Methodology

Section 1: getInput()

This section handles retrieving and parsing data from the user. Input is taken using a Scanner object and stored into 2 strings. From there the decimal is located in each string and the strings are separated into integer and decimal parts. Finally, the 2 integer parts are made to be the same length, same with the 2 decimal parts, this makes addition and subtraction much easier later on.

Section 2: findLarger()

To find the larger of the strings I first recombine them by their parts, then traverse both strings, when the larger hasn’t been found then each character is compared. After the comparison a larger number is either found or not. In the case that it is found the larger is stored, after the loop is finished and a larger is not found both numbers are the exact same, so one is picked to be the larger. After deciding a larger, the numbers are classified as bigger or smaller, this allows the difference to be calculated without finding the larger again.

Section 3: calculateSum()

The sum of 2 numbers is found by first combining each number from it’s parts. Then, starting at the back of both strings, each character is added together along with the carry and then adjusted if it is greater than 10, causing a carry to be set; this occurs in a loop going through each character. After the adjustment the number is concatenated onto the front a string that holds the sum. After the loop the total sum is split into it’s integer and decimal parts, this is done by knowing that the integer part of the sum will have the same length as the integer part of both numbers. Finally, if a carry is found a ‘1’ is added to the front of the integer.

Section 4: calculateDifference()

To find the difference first the bigger number is created from it’s parts, same with the smaller number. Next, a loop traverses the strings from back to front and subtracts the characters, along with the carry adjusting the number if the difference is negative and setting the carry. Next, the difference is added to the front of a string that holds the total difference. After this loop the total difference is split into it’s integer and decimal parts, this is done by knowing that the length of the integer part will be constant.

Section 5: printResults()

This section simply prints out the results, trivial considering that the results are calculated and have been stored prior to using this method.

Bonus Section: padString(s, length, isFront)

This section will loop until the length of the string, s, is equal to the target length, length. This is done with a simple loop, in the loop a ‘1’ is added to the front or back of the string based on the isFront parameter. Finally, the padded string is returned.

Proofs

Section 1: getInput()

{P}

Scanner keyb = new Scanner(System.in);

System.out.print("String1 = ");

num1 = keyb.nextLine();

System.out.print("String2 = ");

num2 = keyb.nextLine();

keyb.close();

{R1}

short idx1 = (short) num1.indexOf(".");

integer1 = num1.substring(0, idx1);

decimal1 = num1.substring(idx1 + 1, num1.length());

short idx2 = (short) num2.indexOf(".");

integer2 = num2.substring(0, idx2);

decimal2 = num2.substring(idx2 + 1, num2.length());

{R2}

if (integer1.length() < integer2.length()) {

integer1 = padString(integer1, (short) integer2.length(), true);

} else if (integer2.length() < integer1.length()) {

integer2 = padString(integer2, (short) integer1.length(), true);

}

if (decimal1.length() < decimal2.length()) {

decimal1 = padString(decimal1, (short) decimal2.length(), false);

} else if (decimal2.length() < decimal1.length()) {

decimal2 = padString(decimal2, (short) decimal1.length(), false);

}

{Q}

{P} … {Q}: Pre condition

1) num1, num2, integer1, decimal1, integer2, decimal2 have been declared but not initialized

Post condition

1. num1, num2 have the user’s values
2. integer1, integer2 have the content of respective numbers before the decimal
3. decimal1, decimal2 have the content of respective numbers after the decimal
4. integer1, integer2 have the same length
5. decimal1, decimal2 have the same length

{P} … {R1}: Pre condition

1. num1, num2 have been declared but not initialized

Post condition

1. num1, num2 have the user’s values

Proof

Trivial since only Java methods are used for asking for and receiving input then simply assigning it to a variable

{R1} … {R2}: Pre condition

1. num1, num2 have values from the user
2. integer1, integer2, decimal1, decimal2 have been declared but not initialized

Post condition

1. integer1, integer2 contain the integer parts of their respective strings
2. decimal1, decimal2 contain the decimal parts of their respective strings
3. num1, num2 have not been altered

Proof

Trivial, the strings are split from [0, decimal) and [decimal, stringLength) and stored in their respective variables.

{R2} … {Q}: Pre condition

1. integer1, integer2, decimal1, decimal2 have values

Post condition

1. integer1, integer2 have the same length
2. decimal1, decimal2 have the same length

Proof

Assuming the padString method is totally correct then the shorter of both string will be padded unless both are the same length in which case nothing will happen to the strings.

Section 2: findLarger()

{P}

boolean largerFound = false;

String s1 = integer1 + decimal1;

String s2 = integer2 + decimal2;

short idx = 0;

{R1}

while (idx < s1.length()) {

{I1}

if (!largerFound) {

short a = (short) Character.getNumericValue(s1.charAt(idx));

short b = (short) Character.getNumericValue(s2.charAt(idx));

if (a < b) {

larger = num2;

largerFound = true;

} else if (a > b) {

larger = num1;

largerFound = true;

}

}

{I2}

idx++;

}

{R2}

if (!largerFound) {

larger = num1;

}

{R3}

if (larger.equals(num1)) {

biggerInteger = integer1;

biggerDecimal = decimal1;

smallerInteger = integer2;

smallerDecimal = decimal2;

} else {

biggerInteger = integer2;

biggerDecimal = decimal2;

smallerInteger = integer1;

smallerDecimal = decimal1;

}

{Q}

{P} … {Q}: Pre condition

1. num1, num2, integer1, integer2, decimal1, decimal2 have values
2. larger, biggerInteger, biggerDecimal, smallerInteger, smallerDecimal have been declared but not initialized

Post condition

1. num1, num2 are not changed
2. larger has the value of the larger of num1 and num2
3. biggerInteger, smallerInteger have the value before the decimal of the respective number
4. biggerDecimal, smallerDecimal have the value after the decimal of the respective number

{P} … {R1}: Pre condition

1. integer1, integer2, decimal1, decimal2 have values

Post condition

1. integer1, integer2, decimal1, decimal2 have not been changed
2. largerFound is set to false, idx is set to 0

Proof

Trivial since values are just being set

{R1} … {R2}: Pre condition

1. idx is 0
2. largerFound is false
3. s1, s2 have values
4. larger is declared but not initialized

Post condition

1. idx is equal to the length of s1
2. largerFound is true if the numbers are not the same
3. large has a value if the numbers are not the same
4. s1, s2 have not been changed

Proof: Invariant

The invariant is idx < s1.length()

Base case: idx = 0 and s1.length is at least 2 (x.x) so 0 < 2 and the loop starts

Inductive step: idx = k for k > 0, when idx = k if k < s1.length() then the loop will start. If the loop has started then k must be less than s1 other wise the loop will not continue as the invariant is not satisfied.

Proof: Variant

The variant, f(idx, s1.length()) = s1.length() – idx. After every iteration idx increases by one making the variant decrease by 1. When the variant is less than 0 then idx is greater than s1.length(), this prevents the loop from running once more and ensures termination.

{I1} … {I2}: Pre condition

1. idx, largerFound have values
2. larger is declared but not initialized

Post condition

1. idx is not changed
2. largerFound is either true or false

Proof

If a larger has not been found yet then largerFound is false and the body of the IF is executed, in this case the short value of the character at the index of both strings is found, this is trivial using Java’s built in methods. Next the values are compared in the event that the values are not equal a larger is found and the largerFound variable is true.

{R2} … {R3}: Pre condition

1. largerFound has a value
2. larger has been declared

Post condition

1. larger has a value
2. largerFound has not changed value

Proof

In the case that a larger value has not been found, largerFound is still false. This means that both values are equivalent, the body of the if sets larger to be num1.

{R3} … {Q}: Pre condition

1. larger, num1, integer1, integer2, decimal1, decimal2 have values
2. biggerInteger, biggerDecimal, smallerInteger, smallerDecimal have been declared but not initialized

Post condition

1. larger, num1 have not been changed
2. biggerInteger, biggerDecimal, smallerInteger, smallerDecimal have values

Proof

Based on the value of num1 and larger the IF condition is evaluated, in both cases the values of biggerInteger, biggerDecimal, smallerInteger, smalledDecimal are set.

Section 2: calculateSum()

String s1 = integer1 + decimal1;

String s2 = integer2 + decimal2;

String totalSum = "";

short carry = 0;

short idx = (short) (s1.length() - 1);

while (idx >= 0) {

short a = (short) Character.getNumericValue(s1.charAt(idx));

short b = (short) Character.getNumericValue(s2.charAt(idx));

short sum = (short) (a + b + carry);

if (sum >= 10) {

sum = (short) (sum - 10);

carry = 1;

} else {

carry = 0;

}

totalSum = String.valueOf(sum) + totalSum;

idx--;

}

sumInteger = totalSum.substring(0, integer1.length());

sumDecimal = totalSum.substring(integer1.length(), totalSum.length());

if (carry != 0) {

sumInteger = "1" + sumInteger;

}