**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.

**Serialization:** A way to convert a programming object into a stream of bits that preserve the object state at the time of saving to either a file, database or memory.

**ActionListner:** A method in charge of listening to actions performed by a user through a gui. This method will then add functionality to the gui.

**Exception:** An event that happens that changes the expected behavior of a running program. These can be unforeseen events that can disrupt the program.

**2. Product Overview**

**2.1 Assumptions**

**● Minimum Hardware:**

**○** Intel i3 2.0 GHz CPU

○ 10 GB Hard Drive

○ 2 GB RAM

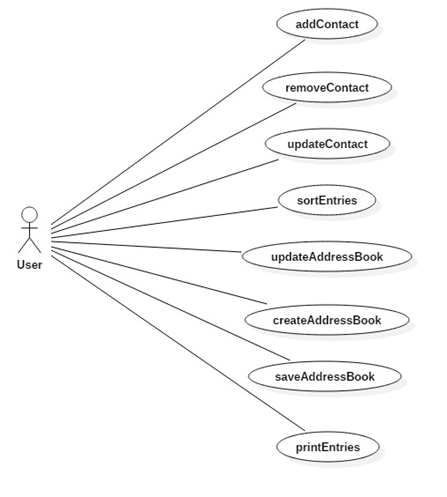
**● Input data**

○ firstName, lastName, city, and state fields must contain only alphabetic characters

○ Name of Address Book as the title of the AddressBook

○ Zip and phone fields must contain only numeric characters and must not exceed the permitted size for each field respectively.

**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 first and last name. |
| **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 prompt a user on exit to save. |
| **Test Criteria:** The program shall make the save function usable after the user has modified the contents of the address book and tries to close out. |

|  |
| --- |
| **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 in a table. |
| **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. |

|  |
| --- |
| **No: 10** |
| **Statement:** The Software shall save data to a serialized 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 File System must be load an AddressBook from file. |
| **Test Criteria:** Test case should save an address book and then load it again to ensure data integrity is kept. |

|  |
| --- |
| **No: 12** |
| **Statement:** The Software should load a GUI screen that can create an AddressBook. |
| **Test Criteria:** The program will be booted and a GUI should come up and an option to create an AddressBook should be available. |

|  |
| --- |
| **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 at minimum. |

|  |
| --- |
| **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 changes 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. |

|  |
| --- |
| **No: 21** |
| **Statement:** The GUI must display all the entries of an Address Book. |
| **Test Criteria:** The program must be opened and we must be able to see the entries in an organized way. |

|  |
| --- |
| **No: 22** |
| **Statement:** The GUI must prompt the user before closing out to ensure the user really wants to close. |
| **Test Criteria:** The GUI must be closed randomly and test if the user is prompted to confirm. |

|  |
| --- |
| **No: 23** |
| **Statement:** If the system opens a corrupted file or missing file it must throw a FileNotFoundException. |
| **Test Criteria:** The program shall test the FileNotFoundException is thrown when the user opens a file that does not exist. |

**4. Detailed Design and Implementation**

**4.1 Interfaces and Classes**

The user interface is made using Java Swing components. The action listeners are programmed as methods underneath each GUI component method to catch input from the user. These methods have access to the AddressBook and AddressBookController instances. This is crucial in communication purposes between different classes. The GUI class should be the one to catch any exception thrown at RunTime or during program exection.

There is a menu where the user can save, open, and create address books. Once the address book is opened then we assume that the objects of these AddressBook are put into the allocated memory space. The next step in this process is to provide the User a way to view this information.

The contacts get put into a JTable where the user can add, edit, or delete contacts based on the buttons on the bottom. This buttons will not work unless an entry in the JTable is selected. The FileSystem class is what saves and edits the file system. The AddressBookGUI is where all of the GUI components and logic are handled. The controller holds a lot of the basic logic of the program.

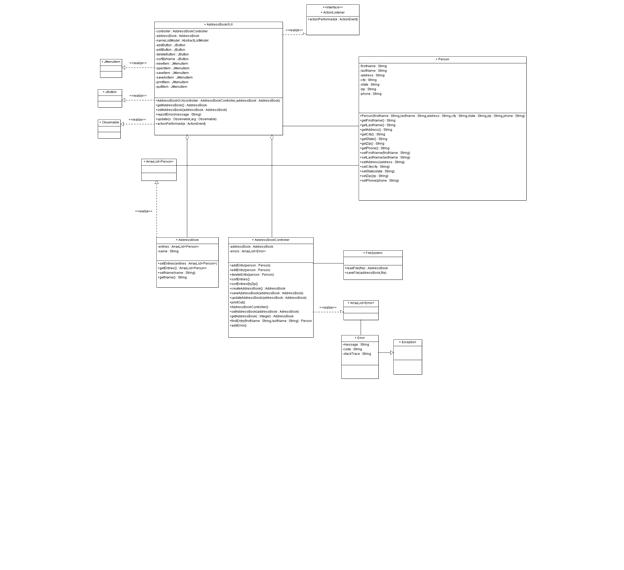
**4.2 Class Diagram**

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. If we notice this 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 we will start the 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 we are look 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 Unit Test Plan**

The testing plan for the program produced is to test by individual method. We will be using JUnit 4.0 to test each individual method. We will test the functionality of the software first which include the FileSystem, AddresBookController, and Person class. We will test each method using a test class exlusive to the class being tested. An important part of JUnit testing is to ensure our tests cover all the methods and branches within these classes. Deciding to test the backend is crucial because our GUI will not work properly unless we have the backend working. As the test methods execute we want to have the following information in our test methods:

* Test boundary Points
* Test if-else statements in methods
* Test exceptions
* Test proper behavior
* Test timeouts

After testing the functionality of the backend of the software we will test the GUI methods. These methods include getting and setting the addressBook variable within the GUI class and testing for error reporting (Copeland 9). In the process of GUI testing its essential to know that not every individual method can be tested.

**4.4 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 (Copeland 140).

**4.5 Team Assignments**

|  |  |  |
| --- | --- | --- |
| **Name** | **Assignment** | **Completion** |
| Jason Ricardo | Functionality of FileSystem, AddressBookController and Person. Testing of FileSystem, AddressBookController, and Person. | 2/3 Complete |
| William Rivera | Functionality of FileSystem, AddressBookController and Person. Testing of FileSystem, AddressBookController, and Person. | 2/3 Complete |
| Kevin Zamora | Functionality of AddressBookGUI and testing of GUI system. | 2/3 Complete |
| Kyle Zinsser | Functionality of AddressBookGUI and testing of GUI system. | 2/3 Complete |

**5. Test Report**

**5.1. Test Plan**

|  |  |
| --- | --- |
| **Test Criteria#1** | |
| **Requirement#1:** | **The user shall be able to add the first and last name into a contact** |
| **Input:** | **testPerson()** |
| **Test method or procedure description:** | **Creating a person object with information of the user. Assert that the setFirstName and setLastName methods are equal to information of user.** |
| **Test Method/Procedure (Code)** | **Person bobPerson = new Person("Bob", "Saget", "1000 Maple Street",**  **"San Francisco", "CA", "55555", "1234567890");**  **assertEquals(bobPerson.getFirstName(), "Bob");**  **assertEquals(bobPerson.getLastName(), "Saget");** |
| **Expected Output** | **AssertEquals the information of the user with the method inputs thru the object** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **100** |
| **Statement Coverage (%)** | **100** |

|  |  |
| --- | --- |
| **Test Criteria#2** | |
| **Requirement#1:** | **The user shall be able to add the address and phone into a contact** |
| **Input:** | **testPersonDetails()** |
| **Test method or procedure description:** | **Creating a person object with address and phone of the user. Assert that the setAddress, setCity, setState, setZip and setPhone are equal to the information of the user** |
| **Test Method/Procedure (Code)** | **Person bobPerson = new Person("Bob", "Saget", "1000 Maple Street",**  **"San Francisco", "CA", "55555", "1234567890");**  **assertEquals(bobPerson.getAddress(), "1000 Maple Street");**  **assertEquals(bobPerson.getCity(), "San Francisco");**  **assertEquals(bobPerson.getState(), "CA");**  **assertEquals(bobPerson.getZip(), "55555");**  **assertEquals(bobPerson.getPhone(), "1234567890");** |
| **Expected Output** | **AssertEquals the information of the user with the method inputs thru the object** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **100** |
| **Statement Coverage (%)** | **100** |

|  |  |
| --- | --- |
| **Test Criteria#3** | |
| **Requirement#1:** | **The user shall be able to add the title/name of the address book** |
| **Input:** | **testTitle()** |
| **Test method or procedure description:** | **Creating an address book controller, set the title to “pizza” and confirmed that the method getTitle() equals to the word “pizza”** |
| **Test Method/Procedure (Code)** | **AddressBookController controller = new AddressBookController();**  **controller.setTitle("pizza");**  **assertEquals(controller.getTitle(), "pizza");** |
| **Expected Output** | **AssertEquals the title of the address book with the getTitle()** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **100** |
| **Statement Coverage (%)** | **100** |

|  |  |
| --- | --- |
| **Test Criteria#4** | |
| **Requirement#1:** | **The user shall be able to add his contact information and search for it** |
| **Input:** | **testPersonAddressbook()** |
| **Test method or procedure description:** | **Created an address book controller and a person object. Added the user information entry. Searched for entry matching the name of the person object** |
| **Test Method/Procedure (Code)** | **AddressBookController controller = new AddressBookController();**  **Person bobPerson = new Person("Bob", "Saget", "1000 Maple Street",**  **"San Francisco", "CA", "55555", "1234567890");**  **controller.addEntry(bobPerson);**  **assertEquals(controller.findEntry("Bob", "Saget"), bobPerson);** |
| **Expected Output** | **AssertEquals the person add entry with the actual name** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **100** |
| **Statement Coverage (%)** | **100** |

|  |  |
| --- | --- |
| **Test Criteria#5** | |
| **Requirement#1:** | **The user shall be able to create an address book** |
| **Input:** | **testCreateAddressbook** |
| **Test method or procedure description:** | **Created an address book controller and a file system. Created the addressbook with the file system. Tested to see if the new address book does not return null** |
| **Test Method/Procedure (Code)** | **AddressBookController s = new AddressBookController();**  **FileSystem fs = new FileSystem();**  **fs.createAddressBook("addressbook");**  **String addressName = "addressbook";**  **assertNotNull(s);** |
| **Expected Output** | **AssertNotNull the creation of the address book** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **100** |
| **Statement Coverage (%)** | **100** |

|  |  |
| --- | --- |
| **Test Criteria#6** | |
| **Requirement#1:** | **The user shall be able to save an address book** |
| **Input:** | **testSaveFile** |
| **Test method or procedure description:** | **Created an address book controller, added an entry, and assert true that the FileSystem saved the addressbook** |
| **Test Method/Procedure (Code)** | **testBook.addEntry(new Person("Bob", "Saget", "1000 Maple Street",**  **"San Francisco", "CA", "55555", "1234567890"));**  **testBook.addEntry(new Person("Jack", "Frost", "4000 Oak Lane", "Seatle",**  **"WA", "12345", "0987654321"));**  **testBook.setTitle("Kyle");**  **try { assertTrue(FileSystem.saveFile(testBook,testBook.getTitle()));**  **} catch (IOException e1) {**  **}** |
| **Expected Output** | **AssertTrue that the file system saved the address book into the user computer** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **97.5** |
| **Statement Coverage (%)** | **100** |

|  |  |
| --- | --- |
| **Test Criteria#7** | |
| **Requirement#1:** | **The user shall be able to delete a contact from the address book** |
| **Input:** | **testDeletePerson** |
| **Test method or procedure description:** | **Created an address book controller and a person object. Added the user information to the object, added the object to the controller, and deleted the object twice. The first time asserting that it deleted the contact, the second one asserting that it cannot delete what is not there.** |
| **Test Method/Procedure (Code)** | **AddressBookController controller = new AddressBookController();**  **Person bobPerson = new Person("Bob", "Saget", "1000 Maple Street",**  **"San Francisco", "CA", "55555", "1234567890");**  **controller.addEntry(bobPerson);**  **assertTrue(controller.deletePerson(bobPerson));**  **assertFalse(controller.deletePerson(bobPerson));** |
| **Expected Output** | **AssertTrue the deletion of a contact, and assertFalse the deletion of a missing contact.** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **100** |
| **Statement Coverage (%)** | **100** |

|  |  |
| --- | --- |
| **Test Criteria#8** | |
| **Requirement#1:** | **The user shall be able to retrieve the title of the address book** |
| **Input:** | **testGetTitle** |
| **Test method or procedure description:** | **Created an address book controller, set the title to “TestAddressBook” and asserted that the getTitle function returns “TestAddressBook”** |
| **Test Method/Procedure (Code)** | **testBook.setTitle("TestAddressBook");**  **assertEquals(testBook.getTitle(), "TestAddressBook");** |
| **Expected Output** | **AssertTrue the getTitle with the word “TestAddressBook”** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **100** |
| **Statement Coverage (%)** | **100** |

|  |  |
| --- | --- |
| **Test Criteria#9** | |
| **Requirement#1:** | **The user shall be able to edit a current contact in the address book** |
| **Input:** | **testEditPerson** |
| **Test method or procedure description:** | **Created an address book controller and three Person objects. Added the first Person object to the controller and then edited the first person object with the information of the second person object. Then asserted that the second Person object is not null. Then the second person object was edited with the third person object and asserted that the third person object name is not the second object first and last name** |
| **Test Method/Procedure (Code)** | **AddressBookController controller = new AddressBookController();**  **Person bobPerson = new Person("Bob", "Saget", "1000 Maple Street",**  **"San Francisco", "CA", "55555", "1234567890");**  **Person bobPersonEdit = new Person("Bobby", "Sageto", "2000 Maple Street",**  **"Santo Francisco", "FL", "12345", "9876543210");**  **Person bobPersonEdit2 = new Person("Bo", "Sag", "2000 Maple Street",**  **"Sn Francisco", "NC", "98765", "555555555");**  **controller.addEntry(bobPerson);**  **controller.editPerson(bobPerson, bobPersonEdit);**  **assertFalse((controller.findEntry("Bobby", "Sageto")) == null);**  **controller.editPerson(bobPersonEdit2, bobPersonEdit);**  **assertFalse((controller.findEntry("J", "F")) != null);** |
| **Expected Output** | **AssertFalse the edit of the first object is not null and AssertFalse the edit of the second object is the third object** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **94** |
| **Statement Coverage (%)** | **100** |

|  |  |
| --- | --- |
| **Test Criteria#10** | |
| **Requirement#1:** | **The user shall be able to search for an entry in the address book** |
| **Input:** | **testFindEntry()** |
| **Test method or procedure description:** | **Created an address book controller and a Person object. Added the information to the Person object and assert that the findEntry first name and last name is the same as the person object** |
| **Test Method/Procedure (Code)** | **AddressBookController controller = new AddressBookController();**  **Person bobPerson = new Person("Bob", "Saget", "1000 Maple Street",**  **"San Francisco", "CA", "55555", "1234567890");**  **assertFalse((controller.findEntry("Bobny", "Saget")) == bobPerson);** |
| **Expected Output** | **AssertFalse that an incorrect name equals to the person object** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **92.3** |
| **Statement Coverage (%)** | **100** |

|  |  |
| --- | --- |
| **Test Criteria#11** | |
| **Requirement#1:** | **The user shall be able to select the contact from the address book** |
| **Input:** | **testGetEntry()** |
| **Test method or procedure description:** | **Created an address book controller and a Person object. Added the information to the Person object and then added them to the controller. Then assert that user search can select the person object** |
| **Test Method/Procedure (Code)** | **AddressBookController controller = new AddressBookController();**  **Person bobPerson = new Person("Bob", "Saget", "1000 Maple Street",**  **"San Francisco", "CA", "55555", "1234567890");**  **Person johnPerson = new Person("John", "Smith", "4240 Oak Ave",**  **"San Diego", "CA", "44444", "0987654321");**  **controller.addEntry(bobPerson);**  **controller.addEntry(johnPerson);**    **ArrayList<Person> ent = controller.getEntries();**  **assertTrue(ent.get(0).getFirstName().equals("Bob"));** |
| **Expected Output** | **AssertTrue that the entry that was searched can be selected for edit or deletion** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **100** |
| **Statement Coverage (%)** | **100** |

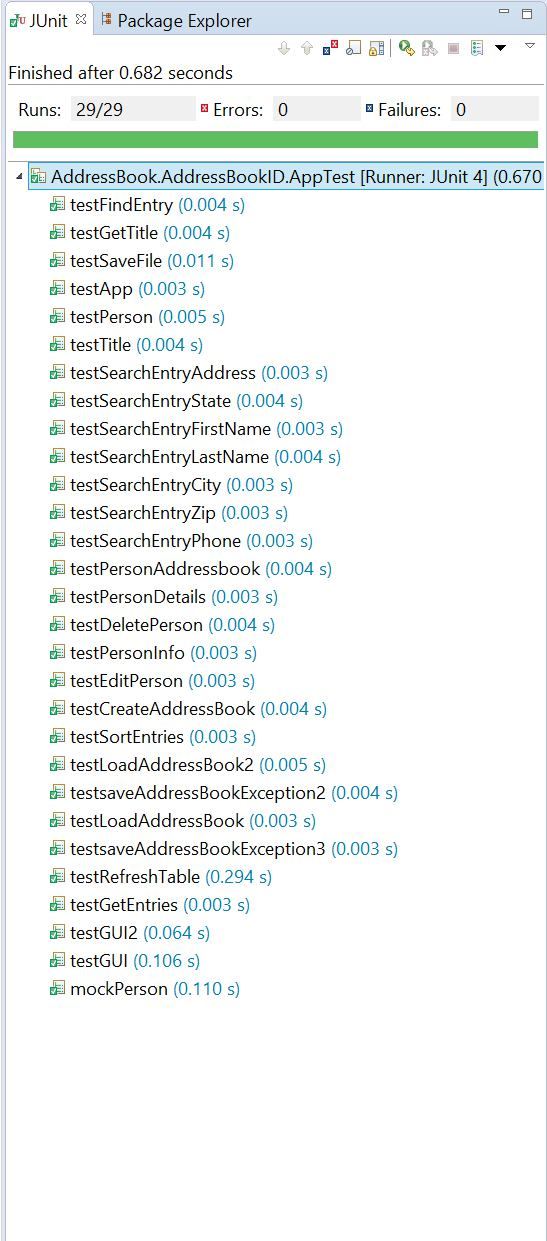
|  |  |
| --- | --- |
| **Test Criteria#12** | |
| **Requirement#1:** | **The program shall be able to sort the contacts in the address book in order** |
| **Input:** | **testSortEntries()** |
| **Test method or procedure description:** | **Created an address book and 2 person objects. Added the person information into the person objects. Then added the objects into the controller. Finally sorted the ArrayList in the controller and assert that the first entry is on ascending order (Saget < Smith)** |
| **Test Method/Procedure (Code)** | **AddressBookController controller = new AddressBookController();**  **Person johnPerson = new Person("John", "Smith", "4240 Oak Ave",**  **"San Diego", "CA", "44444", "0987654321");**  **Person bobPerson = new Person("Bob", "Saget", "1000 Maple Street",**  **"San Francisco", "CA", "55555", "1234567890");**  **controller.addEntry(johnPerson);**  **controller.addEntry(bobPerson);**  **controller.sortEntries();**  **ArrayList<Person> ent = controller.getEntries();**  **assertTrue(ent.get(0).getLastName().equals("Saget"));** |
| **Expected Output** | **AssertTrue that the first entry is on ascending order and the first in order.** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **100** |
| **Statement Coverage (%)** | **100** |

|  |  |
| --- | --- |
| **Test Criteria#13** | |
| **Requirement#1:** | **The user shall be able to load an address book saved in the system.** |
| **Input:** | **testLoadAddressBook()** |
| **Test method or procedure description:** | **Created an address book and a file system. Made a string path of “Kyle.ser” which is in the system. Loaded the path into the program and assert that is not null the loaded address book.** |
| **Test Method/Procedure (Code)** | **AddressBookController s = new AddressBookController();**  **Person person = new Person("Bob", "Saget", "1000 Maple Street",**  **"San Francisco", "CA", "55555", "1234567890");**  **FileSystem load = new FileSystem();**  **String path = "Kyle.ser";**  **s.addEntry(person);**  **load.loadAddressBook(path);**  **String title = "addressbook";**  **s.setTitle(title);**  **assertNotNull(load.loadAddressBook(path));** |
| **Expected Output** | **AssertNotNull the loading of the address book from the given path** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **100** |
| **Statement Coverage (%)** | **100** |

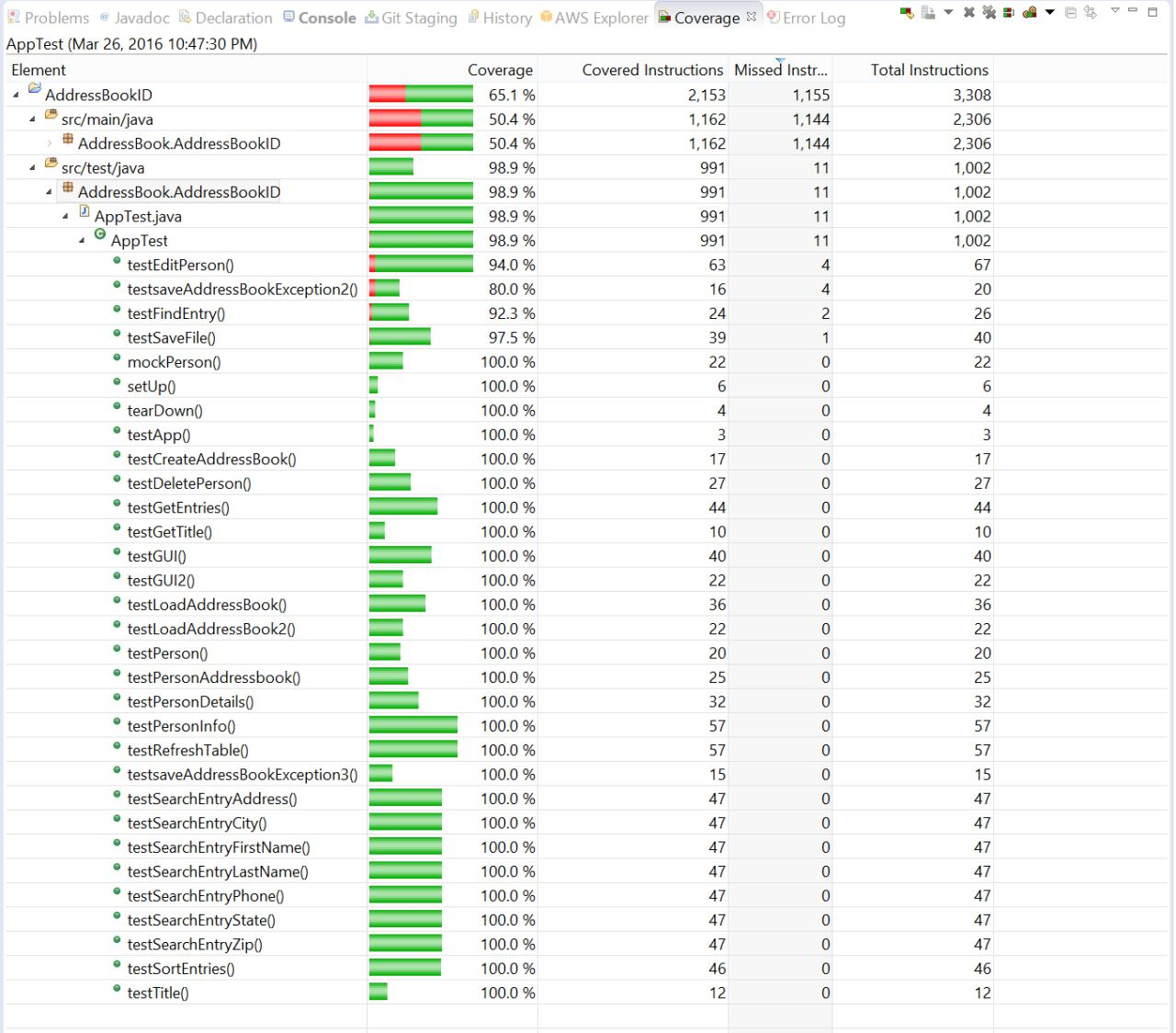
|  |  |
| --- | --- |
| **Test Criteria#14** | |
| **Requirement#1:** | **The user shall be able to interact with a graphical user interface** |
| **Input:** | **testGUI()** |
| **Test method or procedure description:** | **Created an address book and an address book graphical user interface. Created a NameList and retrieved the size of the list. Then set the controller of the GUI and assert that both are the same** |
| **Test Method/Procedure (Code)** | **AddressBookController x = new AddressBookController();**  **AddressBookGUI y = new AddressBookGUI();**  **AddressBookGUI ff = new AddressBookGUI();**  **NameList z = new NameList();**  **z.getElementAt(1);**  **int element = z.getSize();**  **assertEquals(element, 0);**  **y.setController(x);**  **ff.setController(x);**  **assertSame(y.getController(), ff.getController());** |
| **Expected Output** | **AssertSame the GUI controllers that were added** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **100** |
| **Statement Coverage (%)** | **100** |

|  |  |
| --- | --- |
| **Test Criteria#15** | |
| **Requirement#1:** | **The user shall be able to set all his user information and retrieve it** |
| **Input:** | **testPersonInfo** |
| **Test method or procedure description:** | **Created a person object and added the full name, address and phone number. Then asserted that the information matched from the get functions to the input information.** |
| **Test Method/Procedure (Code)** | **Person p = new Person("Julian", "Rivers", "33916");**  **p.setFirstName("Juliana");**  **p.setLastName("Juliusini");**  **p.setPhone("1234567890");**  **p.setZip("12345");**  **p.setState("New York");**  **p.setCity("Orlando");**  **p.setAddress("123 Main Street");**  **assertEquals(p.getFirstName(), "Juliana");**  **assertEquals(p.getCity(), "Orlando");**  **assertEquals(p.getAddress(), "123 Main Street");**  **assertEquals(p.getLastName(), "Juliusini");**  **assertEquals(p.getPhone(), "1234567890");**  **assertEquals(p.getState(), "New York");**  **assertEquals(p.getZip(), "12345");** |
| **Expected Output** | **AssertEquals the user's name, address and phone number.** |
| **Test Result (Actual Output)** | **Pass** |
| **Discrepancies** | **N/A** |
| **Branch Coverage (%)** | **100** |
| **Statement Coverage (%)** | **100** |

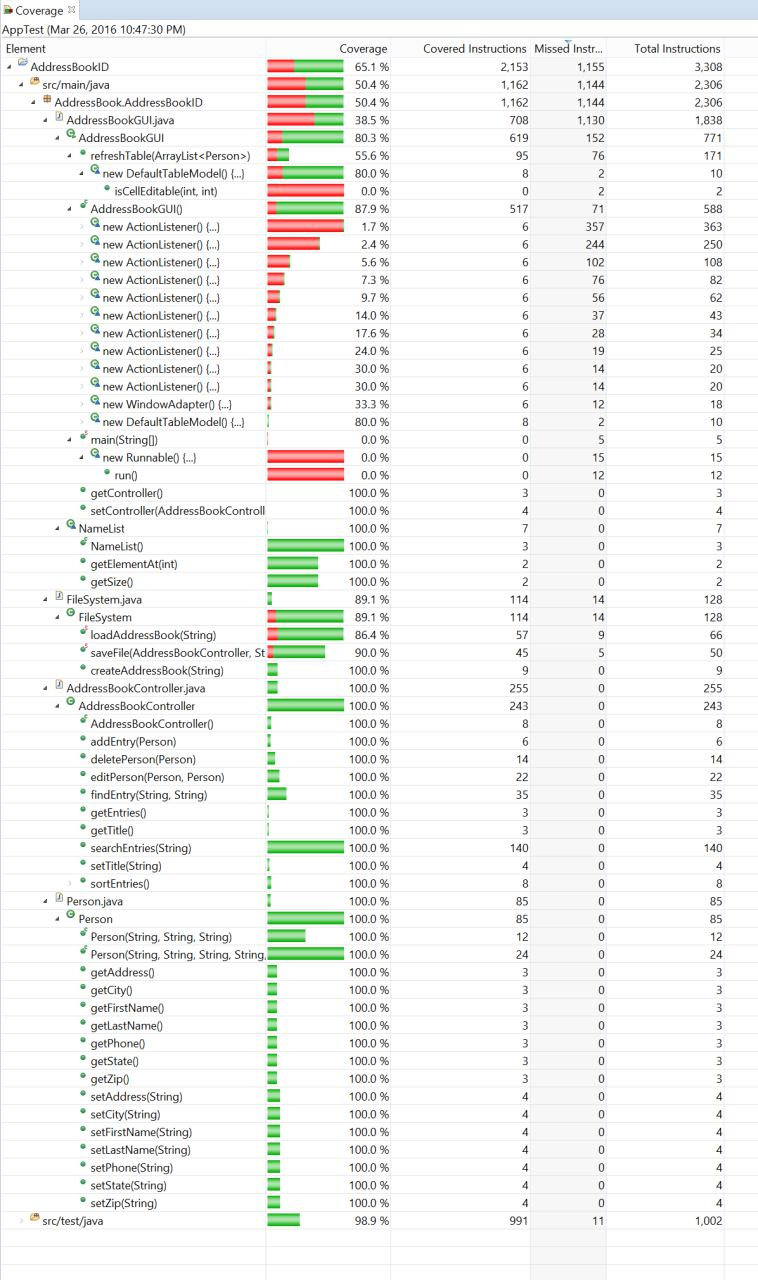
**5.2. Screenshots**



JUnit Tests



ECLEmma Branch Coverage without GUI



ECLEmma Overall Branch Coverage

**5.3. Issues**

The results of the JUnit tests passed for everything with a few exceptions. The GUI has very little coverage and this is primarily due to the anonymous inner classes that are used for the many ActionListeners required to handle each user input.. These inner classes are not able to be tested with JUnit. Other places where there was little coverage included that places where there was exception handling. The reason these spots were missed is because the code inside of the exception handlers could not be tested either.

**5.4. Resources**

JUnit, EclEmma, and EasyMock were used for testing and calculating coverage of the tests in this program. Maven was used for handling dependencies and GitHub was used for version control. All coding was done in the Eclipse IDE.

**5.5. Conclusion**

Ultimately test coverage was good, with the Person class having 100 %, AddressBookController having 100 %, FileSystem 89.1 %, and AddressBookGUI 38.5 %. This gives us a total coverage of 64.9 % if you include the GUI, or 96.37 % without it. As stated before, much of the GUI was difficult or impossible to test with EclEmma, and the relatively large size of it brings the overall coverage down significantly. For future projects, it would be much easier to perform the testing concurrently with the implementation so that the testing process is much smoother and so that methods could be adjusted to accommodate the tests.

**6. References**

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

**7. Appendix**

**AddressBookGUI.java**

package AddressBook.AddressBookID;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.EventQueue;

import java.awt.Font;

import java.awt.Toolkit;

import java.util.ArrayList;

import java.util.Observable;

import javax.swing.AbstractListModel;

import javax.swing.JButton;

import javax.swing.JDialog;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JList;

import javax.swing.JMenuItem;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTextField;

import javax.swing.RowSorter;

import javax.swing.SwingConstants;

import javax.swing.SwingUtilities;

import javax.swing.border.EmptyBorder;

import javax.swing.event.TableModelEvent;

import javax.swing.filechooser.FileNameExtensionFilter;

import javax.swing.table.DefaultTableModel;

import javax.swing.table.TableModel;

import javax.swing.table.TableRowSorter;

import javax.swing.JMenuBar;

import javax.swing.JMenu;

import java.awt.GridLayout;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;

import java.awt.print.PrinterException;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import javax.swing.JTable;

import javax.swing.JScrollBar;

import javax.swing.JScrollPane;

import javax.swing.border.BevelBorder;

import javax.swing.ListSelectionModel;

import java.awt.FlowLayout;

public class AddressBookGUI extends JFrame {

private JPanel contentPane;

AddressBookController controller;

AbstractListModel<Person> nameListModel;

NameList nameList;

JButton addButton;

JButton editButton;

JButton deleteButton;

JButton sortByNameButton;

JButton sortByZipButton;

JMenuItem newItem;

JMenuItem openItem;

JMenuItem saveItem;

JMenuItem saveAsItem;

JMenuItem printItem;

JMenuItem quitItem;

private JPanel listPanel;

private JPanel buttonPanel;

private JFrame frame;

JFileChooser chooser = new JFileChooser();

FileSystem fs = new FileSystem();

private File file;

private JScrollPane scrollPane;

private JTable table;

private JPanel panel;

private JTextField searchField;

private JPanel panel\_1;

private JPanel panel\_2;

/\*\*

\* Launch the application.

\*/

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();

}

}

});

}

/\*\*

\* Create the frame.

\*/

public AddressBookGUI() {

frame = this;

setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE);

setBounds(0, 0, 1280, 720);

//setResizable(false); //make window non-resizable

frame.addWindowListener(new java.awt.event.WindowAdapter() {

@Override

public void windowClosing(java.awt.event.WindowEvent windowEvent) {

if (JOptionPane.showConfirmDialog(frame,

"Are you sure to close this window?",

"Really Closing?", JOptionPane.YES\_NO\_OPTION,

JOptionPane.QUESTION\_MESSAGE) == JOptionPane.YES\_OPTION) {

System.exit(0);

}

}

});

Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();

this.setLocation(dim.width / 2 - this.getSize().width / 2, dim.height

/ 2 - this.getSize().height / 2);

FileNameExtensionFilter filter = new FileNameExtensionFilter(

"Address Book Files (.ser)", "ser");

chooser.setFileFilter(filter);

chooser.setAcceptAllFileFilterUsed(false);

JMenuBar menuBar = new JMenuBar();

setJMenuBar(menuBar);

JMenu mnNewMenu = new JMenu("File");

mnNewMenu.setFont(new Font("arial", Font.BOLD, 20));

menuBar.add(mnNewMenu);

controller = new AddressBookController();

newItem = new JMenuItem("New Address Book");

newItem.setFont(new Font("arial", Font.BOLD, 20)); //font

newItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent ev) {

file = null;

String path = "";

int returnVal = chooser.showSaveDialog(null);

if (returnVal == chooser.APPROVE\_OPTION) { // OK button pressed

// by user

file = chooser.getSelectedFile(); // get File selected by

// user

if (file.getName().contains(".ser")) {

try {

BufferedWriter bw = new BufferedWriter(

new FileWriter(file));

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} // use its name

} else {

file = new File(file.getAbsolutePath() + ".ser");

try {

BufferedWriter bw = new BufferedWriter(

new FileWriter(file));

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} // use its name

}

}

}

});

mnNewMenu.add(newItem);

openItem = new JMenuItem("Open...");

openItem.setFont(new Font("arial", Font.BOLD, 20)); //font

openItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

// In response to a button click:

int returnVal = chooser.showOpenDialog(AddressBookGUI.this);

if (returnVal == chooser.APPROVE\_OPTION) {

file = chooser.getSelectedFile();

String fileName = file.getAbsolutePath();

controller = fs.loadAddressBook(fileName.substring(0,

fileName.indexOf(".ser")));

}

refreshTable(null);

}

});

mnNewMenu.add(openItem);

saveItem = new JMenuItem("Save");

saveItem.setFont(new Font("arial", Font.BOLD, 20)); //font

saveItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if (file == null) {

JOptionPane.showMessageDialog(saveItem,

"There is no open file!");

} else {

try {

fs.saveFile(controller, file.getAbsolutePath());

} catch (IOException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

}

refreshTable(null);

}

});

mnNewMenu.add(saveItem);

saveAsItem = new JMenuItem("Save as...");

saveAsItem.setFont(new Font("arial", Font.BOLD, 20)); //font

saveAsItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent ev) {

String path = "";

int returnVal = chooser.showSaveDialog(null);

if (returnVal == chooser.APPROVE\_OPTION) { // OK button pressed

// by user

file = chooser.getSelectedFile(); // get File selected by

// user

if (file.getName().contains(".ser")) {

try {

BufferedWriter bw = new BufferedWriter(

new FileWriter(file));

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} // use its name

try {

fs.saveFile(controller, file.getAbsolutePath());

} catch (IOException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

} else {

file = new File(file.getAbsolutePath() + ".ser");

try {

BufferedWriter bw = new BufferedWriter(

new FileWriter(file));

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} // use its name

try {

fs.saveFile(controller, file.getAbsolutePath());

} catch (IOException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

}

}

refreshTable(null);

}

});

mnNewMenu.add(saveAsItem);

printItem = new JMenuItem("Print");

printItem.setFont(new Font("arial", Font.BOLD, 20)); //font

printItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

try {

boolean complete = table.print();

if (complete) {

/\* show a success message \*/

JOptionPane.showMessageDialog(saveItem,

"Printing complete!");

} else {

/\* show a message indicating that printing was cancelled \*/

System.err.println("User cancelled printing");

}

} catch (PrinterException pe) {

/\* Printing failed, report to the user \*/

}

}

});

mnNewMenu.add(printItem);

JMenuItem mntmQuit = new JMenuItem("Quit");

mntmQuit.setFont(new Font("arial", Font.BOLD, 20)); //font

mntmQuit.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

if (JOptionPane.showConfirmDialog(frame,

"Are you sure to close this window?",

"Really Closing?", JOptionPane.YES\_NO\_OPTION,

JOptionPane.QUESTION\_MESSAGE) == JOptionPane.YES\_OPTION) {

frame.dispose();

}

}

});

mnNewMenu.add(mntmQuit);

contentPane = new JPanel();

contentPane.setBorder(new EmptyBorder(10, 10, 10, 10));

setContentPane(contentPane);

contentPane.setLayout(new BorderLayout(0, 0));

nameList = new NameList();

listPanel = new JPanel();

contentPane.add(listPanel, BorderLayout.CENTER);

listPanel.setLayout(new BorderLayout(0, 0));

table = new JTable();

table.setFont(new Font("arial", Font.BOLD, 45));

table.setAutoCreateRowSorter(true);

table.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

ArrayList<Person> entries = controller.getEntries();

Object[][] people = new Object[entries.size()][7];

for (int i = 0; i < entries.size(); i++) {

people[i][0] = entries.get(i).getFirstName();

people[i][1] = entries.get(i).getLastName();

people[i][2] = entries.get(i).getPhone();

people[i][3] = entries.get(i).getAddress();

people[i][4] = entries.get(i).getCity();

people[i][5] = entries.get(i).getState();

people[i][6] = entries.get(i).getZip();

}

@SuppressWarnings("serial")

DefaultTableModel myModel = new DefaultTableModel(people, new String[] {

"First Name", "Last Name", "Phone Number", "Address", "City",

"State", "Zip" }) {

@Override

public boolean isCellEditable(int row, int column) {

return false;

}

};

table.setModel(myModel);

scrollPane = new JScrollPane(table);

listPanel.add(scrollPane, BorderLayout.CENTER);

buttonPanel = new JPanel();

contentPane.add(buttonPanel, BorderLayout.SOUTH);

buttonPanel.setLayout(new GridLayout(1, 0, 0, 0));

addButton = new JButton("Add Contact");

addButton.setBackground(Color.lightGray);

addButton.setFont(new Font("Arial", Font.PLAIN, 25)); //Add Contact Button font size

addButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JPanel p = new JPanel(new BorderLayout(5, 5));

JPanel labels = new JPanel(new GridLayout(7, 2));

labels.setFont(new Font("arial", Font.PLAIN, 20));

labels.add(new JLabel("First Name", SwingConstants.RIGHT));

labels.add(new JLabel("Last Name", SwingConstants.RIGHT));

labels.add(new JLabel("Phone Number", SwingConstants.RIGHT));

labels.add(new JLabel("Address ", SwingConstants.RIGHT));

labels.add(new JLabel("City", SwingConstants.RIGHT));

labels.add(new JLabel("State ", SwingConstants.RIGHT));

labels.add(new JLabel("Zip", SwingConstants.RIGHT));

p.add(labels, BorderLayout.WEST);

JPanel controls = new JPanel(new GridLayout(7, 2));

JTextField firstNameField = new JTextField();

controls.add(firstNameField);

JTextField lastNameField = new JTextField();

controls.add(lastNameField);

JTextField phoneField = new JTextField();

controls.add(phoneField);

JTextField addressField = new JTextField();

controls.add(addressField);

JTextField cityField = new JTextField();

controls.add(cityField);

JTextField stateField = new JTextField();

controls.add(stateField);

JTextField zipField = new JTextField();

controls.add(zipField);

p.add(controls, BorderLayout.CENTER);

if (file == null) {

JOptionPane

.showMessageDialog(frame,

"Please open an address book to add a new contact!");

} else {

int returnVal = JOptionPane.showConfirmDialog(frame, p,

"Create Contact", JOptionPane.OK\_CANCEL\_OPTION);

if (returnVal == JOptionPane.OK\_OPTION) {

// required parameters

Person person = new Person(firstNameField.getText(),

lastNameField.getText(), zipField.getText());

// optional parameters

if (!phoneField.getText().equals(""))

person.setPhone(phoneField.getText());

if (!addressField.getText().equals(""))

person.setAddress(addressField.getText());

if (!cityField.getText().equals(""))

person.setCity(cityField.getText());

if (!stateField.getText().equals(""))

person.setState(stateField.getText());

controller.addEntry(person);

// reset fields

firstNameField.setText("");

lastNameField.setText("");

phoneField.setText("");

addressField.setText("");

cityField.setText("");

stateField.setText("");

zipField.setText("");

}

}

refreshTable(null);

}

});

buttonPanel.add(addButton);

editButton = new JButton("Edit Contact");

editButton.setBackground(Color.lightGray);

editButton.setFont(new Font("Arial", Font.PLAIN, 25)); //Edit button font size

editButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JTextField firstNameField = null;

JTextField lastNameField = null;

JTextField phoneField = null;

JTextField addressField = null;

JTextField cityField = null;

JTextField stateField = null;

JTextField zipField = null;

if (file == null) {

JOptionPane.showMessageDialog(frame,

"Please open an address book to edit a contact!");

} else {

int returnVal = -1;

JPanel p = new JPanel(new BorderLayout(5, 5));

JPanel labels = new JPanel(new GridLayout(7, 2));

labels.add(new JLabel("First Name", SwingConstants.RIGHT));

labels.add(new JLabel("Last Name", SwingConstants.RIGHT));

labels.add(new JLabel("Phone Number", SwingConstants.RIGHT));

labels.add(new JLabel("Address ", SwingConstants.RIGHT));

labels.add(new JLabel("City", SwingConstants.RIGHT));

labels.add(new JLabel("State ", SwingConstants.RIGHT));

labels.add(new JLabel("Zip", SwingConstants.RIGHT));

p.add(labels, BorderLayout.WEST);

int rowID = table.getSelectedRow();

if (rowID == -1) {

JOptionPane.showMessageDialog(frame,

"Please select a row to edit a contact!");

} else {

JPanel controls = new JPanel(new GridLayout(7, 2));

firstNameField = new JTextField((String) table

.getValueAt(rowID, 0));

controls.add(firstNameField);

lastNameField = new JTextField((String) table

.getValueAt(rowID, 1));

controls.add(lastNameField);

phoneField = new JTextField((String) table.getValueAt(

rowID, 2));

controls.add(phoneField);

addressField = new JTextField((String) table

.getValueAt(rowID, 3));

controls.add(addressField);

cityField = new JTextField((String) table.getValueAt(

rowID, 4));

controls.add(cityField);

stateField = new JTextField((String) table.getValueAt(

rowID, 5));

controls.add(stateField);

zipField = new JTextField((String) table.getValueAt(

rowID, 6));

controls.add(zipField);

p.add(controls, BorderLayout.CENTER);

returnVal = JOptionPane.showConfirmDialog(frame, p,

"Edit Contact", JOptionPane.OK\_CANCEL\_OPTION);

}

rowID = table.getSelectedRow();

Person oldPerson = null;

if (rowID != -1) {

String firstName = (String) table.getValueAt(rowID, 0);

String lastName = (String) table.getValueAt(rowID, 1);

oldPerson = controller.findEntry(firstName, lastName);

} else {

}

if (returnVal == JOptionPane.OK\_OPTION) {

// required parameters

Person newPerson = new Person(firstNameField.getText(),

lastNameField.getText(), zipField.getText());

// optional parameters

if (!phoneField.getText().equals(""))

newPerson.setPhone(phoneField.getText());

if (!addressField.getText().equals(""))

newPerson.setAddress(addressField.getText());

if (!cityField.getText().equals(""))

newPerson.setCity(cityField.getText());

if (!stateField.getText().equals(""))

newPerson.setState(stateField.getText());

controller.editPerson(oldPerson, newPerson);

// reset fields

firstNameField.setText("");

lastNameField.setText("");

phoneField.setText("");

addressField.setText("");

cityField.setText("");

stateField.setText("");

zipField.setText("");

}

refreshTable(null);

}

}

});

buttonPanel.add(editButton);

deleteButton = new JButton("Delete Contact");

deleteButton.setFont(new Font("Arial", Font.PLAIN, 25)); //Delete button font size

deleteButton.setBackground(Color.lightGray);

deleteButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if (file == null) {

JOptionPane.showMessageDialog(frame,

"Please open an address book to delete a contact!");

} else {

int rowID = table.getSelectedRow();

if (rowID != -1) {

String firstName = (String) table.getValueAt(rowID, 0);

String lastName = (String) table.getValueAt(rowID, 1);

controller.deletePerson(controller.findEntry(firstName,

lastName));

} else {

JOptionPane.showMessageDialog(frame,

"Please select a row to delete a contact!");

}

}

refreshTable(null);

}

});

buttonPanel.add(deleteButton);

panel = new JPanel();

contentPane.add(panel, BorderLayout.NORTH);

panel.setLayout(new GridLayout(0, 3, 0, 0));

panel\_1 = new JPanel();

panel.add(panel\_1);

searchField = new JTextField();

searchField.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

ArrayList<Person> results = controller.searchEntries(searchField.getText());

refreshTable(results);

}

});

panel.add(searchField);

searchField.setColumns(10);

panel\_2 = new JPanel();

panel.add(panel\_2);

}

public void refreshTable(ArrayList<Person> searchResults) {

listPanel.remove(scrollPane);

table = new JTable();

table.setFont(new Font("Arial", Font.PLAIN, 20)); //populated list font size

ArrayList<Person> entries;

if(searchResults == null)

entries = controller.getEntries();

else {

System.out.println("searchResults != null");

entries = searchResults;

}

Object[][] people = new Object[entries.size()][7];

for (int i = 0; i < entries.size(); i++) {

people[i][0] = entries.get(i).getFirstName();

people[i][1] = entries.get(i).getLastName();

people[i][2] = entries.get(i).getPhone();

people[i][3] = entries.get(i).getAddress();

people[i][4] = entries.get(i).getCity();

people[i][5] = entries.get(i).getState();

people[i][6] = entries.get(i).getZip();

}

@SuppressWarnings("serial")

DefaultTableModel myModel = new DefaultTableModel(people, new String[] {

"First Name", "Last Name", "Phone Number", "Address", "City",

"State", "Zip" }) {

@Override

public boolean isCellEditable(int row, int column) {

return false;

}

};

table.setModel(myModel);

table.setAutoCreateRowSorter(true);

scrollPane = new JScrollPane(table);

listPanel.add(scrollPane, BorderLayout.CENTER);

listPanel.revalidate();

}

public AddressBookController getController() {

return controller;

}

public void setController(AddressBookController controller) {

this.controller = controller;

}

}

class NameList extends AbstractListModel<Person> {

public NameList() {

}

public Person getElementAt(int arg0) {

return null;

}

public int getSize() {

return 0;

}

}