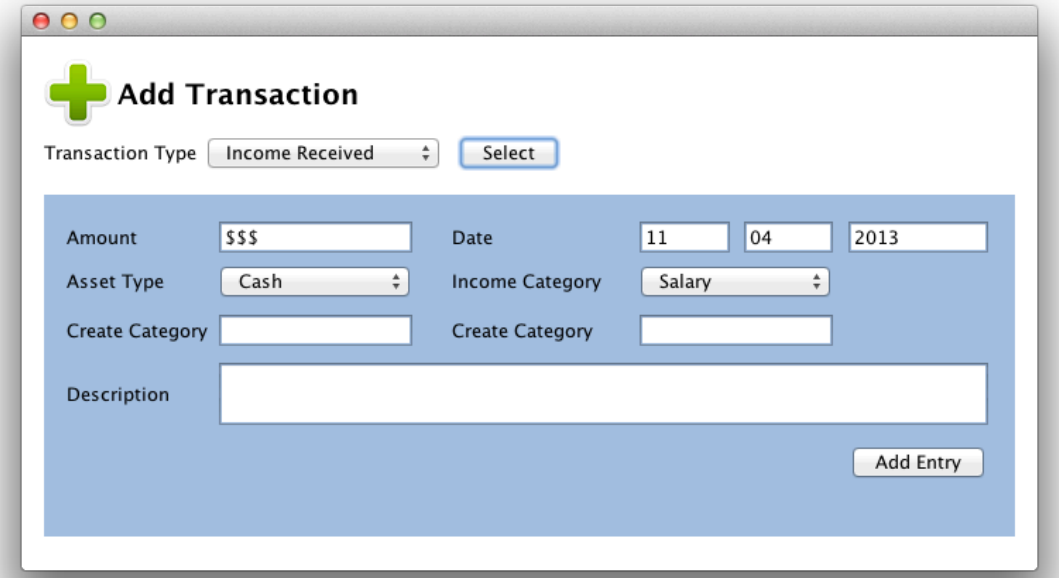
Number	Name	Description
1	Balance Field	To show the balance according to the formula: Balance = Asset - Liability
2	Asset/ Liability/ Income/ Expense Panel	Use bar charts to display the values of different categories within each panel
3	Rename Asset/ Liability/ Income/ Expense categories	Press the button to change the name of a category for asset/liability/income/expense
4	Intra-transfer asset/ Liability	Press the button to transfer money from one asset/ liability category to another
5	Delete Category	Press the button to delete a category name permanently
6	Scrollable Transaction List	To display various transactions made by the user. Different color codes represent different transactions. Each transaction is labeled with an ID number.
7	Add Transaction	Press the button to add new transaction
8	Edit Transaction	Press the button to edit an existing transaction details
9	Delete Transaction	Press the button to delete an existing Transaction
10	Undo Transaction	Press the button to undo the last transaction
11	Search	Press the button to search for a particular transaction based on its date, transaction amount, description and transaction type
12	Extra Function	Press the button for import, export and exit

Add Transaction Panel

When Add Transaction button is pressed, a pop-up window that looks like the diagram below will come out. The user is required to select a transaction from the drop-down list.

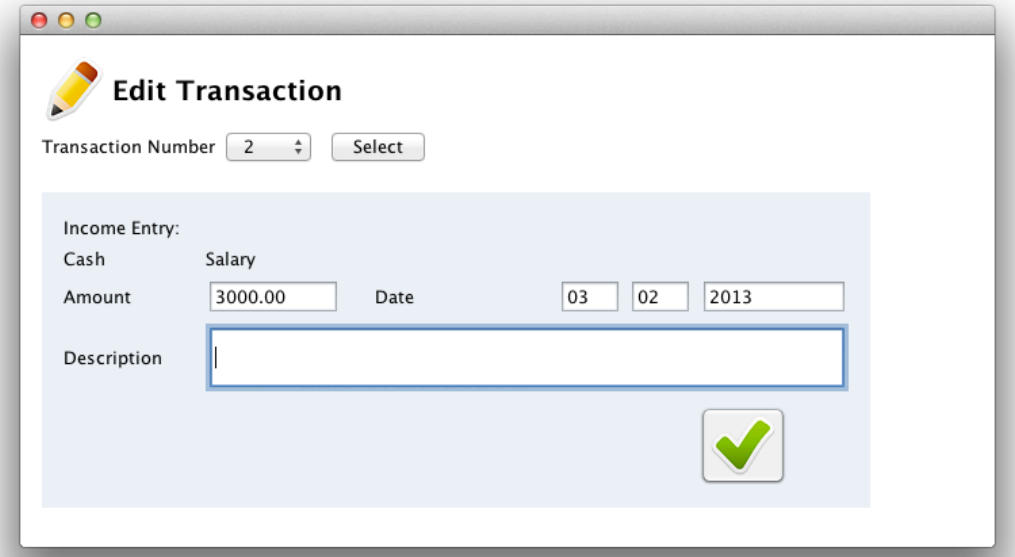


Upon selecting a transaction, a new panel will pop out in the current window, with the colour rendered according to the transaction type for visual feedback. Over here, the user is to fill in the transaction information before pressing the Add Entry button to confirm. If the user has a new Asset Type or Income Category that is not in the drop-down list, he/she is able to add new type/category as long as it does not have the same name as the existing ones. A unique transaction ID is given to every successful transaction.



Edit Transaction Panel

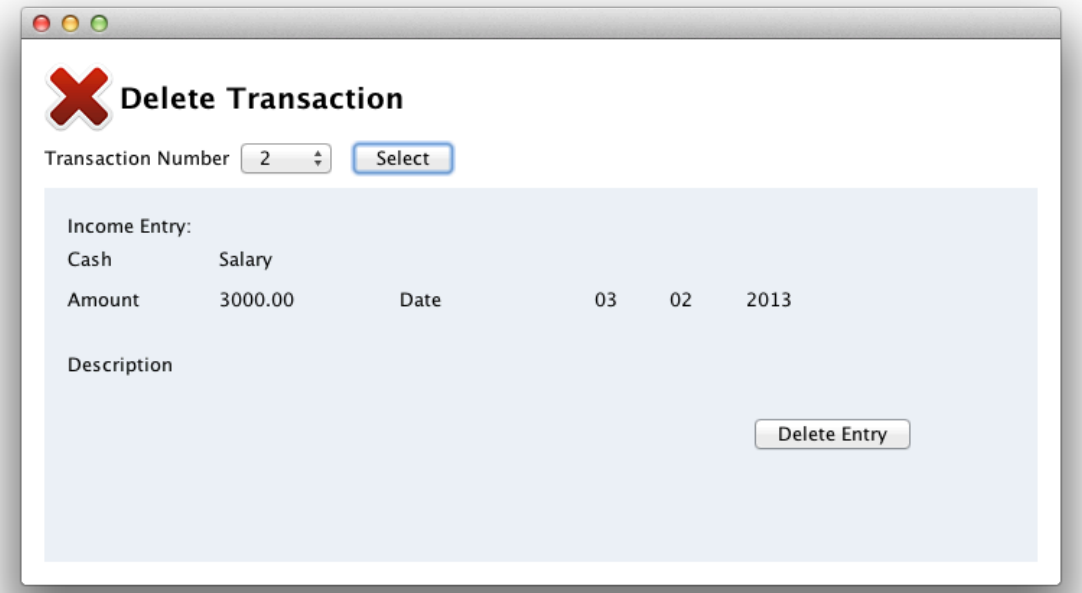
Edit transaction function is for users to change the transaction information of a particular transaction based on its ID. The ID of the transaction can be easily found from the Scrollable Transaction List. When Edit Transaction Button is pressed, Edit Transaction Panel pops out. As in the Add Transaction Panel, the colour of the panel matches the transaction type to facilitate visual feedback.



User can just edit the information that he / she wants to change before pressing the Edit Entry button to confirm.

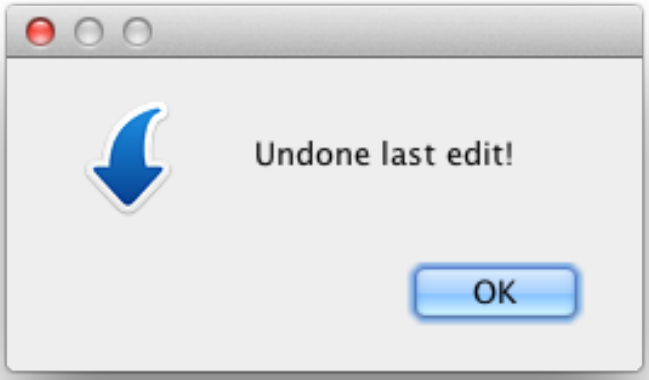
Delete Transaction Panel

If a transaction is not long valid, a user can choose to delete it by pressing the Delete Transaction button . The Delete Transaction panel looks something like this. Press Delete Entry button to confirm.



Undo Panel

When a user wants to undo an operation he just did, he can do so by pressing the Undo Button . A pop up will confirm that the last transaction has been undone, and depending whether the user added, edited, or deleted a transaction previously, the pop up message changes accordingly. If there are no new operations performed since the start of the new session, the pop up will remind the user that there are no operations left to undo.

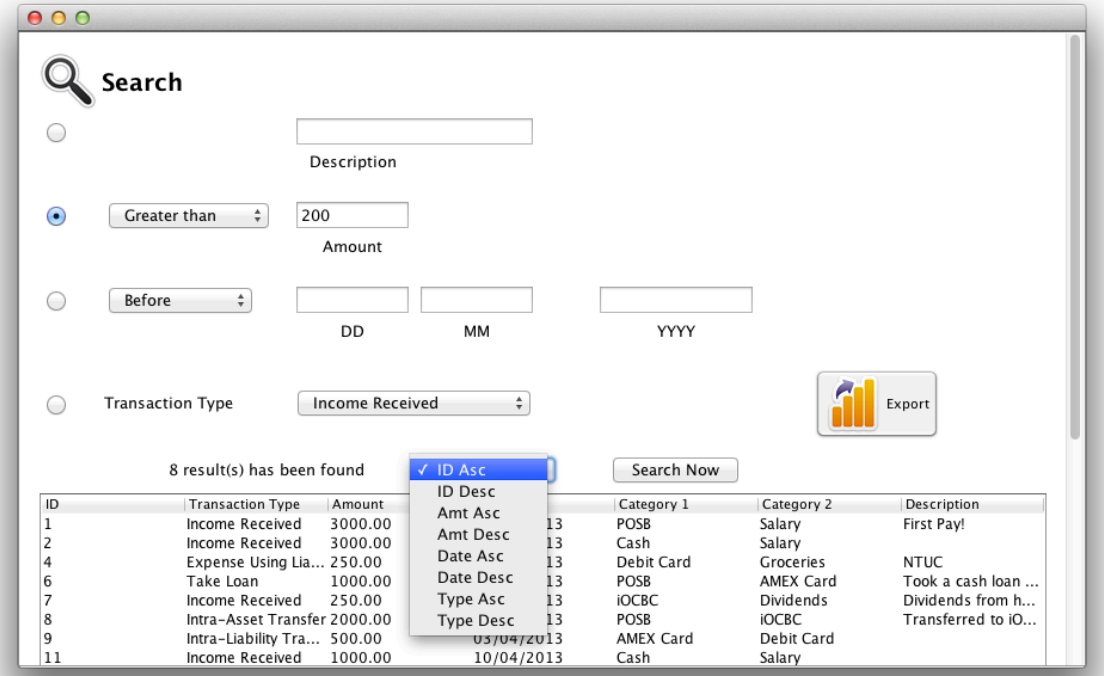


Search Panel and Export Panel

If the user wants to search for a particular transaction / transactions, he can do so by clicking the search button.
There are 4 search criteria provided; searching by

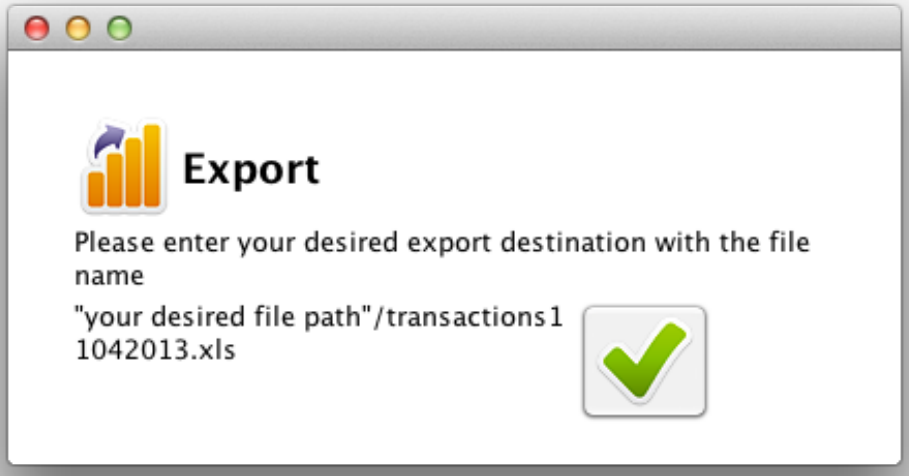
- Description
- Amount
- Date
- Transaction type

After searching, users can sort the search results according to their preferences.



There is also an export function for users to extract transactions to an excel file. Users may do so by clicking the Export button

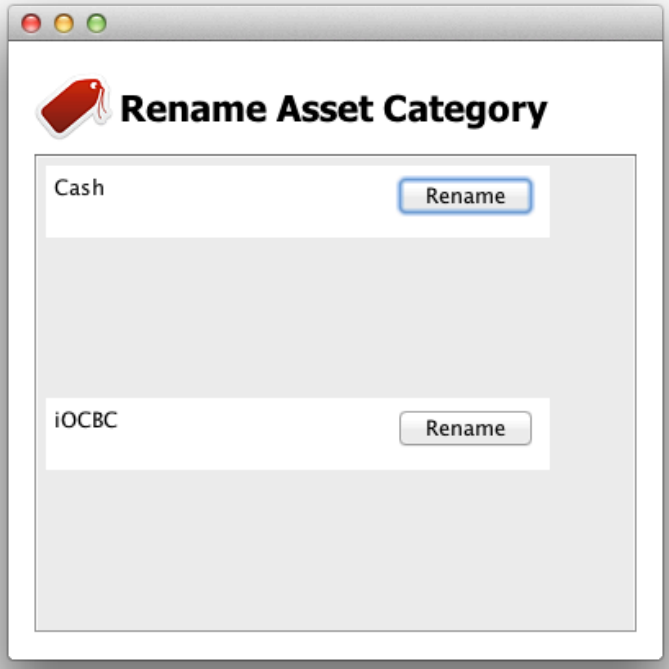
. An export window will pop up, users must key in his/her desired file path before hitting the confirm button. In the path typed, users need to include an absolute path starting from their home directory, and end with “a_file_name”.xls. Any other file extensions will give an error.



Rename Category Panel

A user is also able to change a category name by clicking the Rename Categories Button,

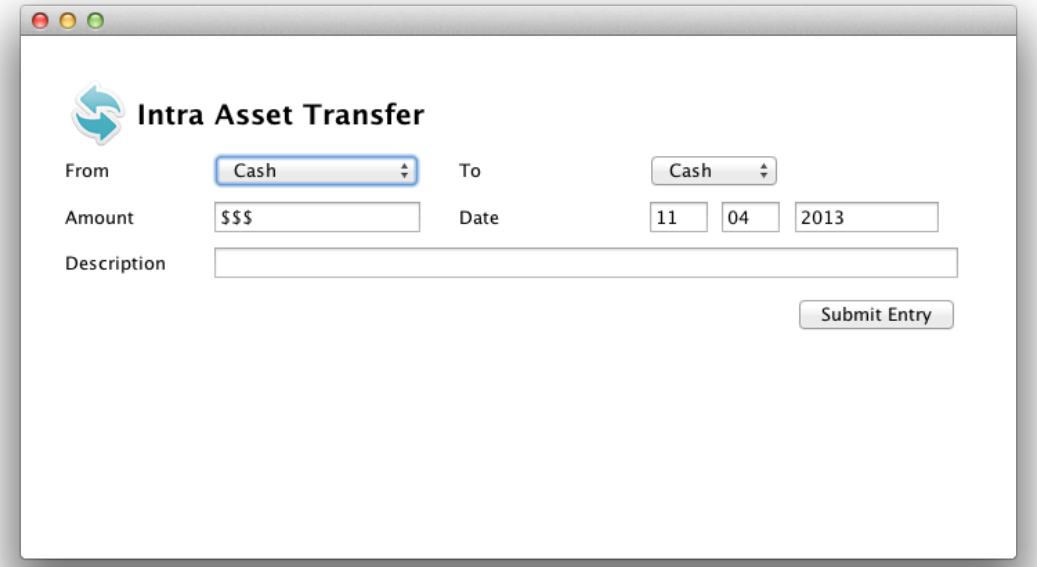
Which is located below the Asset / Liability / Income / Expense Charts.



Inter-asset/Inter-liability Transfer Panel

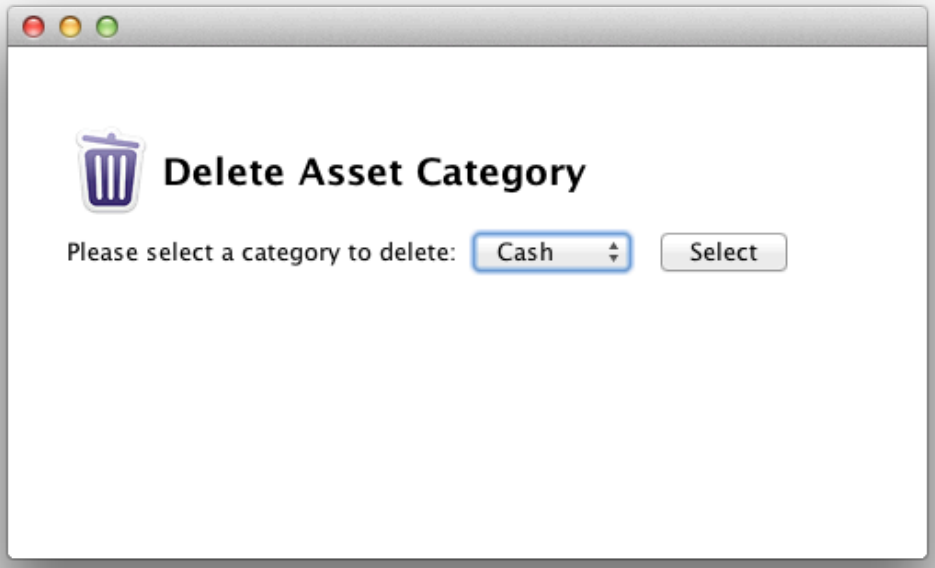
Sometimes, a user wants to transfer money from one category to another category within asset or liability. Examples for such scenarios can be a transfer of money from one back account to another bank account (intra-asset transfer) or the use of a credit card to repay a loan (intra-liability transfer). To do so, the user may click the Transfer button,

Which is located below the Asset/Liability charts, to perform this operation.



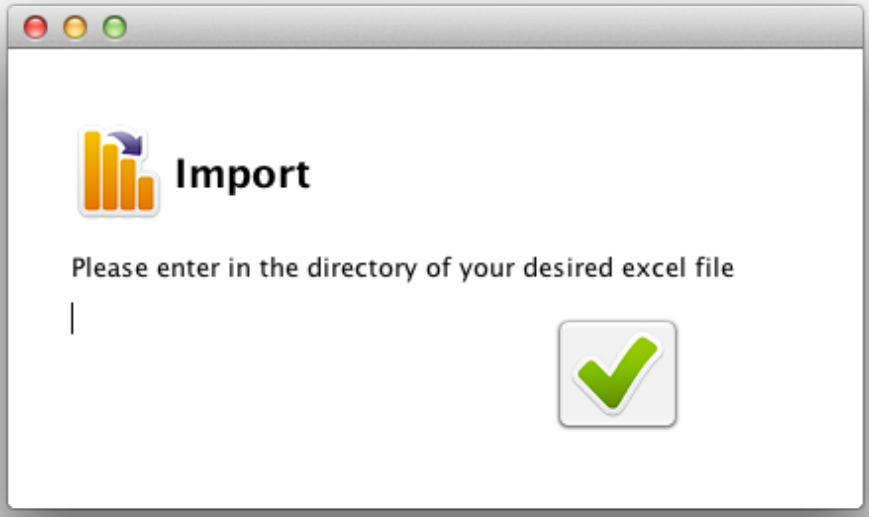
Delete Category Panel

When a user decides to permanently delete an asset/liability category, he/she can do so by clicking the Delete Category Button. However, the category that the user wants to delete must have zero amount in it. If not, the system will give an error message, prompting the user to transfer the balance out of the category, which reflects an actual transaction in real life.



Import Panel

When a user wants to import some data from external excel files, he/she can do this by clicking the import button in the extra function folder File located at the top right corner of the main window. The import panel looks like this:



Hot-keys

To help power-users who prefer to use the keyboard to access commands, we have implemented the following key accelerators to enhance their user experience. Do note that the analogue of the [CTRL] key for Windows users would be the [Command] key for Mac users.

Hot-keys	**Function**
[CTRL]+I	Import
[CTRL]+E	Export
[CTRL]+W	Quit
A	Add Transaction
E	Edit Transaction
D	Delete Transaction
U	Undo Last Operation
S	Search

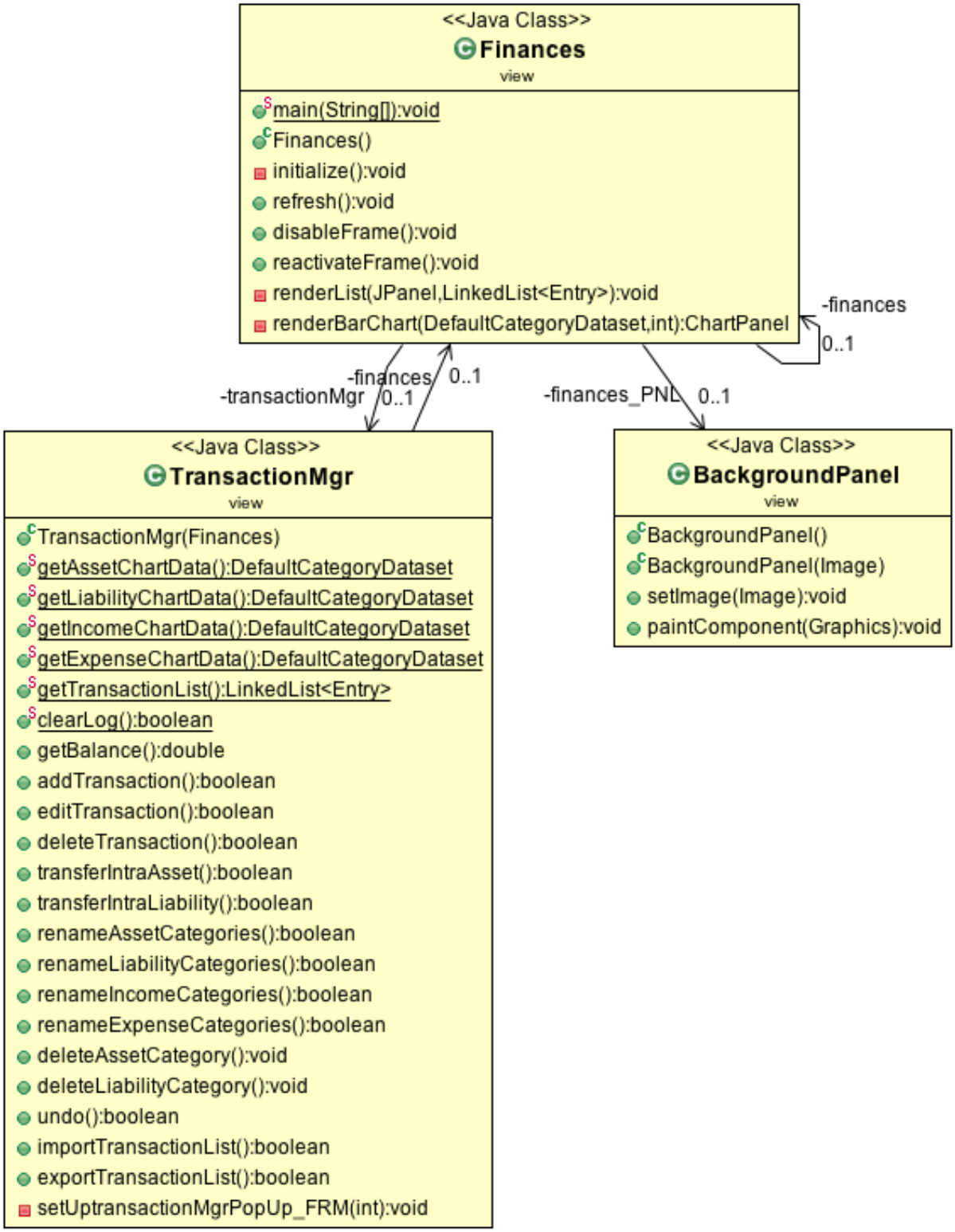
Things to Take Note

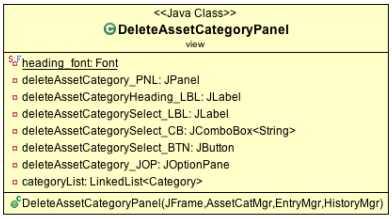
This is a table that guides the user regarding the details of input. As this errors will be detected automatically by the software and the relevant prompt displayed, the user may skip this part of the report with no loss of continuity.

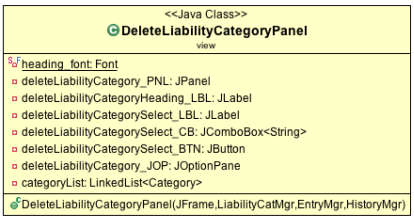
Type	**Things to Take Note**
Date Format	dd/mm/yyyy • Date value range is 0<dd<=31 • Month value range is 0<mm<=12 • Year value range is 1900<=yyyy<= current year • Date must be today or before today's date • Date has to be a valid date (31 Feb is not a valid date)
Amount Value	• All amount values filled in cannot be negative • The amount value after any transaction operation cannot be negative e.g. cash is $1000, a transfer of $1500 from cash to OCBC bank account is an invalid transaction *While categories cannot have a negative balance, the user's overall balance can be negative. A pop up will alert the user if it is so.*
Category Name, Description	• No Pipe sign " |" is allowed

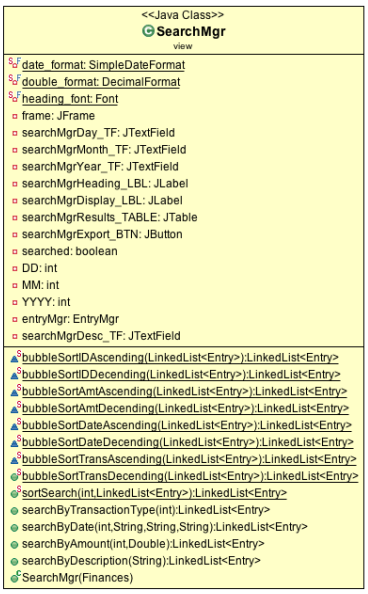
Class Diagram

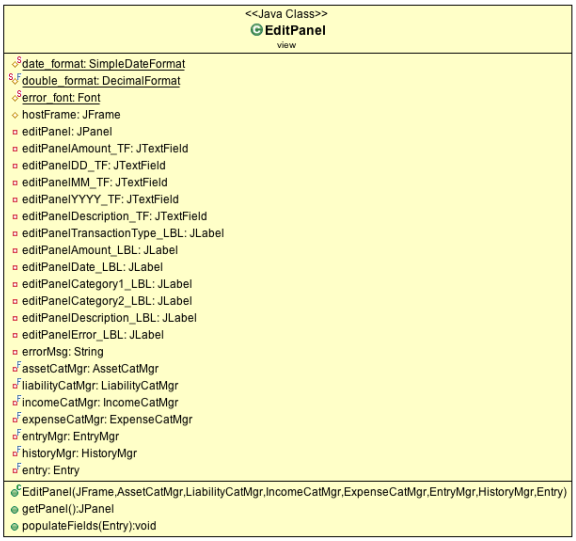
The following class diagrams are the detailed diagrams from the view package

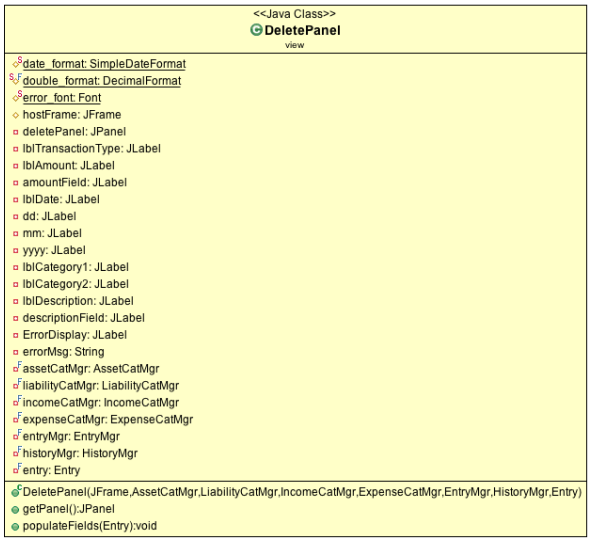


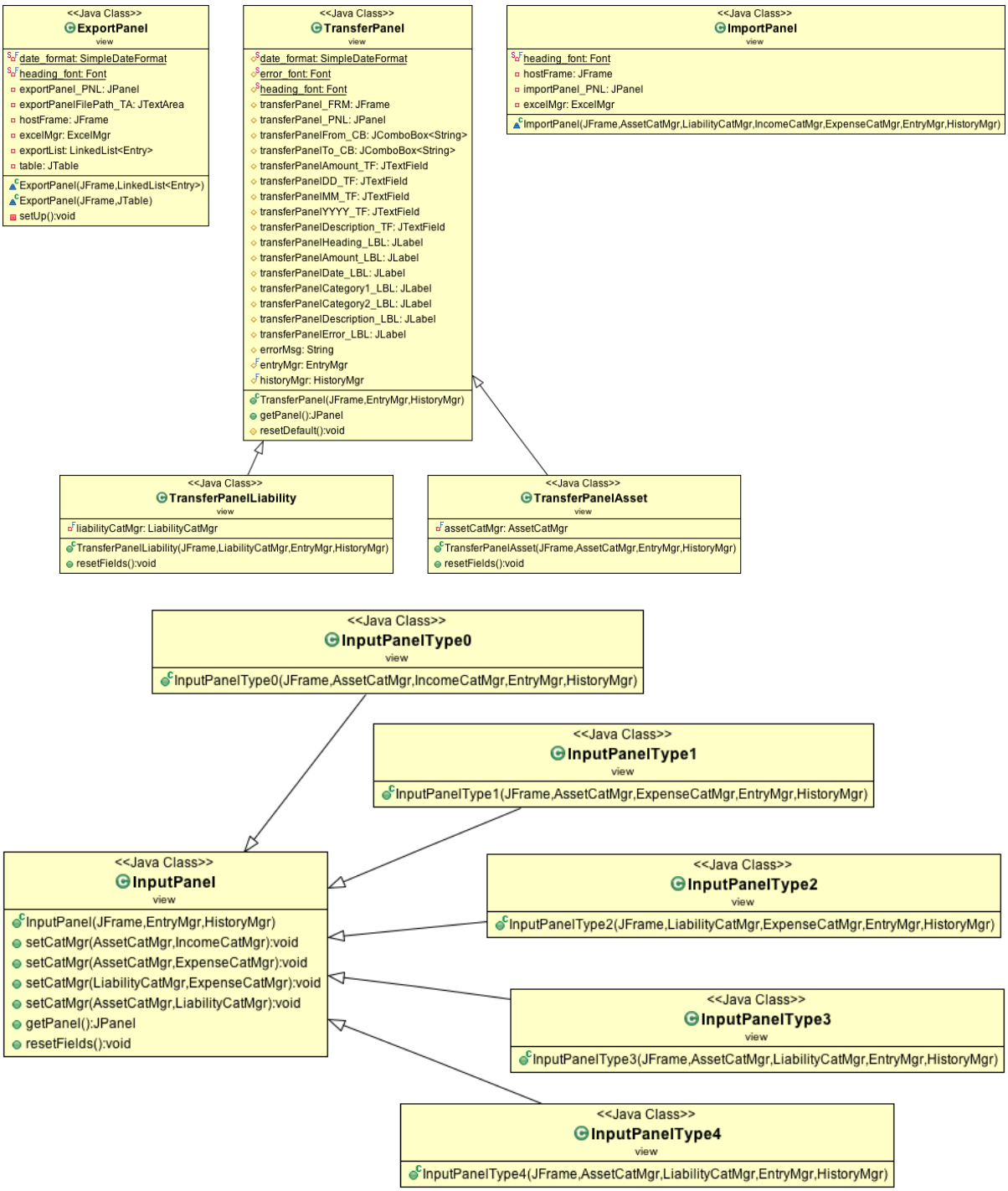

<<Java Class>>
DeleteAssetCategoryPanel
view
heading_font: Font
deleteAssetCategory_PNL: JPanel
deleteAssetCategoryHeading_LBL: JLabel
deleteAssetCategorySelect_LBL: JLabel
deleteAssetCategorySelect_CB: JComboBox<String>
deleteAssetCategorySelect_BTN: JButton
deleteAssetCategory_JOP: JOptionPane
categoryList: LinkedList<Category>
DeleteAssetCategoryPanel(JFrame,AssetCatMgr,EntryMgr,HistoryMgr)


<<Java Class>>
DeleteLiabilityCategoryPanel
view
heading_font: Font
deleteLiabilityCategory_PNL: JPanel
deleteLiabilityCategoryHeading_LBL: JLabel
deleteLiabilityCategorySelect_LBL: JLabel
deleteLiabilityCategorySelect_CB: JComboBox<String>
deleteLiabilityCategorySelect_BTN: JButton
deleteLiabilityCategory_JOP: JOptionPane
categoryList: LinkedList<Category>
DeleteLiabilityCategoryPanel(JFrame,LiabilityCatMgr,EntryMgr,HistoryMgr)


<<Java Class>>
SearchMgr
view
date_format: SimpleDateFormat
double_format: DecimalFormat
heading_font: Font
frame: JFrame
searchMgrDay_TF: JTextField
searchMgrMonth_TF: JTextField
searchMgrYear_TF: JTextField
searchMgrHeading_LBL: JLabel
searchMgrDisplay_LBL: JLabel
searchMgrResults_TABLE: JTable
searchMgrExport_BTN: JButton
searched: boolean
DD: int
MM: int
YYYY: int
entryMgr: EntryMgr
searchMgrDesc_TF: JTextField
bubbleSortIDAscending(LinkedList<Entry>):LinkedList<Entry>
bubbleSortIDDecending(LinkedList<Entry>):LinkedList<Entry>
bubbleSortAmtAscending(LinkedList<Entry>):LinkedList<Entry>
bubbleSortAmtDecending(LinkedList<Entry>):LinkedList<Entry>
bubbleSortDateAscending(LinkedList<Entry>):LinkedList<Entry>
bubbleSortDateDecending(LinkedList<Entry>):LinkedList<Entry>
bubbleSortTransAscending(LinkedList<Entry>):LinkedList<Entry>
bubbleSortTransDecending(LinkedList<Entry>):LinkedList<Entry>
sortSearch(int,LinkedList<Entry>):LinkedList<Entry>
searchByTransactionType(int):LinkedList<Entry>
searchByDate(int,String,String,String):LinkedList<Entry>
searchByAmount(int,Double):LinkedList<Entry>
searchByDescription(String):LinkedList<Entry>
SearchMgr(Finances)


<<Java Class>>
EditPanel
view
date_format: SimpleDateFormat
double_format: DecimalFormat
error_font: Font
hostFrame: JFrame
editPanel: JPanel
editPanelAmount_TF: JTextField
editPanelDD_TF: JTextField
editPanelMM_TF: JTextField
editPanelYYYY_TF: JTextField
editPanelDescription_TF: JTextField
editPanelTransactionType_LBL: JLabel
editPanelAmount_LBL: JLabel
editPanelDate_LBL: JLabel
editPanelCategory1_LBL: JLabel
editPanelCategory2_LBL: JLabel
editPanelDescription_LBL: JLabel
editPanelError_LBL: JLabel
errorMsg: String
assetCatMgr: AssetCatMgr
liabilityCatMgr: LiabilityCatMgr
incomeCatMgr: IncomeCatMgr
expenseCatMgr: ExpenseCatMgr
entryMgr: EntryMgr
historyMgr: HistoryMgr
entry: Entry
EditPanel(JFrame,AssetCatMgr,LiabilityCatMgr,IncomeCatMgr,ExpenseCatMgr,EntryMgr,HistoryMgr,Entry)
getPanel():JPanel
populateFields(Entry):void


<<Java Class>>
DeletePanel
view
date_format: SimpleDateFormat
double_format: DecimalFormat
error_font: Font
hostFrame: JFrame
deletePanel: JPanel
lblTransactionType: JLabel
lblAmount: JLabel
amountField: JLabel
lblDate: JLabel
dd: JLabel
mm: JLabel
yyyy: JLabel
lblCategory1: JLabel
lblCategory2: JLabel
lblDescription: JLabel
descriptionField: JLabel
ErrorDisplay: JLabel
errorMsg: String
assetCatMgr: AssetCatMgr
liabilityCatMgr: LiabilityCatMgr
incomeCatMgr: IncomeCatMgr
expenseCatMgr: ExpenseCatMgr
entryMgr: EntryMgr
historyMgr: HistoryMgr
entry: Entry
DeletePanel(JFrame,AssetCatMgr,LiabilityCatMgr,IncomeCatMgr,ExpenseCatMgr,EntryMgr,HistoryMgr,Entry)
getPanel():JPanel
populateFields(Entry):void


<<Java Class>>
ExportPanel
view
date_format: SimpleDateFormat
heading_font: Font
exportPanel_PNL: JPanel
exportPanelFilePath_TA: JTextArea
hostFrame: JFrame
excelMgr: ExcelMgr
exportList: LinkedList<Entry>
table: JTable
ExportPanel(JFrame,LinkedList<Entry>)
ExportPanel(JFrame,JTable)
setUp():void
<<Java Class>>
TransferPanel
view
date_format: SimpleDateFormat
error_font: Font
heading_font: Font
transferPanel_FRM: JFrame
transferPanel_PNL: JPanel
transferPanelFrom_CB: JComboBox<String>
transferPanelTo_CB: JComboBox<String>
transferPanelAmount_TF: JTextField
transferPanelDD_TF: JTextField
transferPanelMM_TF: JTextField
transferPanelYYYY_TF: JTextField
transferPanelDescription_TF: JTextField
transferPanelHeading_LBL: JLabel
transferPanelAmount_LBL: JLabel
transferPanelDate_LBL: JLabel
transferPanelCategory1_LBL: JLabel
transferPanelCategory2_LBL: JLabel
transferPanelDescription_LBL: JLabel
transferPanelError_LBL: JLabel
errorMsg: String
entryMgr: EntryMgr
historyMgr: HistoryMgr
TransferPanel(JFrame,EntryMgr,HistoryMgr)
getPanel():JPanel
resetDefault():void
<<Java Class>>
ImportPanel
view
heading_font: Font
hostFrame: JFrame
importPanel_PNL: JPanel
excelMgr: ExcelMgr
ImportPanel(JFrame,AssetCatMgr,LiabilityCatMgr,IncomeCatMgr,ExpenseCatMgr,EntryMgr,HistoryMgr)
<<Java Class>>
TransferPanelLiability
view
liabilityCatMgr: LiabilityCatMgr
TransferPanelLiability(JFrame,LiabilityCatMgr,EntryMgr,HistoryMgr)
resetFields():void
<<Java Class>>
TransferPanelAsset
view
assetCatMgr: AssetCatMgr
TransferPanelAsset(JFrame,AssetCatMgr,EntryMgr,HistoryMgr)
resetFields():void
<<Java Class>>
InputPanelType0
view
InputPanelType0(JFrame,AssetCatMgr,IncomeCatMgr,EntryMgr,HistoryMgr)
<<Java Class>>
InputPanelType1
view
InputPanelType1(JFrame,AssetCatMgr,ExpenseCatMgr,EntryMgr,HistoryMgr)
<<Java Class>>
InputPanel
view
InputPanel(JFrame,EntryMgr,HistoryMgr)
setCatMgr(AssetCatMgr,IncomeCatMgr):void
setCatMgr(AssetCatMgr,ExpenseCatMgr):void
setCatMgr(LiabilityCatMgr,ExpenseCatMgr):void
setCatMgr(AssetCatMgr,LiabilityCatMgr):void
getPanel():JPanel
resetFields():void
<<Java Class>>
InputPanelType2
view
InputPanelType2(JFrame,LiabilityCatMgr,ExpenseCatMgr,EntryMgr,HistoryMgr)
<<Java Class>>
InputPanelType3
view
InputPanelType3(JFrame,AssetCatMgr,LiabilityCatMgr,EntryMgr,HistoryMgr)
<<Java Class>>
InputPanelType4
view
InputPanelType4(JFrame,AssetCatMgr,LiabilityCatMgr,EntryMgr,HistoryMgr)

Algorithms

Storing Data
Data throughout the application are stored in txt files. These files consists of the Entries that the user has transacted, and categories which the user has created. 6 txt files are used:

1. EntryList.txt is used to store all Entries by the user.
2. History.txt stores the CRUD Entries made in the current session of the web application. This file is used in order to undo transactions by keeping track of the latest transactions being made.
3. AssetCategory.txt is to store all previously created Asset Categories.
4. LiabilityCategory.txt is to store all previously created Liability Categories.
5. IncomeCategory.txt is to store all previously created Income Categories.
6. ExpenseCategory.txt is to store all previously created Expense Categories.

In order to create, update, query or delete the stored data, buffered readers and writers are used to manipulate these files.

Storing Data: Creating an Entry
When the user creates a transaction, he/she is required to key in details such as the transaction type, amount, date, description (optional), and the categories in which the transaction occurs in.
When inputs are completed, a buffered writer writes to the designated category txt files, whether a new category has been made, or update an existing category's amount. Also this new transaction will be added into the list of entries (EntryList.txt).
Stored Data in Category Format:
Cat_Name | Cat_Total_Amt
Stored Data in Entry List Format:
ID | Type | Amount | Date | Category_1| Category_2 | Description

Storing Data: Reading Entries
Entries are read from the EntryList.txt file. They are read using a buffered reader, and tokenized using the delimiter of the pipe character, |.

Storing Data: Updating Entries
When there is an update of an Entry, the EntryList.txt is read. Similarly using buffered reader and tokenizer, every Entry which does not match the updated Entry is copied into a temporarily created txt file, and writing the new updated Entry in-place of the old Entry, into the temporary txt file in the same line space in order to ensure the latest updated information. The temporary txt file then replaces the original txt file.

Storing Data: Deleting an Entry
The algorithm used is the same as Updating an Entry, where only every Entry which does not match the Entry to be deleted is copied into a temporarily created txt file. The temporary txt file then replaces the original txt file.

Storing Data: CRUD of Categories
The creating, retrieving (reading), renaming, revalidate of total amount or inter asset transfer (updating), and deleting of categories are very much similar to that of CRUD of Entries.

The format of a Category in any of the category txt files:
Category_Name | Total_Amount
Where Category_Name is the category name of whether an Asset, Liability, Income or Expense category. Total_Amount is the total amount of what each category has, with respect to every transaction made in the list of entries.

Storing Data: CRUD of Categories: Renaming Categories
When there is an renaming of a category, the designated category txt file is read. Similarly using buffered reader and tokenizer, every Category which does not match the updated Category is copied into a temporarily created txt file, and writing the new update Category in-place of the old Category, into the temporary txt file in the same line space in order to ensure the latest updated information. The temporary txt file then replaces the original txt file.

Storing Data: CRUD of Categories: Inter-Category Amount Transfer
When there is inter-category amount transfer between categories, the designated category txt file is read. Similarly using buffered reader and tokenizer, every Category which does not match the updated Category is copied into a temporarily created txt file. For example if there is a transfer of asset 'A' of $500 to asset 'B':

Old txt		Temporary txt
A | 5000		A | 4500
B | 100	→	B | 600

The 2 categories in which amount has been changed will be written as the following above. The temporary txt file then replaces the original txt file.

Searching Data
A linked list of all Entries are returned using an Entry Manager's function. Depending on the type of search, these Entries are manipulated differently in order to get a list of result.
Again, each Entry has the format:
ID | Type | Amount | Date | Category_1| Category_2 | Description

Searching Data: Search by Transaction Type
The transaction type of an Entry is in the 2nd segment of each segregated delimiter. A string tokenizer with delimiter of the pipe character, |, is used to skip the first attribute and read the transaction type attribute. If the transaction type matches the searched transaction type, the entry is added into a linked list of Entry results. These results are then shown in the Search Manager's result table.

Searching Data: Search by Amount
The Amount of an Entry is in the 3rd segment of each segregated delimiter. A string tokenizer with delimiter of the pipe character, |, is used to skip the first and second attribute and read the Amount attribute. If the Date matches the searched Amount and parameters, the entry is added into a linked list of Entry results. These results are then shown in the Search Manager's result table.

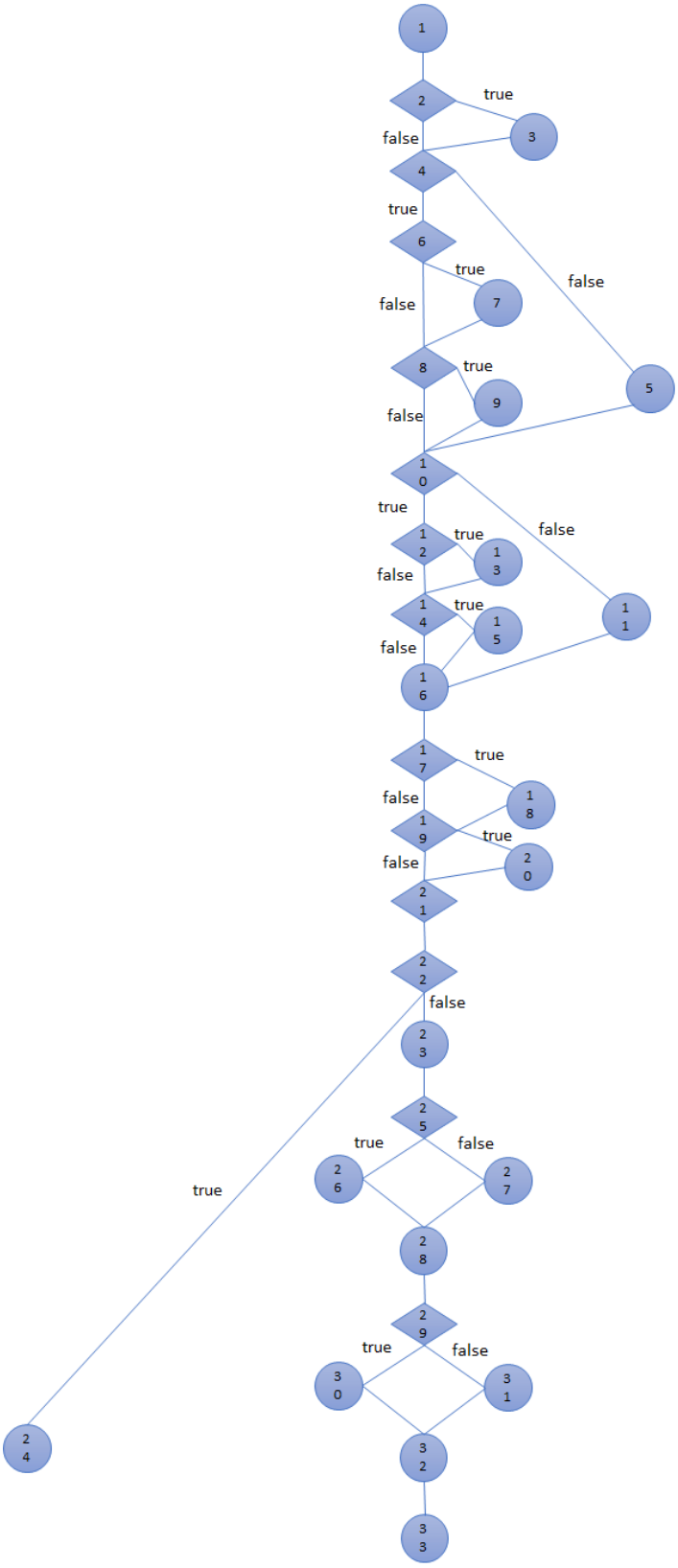
Searching Data: Search by Date
The Date of an Entry is in the 4th segment of each segregated delimiter. A string tokenizer with delimiter of the pipe character, |, is used to skip the first to third attribute and read the Date attribute. If the Date matches the searched Date and parameters, the entry is added into a linked list of Entry results. These results are then shown in the Search Manager's result table.

Searching Data: Search by Keywords
For each Entry in the linked list of all Entries, the Entry data of all attributes are concatenated into an upper cased String of text without the pipe character, |. For each concatenated String that contains any of the input keywords, that Entry is added into a linked list of Entry results. These results are then shown in the Search Manager's result table

Searching Data: Sorting the Results
Depending on the search sorting parameters, whether sort the results ascending or descending by the ID/amount/date/transaction type, the linked list of result Entries are then sorted accordingly using Bubble Sort.

Refreshing Charts
With the use of bar charts, our application provides instant statistical feedback on the latest transactions and entries a used has made, in the application's main window, Finances.
During the creation of a transaction, users require the transaction manager GUI in order to process their entry. Once it is completed, the transaction manager will close.
An ***action listener*** on the Finances Window, continuously listens for a "window close" action, calling the refresh function of the charts immediately. All categories with the total amount of each category are read from the designated files, and the charts are refreshed with the latest information.

Cyclomatic Complexity Diagram



Unit Testing

The unit test scripts below are similar in all the sub Category Managers, AssetCatMgr, LiabilityCatMgr, IncomeCatMgr and ExpenseCatMgr as they provide the same functionalities. The example below is using the AssetCatMgr test. Tests are used based on the set of data:
House|7000.0
Cash|3000.0
DBS|3000.0
MayBank|1000.0

```
@Test
public void testGetSubtotal() {
        AssetCatMgr acm = new AssetCatMgr();
        try{
                //Test Passed as there is indeed a subtotal of $14,000
                assertTrue(acm.getSubtotal()==14000.0);
        } catch(Exception e){
                System.out.println(e);
                assertTrue(false);
        }
}


@Test
public void testGetAmount() {
        AssetCatMgr acm = new AssetCatMgr();
        try{
                //Test Passed as there is $3,000 in Cash
                assertTrue(acm.getAmount("Cash")==3000.0);
        } catch(Exception e){
                System.out.println(e);
                assertTrue(false);
        }
        assertTrue(true);
}

@Test
public void testAddCategory() {
        AssetCatMgr acm = new AssetCatMgr();
        boolean cond1 = false;
        boolean cond2 = false;
        try{
                //Testing of adding a Category "Test" of $1,000
                acm.addCategory("Test", 1000.0);

//Condition 1 is now true as the subtotal is now $15,000 ($14,000 + $1000)
```

```
                if(acm.getSubtotal()==15000.0){
                        cond1=true;
                }

//Condition 2 is now true as there is $1,000 in "Test"
                if(acm.getAmount("Test")==1000.0){
                        cond2=true;
                }

                //Test Passed condition 1 and 2 are met
                assertTrue(cond1 && cond2);

        } catch(Exception e){
                System.out.println(e);
                assertTrue(false);
        }
    }

    @Test
    public void testAddAmountToCategory() {
        AssetCatMgr acm = new AssetCatMgr();
        try{

                boolean cond1 = false;
                boolean cond2 = false;

//Testing of adding an Amount to "Test" of $100
                acm.addAmountToCategory("Test", 100.0);

//Condition 1 is now true as the subtotal is now $15,100 ($15,000 + $100)
                if(acm.getSubtotal()==15100.0){
                        cond1=true;
                }

//Condition 2 is now true as there is $1,100 in "Test"
                if(acm.getAmount("Test")==1100.0){
                        cond2=true;
                }

                //Test Passed condition 1 and 2 are met
                assertTrue(cond1 && cond2);

        } catch(Exception e){
                System.out.println(e);
                assertTrue(false);
        }
        assertTrue(true);
    }
```

```
        @Test
        public void testEditCategoryName() {
                AssetCatMgr acm = new AssetCatMgr();
                try{
                        boolean cond1 = false;
                        boolean cond2 = false;

//Test of updating a Category "Test" to "Test1"
acm.editCategoryName("Test", "Test1");

//Condition 1 is now true as the subtotal is carried forward
                        if(acm.getSubtotal()==15100.0){
                                cond1=true;
                        }

//Condition 2 is now true as there is $1,100 in such a "Test1" category from
renaming
                        if(acm.getAmount("Test1")==1100.0){
                                cond2=true;
                        }

                        //Test Passed condition 1 and 2 are met
                        assertTrue(cond1 && cond2);

                } catch(Exception e){
                        System.out.println(e);
                        assertTrue(false);
                }
                assertTrue(true);
        }

        @Test
        public void testDeleteCategory() {
                AssetCatMgr acm = new AssetCatMgr();
                try{
//Test of deleting a the Category "Test1"
                        acm.deleteCategory("Test1");

//Test Passed as there is only left $14,000 ($15,100 – $1,100)
                        assertTrue(acm.getSubtotal()==14000.0);
} catch(Exception e){
                        System.out.println(e);
                        assertTrue(false);
                }
                assertTrue(true);
        }

}
```

Bug List

The following list contains the significant 20% of our bugs which we felt were worth documenting and raising each other's awareness as we progressed in our development. They are far from exhaustive, and have excluded the minor errors which we felt were trivial and would only clutter up the report.

ID	Classes in which bug was found	Description of problem	Source of problem	Fix
1	Entry Log CatMgr EntryMgr HistoryMgr	Date displayed was different from input date Unable to sort list of entries correctly	date_format was initialized to "dd/mm/yyyy", where mm represented minutes instead of months	date_format changed to "dd/MM/yyyy", where MM represents the month
2	EntryMgr	Current_id overlapped with id of some new entries	Private method init_id() It initialized current_id to the last entry's id+1. Entries are not necessarily ordered in increasing id in EntryList.txt, wrong algorithm used.	Changed the algorithm in init_id() to search for the largest id in EntryList.txt and set current_id as max id + 1
3	EditPanel DeletePanel	The combo box which contained the list of edit-able/delete-able transaction IDs did not display the right IDs	The algorithm for selecting edit-able/delete-able transaction IDs filtered out the irrelevant IDs, but the parsing of input in the combo box did not parse for the ID, but parsed for the index number of the selected option instead.	Changed the parsing method to parse for the ID (which is the 'item' in the combo box) instead of the selected index.
4	Finances	Transaction list gets display messed up when the user clicks and drags within the transaction list scroll pane.	Unidentified. Suspected problem with Swing components. If more time is available, we intend to change the design to use other components (JPanel) and see if the problem persists	To be fixed.

Glossary

The following table contains a list of terms and our specific interpretation in the context of our software

Term	Meaning
Asset	An asset has cash value, and can be converted into cash if necessary. Examples include a bank account, cash deposits, stock holdings, and even physical possessions of significant value such as property, land, or vehicles.
Liability	A liability is a form of debt, which has to be paid off. Examples include credit card debt, cash loans, and house loans.
Income	An income is a source that increases one asset. Examples include salary, dividends, gifts, rebates and handouts.
Expense	An expense is an accrual of expenditure. Here we use the GAAP (general accepted accounting practices) standard, which recognizes expenses as they are accrued, even before they are paid in cash. This allows for the accounting of debt , for example when the user spends beyond his or her present ability to pay. Examples include phone bills, money spent on meals, transport, rent, utilities and taxes. In more complicated scenarios, intangible losses such as depreciation of property or vehicles, or loss through stock trading can be represented as an expense.
Transaction	An exchange between the user and a third party involving some form of value. Examples include receiving a salary, spending cash on meals, paying bills with a credit card, or signing up for a loan. In more general cases (also for our case), the exchange can be within the user's own assets or liabilities. Examples include topping up one's ez-link card with cash, drawing cash from a bank's ATM, or paying a credit card bill with another credit card.
GUI	Graphical User Interface. It is what the user sees when he/she uses the program, and it is the means by which he/she gives interacts with the program.