**Employee Database: Testing Outline**

**Overview**

First, each part of the program (in this case, each tab) will be tested separately. In each part, white box (by the developer) and black box (by a person who does not know the code) will be done separately. Note that while testing each tab’s functionality, one may use the “Display” tab to keep track of the information held in the database and in each employee, but then this is assuming that the testing of the “Display” tab has already taken place and it is working correctly. Instead, the tester may want to use debug mode and pause the program at appropriate points to ensure the data in the database and in each employee is as it should be. Then, the text file reading/writing will be tested by a developer. After that, some parts of the program will be tested altogether (and some parts will not be tested together; an explanation for this will be given). Finally, an example of a specific test case will be included at the end.

**Adding**

In this tab, first having different text fields empty will be tested. For example, the first test case would be clicking the “Add New Employee” button without entering anything into any field (and not selecting location, not selecting “Part-time” or “Full-time”, etc.) Then, a single field will be left empty/not selected, while all others will be correctly entered (i.e. in the correct format.) All such fields will be tested this way. These should all give the appropriate error message, and should not allow the employee to be added. For some cases, a developer should be doing the testing, and should use the debug mode to see (for certain) that the employee was not added to the database.

Next, all text fields will have something written into them, but, one by one, whatever is entered will be in an invalid type (i.e. entering a string when an integer is required). (First name and last name allow any kind of string, so these fields cannot be tested, and depend on the user to input the correct names.) So, for example, for Employee Number, the tester would enter “abc”, while keeping all other entries in the correct format, and this should give an error. After this, while correct types will be entered, they will not be in the correct range. For example, Employee number as “-349”, or deductions rate greater than 1 or less than zero, annual salary less than 0, etc. (Since Location is selected from a drop-down menu, it cannot be tested this way (until the “Reading/Writing” part)). Then, employee number will be tested specifically, with the user entering an employee such that they have the employee number of a previously entered employee. Again, none of these should allow the employee to be added to the database, and randomly selected cases should be done by the developer in debug mode, to ensure that an employee was not added.

Then, extreme test case scenarios will be done. These will be the values that are *just* valid and can be entered, but anything greater (or less than, in the other extreme) would not be valid to enter. In cases where the extremities are known to everyone (e.g. deductions rate cannot be 1 or greater, cannot be less than zero (which would mean testing 0.9999, and testing 0)), it will be done as black box testing. In cases where the extremities may not be known to everyone (e.g. the maximum number of characters allowed in a string, the maximum integer allowed, etc.), the testing would have to be done by the developer, or at least someone that knows how to code in Java. All of these cases *should* allow the employee to be added to the database.

Lastly, for this tab, black box ad-hoc testing will be done, with a user entering random valid values for all fields, and each of these should allow the employee to be added to the database. This will be done in case the developer unknowingly added certain restrictions to some fields that should not be there. For example, for whatever reason, the database might not allow someone to add an employee with a deductions rate of a previous employee, even though deductions rate need not be unique for every employee. If any of these tests do not allow the employee to be added, then there is a problem in the code.

**Searching, Removing, & Modifying**

Each of these tabs begin with a prompt for the user to enter the employee’s employee number they wish to search for to see if they are in the database/to remove them/to modify them. Therefore, initially, they can all be tested with similar test cases. (However, the testing of each of these will not be done simultaneously; this is further explained in the “Combined” section.)

So, first, for each of them, the first test would be clicking the “Search” button without having typed in any number. Then, one would type in a random string of letters (containing letters, punctuation, etc.), and then click the search button. Then, one would type in a number that is out of range (for example, “-390”, without quotation marks), and then click the search button. Each of these cases should result in an error message that tells the user to enter an appropriate number in the search field.

Continuing the similar test cases, next, one would (after having added some (e.g. 10) employees in the database) enter a valid employee number (i.e. an integer greater than zero) into the search field, except that there will be no employee in the database that exists with that number. A message saying that an employee with that number does not exist should appear. Then, one would search for an employee number that does exist in the database. In the Search tab, a message simply saying that the employee was found should appear. In the Remove tab, a prompt asking the user if they wish to remove the employee (or not) should appear. In the Modify tab, a prompt asking the user if they wish to modify the employee (or not) should appear.

The search tab does not have any further functionality, so its testing is complete. For the remove tab, cases which do not remove the employee (i.e. click “Cancel”) as well as cases that include removing the employee will be included, and in each case, each employee would be different (i.e. test removing the only employee in the database, test removing the last employee added to the database, test removing the first employee added to the database, test removing part time vs full time, etc.)

As for modify, different cases will involve modifying the employee, while some will test clicking “Cancel”. If one clicks the cancel button, the employee should remain in the database unchanged. If one clicks the modify button, it should take one to the “Information Form” (i.e. Adding) tab, with all information of the employee correctly input already into each of the fields. In this tab, all the testing done in the “Adding” section (above) will be repeated (except Employee Number, which cannot be changed even in the modify tab), except the button clicked will now be “Modify Employee”. After doing this testing, when testing with valid information (i.e. an employee is successfully modified), one should ensure that the previous employee (the unmodified one) is removed from the database, and the new employee (the modified one) is added. (In the Display Table, even if the modified employee was one of the first employees added originally, the new modified employee will be added to the end of the table as if they were a completely new employee.) Additionally, there is a second “Cancel” button which appears when modifying the employee’s information. Similar to the previous cancel button, if one clicks this, the employee should be left in the database un-modified. This button will be tested by modifying some fields, then clicking “Cancel”, and ensuring that the employee was not modified even though those fields were.

**Reading & Writing**

The testing in this case will be, in most cases, white box testing, because one must know how the read function reads from a file and how the write function writes into a file, to see if they are doing what they should do. That is why it is recommended that a developer do this part of the testing.

First, without having a text file in the directory from which the program would read from, the tester must attempt to “Read” from the file. This should result in an error message and it should not read anything. Then, a text file without the appropriate name should be there, and again an attempt to read it should result in an error. Then, the text file (with the appropriate name) should be there, but its contents should be empty, and an attempt to read it should result in an error again. Then, the same text file containing only “0” (this tells the Reader function how many employees are in the file) should be read, and the result should not be an error, but regardless, zero employees should be added to the database (as with the other cases so far). Each of these will have cases such that employees do and don’t already exist in the database, and in each case, any employees that are already exist in the database should be removed after clicking “Read” (regardless of whether an error message is outputted or not.)

Next, reading from a file that contains random (invalid) data will be attempted. Different types of random data will be used, including special characters, numbers, etc. Also, tests that attempt to read from the file containing data that seems valid but is not in the file format specified in the User Guide will be done, (e.g. not in the correct order, having more than the required number of new lines, putting all data in a single line, etc.) All of these should result in an error and the file should not be able to be read correctly.   
 After that, a file containing almost all correct data (in the correct format) will be written. This part will contain cases akin to the “Adding” section’s testing. That is, all fields contain valid data except one of them is null, a single field will contain an incorrect data type while all others contain valid data, a single field contains the correct type but it is out of range, a single field contains an extremity of data, and so on, for each field (and combinations of some fields). Also, some tests that could not be done in the Adding section (because they were impossible), like selecting a location that the program did not allow, or putting in part time information for a full time employee, will be done in this section. All of these tests (except the one with extreme but valid data) should result in an error, and again, all existing employees in the database (if any) should be removed, while none should be added, after clicking “Read”.

After that, files written containing all valid data, all in the correct format, will be written. Attempts to read these files should be successful, with the correct number of employees (and the correct information in each employee) in the database. Different files will be written with different kinds of employees, different numbers of employees, etc.

As for saving, the first test will involve saving a database with no employees in it, with no already existing file. This should result in a text file created (titled “Employee Database”), with the number “0” written in it (only). Then, saving a database with no employees in it, with a file already created (different cases having different amounts of data in the file (if any)), will be tested, and should result in the file being overwritten with the number “0” in it (only) as the first line. After this, saving a database with employees in it onto the file will be tested, and all information of each employee should be correctly written into the file, in the correct format. After all this, successions of saving and reading from the file (one after the other) will be tested (e.g. put some employees into a database, save it, close the program, open the program and read it, should result in the same database with the same info as the prior one.)

**Combined**

Whereas before, test cases were done as “unit” testing (testing each part of the program separately), these will be integration tests. This is also where the “Display” tab will be tested. Different functions of the program will be run (Add, Modify, Remove, Search, Read, Save), and the display table will be viewed after each one, to see if it correctly changes as data changes. Additionally, net annual income will also be tested to see if it is calculated correctly and shown/rounded correctly in the display table. Note that when using Remove or Modify, if the prompt appears asking the user whether they would like to remove/modify the employee or not, one should not switch tabs at that point, and instead clicking either “Remove”/“Modify” or “Cancel”. For this reason, the functionalities of the Remove and Modify tabs (when this prompt appears) will not be tested together, as it may cause the program to remove the employee one wishes to modify, or vice versa.   
 Different tabs will be tested together in different orders as well, with sometimes invalid information. For example, one may read from a file containing an employee with an employee number of 5, then check the display table to see if it displays the correct information, then try adding a new employee with an employee number of 5 (which should result in an error), then try removing the employee with an employee number of 5 (which should be removed), then try to add (again) the new employee with an employee number of 5, which should then (finally) be added. (That would be a *single* case of testing the functionality of all tabs combined.)

This part of the testing should also contain (mostly) white box testing, as a person that knows how a program functions may knowingly or unknowingly avoid using the program’s tabs in such a way that leads to some problem. Also, because of the nature of combined testing (using different parts of the program in one go), ad-hoc tests will also be used in this part, where the tester would simply use the program and all of its functionalities in any way he/she wishes, to see if it runs as it should.

**Test Case Example**

This is an example of a test case (which would fall under the “Combined” section of the tests):

Purpose: to test if, after reading from a text file, the Display and Remove parts of the program work, and if they work, if saving to a text file works correctly after using them, using all valid data.

Pre-requisite:   
File titled “Employee Database.txt” located in the NetBeans project folder titled “Employee Database”, containing (exactly) the following (excluding the curly braces):  
{  
2

1

Ferst  
Persin

0.1

1

P

15

40

30

2

Sekind

Huemin

0.2

0

F

18000  
}

Run the program, then click “Read”. There should be no error. Go to the Display tab. There should be two rows of employees:

the first should have Emp No. as 1, First Name and Last Name as “Ferst” “Persin” respectively, Deductions as 0.1, Location as “Mississauga”, and Net Annual Income as 16200, from 15\*40\*30\*(1-0.1)

the second should have Emp No. as 2, First Name and Last Name as “Sekind” “Huemin” respectively, Deductions as 0.2, Location as “Toronto”, and Net Annual Income as 14400, from 18000\*(1-0.2)

Then, go to the remove tab, and search “1”. There should be no error, and the employee should be found. When asked whether you wish to remove the employee or not, click “Remove”. Then, go to the “Display” tab; the previous first row should be gone, and only one row of information (containing the previously second row) should be there. Then, click “Save”, and close the program. Locate the text file. It should contain (exactly) the following (excluding the curly braces):

{  
1

2

Sekind

Huemin

0.2

0

F

18000  
}

If it does not contain this, or if the display table does not display the information written above, or if at any point while running program, an error message appears, the test case has failed, because this was a test with all valid data. Otherwise, the test case has succeeded and the program’s Read, Display, Remove, Display, and Save functions all work together correctly (in that order) when dealing with valid data.