

Project Portfolio - Ho Si Shi Annette

1. Overview

This portfolio describes the contributions that I made to my school project, [WalletCLI](#). As a team with 3 other CS2113T Software Engineering students, our group implemented a Command Line Interface application, called **WalletCLI**. **WalletCLI** primarily caters to NUS students and staff who prefer to use a desktop application for managing their monthly expenditure. Users can store their records of their expenses and loans with **WalletCLI**. In addition, **WalletCLI** provide features for users to keep track of their monthly budget and analyse their expenditure through statistics. The application is written in Java and compiled into a jar file for users to run and use.

My main contributions for the project were to design and code the contact management, import data, export data and help section features. In the following sections, these features are being explained in more detail and the last section will describe the contributions I made for the user and developer guides.

2. Summary of Contributions

This section shows a summary of contributions which I made for **WalletCLI**.

2.1 Code Contribution

Lines of codes that I wrote can be viewed from this link: [Contribution Report](#)

2.2 Features Implemented

Contact Management

- **What it does :** With add contact, edit contact and delete contact commands, users can manage their contact list. Contacts are used to tag each loan record with the contact details of the person whom they borrow from or lend money to.

Import Data

- **What it does:** The import command allow users to transfer records of their loans or expenses from CSV files to WalletCLI.
- **Justification:** Users can have the choice to store records in csv files instead, but still able to utilise **WalletCLI** statistics feature to have a quick analysis of their monthly expenditure by importing their data.
- **Resource Credit:** [OpenCSV](#)

Export Data

- **What it does:** The export command allow users to transfer records of their loans or monthly expenses from **WalletCLi** to CSV files.
- **Justification:** Users can utilize **WalletCLi** to track group expenses that they are in charge of. Examples of such expenses can be household expenses, holiday expenses shared with friends or event and project budgeting. In view of this, this feature will be useful when users want to send a copy of the data to other people.
- **Resource Credit:** [OpenCSV](#)

Help Section

- **What it does:** The help command allow users to see the different command syntaxes, command description and usage examples.

2.3 Other Contributions

Integrate Tools




- Integrated [OpenCSV library](#) to project (for export/import feature) : [#97](#)
- Integrated [Gradle Shadow](#) to compile project code into jar : [#42](#)
- Added License for project code : [#64](#)

Community

- Reviewed Github Pull Requests (examples: [#78](#), [#84](#), [#103](#), [#106](#), [#127](#), [#135](#))
- Resolved bugs in other application features (examples: [#63](#), [#143](#), [#144](#))
- Reported bugs and suggestions for another team (examples: [1](#), [2](#), [3](#))

3. Documentation Contribution

This section will show my documentation contributions and showcase my documentation skills through some samples from sections that I wrote. As the samples are abbreviated, you may read the full content of all sections I wrote at the [User Guide](#) and [Developer Guide](#). Some symbols will be found in the samples and this table will explain their purpose:

	This is a tip for user to use WalletCLi more easily.
	This is a note for users to take note of when using WalletCLi .
	This is a rule that users need to follow to ensure proper usage of WalletCLi .

3.1 User Guide

List of sections that I individually wrote: Viewing help, Listing data (list contact), Adding data (add contact), Editing data (edit contact), Deleting data (delete contact), Export, Import, Contact Management Commands, Porting Data Commands.





Sample 1. Adding data (add contact)

The following command adds a single contact that consists of the name Mary, a description of Mary being your sister and her phone number, i.e. number 8728 1831.

Command: add contact mary /p 8728 1831 /d sister

```
add contact mary /p 8728 1831 /d sister
Got it. I've added this contact.
-----
| ID | Name      | Phone    | Detail |
-----
| 9  | mary     | 8728 1831 | sister |
-----
```

Figure1: Adding contacts

	WalletCLI does not check if the phone number input is just only digits, thus allowing you to input phone number in your preferred format e.g. (+65)8543 2124, 98765432, 6543-2315
	The optional parameters (/p and /d) do not need to be keyed in a specific order. E.g. add contact Mary /d sister /p 8728 1831 will produce the same output as shown in the Figure 5.3.1.
	If you keyed in /p and /d without any arguments, WalletCLI will leave the phone and details as empty.
	WalletCLI currently allow duplicate contacts.

Sample 2. Import

You can create expenses or loan records in csv files and import them from your **WalletCLI** home directory into the application. Sample files can be found at this [link](#) and you can try importing them into **WalletCLI**. Command Format: import <loan OR expense> <FILENAME>

data		26/10/2019 4:50 PM
importExpenses	Store your csv files in home directory before import	26/10/2019 1:56 PM
importLoans		26/10/2019 1:56 PM
WalletCLI-expenses-07.11.2019-12.04.04		7/11/2019 12:04 PM
WalletCLI-loans-07.11.2019-12.02.57		7/11/2019 12:02 PM
WalletCLI-v1.4	Jar file	7/11/2019 8:30 AM

Figure 2.1: Sample Files in Home Directory

Example:

- import loan importLoans.csv

Importing loans from sample csv file, importLoans.csv.

```

Importing records...
Got it. I've added this contact:
[ID: 5]Lauren
Got it. I've added this loan:
[ID: 5][Settled][Lend] i»¿lunch Amount:$30.0 Date:01 Oct 2019[Contact: [ID: 5]Lauren ]
Got it. I've added this contact:
[ID: 6]Ben brother
Got it. I've added this loan:
[ID: 6][Not Settled][Borrow] dinner Amount:$15.5 Date:02 Oct 2019[Contact: [ID: 6]Ben brother ]
Got it. I've added this contact:
[ID: 7]Jane 90181829
Got it. I've added this loan:
[ID: 7][Settled][Borrow] breakfast Amount:$1.5 Date:15 Oct 2019[Contact: [ID: 7]Jane 90181829]
Got it. I've added this contact:
[ID: 8]Ryan Tang (+65)81731829
Got it. I've added this loan:
[ID: 8][Not Settled][Lend] supper Amount:$25.0 Date:23 Oct 2019[Contact: [ID: 8]Ryan Tang (+65)81731829]
Finish Import!

```

Figure 2.2: Importing Loans

⚠	Key in only the filename and csv extension for FILENAME e.g. marExpenses.csv
⚠	File name should not contain any spaces
i	When importing loans, WalletCLI will create a new contact in the application based on the contact information detected in each row of the csv file. <i>Merging of duplicate contact records will be implemented in future releases.</i>
i	As seen in Figure 5.14.2, due to some encoding issues, extra characters like i»¿ may appear when data is imported.

The next section explains how expenses records should be formatted in csv (comma-separated values) before importing into **WalletCLI**.

A	B	C	D	E	F
shirt	7/10/2019	17	SHOPPING		
supper	9/10/2019	13.5	FOOD	yes	MONTHLY
phone bill	30/10/2019	2.6	BILLS		

Figure 2.3: Expense Records CSV in Excel View

For each row of expense record, it should be formatted in this order:

1. Description
2. Date(dd/MM/yyyy)
3. Amount (digits)
4. Category (in capital letters). The following categories are allowed:
 - a. FOOD
 - b. TRANSPORT
 - c. BILLS
 - d. SHOPPING

- e. OTHERS
- 5. (Optional) Recurring Record: Indicate a recurring record by keying in yes.
- 6. (Optional) Frequency: If record is recurring, key in the frequency in capital letters.
Choose one of the following:
 - a. DAILY
 - b. WEEKLY
 - c. MONTHLY

⚠	Recurring Record requires a Frequency specified on the same row.
⚠	The input for Recurring Record should be only yes. Any other values keyed in for Recurring Record and Frequency will be ignored and the record will be treated as a non-recurring expense when importing into WalletCLI .
i	For Windows Users, if you wrote the csv in Microsoft Excel, do double-check the format of Date in text editors, e.g. Notepad. By default, Excel may store the value of Date in d/MM/yyyy format instead.

3.2 Developer Guide

List of sections that I individually wrote: Managing Contacts, Help, Export Data, Import Data

Other sections that I made substantial contributions to: Use Cases, Instructions for Manual Testing (specifically for features I designed)

Sample 1. Adding Contact Implementation (under Managing Contacts) [Sequence Diagram]

The following sequence diagram shows the implementation of adding contacts. Example command used here is add contact Mary Tan /p 8728 1831 /d sister.

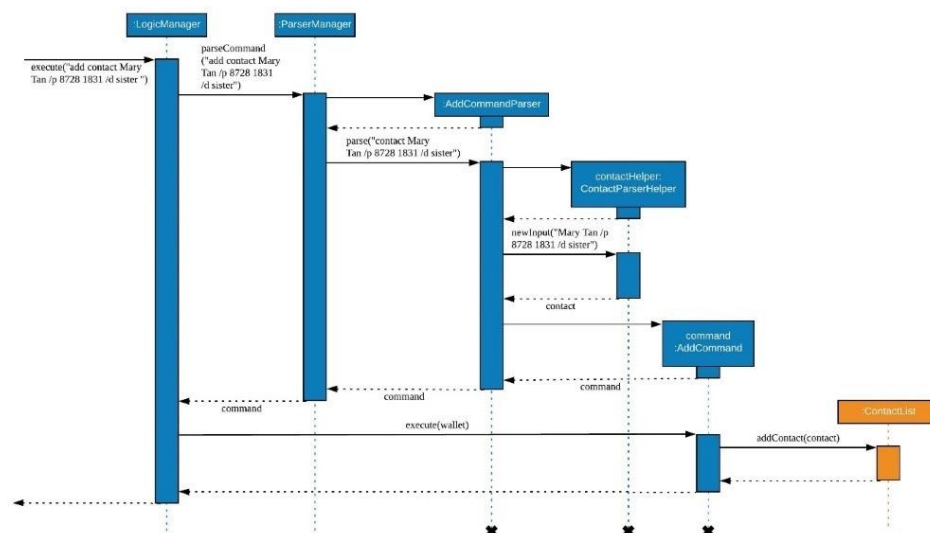


Figure 5.X.2.1 Sequence diagram for Add Contact

The steps below will explain the sequence diagram shown above:

1. At the Logic Component, AddCommandParser will parse the command and calls ContactParserHelper which processes the input into a contact object with Name, Phone Number, and Details according to the following rules:
 - a. Name must not be empty. In this case, it would be Mary Tan.
 - b. Process any arguments after /p and /d into Phone Number and Details respectively. In this case, it would be 8728 1831 and sister respectively processed.
 - c. If there is no /p, Phone Number will be set as a null value. The null value also applies if /p exists but does not have any arguments.
 - d. If there is no /d, Details will be set as a null value. The null value also applies if /d exists but does not have any arguments.
2. If the input is successfully processed, a command object which includes the contact object is returned to LogicManager.
3. LogicManager executes the command object, which adds the contact into ContactList.

i	At Step 2a, if Name is empty, the steps after will not be executed and an error message will display.
i	Users can include spaces in their command arguments and the optional command line arguments (/d and /p) do not need to be entered in a particular order. ContactParserHelper will process the input accordingly.

Sample 2. Import Data [Sequence Diagram]

The following sequence diagram shows the main flow of importing data. Example command used here is `import loan importLoans.csv`.

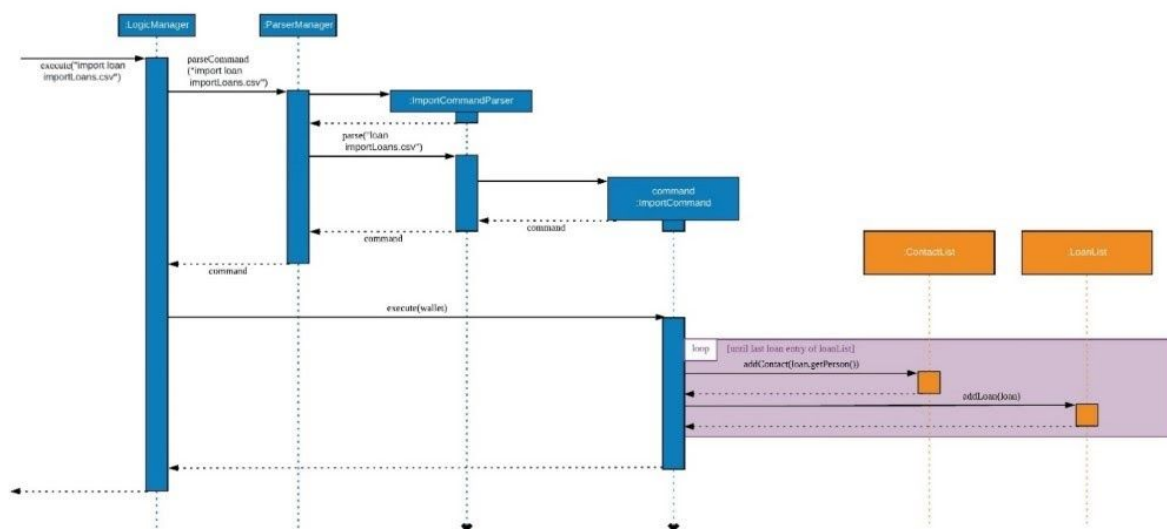


Figure 5.X.1 Sequence diagram for Import Data

The steps below further explain the sequence diagram and logic of code:

1. At the Logic Component, `ImportCommandParser` will parse the command to check if it is valid.
2. `ImportCommandParser` will attempt to find the file based on the `FILENAME` provided by user by searching for it in the directory from which the application is running. In this case, it would be `importLoans.csv`.
3. If file is found, using `CSVReader` from a third-party library, `OpenCSV`, `ImportCommandParser` will read each record from each row of the file and checks if it is formatted correctly.
4. `ImportCommandParser` will save the retrieved records into a list, called `loanList`.
5. `ImportCommandParser` returns a command object to the `LogicManager`. In the command object, `loanList` is stored.
6. `LogicManager` executes the command object.
7. As shown in the diagram above, new contacts and loans will be created from `loanList`. The new records are added to `ContactList` and `LoanList`, which stores the contact and loan data respectively. In the event if the data type was expenses instead, new expenses will be added to `ExpenseList` instead.



At Step 2 and 3, error messages will display if the file can't be read or if the records in the file is formatted incorrectly. The process will be terminated and steps after will not be executed.

Sample 3. Help

[General Instructions and Future Implementation]

To provide an easier way for developers to update the help sections in **WalletCLI**, each help section has its own content stored in a text file under `/src/main/resources`. **WalletCLI** will load all content from the text files into in-app memory at the start of the application by referencing the file names from `/src/main/resources/helppaths.txt`. The following shows the current list of in-app help section indexes and names against the text file **WalletCLI** will retrieve:

Index	Section Name	Text File Name
1	General	general.txt
2	Expense	expense.txt
3	Loans	loan.txt
4	Contacts	contact.txt
5	Command History	cmdhistory.txt

For each command in the text files, most of the content is formatted with `[field] [value]` format, delimited by pipeline character (`|`). The following shows the content and formatting required for each command in the text files.

1. Header
 - a. Format: `--[Command Header]--`

- b. Example:


```
--Add Loan--
```
2. Command Syntax
 - a. Format: command | [syntax]
 - b. Example:


```
command | add loan <DETAILS> <AMOUNT> [DATE] </l OR /b>
```
3. Description of Command
 - a. Format: desc | [description]
 - b. Example:


```
desc | add new loan entry
```
4. (Optional) For parameters which need further explanation or need a specific format
 - a. Format: [parameter] | [explanation or format]
 - b. Example:


```
DATE | dd/mm/yyyy
      i. /l | use this if entry indicates the amount you lend
      ii. /b | use this if entry indicates the amount you borrow
```
 - c. Note: Insert these contents between Command Syntax and Description of Command

Future Implementation

These are features considered for future implementation:

1. Admin console to update content in help section.
2. Provide additional option for user to get help for a specific command instead of finding and reading off a section.

Sample 4. Export Data

[Design Considerations]

The following section explains the design considerations for export feature in the aspect of how exported files should be named. Alternative 2 was chosen as it was easier to implement.

- **Alternative 1: Use an id to name an exported file.**
E.g. **WalletCLi-expenses-id000001.csv**

Pros: Users can differentiate the files being exported to their **WalletCLi** home directory. This can prevent OpenCSV from overwriting files with identical names, since all filenames are now unique.

Cons: ExportCommand must always generate a unique id for the file. Id cannot be reused.

- **Alternative 2: Use date and time to name an exported file.**
E.g. **WalletCLi-expenses-08.11.2019-10.11.33.csv** **[Current Implementation]**

Pros: Users can differentiate the files being exported to their **WalletCLi** home directory. Users can see the file creation date and time from the file name.

Cons: During code implementation, must ensure that the local date and time is correctly retrieved in order to name the file.