Test Plan Expense Tracker

Radhika Sharma
Alexander Jackson
Zachary Bazen

McMaster University

Revision 1 - December 8th 2015

Contents

1 Revision History 4

General Information 5

2 Summary 5

3 Environment and Pretest Background 5

4 Test Objectives 5

4.1 Reliability 5

4.2 File Integrity 5

4.3 Access Control 5

4.4 Correctness 5

4.5 Ease of Use 6

4.6 Portable 6

4.7 Efficiency 6

5 Expected Defect Rates 6

6 References 6

6.1 Project request authorization 6

6.2 Previously published documents on the project 6

6.3 Documentation concerning related projects 6

Plan 6

7 Software Description 6

8 Test Team 7

9 Milestones 8

10 Budget 8

11 Testing 8

11.1 Schedule 8

11.2 Requirements 9

11.2.1 Equipment 9

11.2.2 Software 9

11.2.3 Personnel 9

12 Testing Material 10

12.0.1 System Documentation 10

12.0.2 Software to be tested and its medium 10

12.0.3 Test Inputs 10

12.0.4 Test Documentation 11

12.1 Test Training 11

12.2 Tests to be Conducted . . . 11
13 Testing (System Checkpoint) . . . 11
Specification and Evaluation . . . 11
14 Specifications . . . 12
14.1 Business Functions . . . 12
14.2 Structural Functions . . . 12
14.3 Test/Functions Relationships . . . 12
14.4 Test Progressions . . . 13
15 Methods and Constraints . . . 13
16 Evaluation . . . 15
16.1 Criteria . . . 15
16.2 Data Reduction . . . 15
Test Descriptions . . . 15
17 Testing File Input and File Output . . . 15
17.1 Expense Tracker Main . . . 15
17.2 Inputs . . . 15
17.3 Outputs . . . 16
17.4 Procedures . . . 16
18 Testing Addition of Entries . . . 16
18.1 Control . . . 16
18.2 Inputs . . . 16
18.3 Outputs . . . 16
18.4 Procedures . . . 16
19 Testing Modification of Entries . . . 17
19.1 Control . . . 17
19.2 Inputs . . . 17
19.3 Outputs . . . 17
19.4 Procedures . . . 17
20 Testing Search Functionality . . . 17
20.1 Control . . . 18
20.2 Inputs . . . 18
20.3 Outputs . . . 18
20.4 Procedures . . . 18

21 Testing Sort Functionality 18
21.1 Control 19
21.2 Inputs 19
21.3 Outputs 19
21.4 Procedures 19
22 Testing GUI 19
22.1 Control 19
22.2 Inputs 19
22.3 Outputs 19
22.4 Procedures 20
23 Testing Ease of Use 20
23.1 Control 20
23.2 Inputs 20
23.3 Outputs 20
23.4 Procedures 20
24 Testing Boundary Cases and Exception Handling 21
24.1 Control 21
24.2 Inputs 21
24.3 Outputs 21
24.4 Procedures 21

List of Tables

2 Class input output description 7
3 Test Schedule 9
4 Test Input Material 11

1 Revision History

Revision #	Revision Date	Description of Change	Author(s)
Revision 0	October 18, 2015	Document Outline	Zachary Bazen
Revision 1	October 19, 2015	Sections 7-11	Zachary Bazen
Revision 2	October 19, 2015	Edits	Alexander Jackson
Revision 3	October 19, 2015	Sections 2-6	Radhika Sharma
Revision 4	October 20, 2015	Sections 17 -22	Alexander Jackson
Revision 5	October 20, 2015	Sections 11-13	Zachary Bazen
Revision 6	October 21, 2015	Section 14 and Edits	Radhika Sharma
Revision 7	October 22, 2015	Edits to Sections 17-22	Alexander Jackson
Revision 8	October 22, 2015	Sections 15-16	Radhika Sharma
Revision 9	December 7, 2015	GUI Test Plan	Radhika Sharma

General Information

2 Summary

The Expense Tracker will enable companies to efficiently record costs of current and past projects. The application will allow companies to add entries for new projects, search for transactions and costs pertaining to a certain project, as well as modifying existing records.

3 Environment and Pretest Background

The project was originally conceived due to the lack of software available for the aforementioned issues. Often times, companies use old fashioned paper records to keep track of expenses, however this method of book keeping allows for human error and is not efficient for searches. On-line banking systems are better at tracking expenses than a paper system, however searches are difficult to perform. On-line banking systems also do not account for cash transactions.
The user organization will encompass all small scale companies who wish to track expenses efficiently.

The testing environment will be on any computer that is able to run the Java programming language.

No prior testing has been completed.

4 Test Objectives

4 Reliability

The testing must ensure that the program is reliable for use. The program will perform the prescribed functions under normal conditions. This will also mean that the program can run with minimal maintenance down time.

4 File Integrity

The program must access the correct files when doing searches, modifications and additions.

4 Access Control

The program must not access undesired files on the user's computer. The user must not be able to modify the program itself through the command line.

4 Correctness

The program must be able to interpret the user information entered, and perform the task that the user has specified, provided the user has inputed the correct commands.

4 Ease of Use

Any user who has a basic knowledge of computers and command line operations will be able to use the program. Assistance will be provided through the command line as well as an accompanying text file describing the commands. The combination of the in program help as well as the help text file should ensure that any user will find minimal hardships when using the program.

4 Portable

The application must be usable on any computer that is able to run the Java programming language.

4 Efficiency

The program should be able to sort the information (based on the user preference) in a reasonable amount of time. The program should be able to perform searches on the existing information in a reasonable amount of time. For the purposes of this document, a reasonable amount of time will be defined as 10 seconds.

5 Expected Defect Rates

The application is expected to make 10 errors for every 100 operations/commands.

6 References

6 Project request authorization

Project request authorization was issued during the week of September 16, 2015 and authorization was granted the same week.

6 Previously published documents on the project

Expense tracker has no previously published documents.

6 Documentation concerning related projects

Expense tracker is unique and has few to none related projects. Often the functionality this program gives is buried in a large program that has its own documentation.

Plan

7 Software Description

Class Input and Output Description

Function	Description	Input	Output

Add	Allows the user to add a new entry to the existing entries.	The information the user provides for each field of the entry	The new updated internal array or the updated text file after exiting program. An action complete message would also be output to the user
Modify	Allows users to modify fields of an existing entry	Updated information for a field of an entry	The new updated internal array or the updated text file after exiting program. An action complete message would also be output
Search	Allows the user to input up to two search conditions and searches for entries based on the conditions.	Fields to be search upon and data that needs to be found	All entries containing the search conditions. When a search returns null an element not found message will be displayed.
Sort	An internal function that will be used to assist search and modify functions.	Internal representation of the data with parameter to indicate which field to sort on.	Internal representation of the data that will be new in new sorted format.

Table 2: Class input output description

8 Test Team

The test team is comprised of the following members: Radhika Sharma, Alexander Jackson, and Zachary Bazen.
The following are the test assignments:

- Zachary Bazen
  - White Box Testing
  - Unit Testing
  - Black Box Testing
- Radhika Sharma
  - Boundary Case Testing
  - User Testing
  - Black Box Testing
- Alexander Jackson
  - Error Handling Testing
  - Software Requirements Testing
  - Black Box Testing

9 Milestones

- Program is able to read data from UTF - 8 encoded comma separated value file
  Testing week October 26, 2015
- Program can read from the console for input indefinitely until exit command is issued
  Testing week of October 26, 2015
- Internal sort works on multiple keys
  Testing week of October 26, 2015
- User can search for a single entry or a range of entries based on what the user has specified
  Testing week of November 2, 2015
- User can modify a specific field of an entry in the file
  Testing week of November 9, 2015
- User can add an entry
  Testing week of November 9, 2015
- Program is able to output data to a UTF - 8 encoded file
  Testing week of November 16, 2015
- Program is able to handle exceptions
  Testing week of October November 16, 2015

Items will be tested as the software becomes available. All tests will be completed between October 27, 2015 - November 27, 2015.

10 Budget

This project does not require budgeting because it requires minimal resources.

11 Testing

11 Schedule

Test Schedule

Element being Tested	**Test Type**	**Test Date**	**Location**
Expense Entry	Unit Test Black Box	October 30, 2015	Office
FInput	Unit Test Black Box	October 30, 2015	Office
Expense	Unit Test Black Box	October 30, 2015	Office
Main	Unit Test Black Box	October 31, 2015	Office

Sort	Unit Test White Box	October 31, 2015	Office
Search	Unit Test Black Box	November 6, 2015	Office
Add	Unit Test Black Box	November 13, 2015	Office
ModifyEntry	Unit Test Black Box	November 13, 2015	Office
Output	Unit Test Black Box	November 20, 2015	Office
Exception Handling	System Test Boundary Case Error Handling	November 20, 2015	Office
Usability	User Testing	November 23, 2015	To be Determined
Correctness	Software Requirements Test Black Box	November 27, 2015	Office

Table 3: Test Schedule

11 Requirements

11.2.1 Equipment

The equipment required for testing purposes is a computer. No additional equipment is required to run the Expense Tracker. The expected period of use is 15 hours for testing purposes alone.

11.2.2 Software

The Java programming language as well as the JUnit testing application is required to test the program. The program saves the transactions in a CSV file; therefore CSV processing software will also be required for testing purposes. No additional support software is needed for testing purposes.

11.2.3 Personnel

The personnel required for testing are the three team members, Radhika Sharma,Alexander Jackson, and Zachary Bazen. It is required that the members must have sufficient knowledge of the Java programming language such that they can read the code, understand it, and test the internal components as well as the external components. It is also required that the

members have basic computer skills. Other personnel required include people who wish to use the application (defined previously as small business owners). Users will be required to test the ease of use of the application, and therefore are also required to have basic computing skills such as how to use a command line.

12 Testing Material

12.0.1 System Documentation

The software requirements specification is a required for testing because it will allow for software developers to trace the program to its requirements, ensuring that it is doing what it is supposed to do as outlined by the requirements.

12.0.2 Software to be tested and its medium

The entire Expense Tracker source code will be needed during testing. All classes and methods will be tested during the duration of the tests in an effort to achieve comprehensive statement coverage. The testing medium is the Java console window and J - Unit Test Suite in addition to the Eclipse IDE. Ease of Use will be tested by allowing users to interact with the program and its commands.

12.0.3 Test Inputs

Test input material required for Expense Tracker is a comma separated value file that contains expense entries. All operations completed in expense tracker will use or manipulate the data that is stored in the file.
Test input material required for each class of expense tracker is listed in the following table.

Test Input Material

Class	**Input Material**
Main	Console input from users as strings
ExpenseEntry	Expense field entries as strings
AddEntry	Expense field entries as strings Array entry location as integer
Expenses	ArrayList with elements of type ExpenseEntry
Sort	Sort parameters as strings ArrayList with elements of type ExpenseEntry
Search	Search parameters as strings
ModifyEntry	Entry ID as an integer Expense field entries as strings

FInput	File path as a string
Output	ArrayList with elements of type ExpenseEntry

Table 4: Test Input Material

12.0.4 Test Documentation

The test plan document will be crucial for testing purposes. It outlines what should be tested and by what methods. Using a test plan will ensure that the testing process is efficient and that all aspects of the application are tested.

12 Test Training

Those testing the program will need to be trained on how to enter file paths correctly and how to find correct file paths for the particular file they wish to open in the program. Training may also be required to show users how to respond to prompts when the program is running. The training will be provided on site at the time of testing. Personnel to be trained will include users (previously defined as small business owners who wish to use the program).Training staff will include the software developers who created the program.

12 Tests to be Conducted

The types of testing that the Expense tracker will undergo include:
Unit Testing
White Box Testing
Software Requirements Testing
Ease of Use Testing
Boundary Case Testing
Error Handling Testing
Unit testing
These tests will be used to ensure each class works as anticipated. White box testing will test overall program correctness, boundary cases and error handling. Ease of use testing will be conducted by select personnel to test the ease of use of program.

13 Testing (System Checkpoint)

The next system check point will take place after the project has been fully developed and optimized. At this stage ease of use will be tested to see if requirements are met. Overall product correctness will also be verified at this point.

Specification and Evaluation

14 Specifications

14 Business Functions

Any client user with minimal training in the use of computers must be able to use the program.

14 Structural Functions

The user can specify one of three actions, either search, modify or add.The program lets the user chose actions by providing a list of numbers and the actions that they correspond. The user must then input the number corresponding to the action they wish to execute.

If the user chooses to search for an entry, the program asks the user for a parameter to search on and what string to search for, the input class must pass this information to the search class.The search class must search for the specified transaction, and send this information to the output class which will output it to the console.

If the user chooses to modify an entry, the program must ask the user for a parameter to search on and a string to search for, the input class must interpret the information and send it to the search class. The search class must find a list of transactions that match the criteria that the user specified; this information must then be passed to the output class. The output class must display the list of transactions to the user.The user must then specify the specific transaction to be modified, the parameter to be modified, and what the modification should be. The input class must send the information to the modify class where the master arrayList will be modified.

If the user chooses to add an entry, the user will be asked by the program to enter all the parameters into the console. This information is read by the input class and must be send to the add class where it will be added to the master array as a new entry.

14 Test/Functions Relationships

- Black Box Testing - The system will be tested using black box testing to ensure that the program is able to read user input correctly and is able to output information to the user. Black box testing will also be used to ensure that the program is able to handle errors by displaying an error message to the user when incorrect information is provided. If these functions do not work, the rest of the program cannot work either.

- Unit Testing - Unit testing will be used to test all of the basic functions of the program. This will include: adding entries, saving changes to existing entries (modifications), and whether the system is able to correctly search for an entry as specified by the user. This relates to the major portions of the program and helps to test that the program works as desired.

- White Box Testing - The system will be tested using white box testing to ensure that the sort functionality works as desired. The sort function is an internal function that the user does not interact with, therefore white box testing must be used. This relates to the structural functionality of the system because to search for entries efficiently, the system must first sort the entries on a specified parameter. White box testing will also be used to test the boundary cases of the system, such as how the system handles when a transaction entered has a zero value.

- User Testing - The system will be tested by users to ensure ease of use. This related to all portions of the system because it is essential to the success of the program that the users are able to intuitively execute commands for the program.

14 Test Progressions

To ensure that the testing is done efficiently, the input and output functions must be tested first. If the program cannot read input correctly, then all other functions cannot be executed, therefore it is essential to test that the program can handle input correctly. As mentioned earlier in the document, the input and output functions will be tested using Black Box testing, therefore this must be done first.

Once it has been proven that the input can be read correctly, the basic functionality of the program (search, modify, and add) can be tested. As mentioned before, unit testing would be used to test the basic functionality, therefore intuitively, this is next type of testing that will be done. The unit testing must prove that the functions can take the input, perform the desired task, and output the correct output to the user.

Once it has been proven that the program runs as expected, the system can then be tested to see if the internal mechanisms work as expected. For this project, the internal mechanism that must be tested is the sort function. In addition to the sort function, the program must be tested for boundary cases and exception handling at this stage. All this will be accomplished using White Box testing.

After all the different components of the system have been tested, the entire system can then be tested for ease of use. Ease of use will be verified with user testing (having potential users try to add entries, modify existing entries, and search for existing entries in the system).

15 Methods and Constraints

Black Box Testing

- Methodology - The strategy for this type of testing, in terms of this project, is to try to ensure that the code does what it asked without concern of how it does it.
- Test Tools - Any computer that can run the Java programming language, as well as the JUnit testing application.

- Extent - This testing will be total testing. All functions will be tested using black box testing and will cover all domains for each function
- Data Recording - This type of testing will be recorded using a text document. The problem description, actual results, expected results, effect of deviation, cause of problem, location of problem, and recommended action will be recorded.
- Constraints - There are no anticipated limitations on this type of testing.

Unit Testing

- Methodology - The strategy for this type of testing, in term of this project, is to try to ensure that each function (add, search and modify) is able to run correctly on its own.
- Test Tools - Any computer that can run the Java programming language, as well as the JUnit testing application.
- Extent - This testing will be total testing. All functions will be tested as a unit and will be tested over all domains for each function.
- Data Recording - This type of testing will be recorded using a text document. The problem description, actual results, expected results, effect of deviation, cause of problem, location of problem, and recommended action will be recorded.
- Constraints - There are no anticipated limitations on this type of testing.

White Box Testing

- Methodology - The strategy for this type of testing, in terms of this project is to ensure that the internal mechanisms of the project work correctly. It is imperative that points in the program where the functions interact with each other be tested for weaknesses.
- Test Tools - Any computer that can run the Java programming language, as well as the JUnit testing application.
- Extent - Partial coverage is expected for this type of testing. This is due to the extent of the program. To test all internal mechanisms of the program within the given time constraints, is not feasible, therefore only points of interest (where functions interact) will be tested.
- Data Recording - This type of testing will be recorded using a text document. The problem description, actual results, expected results, effect of deviation, cause of problem, location of problem, and recommended action will be recorded.
- Constraints - Time is a constraint for this type of testing.

User Testing

- Methodology - The strategy for this type of testing, in terms of this project is to ensure that the specified users are able to intuitively use the program and its commands.
- Test Tools - Any computer that can run the Java programming language, as well as the target client.
- Extent - This testing will be partial. Although the program may be fairly usable for most users, it is impossible to test that the program will be intuitive for all possible

users. This is because there are many potential users, and therefore many different skill sets to accommodate to.
- Data Recording - This type of testing will be recorded using a text document. The problem description, actual results, expected results, effect of deviation, cause of problem, location of problem, and recommended action will be recorded.
- Constraints - This type of testing is constrained by the personnel used for this test.

16 Evaluation

16 Criteria

The range of data used for the Black Box, White Box and Unit testing will be limited to the expected inputs and commands. This will include the information that the user wished to manipulate with the program. This information will be interpreted by the program as a string. Along with the strings, the user will be able to enter commands. The commands will be represented by integers. To test for exception handling, data outside the previously mentioned range will be used to purposely try to induce exception handling.

16 Data Reduction

To manipulate the results from the testing, graphs will be generated using excel to compare errors against the correctness of the program.

Test Descriptions

17 Testing File Input and File Output

Black-box testing will be used to ensure that the program can read inputs from the console and output to the console.

17 Expense Tracker Main

This test will be controlled manually, including manual insertion of inputs, manual sequencing of operations, and manual recording of results. The tester will test various inputs and commands by hand and record any and all outputs.

17 Inputs

- All commands featured in the program, and their respective parameters, to test the program's recognition of command inputs
- Inputs that do not match any commands to test the program's ability to catch improper inputs
- Commands with invalid parameters to test the program's ability to catch invalid command parameters

17 Outputs

- Listing available commands
- The program's response(s) to proper input commands
- The program's response(s) to inputs that are not valid commands
- The program's response(s) to invalid command parameters
- Any and all error message outputs

17 Procedures

- Input each available command, one at a time
- Record the program's response to each command
- Record the series of inputs and outputs that follow each command
- Verify that the program responds properly to each input
- Input various forms of invalid commands
- Verify that the program properly handled invalid input

18 Testing Addition of Entries

Unit testing will be used to ensure that the user can successfully add new expense entries.

18 Control

This test will be done semi-automatically in Eclipse using JUnit, including insertion of inputs, sequencing of operations, and comparing expected and actual results. Various sample entry additions will be pre-programmed into Eclipse along with the expected output for each.

18 Inputs

- Realistic expense entries to test that the program successfully adds new entries
- Expense entries with invalid/missing inputs to ensure that the program does not add invalid entries

18 Outputs

- The program's response(s) to adding a proper new entry
- The program's response(s) to attempting to add an invalid entry
- The program's list of entries before and after adding each entry
- Any error messages produced when attempting to add invalid entries

18 Procedures

- Attempt to add new expense entries
- View the current entries before and after each new addition
- Verify that the program added each new entry
- Attempt to add invalid entries
- View the current entries before and after each invalid addition
- Verify that the program correctly handled each invalid addition

- Verify that no invalid additions were added to the entries

19 Testing Modification of Entries

Unit testing will be used to ensure that the user can successfully make and save changes to existing expense entries.

19 Control

This test will be done semi-automatically in Eclipse using JUnit, including insertion of inputs, sequencing of operations, and comparing expected and actual results. Various sample modifications will be pre-programmed into Eclipse along with the expected output for each.

19 Inputs

- Realistic changes to a selected entry to test that the program can successfully modify existing entries
- Expense entries with invalid/missing inputs to ensure that the program does not add invalid entries

19 Outputs

- The program's response(s) to modifying a selected entry
- The program's response(s) to invalid modifications to an entry
- The program's list of entries before and after each modification
- Any error messages produced when attempting to modify an entry in an invalid manner

19 Procedures

- Attempt to select existing expense entries
- Attempt to make valid modifications to selected entries
- View the current entries before and after each modification
- Verify that the program modified the correct entries
- Attempt to make invalid modifications to selected entries
- View the current entries before and after each invalid modification
- Verify that the program correctly handled each invalid modification
- Verify that no invalid modifications were made to the entries

20 Testing Search Functionality

Proof of Concept Test

Unit testing will be used to ensure the correctness of the program's search functions.

20 Control

This test will be done semi-automatically in Eclipse using JUnit, including insertion of inputs, sequencing of operations, and comparing expected and actual results. Various sample searches will be pre-programmed into Eclipse along with the expected output for each.

20 Inputs

- Valid entry-ID-based searches to test the program's ability to return the entry with a specified ID
- Invalid entry-ID-based searches to test the program's handling of invalid IDs and IDs with no associated entries
- Valid single-attribute-based searches to test the program's ability to return entries that contain a specific attribute
- Invalid single-attribute-based searches to test the program's handling of invalid attributes and attributes not found in any existing entries
- Valid multiple-attribute-based searches to test the program's ability to return entries that contain multiple specific attributes
- Invalid multiple-attribute-based searches to test the program's handling of invalid attributes and combinations of attributes not found in any existing entries

20 Outputs

- Search result for each valid entry-ID-based search
- Search result(s) for each valid single-attribute-based search
- Search result(s) for each valid multiple-attribute-based search
- Any error messages produced when attempting an invalid search

20 Procedures

- Attempt searches based on entry ID
- View the current entries before each search
- Verify that each search returns the correct entry
- Attempt searches based on single criteria each
- View the current entries before each search
- Verify that each search returns the correct entry or entries
- Attempt searches based on multiple criteria each
- View the current entries before each search
- Verify that each search returns the correct entry or entries

21 Testing Sort Functionality

Unit testing and White Box testing will be used to ensure the correctness of the program's sort function.

21 Control

This test will be done semi-automatically in Eclipse using JUnit, including insertion of inputs, sequencing of operations, and comparing expected and actual results. Various sample sort commands will be pre-programmed into Eclipse along with the expected output for each.

21 Inputs

- Valid sort commands across all attributes to test the program's ability to sort expense entries by any attribute
- Sort commands with invalid/missing parameters to test the program's handling of invalid sort commands

21 Outputs

- The program's response(s) to attempting to perform a sort
- The list of expense entries before and after each sort
- Any error messages produced when attempting an invalid sort

21 Procedures

- Attempt to sort the current entries by each possible attribute
- View the current entries before and after each sort attempt
- Verify that the entries have been correctly sorted each time
- Input invalid sort commands
- View the current entries before and after each invalid sort attempt
- Verify that the program correctly handled each invalid sort attempt
- Verify that the arrangement of the entries has been unchanged after each invalid sort attempt

22 Testing GUI

Manual user testing to ensure that the program does the correct thing (the right windows pop up or close) when mouse clicks occur

22 Control

Testing will occur manually through use of a computer(complete with LCD screen, mouse, and keyboard) and be conducted by one of the software developers of this program.

22 Inputs

- Clicks on the appropriate buttons and menus
- Information for mock entries

22 Outputs

- The correct window pops up

- The correct windows closes
- Information inputted into the GUI is processed

22 Procedures

- Click on the jar file
- Click open and select a file
- Click on Add from the menu and input information
- Right Click on an entry, select modify and input information
- Right click on an entry, select delete
- Click on search from the menu and input data to search by
- Clear Search results button
- Click on save file and specify file to save to

23 Testing Ease of Use

User testing will be used to ensure the program is usable and understandable.

23 Control

Test groups consisting of other students will be given simple predetermined tasks to complete using the program. Their ease in accomplishing these tasks will be recorded by hand. Additionally, the test groups will be allowed to interact with the program freely, and any feedback will be recorded.

23 Inputs

- Realistic simple tasks that would be expected of a typical user of the program
- Combinations of inputs and commands for the test groups to attempt

23 Outputs

- The results of the test groups' attempts to complete the given sample tasks using the program
- The test users' reactions to the program and its user interface
- The time needed for the test groups to complete the given tasks
- Any and all feedback from the test groups regarding the program's commands and user interface

23 Procedures

- Ask the test users to perform simple entry additions
- Ask the test users to perform simple entry modifications
- Ask the test users to perform simple entry searches
- Ask the test users to perform simple entry sorts
- Ask the test users to perform more complex combinations of commands
- Record how quickly and with how much ease each test user performed each task

- Ask each test user what would make the tasks easier
- Record each test user's feedback

24 Testing Boundary Cases and Exception Handling

Boundary Cases will be tested to ensure robustness of the program.

24 Control

The test control for boundary cases will be manual. Strings of information as well as commands will be entered, and the actual results will be compared to expected results.

24 Inputs

- Adding a transaction where the monetary value is 0
- Adding a transaction where there is one monetary value, but the user tries to input several names for the single input
- Adding a transaction where no name for the business is entered
- Adding an empty transaction (where all fields are null or 0)

24 Outputs

- Output message warning the user

24 Procedures

- Input command for adding an entry
- Enter each input as defined above
- Record the output message that the program gives