**METHODOLOGY**

1. **Asking the user for input of two decimal numbers**

The program will ask the user to input two decimal numbers. This will be Pre-Condition of running the program. If the user made a mistake in the input or did not enter a valid input, then the program will crash. The user will just then have to restart the program again.

1. Once two decimal numbers are typed-in, the input will be recorded as Two Set of Strings. From these two sets, a division of four strings will be formed. The divisions will be:
2. Numbers before the Decimal for the first decimal number
3. Numbers before the Decimal for the second decimal number
4. Numbers after the Decimal for the first decimal number
5. Numbers after the Decimal for the second decimal number
6. Next, the Strings are then compared whether the numbers before the decimal have the same length (number of digits) or not. If not, zeroes are added to even out the difference in length. Having the same number of digits is crucial in calculating the sum and differences of the two numbers. Likewise, this method also happens to the numbers after the Decimal. Then these values are stored in the variables with the name Final in it.
7. **Finding out which number is larger**

After obtaining the values for the four main crucial variables as this will be the Pre-Condition for calculating the number with the larger value.

* 1. Calculating the larger value starts from the numbers before the decimal. From the numbers before the decimal, it will be obvious that the more digits a number have the larger its value so this condition will be checked first. If this condition is met, then the larger number is already determined and there is no need to iterate each digits and compare the numbers or check for the numbers after the decimal.
  2. Otherwise, if the numbers before the decimal have the same number of digits, then it is necessary to iterate each digits and compare the values. It is a must that when comparing, digits of the same index be compared and start comparing from the rightmost digits of the numbers. Iterating, if one is larger than the other, then the number that the larger digit came from is the larger number.
  3. If both digits are the same value, then move on to the next digit (digits on the left of the current one) and do step 2 unless you are at the last digit of the String number.
  4. It can also be the case that, if the numbers before the decimal are the exact same numbers then we also have to check the numbers after the decimal and iterate on each of the digits for comparison. Do step 2 and 3 for the numbers After the Decimal.
  5. Last case will be is: if the numbers have the exact same value, then we can just store the first number as the larger number.
  6. Post-Condition will be for this is the value of the larger number stored as a String.

1. **Calculating the Sum of the Numbers**

Next calculation will be: is to calculate the sum of the two numbers. Pre-Condition for this is also the four variables with the Final name on it as the numbers will be needed with the same length for numbers before the decimal and numbers after the decimal.

1. When adding, we start off from the numbers after the decimal’s rightmost digits. From Strings we convert each characters as short by iterating on each characters using a while loop.
2. Once, we have them as Short values, we can then add the digits of the same index.
3. If there are any carry-overs, there is a Boolean that will keep track of it and therefore a ‘1’ is added to the next digit to compensate for that carry-over. If no carry over, move on to the next digit without adding ‘1’ to it and repeat step 2.
4. Once we have done adding each digits of the numbers after the decimal, the value is stored and then the same process happens for the numbers before the decimal.
5. Post-Condition for this is the String representation of the Sum of the Two Numbers.
6. **Calculating the Difference of the Two Numbers**

Calculating the difference of the two numbers undergoes almost the same process as adding. The pre-conditions are also the four numbers with the same length for each two numbers depending on their location.

1. To start subtracting, we need to determine first which number is the larger number (this is already done in Section II – Finding the Larger Number) as we need to subtract the smaller from it or else, there would be an error in our calculations.
2. Next, we then start on iterating from the rightmost digit of the number after the decimal to convert from String to Short. On each iteration, we can then now subtract the digit of the smaller number to the larger number.
3. Keep in mind of any possible borrow over (happens when the digit of the larger number is smaller than the digit of the smaller number). A Boolean borrowOver will keep track of this to compensate for any borrow over that we needed as this means that we will subtract ‘1’ from the next digit to compensate for it. If there is no borrow Over, Boolean will remain false. Move on to the next digits and repeat Step 2.
4. Once we finish subtraction from the numbers after the decimal, the same process happens to the numbers before the decimal. Repeat Step 2 & 3 to numbers before the decimal.
5. The post-condition is the String representation of the difference between the two numbers.
6. **Printing in the Screen the Results**

Lastly, once we have all three String values (Larger Number, Sum, Difference) that we needed, we can then now print off these results to show the user. The three Strings will be our pre-condition for this. This very straightforward as we just have to make a String sentence and pass it in as an Argument for System.out.println(Argument). Post-Condition is the results showing in the terminal for the user to analyze.

**Correctness**

1. **Finding out which number is larger**

**Assertion R is a loop invariant:** in themethod compareNumbers

* i < num1.length() == num2.length()

**Base Property:** Before the first iteration of the loop

* i = 0; i < num1.length()

**Inductive Property:**

* Assume that the loop body is executed I > 0 times and that R is satisfied at the beginning of the I – th execution. At the end of the I – th execution, since i is increased by 1:
* If the if-condition was true, then num1[i]>num2[i], Boolean bool = true, and we break out of the loop because we have achieved the condition that we have been looking for.
* If the else if-condition was true, num1[i] == num2[i], and since the loop invariant held at the beginning of the loop body, we have 0 <= i <num1.length(), for num1.length() > 0.
* Otherwise, num1[i]<num2[i], Boolean bool = false, and we break out of the loop because we have achieved the condition that we have been looking for.

If there is a I + 1st execution of the loop body, then the loop test must pass before it, therefore R is satisfied (i < num1.length()).

**Termination:**

* Loop variant: f(num1.lenght(), i) = num1.length() -i
* f(num1.length(), i) is an integer valued function
* after every iteration, i increases by 1 and thus f(num1.length(), i) decreases by 1
* if f(num1.length(), i) <= 0 then i >=num1.length() and the loop terminates

1. **Calculating the Sum of Numbers**

**Assertion R is a loop invariant:** in themethod getSum

* i >0

**Base Property:** Before the first iteration of the loop

* i = num1.length() ; num1.length()>0

**Inductive Property:**

* Assume that the loop body is executed I > 0 times and that R is satisfied at the beginning of the I – th execution. At the end of the I – th execution, since i is decreased by 1:
* Case 1: carryOver == false
* Sub-Case 1: num1[i]+num2[i] > 9
* Then Boolean carryOver = true, and we concatenate the sum of num1[i]+num2[i] to sumStr (String representation of the sum)
* And since the loop invariant held at the beginning of the loop body, we have 0> i >=num1.length() for 0 < num1.length().
* Sub-Case 2: num1[i]+num2[i]<=9
* Then Boolean carryOver = false, and we concatenated the sum of num1[i]+num2[i] to sumStr (String representation of the sum)
* And since the loop invariant held at the beginning of the loop body, we have 0> i >=num1.length() for 0 < num1.length().
* Case 2: carryover == true
* Sub-Case 1: num1[i]+num2[i]+1 > 9
* Then Boolean carryOver = true, and we concatenate the sum of num1[i]+num2[i]+1 to sumStr (String representation of the sum)
* And since the loop invariant held at the beginning of the loop body, we have 0> i >=num1.length() for 0 < num1.length().
* Sub-Case 2: num1[i]+num2[i]+1<=9
* Then Boolean carryOver = false, and we concatenated the sum of num1[i]+num2[i]+1 to sumStr (String representation of the sum)
* And since the loop invariant held at the beginning of the loop body, we have 0> i >=num1.length() for 0 < num1.length().

If there is a I + 1st execution of the loop body, then the loop test must pass before it, therefore R is satisfied (i >0).

**Termination:**

* Loop variant: f(i, num1.lenght()) = I - num1.length()
* f(i, num1.lenght()) is an integer valued function
* after every iteration, i decreases by 1 and thus f(i , num1.lenght()) decreases by 1
* if f(i, num1.lenght()) <= 0 then i >=0 and the loop terminates

1. **Calculating the Difference of Numbers**

**Assertion R is a loop invariant:** in themethod getDifference

* i >0

**Base Property:** Before the first iteration of the loop

* i = num1.length(); num1.length()>0

**Inductive Property:**

* Assume that the loop body is executed I > 0 times and that R is satisfied at the beginning of the I – th execution. At the end of the I – th execution, since i is decreased by 1:
* Case 1: borrowOver == false
* Sub-Case 1: num1[i]<num2[i]
* Then Boolean borrowOver = true, and we concatenate the difference of (num1[i]+10)-num2[i] to differenceStr (String representation of the difference)
* And since the loop invariant held at the beginning of the loop body, we have 0> i >=num1.length() for 0 < num1.length().
* Sub-Case 2: num1[i]>num2[i]
* Then Boolean borrowOver = false, and we concatenate the difference of num1[i]-num2[i] to differenceStr (String representation of the difference)
* And since the loop invariant held at the beginning of the loop body, we have 0> i >=num1.length() for 0 < num1.length().
* Case 2: carryover == true
* Sub-Case 1: num1[i]-1<num2[i]
* Then Boolean borrowOver = true, and we concatenate the difference of (num1[i]+10)-num2[i] to differenceStr (String representation of the difference)
* And since the loop invariant held at the beginning of the loop body, we have 0> i >=num1.length() for 0 < num1.length()
* Sub-Case 2 num1[i]-1>num2[i]
* Then Boolean borrowOver = false, and we concatenate the difference of (num1[i]-1)-num2[i] to differenceStr (String representation of the difference)
* And since the loop invariant held at the beginning of the loop body, we have 0> i >=num1.length() for 0 < num1.length().

If there is a I + 1st execution of the loop body, then the loop test must pass before it, therefore R is satisfied (i >0).

**Termination:**

* Loop variant: f(i, num1.lenght()) = i - num1.length()
* f(i, num1.lenght()) is an integer valued function
* after every iteration, i decreases by 1 and thus f(i , num1.lenght()) decreases by 1

if f(i, num1.lenght()) <= 0 then i >=0 and the loop terminates

**Analysis of Program**

**Let’s analyze the unit cost of the program by calculating the unit cost for each subprogram and the add them all up at the end**

**Structure to Consider: S1 = compareNumbers, S2 = getSum, S3 = getDifference, S4 = getFinalNumber**

Let x be the length of the longer String between num1 and num2.

Program S1

While i < x do

….

i = i +1;

end while

Then worse case running time for S1 isTs1 = 2(x+1) + x = 3x +2 where T represents time

Program S2

while x>0 do

…

x = x-1;

end while

Then the worse case running time for S2 is TS2 = 3x +2.

Program S3

while x>0 do

…

x = x-1;

end while

Then the worse case running time for S3 is TS3 = 3x +2.

Program S4

Let n be the difference between the length of the smaller number from the larger number.

while i<n do

…

i = i+1;

end while

Then the worse case running time for S4 is TS4 = 2(n+1) + n = 3n+2.

Suppose the program is executed once. Then the cost for all evaluations of test TS1 + TS2 + TS3 + TS4 = 3x +2+3x +2+3x +2+ 3n +2 = 9x +3n +6.