SOFTWARE DESIGN SPECIFICATION
PROJECT 1 : ADDRESS BOOK

Edited by

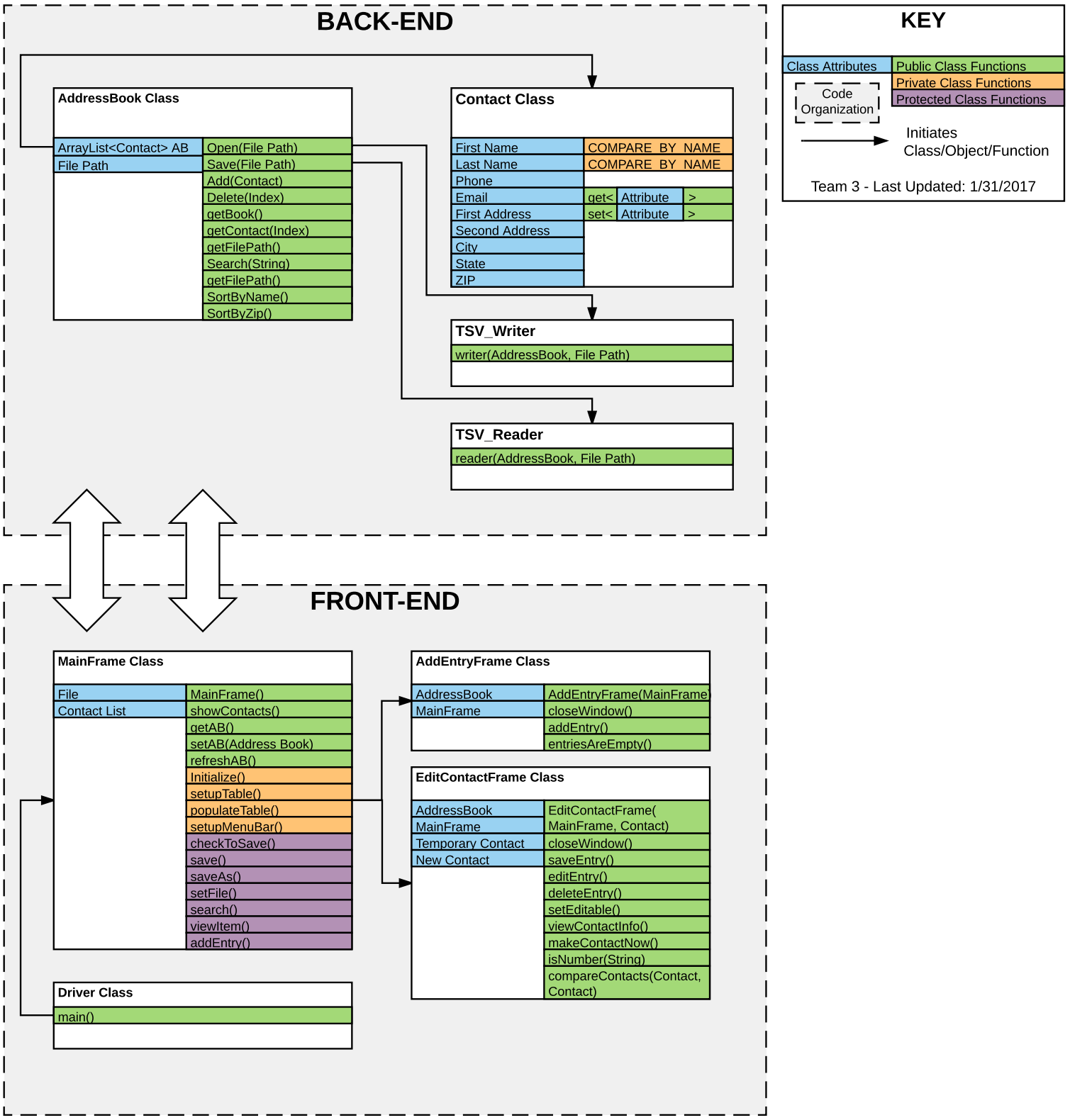
Frazer Bayley
Haley Whitman
Abdulaziz Al-Heidous
Alison Legge
Jeremy Brennan

Project 1 - Team 3

Contents

Revision History

Revision	Date	Author(s)	Description
.1	23.01.17	Haley Whitman	Created initial outline of document
.2	25.01.17	Haley Whitman	Added all code snippets
1.0	31.01.17	Haley Whitman	Added diagram and explanations


BACK-END
AddressBook Class
ArrayList<Contact> AB
File Path
Open(File Path)
Save(File Path)
Add(Contact)
Delete(Index)
getBook()
getContact(Index)
getFilePath()
Search(String)
getFilePath()
SortByName()
SortByZip()
Contact Class
First Name
Last Name
Phone
Email
First Address
Second Address
City
State
ZIP
COMPARE_BY_NAME
COMPARE_BY_NAME
get< Attribute >
set< Attribute >
TSV_Writer
writer(AddressBook, File Path)
TSV_Reader
reader(AddressBook, File Path)
FRONT-END
MainFrame Class
File
Contact List
MainFrame()
showContacts()
getAB()
setAB(Address Book)
refreshAB()
Initialize()
setupTable()
populateTable()
setupMenuBar()
checkToSave()
save()
saveAs()
setFile()
search()
viewItem()
addEntry()
Driver Class
main()
AddEntryFrame Class
AddressBook
MainFrame
AddEntryFrame(MainFrame)
closeWindow()
addEntry()
entriesAreEmpty()
EditContactFrame Class
AddressBook
MainFrame
Temporary Contact
New Contact
EditContactFrame(MainFrame, Contact)
closeWindow()
saveEntry()
editEntry()
deleteEntry()
setEditable()
viewContactInfo()
makeContactNow()
isNumber(String)
compareContacts(Contact, Contact)
KEY
Class Attributes
Public Class Functions
Private Class Functions
Protected Class Functions
Code Organization
Initiates Class/Object/Function
Team 3 - Last Updated: 1/31/2017

1 Introduction

1.1 Intended Audience

The following document covers all functionality of the front and back end code and how they relate with the user's experience. This document is intended for programmers, team managers, and quality assurance on the developing team to ensure that the code is running to this specification.

1.2 How to Use this Document

This document is intended to help organize all modules, classes, and functions found in the Address Book code. It is meant to help all team members design their components to this specification, as well as having a straightforward method of analyzing interactions between different components. Reasonings for why design choices have been made are found at the end of the document.

2 Summary

The entirety of the program is based off of Java JDK 8, which is using a TSV file format to both load and export address book files. This document itself was created from the list of customer specifications which, with customer meetings, guided the creation of the Software Requirements Analysis, which was used to create the requirements needed to begin programming. This document will help create an Quality Assurance documents needed to test the quality of the resulting application.
This document will contain all programming interactions between modules, classes, and functions. This will be achieved by both diagrams and showing all function headers and descriptions of what they achieve and their interactions.

3 User Interface Architecture

3.1 GUI Handbook

3.2 Expected Input

The user is prompted to enter values for the following variables via text fields:

1. First name
2. Last name
3. Phone number
4. Email
5. Address: Street and 2nd Address
6. City
7. State
8. ZIP code

Other use inputs are included as buttons and from file menus, which are accessed by mouse clicks.

1. Close (Both program, and for adding a contact)
2. Edit (Accessed by mouse clicking on a contact)
3. Delete (Accessed within the edit contact menu)

4. Search (Accessed by mouse clicking the search button next to the search bar)
5. New (Accessed in file menu, creates and loads a new address book)
6. Open (Accessed in file menu, opens an existing address book from a file directory)
7. Save (Accessed in file menu,)
8. Save As... (Accessed in file menu)
9. Quit (Accessed in file menu, quits application completely.)

3.2.1 Output

The user receives a list of contacts that they are able to view through, by scrolling a scroll bar on the side of the menu. They are able to access any visible contact by mouse clicks, and able to sort through this displayed list by sorting or searching. A user is allowed to edit or add a contact, which once changed can be elected to be saved and added to the current address book for future use. Upon loading any address book the current address book will be updated visually with the new address book.

3.3 GUI Window

```
public class AddressEntryFrame extends JFrame {}
```

```
public AddressEntryFrame() {}
```

3.3.1 Buttons

```
Void deleteButton() {}
```

```
void editButton() {}
```

```
public class NewEntry {}
```

```
public void addEntry() {}
```

```
public void closeWindow() {}
```

3.4 Back-End Architecture

3.5 Contact Class

The contact class is where all data regarding each entry in the address is kept. This class utilizes strings to hold all information about each entry.

3.5.1 Attributes of Contact Class

```
public Contact(String fn, String ln, String phone, String email,
        String add, String city, String state, String zip) { }
```

This class holds the String data for first name, last name, phone number, email, address, 2nd address, city, state, and ZIP code.

3.5.2 Functions of Contact Class

Functions: All Getters and Setters for Contact Class
Precondition:
Caller has a Contact Object if using a getter, or a String and a Contact object for input as a

setter. Postcondition: All getters will return a String, all setters will return void, but set the inputted String for the corresponding contact.

```
public String getFirstName() {
        return _firstName;
}
public void setFirstName(String value) {
        _firstName = value;
}
```

```
public String getLastName() {
        return _lastName;
}
public void setLastName(String value) {
        _lastName = value;
}
```

```
public String getPhone() {
    return _phone;
}
public void setPhone(String value) {
    _phone = value;
}
```

```
public String getEmail() {
    return _email;
}
public void setEmail(String value) {
    _email = value;
}
```

```
public String getAddress() {
    return _address;
}
public void setAddress(String value) {
    _address = value;
}
```

```
public String getCity() {
    return _city;
}
public void setCity(String value) {
    _city = value;
}
```

```
public String getState() {
    return _state;
}
public void setState(String value) {
    _state = value;
}
```

```
public String getZip() {
    return _zip;
}
public void setZip(String value) {
    _zip = value;
}
```

3.5.3 AddressBook Class

This class contains an arraylist of Contact objects and a few series of functions that allow different manipulations of these list of Contact objects. On constructing an Address Book, an ArrayList of Contacts is created and the following functions can be applied to this Address Book object:

```
public void Save(String filepath) { }
```

Functions: Save
Precondition: String of intended file path for AddressBook object to be exported to.
Postcondition: A boolean value, True if save was successful.

```
public void Add(Contact data) { }
```

Functions: Add
Precondition: A Contact Object
Postcondition: Inputted Contact is saved into the applied Address Book.

```
public void Delete(Contact data) { }
```

Functions: Delete
Precondition: An Integer Index of the desired Contact to delete
Postcondition: The Contact at the specific instance will be deleted from the Address Book.

```
public Contact SearchBy(Integer key) { }
```

Functions: Search
Precondition: An Address Book and an inputted String to search for.
Postcondition: An ArrayList of Contacts is returned, which will contain all Contacts with any set of the inputted String found in any of the Contacts fields. This will be displayed on the Main GUI window.

3.5.4 Sorting

```
public void SortByName() { }
```

```
public void SortByZip() { }
```

```
public static Comparator<Contact> COMPARE_BY_NAME =
                                  new Comparator<Contact>() {
    public int compare(Contact one, Contact other) { }
}
```

```
public static Comparator<Contact> COMPARE_BY_ZIP =
                                  new Comparator<Contact>() {
    public int compare(Contact one, Contact other) { }
}
```

4 Module/Class Explanations

4.1 Driver

This main class was created to initiate the main GUI window, and is intended to serve in the future as the "Main" of the program.

4.2 MainFrame

This class is considered the main link between the back-end and the front-end of this application. It is used to initiate and display any Address Book that the user loads, and allows for the user to open up ancillary edit and add windows for managing contacts.

4.3 AddEntryFrame

This class is used for creating contacts. AddEntryFrame shares many similarities to EditContactFrame, but focuses only on making the constructor for the Contact class. The reason why both the Add and Edit GUI frames are separate is to make the code more manageable in the future, such that in later developments there can be more changes to both activities. Having separate frames also will allow multiple address books to be opened at once, as global variables should be avoided for this reason.

4.4 EditContactFrame

This class is used for editing contacts. EditContactFrame shares many similarities to AddEntryFrame, but uses the back-end Contact Setters and getters to edit the current Contact's information. One negative that comes from this type of design is that some changes to the AddEntryFrame would have to be replicated here, such as prompting to save on exiting with unsaved information. With multiple windows, this code replication is not ideal. However, the benefit of not dealing with global variables outweighs this.

4.5 AddressBook

This class is used to store all the Contacts created for the application. Since the Address Book needs to deal with both loading and exporting TSV files, a simple ArrayList was chosen to store all of this information. This class depends on TSV_Writer and TSV_Reader to do file handling. Since the requirements of the AddressBook are only needed to store approximately hundreds of contacts, no optimizations were needed to be done to manage the number of Contacts within searching and sorting. This lets the complexity of the code be much lower, and is easier to manage.

4.6 Contact

This class is used to store all the needed fields of information for Contacts, and includes a set of Getters and Setters for each. If any new fields are needed for Contacts, they can simply be added to this Class as well as the corresponding Getters and Setters.

4.7 TSV_Reader

For this project, it is not required to only need to import TSV files, but for this implementation, both our imported and exported files are the same. This implementation is helpful because it

allows us to keep the complexity of the code low, as well as keep the same standard file structure present throughout the application.

4.8 TSV_Writer

This Class takes care of the Export feature of the Address Book, which is standardized as per the requirements of the customer. This will read through a provided AddressBook Class and output a standardized format TSV file.

5 Appendices

5.1 Definitions and Acronyms

5.1.1 Definitions

5.1.2 Acronyms and Abbreviations

GUI	Graphical User Interface
SDS	Software Design Specification
SRS	Software Requirement Specification
TSV	Tab-Separated-Values