Software Design Document

for

Personal Budget Manager Application

Version 0.2

Prepared by

Name	**Student ID**	**e-mail**
Danny Shash	**29548912**	**danny.shash@gmail.com**
Ganji Amarendher	**25764246**	**a_ganji@encs.concordia.ca**
Siming Huang	**40081588**	**hsim47@163.com**
Tony Lac	**40049123**	**tony.lac@polymtl.ca**
Xinjie Zeng	**27238223**	**xinjiezeng@gmail.com**

24/3/2019

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

1.1 Document Purpose

The purpose of this document is to describe the detailed structure of the components of the Personal Budget Manager Application and the precise implementation details to satisfy the requirements in the Software Requirements Document. This includes the Architectural Design (AD), Interface Specification (IS) and Detailed Class Design (DD) of the application. The AD part focuses on subsystem and module diagram, the IS part focuses on describing services provided by modules, and the DD part focuses on data description and declaration.

1.2 Product Scope

The development plan is divided into 3 iterations and the document will be updated accordingly to address the content in that phase. This iteration of the Personal Budget Manager extends the previous iteration by adding the following features: The expense types (purchase, bill) instead of being atomic, they can be composite of sub-items expenses, which in turn may be complex expenses. Sub-items expenses are dependent on their super-expenses. If a super-item expense is removed from the expense list all its sub-items expense are also removed. Similarly, if the payment status of a super-item is set to complete or incomplete, the completion status of all sub-items will change accordingly.

In this increment implement the following:
– View expenses with their hierarchy in case of composite expenses.
– Delete expense.
– Set expense to paid/unpaid.
– Add expense: purchase or bill as single/periodic/composite.
– Hiding / unhiding paid expenses: where the user has the option to change the current view and to hide/unhide paid expenses (at the root level).

1.3 Definitions and Acronyms

1.3.1 Definitions

Purchase	A type of day-to-day expense
Bill	A type of recurring expense
Composite	A composite of sub-item expenses

1.3.2 Acronyms

PBM	Personal Budget Manager Application
SRS	Software Requirement Specification
SDD	Software Design Document

2. Architecture Design

2.1 Rationale

The architecture chosen for the PBM is the Model View Controller model (MVC). The MVC architecture is made up of 3 separate components. There are one model, several views and controllers. This architecture allows the three components to be developed separately from one another and they can be done in parallel. Either of the components can be updated without affecting the other components as long as their application program interface remains the same. With a common API, the 3 models can be seamlessly integrated into one App.

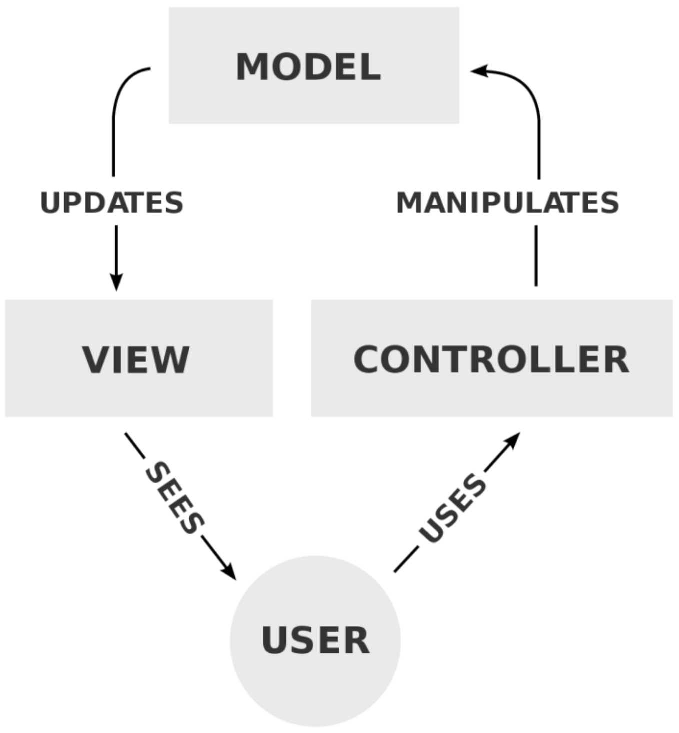


Figure 1. MVC (a)

2.2 Software Architecture Diagram

The model is the core of the PBM where the functionality and data are stored and manipulated. All of the computations that are performed are done in this component as well as all data that needs to be processed. Moreover, whenever there are changes made to the model, the model updates the view.

The view is the graphical user interface (GUI) of the PBM and displays the data from the model. Whenever the model changes, the view responds to those changes by updating itself. The view also gets updated by the controller, as the controller performs simple data validations on user input and updates the view. Different views can be developed in order to present different interfaces.

The controller is what the players use to interact with the PBM. All information is entered via the controller, which it then passes on to the model.

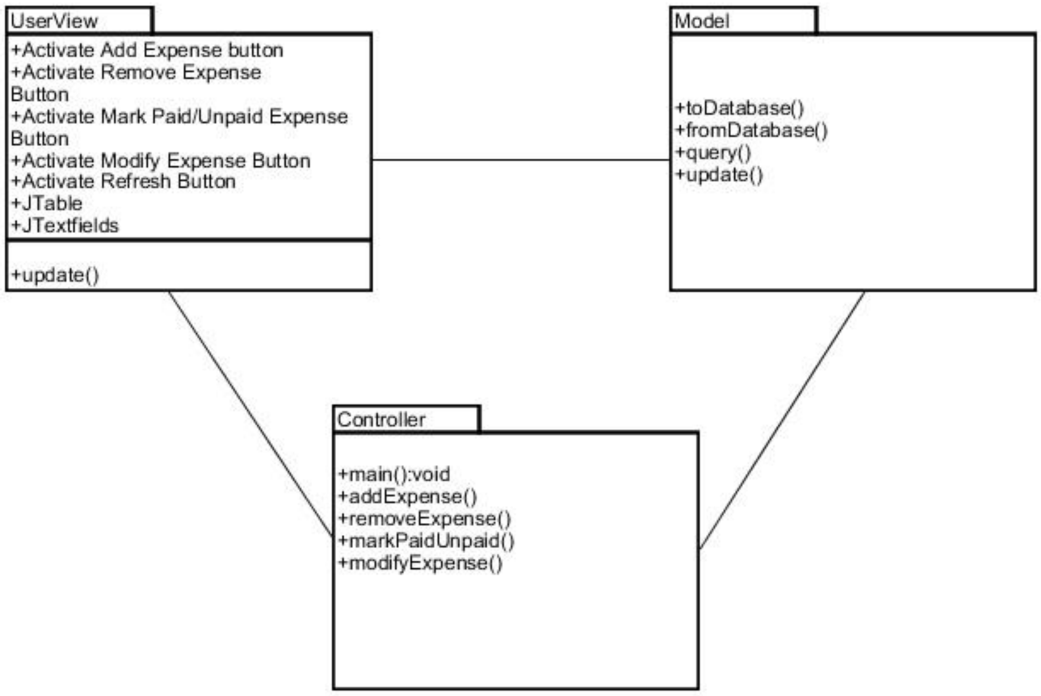


Figure 2.MVC (b)

2.3 System Topology

The PBM is to be developed for a standalone environment. All three components of the MVC model will be integrated into one executable. Moreover, the PBM does not require any third party software to run, or an Internet connection.

3. Software Interface Design

3.1 System Interface Diagrams

The only system level interface in the PBM is the user interface, as the game does not employ any software or hardware interfaces. The user interface is GUI, which allows users to interact with the application.

3.1.1 Use Case Scenario

Use Case 1: Launch the application

Description: User able start the application.

Actor(s): User

Goal: User able to launch the PBM application see the existing Purchase items in the system.

Pre-conditions: The project from the github is cloned on the local PC and opened the project called Project1.

Main scenario:

1.User selects UserInterface class from Eclipse project Project1, right click on it to select run as application option

Alternative scenarios:

1.User changes the expense type by selecting an expense type from the drop-down list.

2.The corresponding expense items are displayed on the main window.

Post-conditions: The list of all the expenses for the selected expense type are displayed on the main window.

Note: All the Purchases and Composite Purchases are displayed by selecting either of corresponding Expense Type. Similarly, all the Bills and Composite Bills are displayed by selecting either of corresponding Expense Type.

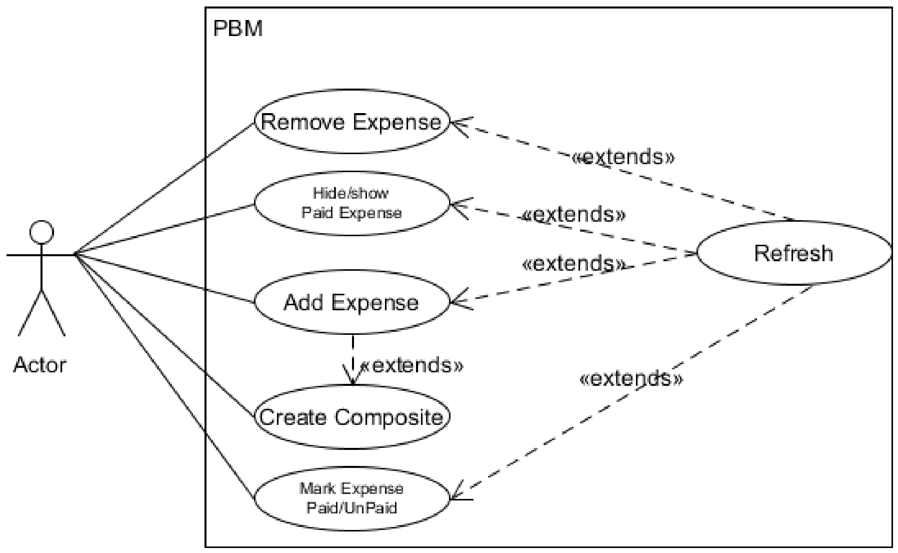


Figure 3. UCS - Launch the application

Use Case 2-1: Add a simple expense

Description: User adds a new expense

Actor(s): User

Goal: User able to add 2 types of simple expenses, Purchase and Bill

Pre-conditions: User already started PBM application.

Main scenario:

1. User chooses to add a new expense
2. System opens a new add expense panel where the user can enter expense details.
3. User selects the simple expense type, Purchase or Bill
4. User enters the details for the selected expense types
5. System validates the data
6. System add the expense into the corresponding expense list
7. System refreshes the expense list in the main window with the new expense added.

Alternative scenarios:

1. The information is incorrect (amount entered is negative)

1.1 User will receive a prompt indicating which info is incorrect

1.2 After correcting the info, it will be added to the expense list.

Post-conditions: The list of all the expenses displayed in the main window for the selected expense list.

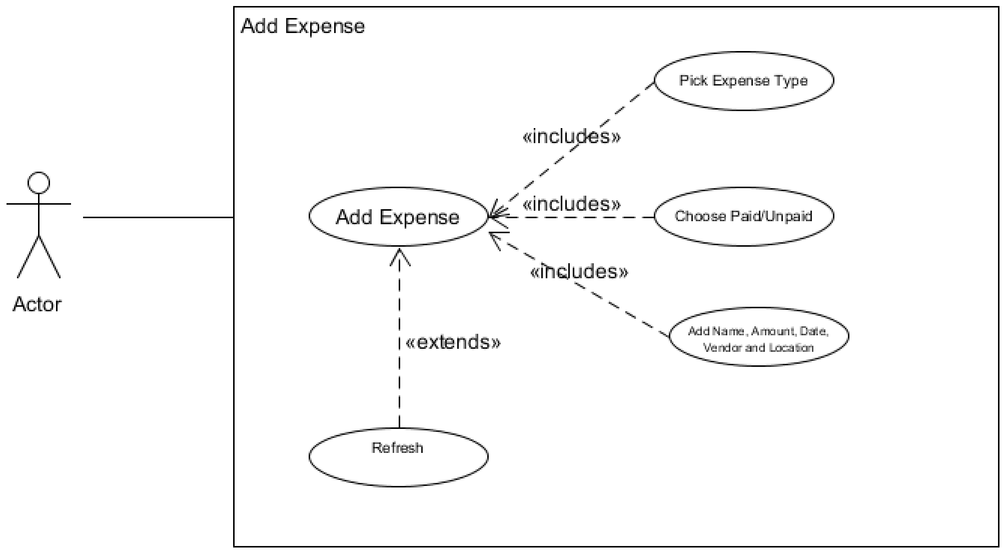


Figure 4. USC - Add an expense

Use Case 2-2: Add a composite expense

Description: User adds a new composite expense

Actor(s): User

Goal: User able to add 2 types of composite expenses, Composite Purchase and Composite Bill

Pre-conditions: User already started PBM application.

Main scenario:

1. User selects the expenses to be grouped as composite.
2. User chooses to create a new composite expense by selecting "Composite Expense" button.
3. System opens a new add expense panel where the user can enter composite expense details.
4. User selects the composite expense type, Purchase or Bill
5. User enters the details for the selected composite expense type
6. System validates the data
7. System add the expense into the corresponding expense list
8. System refreshes the expense list in the main window with the new expense added.

Alternative scenarios:

1. The information is incorrect (amount entered is negative)

1.1 User will receive a prompt indicating which info is incorrect

1.2 After correcting the info, it will be added to the expense list.

Post-conditions: The list of all the expenses displayed in the main window for the selected expense list.

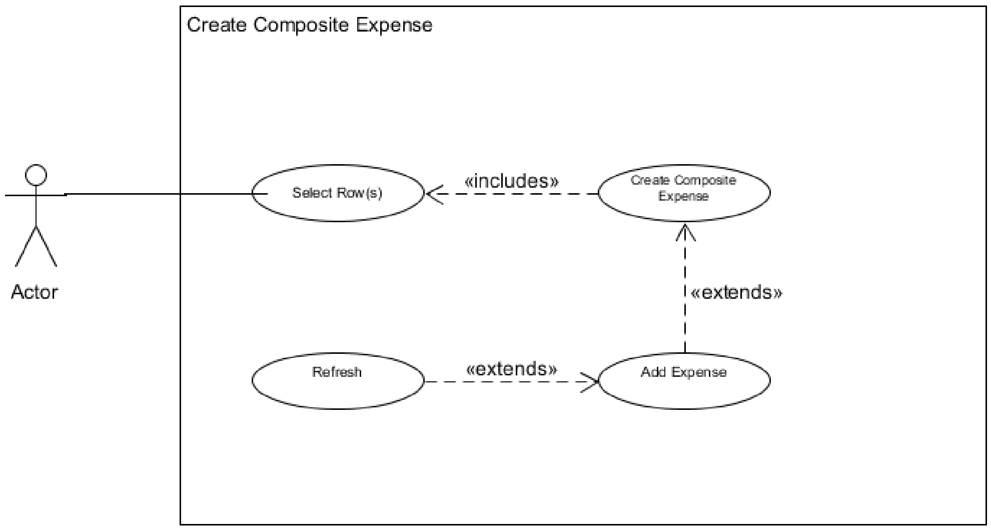


Figure 5. UCS - Add a composite expense

Use Case 3: Modify an existing expense

Description: User can modify an existing expense

Actor(s): User

Goal: user is able to modify an expense type belongs to a certain period

Pre-conditions: User already started PBM application.

Main scenario:

1. User selects a row from the list and modifies allowed details, like paid, amount, any additional details, etc...
2. System modifies the expense entry
3. System refreshes the main window with existing expense list

Alternative scenarios:

1. User changes the expense type for the displayed period

1.1 System fetched the expense list for the period chosen by the user

1.2 System displays the expense list in the main window

Same steps as in the main flow.

2. User changes the time period for the displayed expense type

2.1 System fetched the expense list for the period chosen by the user

2.2 System displays the expense list in the main window

Same steps as in the main flow.

3. User changes the expense type and the time period

3.1 System fetched the expense list for the period chosen by the user

3.2 System displays the expense list in the main window

Same steps as in the main flow.

Post-conditions: The list of all the expenses displayed on the main window for the selected time period.

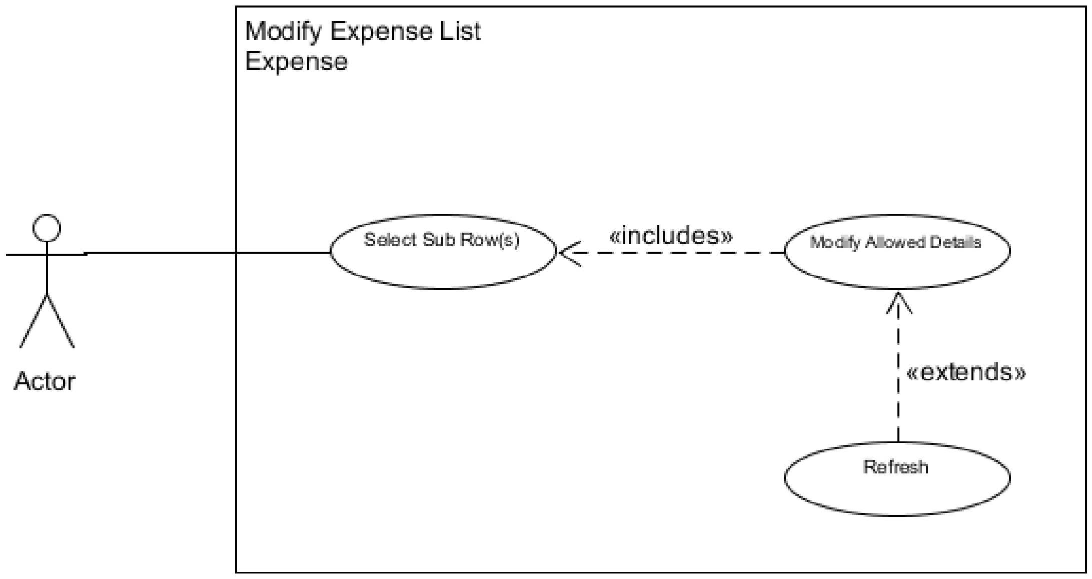


Figure 6. UCS - Modify an existing expense

Use Case 4: Mark an expense as paid or not paid

Description: User Mark an expense as paid or not paid

Actor(s): User

Goal: User able to Mark an expense as paid or not paid

Pre-conditions: User already started PBM application.

Main scenario:

1. User selects an expense from the list of expenses
2. User selects the ‘Mark expense paid/unpaid’ button
3. The status is changed to Paid if the status was Unpaid, or Unpaid for Paid.
4. The information will be sent and stored in the expense list

Alternative scenarios:

None

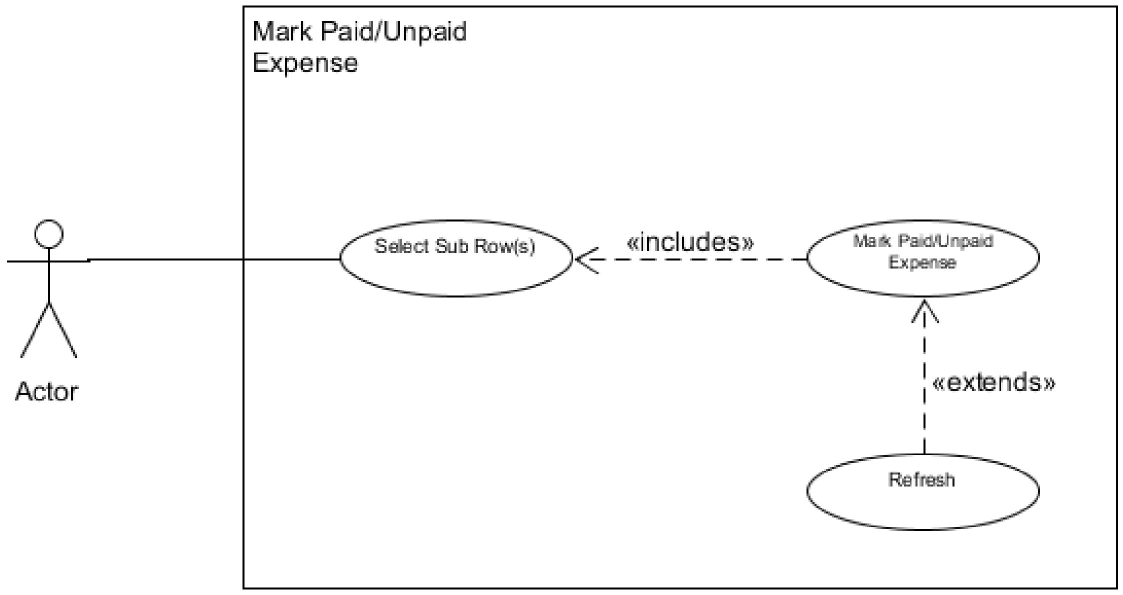


Figure 7. UCS - Mark expense as paid/unpaid

Use Case 5: Hide/show paid expenses

Description: User hides/show paid expense from the list

Actor(s): User

Goal: User able to hide/show paid expense from the list

Pre-conditions: User already started PBM application.

Main scenario:

1. User selects an expense from the list of expenses
2. User selects the ‘hide/show paid expenses’ button
3. The paid expense is hidden from view, or a hidden paid expense gets unhidden.

Alternative scenarios:

None

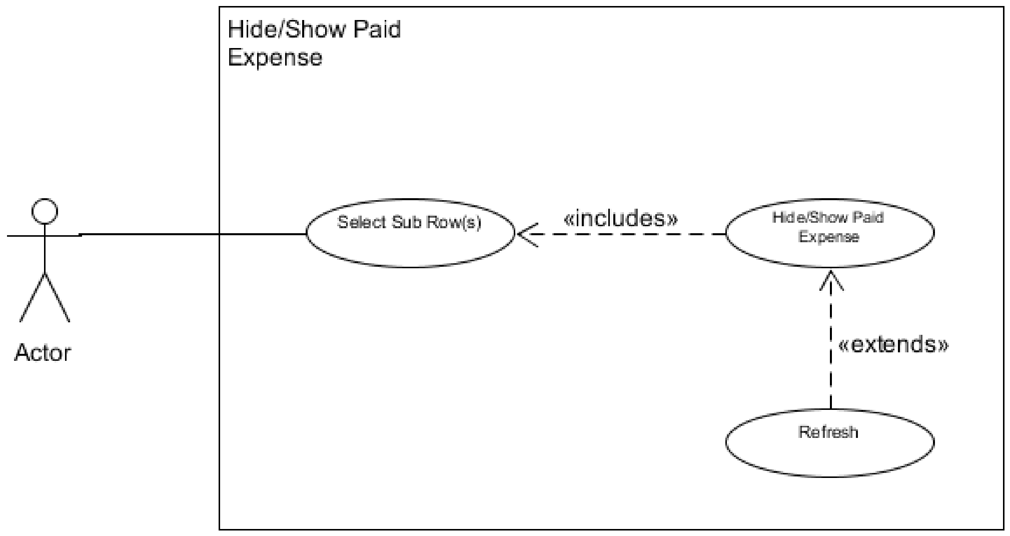


Figure 8. UCS - Hide/show paid expenses

Use Case 6: Remove an expense

Description: User chooses to remove an expense

Actor(s): User

Goal: User able to remove selected expense from the expense list

Pre-conditions: User already started PBM application.

Main scenario:

1. User selects a row in the list to be removed
2. User selects the ‘Remove expense’ button
3. System refreshes the main window with existing expense list

Alternative scenarios:

None

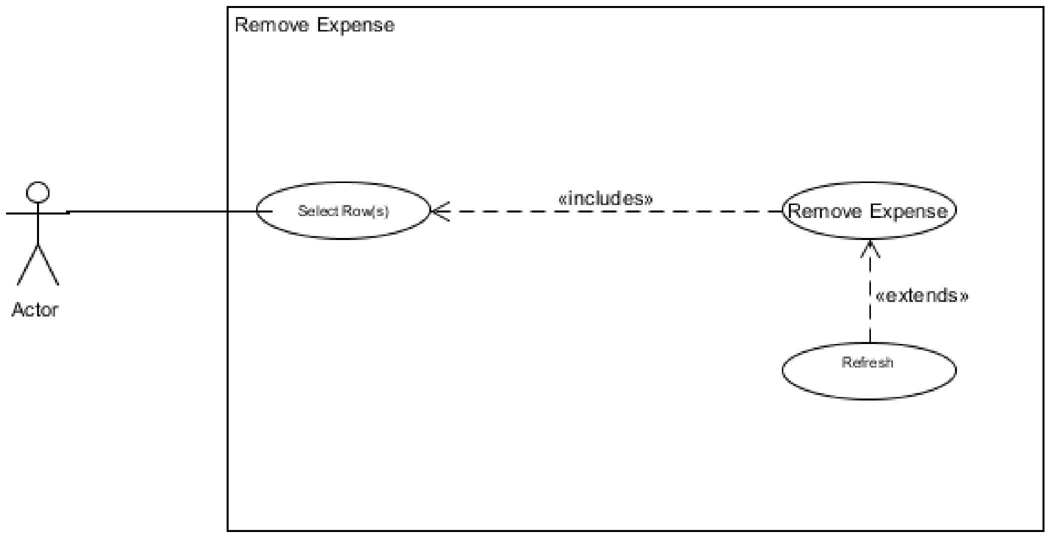
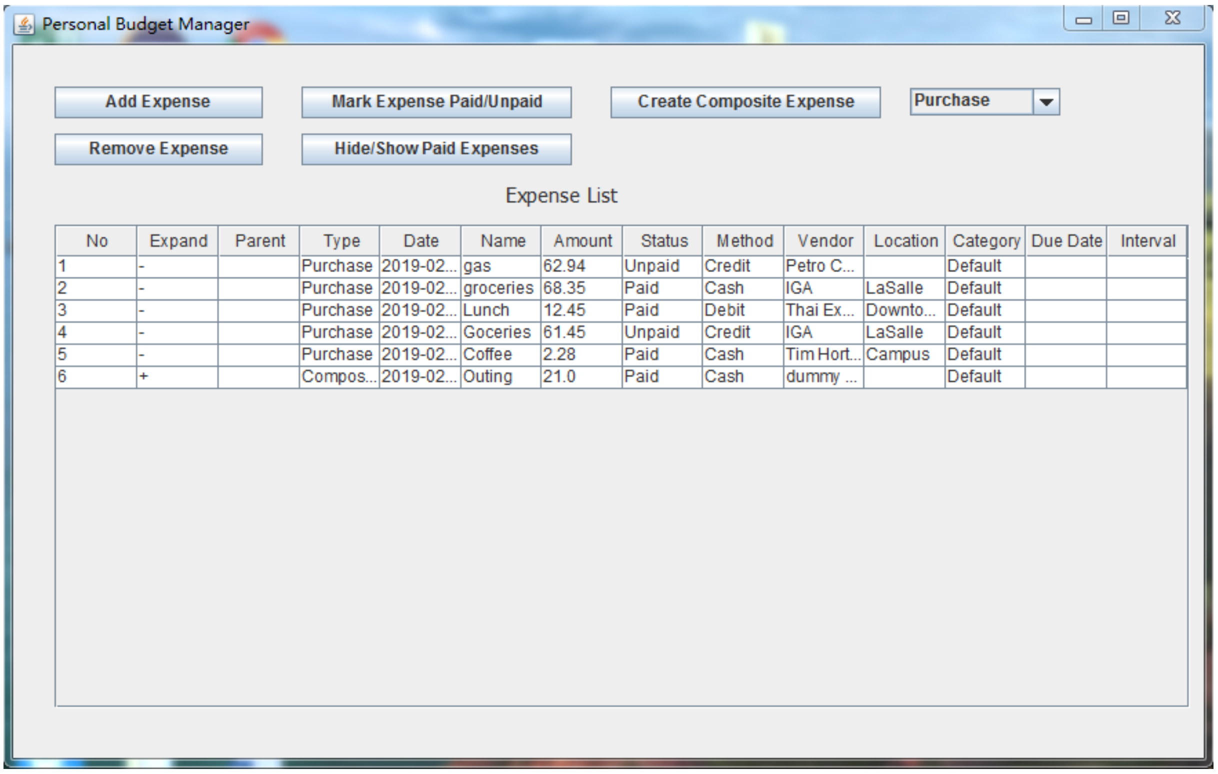


Figure 9. UCS - Remove an expense

3.1.2 User Interface

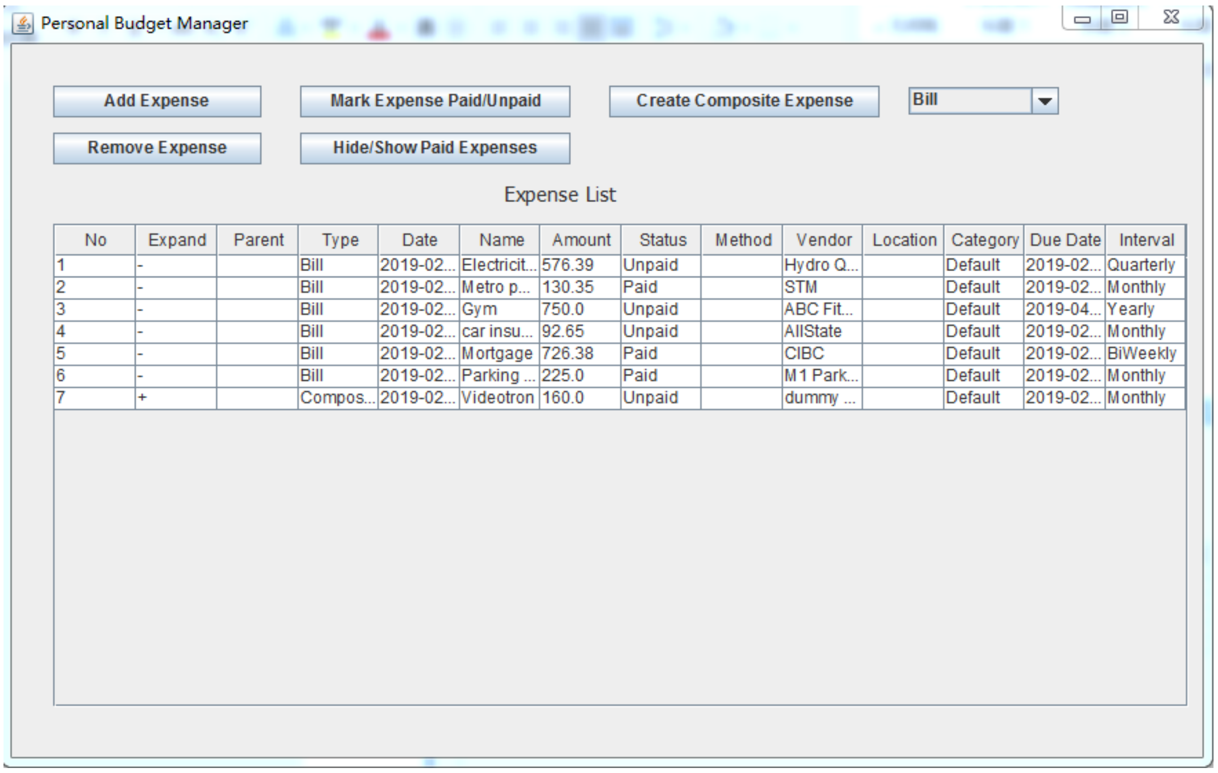
Main Panel

This is the main panel of the PBM. There is an expense list, a dropdown panel: with purchase and bill two types and five buttons: add expense, remove expense, mark expense paid/ unpaid, hide/show paid expenses and create composite expense. With different expense types, they may input different details.



No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Purchase	2019-02...	gas	62.94	Unpaid	Credit	Petro C...		Default		
2	-		Purchase	2019-02...	groceries	68.35	Paid	Cash	IGA	LaSalle	Default		
3	-		Purchase	2019-02...	Lunch	12.45	Paid	Debit	Thai Ex...	Downto...	Default		
4	-		Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		
5	-		Purchase	2019-02...	Coffee	2.28	Paid	Cash	Tim Hort...	Campus	Default		
6	+		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		

Figure 10. User Interface - Purchase list

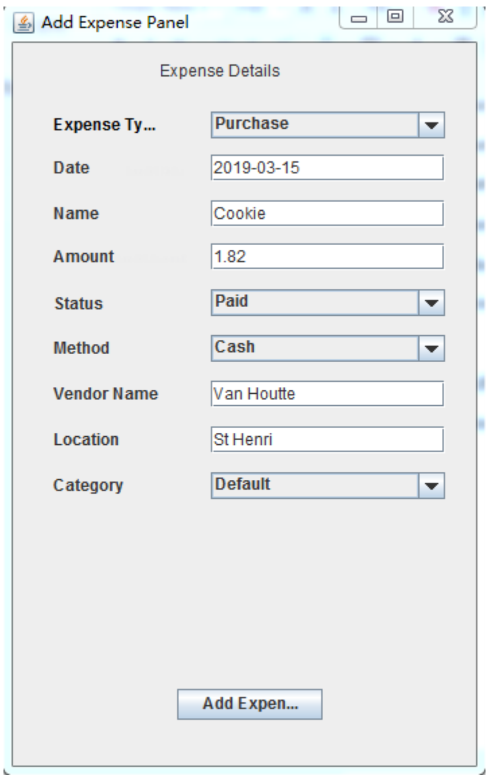


No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Bill	2019-02...	Electricit...	576.39	Unpaid		Hydro Q...		Default	2019-02...	Quarterly
2	-		Bill	2019-02...	Metro p...	130.35	Paid		STM		Default	2019-02...	Monthly
3	-		Bill	2019-02...	Gym	750.0	Unpaid		ABC Fit...		Default	2019-04...	Yearly
4	-		Bill	2019-02...	car insu...	92.65	Unpaid		AllState		Default	2019-02...	Monthly
5	-		Bill	2019-02...	Mortgage	726.38	Paid		CIBC		Default	2019-02...	BiWeekly
6	-		Bill	2019-02...	Parking ...	225.0	Paid		M1 Park...		Default	2019-02...	Monthly
7	+		Compos...	2019-02...	Videotron	160.0	Unpaid		dummy ...		Default	2019-02...	Monthly

Figure 11. User Interface - Bill list

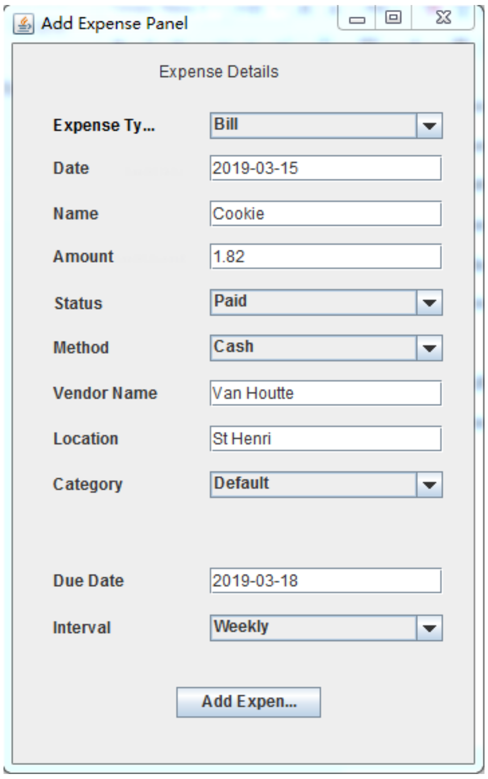
Add expense

When users click the button 'add expense', another interface will pop up. Here they can input description of their expenses. There are four expense types but users can only choose purchase or bill. If users choose purchase expense, there are four drop down buttons where users can change paid status: paid or unpaid, paid method: cash, debt or credit and expense category: default, food, dining, grocery, gas, parking, transportation, taxi, mortgage, saving, insurance, utilities, wireless, health, cable, internet, automobile and entertainment. If users choose bill expense, there is one more button where they can set interval due date: weekly, biweekly, monthly, quarterly or yearly.



Expense Ty...	Purchase
Date	2019-03-15
Name	Cookie
Amount	1.82
Status	Paid
Method	Cash
Vendor Name	Van Houtte
Location	St Henri
Category	Default

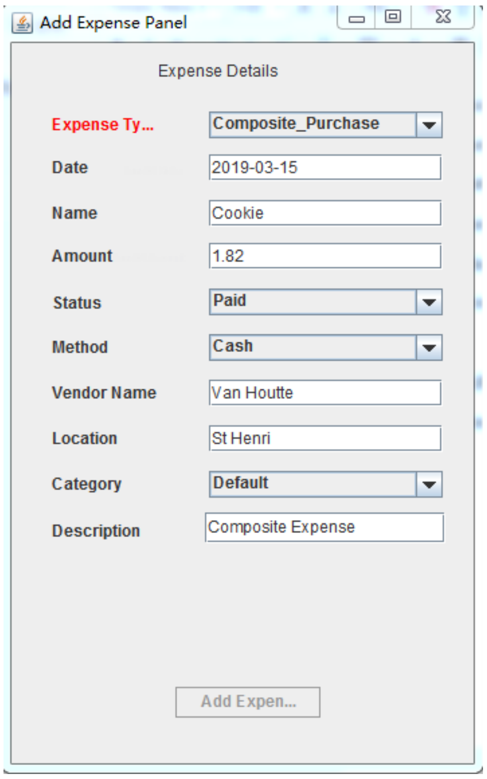
Figure 12. User Interface - Add Expense (a)


Add Expense Panel

Expense Details

Expense Ty...	Bill
Date	2019-03-15
Name	Cookie
Amount	1.82
Status	Paid
Method	Cash
Vendor Name	Van Houtte
Location	St Henri
Category	Default
Due Date	2019-03-18
Interval	Weekly

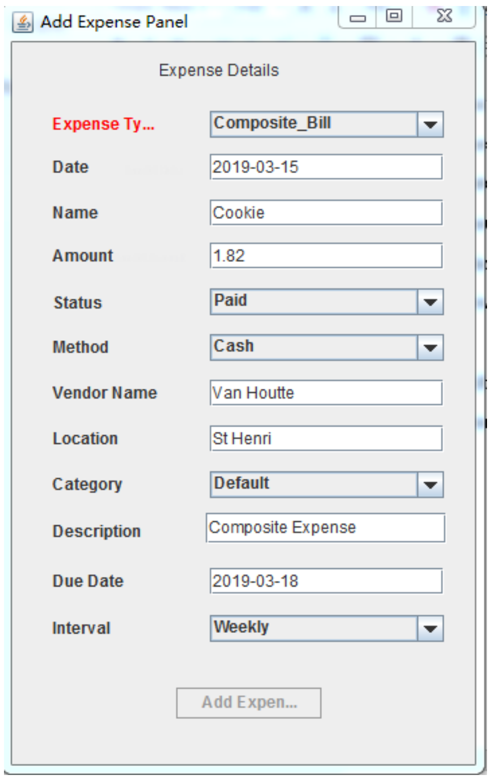
Figure 13. User Interface - Add Expense (b)


Add Expense Panel

Expense Details

Expense Ty...	Composite_Purchase
Date	2019-03-15
Name	Cookie
Amount	1.82
Status	Paid
Method	Cash
Vendor Name	Van Houtte
Location	St Henri
Category	Default
Description	Composite Expense

Figure 14. User Interface - Composite expense(a)


Add Expense Panel

Expense Details

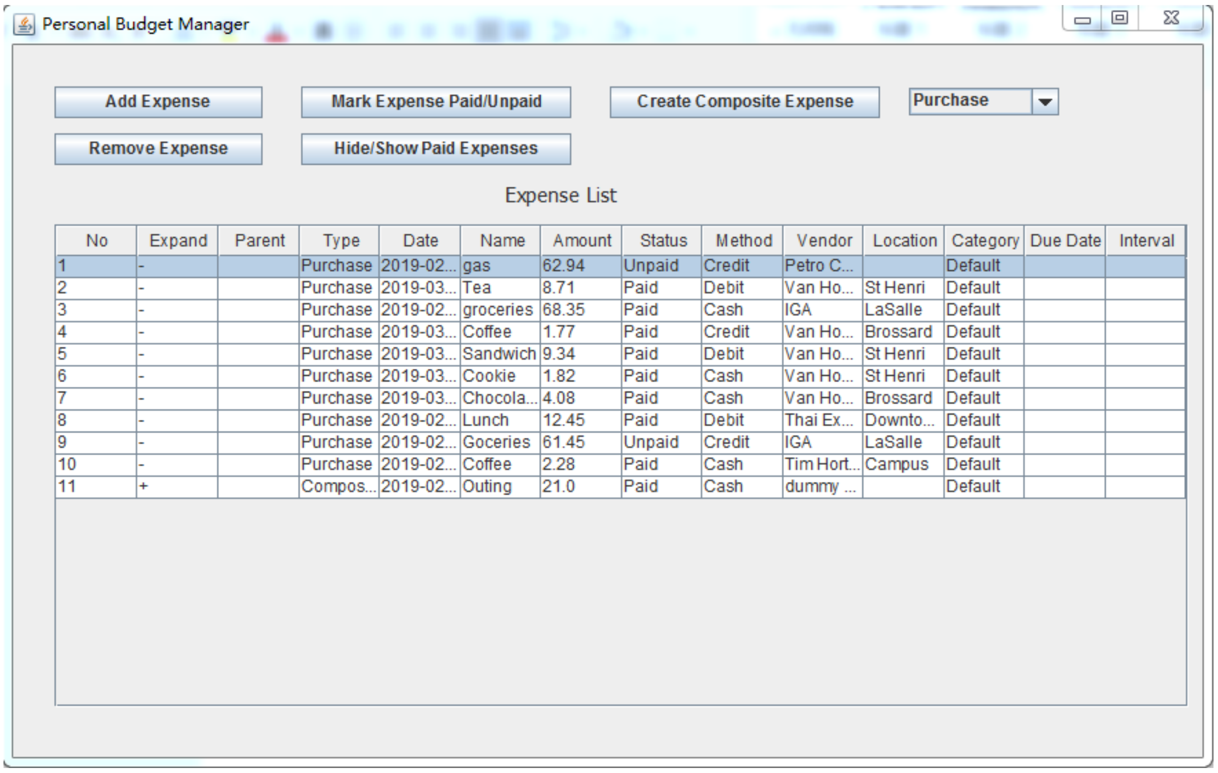
Expense Ty...	Composite_Bill
Date	2019-03-15
Name	Cookie
Amount	1.82
Status	Paid
Method	Cash
Vendor Name	Van Houtte
Location	St Henri
Category	Default
Description	Composite Expense
Due Date	2019-03-18
Interval	Weekly

Figure 15. User Interface - Composite expense (b)

After completing all the information, click the button 'add expense' then the data will be stored in database and added on the list.

Remove expense

If users want to delete a normal expense or composite expense, just chose a line, click the button 'Remove expense', then the data will be deleted from the list and database.

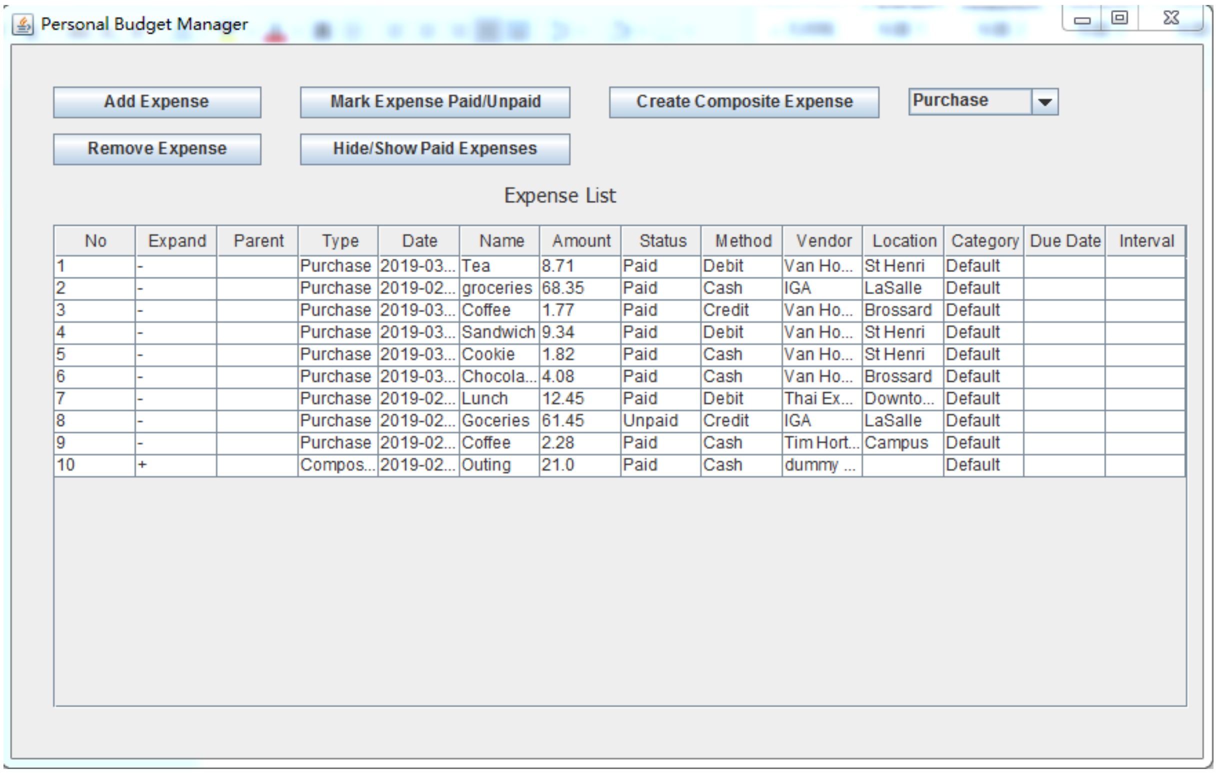


Personal Budget Manager

Expense List

No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Purchase	2019-02...	gas	62.94	Unpaid	Credit	Petro C...		Default		
2	-		Purchase	2019-03...	Tea	8.71	Paid	Debit	Van Ho...	St Henri	Default		
3	-		Purchase	2019-02...	groceries	68.35	Paid	Cash	IGA	LaSalle	Default		
4	-		Purchase	2019-03...	Coffee	1.77	Paid	Credit	Van Ho...	Brossard	Default		
5	-		Purchase	2019-03...	Sandwich	9.34	Paid	Debit	Van Ho...	St Henri	Default		
6	-		Purchase	2019-03...	Cookie	1.82	Paid	Cash	Van Ho...	St Henri	Default		
7	-		Purchase	2019-03...	Chocola...	4.08	Paid	Cash	Van Ho...	Brossard	Default		
8	-		Purchase	2019-02...	Lunch	12.45	Paid	Debit	Thai Ex...	Downto...	Default		
9	-		Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		
10	-		Purchase	2019-02...	Coffee	2.28	Paid	Cash	Tim Hort...	Campus	Default		
11	+		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		

Figure 16. User Interface - Remove expense (a)

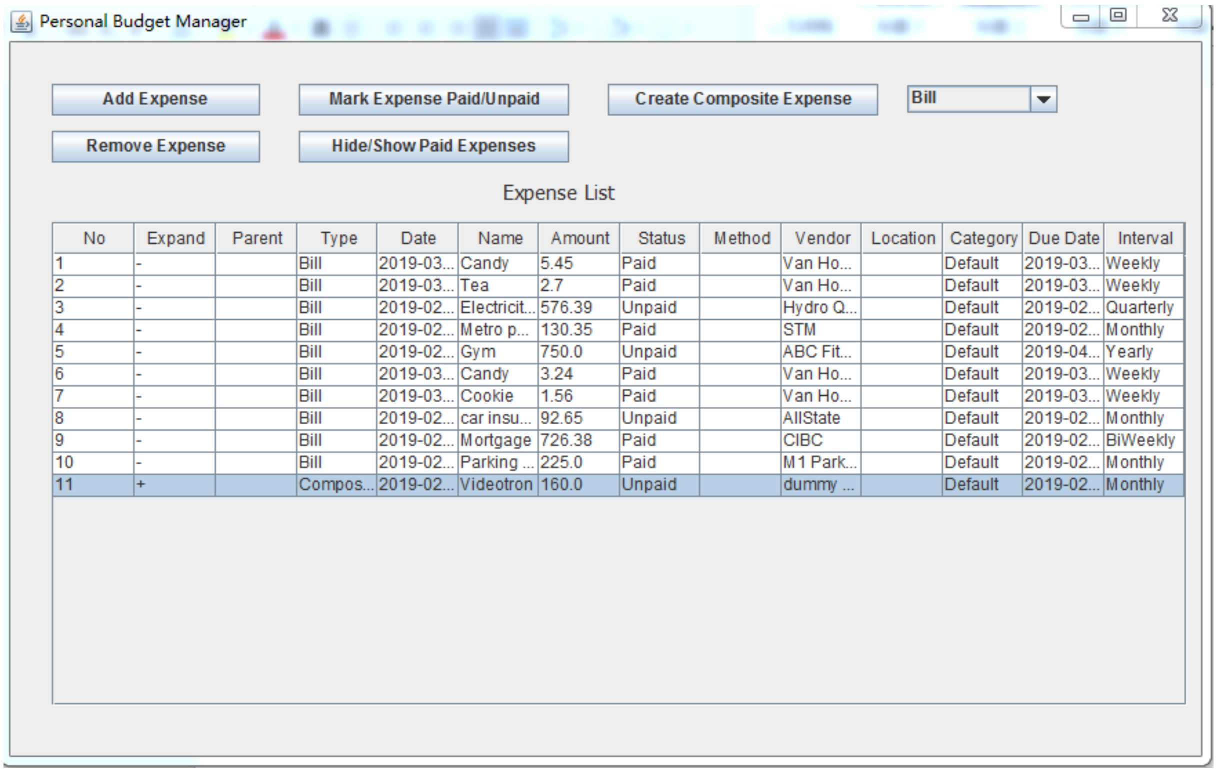


Personal Budget Manager

Expense List

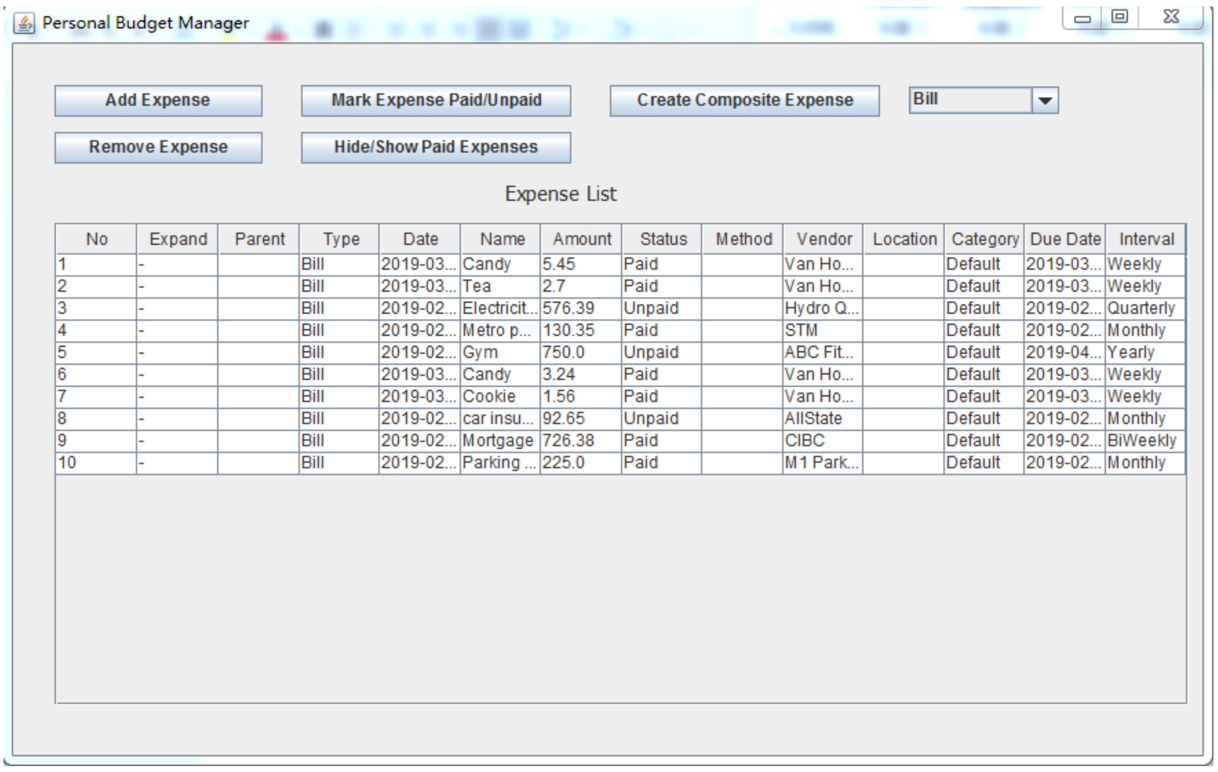
No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Purchase	2019-03...	Tea	8.71	Paid	Debit	Van Ho...	St Henri	Default		
2	-		Purchase	2019-02...	groceries	68.35	Paid	Cash	IGA	LaSalle	Default		
3	-		Purchase	2019-03...	Coffee	1.77	Paid	Credit	Van Ho...	Brossard	Default		
4	-		Purchase	2019-03...	Sandwich	9.34	Paid	Debit	Van Ho...	St Henri	Default		
5	-		Purchase	2019-03...	Cookie	1.82	Paid	Cash	Van Ho...	St Henri	Default		
6	-		Purchase	2019-03...	Chocola...	4.08	Paid	Cash	Van Ho...	Brossard	Default		
7	-		Purchase	2019-02...	Lunch	12.45	Paid	Debit	Thai Ex...	Downto...	Default		
8	-		Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		
9	-		Purchase	2019-02...	Coffee	2.28	Paid	Cash	Tim Hort...	Campus	Default		
10	+		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		

Figure 17. User Interface - Remove expense (b)



No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Bill	2019-03...	Candy	5.45	Paid		Van Ho...		Default	2019-03...	Weekly
2	-		Bill	2019-03...	Tea	2.7	Paid		Van Ho...		Default	2019-03...	Weekly
3	-		Bill	2019-02...	Electricit...	576.39	Unpaid		Hydro Q...		Default	2019-02...	Quarterly
4	-		Bill	2019-02...	Metro p...	130.35	Paid		STM		Default	2019-02...	Monthly
5	-		Bill	2019-02...	Gym	750.0	Unpaid		ABC Fit...		Default	2019-04...	Yearly
6	-		Bill	2019-03...	Candy	3.24	Paid		Van Ho...		Default	2019-03...	Weekly
7	-		Bill	2019-03...	Cookie	1.56	Paid		Van Ho...		Default	2019-03...	Weekly
8	-		Bill	2019-02...	car insu...	92.65	Unpaid		AllState		Default	2019-02...	Monthly
9	-		Bill	2019-02...	Mortgage	726.38	Paid		CIBC		Default	2019-02...	BiWeekly
10	-		Bill	2019-02...	Parking ...	225.0	Paid		M1 Park...		Default	2019-02...	Monthly
11	+		Compos...	2019-02...	Videotron	160.0	Unpaid		dummy ...		Default	2019-02...	Monthly

Figure 18. User Interface - Remove composite expense (a)

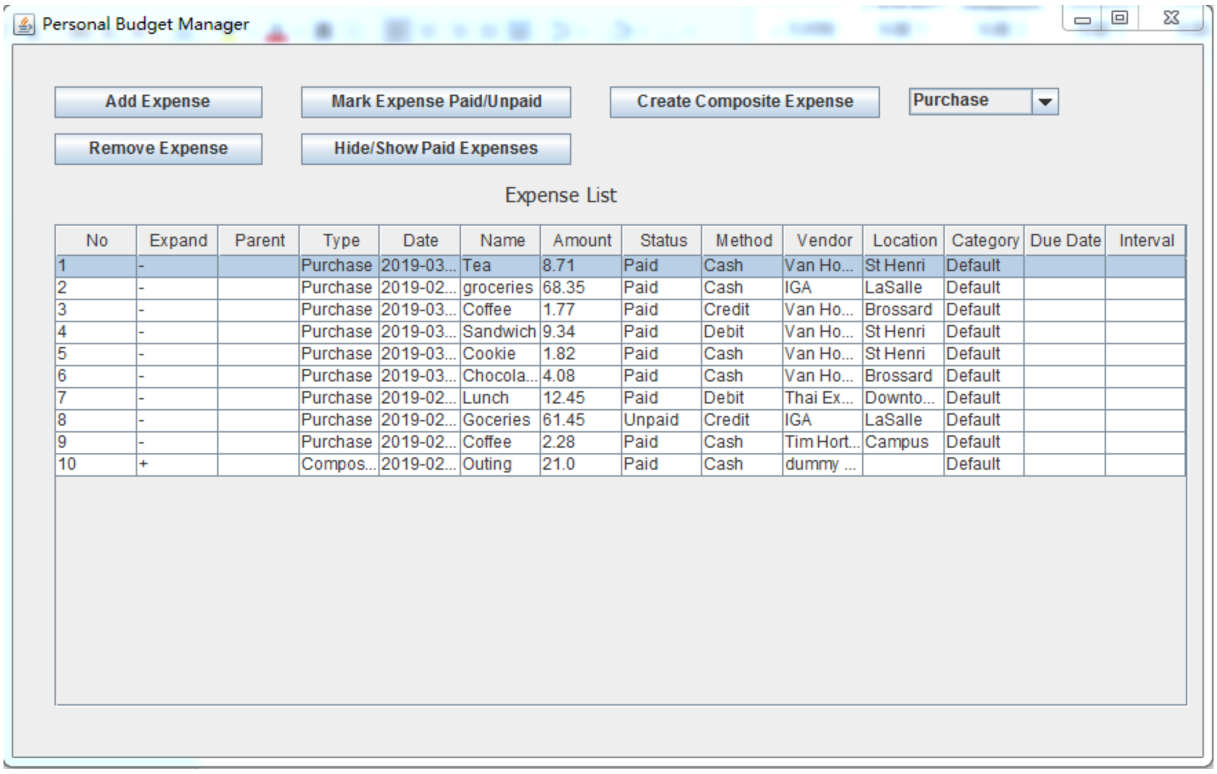


No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Bill	2019-03...	Candy	5.45	Paid		Van Ho...		Default	2019-03...	Weekly
2	-		Bill	2019-03...	Tea	2.7	Paid		Van Ho...		Default	2019-03...	Weekly
3	-		Bill	2019-02...	Electricit...	576.39	Unpaid		Hydro Q...		Default	2019-02...	Quarterly
4	-		Bill	2019-02...	Metro p...	130.35	Paid		STM		Default	2019-02...	Monthly
5	-		Bill	2019-02...	Gym	750.0	Unpaid		ABC Fit...		Default	2019-04...	Yearly
6	-		Bill	2019-03...	Candy	3.24	Paid		Van Ho...		Default	2019-03...	Weekly
7	-		Bill	2019-03...	Cookie	1.56	Paid		Van Ho...		Default	2019-03...	Weekly
8	-		Bill	2019-02...	car insu...	92.65	Unpaid		AllState		Default	2019-02...	Monthly
9	-		Bill	2019-02...	Mortgage	726.38	Paid		CIBC		Default	2019-02...	BiWeekly
10	-		Bill	2019-02...	Parking ...	225.0	Paid		M1 Park...		Default	2019-02...	Monthly

Figure 19. User Interface - Remove composite expense (b)

Mark expense paid/unpaid

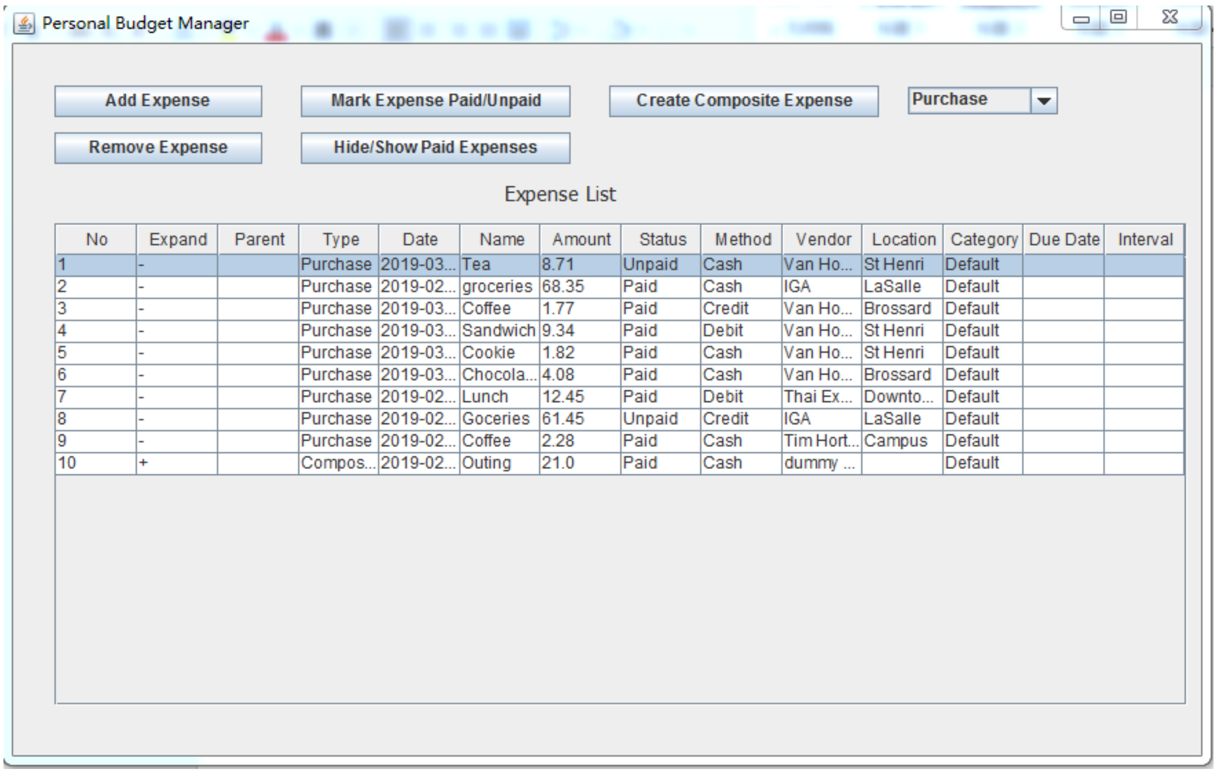
If users want to change an expense status, just chose a line, click the button 'Mark expense paid/unpaid', then the change will be stored in the database.



Expense List

No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Purchase	2019-03...	Tea	8.71	Paid	Cash	Van Ho...	St Henri	Default		
2	-		Purchase	2019-02...	groceries	68.35	Paid	Cash	IGA	LaSalle	Default		
3	-		Purchase	2019-03...	Coffee	1.77	Paid	Credit	Van Ho...	Brossard	Default		
4	-		Purchase	2019-03...	Sandwich	9.34	Paid	Debit	Van Ho...	St Henri	Default		
5	-		Purchase	2019-03...	Cookie	1.82	Paid	Cash	Van Ho...	St Henri	Default		
6	-		Purchase	2019-03...	Chocola...	4.08	Paid	Cash	Van Ho...	Brossard	Default		
7	-		Purchase	2019-02...	Lunch	12.45	Paid	Debit	Thai Ex...	Downto...	Default		
8	-		Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		
9	-		Purchase	2019-02...	Coffee	2.28	Paid	Cash	Tim Hort...	Campus	Default		
10	+		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		

Figure 20. User Interface - Mark expense paid/unpaid (a)



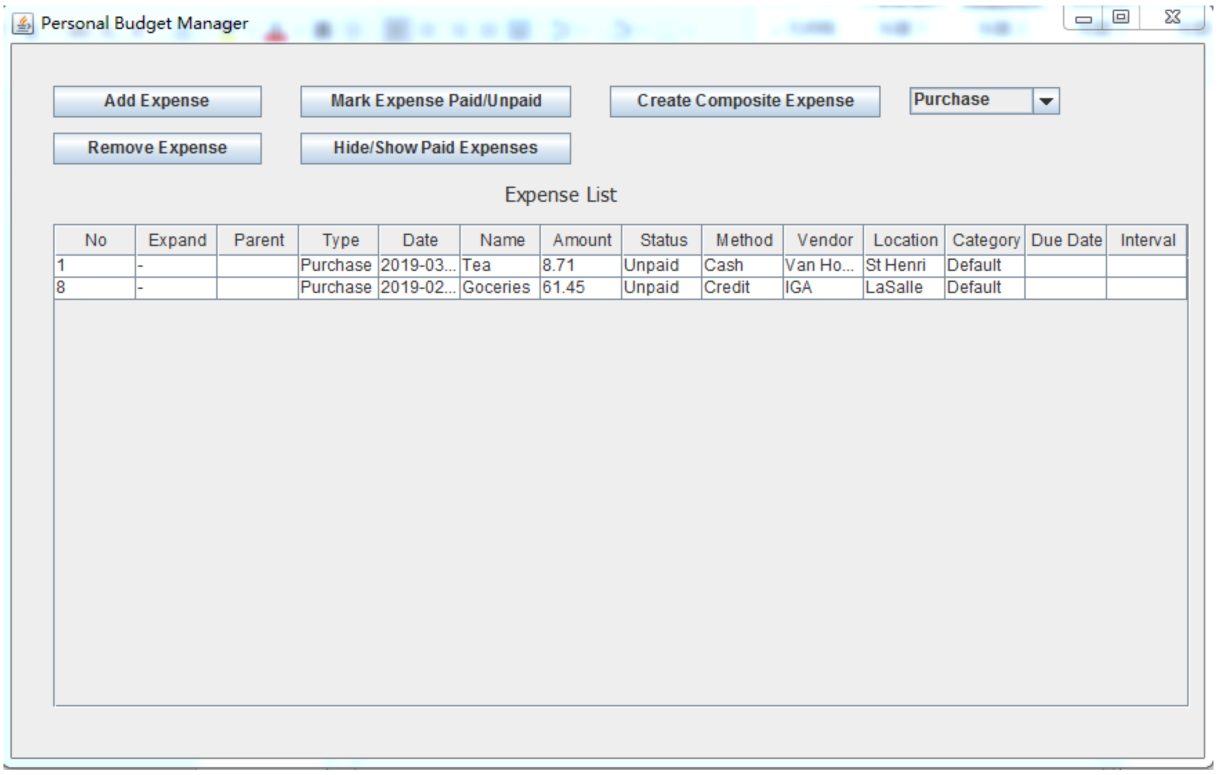
Expense List

No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Purchase	2019-03...	Tea	8.71	Unpaid	Cash	Van Ho...	St Henri	Default		
2	-		Purchase	2019-02...	groceries	68.35	Paid	Cash	IGA	LaSalle	Default		
3	-		Purchase	2019-03...	Coffee	1.77	Paid	Credit	Van Ho...	Brossard	Default		
4	-		Purchase	2019-03...	Sandwich	9.34	Paid	Debit	Van Ho...	St Henri	Default		
5	-		Purchase	2019-03...	Cookie	1.82	Paid	Cash	Van Ho...	St Henri	Default		
6	-		Purchase	2019-03...	Chocola...	4.08	Paid	Cash	Van Ho...	Brossard	Default		
7	-		Purchase	2019-02...	Lunch	12.45	Paid	Debit	Thai Ex...	Downto...	Default		
8	-		Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		
9	-		Purchase	2019-02...	Coffee	2.28	Paid	Cash	Tim Hort...	Campus	Default		
10	+		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		

Figure 21. User Interface - Mark expense paid/unpaid (b)

Hide/show paid expenses

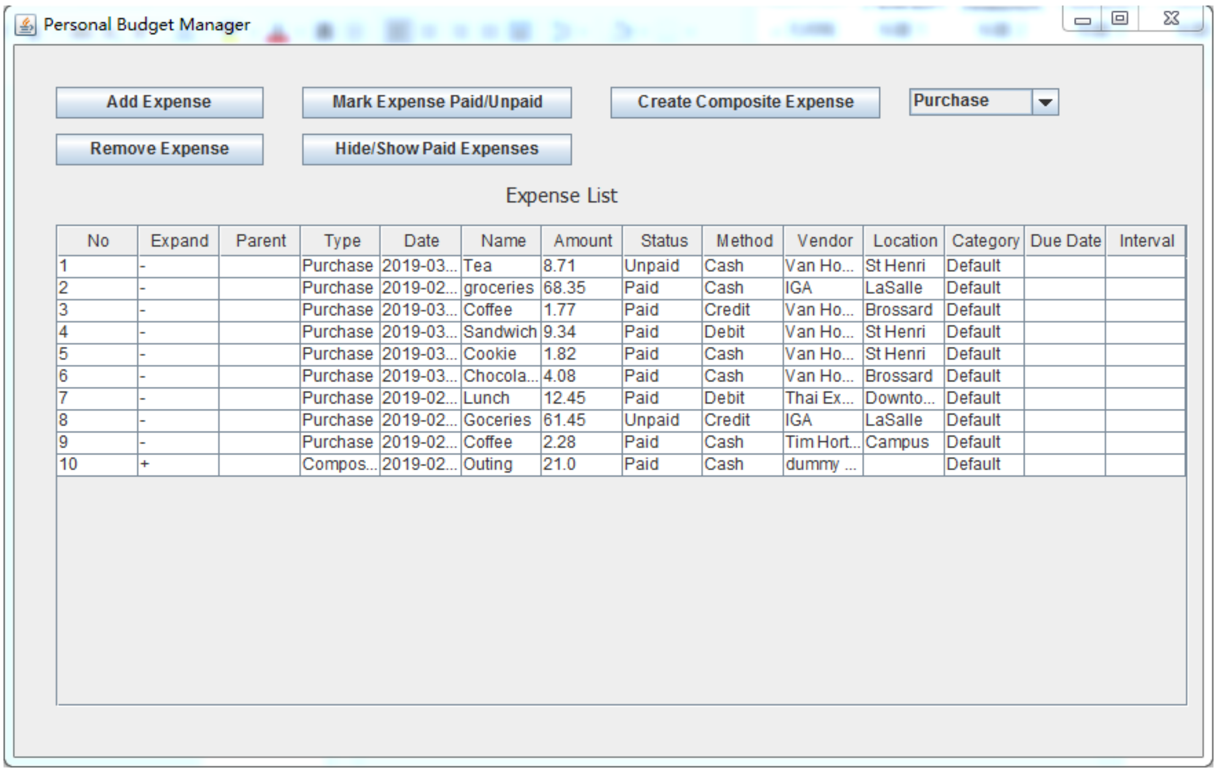
Click the button 'hide/show paid expense', then the list will only show the unpaid expenses. Click the button again, the list will back to original look.



Expense List

No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Purchase	2019-03...	Tea	8.71	Unpaid	Cash	Van Ho...	St Henri	Default		
8	-		Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		

Figure 22. User Interface - Hide/Show paid expenses (a)

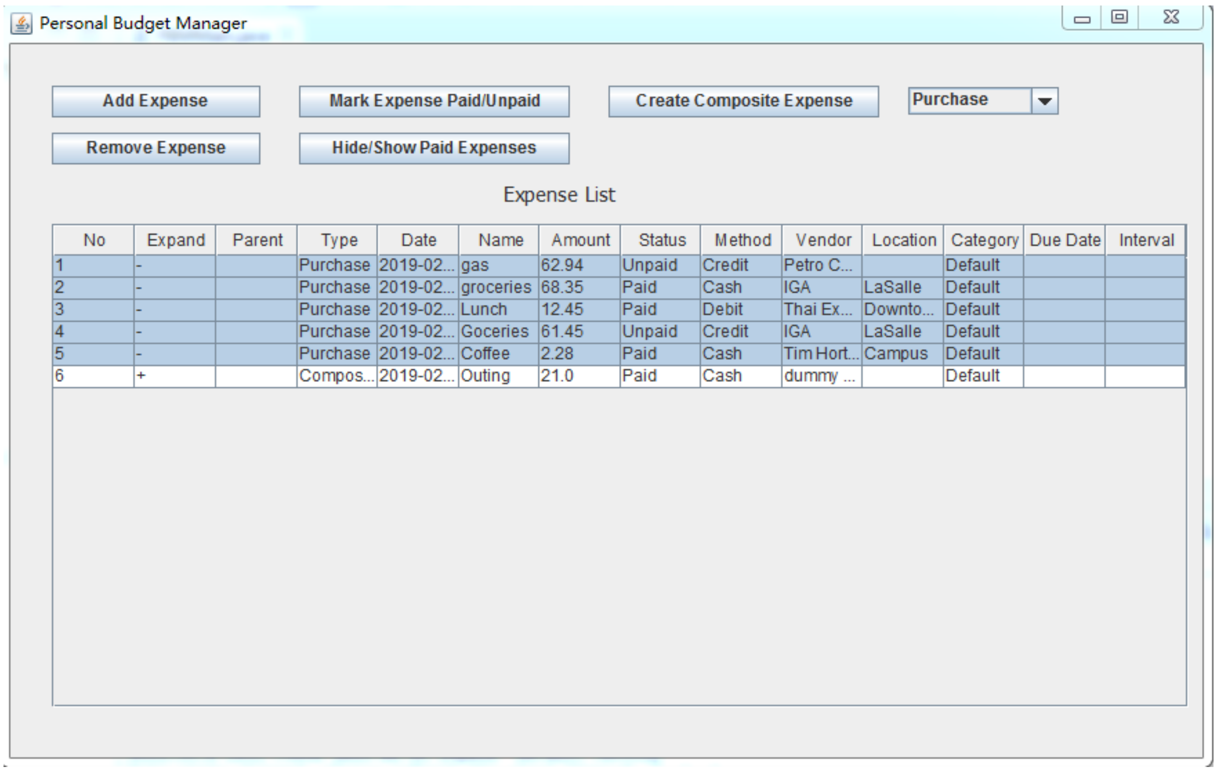


No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Purchase	2019-03...	Tea	8.71	Unpaid	Cash	Van Ho...	St Henri	Default		
2	-		Purchase	2019-02...	groceries	68.35	Paid	Cash	IGA	LaSalle	Default		
3	-		Purchase	2019-03...	Coffee	1.77	Paid	Credit	Van Ho...	Brossard	Default		
4	-		Purchase	2019-03...	Sandwich	9.34	Paid	Debit	Van Ho...	St Henri	Default		
5	-		Purchase	2019-03...	Cookie	1.82	Paid	Cash	Van Ho...	St Henri	Default		
6	-		Purchase	2019-03...	Chocola...	4.08	Paid	Cash	Van Ho...	Brossard	Default		
7	-		Purchase	2019-02...	Lunch	12.45	Paid	Debit	Thai Ex...	Downto...	Default		
8	-		Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		
9	-		Purchase	2019-02...	Coffee	2.28	Paid	Cash	Tim Hort...	Campus	Default		
10	+		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		

Figure 23. User Interface - Hide/show paid expenses (b)

Create composite expense

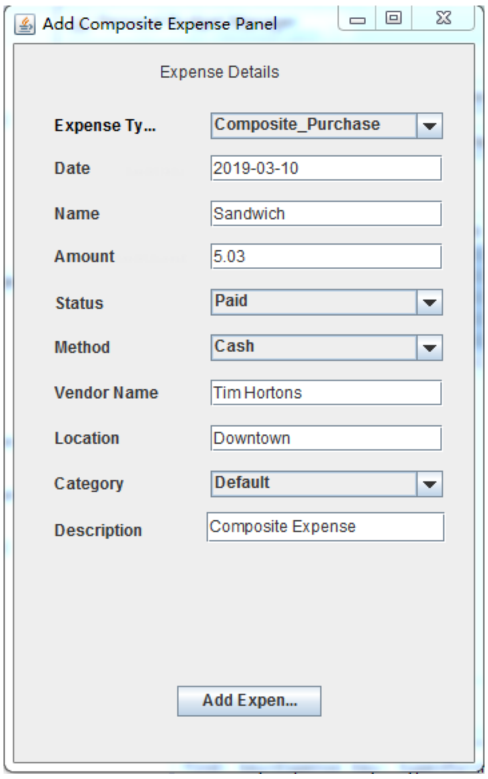
If users want to create a composite expense, just choose lines, click the button "Create Composite Expense", another interface will pop up. Here they can input description of the composite expense. Click the button "Add Expense", a composite expense can be created. Double click "+" it will show details.



Expense List

No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Purchase	2019-02...	gas	62.94	Unpaid	Credit	Petro C...		Default		
2	-		Purchase	2019-02...	groceries	68.35	Paid	Cash	IGA	LaSalle	Default		
3	-		Purchase	2019-02...	Lunch	12.45	Paid	Debit	Thai Ex...	Downto...	Default		
4	-		Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		
5	-		Purchase	2019-02...	Coffee	2.28	Paid	Cash	Tim Hort...	Campus	Default		
6	+		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		

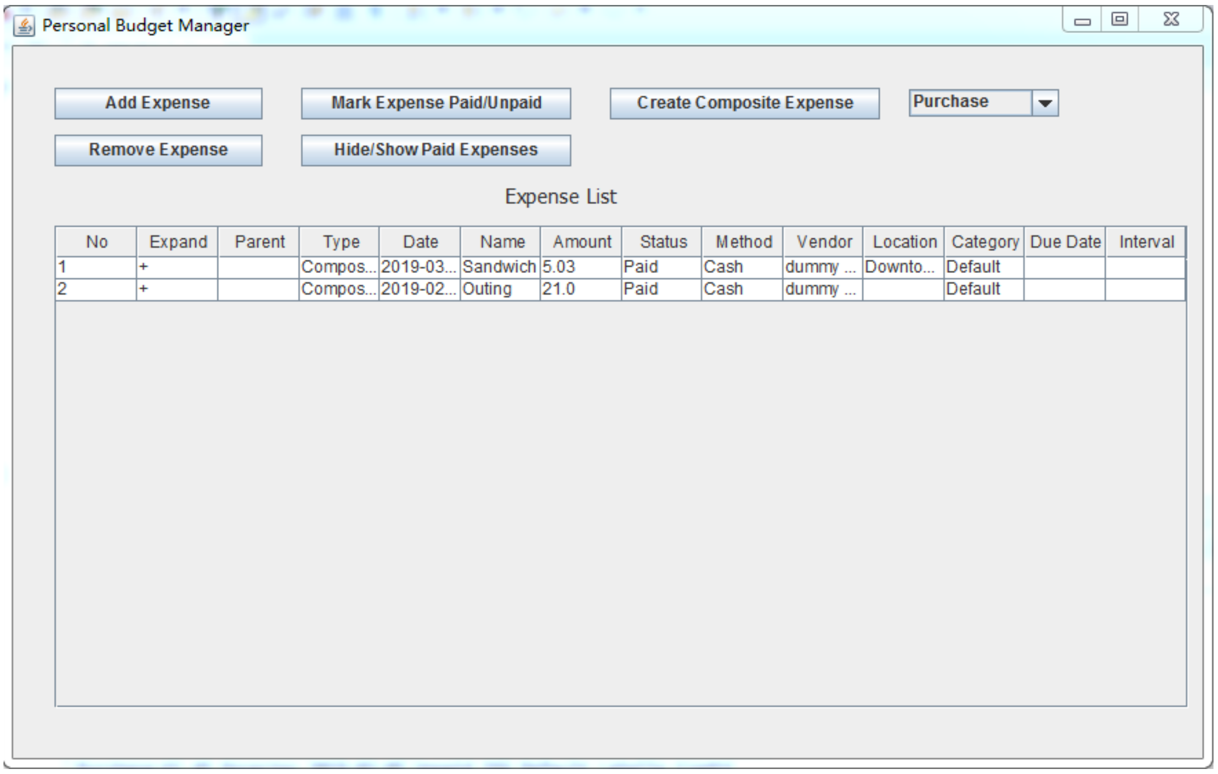
Figure 24. User Interface - Create composite Expense (a)



Expense Details

Expense Ty...	Composite_Purchase
Date	2019-03-10
Name	Sandwich
Amount	5.03
Status	Paid
Method	Cash
Vendor Name	Tim Hortons
Location	Downtown
Category	Default
Description	Composite Expense

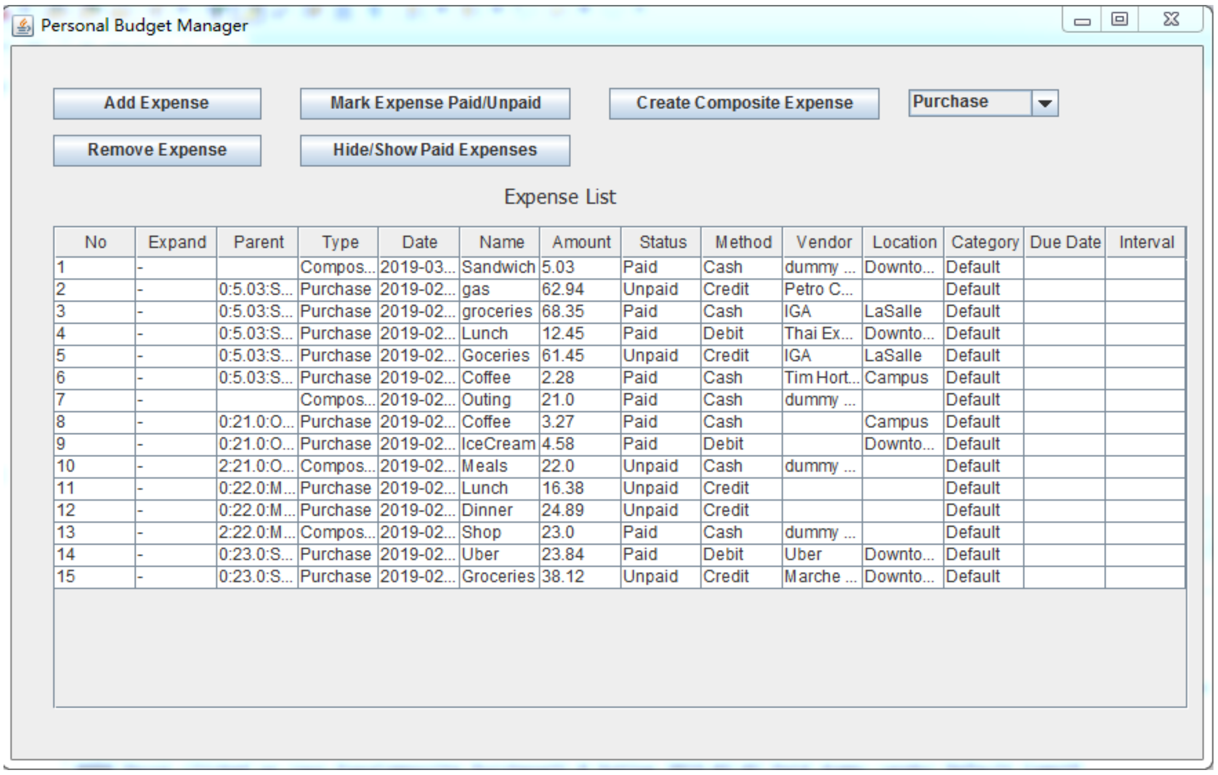
Figure 25. User Interface - Create composite expense (b)



Expense List

No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	+		Compos...	2019-03...	Sandwich	5.03	Paid	Cash	dummy ...	Downto...	Default		
2	+		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		

Figure 26. User Interface - Create composite expense (c)



Expense List

No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Compos...	2019-03...	Sandwich	5.03	Paid	Cash	dummy ...	Downto...	Default		
2	-	0:5.03:S...	Purchase	2019-02...	gas	62.94	Unpaid	Credit	Petro C...		Default		
3	-	0:5.03:S...	Purchase	2019-02...	groceries	68.35	Paid	Cash	IGA	LaSalle	Default		
4	-	0:5.03:S...	Purchase	2019-02...	Lunch	12.45	Paid	Debit	Thai Ex...	Downto...	Default		
5	-	0:5.03:S...	Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		
6	-	0:5.03:S...	Purchase	2019-02...	Coffee	2.28	Paid	Cash	Tim Hort...	Campus	Default		
7	-		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		
8	-	0:21.0:O...	Purchase	2019-02...	Coffee	3.27	Paid	Cash		Campus	Default		
9	-	0:21.0:O...	Purchase	2019-02...	IceCream	4.58	Paid	Debit		Downto...	Default		
10	-	2:21.0:O...	Compos...	2019-02...	Meals	22.0	Unpaid	Cash	dummy ...		Default		
11	-	0:22.0:M...	Purchase	2019-02...	Lunch	16.38	Unpaid	Credit			Default		
12	-	0:22.0:M...	Purchase	2019-02...	Dinner	24.89	Unpaid	Credit			Default		
13	-	2:22.0:M...	Compos...	2019-02...	Shop	23.0	Paid	Cash	dummy ...		Default		
14	-	0:23.0:S...	Purchase	2019-02...	Uber	23.84	Paid	Debit	Uber	Downto...	Default		
15	-	0:23.0:S...	Purchase	2019-02...	Groceries	38.12	Unpaid	Credit	Marche ...	Downto...	Default		

Figure 27. User Interface - Create composite expense (d)

3.2 Module Interface Diagrams

The MVC model has interfaces that link the like actions and updates across the Model, View and Controller as shown below in the figure. The methods in the various modules are accessed by these interfaces are described in the following subsections.

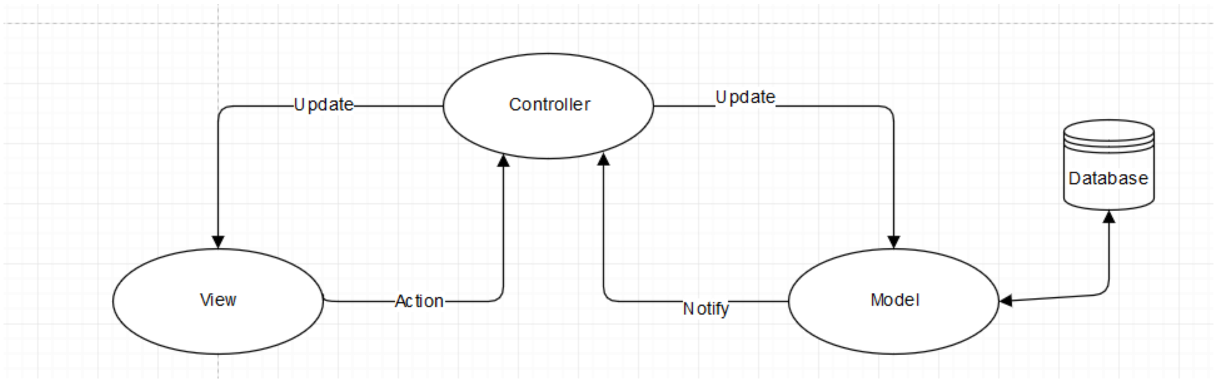
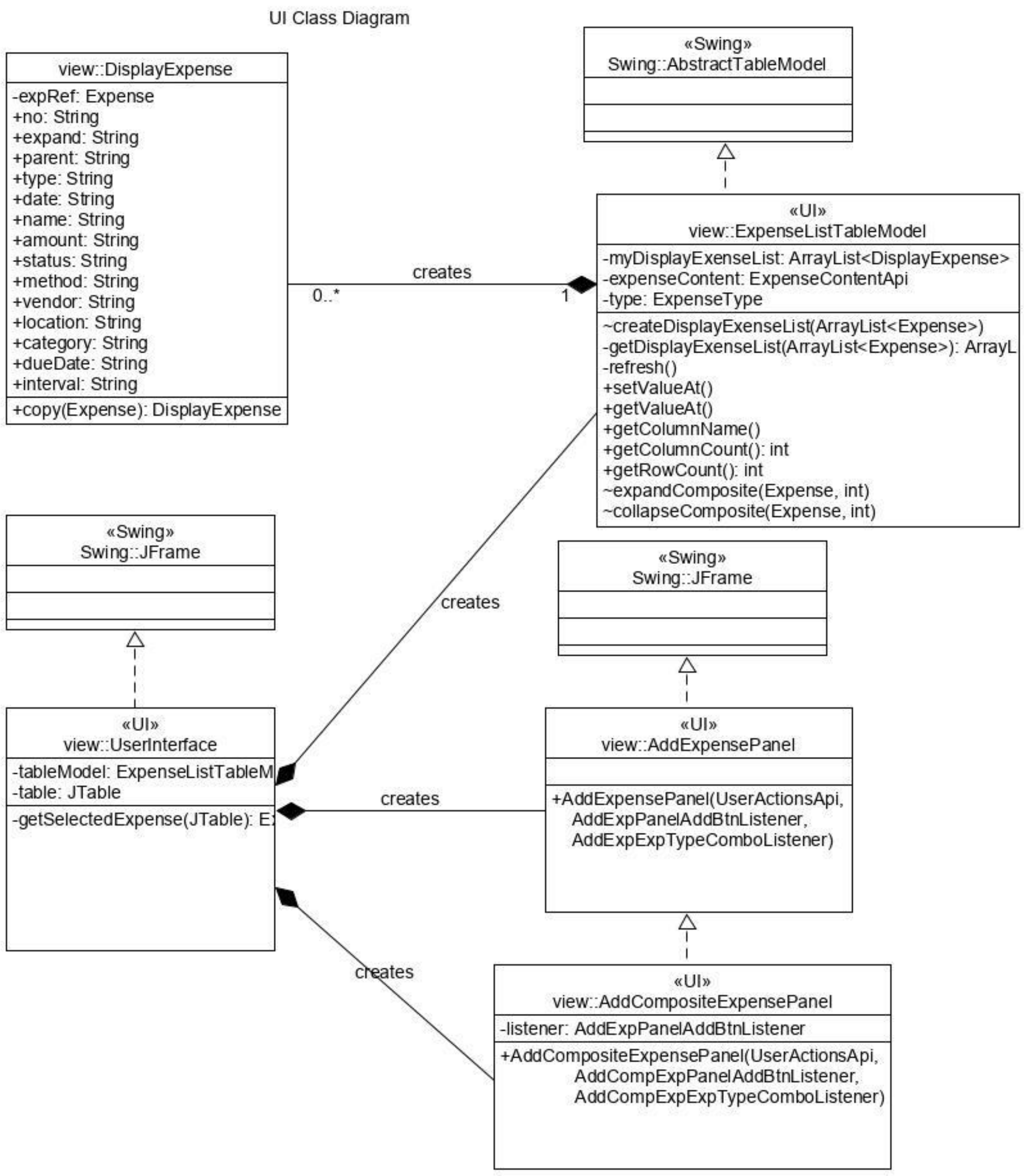


Figure 28. Module Interface Diagram – MVC

The following figure below illustrates the class and their methods that implement the UI, which was described in detail in the section above.



UI Class Diagram
view::DisplayExpense
-expRef: Expense
+no: String
+expand: String
+parent: String
+type: String
+date: String
+name: String
+amount: String
+status: String
+method: String
+vendor: String
+location: String
+category: String
+dueDate: String
+interval: String
+copy(Expense): DisplayExpense
«Swing»
Swing::AbstractTableModel
«UI»
view::ExpenseListTableModel
-myDisplayExenseList: ArrayList<DisplayExpense>
-expenseContent: ExpenseContentApi
-type: ExpenseType
~createDisplayExenseList(ArrayList<Expense>)
-getDisplayExenseList(ArrayList<Expense>): ArrayL
-refresh()
+setValueAt()
+getValueAt()
+getColumnName()
+getColumnCount(): int
+getRowCount(): int
~expandComposite(Expense, int)
~collapseComposite(Expense, int)
creates
0..*
1
«Swing»
Swing::JFrame
creates
«Swing»
Swing::JFrame
«UI»
view::UserInterface
-tableModel: ExpenseListTableM
-table: JTable
-getSelectedExpense(JTable): E
creates
«UI»
view::AddExpensePanel
+AddExpensePanel(UserActionsApi, AddExpPanelAddBtnListener, AddExpExpTypeComboListener)
creates
«UI»
view::AddCompositeExpensePanel
-listener: AddExpPanelAddBtnListener
+AddCompositeExpensePanel(UserActionsApi, AddCompExpPanelAddBtnListener, AddCompExpExpTypeComboListener)

Figure 29. Module Interface Diagram - UI Class Diagram

3.2.1 View Interface

The following interfaces between the controller and view are called whenever there is a update change in its internal data of the model interface or an action performed by the user on the graphical user interface.

The following are the interfaces and their various methods:

3.2.1.1 ExpenseContentApi

This interface is the API for the viewer and the following are the data access methods.

findExpense(ExpenseKey)
getAllBills()
getAllPurchases()
getBills()
getCompositeBills()
getCompositePurchases()
getPurchases()

3.2.1.2 ExpenseObserver

The following controller interface updates the viewer module.

The following methods notify data change to controller：

init(ExpenseSubject)
update(List<Map<ExpenseKey, Expense>>)

The following are methods to get the data based on the display parameters:

find(ExpenseKey)
getData(DisplayParameters)
getDataByExpenseType(DisplayParameters)

3.2.1.3 ExpenseReportDisplay

This interface is for the viewer to browse through the Expense list.

The following method is to display the parameters:

getContext()

The following method is to get data for the next page in the list:

getNextPage()

The following method is to reset the offset, when data state is changed:

resetOffset()

The following method is to set the index to the current page being displayed:

setOffset(int)

The following method is to set the no of rows displayed per page:
setPageSize()

The following method to refreshes the view/report when data state is changed:
shouldReload()

3.2.1.4 UserActionsApi

Interface to send action in the view module to the controller:
addExpense(Expense)
AddExpensesToComposite(Expense, ArrayList<Expense>)
AddExpenseToComposite(Expense, Expense)
changePaymentStatus(Expense)
changePaymentStatus(Expense, Expense)
modifyExpense(Expense, Expense)
removeExpense(Expense)
removeExpense(Expense, Expense)

3.2.1.5 UserInputValidation

This interface performs UI input validation for the view module:
validate(Expense)
validateToModify(Expense, Expense)

3.2.2 Controller Interface

The controller interfaces update and performes action between the view and model modules respectively. The following below are the various interfaces and their methods, respectively.

3.2.2 .1 Dataloader

This control interface updates data to the view module:
dataLoad(List<Map<ExpenseKey, Expense>>)

3.2.2.3 ExpenseContainerApi

This controller interface is a data container. It updates the view module.

The following methods initialize the data member:

init(Store)

getStore()

The following are Data write methods for viewer:

addExpense(Expense)

addExpenseIntoComposite(Expense, Expense)

addExpensesIntoComposite(Expense, ArrayList<Expense>)

changePaymentStatus(Expense, Date)

getExpenses(int)

getMatching(Expense)

modifyExpense(Expense, Expense)

removeExpense(Expense)

removeExpenseFromComposite(Expense, Expense)

3.2.2 .4 ExpenseSubject

This control Interface is a subject class that updates the view module. The following are its methods:

register(ExpenseObserver)
resetStateChange()
start()

3.2.3 Model Interface

The interface between the controller and model is called whenever the controller receives input from the player. The following methods are part of the interface:

3.2.3.1 Expense

This interface updates the model module and notifies the controller module if prompted by input from the user.

The following is a key accessor method:

getKey()

The following are Expense medication and accessor methods:

add(Expense expense);
remove(Expense expense);
changePaymentStatus(Date date);
changePaymentStatus(Status status, Date date);
setParent(Expense parent);

The following is a get index method:

get(int index);

The following is a find methods only among the current composite children:

get(Expense exp);

The following methods are recursive functions among the sub-children:

find(Expense exp);
ArrayList<Expense> getSubItems();
getNoOfSubItems();

The following methods return the parent in case of composite type:

getParent();
getRoot();
iseqal(final Expense expense);
display();

The following methods are Accessors for Viewer:

getType();
getAmount();
getName();
getDate();
getStatus();
getMode();
getVendor();
getLocation();
getCategory();
getPaymentDate();
getDescription();

getDueDate();
getInterval();

3.3 Dynamic Models of System Interface

In order to better portray the interactions between the system modules, we have chosen some scenarios, or major functionalities of the system and will explain and depict them using sequence diagrams. We have elected to use sequence diagrams because they depict the interaction between the classes (or objects) of the system and also show the sequence of calls (or messages) that occur.

3.3.1 Add Expense Scenario

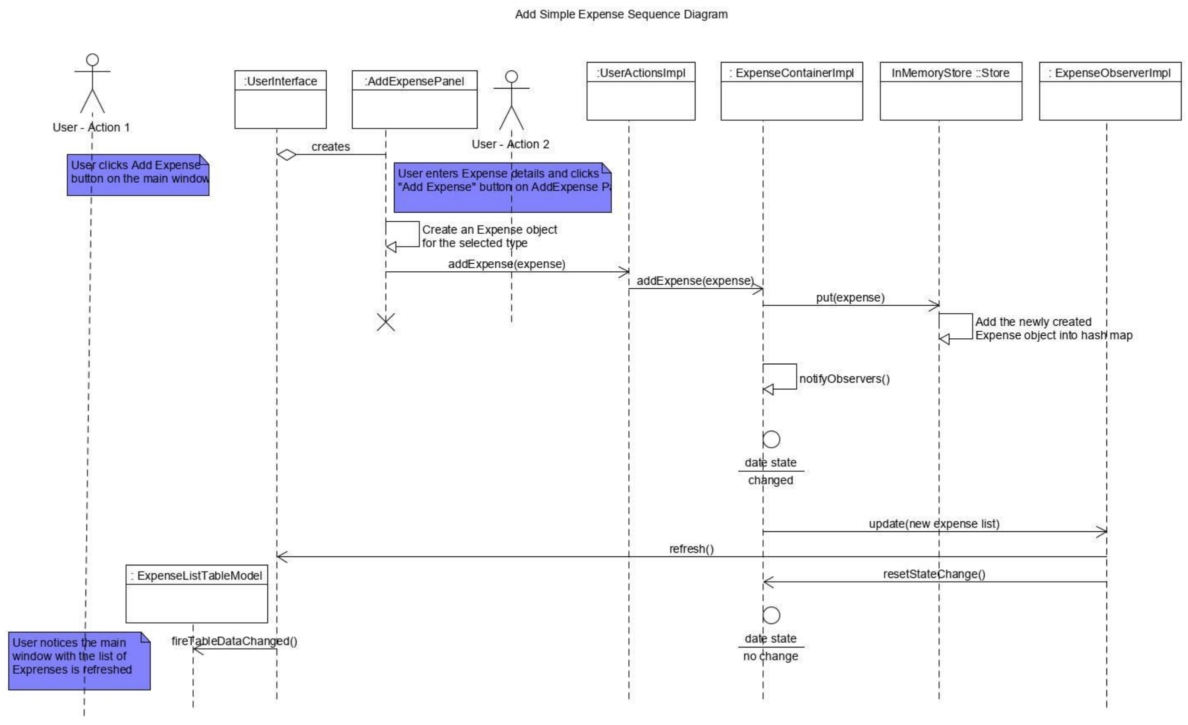


Figure 30. Add expense sequence scenario

3.3.2 Change Payment Status Scenario

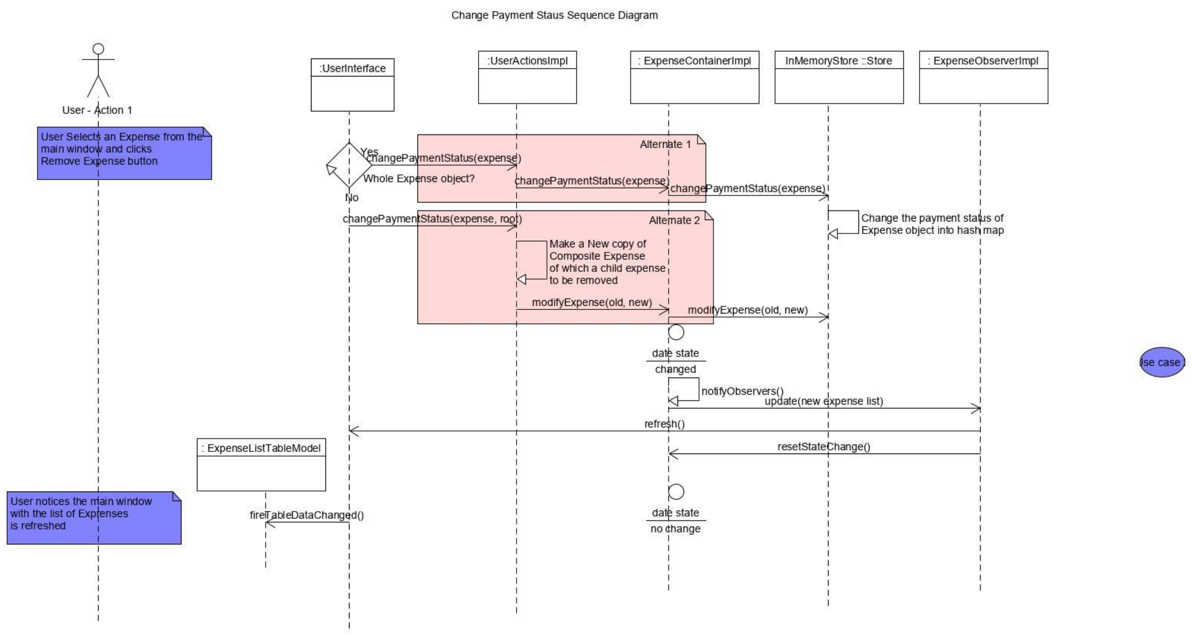

Change Payment Staus Sequence Diagram
User - Action 1
:UserInterface
:UserActionsImpl
: ExpenseContainerImpl
InMemoryStore ::Store
: ExpenseObserverImpl
User Selects an Expense from the main window and clicks Remove Expense button
Alternate 1
Yes
changePaymentStatus(expense)
Whole Expense object?
No
changePaymentStatus(expense)
changePaymentStatus(expense)
changePaymentStatus(expense, root)
Alternate 2
Change the payment status of Expense object into hash map
Make a New copy of Composite Expense of which a child expense to be removed
modifyExpense(old, new)
modifyExpense(old, new)
date state changed
notifyObservers()
update(new expense list)
refresh()
: ExpenseListTableModel
resetStateChange()
User notices the main window with the list of Expenses is refreshed
fireTableDataChanged()
date state no change
Use case

Figure 31. Change payment status sequence scenario

3.3.3 Remove Expense scenario

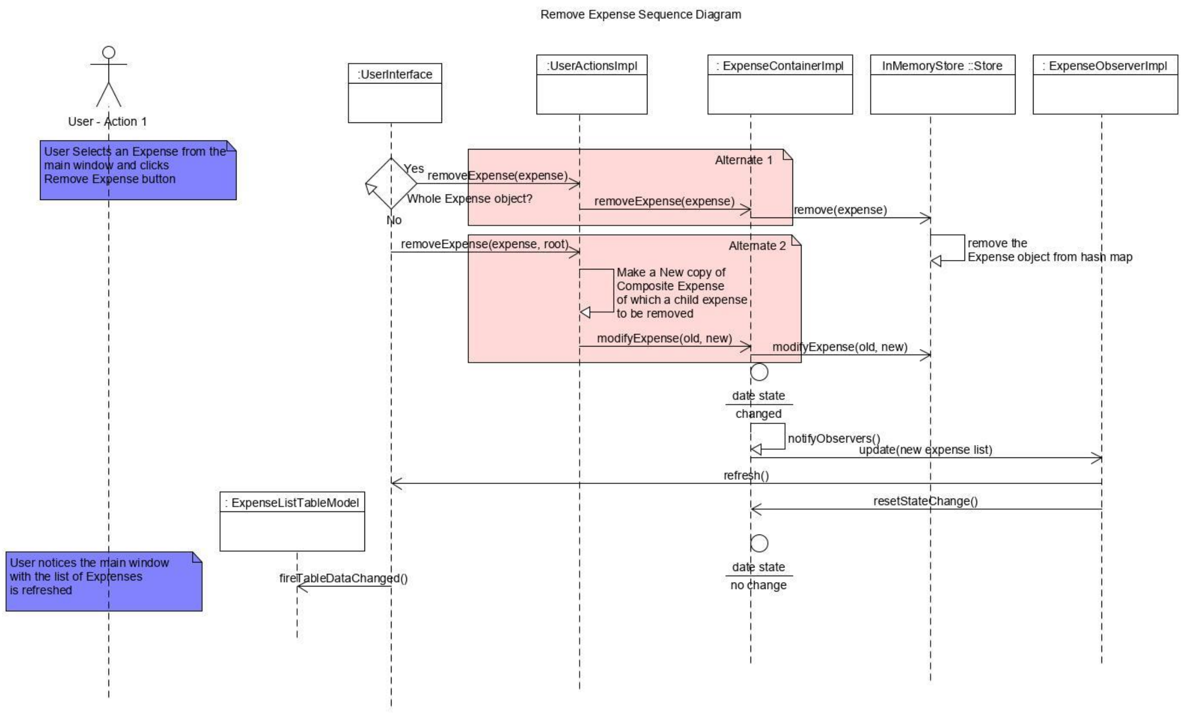


Figure 32. Remove expense sequence scenario

4. Internal Module Design

The system is divided into three modules, the model, the view and the controller. The modules interact together using their respective interface. In this section, we will describe the detailed design of three modules.

4.1 Module <Model>

The most important module is the model. This module is used to implement application methods and store data. The model consists of six classes, including *AbstractExpense, ExpeenseKey, Bill*, *Purchase*, *CompositeBill* and *CompositePurchase.* Each class contains the attributes and methods.

4.1.1 Model Class Diagram

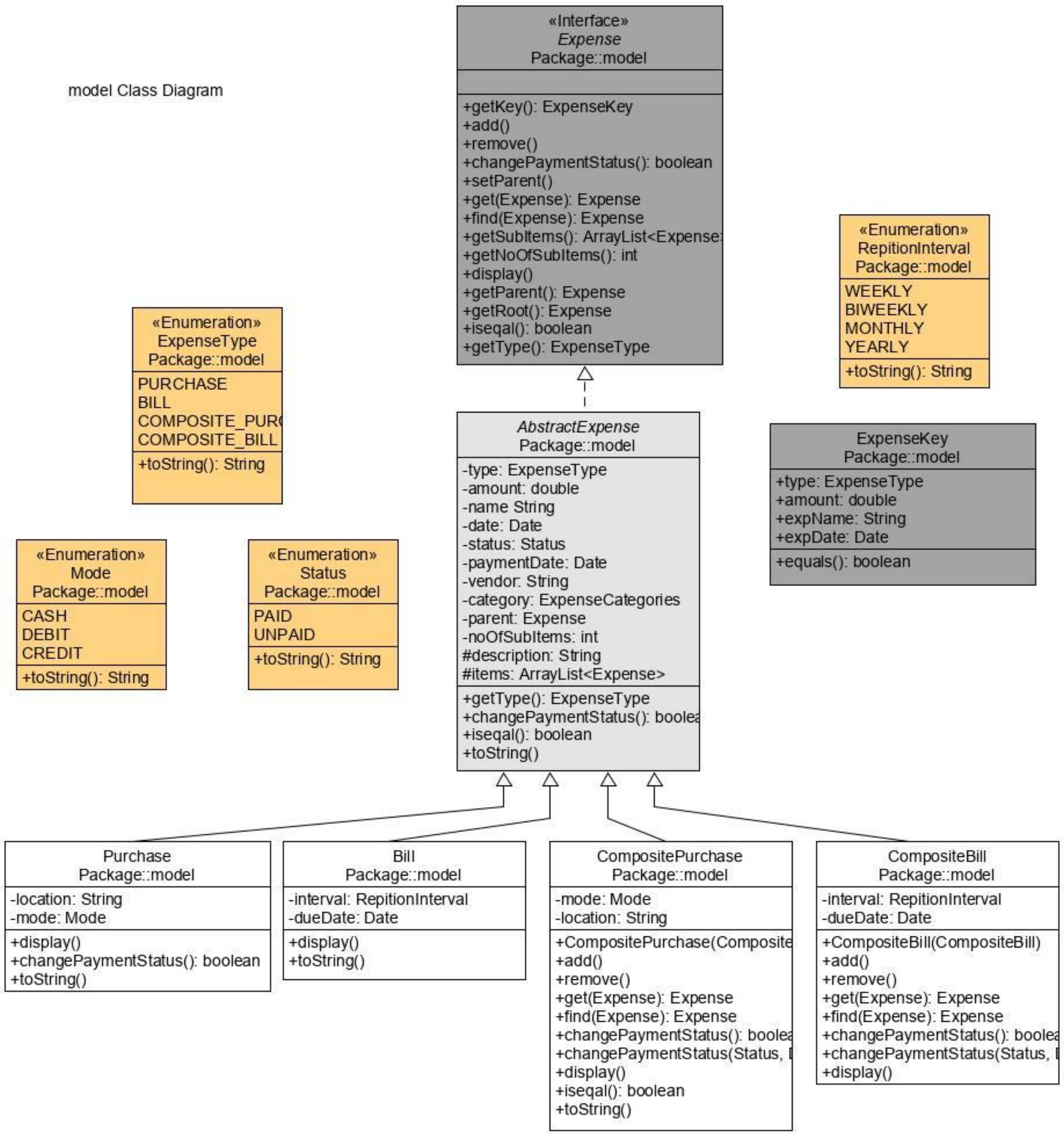


Figure 33. Model Class Diagram

4.2 Module <View>

The module View consists of different views that users can see in the system.

4.2.1 View Class Diagram

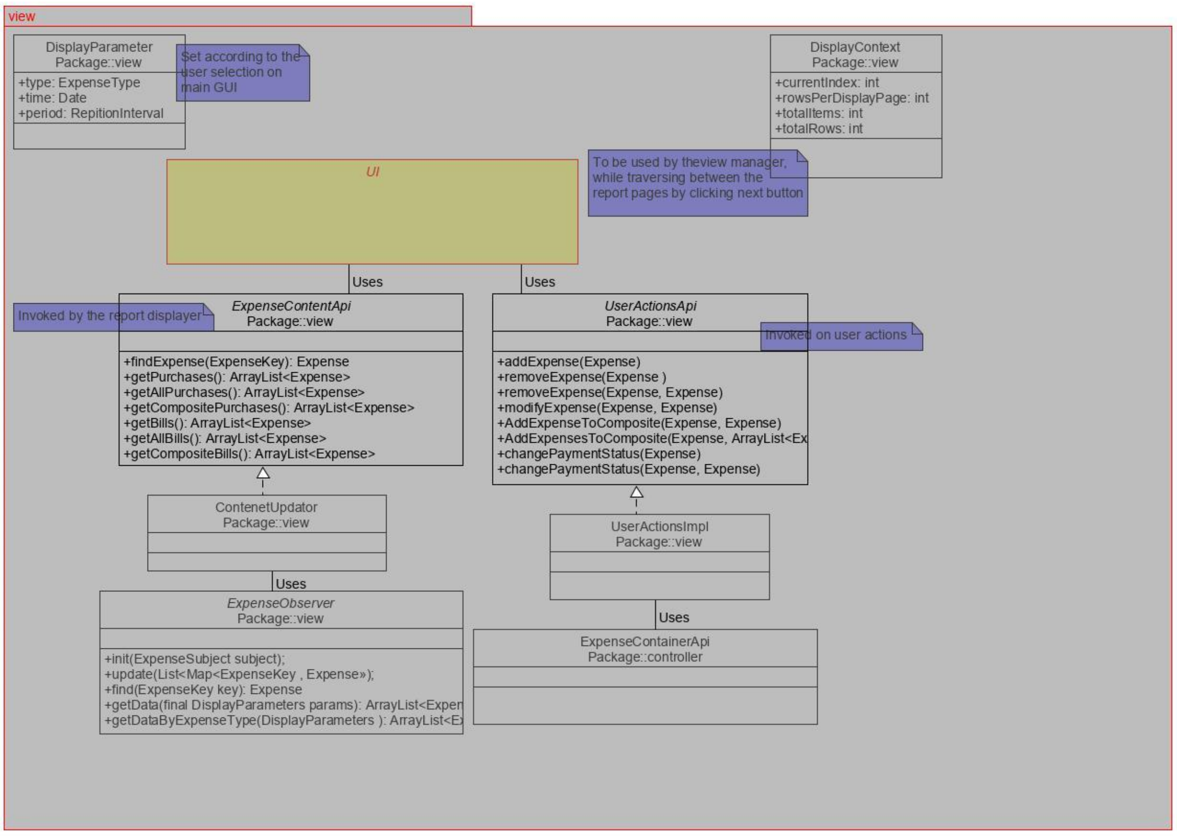

view
DisplayParameter
Package::view
+type: ExpenseType
+time: Date
+period: RepitionInterval
Set according to the user selection on main GUI
DisplayContext
Package::view
+currentIndex: int
+rowsPerDisplayPage: int
+totalItems: int
+totalRows: int
To be used by theview manager, while traversing between the report pages by clicking next button
UI
Uses
Uses
Invoked by the report displayer
ExpenseContentApi
Package::view
+findExpense(ExpenseKey): Expense
+getPurchases(): ArrayList<Expense>
+getAllPurchases(): ArrayList<Expense>
+getCompositePurchases(): ArrayList<Expense>
+getBills(): ArrayList<Expense>
+getAllBills(): ArrayList<Expense>
+getCompositeBills(): ArrayList<Expense>
UserActionsApi
Package::view
Invoked on user actions
+addExpense(Expense)
+removeExpense(Expense)
+removeExpense(Expense, Expense)
+modifyExpense(Expense, Expense)
+AddExpenseToComposite(Expense, Expense)
+AddExpensesToComposite(Expense, ArrayList<Ex
+changePaymentStatus(Expense)
+changePaymentStatus(Expense, Expense)
ContenetUpdator
Package::view
UserActionsImpl
Package::view
Uses
ExpenseObserver
Package::view
+init(ExpenseSubject subject);
+update(List<Map<ExpenseKey , Expense»);
+find(ExpenseKey key): Expense
+getData(final DisplayParameters params): ArrayList<Exper
+getDataByExpenseType(DisplayParameters): ArrayList<E
Uses
ExpenseContainerApi
Package::controller

Figure 34. View Class Diagram

4.3 Module <Controller>

For the Controller module, we have five classes, including DatabaseStore, ExpenseContainerImpl, FileLoaderImpl, FileStore and InMemoryStore.

4.3.1 Controller Class Diagram

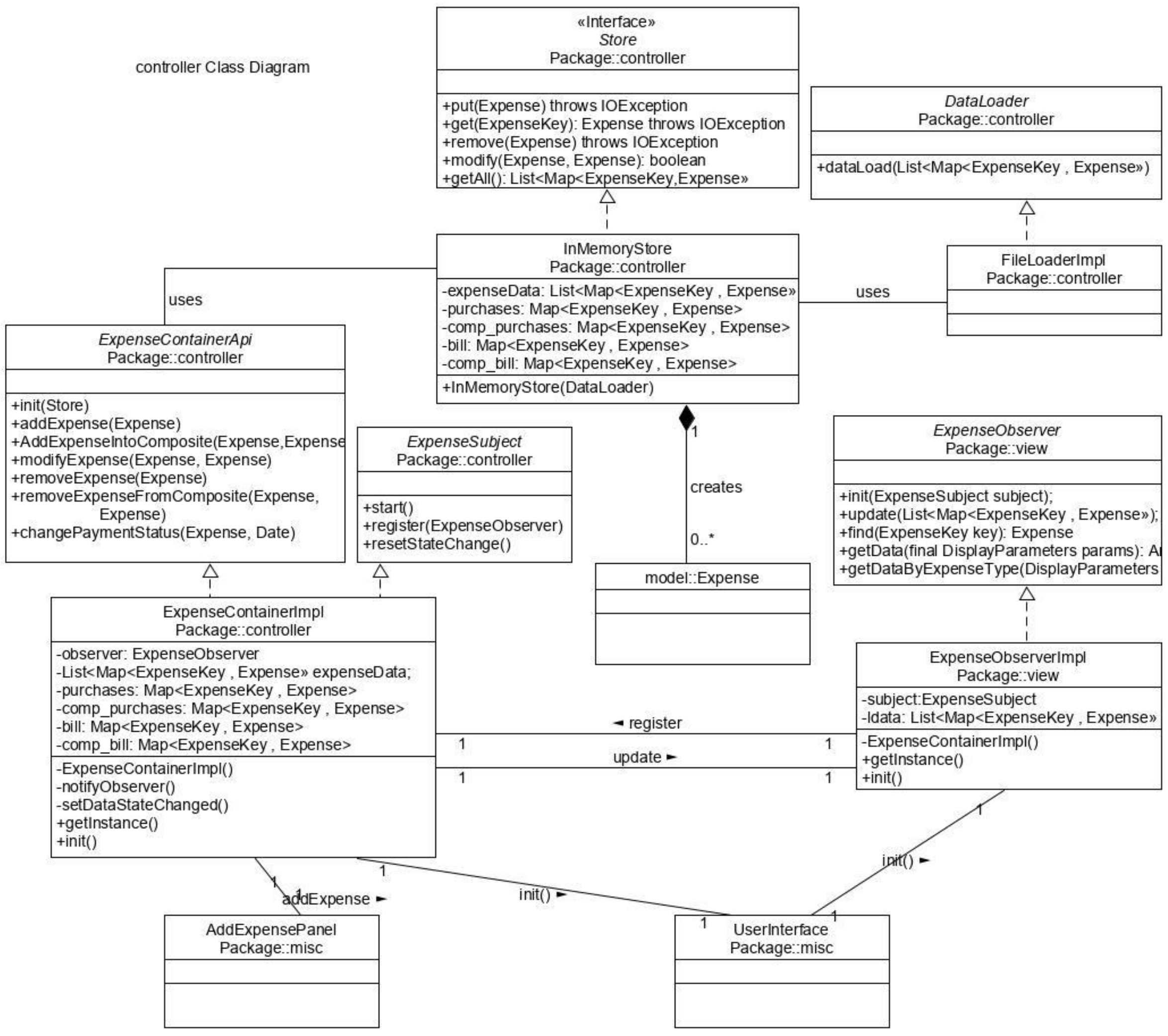

controller Class Diagram
«Interface»
Store
Package::controller
+put(Expense) throws IOException
+get(ExpenseKey): Expense throws IOException
+remove(Expense) throws IOException
+modify(Expense, Expense): boolean
+getAll(): List<Map<ExpenseKey,Expense»
DataLoader
Package::controller
+dataLoad(List<Map<ExpenseKey , Expense»)
InMemoryStore
Package::controller
-expenseData: List<Map<ExpenseKey , Expense»
-purchases: Map<ExpenseKey , Expense>
-comp_purchases: Map<ExpenseKey , Expense>
-bill: Map<ExpenseKey , Expense>
-comp_bill: Map<ExpenseKey , Expense>
+InMemoryStore(DataLoader)
uses
FileLoaderImpl
Package::controller
uses
ExpenseContainerApi
Package::controller
+init(Store)
+addExpense(Expense)
+AddExpenseIntoComposite(Expense,Expense
+modifyExpense(Expense, Expense)
+removeExpense(Expense)
+removeExpenseFromComposite(Expense, Expense)
+changePaymentStatus(Expense, Date)
ExpenseSubject
Package::controller
+start()
+register(ExpenseObserver)
+resetStateChange()
1
creates
0..*
model::Expense
ExpenseObserver
Package::view
+init(ExpenseSubject subject);
+update(List<Map<ExpenseKey , Expense»);
+find(ExpenseKey key): Expense
+getData(final DisplayParameters params): A
+getDataByExpenseType(DisplayParameters
ExpenseContainerImpl
Package::controller
-observer: ExpenseObserver
-List<Map<ExpenseKey , Expense» expenseData;
-purchases: Map<ExpenseKey , Expense>
-comp_purchases: Map<ExpenseKey , Expense>
-bill: Map<ExpenseKey , Expense>
-comp_bill: Map<ExpenseKey , Expense>
-ExpenseContainerImpl()
-notifyObserver()
-setDataStateChanged()
+getInstance()
+init()
◄ register
update ►
1
1
1
1
ExpenseObserverImpl
Package::view
-subject:ExpenseSubject
-ldata: List<Map<ExpenseKey , Expense»
-ExpenseContainerImpl()
+getInstance()
+init()
1
init() ►
1
1
addExpense ►
1
init() ►
1
1
AddExpensePanel
Package::misc
UserInterface
Package::misc

Figure 35. Controller Class Diagram