

EzXpns Project Report v0.2

Group W14-2j

Andrew Koh Zheng Kang A0097973H

Lee Shu Zhen A0085413J

Yan Ting Zhe A0087091B

Yao Yujian A0099621X

Table of Contents

Project Scope	1
Problem	1
Product Description	1
Vision	1
EzXpns User Guide	2
Main Interface	3
Add a Record	4
Edit/Remove a Record	5
Search a Record	6
Generate Report	7
Manage Category & Target	8
Developer's Guide v0.2	9
1. Introduction	10
2. Development Infrastructure	10
3. Use Cases	10
Use Case Diagram	10
4. Architecture	11
4.1 Storage and Serialisation	13
4.2 Data Management	13
4.3 Summary System	14

4.4 Report System	14
4.5 Target System (with alerts)	14
4.6 Needs, Wants, and Savings System	15
4.7 Graphical User Interface (GUI)	16
4.8 The Undo Framework	18
5. Testing Framework	20
6. Known Issues	20
Project Process & Administration	21
Responsibilities	21
Timeline	21
Appendix	22
Appendix A: Use Case Descriptions	23
Use Case Diagram	23
Use Case: 01 - Add New Expense Record	24
Use Case: 02 - Add New Income Record	24
Use Case: 03 - Add a Target	24
Use Case: 04 - Add an Expense Category	25
Use Case: 05 - Add an Income Category	25
Use Case: 06 - View Report	25
Use Case: 07 - Remove a Category	26
Use Case: 08- Search for Records	26
Use Case 09 - View Summary	27
Use Case 10 - Undo an Operation	27
Appendix B: Sequence Diagram	28
EzXpns Start Sequence	28

Add New Income Record	29
Modify Category	29
Generate Report	30
EzXpns Exit Sequence	30
Appendix C: Class Diagrams	31
DataManager Class Diagram	31
Atomic Data Unit Class Diagram	32
RecordManager Class Diagram	33
Summary System Class Diagram	34
Report System Class Diagram	35
Target System Class Diagram	36
Needs, Wants, & Savings Class Diagram	37
GUI Class Diagram	38
Appendix D: API	39
1) Package ezxpns	39
2) Package ezxpns.data	40
3) Package ezxpns.data.records	53
4) Package ezxpns.GUI	59
5) Package ezxpns.util	62

Project Scope

Problem

There is a lack of expense management system catered to youths in the market.

This is due to two misconceptions.

1. Budgeting are for adults
2. Existing expense management system are suffice for young adults

Budgeting is a lifelong skill. The transition to tertiary education brings about many financial burden. Furthermore, young adults are inexperienced in financial management - they might never be if they don't start managing their expenses. Hence there is a need for a simple and educational expense manager in the market and we want to fill that demand.

Product Description

EzXpns is a standalone desktop expense manager with a simple interface. It is catered for **youths (16-25)** with poor financial planning skills and little motivation to start planning.

We believe less is more. As this product is catered for the general youth population, we have simplified the user interface and focused on features that are most relevant to them. This program has the basic features that allows user to record expense items and do some simple management of their budget and expenses. The user is able to see his/her finance status in one glance.

Many young adults or teenagers are still inexperienced in financial planning. In order to encourage daily use and inculcate good financial planning skills among the youths today, we have introduced gamification to the program and included educational elements to it. It allows them to learn the skills on the way as the software guide them along.

Vision

Our vision is to incorporate 3 values into our product.

Simplicity: Our software will be simple and easy to understand. The best interface is the one you don't notice, where the very first thing you try does exactly what you want it to do, and any accidental actions that prove destructive are easily reversed.

Intelligent: Our software will be intelligent enough to pick up trends on behalf of the user, even before the user himself notices it.

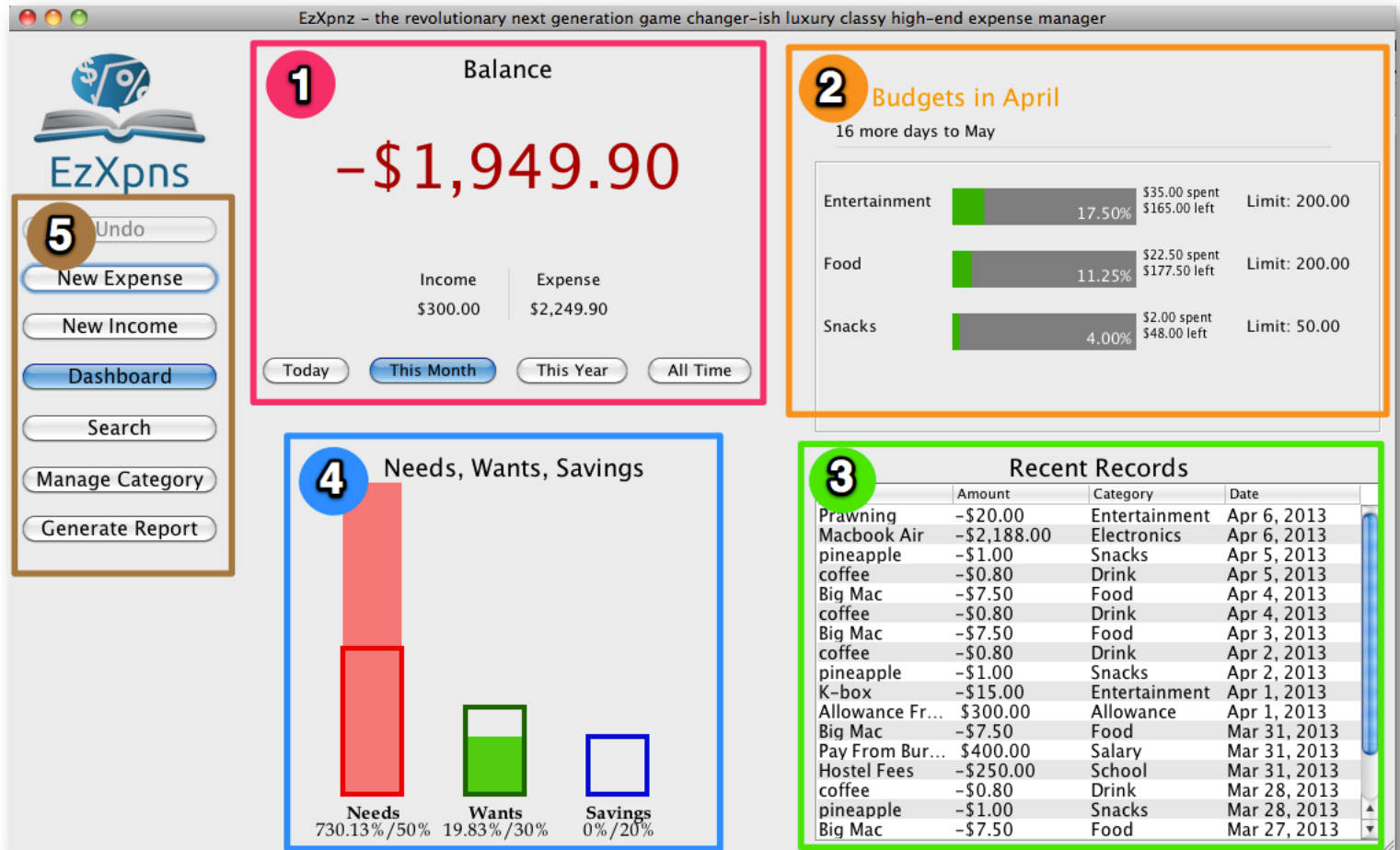
Educational: Our software will teach users on the basics of personal budgeting by giving advice as well as best practice.



EzXpns User Guide

Main Interface

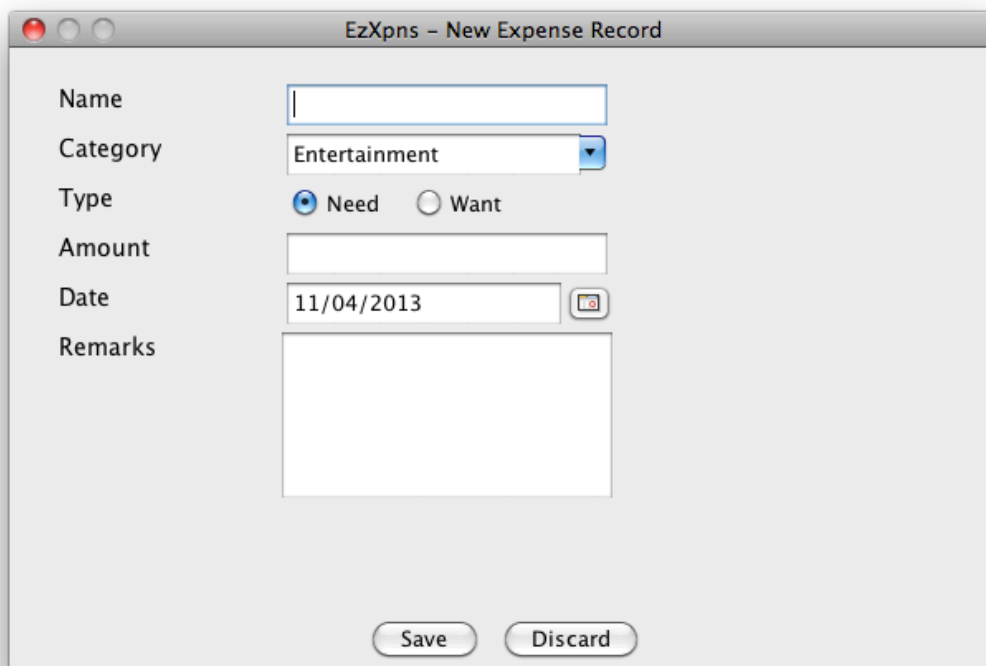
The Dashboard



- (1) Balance
- (2) Targets
- (3) Recent Records
- (4) Needs, Wants, Savings
- (5) Sidebar

Add a Record

1. Click on 'New Expense' or 'New Income' on the sidebar



The screenshot shows a macOS-style dialog box titled "EzXpns - New Expense Record". It contains the following fields and controls:

- Name:** A text input field.
- Category:** A dropdown menu with "Entertainment" selected.
- Type:** Two radio buttons, "Need" (selected) and "Want".
- Amount:** A text input field.
- Date:** A text input field containing "11/04/2013" and a small calendar icon to its right.
- Remarks:** A large text area.
- Buttons:** "Save" and "Discard" buttons at the bottom.

2. Fill up the form. Remarks are optional.

Shortcut: Create a new Category

If you want to create a new category, simply type the name of your new category in the combo box. EzXpns will automatically create a new category for you.

Shortcut: Calculator

Amount =\$12.50

You can enter simple equation in the 'Amount' field. EzXpns will calculate the amount for you.

3. Click 'Save'

Edit/Remove a Record

Recent Records			
Name	Amount	Category	Date
Prawning	-\$20.00	Entertainment	Apr 6, 2013
Macbook Air	-\$2,188.00	Electronics	Apr 6, 2013
pineapple	-\$1.00	Food	Apr 5, 2013
coffee	-\$0.80	Drink	Apr 5, 2013
Big Mac	-\$7.50	Food	Apr 4, 2013
coffee	-\$0.80	Drink	Apr 4, 2013
Big Mac	-\$7.50	Food	Apr 3, 2013
coffee	-\$0.80	Drink	Apr 2, 2013
pineapple	-\$1.00	Snacks	Apr 2, 2013
K-box	-\$15.00	Entertainment	Apr 1, 2013
Allowance From...	\$300.00	Allowance	Apr 1, 2013
Big Mac	-\$7.50	Food	Mar 31, 2013
Pay From Burger...	\$400.00	Salary	Mar 31, 2013
Hostel Fees	-\$250.00	School	Mar 31, 2013
coffee	-\$0.80	Drink	Mar 28, 2013
pineapple	-\$1.00	Snacks	Mar 28, 2013
Big Mac	-\$7.50	Food	Mar 27, 2013
pineapple	-\$1.00	Snacks	Mar 27, 2013

1. Go to the 'Dashboard'.
2. Select a Record from 'Recent Records'.
3. Right click the Record, and select either 'edit' or 'remove'.
4. If 'edit', make your changes in the new window and click 'Save'. If 'remove', an alert dialog will popup. Click 'OK' to remove.

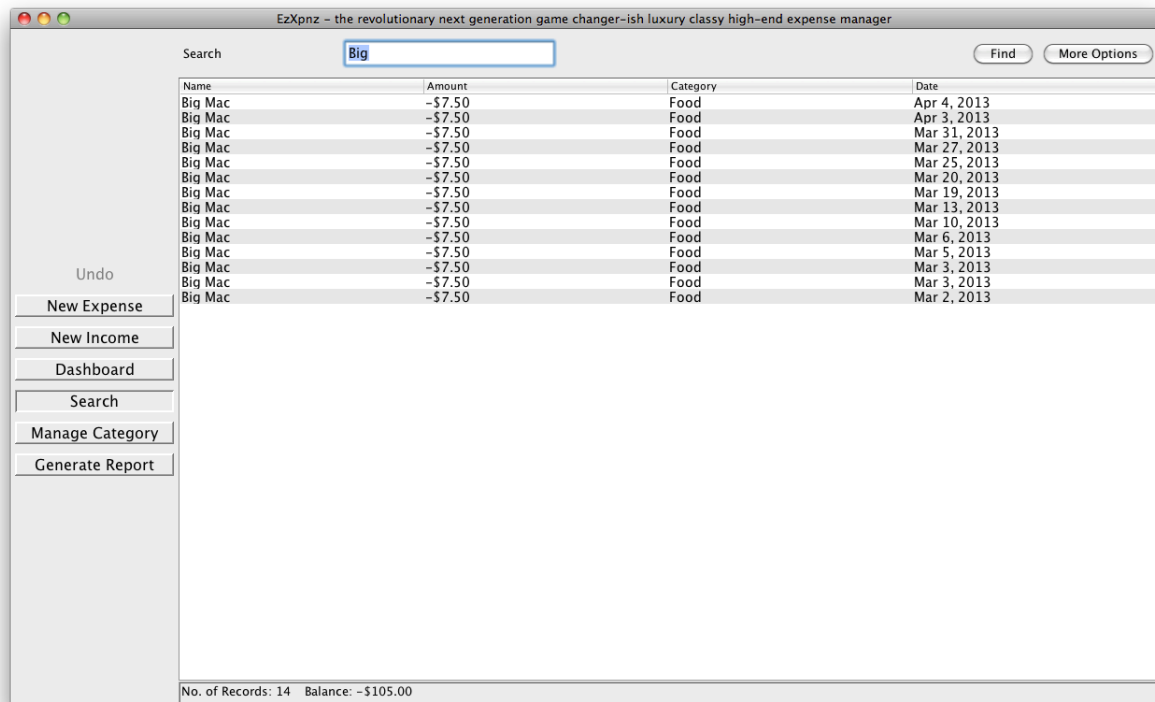
Edit/Remove older records

Recent Records will not display older records. To edit/remove them, you'll have to search and edit/remove the record. Refer to 'Search a Record' on page 6.



Take Note !

Search a Record



1. Click 'Search' on the sidebar.
2. Type your search query in the search box.
3. Search results will be displayed in real time as you type. You can also hit 'Enter' on your keyboard to display the search result.
4. You can also press "More Options" to further refine your search.

Edit/Remove a Record

Just like in 'Recent Records', you can edit or remove a record by right clicking on the selected Record.



Shortcut

Generate Report

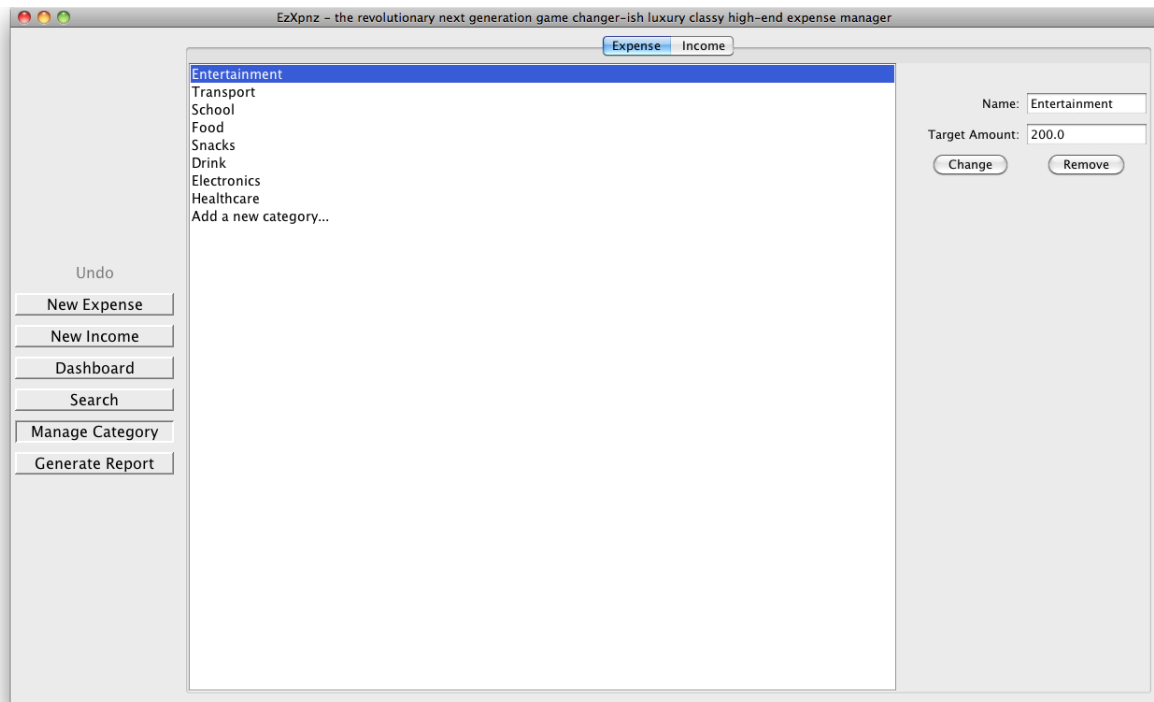
The screenshot shows a web application window titled "EzXpns - Report". The main heading is "Report" with a subtitle "from test start to test end". In the top right corner, there is a button labeled "Generate a new Report". Below this, a teal-colored section titled "Generate a Report" contains the following elements:

- Two input fields for "Start Date" and "End Date", each with a calendar icon to its right.
- Three shortcut buttons: "Last Month", "This Month", and "This Year".
- A "Generate" button on the right side of the teal section.
- A text hint below the date fields: "You can manually type date in this format: dd/mm/yyyy".

Below the teal section, there is a large white area. On the right side of this area, there are three stacked colored boxes: a light blue box labeled "Balance", a light green box labeled "Income", and a light red box labeled "Expense".

1. Click 'Generate Report' on the sidebar.
2. Fill in the 'Start Date' and 'End Date' manually, or choose from 'Last Month', 'This Month' or 'This Year' shortcut buttons.
3. Click 'Generate'.
4. To generate a new report in a different time range, click 'Generate a new Report'.

Manage Category & Target



1. Click 'Manage Category' on the sidebar.
2. Select 'Expense' or 'Income' category from the tab buttons.
3. To **create a new category**, select 'Add a new category...' at the bottom of the list.
4. Fill in the name of your new category, and click 'Add'.
5. To **remove a category**, select the category and click 'Remove'.
6. To **edit a category**, select the category, make the necessary changes, and click 'Change'.
7. To **add a new Target**, select a category, key in the 'Target Amount', and click 'Change'.



Take Note !

Remove a Target

To remove a Target, change the 'Target Amount' to zero.

Developer's Guide v0.2

1. Introduction

EzXpns is a simple finance management utility for young adults.

The purpose of this developer guide is to provide an overview of *EzXpns* with regards to information pertaining to its design and implementation. It will help you to understand the architecture of the program so that you be involved in the coding process.

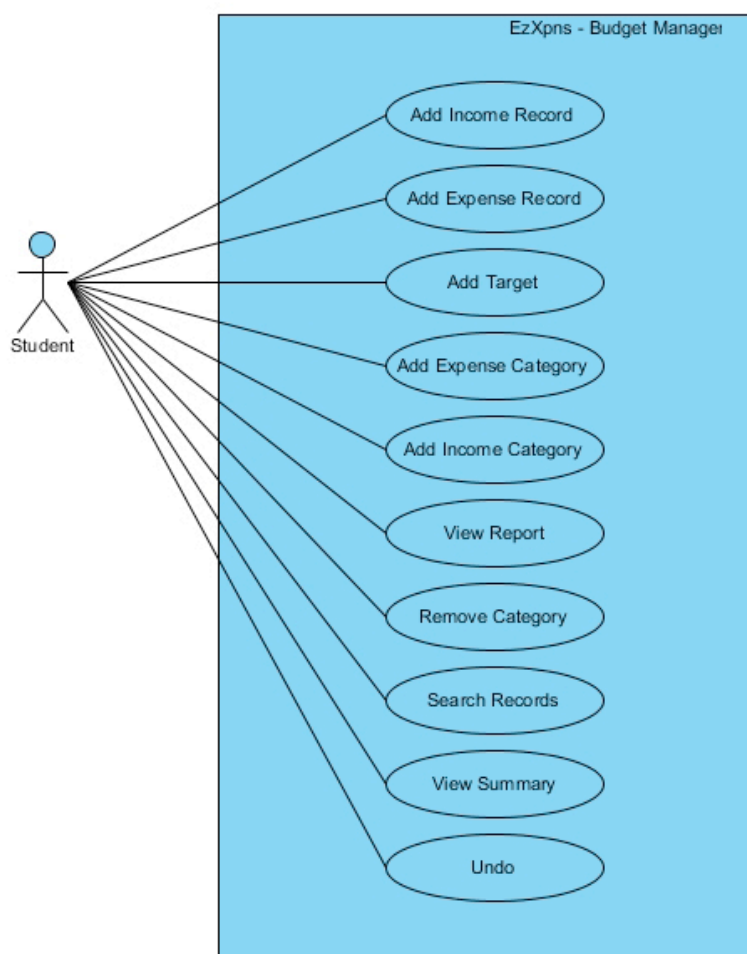
2. Development Infrastructure

The project is written on **Java Platform Standard Edition (Java SE) 6**, using **Eclipse** as the main Integrated Development Environment (IDE). An Eclipse plugin, **WindowBuilder**, is used for the creating the Graphical User Interface (GUI). **JFreeChart** and **JCalendar** libraries are used to generate the chart and date chooser respectively. The source code is currently hosted on Google Code at <http://code.google.com/p/cs2103jan13-w14-2j/>. The repository uses Mercurial as the version control system.

3. Use Cases

Use Case Diagram

(Refer to [Appendix A](#) for Detailed Use Case Description)



4. Architecture

The organisation of each major component is shown in Figure 1.

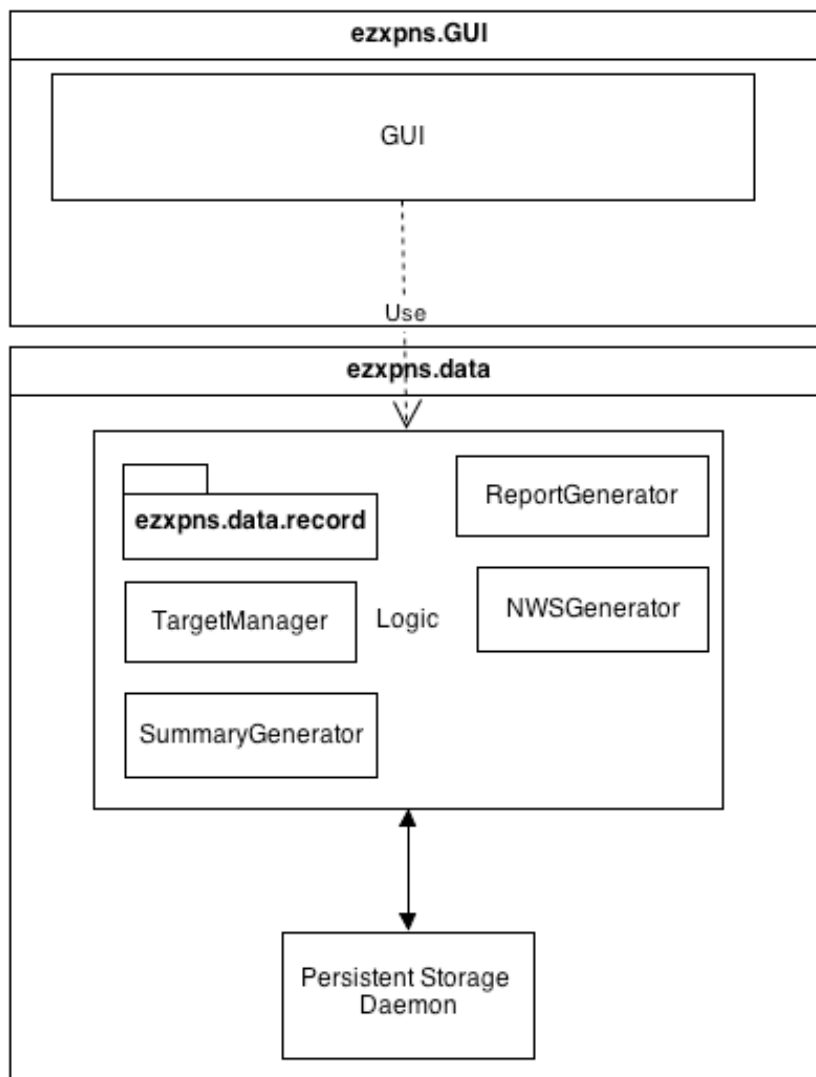


Figure 1. Architecture Diagram

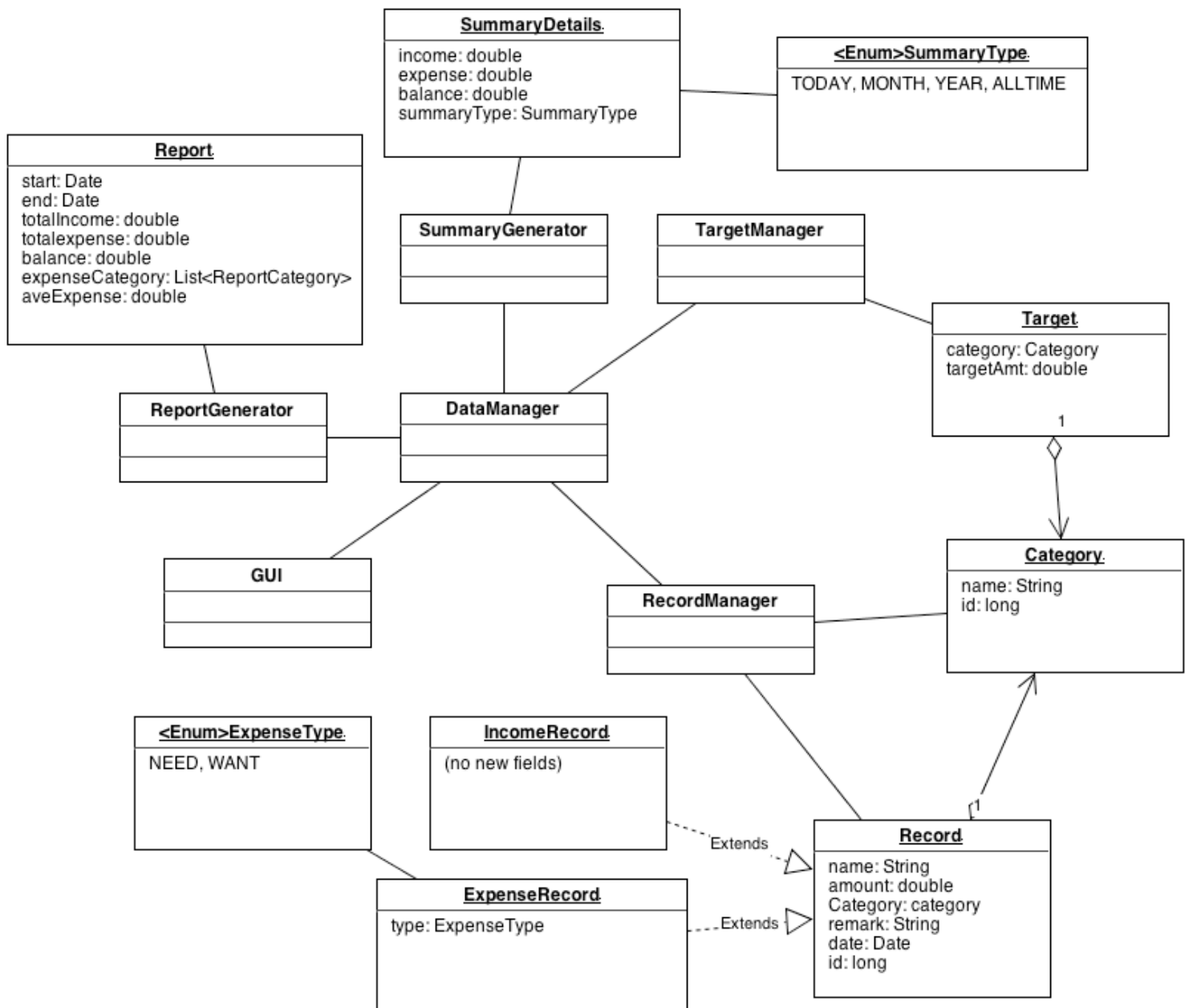
The program is designed such that each component can function with minimum dependency on the others, and that replacing any of them will not have major effect on the rest. As such, the components generally communicate with each other via interfaces. The functions of computation and data processing are also strictly separated from user interactions. This allows the data to be updated easily in the backend, without affecting the frontend.

For logic classes that need data access, it is recommended to define a `DataProvider` interface nested in the class and specify a dependency on it. Frequently, such dependency is enforced by making an object implementing the interface a compulsory parameter in the class constructor. The interface can then be implemented by either the data manager or its components directly. Alternatively, it can also be implemented by the `Ezxpns` class if it needs to have side effects on other objects. It is also possible to extract

methods that implement the `DataProvider` out of `EzXpns` and inject them into anonymous class or a new concrete class for better code organisation.

Please refer to [Appendix B](#) for sequence diagrams of some examples of tasks.

The **domain level** is shown below:



Domain level diagram

Class Diagram

To view the class diagram for each component, please refer to [Appendix C](#)

4.1 Storage and Serialisation

The data is stored in `Json` format, with the open source `gson` library from Google. All data that are persistent should be in `DataManager`. To work with `gson`, attributes of members or sub-members of persistent objects that should not be stored (for example, if they contain duplicate data) and should be marked as `transient`.

Classes that has attributes meant to be stored should also inherit `Storable`, which allows the `StorageManager` to check if data is updated i.e. data has be changed and should be saved. Whenever a `Storable` changes its internal data, it should mark itself as updated, so that `StorageManager` will save the data.

The storage mechanism is fully automatic. `StorageManager` will check whether the data is updated between each interval of 5 seconds (likely longer in the production version). If there is an update, it serializes the whole internal data, saves it, and marks all data as updated. The same checking is also done when the application quits. So the developer does not need to handle the storage once it is set up.

4.2 Data Management

As mentioned, all data will be handled by `DataManager`. Currently, it contains two `RecordManager`, a `TargetManager` and a `NWSGenerator`. Each of these then contains and manages atomic data units such as `Record` and `Category`.

Atomic Data Units

All atomic data units are immutable outside `ezxpns.data.record` package.



Take Note !

The immutability of atomic data allows us to easily optimize queries in `RecordManager` without circular reference or copying of object instances for the output.

Additionally, to prevent corruption of data, when methods like `createRecord(Record r)` are called, the record that is created and inserted into the data manager is not guaranteed to be the same instance as the parameter. It may contain different id, and can be a copy of the original object, while the other attributes should be the same.

4.3 Summary System

The Summary System will retrieve and store information to be displayed in the `OverviewPanel` in the main window.

The main class for the Summary System is the `SummaryGenerator`. `SummaryGenerator` will create four `SummaryDetails`, a data structure containing overall financial information over a specific timeframe. This is done by calling the method, `getSummaryDetails(SummaryType)`. There are four timeframes, expressed as `enums SummaryType`. There will be one `SummaryDetails` for each `SummaryType`. `SummaryGenerator` will retrieve relevant information for the four `SummaryType` through the `DataProvider` interface.

`markDataUpdate()` (in `SummaryGenerator`) should be called when you want to refresh the information (eg. when there is a change in the records).

4.4 Report System

The Report System will retrieve and store the information to be displayed for the `ReportFrame`.

`Report` is a data structure that holds information of the user's total income, total expenses, average expenses and balance within a timeframe, `start` and `end`.

These information is generated through `generateReport(Date start, Date end)`. `ReportGenerator` will retrieve records dated within the timeframe using the `DataProvider`. `ReportGenerator` will then process the records and store them in a `Report`.

`ReportCategory` is another data structure that represents the rows in an expense category table. Similarly, it is created by `ReportGenerator` and stored in `Report`.

4.5 Target System (with alerts)

The `TargetManager` manages the target system. It handles the creation, removal and modification of a `Target`. `Target` is a data class that represents the target that the user set for a particular expense `Category`. It contains a reference to the expense `Category` and the target amount of money.

`TargetManager` uses the `DataProvider` interface to retrieve the total monthly expense for a particular `Category`. It also inherits `Storable` to save the `Target` objects. It uses these information to construct the `Bar` objects.

`Bar` is a data structure that is used to display the user's target progress in the `TargetOverviewPanel` of the GUI. A `Bar` has a `BarColor` enum. It is defined as either `HIGH`, `MEDIUM`, `LOW` or `SAFE`. Each of this enum assigns an actual color bar chart in `TargetOverviewPanel` of the GUI. The intensity of the color signifies how high the user's expense is with respect to the target amount.

`TargetManager` also manages the alerts. Any `Bar` that is that has a `BarColor` of `HIGH` or `MEDIUM` are considered as alerts. `TargetManager` is able to a vector of the alerts using `getAlerts()`. The number of such alerts is reflected in the `TargetOverviewPanel`.

Only the `Target` objects are stored in the database and retrieved during start up. `Bar` is generated at runtime. `markDataUpdated()` should be called when a `Target` is created, modified or removed.

4.6 Needs, Wants, and Savings System

`NWSGenerator` generates the ratios for needs, wants and savings while `NWSdata` is a data structure that holds the information of the these ratios.

`NWSGenerator` uses the `DataProvider` interface to retrieve the user's expense and income records. It also inherits `Storable` to save its attributes. `NWSGenerator` sets the first month's ratios using the default values for needs, wants and savings. For the next month onwards, the new ratios will be generated using the `generateRatios()` method in `NWSGenerator`. The ratios are meant to be targets for the user to achieve. The ratios are generated based on the user's performance in the previous month. i.e the difference between the target percentage and the actual percentage.

In general, `generateNWSratios()` is called when the program enters a new month and there is a need to generate a new set of ratios for this month. However in the case when the user alters the previous month's records, `generateNWSRatios()` will be called to regenerate this month's ratios.

`NWSdata` holds the target ratios and the user's performance i.e. the user's total expense on needs etc. `NWSGenerator` has two instances of `NWSdata`, namely `thisMonthNWS` and `pastMonthNWS`. `thisMonthNWS` holds the target ratios and the user's performance for this month and `pastMonthNWS` holds the target ratios and the user's performance for the previous month. `markDataUpdated()` should be called when there are changes to `thisMonthNWS` and `pastMonthNWS`.

This month's ratio & previous month's ratio

This month's ratio is entirely dependent on the previous month's ratio.

The exceptions are:

- (1) During the first month of usage
- (2) User has not recorded any expenses or income for the whole of the previous month
- (3) User permanently deletes all the expenses and income records of the previous month

In these cases, this month's ratio will be set as default.



Take Note !

`NWSGenerator` provides a copy of `thisMonthNWS` to the GUI to display in the `SavingsOverviewPanel`.

User's Expenses

The user's expenses will not be reflected in the panel until an income record is added for that month.



Take Note !

4.7 Graphical User Interface (GUI)

The GUI is envisioned to be simple and intuitive. Hence, one of the more fundamental aspects of the GUI is the single streamlined modular window interface. A modular window interface usually allows only one active screen at any point of time. `UIControl` is a control component that assists the program in the manipulation of the various windows. This includes the switching from various tabs as well as the opening and closing of dialog windows. The only exception is the `ShutdownHook` that is activated by the `StorageManager` on startup. It handles the exit of the application and saves the changes made to the user's data.

`MainGUI` is another important component in this design. Unlike `UIControl`, `MainGUI` is the main window that displays and maintains the various screens that the user can expect to see, such as dashboard and search. Its contents are governed by the class `EzNavigator`, which uses `CardLayout` in `java.awt` to effectively switch the content that `MainGUI` displays. However, in order to differentiate between opening dialogs and switching from one screen to another, `EzNavigator` contains a few types of `MenuOption`, mainly a normal menu option, or `NormalMenuOpt`, which is used to switch the display. Another type will be a dialog menu option, or `DialogMenuOpt`, which is used to open a dialog popup.

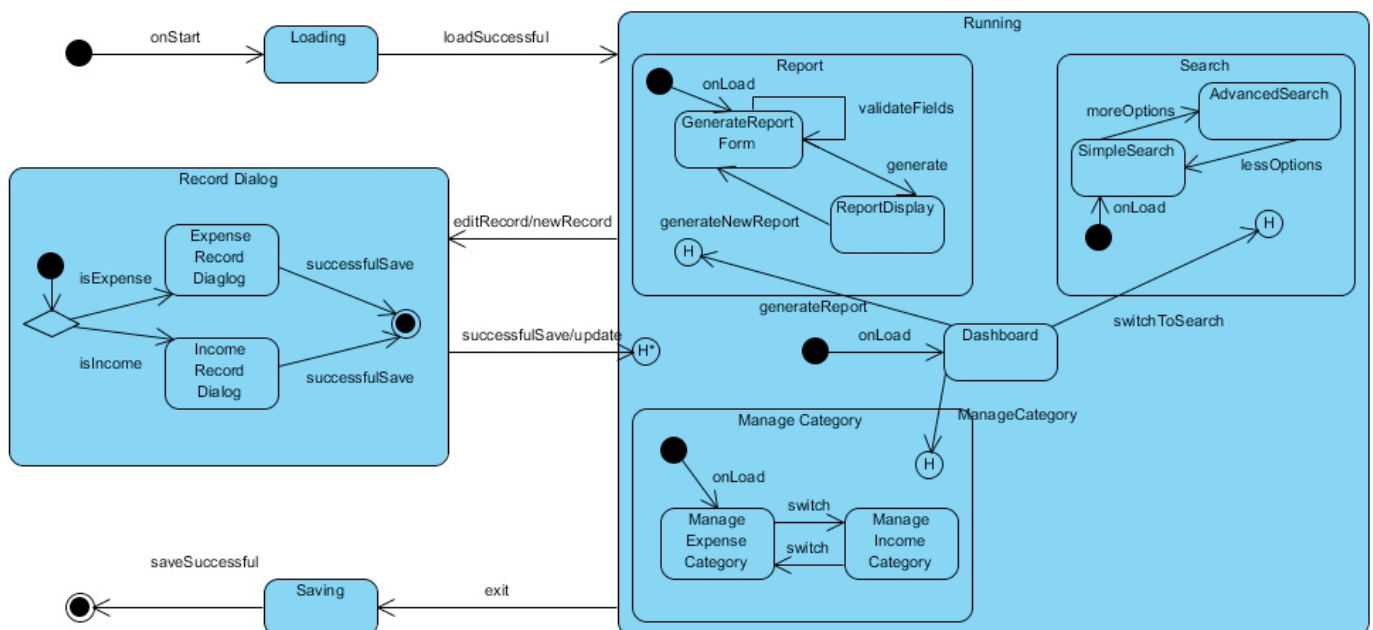
The GUI components use Java libraries such as `javax.swing` for the visual components and `java.awt` for the event handling component. However, at the current iteration as the priority was placed on functionality over visual aesthetics, the `MainGUI` does not have all

the possible functionality of event handlers and visual aids. The ones implemented comprises (but not limited to) of Window Resize and display of error messages for content validation. Some of the unimplemented functions would be an extensive and comprehensive tooltip help or information display for some user controls, and shortcut keys for power users.

Homogenous Look & Feel

To enforce a homogeneous look and feel, it is recommended to use a generic method or class to create GUI components such as buttons and textfields. An example of such would be in `RecordFrame; createLabel(String lblText)` is used to create labels with the same font.

Below is the State-Machine Diagram for *EzXpns*.



4.8 The Undo Framework

Since undo is a purely front-end action, all undo-related implementation is done in the GUI package.

The core of this framework is the `UndoManager`, which maintains a stack of actions in the form of `AbstractAction`. These actions will be performed when the user request to undo. This `AbstractAction` can be bounded to the menu items or buttons directly. A new undo action will be added into the stack through the method `UndoManager.add (AbstractAction,String)` when the user makes changes to the data. The `String` describes the act of undoing the change that the user has made eg. Undo New Income.

The following is an example of adding an `UndoAction` for a creating a new income record:

```
private AbstractAction createUndoAction(  
    final IncomeRecord nRecord,  
    final boolean isNewCat) {  
return new AbstractAction() {  
    public void actionPerformed(ActionEvent event) {  
        if(isEdit) { // Undo an edit.  
            recHandler.modifyRecord(record.getId(), record,  
false);  
        }  
        else { // A new record  
            // Remove new Record  
            recHandler.removeRecord(nRecord.getId());  
        }  
        if(isNewCat) { // New Category Added  
            // Remove new Category  
            catHandler.removeCategory(nRecord.getCategory  
().getID());  
        }  
    }  
}; // End of new AbstractAction  
} // End of createUndoAction
```

An instance of `UndoManager` is contained in `MainGUI` which implements the `UpdateNotifyee` interface. The `UpdateNotifyee` interface provides a proxy method `addUndoAction(AbstractAction,String)` that adds a new undo action to the `UndoManager`. Therefore, any frontend classes that want to make certain user actions revertible should contain a reference to an `UpdateNotifyee`. Currently, the only `UpdateNotifyee` is the `MainGUI`.

To create an undo action, the frontend class will need to override the `ActionPerformed` method of an anonymous `AbstractAction` with the actual code that executes the undo action. This `AbstractAction` is then added to the `UndoManager` through `UpdateNotifys.addUndoAction`.

Below is an example for creating an undo action for modifying income category:

```
private AbstractAction getUndoModifyInCat(final long id,
final Category original){
    return new AbstractAction(){

        @Override
        public void actionPerformed(ActionEvent e) {
            incats.updateCategory(id, original);
        }

    };
}

// in the function that modifies the category
....
//curInCat is the category being modified
Category original = curInCat.copy();
// cat is the modified category
Category cat = incats.updateCategory(curInCat.getID(),
new Category(inNameField.getText()));
notifys.addUndoAction(getUndoModifyInCat(cat.getID(), original),
"Modify category");
```

5. Testing Framework

We currently have automatic testing only for unit testing of backend data management and summary generator, and *JUnit* is used for this. All testing code should be put in the package `ezxpns.test`, with self-explanatory names. It is recommended that the developer should write additional tests for any new features he creates.

Note that `DataManager` is usually loaded from the `Json` file, but this is unnecessary in unit tests. It is recommended to just construct a non-persistent `DataManager` directly for testing, so that the data file will not be corrupted.

Moreover, when testing for classes that retrieve their data from their `DataProvider`, it is usually unnecessary to create the actual `DataProvider`. Using an ad-hoc class that acts as such `DataProvider` is more desirable since it reduces the complexity of the testing code. For an example of such construct, take a look at `test/SummaryTest.java`.

6. Known Issues

1. Serialisation / deserialisation involves reading and writing the whole database into the disk. This can slow down the program if there are many records. We should consider using archive mechanisms, or switch to a real database.
2. When editing records or categories, an undo action will be created and added to the undo stack even if the user does not change anything. This can be confusing to the users.
3. IO errors that may happen during loading and saving are not handled by the GUI at the moment.
4. There are no keyboard shortcuts for most of the tasks.

Project Process & Administration

Responsibilities

Name	Responsibilities
Andrew	GUI and documentation diagrams
Shu Zhen	Target/alert system, design and implementation of Needs, Wants, & Savings system
Ting Zhe	Report and Summary system, graphics in GUI
Yujian	Overall architecture, backend, data management and some minor front end implementation

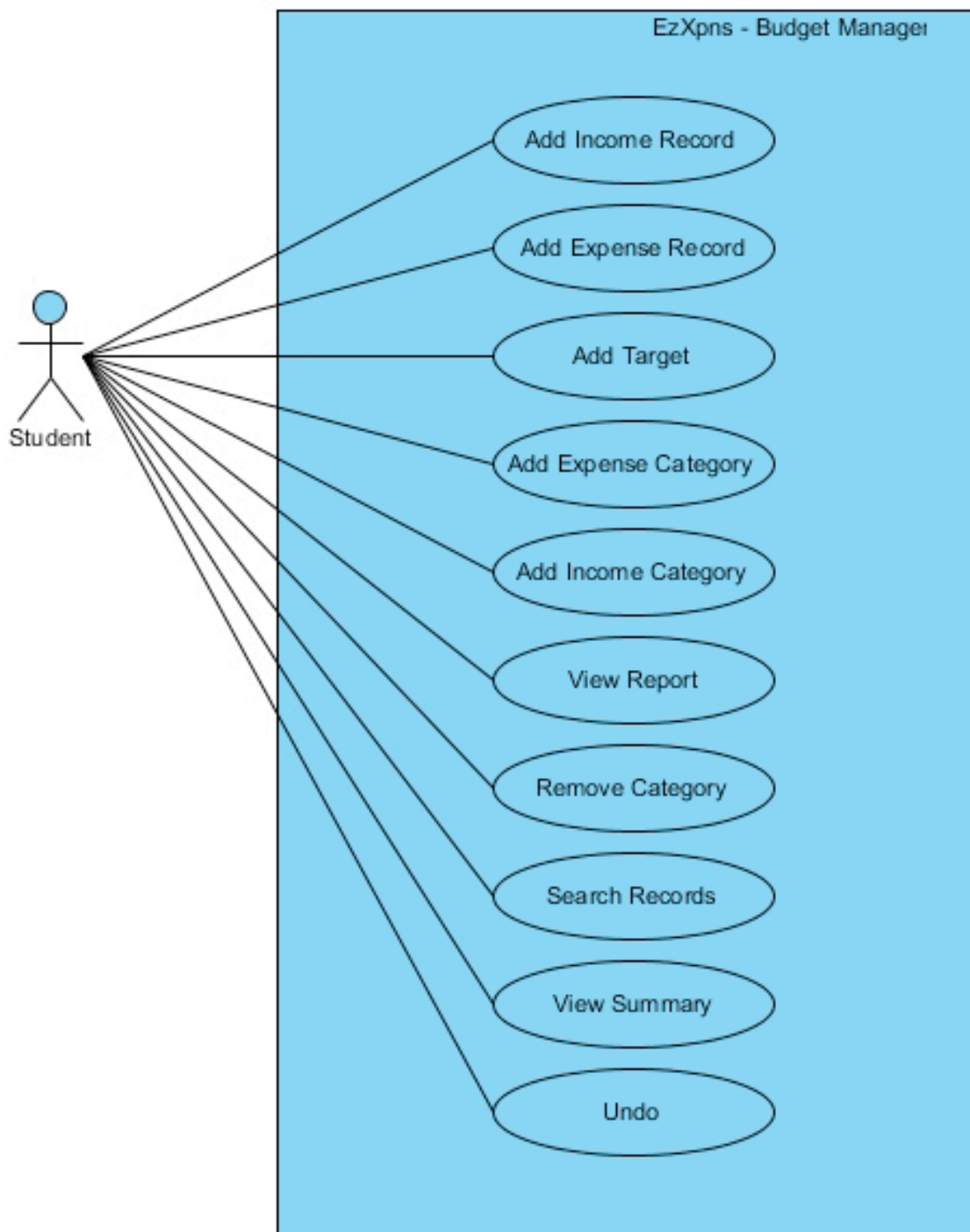
Timeline

Week	Task
7	<ul style="list-style-type: none">• Basic storage and retrieval• Basic search in backend• GUI and implementation for alert• Report fully designed and implemented
8	<ul style="list-style-type: none">• All components integrated• All Basic features done• Start redesign/optimisation of GUI• Start implementing fancy features
9	Version 0.1 is due. Prototype.
9-11	Polish Must-Have and Good-to-have features. Improved GUI
11	Version 0.2 due on Monday 15 April

Appendix

Appendix A: Use Case Descriptions

Use Case Diagram



Use Case: 01 - Add New Expense Record

Actor(s): User

Main Success Scenario (MSS):

1. User requests to add a new expense record
2. System displays new expense record form
3. User fills up the new expense record form
4. System validates the input in the required fields such as amount, date, category ect.
5. System displays success in adding new expense record

Use case ends

Alternate Flow: Users enters invalid input

4a.1 System displays problems with input

Return to MSS: 3

Use Case: 02 - Add New Income Record

Actor(s): User

MSS:

1. User requests to add a new income record
2. System displays new income record form
3. User fills up the new income record form
4. System validates the input in the required fields such as amount, date, category ect.
5. System display success in adding new income record

Use case ends.

Alternate Flow: Users enters invalid input

4a.1 System displays problems with input

Return to MSS: 3

Use Case: 03 - Add a Target

Actor(s): User

MSS:

1. User requests to add a new expense target
2. System display list of expense category
3. User select the category to set a target on
4. System displays form for the target
5. User fills in the required fields
6. System validates the input
7. System stores the target and displays success

Use case ends.

Alternate Flow: User enters invalid input

4a.1. System displays problems with input

Returns to MSS: 4

Use Case: 04 - Add an Expense Category

Actor(s): User

MSS:

1. User requests for Category Manager
2. System creates a window with available options
3. User selects "Add new Category" under expense
4. System creates a field for user to add a new expense category
5. User fills up the required details for adding a new expense category
6. System validates the details (eg. name)
7. System displays success

Use case ends.

Alternate Flow: User entered invalid name

5a.1. System displays problems with input

Return to MSS: 4.

Use Case: 05 - Add an Income Category

Actor(s): User

MSS:

1. User selects the "Manage Category" tab
2. System creates a window with available options
3. User selects the "Income" tab
4. System changes to the "Income" tab with available options
5. User selects "Add new category" in the "Income" tab
6. System creates a field for the user to add a new income category
7. User fills up the required details for adding a new expense category
8. System validates the input
9. System displays success

Use case ends

Alternate Flow: User entered invalid name

7a.1. System displays problems with input

Return to MSS: 6.

Use Case: 06 - View Report

Actor(s): User

MSS:

1. User requests to view report
2. System displays a form to request for the timeframe for the report
3. Users keys in start date and end date
4. System retrieves the records based on the timeframe provided by the User
5. System formats and updates the display of information on the screen

Use case ends.

Alternate Flow: User keys in illegal characters (eg. alphabets)

3a1: System clears input

Return to MSS at step 2

Use Case: 07 - Remove a Category

1. User requests to remove a category
2. System displays a list of category
3. User selects a category to be removed
4. System display warns user that all records under this category will be classified under 'undefined'
5. User accepts condition
6. System displays success

Use case ends

Alternate Flow: User rejects the condition

5a1: Category is not deleted

Returns to MSS at step 2

Use Case: 08- Search for Records

Actor(s): User

MSS:

1. User requests to search past records
2. System displays Search window
3. User fills up the relevant fields for the search
4. System retrieves from data storage the matching searches
5. System displays the search results

Use case ends.

Use Case 09 - View Summary

Actor(s): User

MSS:

1. User requests to view summary
2. System show monthly summary by default and available options
3. User views summary

Use case ends.

Alternate Flow: User requests to view annual summary

2a1: System show annual summary

Returns to MSS: 3

Alternate Flow: User requests to view daily summary

2b1: System show daily summary

Returns to MSS: 3

Use Case 10 - Undo an Operation

Actor(s): User

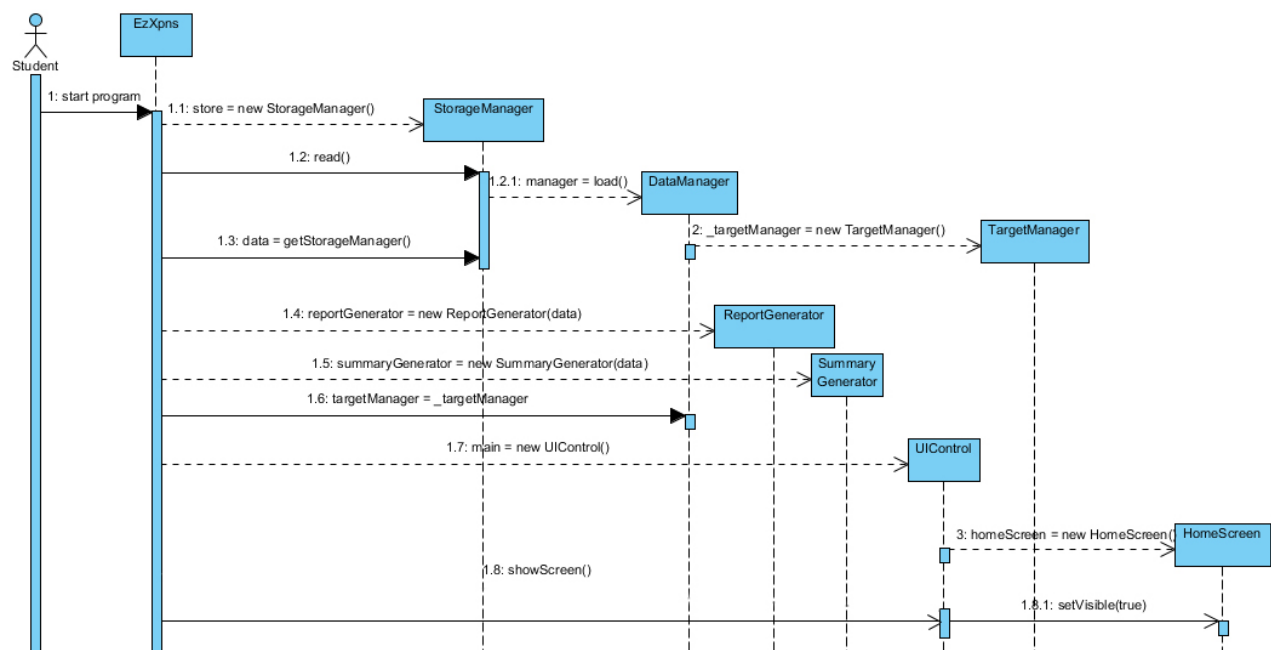
MSS:

1. User requests to undo the previous operation
2. System performs undo operation
3. System displays success

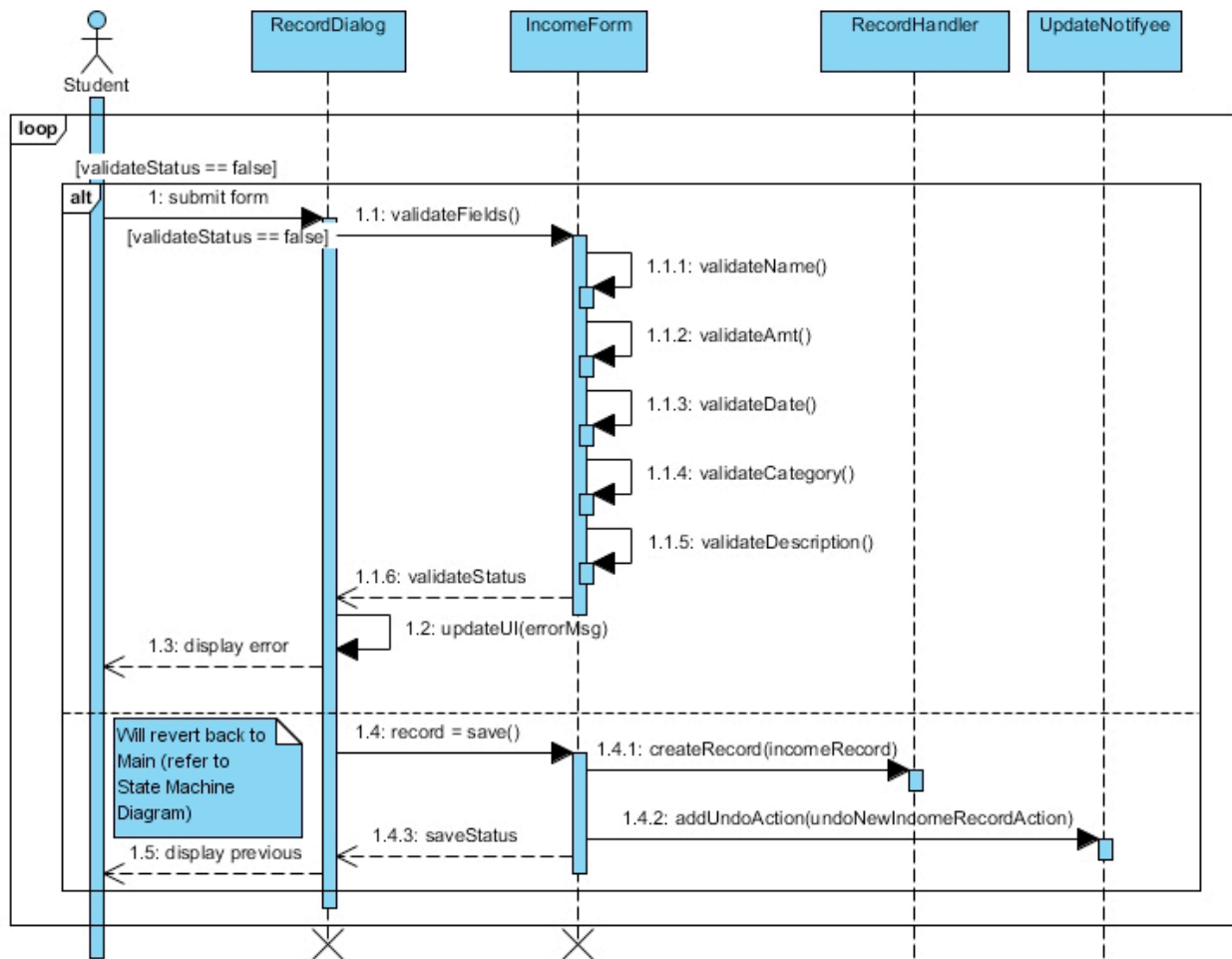
Use case ends.

Appendix B: Sequence Diagram

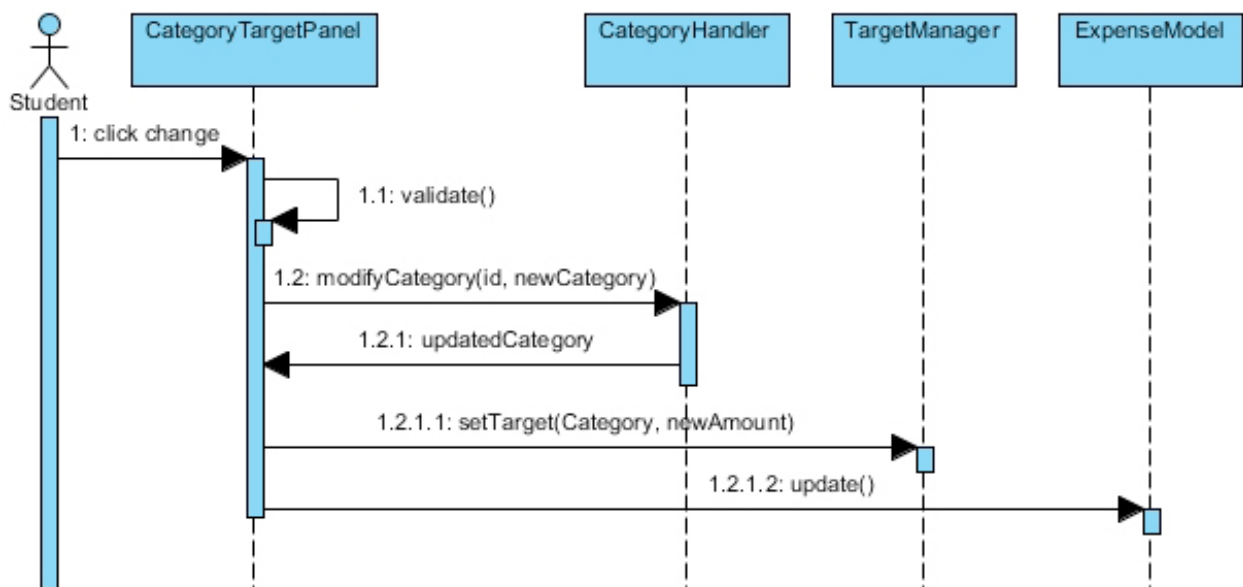
EzXpns Start Sequence



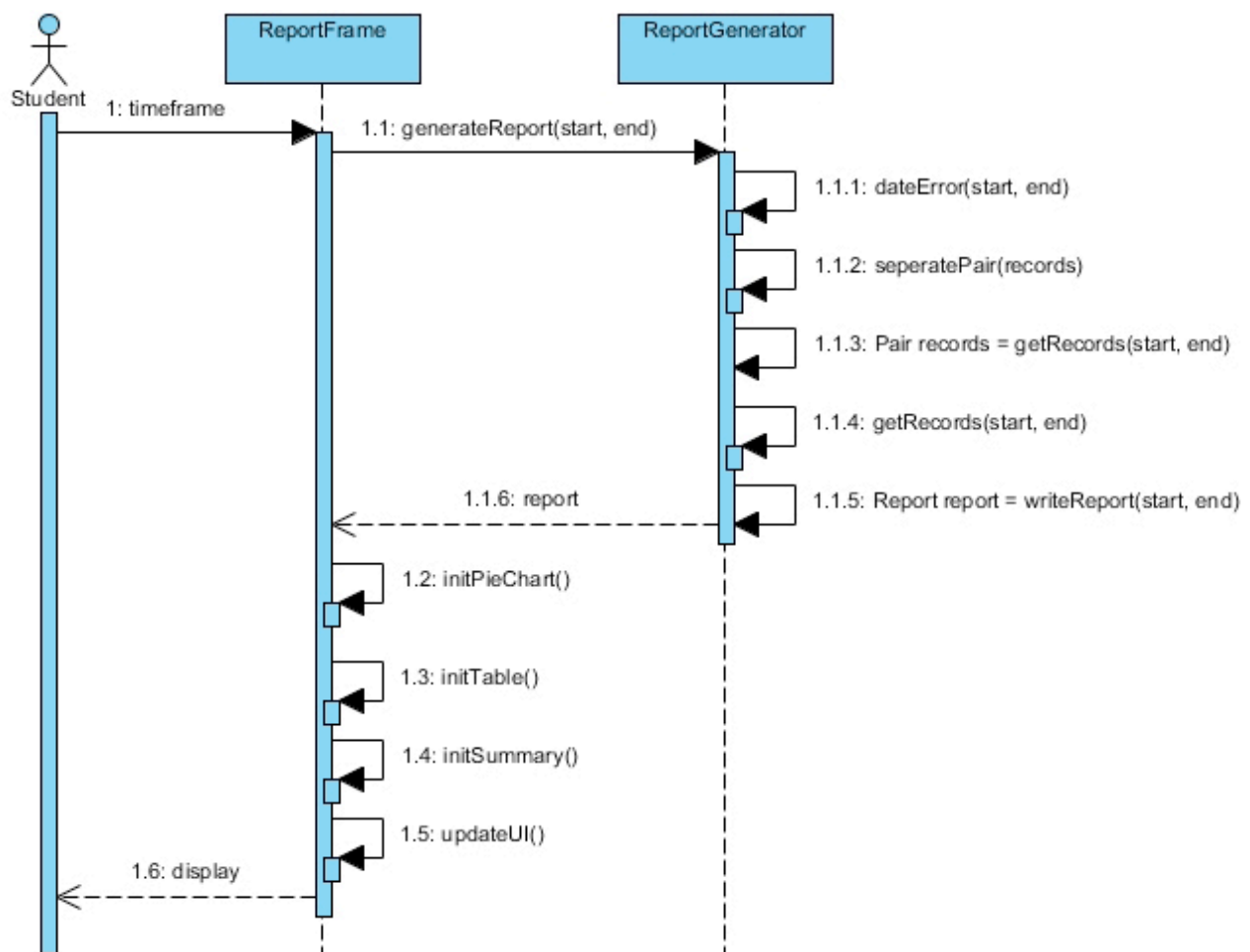
Add New Income Record



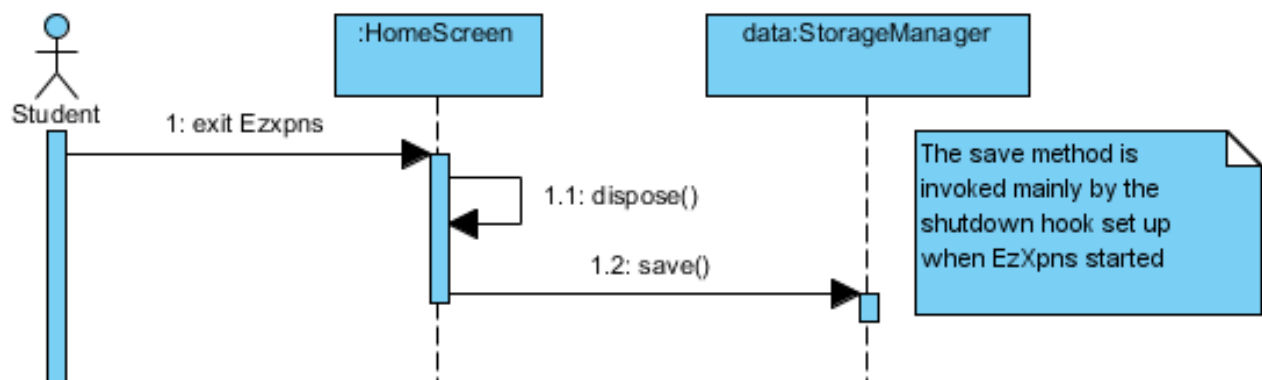
Modify Category



Generate Report

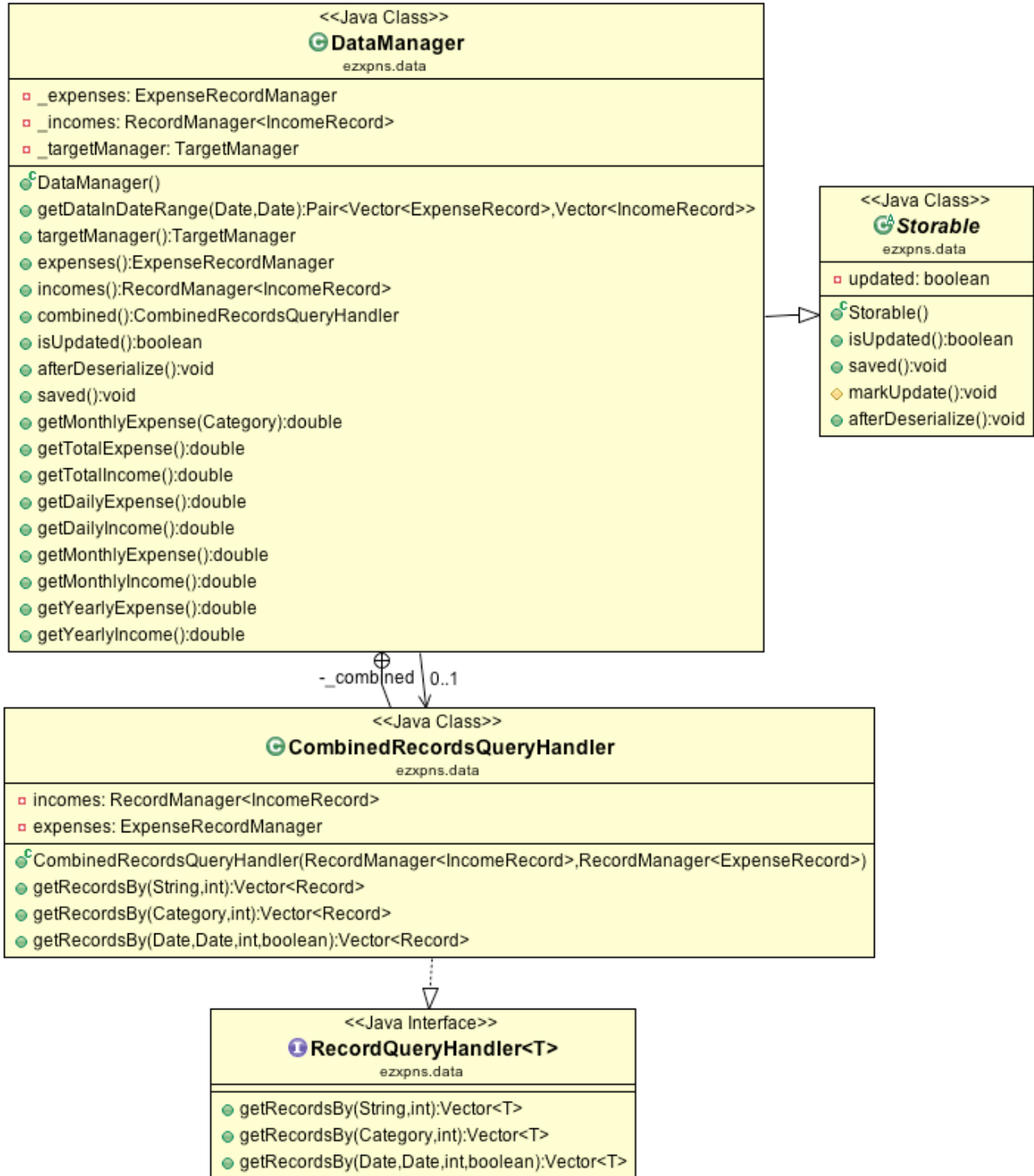


EzXpns Exit Sequence

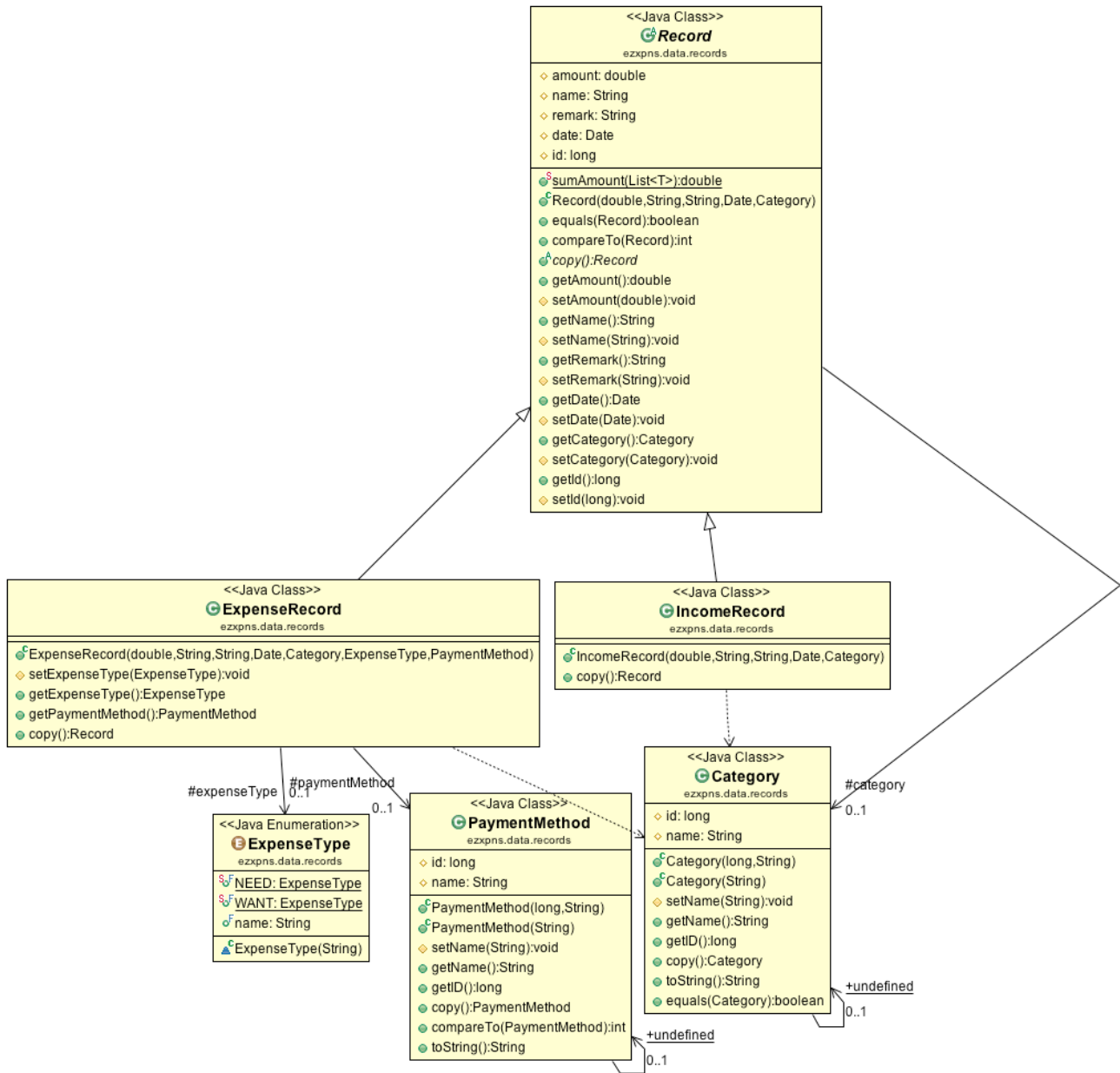


Appendix C: Class Diagrams

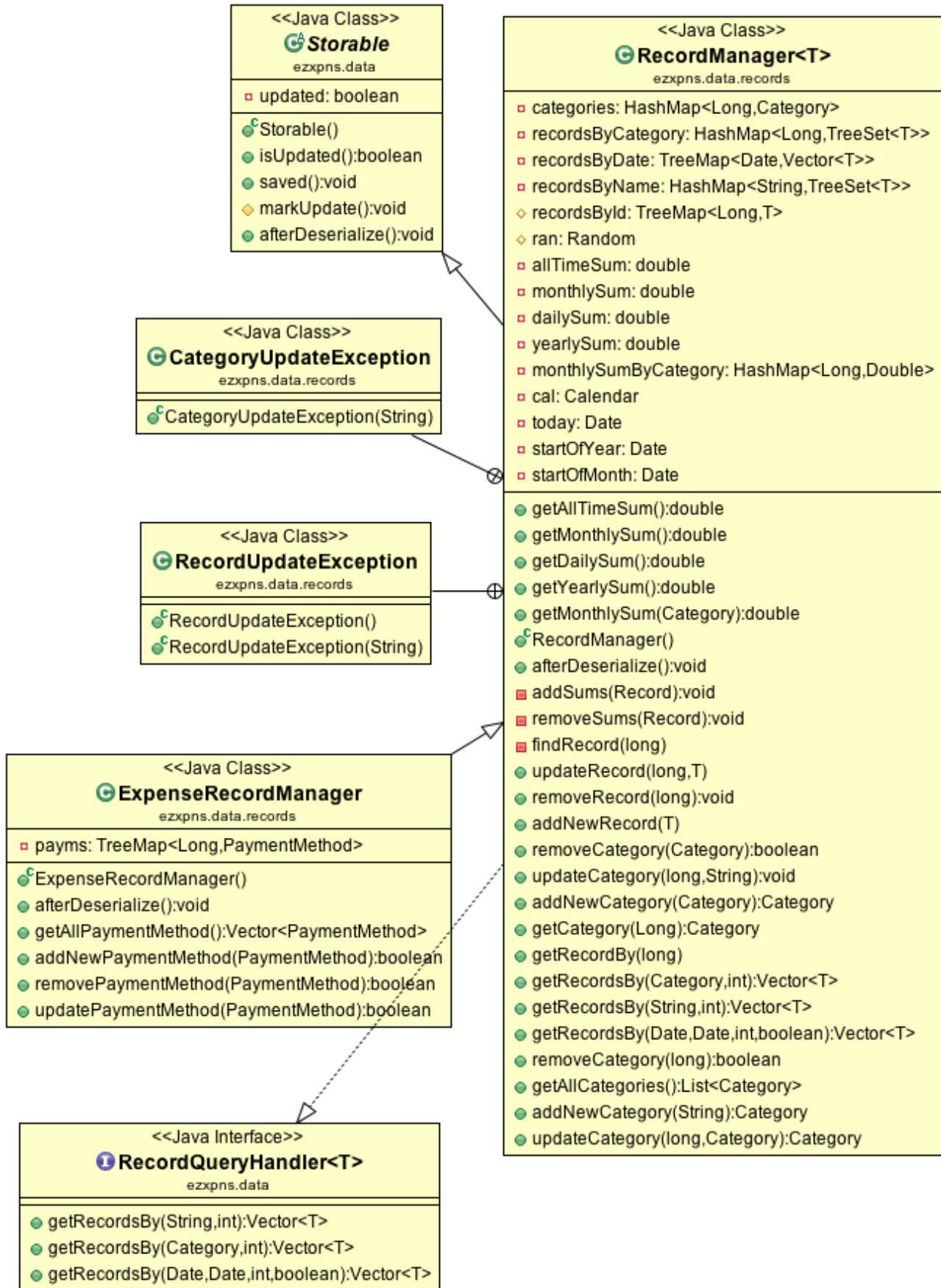
DataManager Class Diagram



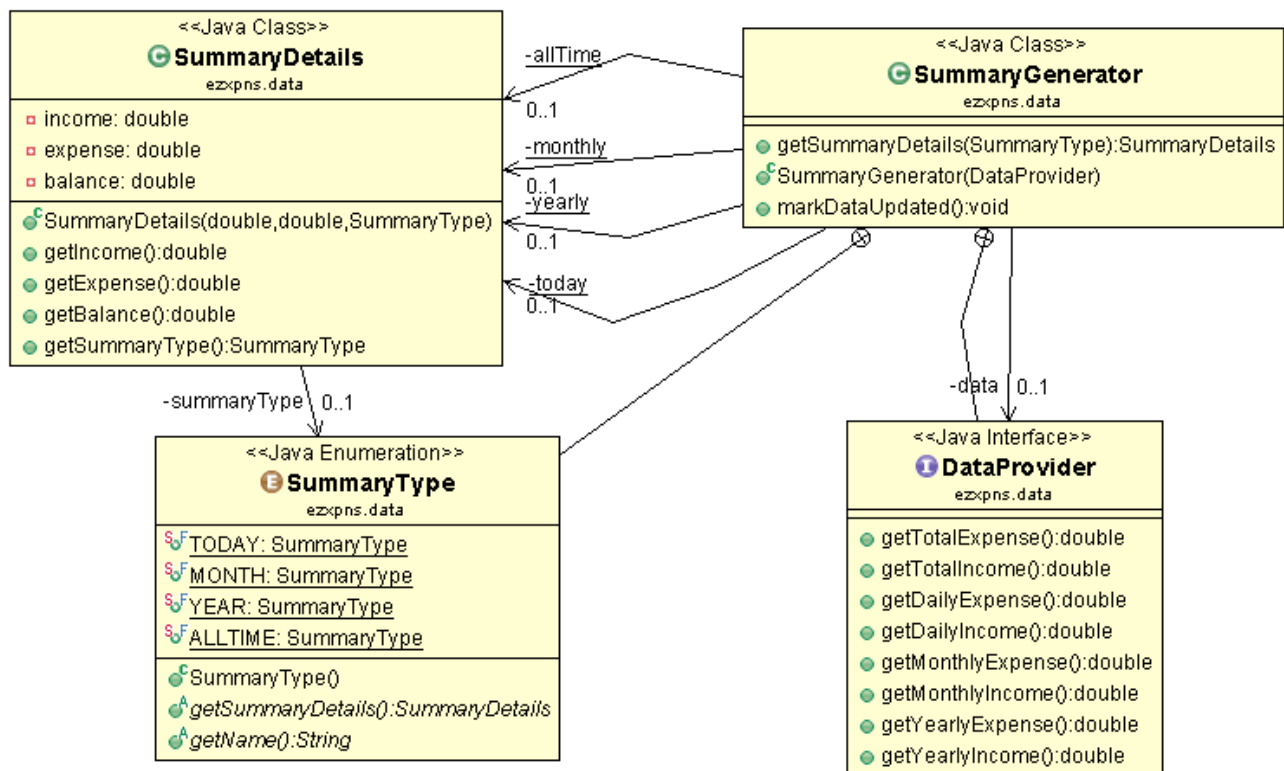
Atomic Data Unit Class Diagram



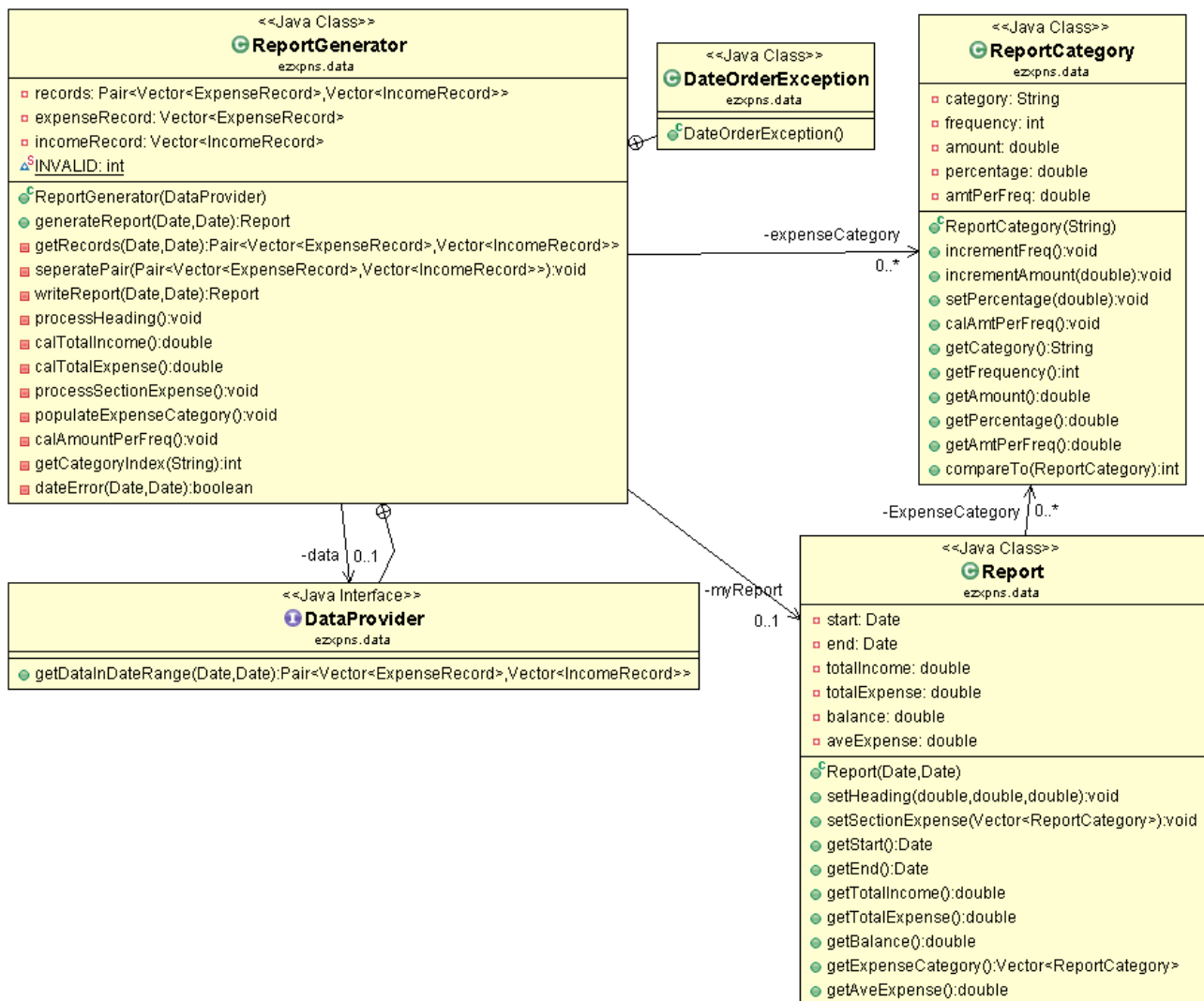
RecordManager Class Diagram



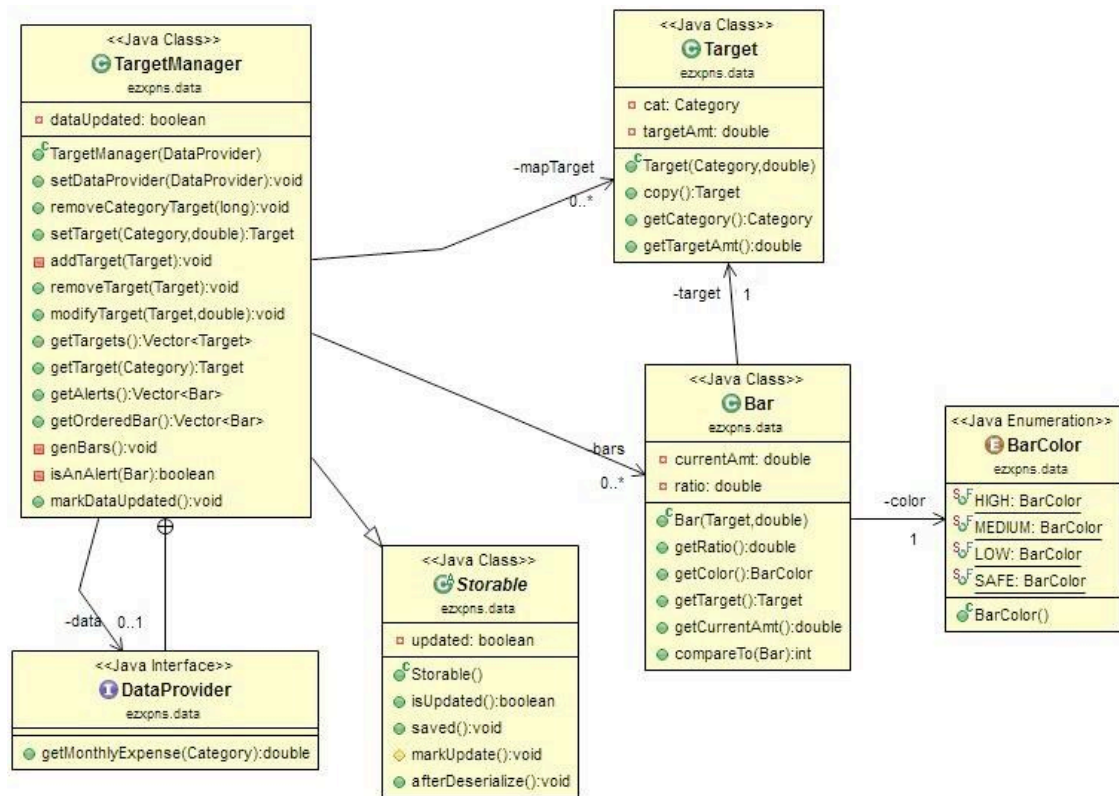
Summary System Class Diagram



Report System Class Diagram



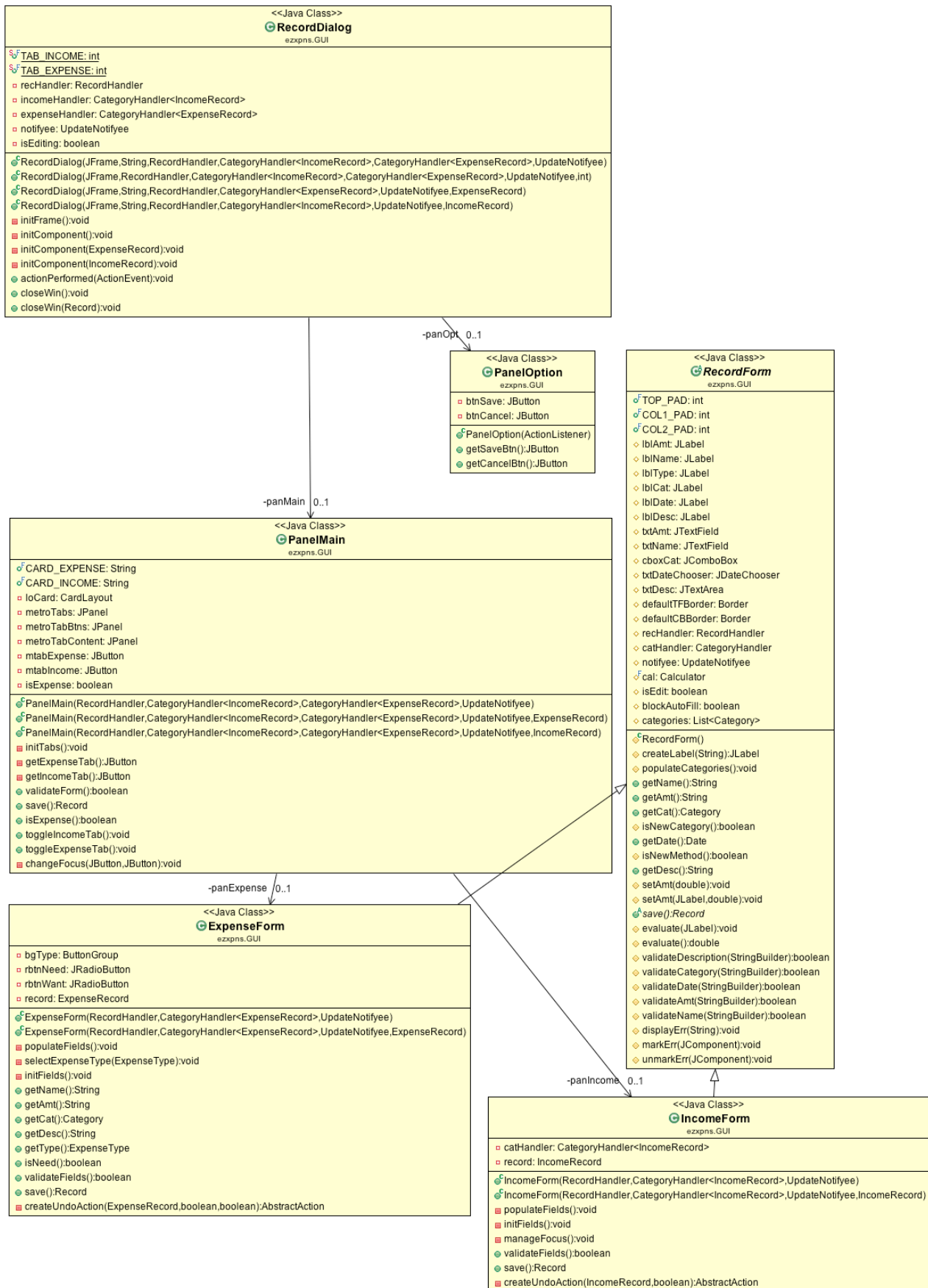
Target System Class Diagram



Needs, Wants, & Savings Class Diagram



GUI Class Diagram



Appendix D: API

1) Package ezxpns

Class Ezxpns

A main class that links up various components.

Method Summary	
Category	addNewCategory (Category newCat) Creates a new category Note: the new category may be copied with a different id
Category	addNewCategory (java.lang.String catName) Creates a new Category with the given name
boolean	addToCategory (java.util.List< ExpenseRecord > records, Category cat) Adds a list of records to a category
boolean	containsCategoryName (java.lang.String name) Returns true if name is in use
ExpenseRecord	createRecord (ExpenseRecord r, boolean newCat, boolean newPay) Creates a new expense record with flags for new expense category and new payment method
IncomeRecord	createRecord (IncomeRecord r, boolean newCat) Creates a new income record with a new category
java.util.List< Category >	getAllCategories () Returns a list of all user defined categories
Category	getCategory (long id) Returns a category that has the specified id
java.util.Vector< Category >	getCategoryWithNamePrefix (java.lang.String prefix) Returns a vector of records that matches the prefix
DataManager	getDataMng () Returns the DataManager that the application uses. It is used for testing
Record	getRecord (long identifier) Returns a record that matches the identifier
java.util.List< Record >	getRecords (int n) Returns a list of the most recent n records
java.util.Vector< ExpenseRecord >	getRecordsBy (Category category, int max) Returns a vector of maximum max records that is under the specified category

StorageManager	getStore() Returns the <code>StorageManager</code> that the application uses. It is used for testing.
TargetManager	getTargetManager() Get the <code>TargetManager</code> the application uses. It is useful for testing.
ExpenseRecord	lastExpenseRecord (java.lang.String name) Return the most recent record that matches the name. If no match is found, it returns null
IncomeRecord	lastIncomeRecord (java.lang.String name) Returns the latest expense record matching the name, or null if the record does not exist
boolean	modifyRecord (long id, ExpenseRecord r, boolean newCat, boolean newPay) Modifies an expense record
boolean	modifyRecord (long id, IncomeRecord r, boolean newCat) Modifies an income record
boolean	removeCategory (long identifier) Removes a user defined category that matches the given identifier
boolean	removeRecord (long identifier) Removes record that matches the given identifier
java.util.Vector< Record >	search (SearchRequest req) Returns a vector of records that matches the search request
java.util.Vector< Record >	search (java.lang.String partialMatch) Returns a vector of records that matches the name partially
Category	updateCategory (long identifier, Category selectedCat) Modifies category and returns the modified category
java.lang.String	validateCategoryName (java.lang.String name) Returns an error message if the name is not accepted, or a confirmation message if the name is accepted.

2) Package ezxpns.data

Class Bar

A data class that holds the data that is meant to be displayed.

Method Summary	
BarColor	getBarColor() Returns the enum of <code>BarColor</code>
double	getCurrentAmt() Returns the current amount of this <code>Bar</code> object

double	getCurrentPercentage() Returns the percentage of current amount with respect to target amount
double	getRatio() Returns the ratio of the current amount with respect to the target amount
double	getRemainingAmt() Returns difference between the current amount and the target amount
Target	getTarget() Returns the <code>Target</code> object of this bar
double	getTargetAmt() Returns the target amount that the user has set for the target of this bar

Enum BarColor

Enum Constant Summary	
HIGH	BarColor is HIGH when the ratio of the current amount with respect to the target amount is more than 0.8
LOW	BarColor is LOW when the ratio of the current amount with respect to the target amount is more or equals to 0.65 and less than 0.8
MEDIUM	BarColor is MEDIUM when the ratio of the current amount with respect to the target amount is more or equals to 0.35 and less than 0.65
SAFE	BarColor is SAFE when the ratio of the current amount with respect to the target amount is less than 0.35

Method Summary	
abstract java.awt.Color	getColor() Returns a Color object based on the BarColor
static BarColor	valueOf(java.lang.String name) Returns the enum constant of this type with the specified name
static BarColor []	values() Returns an array containing the constants of this enum type, in the order they are declared

Class DataManager.CombinedRecordsQueryHandler

A helper class that handles queries regarding `ExpenseRecord` and `IncomeRecord`

Method Summary	
java.util.Vector< Record >	getRecordsBy (Category category, int max) Returns a vector of maximum <code>max</code> records that matches the category
java.util.Vector< Record >	getRecordsBy (java.util.Date start, java.util.Date end, int max, boolean reverse) Returns a vector of maximum <code>max</code> records within the date range, inclusive of both ends
java.util.Vector< Record >	getRecordsBy (java.lang.String name, int max) Returns a vector of maximum <code>max</code> records that matches the name
java.util.Vector< Record >	getRecordsByCategory (java.lang.String name) Returns a vector of maximum <code>max</code> records that matches the category name
java.util.Vector< Record >	getRecordsWithNamePrefix (java.lang.String prefix) Returns a vector of records that matches the prefix

Class DataManager

A wrapper class that contains all the data. It has some helper functions to query both `ExpenseRecord` and `IncomeRecord`.

All the components here uses `StorageManager` to save to `json`.

Method Summary	
void	afterDeserialize () Optional method to populate transient attributes after deserializing the data from <code>json</code>
DataManager.CombinedRecordsQueryHandler	combined () Get a query handler that returns both income and expense records
ExpenseRecordManager	expenses () Get an instance of <code>ExpenseRecordManager</code>
Category	getCategory (long id) Returns the <code>Category</code> that has specified id
double	getDailyExpense () Return the sum of all expenses recorded on same day
double	getDailyIncome () Return the sum of all incomes recorded on same day

Pair < java.util.Vector < ExpenseRecord >, java.util.Vector < IncomeRecord >>	getDataInDateRange (java.util.Date start, java.util.Date end) Returns a Pair of vectors of ExpenseRecord and IncomeRecord within the starting date and ending date, inclusive
double	getMonthlyExpense () Returns the sum of all expenses for this month
double	getMonthlyExpense (Category cat) Returns the sum of all expenses that matches the category for this month
double	getMonthlyExpense (ExpenseType type) Returns the sum of all expenses that matches the expense type for this month
double	getMonthlyIncome () Returns the sum of all incomes for this month
double	getPrevMonthlyExpense (ExpenseType type) Returns the sum of all expenses that matches the expense type for the previous month
double	getPrevMonthlyIncome () Returns the sum of all incomes for the previous month
double	getTotalExpense () Returns the sum of all expenses
double	getTotalIncome () Returns the sum of all incomes
double	getYearlyExpense () Returns the sum of all expenses for this year
double	getYearlyIncome () Returns the sum of all incomes for this year
RecordManager < IncomeRecord >	incomes () Returns a vector of all the IncomeRecord in the database
boolean	isUpdated () Returns true if data has been updated by the user and needs to be stored
NWSGenerator	nwsGen () Creates an instance of NWSGenerator
void	saved () Informs the object that the data has been stored in the database
TargetManager	targetManager () Creates an instance of TargetManager

Class NWSdata

A data class that holds information of Needs, Wants and Savings

Method Summary	
NWSdata	copy() Returns a copy of NWSdata
double	getCurrentNeeds() Returns the current amount of expenses on needs in NWSdata
double	getCurrentSavings() Returns the current amount of savings in NWSdata
double	getCurrentWants() Returns the current amount of expenses on wants in NWSdata
double	getCurrNeedsRatio() Returns the ratio of current amount of expenses on needs with respect to income in NWSdata
double	getCurrSavingsRatio() Returns the ratio of current amount of expenses on wants with respect to income in NWSdata
double	getCurrWantsRatio() Returns the ratio of current amount of savings with respect to income in NWSdata
java.util.Calendar	getDate() Returns the date of NWSdata
double	getIncome() Returns the income of NWSdata
double	getTargetNeedsRatio() Returns the target ratio for needs with respects to income
double	getTargetSavingsRatio() Returns the target ratio for savings with respects to income
double	getTargetWantsRatio() Returns the target ratio for wants with respects to income
boolean	isValid() Returns true if NWSdata is valid
void	setAll (java.util.Calendar date, double tN, double tW, double tS, double cN, double cW, double cS, double ic) Sets all the attributes in NWSdata

Interface NWSGenerator.DataProvider

Method Summary	
double	getMonthlyExpense (ExpenseType type) Returns the sum of all expenses that match the expense type for this month
double	getMonthlyIncome () Returns the sum of all incomes for this month
double	getPrevMonthlyExpense (ExpenseType type) Returns the sum of all expenses for the previous month
double	getPrevMonthlyIncome () Returns the sum of all incomes for the previous month

Class NWSGenerator

This class generates the target ratios for needs, wants and savings.

Method Summary	
void	afterDeserialize () Updates data after deserialization
void	generateNWS () Generates and sets the ratio for NEED, WANT and SAVINGS
NWSdata	getNWSdataCopy () Returns a copy of NWSdata for this month
void	markDataUpdated () Marks data as updated
void	setDataProvider (NWSGenerator.DataProvider data) Sets the DataProvider interface
void	updateNWSdata () Updates NWSdata and saves it in json

Interface RecordQueryHandler<T extends [Record](#)>

This is an interface that handles the search queries for Record

Method Summary	
java.util.Vector<T>	getRecordsBy (Category category, int max) Returns a vector of maximum max records that matches the category
java.util.Vector<T>	getRecordsBy (java.util.Date start, java.util.Date end, int max, boolean reverse) Returns a vector of maximum max records within the date range, inclusive of both ends

java.util.Vector<T>	getRecordsBy (java.lang.String name, int max) Returns a vector of maximum max records that matches the name
java.util.Vector<T>	getRecordsByCategory (java.lang.String name) Returns a vector of records that matches the category name
java.util.Vector<T>	getRecordsWithNamePrefix (java.lang.String prefix) Returns a vector of records that has the same prefix

Class Report

A class to hold report information.

Method Summary	
double	getAveExpense () Returns the average expense for the period between start date and end date
double	getBalance () Returns the balance for the period between start date and end date
java.util.Date	getEnd () Returns the end date
java.util.Vector< ReportCategory >	getExpenseCategory () Returns a vector of ReportCategory
double	getExpensePercentage () Returns the percentage of expense
double	getIncomePercentage () Returns the percentage of income
int	getNumRecords () Returns the number of records found
java.util.Date	getStart () Returns the start date
double	getTotalExpense () Returns the total expense for the period between start date and end date
double	getTotalIncome () Returns total income for the period between start date and end date
void	setHeading (double income, double expense, double balance, int numRecords) Sets the 'Heading' of the report, namely total income, total expense, and balance
void	setSectionExpense (java.util.Vector< ReportCategory > expenseCategory) Set the 'Expense' of the report, namely: Expense table, and average expense per day

Class ReportCategory

A data structure representing the rows for the expense table in the report.

Method Summary	
void	calAmtPerFreq() Calculate the amount per frequency ratio
int	compareTo() (ReportCategory anotherCategory) Comparable, for sorting Vector in descending order according to percentage
double	getAmount() Returns the total expense for this category
double	getAmtPerFreq() Returns the amount per frequency ratio for this category
java.lang.String	getCategory() Returns the name of this category
int	getFrequency() Returns the frequency of this category
double	getPercentage() Returns the percentage of expense of this category
void	incrementAmount() (double amount) Increments by the specified amount
void	incrementFreq() Increments the frequency by 1
void	setPercentage() (double percentage) Sets the percentage for expense of this category

Interface ReportGenerator.DataProvider

Method Summary	
Pair <java.util.Vector< ExpenseRecord >, java.util.Vector< IncomeRecord >>	getDataInDateRange() (java.util.Date start, java.util.Date end) Returns a Pair of ExpenseRecord and IncomeRecord vectors within the given start and end date

Class ReportGenerator

This class generates a Report object, given a date range.

Method Summary

Report	generateReport (java.util.Date start, java.util.Date end) Generates a Report object, given a date range
------------------------	--

Interface SearchHandler

This interface denotes the possible searches the user may do. It also assists the UI in disseminating the search request, and returning formatted, relevant search results back to the UI component.

Method Summary	
java.util.Vector< Record >	search (SearchRequest req) Returns a vector of records that satisfies the search request
java.util.Vector< Record >	search (java.lang.String partialMatch) Returns a vector of records that matches the request partially

Class SearchRequest

This is a wrapper class that is used to contain search query parameters.

It can support multiple queries.

To create a new request, construct it with the required parameter, such as `new SearchRequest (nameToSearch)`

Method Summary	
Category	getCategory () Returns a <code>Category</code> of the <code>SearchRequest</code> .
Pair <java.util.Date, java.util.Date>	getDateRange () Returns the date range of this <code>SearchRequest</code> .
java.lang.String	getName () Returns the name of this <code>SearchRequest</code>
SearchRequest.RecordType	getType () Returns the <code>RecordType</code> of this <code>SearchRequest</code>
boolean	isMultiple () Checks if this <code>SearchRequest</code> is asking for multiple fields
boolean	match (Record r) Tests if a record matches the <code>SearchRequest</code>
static Pair <java.util.Date, java.util.Date>	normalizeDateRange (Pair <java.util.Date, java.util.Date> dateRange) Normalizes the date range to keep only the information on day, month and year

void	setCategory (Category category) Sets the category of this <code>SearchRequest</code>
void	setDateRange (Pair < java.util.Date , java.util.Date > dateRange) Set the date range of this <code>SearchRequest</code>
void	setName (java.lang.String name) Sets the name field of this <code>SearchRequest</code>
void	setType (SearchRequest.RecordType type) Sets the <code>RecordType</code> of this <code>SearchRequest</code>
java.lang.String	toString () Returns a string that represents the current object

Enum `SearchRequest.RecordType`

Enum Constant Summary	
BOTH	<code>SearchRequest</code> is a request for both <code>ExpenseRecord</code> and <code>IncomeRecord</code>
EXPENSE	<code>SearchRequest</code> is a request for <code>ExpenseRecord</code>
INCOME	<code>SearchRequest</code> is a request for <code>IncomeRecord</code>

Method Summary	
static SearchRequest.RecordType	valueOf (java.lang.String name) Returns the enum constant of this type with the specified name.
static SearchRequest.RecordType []	values () Returns an array containing the constants of this enum type, in the order they are declared.

Class `Storable`

Implements basic function to inspect whether data is updated.

Method Summary	
void	afterDeserialize () Optional method to populate transient attributes after deserializing the data from json
boolean	isUpdated () Returns <code>true</code> if data has been updated by the user and needs to be stored

void	saved() Informs object that data has been stored in the database
------	---

Class StorageManager

Manages the storage of all persistent data

Method Summary	
void	addEventListener (StorageManager.StorageEventListener listener) Adds an event listener for IO exceptions
DataManager	getDataManager () Returns the <code>DataManager</code> that the application uses
void	read () Deserializes all the data from the <code>json</code> file and starts the timer that triggers the storing of data at each interval

Interface StorageManager.StorageEventListener

An event listener that will be notified when storage manager fails to do file IO

Method Summary	
void	readFail (<code>java.io.IOException e</code>) A read IO failure has occurred with the exception supplied
void	writeFail (<code>java.io.IOException e</code>) A write IO failure has occurred with the exception supplied

Class SummaryDetails

Data structure that contains information on `Summary` for a specific time range

Method Summary	
double	getBalance () Returns the balance value for this <code>SummaryDetails</code>
double	getExpense () Get expense value for this <code>SummaryDetails</code>
double	getIncome () Get income value for this <code>SummaryDetails</code>
SummaryGenerator.SummaryType	getSummaryType () Returns the <code>SummaryType</code> of this <code>SummaryDetails</code>

Interface SummaryGenerator.DataProvider

Method Summary	
double	getDailyExpense() Returns the sum of all the expenses of the day
double	getDailyIncome() Returns the sum of all the incomes of the day
double	getMonthlyExpense() Returns the sum of all the expenses of the this month
double	getMonthlyIncome() Returns the sum of all the incomes of this month
double	getTotalExpense() Returns the sum of all the expenses
double	getTotalIncome() Returns the sum of all the incomes
double	getYearlyExpense() Returns the sum of all the expenses of this year
double	getYearlyIncome() Returns the sum of all the incomes of this year

Class SummaryGenerator

Generates four SummaryDetails objects for the four timeframes: today, this month, this year, all time

Method Summary	
SummaryDetails	getSummaryDetails(SummaryGenerator.SummaryType myType) Returns a SummaryDetails object based on which SummaryType is in the parameter
void	markDataUpdated() Generates four SummaryDetails objects for the four timeframes.

Enum SummaryGenerator.SummaryType

enum class for the 4 different time ranges

Enum Constant Summary	
ALLTIME	All time's summary
MONTH	This month's summary

TODAY	Today's summary
YEAR	This year's summary

Method Summary	
abstract java.lang.String	getName() Returns the string to be displayed on Main Window based on which time range is selected.
abstract SummaryDetails	getSummaryDetails() Returns the enum type's SummaryDetails.
static SummaryGenerator.SummaryType	valueOf(java.lang.String name) Returns the enum constant of this type with the specified name.
static SummaryGenerator.SummaryType[]	values() Returns an array containing the constants of this enum type, in the order they are declared.

Class Target

This class holds the information of the targets that the user has set.

Method Summary	
Target	copy() Returns a copy of the Target.
Category	getCategory() Returns the Category of the Target.
double	getTargetAmt() Returns the target amount that is set by the user.

Interface TargetManager.DataProvider

Method Summary	
Category	getCategory(long id) Returns the Category that has specified id
double	getMonthlyExpense(Category cat) Returns the total monthly expenses of the given Category.

Class TargetManager

This is generator that creates the `Target` object and manager the alert information.

Method Summary	
void	<code>afterDeserialize()</code> Optional method to populate transient attributes after deserializing the data from <code>json</code> .
java.util.Vector< <code>Bar</code> >	<code>getAlerts()</code> Returns a vector of <code>Bar</code> that is classified as alerts.
java.util.Vector< <code>Bar</code> >	<code>getOrderedBar()</code> Generates an ordered vector of <code>Bar</code> in increasing order of its ratio
<code>Target</code>	<code>getTarget(Category cat)</code> Returns the target of this <code>Category</code>
java.util.Vector< <code>Target</code> >	<code>getTargets()</code> Returns a copy of the internal targets
void	<code>markDataUpdated()</code> Marks data as updated
void	<code>removeCategoryTarget(long identifier)</code> Removes the target that has the category id from the <code>TreeMap</code> This is called when a <code>Category</code> is deleted
void	<code>removeTarget(Target target)</code> Removes target from <code>TreeMap</code> .
void	<code>setDataProvider(TargetManager.DataProvider data)</code> Sets the <code>DataProvider</code> for this class.
<code>Target</code>	<code>setTarget(Category cat, double targetAmt)</code> Sets a new <code>Target</code> and returns it.

3) Package `ezxpns.data.records`

Class `Category`

It is an immutable class to represent the category of a `Record`. It is meant to be used in `Record` as a reference

Method Summary	
<code>Category</code>	<code>copy()</code> Returns a copy of this <code>Category</code> .
long	<code>getID()</code> Returns the id of the <code>Category</code> .
java.lang.String	<code>getName()</code> Returns the name of the <code>Category</code> .

java.lang.String	toString() Returns a string that represents the current object.
------------------	--

Interface CategoryHandler<T extends [Record](#)>

This is an interface to handle the categories between the Graphical User Interface and the data storage (upon exit)

Method Summary	
Category	addNewCategory (Category newCat) Creates a new Category. Note: the new category may be copied with a different id.
Category	addNewCategory (java.lang.String name) Creates a new Category with the given name
boolean	addToCategory (java.util.List< T > records, Category cat) Add a list of records to the Category.
boolean	containsCategoryName (java.lang.String name) Checks if the name is in used
java.util.List< Category >	getAllCategories () Returns a list of all user defined categories
Category	getCategory (long id) Returns the Category that has specified id
java.util.Vector< Category >	getCategoryWithNamePrefix (java.lang.String prefix) Returns a vector of records that matches the prefix
java.util.Vector< T >	getRecordsBy (Category category, int max) Returns a vector of maximum max records that is under the specified category
boolean	removeCategory (long identifier) Remove a user defined category that matches the given identifier
Category	updateCategory (long identifier, Category selectedCat) Modifies a Category.
java.lang.String	validateCategoryName (java.lang.String name) Returns an error message if the name is not accepted, or a confirmation message if the name is accepted.

Class ExpenseRecord

The ExpenseRecord is a Record with two additional attributes: ExpenseType{NEED, WANT}

Method Summary	
Record	copy () Returns a copy of ExpenseRecord

ExpenseType	getExpenseType() Returns the <code>ExpenseType</code> of this <code>ExpenseRecord</code>
-----------------------------	---

Class ExpenseRecordManager

This is a special record manager for `ExpenseRecord`. It stores sums of needs and wants.

Method Summary	
double	getLastNeedSum() Get sum of amounts of records in the last month that are under needs
double	getNeedSum() Get sum of amounts of records in this month that are under needs

Enum ExpenseType

Method Summary	
NEED	The expense is a need.
SAVE	The expense is savings.
WANT	The expense is a want.

Method Summary	
abstract java.awt.Color	getBaseColor() Get base color for NWS chart.
abstract java.awt.Color	getExceedColor() Get exceed color for NWS chart.
abstract java.awt.Color	getNormalColor() Get normal color for NWS chart.
static ExpenseType	valueOf(java.lang.String name) Returns the enum constant of this type with the specified name.
static ExpenseType []	values() Returns an array containing the constants of this enum type, in the order they are declared.

Class IncomeRecord

A `Record` object that specifically stores income details

Method Summary	
Record	copy () Returns a copy of <code>IncomeRecord</code> .

Class Record

A container for some basic record attributes

Method Summary	
int	compareTo (Record other) Compares two records.
abstract Record	copy () Returns a copy of this record.
boolean	equals (Record other) A method to check if the other <code>Record</code> supplied is the same as this <code>Record</code> object.
double	getAmount () Returns the amount of the record
Category	getCategory () Returns the category of the record
java.util.Date	getDate () Returns the date of the record.
long	getId () Returns the id of the record.
java.lang.String	getName () Returns the name of the record.
java.lang.String	getRemark () Returns the remarks of the record.
static <T extends Record > double	sumAmount (java.util.List<T> rs) Calculates and returns the sum of all the records.
static double	sumBalance (java.util.List< Record > rs) Calculates and returns the balance. It sums up all the incomes and deducts the total expense from it.

Interface RecordHandler

To handle the records between the Graphical User Interface and the data storage (upon exit)

Method Summary

ExpenseRecord	createRecord (ExpenseRecord newRecord, boolean newCat, boolean newPay) Creates a new expense record along with flags for the new expense category.
IncomeRecord	createRecord (IncomeRecord newRecord, boolean newCat) Creates a new income record along with flags for a new income category.
Record	getRecord (long identifier) Retrieves a specific record that matches the given identifier.
java.util.List< Record >	getRecords (int n) Returns the most recent n records.
ExpenseRecord	lastExpenseRecord (java.lang.String name) Returns the most recent expense record that matches the given name. Returns null if no match is found.
IncomeRecord	lastIncomeRecord (java.lang.String name) Returns the most recent income record that matches the given name. Returns null if no match is found.
boolean	modifyRecord (long id, ExpenseRecord selectedRecord, boolean newCat, boolean newPay) Modifies an ExpenseRecord.
boolean	modifyRecord (long id, IncomeRecord selectedRecord, boolean newCat) Modifies an IncomeRecord.
boolean	removeRecord (long identifier) Removes the record that matches the identifier.

Class RecordManager<T extends [Record](#)>

A generic class to manage records.

Method Summary	
Category	addNewCategory (Category toAdd) Creates a new Category. Note: the new category may be copied with a different id.
Category	addNewCategory (java.lang.String catName) Creates a new Category with the given name.
T	addNewRecord (T toAdd) Adds and returns a new Record.
boolean	addToCategory (java.util.List< T > records, Category cat) Adds a list of Record to the given Category.
void	afterDeserialize () Optional method to populate transient attributes after deserializing the data from json.

boolean	<code>containsCategoryName</code> (java.lang.String name) Checks if the category name is in used.
java.util.List< <code>Category</code> >	<code>getAllCategories</code> () Returns a list of all user defined categories.
double	<code>getAllTimeSum</code> () Returns the sum of all records.
java.util.Vector< <code>Category</code> >	<code>getCategoriesBy</code> (java.lang.String name) Returns a vector of <code>Category</code> that contains the given name.
<code>Category</code>	<code>getCategory</code> (long id) Returns the <code>Category</code> that has specified id.
<code>Category</code>	<code>getCategory</code> (java.lang.Long id) Returns the <code>Category</code> that has specified id.
java.util.Vector< <code>Category</code> >	<code>getCategoryWithNamePrefix</code> (java.lang.String prefix) Returns a vector of records that matches the prefix.
double	<code>getDailySum</code> () Returns the sum of all the records dated on the same day.
double	<code>getLastMonthSum</code> () Returns the sum of all the records belonging to the previous month.
double	<code>getMonthlySum</code> () Returns the sum of all the records belonging to the current month.
double	<code>getMonthlySum</code> (<code>Category</code> cat) Returns the sum of all records belonging to the current month and the specified category
<code>T</code>	<code>getRecordBy</code> (long id) Gets a <code>Record</code> by its id.
java.util.Vector< <code>T</code> >	<code>getRecordsBy</code> (<code>Category</code> category, int max) Returns a vector of maximum max number of <code>Record</code> that belongs to the specified category
java.util.Vector< <code>T</code> >	<code>getRecordsBy</code> (java.util.Date start, java.util.Date end, int max, boolean reverse) Returns a vector of maximum max number of <code>Record</code> within the date range, inclusive of both ends
java.util.Vector< <code>T</code> >	<code>getRecordsBy</code> (java.lang.String name, int max) Returns a vector of maximum max number of <code>Record</code> that has the same name.
java.util.Vector< <code>T</code> >	<code>getRecordsByCategory</code> (java.lang.String name) Returns a vector of <code>Record</code> that has the same name.
java.util.Vector< <code>T</code> >	<code>getRecordsWithNamePrefix</code> (java.lang.String prefix) Returns a vector of <code>Record</code> that matches the prefix.
double	<code>getYearlySum</code> () Returns the sum of amounts of the records belonging to the current year

boolean	removeCategory (Category category) Removes a category
boolean	removeCategory (long identifier) Removes a user defined category based on the given identifier
void	removeRecord (long id) Removes a record
Category	updateCategory (long identifier, Category selectedCat) Modifies a category.
void	updateCategory (long id, java.lang.String newName) Updates a category.
T	updateRecord (long id, T updated) Updates a record.
java.lang.String	validateCategoryName (java.lang.String name) Returns an error message if the name is not accepted, or a confirmation message if the name is accepted.

4) Package ezxpns.GUI

Interface CategoryHandlerInterface

Interface to handle the categories between the Graphical User Interface and the data storage

Method Summary	
Category	addNewCategory (Category newCat) Creates a new Category. Note: the new category may be copied with a different id
Category	addNewCategory (java.lang.String name) Creates a new Category with the given name
java.util.List< Category >	getAllCategories () Returns a list of all user defined categories
boolean	removeCategory (long identifier) Removes a user defined category that matches the given identifier
Category	updateCategory (long identifier, Category selectedCat) Updates a category.

Class CategoryModel

A model used to display list of category in category frame.

Method Summary

java.lang.Object	getElementAt (int arg0) Returns the element at arg0.
int	getSize () Returns the number of categories.
void	update () Refreshes the whole list.

Class Config

Default Configurations for GUI Component

Method Summary	
static java.util.regex.Pattern	ALPHANUMERIC Alphanumeric pattern matcher
static int	DEFAULT_DIALOG_HEIGHT The default height for the dialog height
static int	DEFAULT_DIALOG_WIDTH The default width for the dialog window
static int	DEFAULT_MAX_AMT_PER_RECORD The preset maximum amount for each record
static int	DEFAULT_MAX_LENGTH_DESC The preset maximum name length
static int	DEFAULT_MAX_LENGTH_NAME The preset maximum name length
static int	DEFAULT_MENU_HEIGHT The default height for the menu button
static int	DEFAULT_MENU_WIDTH The default width for the menu button
static double	DEFAULT_MIN_AMT_PER_RECORD The preset minimum amount for each record
static int	DEFAULT_UI_HEIGHT The default height for the GUI window
static int	DEFAULT_UI_WIDTH The default width for the GUI window
static java.awt.Font	MENU_FONT The preset font for the menu buttons
static int	MIN_UI_HEIGHT The minimum height for the GUI window
static int	MIN_UI_WIDTH The minimum width for the GUI window
static java.text.DecimalFormat	MONEY_FORMAT The preset money format for currency: \$###,###,##0.00

static java.awt.Font	TEXT_FONT The preset font for normal text
static java.awt.Font	TITLE_FONT The preset font for the title

Interface ConfigManager

An interface to manage the configurations of the GUI.

Method Summary	
void	load() To load the configuration from the file xyz.ini (example)
void	save() To save the configuration from the file xyz.ini (example)

Class UIControl

This class assists the EzXpns class to manage the GUI Windows

Method Summary	
void	closeHomeScreen() Closes the main/home screen of EzXpns
void	edit() (Record record, RecordListView display) Edits the given Record
UndoManager	getUndoMgr() Returns the UndoManager object that is used by the program.
void	showHomeScreen() Displays the main screen of EzXpns
void	showRecWin() (int recordType) Displays a new RecordHandler window given the recordType
void	showRecWin() (Record record) Displays the record window in edit mode.
void	showReportWin() Displays a ReportHandler Window

Class UINotify

This is an utility to execute *Minor Notification Messages*.

It displays notification messages to users when they perform an action such as adding new data, editing existing data and removing existing data.

Method Summary	
static void	createErrMsg (javax.swing.JComponent parent, java.lang.String msg) Invokes a JDialog to display an error message.

Class UndoManager

A wrapper class that allows one to add and perform undo actions

Method Summary	
	void add (javax.swing.AbstractAction action, java.lang.String name) Adds an undo action to the stack.
javax.swing.AbstractAction	getAction () Returns the undo action at the top of the stack.
	void setPostUndo (javax.swing.AbstractAction action) Sets an undo action.

Interface UpdateNotifyee

This interface otifies the other UI components. It is invoked when the user performs an action that modifies data.

Method Summary	
void	addUndoAction (javax.swing.AbstractAction action, java.lang.String name) Adds an action done by the user into the undo stack
void	updateAll () Invokes upon user action to alert and update all the other components

5) Package ezxpns.util

Class Pair<L,R>

A class to hold pair of any Object .

Method Summary	
boolean	equals (java.lang.Object o) Checks if two Object are equal
	LgetLeft () Gets the left Object .
	RgetRight () Gets the right Object .

int	<code>hashCode()</code> Return the hashcode of an Object.
-----	--