

Software Testing Address Book

Andrew Henk, Isaac Gainey, Matt Westman

March 24th, 2016

CEN 4072 Software Testing

Instructors: Dr. Ingrid Buckley

Software Engineering Department

Florida Gulf Coast University

Ft. Myers, FL 33965

Table of Contents

1. Software Requirements Specification

I. Introduction

- a. Purpose**
- b. Definitions, Acronyms, and Abbreviations**
- c. Overview**

II. Product Overview

- a. Assumptions**
- b. Use Case Diagram**
- c. Use Case Descriptions**

III. Specific Software Requirements

- a. Functional Requirements**
- b. Non-Functional Requirements**
- c. Performance Requirements**
- d. Design Constraints**

2. Detailed Design & Implementation

I. Detailed Design

- a. Interfaces and Classes**
- b. Class Diagram**
- c. Unit Test Plan**
- d. Integration Tests**
- e. Splitting work**

II. Implementation

3. Test Plan

I. Unit Test Coverage

II. Integration Test Coverage

III. Coverage

IV. Functional Test Coverage

V. Detailed Test Plan

4. Test Report

I. Unit Test Coverage

II. Integration Test Coverage

III. Coverage

IV. Functional Test Coverage

V. Issues

VI. Resources

VII. Conclusion and Summary

5. Team Work

I. Andrew Henk

II. Isaac Gainey

III. Matt Westman

1. Software Requirements Specification

I. Introduction

a. Purpose

Software Project for Software Testing. The project will be an address book. This program will be critically viewed and tested. It will store people's names, their phone number, and their address.

b. Definitions, Acronyms, and Abbreviations

Address Book - record of the names, addresses, and telephone numbers of friends, businesses, etc.

Serialization - is the process of converting an object into a stream of bytes in order to store the object or transmit it to memory, a database, or a file.

c. Overview

This program will be an address book, containing people's names, their phone number, and their address. The address book will serve as a collection of contacts for the user. As well as maintain the ability to edit each contact and able to save and load different address books using a graphical user interface. It will be designed with the intent of it being vigorous tested to the requirements. It will be able to edit each contact and save on a local storage drive to access later.

II. Product Overview

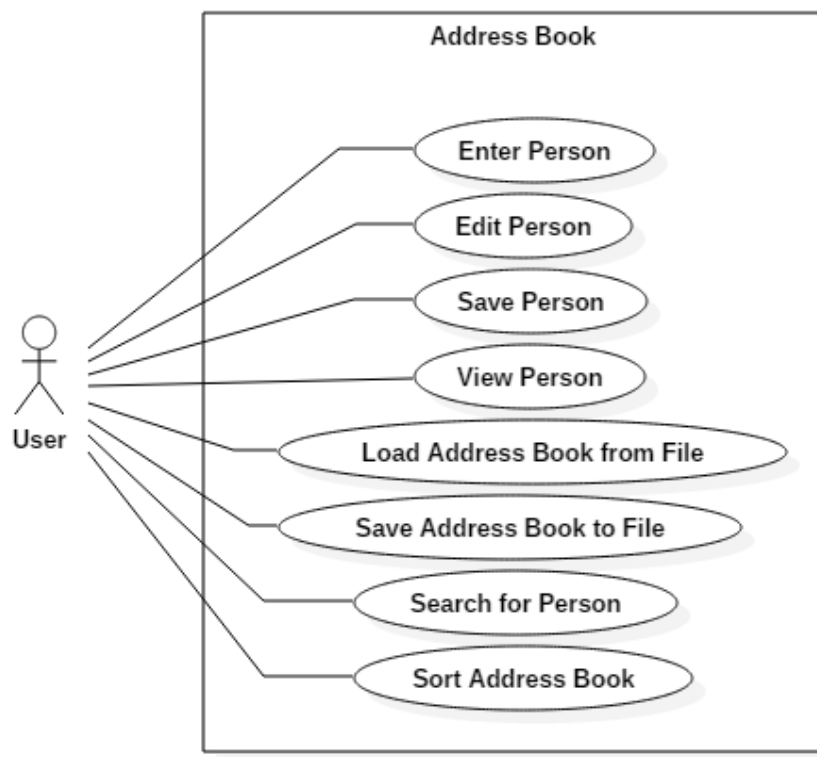
a. Assumptions

- Minimum Hardware:
 - Intel Pentium IV 1.4 GHz
 - 40 GB Hard Drive
 - 2 GB RAM
- Input
 - English Language is used
 - Names are no longer than 50 characters
 - Names do not contain special characters or numbers
 - Phone numbers no longer than 11 digits.

b. Use Case Diagram

Class AddressBookGUI used as a frontend for human users to interact with the program.

Commands are given graphically, which are sent to the appropriate functions.



c. Use Case Descriptions

Use Case	Description Flow
AddressBookController	<ul style="list-style-type: none">• The User enters data into the GUI• The User hits 'add a person' button via GUI• The GUI parses the data and passes the the data to the controller• The controller creates a new person and adds to the collection.• The GUI updates with the new person in the address book• The User will view the GUI and be in aww
FileSystem	<ul style="list-style-type: none">• The User enters the filename of address book data file• The User hits 'load address book'• The Gui launches the file system, parse the file name, and passes the file name• The file system searches for the file• The file system parses the data and passes to the controller• The controller enters the data into the address book and signal the the GUI• The GUI will update with new data with the address book• The User will view the GUI and be in aww
Person	<ul style="list-style-type: none">• The User clicks on a person or types the name of a person• The GUI passes the Name to the controller• The Controller searches for the person and returns the person to the GUI• The GUI access the person and parses the person's data to display to the User• The User will view the GUI and be in aww

III. Specific Software Requirements

No: 1
Statement: The software shall open a stored address book from a file
Test Criteria: verifying that the current address book contains the data imported from the file

No: 2
Statement: The software shall switch between address books from files.
Test Criteria: visual confirmation for loading an existing address book

No: 3
Statement: The software shall be able to create a new address book
Test Criteria: Check that null pointer exception from the GUI will not be created

No: 4
Statement: The software shall edit address books' individual contact
Test Criteria: Select an individual contact and verify changes were made

No: 5
Statement: The software shall save address books to a local file
Test Criteria:

No: 6

Statement: The software shall delete address books

Test Criteria: The local file will be deleted from the storage drive

No: 7

Statement: The software shall hold at least 10 unique identities within an address book

Test Criteria: automated tests for putting 11 names into the system

No: 8

Statement: The software shall delete a contact

Test Criteria: Delete a contact and then searching for that same contact to not be found

No: 9

Statement: The software shall search for a contact

Test Criteria: confirm the table contains contact with search criteria

No: 10

Statement: The software shall ask for a confirmation that the contact will be removed permanently before deleting it.

Test Criteria: Delete a contact and then searching for that same contact

No: 11

Statement: The software shall add entry in an address book for a new contact.

Test Criteria: Searching for the new contact and verify that the contact exist in the database.

No: 12

Statement: The software shall edit current contact information.

Test Criteria: Edit a contact and then searching again for the same contact and verify that the alteration took place.

No: 13

Statement: The software shall sort entries by name and ZIP code

Test Criteria: verify the order of saved entries

No: 14

Statement: The GUI shall display a list of names of persons in the current address book

Test Criteria: visual confirmation of data entered into address book being listed

No: 15

Statement: The GUI shall display the title of the current loaded address book

Test Criteria: visual confirmation of the address book title being displayed

No: 16

Statement: The GUI shall display if the address book has not been saved and needs to be saved

Test Criteria: creating a change to the address book and monitoring for the change in “Save” state

No: 17

Statement: The “Save” button shall save all current contacts of the address book to a file.

Test Criteria: succeeding to save changes when changes have been made

No: 18

Statement: The software shall offer the user to save the current address book after creating or editing a contact

Test Criteria: succeed in attempting to trigger the saving suggestion

a. Functional Requirements

3.1.1 - The program shall display the list of all names collected.

3.1.2 - The names display shall be in alphabetic order.

3.1.3 - After selection of an individual, all available information will be displayed to the user.

b. Non-functional Requirements

3.2.1 - The program shall be functional on all desktop's OS.

c. Performance Requirements

3.3.1 - The machine using this program will have Java.

d. Design Constraints

3.4.1 - The program shall be written in java.

3.4.2 - The collection of names and associated data shall be stored locally.

2. Detailed Design & Overview

I. Detailed Design

a. Interfaces & Classes

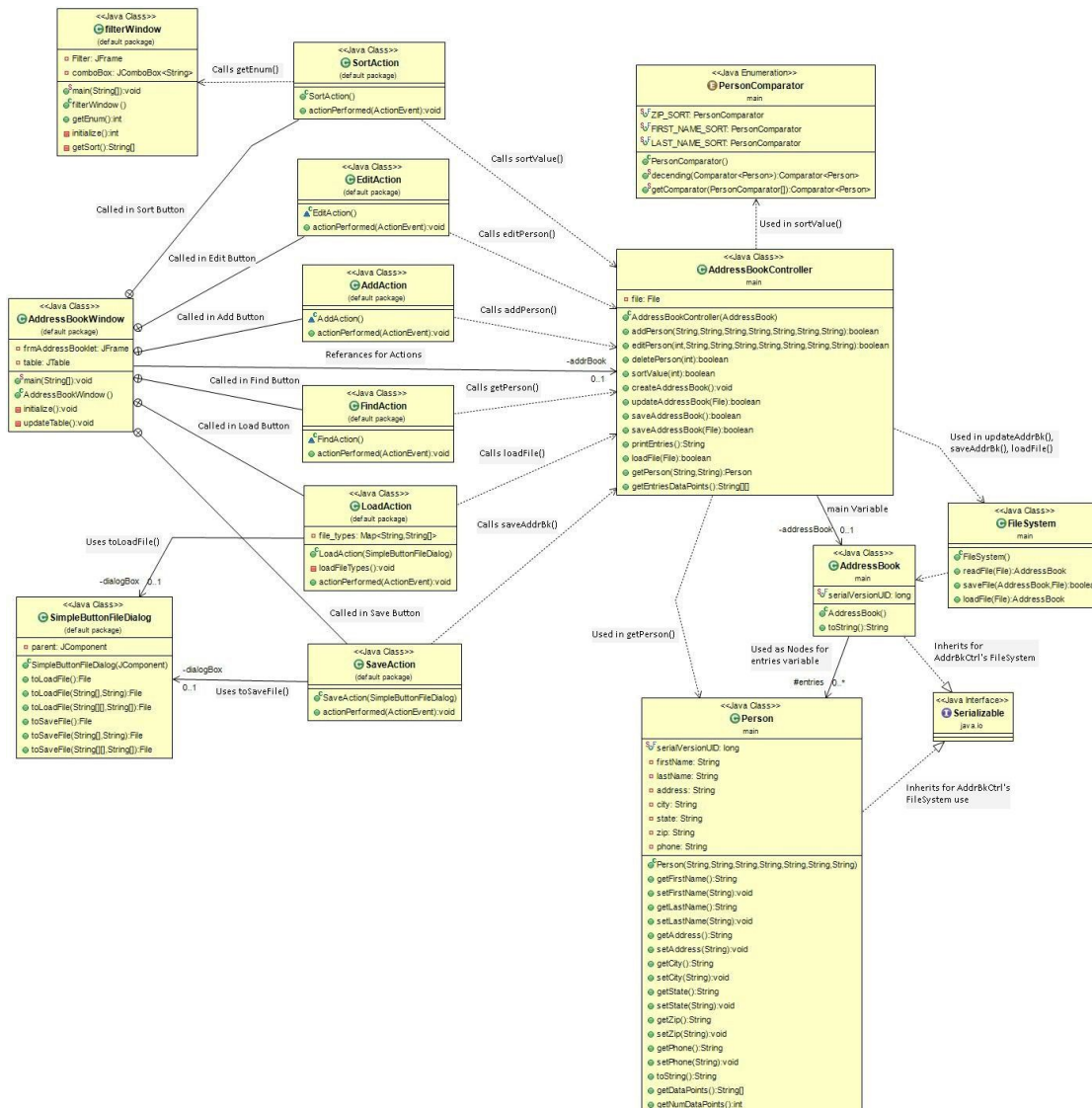
Interfaces

- `FileSystem.java`
 - Saves the `AddressBook` object and by extension `Person` objects to a file that is passed via arguments
 - Loads an `AddressBook` object and its collection from a file.
- `GUI package`
 - Uses java swing to display the `AddressBook`'s collection and allow easy editability.

Classes

- `AddressBook.java`
 - Responsible for keeping the collection of 'Person's as the contact list.
 - Inherits `Serializable` class for the file system class.
- `AddressBookController.java`
 - Attaches `AddressBookGUI` class, `AddressBook` class, and `FileSystem` interface
- `Person.java`
 - Holds all the contact information
 - Name
 - Address (street, state, zip etc)
 - Phone
 - Inherits `Serializable` class for the file system class.
- `PersonComparator.java`
 - Inherits `Comparator` class to enable default sort algorithms that collection classes use.

b. Class Diagram



While the Controller class connects the GUI and AddressBook, there are methods that require the GUI to bypass the controller directly to the AddressBook. Excluding those expectations, the controller bridges the GUI and AddressBook. In which AddressBook has a collection of Person class which are the entries that the user would store their contact information in. The controller also calls the PersonComparator, which compares one attribute of two Persons and enables auto-organization for display purposes.

The FileSystem class is present as functions are directly references AddressBook and are called in the AddressBookController save and load functions. It is also the reason the AddressBook and Person class inherit Serializable, in order to save to a file as their current object state.

c. Unit Test Plan

Each of the classes in the class diagram will be unit tested except for Serializable, Person.java, and PersonComparator.java

- AddressBookGui.java
- AddressBookController.java
- AddressBook.java
- FileSystem.java

For more, see the following test classes for integration test:

- AddressBookGuiTest.java
- AddressBookControllerTest.java
- AddressBookTest.java
- FileSystemTest.java

All the test classes end with “Test”.

We plan to achieve 90% code coverage and 80% branch coverage in unit testing. We will monitor this with Eclemma (<http://eclemma.org/installation.html>)

d. Integration Test Plan

The following classes will be tested with its dependencies in the integration test:

- AddressBookGui.java
- AddressBookController.java
- AddressBook.java

For more, see the following test classes for integration test:

- AddressBookIGuiTest.java
- AddressBookIControllerTest.java
- AddressBookITest.java

All the test classes end with “ITest”.

We plan to achieve 90% code coverage and 80% branch coverage for non-GUI functions with Integration Testing.

f. Splitting the Work

The implementation of the project is split between all members. All members are responsible for unit testing their contributions. Once a class is complete, along with the dependency classes, the integration testing is performed by any of the members.

- Isaac Gainey
 - Implementation
 - AddressBookController.java
 - FileSystem.java
 - GUI package
 - Testing
 - AddressBookControllerTest.java
 - FileSystemTest.java
- Andrew Henk
 - Implementation
 - AddressBook.java
 - Person.java
 - PersonComparator.java
 - Testing
 - AddressBookTest.java
- Matt Westman
 - Testing
 - GUI.java

II. Implementation

The software to be designed is a program that can be used to maintain an address book. An address book holds a collection of entries, each recording a person's first and last names, address, city, state, zip, and phone number. An addressBook is managed by the AddressBookController, which is utilized by the Graphical User Interface.

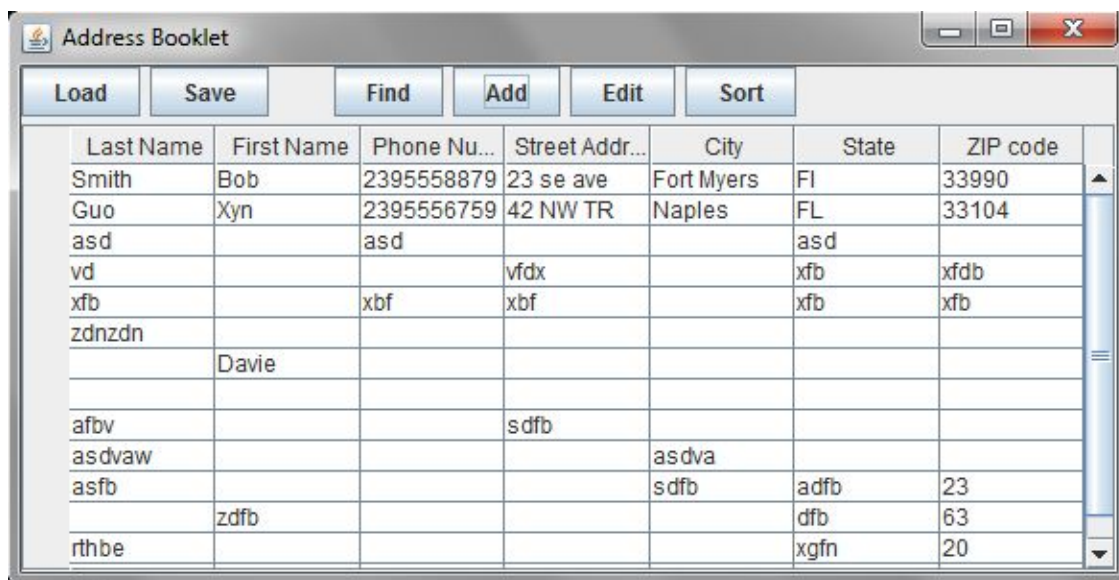
The software was designed using Java, GIT, and Maven on the Eclipse Platform.

AddressBookGUI: The basic responsibility of a GUI object is to allow interaction between the program and the human user.

Responsibilities include:

1. Keep track of the address book object it is displaying
2. Display a list of the names of persons in the current address book
3. Display the title of the current address book - if any
4. Maintain the state of the "Save" menu option - usable only when the address book has been changed since the last time it was opened / saved.
5. Allow the user to request the performance of a use case

Below is a screenshot of the Graphical User interface and the buttons used.



Filesystem: The responsibility of the FileSystem object is to manage interaction between the program and the file system of the computer it is running on.

Responsibilities include:

1. Read a stored serialized address book from a file, given its file name
2. Save an address book as a serialized file, given its file name

The following is the code used for saving and reading a file:

```
/*
 * Saves serialized file
 */
public boolean saveFile(AddressBook addressBook, File file) {
    ObjectOutputStream oos = null;
    try {
        FileOutputStream fout = new FileOutputStream(file);
        oos = new ObjectOutputStream(fout);
        oos.writeObject(addressBook);
        oos.close();
        return true;
    } catch (Exception e) {
        //e.printStackTrace();
        return false;
    }
}

/*
 * Returns address book at file location
 */
public AddressBook readFile(File file) {
    AddressBook addressBook;
    ObjectInputStream ois = null;
    try {
        FileInputStream fin = new FileInputStream(file);
        ois = new ObjectInputStream(fin);
        addressBook = (AddressBook) ois.readObject();
        ois.close();

        return addressBook;
    } catch (Exception e) {
        //e.printStackTrace();
        return null;
    }
}
```

Person: The responsibility of the Person object is to maintain information about a single individual.

Responsibilities include:

1. Creating a new object, given an individual's name, address, city, state, ZIP, and phone
2. Get and Set information
3. Implement serializable

AddressBook: The responsibility of the AddressBook object is s to maintain a collection of Persons.

Responsibilities include:

1. Maintain list of Person objects.
2. Implement serializable

AddressBookController: The responsibility of the AddressBookController object is to manage all manipulation of the AddressBook object.

Responsibilities include:

1. Add, edit, and delete a person
2. Sort entries by name and ZIP code
3. Create, update, save an existing address book
4. Print entries in the address book
5. Allow saving the address book
6. Search the address book by name or phone number

The following is example code for adding a person; edit is a similar function.

```
/*
 * adds a person to the addressBook calls to the addressBook to add returns
 * true if success
 */
public boolean addPerson(String firstName, String lastName, String address, String city,
String state, String zip,
    String phone) {
    Person person = new Person(firstName, lastName, address, city, state, zip,
```

phone);

```
if (!addressBook.entries.contains(person)) {  
    addressBook.entries.add(person);  
    return true;  
} else {  
    return false;  
}
```

}

The sortValue function, uses a switch to choose which enumerator from PersonComparator to use. Currently 4 options are implemented.

```
/*  
* sorts based on the value using a switch to set the comparator  
* 0 = lastNameSortAscending  
* 1 = firstNameSortAscending  
* 2 = zipSort descending  
* 3 = zipSort ascending  
*/  
public boolean sortValue(int option) {  
    Comparator compare = null;  
    if(option == -1){  
        return false;  
    }  
    switch(option){  
        case 0: // last name  
            compare = getComparator(LAST_NAME_SORT);  
            break;  
        case 1: // first name  
            compare = getComparator(FIRST_NAME_SORT);  
            break;  
        case 2: // sortZIP  
            compare = decending(getComparator(ZIP_SORT));  
            break;  
        case 3: // zip  
            compare = getComparator(ZIP_SORT);  
            break;  
    }  
  
    try {  
        Collections.sort(addressBook.entries, compare);  
        return true;  
    } catch (Exception e) {
```

```
        e.printStackTrace();  
        return false;  
    }  
}
```

PersonComparator: The responsibility of the PersonComparator object is to manage all forms of comparison needed for the Person Object.

Responsibilities include:

1. ZIP sort
2. First Name sort
3. Last Name sort

3. Test Plan

I. Unit Test Coverage

Unit Test Coverage will be tracked using ECL Emma for Eclipse. Unit testing will be conducted using JUnit.

- Every functional requirement must be tested separately
- Every non-functional requirement must be tested separately
- Every requirement must be tested against incorrect input/data

II. Integration Test Coverage

Integration testing will be conducted using stubs and mocks. It will cover the complete use of the GUI package, manually.

III. Coverage

Coverage will be tracked and monitored using ECL Emma for Eclipse. Coverage will reach a minimum of 90% code coverage and 80% branch coverage.

IV. Functional Test Coverage

Functional tests will cover each Software Requirements Specification.

V. Detailed Test Plan

The following format will be used to test each requirement.

Test Criteria	
Requirement Number	Provide a description of requirement being tested. Only one requirement must be tested at a time.
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<i>Add the exact code of test method, procedure, stub, mock</i>
Expected Output	State what the expected output of this test should be

Test Result	State the output obtained
Discrepancies	If applicable, describe any discrepancies between the expected and actual results.
Branch Coverage (%)	State the percentage of branch coverage obtained for the requirement
Statement Coverage (%)	State the percentage of code coverage obtained for the requirement

4. Test Report

I. Unit Test Coverage

The Unit testing results are posted below. According to ECL Emma unit testing achieved a 93% code coverage.

Test Criteria 01	
Requirement Number 01	The software shall open a stored address book from a file
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<pre>@Test public void IOExceptionReadTest(){ assertNull(filesys.readFile(null)); }</pre>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	None
Branch Coverage (%)	100%
Statement Coverage (%)	100%

Test Criteria 02	
Requirement Number 02	The software shall switch between address books from files.
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<pre>@Test public void IOExceptionReadTest(){</pre>

	<pre>assertNull(filesys.readFile(null)); }</pre>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	None
Branch Coverage (%)	100%
Statement Coverage (%)	100%

Test Criteria 03	
Requirement Number 03	The software shall save address books to a local file
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<i>Add the exact code of test method, procedure, stub, mock</i>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	None
Branch Coverage (%)	100%
Statement Coverage (%)	100%

Test Criteria 04	
Requirement Number 04	The software shall edit address books' individual contact
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and

	explicitly state which steps you are applying.
Test Method/Procedure	<pre> @Test public void testEditPerson() { int index = 0; String firstName = "Joe"; String lastName = "Bob"; String address = "kowloon ln"; String city = "Punta Rassa"; String state = "FL"; String zip = "33333"; String phone = "239-154-9584"; controller.addPerson(firstName, lastName, address, city, state, zip, phone); // now editing person city = "Miami"; String output = "serialVersionUID : 1 " + "Entries : " + "Person : " + "serialVersionUID : 1 " + "LastName : " + lastName + " FirstName : " + firstName + " Address : " + address + " City : " + city + " State : " + state + " Zip : " + zip + " Phone : " + phone; controller.editPerson(index, firstName, lastName, address, city, state, zip, phone); assertEquals("edit a person", output, controller.printEntries()); } </pre>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	None
Branch Coverage (%)	90.2%

Statement Coverage (%)	93.3%
------------------------	-------

Test Criteria 05	
Requirement Number 05	The software shall save address books to a local file
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<pre>@Test public void IOExceptionSaveFile(){ assertFalse(filesys.saveFile(new AddressBook(), null)); }</pre>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	None
Branch Coverage (%)	100%
Statement Coverage (%)	100%

Test Criteria 06	
Requirement Number 06	The software shall delete address book entries
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<pre>@Test public void testDeletePerson() { int index = 0; String firstName = "joe"; String lastName = "Bob"; String address = "kowloon ln"; String city = "Punta Rassa"; }</pre>

	<pre>String state = "FL"; String zip = "33333"; String phone = "239-154-9584"; controller.addPerson(firstName, lastName, address, city, state, zip, phone); controller.deletePerson(index); String expected = "serialVersionUID : 1 Entries : "; assertEquals("delete a person", expected, controller.printEntries()); }</pre>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	None
Branch Coverage (%)	90.2%
Statement Coverage (%)	90.5%

Test Criteria 07	
Requirement Number 07	The software shall hold at least 5 unique identities within an address book
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<pre>@Test public void testSortNameLast() { controller.addPerson("Joe", "D", "kowloon ln", "Punta Rassa", "FL", "33333", "239-154-9584"); controller.addPerson("Jill", "C", "kowloon ln", "Punta Rassa", "FL", "33333", "239-154-9584"); controller.addPerson("Jan", "B", "kowloon ln", "Punta Rassa", "FL", "33333", "239-154-9584"); controller.addPerson("Zeke", "E", "kowloon ln", "Punta Rassa", "FL", "33333", "239-154-9584"); controller.addPerson("Andrew", "A", "kowloon ln", "Punta Rassa", "FL", "33333", "239-154-9584"); controller.sortValue(0); }</pre>

	<div>String expected = "serialVersionUID : 1 Entries : " + "Person : serialVersionUID : 1 " + "LastName : A " + "FirstName : Andrew " + "Address : kowloon In " + "City : Punta Rassa " + "State : FL " + "Zip : 33333 " + "Phone : 239-154-9584 " + "Person : " + "serialVersionUID : 1 " + "LastName : B " + "FirstName : Jan " + "Address : kowloon In " + "City : Punta Rassa " + "State : FL " + "Zip : 33333 " + "Phone : 239-154-9584 " + "Person : " + "serialVersionUID : 1 " + "LastName : C " + "FirstName : Jill " + "Address : kowloon In " + "City : Punta Rassa " + "State : FL Zip : 33333 " + "Phone : 239-154-9584 " + "Person : " + "serialVersionUID : 1 " + "LastName : D " + "FirstName : Joe " + "Address : kowloon In " + "City : Punta Rassa " + "State : FL " + "Zip : 33333 "</div>
--	--

	<pre> + "Phone : 239-154-9584 " + "Person : " + "serialVersionUID : 1 " + "LastName : E " + "FirstName : Zeke " + "Address : kowloon ln " + "City : Punta Rassa " + "State : FL " + "Zip : 33333 " + "Phone : 239-154-9584"; assertEquals("Sort Test", expected, controller.printEntries()); } </pre>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	None
Branch Coverage (%)	100%
Statement Coverage (%)	100%

Test Criteria 08	
Requirement Number 08	The software shall delete a contact
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<pre> @Test public void testDeletePerson() { int index = 0; String firstName = "joe"; String lastName = "Bob"; String address = "kowloon ln"; String city = "Punta Rassa"; String state = "FL"; } </pre>

	<pre>String zip = "33333"; String phone = "239-154-9584"; controller.addPerson(firstName, lastName, address, city, state, zip, phone); controller.deletePerson(index); String expected = "serialVersionUID : 1 Entries : "; assertEquals("delete a person", expected, controller.printEntries()); }</pre>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	None
Branch Coverage (%)	90.2%
Statement Coverage (%)	90.5%

Test Criteria 09	
Requirement Number 09	The software shall search for a contact
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<pre>@Test public void testGetPerson() { controller.addPerson("Joe", "Bob", "kowloon In", "Punta Rassa", "FL", "33331", "239-154-9584"); controller.addPerson("Jill", "Bob", "kowloon In", "Punta Rassa", "FL", "33332", "239-154-9584"); controller.addPerson("Jan", "Bob", "kowloon In", "Punta Rassa", "FL", "33433", "239-154-9584"); controller.addPerson("Zeke", "Bob", "kowloon In", "Punta Rassa", "FL", "13333", "239-154-9584"); controller.addPerson("Andrew", "Bob", "kowloon In", "Punta Rassa", "FL", "33633", "239-154-9584"); Person p = new Person("Jan", "Bob", "kowloon In", "Punta Rassa", "FL", "33433", "239-154-9584"); assertEquals("getPerson Test : " + p.toString()+ </pre>

	<pre> " :: " + controller.getPerson(p.getLastName(),p.getFirstName()) , p.toString(), controller.getPerson(p.getLastName(),p.getFirstName()) .toString()); } </pre>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	None
Branch Coverage (%)	90.2%
Statement Coverage (%)	100%

Test Criteria 10	
Requirement Number 10	The software shall ask for a confirmation that the contact will be removed permanently before deleting it.
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<i>Add the exact code of test method, procedure, stub, mock</i>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	If applicable, describe any discrepancies between the expected and actual results.
Branch Coverage (%)	State the percentage of branch coverage obtained for the requirement
Statement Coverage (%)	State the percentage of code coverage obtained for the requirement

Test Criteria 11	
Requirement Number 11	The software shall add entry in an address book for a new contact.
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<pre> @Test public void testAddPerson() { String firstName = "Joe"; String lastName = "Bob"; String address = "kowloon ln"; String city = "Punta Rassa"; String state = "FL"; String zip = "33333"; String phone = "239-154-9584"; controller.addPerson(firstName, lastName, address, city, state, zip, phone); String output = "serialVersionUID : 1 " + "Entries : " + "Person : " + "serialVersionUID : 1 " + "LastName : " + lastName + " FirstName : " + firstName + " Address : " + address + " City : " + city + " State : " + state + " Zip : " + zip + " Phone : " + phone; assertEquals("add a person", output, controller.printEntries()); } </pre>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained

Discrepancies	None
Branch Coverage (%)	90.2%
Statement Coverage (%)	92.6%

Test Criteria 12	
Requirement Number 12	The software shall edit current contact information.
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<pre> @Test public void testEditPerson() { int index = 0; String firstName = "Joe"; String lastName = "Bob"; String address = "kowloon ln"; String city = "Punta Rassa"; String state = "FL"; String zip = "33333"; String phone = "239-154-9584"; controller.addPerson(firstName, lastName, address, city, state, zip, phone); // now editing person city = "Miami"; String output = "serialVersionUID : 1 " + "Entries : " + "Person : " + "serialVersionUID : 1 " + "LastName : " + lastName + " FirstName : " + firstName + " Address : " + address + " City : " + city + " State : " + " + state + " Zip : " + zip </pre>

	<pre> + " Phone : " + phone; controller.editPerson(index, firstName, lastName, address, city, state, zip, phone); assertEquals("edit a person", output, controller.printEntries()); } </pre>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	None
Branch Coverage (%)	90.2%
Statement Coverage (%)	93.9%

Test Criteria 13	
Requirement Number 13	The software shall sort entries by name and ZIP code
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<pre> @Test public void testSortZIPAscending() { controller.addPerson("Joe", "Bob", "kowloon In", "Punta Rassa", "FL", "33331", "239-154-9584"); controller.addPerson("Jill", "Bob", "kowloon In", "Punta Rassa", "FL", "33332", "239-154-9584"); controller.addPerson("Jan", "Bob", "kowloon In", "Punta Rassa", "FL", "33433", "239-154-9584"); controller.addPerson("Zeke", "Bob", "kowloon In", "Punta Rassa", "FL", "13333", "239-154-9584"); controller.addPerson("Andrew", "Bob", "kowloon In", "Punta Rassa", "FL", "33633", "239-154-9584"); controller.sortValue(3); String expected = "serialVersionUID : 1 " + "Entries : " + "Person : " </pre>

	<pre> + "LastName : Bob " + "FirstName : Jan " + "Address : kowloon In " + "City : Punta Rassa " + "State : FL " + "Zip : 33433 " + "Phone : 239-154-9584 " + "Person : " + "serialVersionUID : 1 " + "LastName : Bob " + "FirstName : Andrew " + "Address : kowloon In " + "City : Punta Rassa " + "State : FL " + "Zip : 33633 " + "Phone : 239-154-9584"; assertEquals("Sort Test", expected, controller.printEntries()); } </pre>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	None
Branch Coverage (%)	90.2%
Statement Coverage (%)	91.2%

Test Criteria 14	
Requirement Number 14	Provide a description of requirement being

	tested. Only one requirement must be tested at a time.
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<i>Add the exact code of test method, procedure, stub, mock</i>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	If applicable, describe any discrepancies between the expected and actual results.
Branch Coverage (%)	State the percentage of branch coverage obtained for the requirement
Statement Coverage (%)	State the percentage of code coverage obtained for the requirement

Test Criteria 15	
Requirement Number 15	Provide a description of requirement being tested. Only one requirement must be tested at a time.
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<i>Add the exact code of test method, procedure, stub, mock</i>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	If applicable, describe any discrepancies between the expected and actual results.

Branch Coverage (%)	State the percentage of branch coverage obtained for the requirement
Statement Coverage (%)	State the percentage of code coverage obtained for the requirement

Test Criteria 16	
Requirement Number 16	Provide a description of requirement being tested. Only one requirement must be tested at a time.
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<i>Add the exact code of test method, procedure, stub, mock</i>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	If applicable, describe any discrepancies between the expected and actual results.
Branch Coverage (%)	State the percentage of branch coverage obtained for the requirement
Statement Coverage (%)	State the percentage of code coverage obtained for the requirement

Test Criteria 17	
Requirement Number 17	Provide a description of requirement being tested. Only one requirement must be tested at a time.
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.

Test Method/Procedure	<i>Add the exact code of test method, procedure, stub, mock</i>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	If applicable, describe any discrepancies between the expected and actual results.
Branch Coverage (%)	State the percentage of branch coverage obtained for the requirement
Statement Coverage (%)	State the percentage of code coverage obtained for the requirement

Test Criteria 18	
Requirement Number 18	Provide a description of requirement being tested. Only one requirement must be tested at a time.
Input:	Provide the exact input.
Test Method or procedure description:	Describe the test method or procedure and explicitly state which steps you are applying.
Test Method/Procedure	<i>Add the exact code of test method, procedure, stub, mock</i>
Expected Output	State what the expected output of this test should be
Test Result	State the output obtained
Discrepancies	If applicable, describe any discrepancies between the expected and actual results.
Branch Coverage (%)	State the percentage of branch coverage obtained for the requirement
Statement Coverage (%)	State the percentage of code coverage obtained for the requirement



II. Resources

Resources used are as follows:


- Eclipse
- ECL Emma
- JUnit
- EasyMock
- Maven
- GitHub

III. Conclusion and Summary

The code covered a total of 93%. Each test case was successful.

Runs: 33/33  Errors: 0  Failures: 0

- unit.AllTests [Runner: JUnit 4] (0.093 s)
 - unit.AddressBookControllerTest (0.082 s)
 - InvalidLoadFileTest (0.013 s)
 - testGetDataPoints (0.006 s)
 - testSortZIPDescending (0.009 s)
 - testSortNameLast (0.002 s)
 - testSortNameFirst (0.003 s)
 - testSortZIPAscending (0.002 s)
 - testUpdateAddressBook (0.005 s)
 - testPrintEntries (0.002 s)
 - testSaveAddressBook (0.004 s)
 - saveAddressBook (0.005 s)
 - testDeletePerson (0.002 s)
 - invalidUpdateBookTest (0.002 s)
 - invalidSortTest (0.002 s)
 - testAddPerson (0.001 s)
 - testSortError (0.003 s)
 - testAddPersonExists (0.002 s)
 - testEditPersonCatch (0.001 s)
 - testEditPersonEmpty (0.004 s)
 - testEditPerson (0.002 s)
 - testGetPerson (0.002 s)
 - testCreateAddressBook (0.002 s)
 - InvalidSaveAddressBook (0.001 s)
 - testDeletePersonCatch (0.002 s)
 - testDeletePersonNoValues (0.005 s)
 - unit.FileSystemTest (0.008 s)
 - IOExceptionSaveFile (0.001 s)
 - IOExceptionLoadAddressBookTest (0.002 s)
 - readTest (0.002 s)
 - IOExceptionReadTest (0.001 s)
 - loadTest (0.002 s)
 - unit.PersonTest (0.003 s)
 - testNumDataPoints (0.001 s)
 - testToString (0.000 s)
 - testNewPerson (0.000 s)
 - testDataPoints (0.002 s)

 src/main		93.0 %	759	57	816
 src/test		99.8 %	1,635	3	1,638

5. Team Work

I. Andrew Henk

- Documentation
 - Powerpoint
 - Implementation Section
 - Merging Documents
 - Testing Plan Section
 - Test Report Section, but not coverage section
 - Formatting and organizations of final document
- Implementation
 - AddressBook.java
 - Person.java
 - PersonComparator.java
 - FileSystem.java
 - AddressBookController.java
- Testing
 - AddressBookTest.java
 - PersonTest.java
 - AddressBookControllerTest.java

II. Isaac Gainey

- Documentation
 - Contributions layout
 - Made Original and Updated Class Diagram
 - Added link for eclEmma installation
 - Added a few Screen shots for the PowerPoint
 - Added a few requirements both functional and Software
 - Made use case description flows
 - Completed the Introduction section
- Implementation
 - AddressBookController.java (minor)
 - FileSystem.java
 - GUI package
- Testing
 - AddressBookControllerTest.java
 - FileSystemTest.java
 - PersonTest.java (minor)

- Create the Test Results pictures
- Added a few Expected Errors test cases
- Manually and Integrated Tested The GUI

III. Matt Westman

- Documentation
 - Made Use Case Diagram
 - Added requirements both functional and Software
 - Completed Unit Test Coverage tables