**Jacob Alspaw**

**Database Testing Report**

**JUnit Tests**

“*First, middle, last*” and “*0, 1, many*” are ways to test a program. “*First*” refers to the first part of the tested item. The method should be tested against the first part of the item. “*Middle*” refers to the middle part of the tested item. The method should be tested against the middle part of this item. “*Last*” refers to the last part of the tested item. The method should be tested against the last part of the tested item. “*0*” is to use test cases of values of 0. “*1*” is to use values of 1 and non-zero values around one. “*Many*” is to use values larger than non-zero values around one. If the method is subjected to these two forms of testing, then most test cases have been considered and the method can be thought of as well-checked.

The JUnit tests I have used cover all of these cases. I decided to do an overall test of program, rather than testing individual classes. Each class was still tested for the above conditions. I created and set the values of two contacts. These contacts were made for two reasons. The first contact is my own personal contact. It was made to test a Contact object that was fully parameterized. The second contact was essentially someone who let’s say wanted to stay anonymous. This person may have chosen to not disclose parts of their personal information. So I left the values null, which the program will handle. All six classes were successful. For the Database class, I instantiated a database and added the contacts to it. I tested making new lists, deleting contacts, adding them, the three lookup methods, getting the lists, and making indexes. These were tested for null arguments and non-null arguments. For the Contact class, we tested by creating a contact and getting the methods we made for it. The setter methods were also tested. The toString method, along with the getters were tested for null and non-null values. The equals method was tested between contacts that were different and the same, using null, and non-null values. This was essential because the object .equals method would throw null pointer exceptions and I had to accommodate to this. The class database type was not needed to be tested because it is really just an interface and we cannot test the method stub. LLNode was a class given to us in class so it is fine to not test. We can assume it is already correct. Mainly, due to the nature of this project’s nature, the biggest case for testing is null values. Could the program handle them? Mine could.

**\*\*The JUnit testing resulted in all correct classes\*\***

To extra-verify the classes worked, I just played around with my completed code.