By using JUnit test, I was able to develop & test my ContactService class to show clear alignment between the software requirements & the implementation of its’ functionality. The class was designed to manage contacts with select functions such as adding, deleting, & updating the contact information. I had to make sure that the testing approach aligned with the given requirements; verifying its core functions, validating the input constraints, & handle all edge cases differently. The example below is from my test case for adding a contact to the list successfully & properly storing its attributes.

Ex.

*@Test*

*public void testAddContact() {*

*ContactService service = new ContactService();*

*service.addContact("Mike", "Jones", "2225678900", "1222 Side Peak Dr");*

*assertEquals(1, service.contactList.size());*

*assertEquals("John", service.contactList.get(0).getFirstName());}}}*

I also validated the deletion functionality with a test case that confirms a contact was removed from the list:

Ex.

*public void deleteContact(String ID) { int intID = Integer.valueOf(ID); for (int i = 0; i < ContactService.contactList.size(); i++) { if (ContactService.contactList.get(i).getContactID() == intID) { contactList.remove(i); } } }*

Included in both tests are just examples of the core functionality. Being able to add & delete contacts is a requirement. Additionally, I also focused on both positive & negative scenarios to ensure comprehensive coverage. One example is a test case to confirm that editing a contact’s first name behaves as expected.

*Ex.*

*@Test*

*public void editFirstName(String contactID, String firstName) { for (Contact iter : contactList) { if (iter.getContactID() == Integer.valueOf(contactID)) { iter.setFirstName(firstName); } } }*

On the negative side, I also created a test that ensures invalid input results in an appropriate exception:

Ex.

*@Test(expected = IllegalArgumentException.class) public void testAddContactWithInvalidPhoneNumber() { ContactService service = new ContactService(); // This invalid phone number should trigger an exception service.addContact("Mike", "John", "2225678900", "1222 Side Peak Dr"); /// Invalid phone number*

I made sure that if an invalid phone number was entered, such as one that doesn’t

Meet its required 10-digit format, the system should throw an exception. I was able to confirm this by designing tests that simulated such invalid inputs, making sure errors were raised appropriately.

In efforts to further strengthen the system, I added input validation to verify that the phone number had to meet exact format before the contact info was saved. This testing was extensively performed to make sure that phone numbers that didn’t meet strict criteria couldn’t be saved. By recognizing usage of basic lists for storing contacts could lead to slower lookups & deletions as the dataset grows. I was able to optimize the data storage by switching to a more efficient structure that offers constant-time access.