**Address Book GUI**

**Jason Ricardo, Kyle Zinsser, Kevin Zamora and William Rivera**

**Draft Number 2**

**Thursday, March 3, 2016**

**CEN 4072 Software Testing Project**

**Instructors: Dr. Ingrid Buckley**

**Software Engineering Department**

**Florida Gulf Coast University**

**Ft. Myers, FL 33965**

**Modification History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Who | Comment |
| v1.0 | February 2, 2016 | Jason, Kyle, Kevin, William | Initial SRS |
| V1.1 | March 3, 2016 | Jason, Kyle, Kevin, William | Detailed Design and Implementation |

Team Name: DevOps Squad

Team Members: Jason Ricardo, Kyle Zinsser, Kevin Zamora, William Rivera

**Contents of this Document**

Introduction

· Software to be Produced

· Definition, Acronyms, and Abbreviations

Product Overview

· Assumptions

· Use Case Diagram

· Use Case Descriptions

Specific Requirements

Detailed Design and Implementation

References

**1. Introduction**

**1.1 Software to be Produced**

The system should have various classes that will interact with each other to give functionality. The goal is to have an Address Book where a user can Add, Edit, Update and Delete entries within this address book. The user will interact directly with a GUI which will display information related to the address book. The GUI will interact with a class that we will call a Controller. The controller is used so there can be multiple Address Books instantiated from within the GUI. The GUI can take this instance of AddressBook and call on the controller which will manipulate the items of the Address Book.

The AddressBook will be a simple class that will have functionality pertaining to the AddressBook but nothing else. This is done in this matter so the system can be more modular and remains loosely coupled. Once the system is ready to terminate its running state it will create a file where all the data is written. This will be done in the form of a text file carrying the name of the address book being saved. The following time the program starts it will look for the files available and give the user the ability to open the address book her or she desire to work with. Once the system is done it must update the information within the text file.

**1.2 Definitions**

**Object Orientation:** The programming of classes based on the defined attributes that only belong to one of these objects at any given instance.

**GUI:** Graphical User Interface.

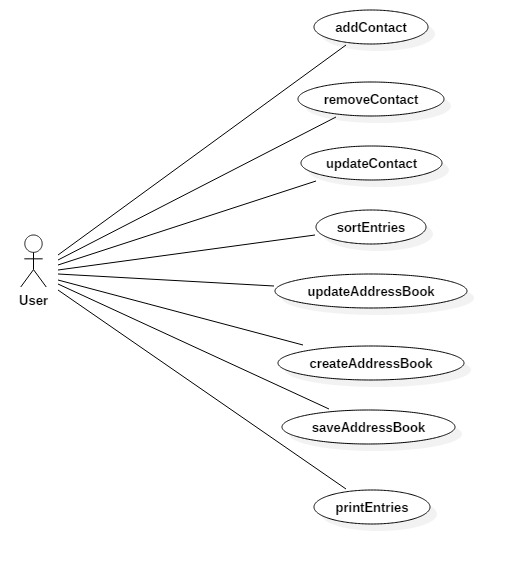
**Program:** Computer software to perform a task in a systematic way. Used interchangeably with the word Software.

**2. Product Overview**

**2.1 Assumptions**

* **Minimum Hardware:**
  + Intel i3 2.0 GHz CPU
  + 10 GB Hard Drive
  + 2 GB RAM
* **Input data**
  + firstName, lastName, city, and state fields must contain only alphabetic characters
  + zip and phone fields must contain only numeric characters

**2.2 Use Case Diagram**



**2.3 Use Case Descriptions**

|  |  |
| --- | --- |
| **Use Case** | **Description** |
| Add Contact | The user will be able to add a new contact to the address book. |
| Remove Contact | The user will be able to remove a contact from the address book. |
| Update Contact | The user will be able to update information of a contact. |
| Sort Entries | The user will be able to sort the contacts. |
| Update Address Book | The user will be able to update the address book. |
| Create Address Book | The user will be able to create a new address book. |
| Save Address Book | The user will be able to save to the address book. |
| Print Entries | The user will be able to print any of the contacts. |

**3. Specific Requirements**

**3.1 Requirements**

|  |
| --- |
| **No: 1** |
| **Statement:** The program shall add a user to the address book. |
| **Test Criteria:** Test shall add an entry to the address book and verify that the entry the user information was properly added. |

|  |
| --- |
| **No: 2** |
| **Statement:** The program shall sort the entries by name of the address book. |
| **Test Criteria:** Test shall sort the entries alphabetically and verify that the address book starts with A, followed by B, followed by C, all the way up to Z. |

|  |
| --- |
| **No: 3** |
| **Statement:** The program shall create an address book. |
| **Test Criteria:** Test shall add an address book and verify that the address book was properly added. |

|  |
| --- |
| **No: 4** |
| **Statement:** The program shall print the contents of an address book. |
| **Test Criteria:** The program shall aggregate the contents of the address book and output the contents to a print job. |

|  |
| --- |
| **No: 5** |
| **Statement:** The program shall offer the user to save changes to an address book. |
| **Test Criteria:** The program shall make the save function usable after the user has modified the contents of the address book. |

|  |
| --- |
| **No: 6** |
| **Statement:** The program shall display the title of the current address book. |
| **Test Criteria:** The program shall get the title of the address book and verify that matches the same name in the title bar of the user interface. |

|  |
| --- |
| **No: 7** |
| **Statement:** The program shall make the Save menu option usable after any changes to an address book. |
| **Test Criteria:** The program shall disable the save menu option when no changes were made to the address book after it was previously saved. |

|  |
| --- |
| **No: 8** |
| **Statement:** The program shall display the list of names of persons in the current address book selected. |
| **Test Criteria:** The program shall display the names of the address book and will match the contents back to the address book to compare authenticity. |

|  |
| --- |
| **No: 9** |
| **Statement:** The program shall keep track of the address book objects it displays. |
| **Test Criteria:** The test should instantiate multiple address books and |

|  |
| --- |
| **No: 10** |
| **Statement:** The Software shall save data to a text file using the title of the AddressBook as the file name. |
| **Test Criteria:** Test will create an addressbook by passing it a name and then will read a file using the name originally passed in. |

|  |
| --- |
| **No: 11** |
| **Statement:** The title of the Address Book should be inputted by the user at creation time. |
| **Test Criteria:** Test case should create an addressbook and it shall prompt for a title. |

|  |
| --- |
| **No: 12** |
| **Statement:** The Software should read from a File based on the title of the AddressBooks. |
| **Test Criteria:** The title of a file should be verified against the test imputed value |

|  |
| --- |
| **No: 13** |
| **Statement:** The software should handle errors in opening and saving of the AddressBook File by throwing an exception. |
| **Test Criteria:** Test case should expect an exception class defined by the system called ReadWriteException. |

|  |
| --- |
| **No: 14** |
| **Statement:** The Software shall create a person object that can be added to the AddressBook. |
| **Test Criteria:** The program shall be able to read the contents of the person object after it was added to the address book. |

|  |
| --- |
| **No: 15** |
| **Statement:** The Person object shall contain at minimum firstname, and lastname. |
| **Test Criteria:** The program shall reject any entries that do not contain a first name or a last name. |

|  |
| --- |
| **No: 16** |
| **Statement:** The Person object should additionally be able to store Street, City, State, Zip, and phone variables |
| **Test Criteria:** The program shall be able to read the contents of the address variables after then have been stored. |

|  |
| --- |
| **No: 17** |
| **Statement:** The Person object variables should be instantiated through constructors and secondarily through setter methods. |
| **Test Criteria:** The program shall verify that when the object is instantiated in the test method, the person object was also created. |

|  |
| --- |
| **No: 18** |
| **Statement:** The Person variables should be private to the class. |
| **Test Criteria:** The program shall not be able to call any variables from outside of the Person Class. |

|  |
| --- |
| **No: 19** |
| **Statement:** The Person information can be updated with the exception of the name of the person. |
| **Test Criteria:** The program shall verify that the person’s name in the address book remains constant and the rest of the entries can change. |

|  |
| --- |
| **No: 20** |
| **Statement:** The Calling entity shall be able to get any information of the person by using the getters within the entity. |
| **Test Criteria:** The program shall match the calling entry with the specific person information that needs to be searched. |

**4. Detailed Design and Implementation**

**4.1 Interfaces and Classes**

Interfaces

* Serializable

Classes

* Controller
  + AddressBookController – entries (ArrayList<Person>)
  + AddressBookController – sortEntries (Comparator<Person>)
* Graphical User Interface
  + AddressBookGUI – frame (JFrame)
  + AddressBookGUI – listPanel (JPanel)
  + AddressBookGUI – contentPanel (JPanel)
* File System
  + FileSystem – file (InputStream) (OutputStream)
  + FileSystem – buffer (InputStream) (OutputStream)
  + FileSystem – input (ObjectInput) (ObjectOutput)

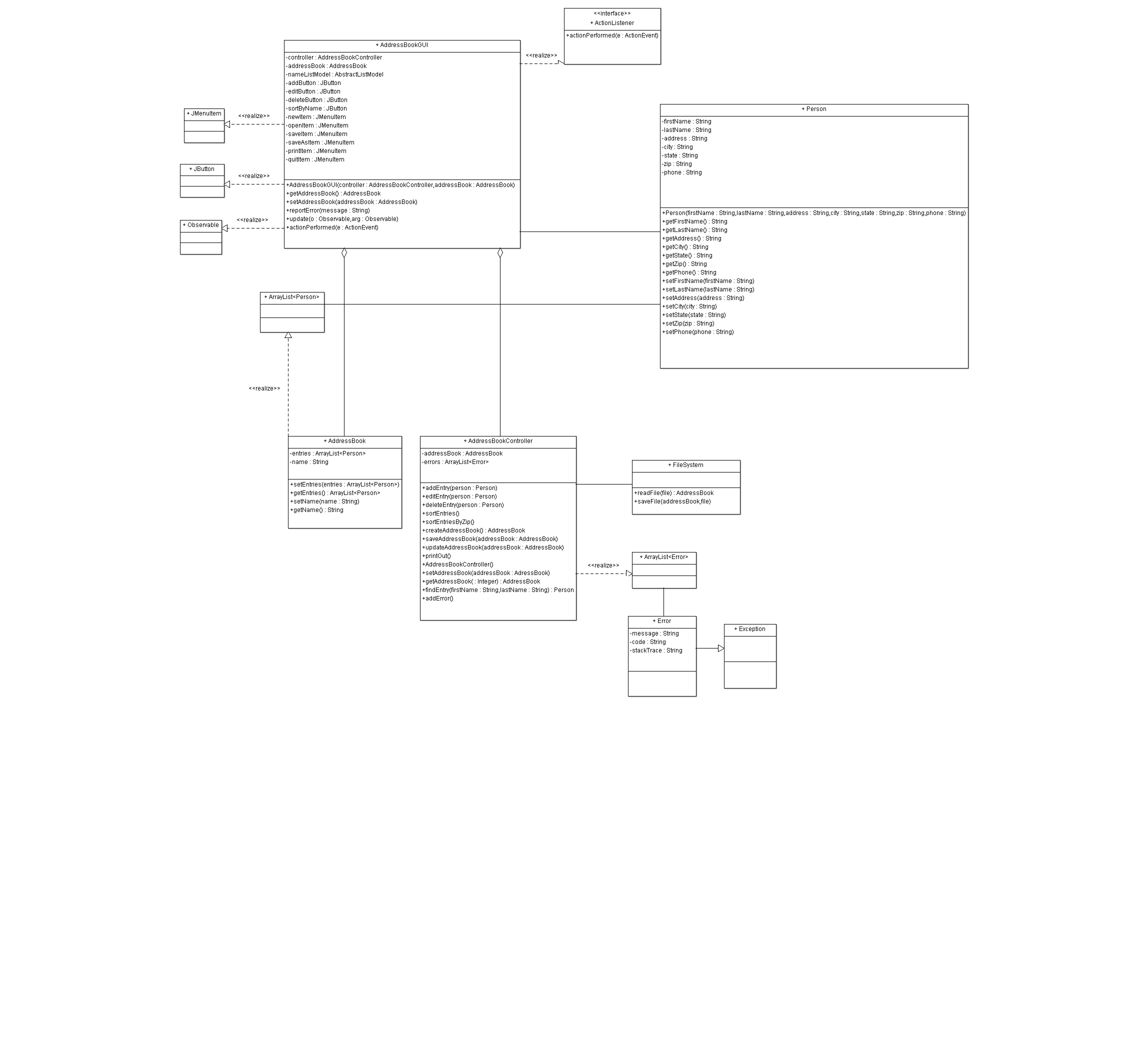
**4.2 Class Diagram**

The following class diagram depicts the design of classes for the Address Book. The Program first boots a GUI in which the user can have an easier interaction with the functionality. This class is called in the diagram AddresssBookGUI. The class will implement JButtons, Menu Items and will interact directly with the AddressBookController class. The interaction between the AddressBookController and the AddressBookGUI will be established from the calling class which we call Application. Application will instantiate GUI, AddressBook and Controller. Application will pass as instantiation arguments AddressBook and Controller to the GUI. After Application class boots the program structure then it will start the user interaction with the GUI.

The GUI class will make calls onto the controller for saving and loading an address book. The buttons will implement a Listener to catch when the button was clicked. The listener will be a method within the GUI therefore AddressBookGUI will implement ActionListener. The GUI will implement a way to look through the entries and find the one the user is trying to find. Also there will be search functionality by using first name and last name. In order for the GUI to search the GUI will call on the Controller method findEntry().

The controller class will handle the core functionality of the program. Therefore if an address Book needs to be sorted by names or by zip this class will be called upon. The controller will interact directly with the FileSystem to save and load an AddressBook. The system will save the addressBook by the name of this addressBook. Additionally the FileSystem class will be able to search a main directory to know which addressBooks are available as saved addressBooks based on the name under which it was saved.

A final note regarding the class diagram is the realization of other libraries contained within Java. For example we will implement JButtons and therefore we annotate it as a realization of the blueprint provided by Java Libraries.



**4.3 Dependencies**

JFrame dependencies

public class AddressBookGUI extends JFrame {

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

AddressBookGUI frame = new AddressBookGUI();

frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

}

**4.4 Unit Test Plan**

The testing plan for the program produced is to test by individual method. JUnit 4.0 will be used to test each individual method. Testing the functionality of the software will be done first which includes the FileSystem, AddresBookController, and Person class, followed by the method using a test class exclusive to the class being tested.

After testing the functionality of the back end of the software, testing of the GUI methods will follow. These methods include getting and setting the addressBook variable within the GUI class and testing for error reporting.

**4.5 Integration Test**

Integration testing will be performed using Bottom-up testing. This testing method fits the development cycle being used because the software modules of the bottom are being built first. Therefore as individual modules are completed by the developer, the tester of the module can move into testing the module. At completion time of the program the software will be tested at the highest level module.

**4.6 Team Assignments**

|  |  |  |
| --- | --- | --- |
| Name | Assignment | Completion |
| Jason Ricardo | Functionality of FileSystem, AddressBookController and Person. Testing of FileSystem, AddressBookController, and Person. |  |
| William Rivera | Functionality of FileSystem, AddressBookController and Person. Testing of FileSystem, AddressBookController, and Person. |  |
| Kevin Zamora | Functionality of AddressBookGUI and testing of GUI system. |  |
| Kyle Zinsser | Functionality of AddressBookGUI and testing of GUI system. |  |

**4. References**

Copeland, Lee. *A Practitioner's Guide to Software Test Design*. Norwood: Artech House Publishers, 2003. Paper.