Park Ye Won - Project Portfolio

1. Introduction

This project portfolio documents my contribution to the software project PalPay.

1.1. About the team

PalPay is a project collectively done by five Sophomore Computer Science Undergraduate students in National University of Singapore. As part of a requirement for our module - CS2103T Software Engineering, we morphed an existing product called Address Book (Level 3) given constraints that the user must interact with it using Command Line Interface (CLI).

1.2. About the project

PalPay is a desktop application that is target to students who have limited time to keep track of their incomes and expenditure and wish to have a simple and quick means to take charge of their finances. Also recognising the problem faced by many students that they cannot keep track of lending and borrowing of money with friends, PalPay offers a ledger function which keeps track of the movement of money between friends on a daily basis.

This is what PalPay looks like:

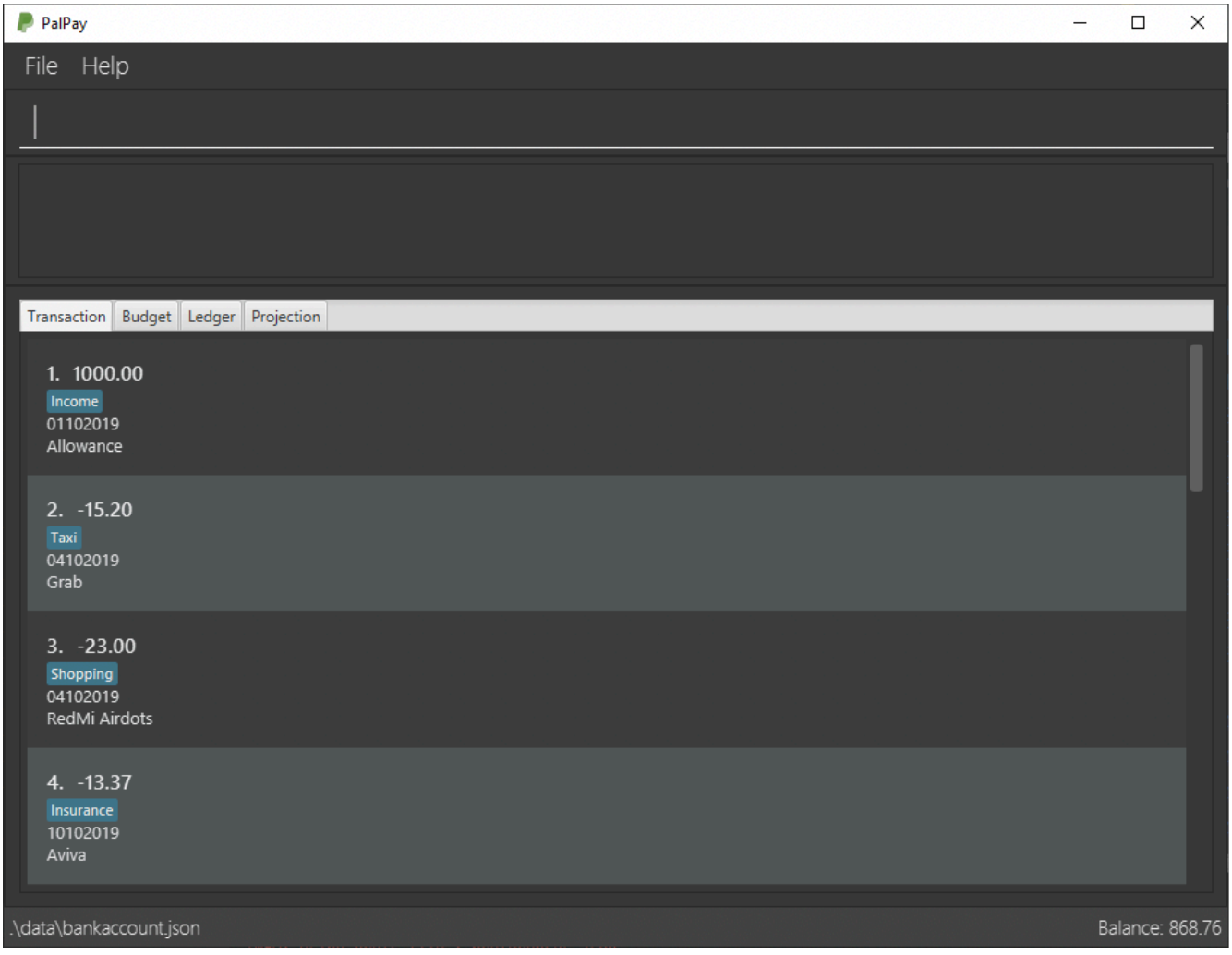

Figure 1. PalPay's Graphical User Interface

With my role as a developer for this project, I implemented the budget function. Understanding that many students have a poor management of their expenses, PalPay budget function allows them to set a budget so that they can be more conscious of their expenditure. As they make expenses in that particular category, the budget will indicate the amount of money left from the budget and serve as a constant reminder for them to make smart financial choices.

2. Summary of contributions

This section shows a summary of my coding, documentation and other helpful contributions to the team in this project.

Enhancement added: I added the ability to set a budget in the bank account * What it does: The set command allows the user to set a budget until a deadline in a given category. As the user makes OutTransaction s of the given category, he can see the percentage of remaining budget, as well as the number of days remaining.

• Justification: As the user wishes to manage his expenses and restrict the overspending of money, setting the budget is useful in allowing the user to track the outflow of his money. Moreover, as the deadline he set approaches or upon exceeding the set budget, the budget card displays red font which alerts the user that he needs to manage his finance better. More details will be discussed below in the user guide and developer guide.

• Highlights: This enhancement is based on Budget class which is a build-up of the current Transaction class. While Transaction class allows the user to key in inflows and outflows of money, Budget allows the user to have an indication of the outflows in a given category. As the two classes behaves as though Budget is a child class of Transaction, an in-depth analysis of design alternatives was necessary to ensure that Liskov Substitution Principal was not violated. More details will be elaborated in section Design Considerations of developer guide excerpt below.

Code contributed: Please click these links to see a sample of my code: RepoSense

Other contributions :

• Project management:

- Managed several issue trackers on GitHub: (Issues: #92, #119, #199)

• Enhancements to existing features:

- Comprehensive Unit and Integration Tests (Pull requests: #204)
- Updated the GUI colour scheme for budget tab so that necessary messages can be brought across the user such as exceeding the set budget (Pull requests: #212 #223)

• Documentation:

- Made updates to the User Guide and Developer Guide to make them more informative for the following sections: (Pull requests: #107 #127 #227)

• Community:

- Reported bugs and offered suggestions for other teams in the class (Issues: #253, #255, #256, #260)

3. Contributions to the User Guide

Original AddressBook User Guide was updated with instructions for the enhancements we added. The following is an excerpt from our PalPay User Guide, showing additions that I have made for the set command feature.

<Start of Excerpt>

3.1. Setting a Budget : set

You can set a budget for a particular category until a certain date, given it is not already present in the budget list. A duplicate budget is a budget with the same AMOUNT and DATE and CATEGORY.
If you attempt to do so, you will receive an error message: This budget already exists.

3.1.1. Command Syntax

Format: set $/AMOUNT d/DATE [c/CATEGORY]...

Parameters follow the same restrictions as highlighted in parameter constraints.

• AMOUNT input accepts the budget amount to be set.

• DATE input accepts the deadline to be set. It cannot be a date in the past.

• CATEGORY accepts the CATEGORY for the budget. A budget can be created without CATEGORY inputs in which case, the budget will automatically be assigned `GENERAL' category.

3.1.2. Important Details

Let's say you want to restrict your spending for a certain category until a certain deadline. PalPay allows you to set a budget and serve as a reminder to show how much of the budget set you have left until the deadline (inclusive). You will be more self-conscious of your spending and minimise your spending by setting a budget.

To set a new budget:
1. Type set and enter the relevant details (amount, deadline, category) in the format given above.
2. The result box will display the message New budget successfully set.
3. If the budget already exists in the budget list, the result box will display the message This budget already exists.
4. Now you can see the newly set budget in the budget list.

As you log an expenditure of a particular CATEGORY, your budgets with the same CATEGORY will be adjusted to display the remaining amount of budget. Other budgets in the list belonging to different CATEGORY will not be adjusted.

For example, you went out with your friends and bought a cup of Gong Cha. Before you log your spending, your budget list looks like this:

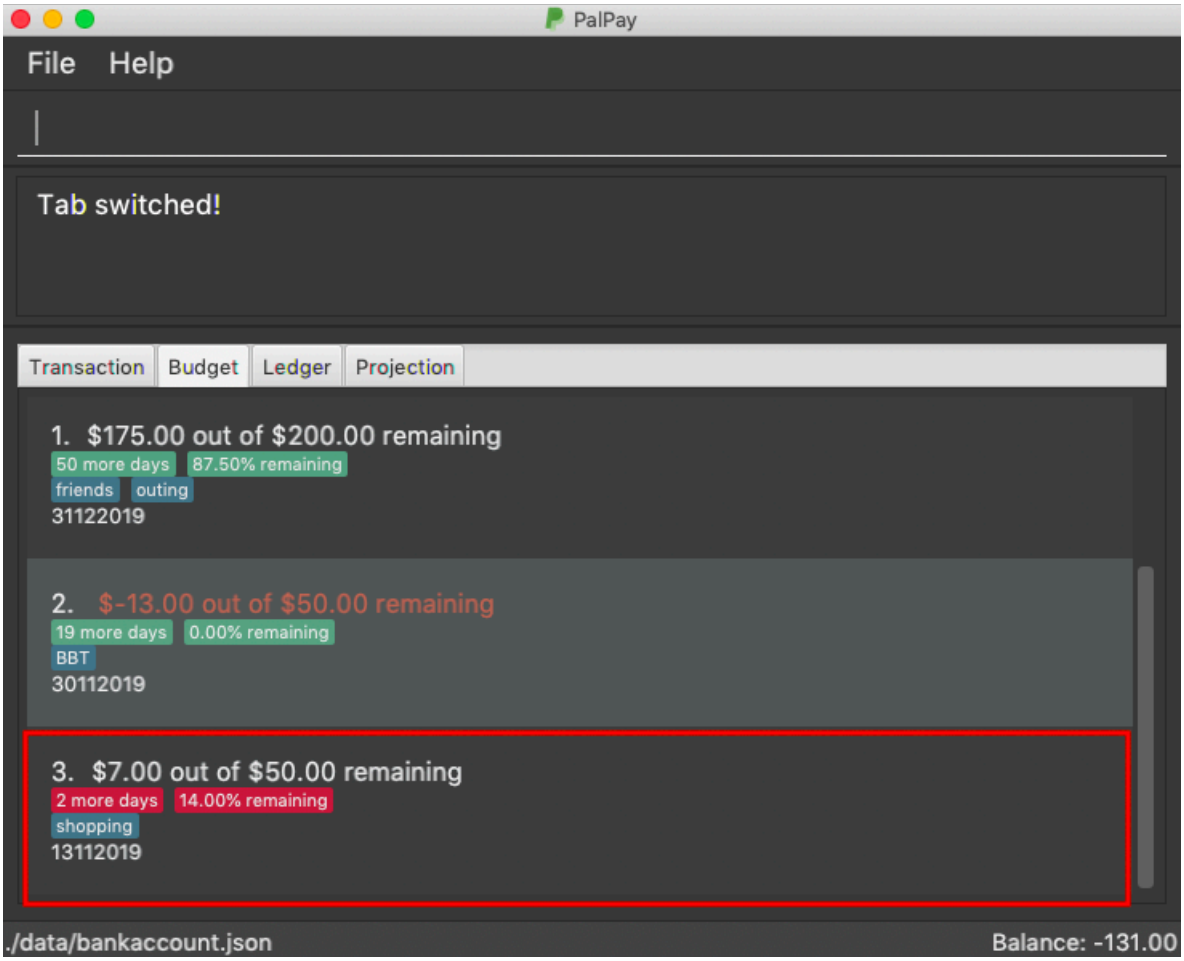

Figure 2. Budget List before Executing OutTransaction

You then type in the command out $/5 c/BBT c/friends n/gong cha d/11112019.

Figure 3. New OutTransaction Command

Your budget list now shows the updated budgets. Observe how Budget 3 is not affected because it does not belong to the relevant **category**.

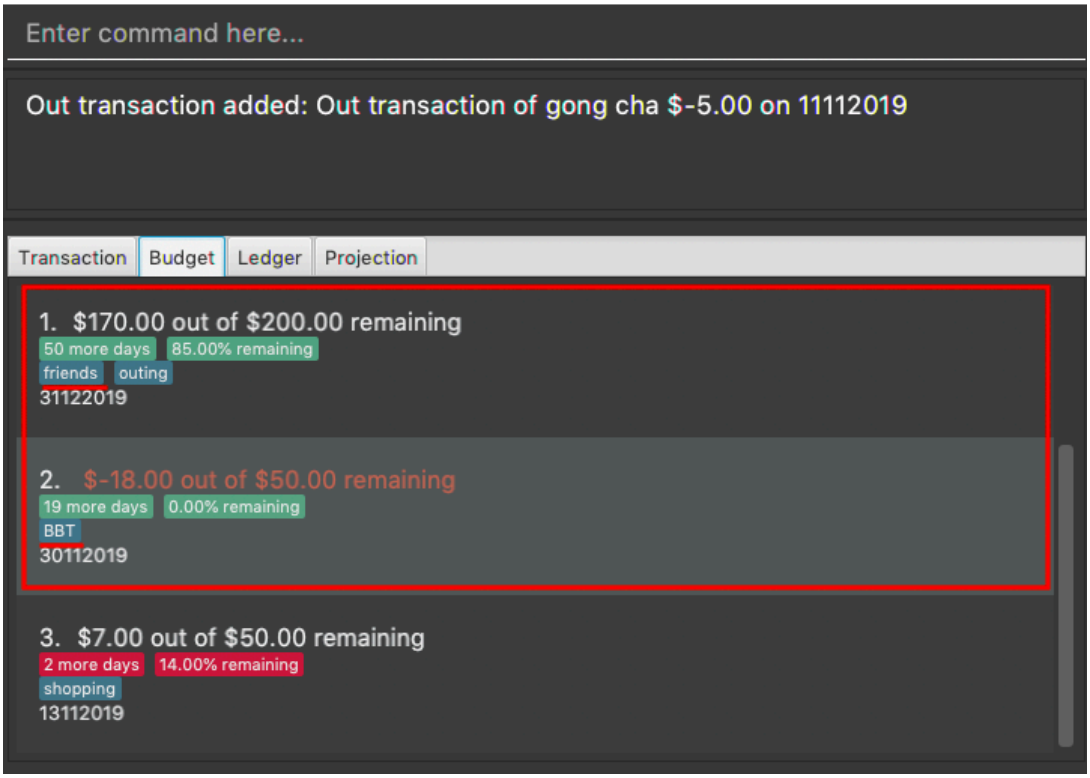

Figure 4. Updated Budget List

Budget will not take into consideration past **OutTransaction** when calculating the remaining budget. Remember, you are setting a budget from TODAY till the stated DATE (inclusive)!

If you overspend beyond a set budget, the overspent budget will be displayed in red. Shown below as budget index 3 is an example of an overspent budget:

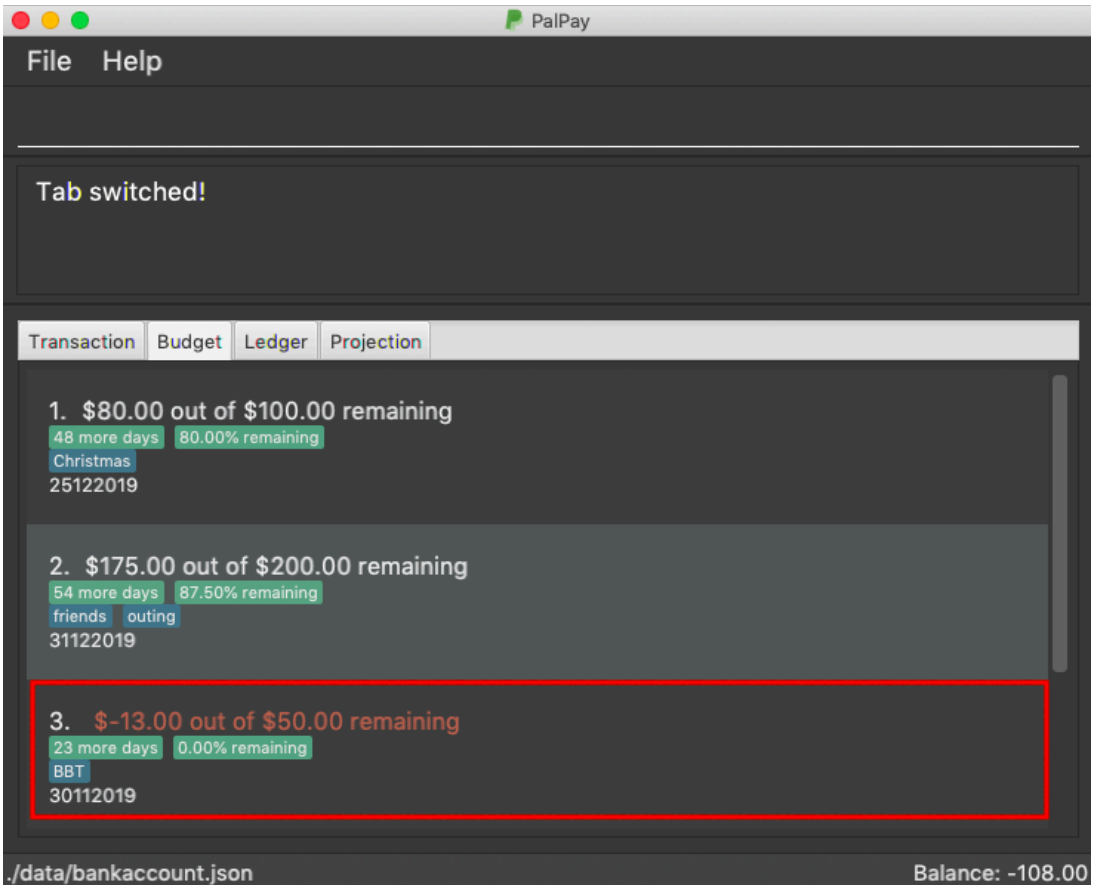

Figure 5. Overspent Budget

As the day you have set for the budget approaches, the countdown placeholder as well as the percentage remaining placeholder will turn to red when the number of remaining days reaches 3 and below. Shown below as budget index 4 is an example of a budget approaching its deadline:

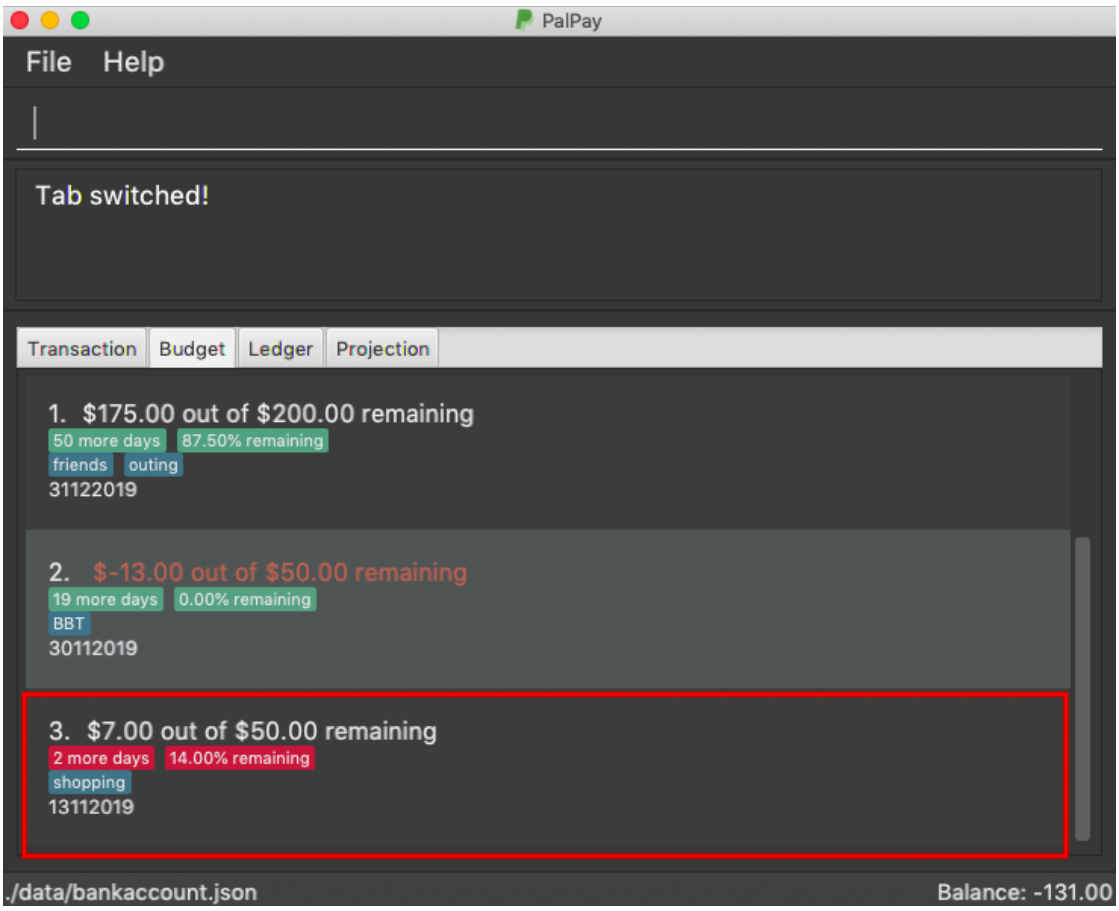

Figure 6. Budget approaching deadline

Examples:
• set $/100 d/010120120 c/BBT

• set $/300 d/29022020 c/shopping <End of Excerpt>

4. Contributions to the Developer Guide

The following excerpt shows my contribution to PalPay Developer Guide for the section set command. The section also highlights the relationship of Budget class with the overall system.

<Start of Excerpt>

4.1. Set Budget Feature: set

The Budget class allows the user to set a budget for a given time period for a category, if specified. The user is allowed to set multiple budgets, but duplicate budgets (budgets with the same identity in terms of **amount**, **date** and **category**) are not allowed. Upon setting the budget, making

OutTransaction will deduct the amount from relevant budgets in the list. The detailed implementation of the process of updating the budget is explained further below in Current Implementation.

4.1.1. Current Implementation

The set command is an extension of parent Command class, facilitated by the *Logic* and *Model* components of the application, *PalPay.*
Given an **amount** and **date**, a new Budget is set for the user.
Upon setting a new budget, a BudgetCard is created and displayed in a list in the application window till the date set by the user.

A Budget stores an **initial amount**, **amount** (the current amount), **deadline**, **categories**. There is a need for a Budget to store both **initial amount** and **amount** as it allows for percentage of budget remaining to be calculated.
Shown below is the class diagram of Budget class:

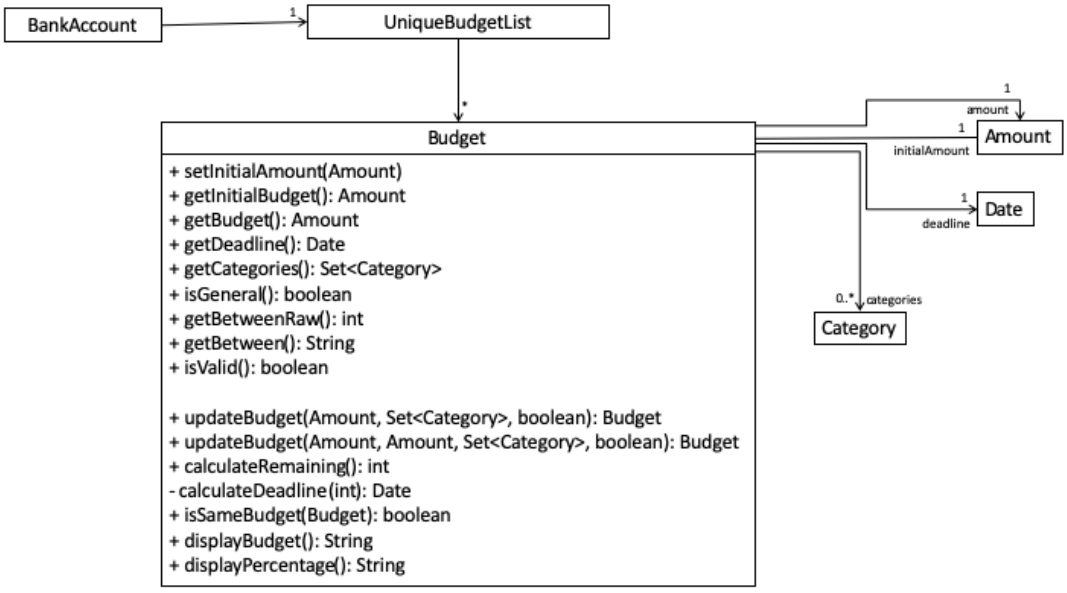

Figure 7. Class diagram of Budget class

Displaying the percentage remaining improves the user experience greatly as our target user is a

visual person who wants to see how much budget he has left in each category so as to cut down on spending as necessary as specified in the User Story. Hence, taking a quick glance at the Budget card allows the user to determine how much of budget he has left, as well as be alarmed by the red font colour to spend less if he has overspent beyond the budget set.

A snippet of the code which calculates the percentage of budget left is shown below:

```
public String displayPercentage() {
    double percentage = this.amount.divideAmount(this.initialAmount) * 100;
    if (percentage < 0.00) {
        percentage = 0.0; // should not display a negative percentage
    } else if (percentage > 100.00) {
        percentage = 100.0; // should not display a percentage greater than 100%
    }
    return String.format("%.2f%% remaining", percentage);
}
```

Moreover, as our user is a visual person, PalPay makes use of colour to display different messages. For instance, budget is displayed in red to alert the user that he has overspent beyond the set budget.

Shown below is an example of an overspent budget:

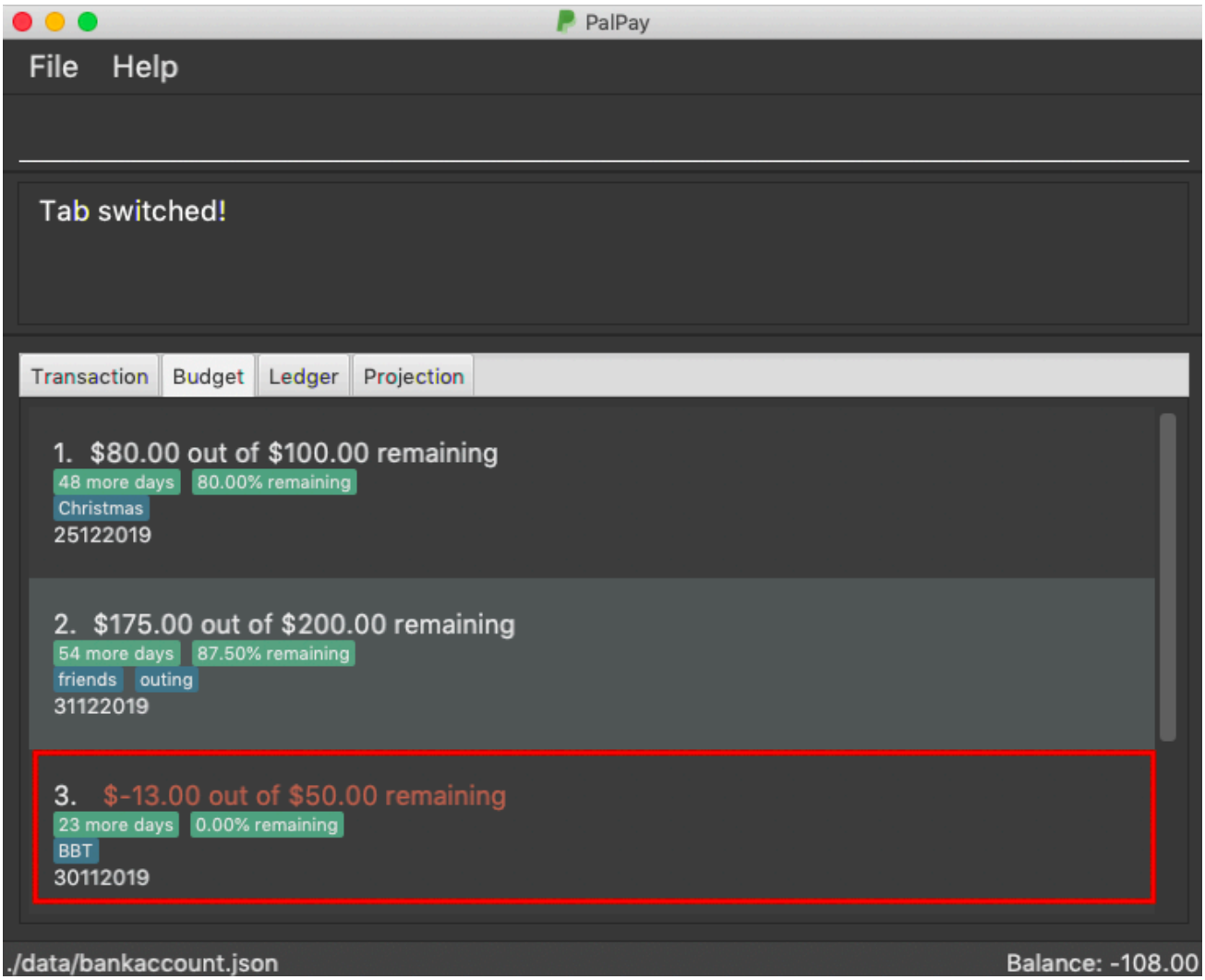

Figure 8. Example of an Overspent Budget

When setting a new Budget, existence of a duplicate budget is checked through a sequence of checks. The activity diagram below shows the activity diagram of setting a new budget:

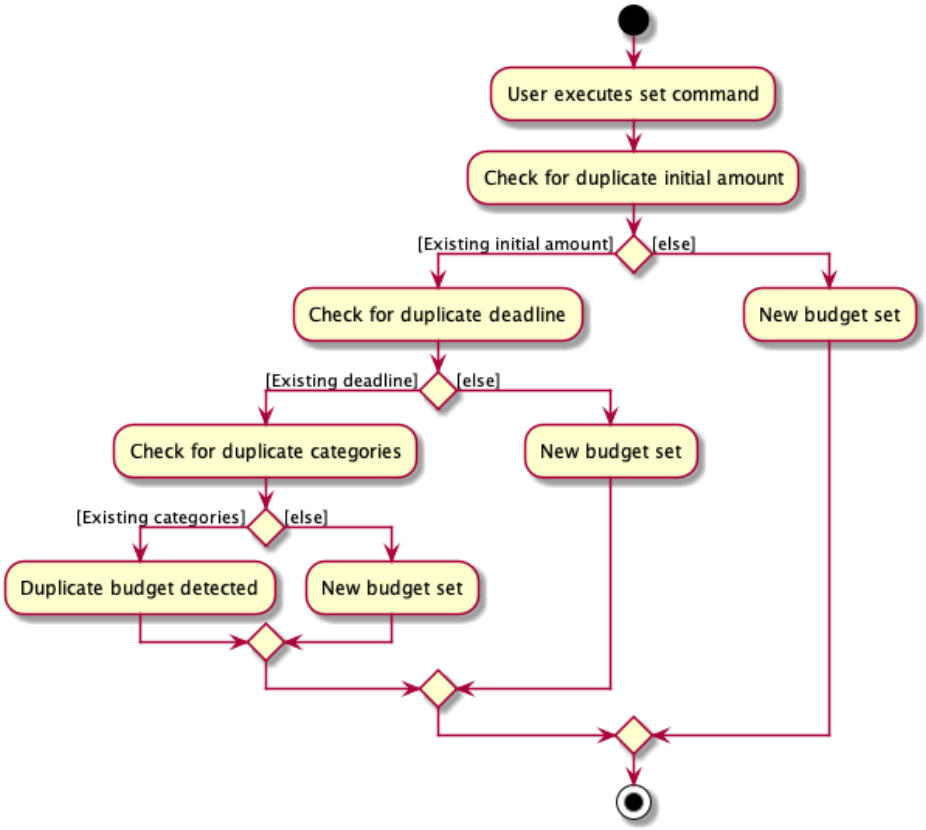

Figure 9. Activity Diagram of setting a New Budget Successfully

As shown, a new budget cannot have the same **initalAmount**, **deadline** and **categories** as any other existing budget in budget list. Allowing existence of duplicate budgets will crowd the interface of Budget tab, which prevents the user from getting a quick overview of his budget status. Hence, a duplicate check is essential in providing a pleasing user experience.

4.1.2. Example of Usage

Given below is an example usage of how set behaves at each step.

Step 1. The user executes set $/100 d/31122019 c/shopping to set a new budget of $100 until 31st December 2019 under the **category** shopping.

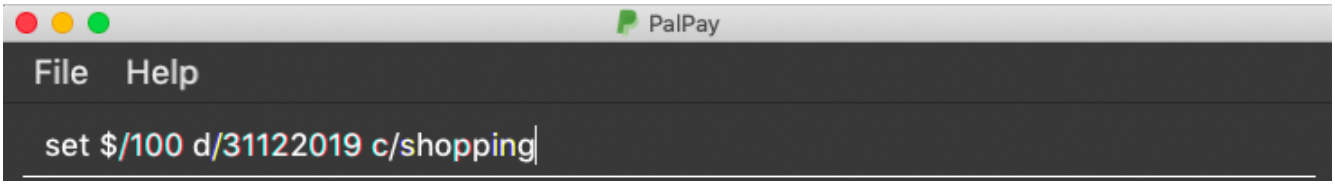

Figure 10. User Inputs set $/100 d/31122019 c/shopping

Step 2. Upon executing the command, LogicManager uses MainParser#parse to parse the input from the user.

Step 3. MainParser determines which command is being executed and creates SetCommandParser to further parse the input.

Step 4. SetCommandParser parses the argument and checks if it is valid. If it is invalid, an exception is thrown. Else, it returns a SetCommand.

Step 5. LogicManager uses SetCommand#execute() to add a new budget.
SetCommand uses ModelManager#has(Budget) to check if it is a duplicate of an existing budget in the UniqueBudgetList as shown above in the above diagram.

Step 6. SetCommand uses Model#commitUserState() to save the latest state of the application. It then returns a CommandResult to the LogicManager and the result will be displayed to the user at the end.

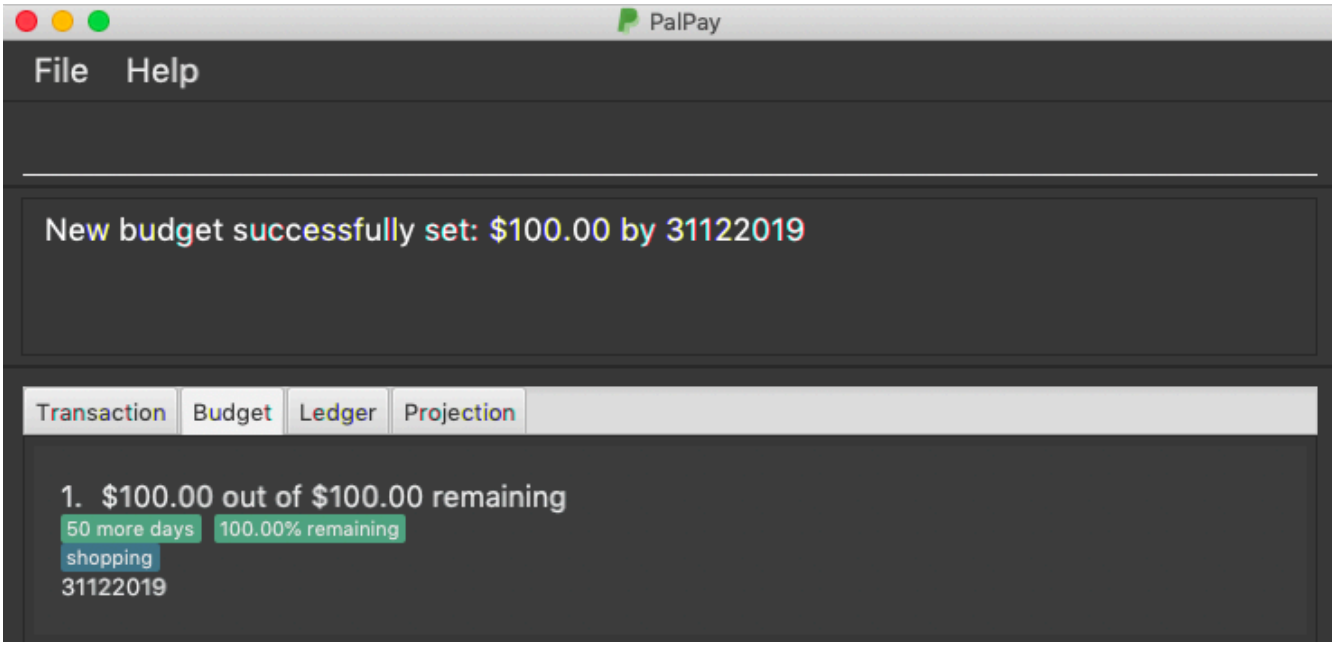

Figure 11. New Budget Successfully Created

4.1.3. Design Considerations

Currently, Budget does not extend from Transaction although the two behave in a similar way. There is an aggregation between Budget and Transaction as the two can exist independent of each other, although an effect on one may also cause an impact on the other. The current design was chosen over the former design of inheritance as there is a stark difference in the two in a way that Budget does not affect the balance of the user's bank account directly while Transaction does. Hence, by Liskov Substitution Principle, inheritance is not a suitable design.

<End of Excerpt>