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.

Section 2: getProduct()

This section finds all intermediate products of the multiplication with nested loops that traverse the number strings, starting at the back, and multiplying each character and adding a carry. The carry is adjusted when the result is greater than 10 and the result is also adjusted. After this the result is added to the front of the current intermediate string. At the end of the inner loop any remaining carry is added to the front of the intermediate, then the carry is reset and the intermediate is pushed to the end of the intermediate queue. Then the intermediate is added to the current product to find the total product, after this the current intermediate is reset and filled with zeroes depending on the current iteration. Finally, the loop counter is decremented and the loop continues.

Section 3: printResults()

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

Section 4: printIntermediates()

This section prints out the intermediates by dequeuing them from the queue and printing until the queue is empty.

Helper Section 1: padString(s, length)

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 ‘0’ is added to the front of the string. Finally, the padded string is returned.

Helper Section 2: sumNumbers(s1, s2)

This section will loop through both string, starting at the end, and at each character and carry together to find the result. The result and carry are adjusted and the result is joined to the front of the sum string. When the loop has ended, any remaining carry is added to the front of sum and sum is returned.

Proofs

Section 1: getInput()

Line 48-62

Pre-Condition

1. num1, num2 have been declared but not initialized

Post-Condition

1. num1, num2 have the user’s values

Proof

Trivial, prebuilt java class (Scanner) takes input and assigns to a variable.

Section 2: getProduct()

Line 76-157

Pre-Condition

1. intermediateQueue, product have been declared
2. num1, num2 have values

Post-Condition

1. intermediateQueue contains all intermediates
2. product is the sum of intermediates
3. num1, num2 have not been changed

Proof

Proof by parts below

Line 76-90

Pre-Condition

1. intermediateQueue has been declared but not initialized
2. num2 has value

Post-Condition

1. intermediateQueue has been initialized
2. carry, zeroes are equal to 0
3. num2Index is num2 length – 1
4. intermediate is “”
5. num2 has not been changed

Proof

Trivial, variables are being created and initialized, there values are guaranteed by pre-condition.

Line 93-156

Pre-Condition

1. num2Index is num2 length – 1
2. intermediate is “”
3. carry, zeroes are 0
4. product has been declared
5. intermediateQueue is initialized
6. num1, num2 have values

Post-Condition

1. num2Index, num1Index are -1
2. carry is 0
3. zeroes = num2 length – 1
4. product has a value
5. intermediateQueue has all intermediates
6. num1, num2 have not changed

Proof

Upon entering this loop num1Index is set to num1 length minus 1 then another loop is entered, upon successful completion of the inner loop (proven later) num1Index will be -1. At that point if a carry exists it is added to the front of the current intermediate, and reset. The current intermediate is added to the end of the queue. Next if product is null then the current product is set to this intermediate else the current product is padded to the same length as current intermediate and the new product is set to the sum of the padded product and the intermediate. Finally, intermediate is reset and padded to the correct number of zeroes and num2Index gets decremented.

Proof: Invariant

The invariant is num2Index > -1

Base case: num2Index = num2 length – 1, this is greater than -1 because the num2 is a number with at least length 1

Inductive step: num2Index = k for k < num2length – 1, when num2Index = k if k > -1 then the loop will start. If the loop has started, then k must be greater than -1 otherwise the loop will not continue as the invariant is not satisfied.

Proof: Variant

The variant, f(num2Index, -1) = num2Index - -1 = num2Index + 1. After every iteration num2Index decreases by one making the variant decrease by 1. When the variant is less than or equal to 0 num2Index must be -1, this prevents the loop from running once more and ensures termination.

Line 99-119

Pre-Condition

1. num1Index is num1 length – 1
2. num1, num2 have values
3. carry has a value
4. intermediate is “”
5. num2Index has a value

Post-Condition

1. num1Index is -1
2. num1, num2 have not changed values
3. carry has a value
4. intermediate has a value
5. num2Index is unchanged

Proof

Upon entry, the character at the current index of both numbers is retrieved and multiplied together. Then, the carry is added, the new carry is set to the first digit of the result if the result is greater than 9, else it is 0. Result is adjusted to be a number between 0 and 9, this result is then attached to the front of the current intermediate. Finally, num1Index is decremented.

Proof: Invariant

The invariant is num1Index > -1

Base case: num1Index = num1 length – 1, this is greater than -1 because the num1 is a number with at least length 1

Inductive step: num1Index = k for k < num1length – 1, when num1Index = k if k > -1 then the loop will start. If the loop has started, then k must be greater than -1 otherwise the loop will not continue as the invariant is not satisfied.

Proof: Variant

The variant, f(num1Index, -1) = num1Index - -1 = num1Index + 1. After every iteration num1Index decreases by one making the variant decrease by 1. When the variant is less than or equal to 0 num1Index must be -1, this prevents the loop from running once more and ensures termination.

Section 3: printResults()

Line 169-173

Pre-Condition

1. product has a value

Post-Condition

1. product has not change value
2. product has been printed

Proof

Trivial, pre-condition guarantees that product is valid and Java methods are used to print.

Section 4: printIntermediates()

Line 185-201

Pre-Condition

1. intermediateQueue has been declared and has values

Post-Condition

1. intermediateQueue is empty
2. all elements from intermediateQueue have been printed

Proof

Trivial for the initial print and the index declaration.

Line 193-200

Pre-Condition

1. intermediateQueue has values
2. index is set

Post-Condition

1. intermediateQueue is empty
2. index is equal to the total number of elements in intermediateQueue
3. all elements of intermediate have been printed

Proof

Every time the loop executes a value will be taken from the front of the intermediate queue, then this value will be printed. Due to the queue invariant, the first value is the oldest and therefore the first calculated intermediate. Finally, the index is incremented.

Proof: Invariant

The invariant is intermediateQueue length doesn’t equal 0

Base case: intermediateQueue length is at least 1 from multiplying to single digit numbers, since this is the case the loop will start.

Inductive step: the loop has run k > 0 times, so intermediateQueue length is starting length – k, if this value doesn’t equal 0 the loop will start; if the loop has started then this difference is not 0 otherwise the loop will not continue as the invariant is not satisfied.

Proof: Variant

The variant, f(intermediateQueue length, 0) = intermediateQueue length + 0. After every iteration, an element is removed from intermediateQueue making the length decrease by one, this also means the variant decreases by 1. When the variant is 0 intermediateQueue length must be 0, this means the queue is empty and the loop will not run again ensuring termination.

Helper Section 1: padString(s, length)

Line 215-225

Pre-Condition

1. s is a string
2. length is a short

Post-Condition

1. s is unchanged
2. length is unchanged
3. a padded copy of s is returned

Proof

When the loop runs a single 0 is added to the front of the string, this will increase its length by 1. This happens until the length of s is equal to length.

Proof: Invariant

The invariant is s.length() < length

Base case: s.length() is a number >= 1 since length is greater than s.length() the loop will start

Inductive step: s.length() has increased, if s.length() < length then the loop will start. If the loop has started then s.length() must be less than length otherwise the loop will not continue as the invariant is not satisfied.

Proof: Variant

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

Helper Section 2: sumNumbers(s1, s2)

Line 238-280

Pre-Condition

1. s1, s2 are strings representing positive integers
2. s1, s2 have the same length

Post-Condition

1. s1, s2 have not changed values
2. the sum of s1 and s2 is returned

Proof

Trivial before the loop as values are being set only.

Line 249-269

Pre-Condition

1. sum, index, carry have values

Post-condition

1. index is -1
2. carry, sum have values

Proof

When the loop starts the value of each string at the index is retrieved and summed along with carry. Carry is set to the first digit of result when result is greater than 10 else it is 0. Result is adjusted to be between 0 and 9 then added to the front of sum, finally index is decremented.

Proof: Invariant

The invariant is index > -1

Base case: index is s1.length() – 1, this is greater than -1 so the loop starts.

Inductive step: index has decreased by one, if index > -1 then the loop will start. If the loop has started, then index must be greater than -1 other wise the loop will not continue as the invariant is not satisfied.

Proof: Variant

The variant, f(index, -1) = index - -1 = index + 1. After every iteration index decreases by one so the variant will decrease by 1. When the variant is less than or equal to 0 index is less than zero, this will prevent the loop from running again and ensuring termination.

Line 272-278

Pre-Condition

1. carry, sum have values

Post-Condition

1. carry has not changed
2. sum has a value
3. sum has been returned

Proof

If carry is not 0 then it will be added to the front of sum and sum is then returned.

Analysis

Section 1: getInput() = 9 units

Section 2: getProduct() = 18n2 + 23n + 9 units

Section 3: printResults() = 2 units

Section 4: printIntermediates() = 6n + 4 units

Helper Section 1: padString(s, length) = 4n + 2 units

Helper Section 2: sumNumbers(s1, s2) = 18n + 10 units

Total = (9) + (18n2 + 23n + 9) + (2) + (6n+4) + (4n+2) + (18n+10)

= 18n2 + 51n + 36

This program is O(n2) for C = 105 and N0 = 1 because 18n2 + 51n + 36 is always less than or equal to 105n2

Experimental Analysis

Case 1) Both length 10

String1 = 4845246747

String2 = 3929126387

Time = 4ms

Case 2) Both length 100

String1 = 2552317406837442909173953092142885306732505758398659272628611563146702604575249915845428227618660542

String2 = 3298356118104555501745020061857795176731945841371869789058588152952805732224136969373612543683309640

Time = 27ms

Case 3) Both length 1000

String1 = 2241991281473736176021614853477782963026526942588699863514706748321410085064496471018551579617293786019413430507178833726708526817059363628144333639239809031134289952518331840586377649675698409282160646616495282744820954391604242610260676603100328853712933393010862545831759142110361864065073450672002307375400384344651285864066933030664271516113402020510038210848148147948572620485155227809292846946210591456262717596712507085978166467148299821018524575278726165235742953876887776904735447923704549395011960663315339739442851523322293806675448486835128473214134029347631857497977815319870321204280404755040392209424417274380087313040851389143257904956292600953053003210972472922828649345513442466907230030758141062213447808224056099229051706492615806760689812726307337075832563676302248736944947893466810200988658067501418699790364945928224226192683341330285052916558769759056139267091767819624190282036787353807839131892239914188468612666166636344881089248461400187460467780605852089040603151983768

String2 = 9699633215752301486072509137854588598275135736557654851974101401913677996822179094660436545681634310329160518755480185557440831488201963462907904899904935999176040776354609146151154311703130426275531667214507924285270377692105405287351151009032946563641199256697283890871715279368846431770338453107606491314481507926543093639882057721327484268378734534257580043588282296246171669898375031823888355814973785334793150576275993669761907090471664937693151538769792883675779647024464548980070988189507113953308693041568356059947341418829230683394591585507857905080859735868213257846671211967069082124584501265873607656550043966085332869321399171328355239650002177189301609607186007280555756744074619218119752958153059850278964119918261620671874524179511386292336580301660407888680731926079151678268055179072442964279034321185266599154837948829195194178024171601215641987990560713914812624060686418413018627159818304452751480475675787375200676812897185266562875331567879422854362801782040884391715184545506

Time = 3206ms

Case 4) 100 length and 10 length

String1 = 1269130447317838810059888754857927441204188969787970272732005641249787218198489064292920436348000889

String2 = 4082266538

Time = 5ms

Case 5) 10 length and 100 length

String1 = 4082266538

String2 = 1269130447317838810059888754857927441204188969787970272732005641249787218198489064292920436348000889

Time = 16ms

Observations

For equal length strings run time increases roughly quadratically however the order of the strings matters for non-equal sized strings. Generally, the shorter string should be the second string for a quicker execution.