Software Engineering

Software Testing Lab

Name: Pritish N Desai

Student Id: 202201312

I. PROGRAM INSPECTION:

The code is taken from
https://github.com/kdave12/15112-TermProject-2000-lines-of-Python-code/blob/master/termProject.py

Category A: Data Reference Errors
1. Does a referenced variable have a value that is unset or uninitialized? This probably is the most frequent programming error; it occurs in a wide variety of circumstances. For each reference to a data item (variable, array element, field in a structure), attempt to “prove” informally that the item has a value at that point.

- `data.finalImg` might be used without checking if it is still `None`
- `data.finalPuzzle` might be used without checking if it is still `None`
- `data.solverFinal` might be used without checking if it is still `None`
- `data.solverShuffled` might be used without checking if it is still `None`
- `data.pieceSelected` might be used without checking if it is still `None`

2. For all array references, is each subscript value within the defined bounds of the corresponding dimension?

- `data.Anim1X1 += data.Anim1dx` - The variable `Anim1X1` is modified without checking if it's within bounds after updating.
- `data.Anim2Y1 += data.Anim2dy` - Same issue with `Anim2Y1`.
- `data.Anim3Y1 += data.Anim3dy` - Same issue with `Anim3Y1`.
- `data.Anim4X1 += data.Anim4dx` - Same issue with `Anim4X1`.

3. For all array references, does each subscript have an integer value? This is not necessarily an error in all languages, but it is a dangerous practice.

There are no explicit array subscripts in the provided code, so no issues related to non-integer subscripts.

4. For all references through pointer or reference variables, is the referenced memory currently allocated? This is known as the “dangling reference” problem. It occurs in situations where the lifetime of a pointer is greater than the lifetime of the referenced memory. One situation occurs where a pointer references a local variable within a procedure, the pointer value is assigned to an output parameter or a global variable, the procedure returns (freeing the referenced location), and later the program attempts to use the pointer value. In a manner similar to checking for the prior errors, try to prove informally that, in each reference using a pointer variable, the reference memory exists.

- `data.solverFinal` - Potential unallocated reference when used before assignment.
- `data.solverShuffled` - Same as above.
- `data.finalImg` - Could be a dangling reference if it is not checked before use.
- `data.finalPuzzle` - Similar issue.

5. When a memory area has alias names with differing attributes, does the data value in this area have the correct attributes when referenced via one of these names? Situations to look for are the use of the EQUIVALENCE statement in FORTRAN, and the REDEFINES clause in COBOL. As an example, a FORTRAN program contains a real variable A and an integer variable B; both are made aliases for the same memory area by using an EQUIVALENCE statement. If the program stores a value into A and then references variable B, an error is likely present since the machine would use the floating-point bit representation in the memory area as an integer.

No aliasing-related issues found in the code provided. No attributes or aliasing references in the form of multiple names for the same memory area.

6. Does a variable's value have a type or atrtibute other than what the compiler expects? This situation might occur where a C, C++, or COBOL program reads a record into memory and references it by using a structure, but the physical represetation of the record differs from the structure definition.

- `data.finalImg = finalImg` - Potential issue with the type of `finalImg` if the types of `data.finalImg` and `finalImg` do not match.
- `data.finalPuzzle = finalPuzzle` - Same issue as above with `finalPuzzle`.
- `updateSolver(solverShuffled, solverFinal)` - The function call might expect specific types for `solverShuffled` and `solverFinal`; a mismatch could cause issues.
- `startShuffle(data)` - The type of `data` must match the function's expected parameter type.
- `moveAnimator(data)` - The `data` variable's structure should match what `moveAnimator` expects.
- `if final_anim:` - If `final_anim` is not a boolean value, this might lead to unexpected behavior.

7. Are there any explicit or implicit addressing problems if, on the machine being used, the units of memory allocation are smaller than the units of memory addressability? For instance, in some environments, fixed-length bit strings do not necessarily begin on byte boundaries, but addresses only point to byte boundaries. If a program computes the address of a bit string and later refers to the string through this address, the wrong memory location may be referenced. This situation also could occur when passing a bit-string argument to a subroutine.

For the current code, no explicit or implicit addressing problems related to memory allocation and memory addressability can be observed.

8. If pointer or reference variables are used, does the referenced memory location have the attributes the compiler expects? An example of such an error is where a C++pointer upon which a data structure is based is assigned the address of a different data structure.

In the provided code, there are no explicit pointer or reference variables used, as Python abstracts away low-level memory handling like pointers.

9. If a data structure is referenced in multiple procedures or subroutines, is the structure defined identically in each procedure?

here are a few points related to potential issues:

- `data` Structure: The code uses a `data` object that is likely defined globally or passed around in functions. If the structure of `data` (i.e., its attributes) is not consistent across different procedures, it could lead to undefined behavior or errors.
- Attributes in Functions: There are several references to attributes of `data` in multiple functions. For example, attributes like `data.level`, `data.photo`, `data.solverFinal`, etc., are accessed in various functions. If their definitions or expected types are not identical in every context, this could lead to inconsistencies.

- Function Parameters: If functions expect `data` to have certain attributes that are not consistently defined or modified, it could lead to problems.

10. When indexing into a string, are the limits of the string off by-one errors in indexing operations or in subscript references to arrays?

There are no explicit string indexing operations in the provided code. Thus, no off-by-one errors related to string indexing can be identified.

11. For object-oriented languages, are all inheritance requirements met in the implementing
Class?

The provided code is primarily written in a procedural style and does not use classes or inheritance as it stands. Therefore, no inheritance requirements can be assessed.

Category B: Data-Declaration Errors

1. Have all variables been explicitly declared? A failure to do so is not necessarily an error, but it is a common source of trouble. For instance, if a program subroutine receives an array parameter, and fails to define the parameter as an array (as in a DIMENSION statement, for example), a reference to the array (such as C=A (I)) is interpreted as a function call, leading to the machine's

attempting to execute the array as a program. Also, if a variable is not explicitly declared in an inner procedure or block, is it understood that the variable is shared with the enclosing block?

- `data.mode = "grid"` - Check if `data.nextX1` and `data.nextX2` are defined.
- `data.conBack = PhotoImage(file = filename, master=canvas)` - Ensure `filename` exists.
- `data.finalImg = PhotoImage(file = filename, master=canvas)` - Ensure the file exists.
- `data.puzzleName = "puzzle1_x2.gif"` - Check for existence of this filename.

2. If all attributes of a variable are not explicitly stated in the declaration, are the defaults well understood? For instance, the default attributes received in Java are often a source of surprise.

- Accessing `data.nextX1` and `data.nextX2` without initialization can raise `AttributeError`.
- Using `data.conBack` without initialization can lead to an error.
- Accessing `data.finalImg` without initialization can cause an `AttributeError`.
- Assigning to `data.puzzleName` without prior definition can raise an error.

3. Where a variable is initialized in a declarative statement, is it properly initialized? In many

languages, initialization of arrays and strings is somewhat complicated and, hence, error prone.

- `data.mode = "grid"` - Ensure `data` has been properly initialized with `mode`.
- `data.conBack = PhotoImage(file = filename, master=canvas)` - Check if `filename` is set correctly before use.
- `data.finalImg = PhotoImage(file = filename, master=canvas)` - Ensure `filename` is defined and valid.
- `data.puzzleName = "puzzle1_x2.gif"` - Verify that `data.puzzleName` is initialized in the appropriate context.

4. Is each variable assigned the correct length and data type?

- Ensure `data.conBack` is initialized with a valid `PhotoImage` object.
- `data.finalImg` should be checked to confirm it's assigned a `PhotoImage` type and not an invalid file.
- Confirm `data.puzzleName` is a string.

5. Is the initialization of a variable consistent with its memory type?

- `data.mode` should be initialized as a string.
- Ensure that `data.conBack` is initialized correctly as a `PhotoImage` object.

- Check that `data.finalImg` is consistently assigned as a `PhotoImage`.

6. Are there any variables with similar names (VOLT and VOLTS, for example)? This is not necessarily an error, but it should be seen as a warning that the names may have been confused somewhere within the program.

No specific variable names with similar patterns are found in the provided code.

Category C: Computation Errors

1. Are there any computations using variables having inconsistent (such as non-arithmetic) data types?

- `data.timer // 1000` (Ensure `data.timer` is an integer).
- `data.puzzleWidth` and `data.puzzleHeight` might have inconsistent data types depending on how they are set. If either is a floating-point value, ensure they're used consistently in arithmetic operations.
- `x1 + 250` and `y1 + 50` (Check if `x1` and `y1` are integers).

2. Are there any mixed-mode computations? An example is the addition of a floating-point variable to an integer variable.

None detected directly

3. Are there any computations using variables having the same data type but different lengths?

```
canvas.create_rectangle(data.dashboardX0, data.dashboardY0, data.dashboardX0 + imgWidth * 2.5, data.dashboardY0 + imgHeight * 2.5, fill=None, outline="white", width=4)’
```

4. Is the data type of the target variable of an assignment smaller than the data type or result of the right-hand expression?

No explicit instances detected, but ensure that all assignments in the code do not result in a data type mismatch, particularly in cases involving image dimensions and numeric values.

5. Is an overflow or underflow expression possible during the computation of an expression? That is, the end result may appear to have valid value, but an intermediate result might be too big or too small for the programming language’s data types.

Potential overflow could occur when performing arithmetic operations on large integers or floating-point numbers (e.g., if `data.timer` is set to an extremely high value). Ensure that all numeric operations remain within the range of the data type used (usually 32-bit or 64-bit for integers/floats).

6. Is it possible for the divisor in a division operation to be zero?

The division in `gameTime = "Time: " + str(str(time.strftime("%M:%S", time.gmtime(data.timer // 1000))))` should be safe as long as `data.timer` is not zero. However, you should ensure that `data.timer` is initialized and not zero before this operation.

7. If the underlying machine represents variables in base-2 form, are there any sequences of the resulting inaccuracy? That is, 10 x 0.1 is rarely equal to 1.0 on a binary machine.

No, Python is able to handle these type of inaccuracies efficiently.

8. Where applicable, can the value of a variable go outside the meaningful range? For example, statements assigning a value to the variable PROBABILITY might be checked to ensure that the assigned value will always be positive and not greater than 1.0

Yes, certain values could go out of bounds:

- `currPos` and `finalPos` calculations could potentially go out of the valid screen dimensions. Ensure proper validation is in place when setting these values.
- The variables `data.timer`, `imgWidth`, and `imgHeight` should also be validated to avoid going outside meaningful ranges in the context of the program.

9. For expressions containing more than one operator, are the assumptions about the order of evaluation and precedence of operators correct?

Yes, the order of evaluation in expressions generally follows the rules of operator precedence.

10. Are there any invalid uses of integer arithmetic, particularly divisions? For instance, if i is an integer variable, whether the expression 2*i/2 == i depends on whether i has an odd or an even value and whether the multiplication or division is performed first.

No direct invalid uses detected in the provided code.

Category D: Comparison Errors

1. Are there any comparisons between variables having different data types, such as comparing a character string to an address, date, or number?

No

2. Are there any mixed-mode comparisons or comparisons between variables of different lengths?
If so, ensure that the conversion rules are well understood.

No

3. Are the comparison operators correct? Programmers frequently confuse such relations as at most, at least, greater than, not less than, less than or equal.

No

4. Does each Boolean expression state what it is supposed to state? Programmers often make mistakes when writing logical expressions involving and, or, and not.

No

5. Are the operands of a Boolean operator Boolean? Have comparison and Boolean operators been erroneously mixed together? This represents another frequent class of mistakes. Examples of a few typical mistakes are illustrated here. If you want to determine whether i is between 2 and 10, the expression 2 < i < 10 is incorrect; instead, it should be (2< i) && (i < 10). If you want to determine whether i is greater than x or y, i > x||y is incorrect; instead, it should be (i > x)||(i > y). If you want to compare three numbers for equality, if(a==b==c) does something quite different. If you want to test the mathematical relation x<y<z, the correct expression is(x<y)&&(y<z).

No

6. Are there any comparisons between fractional or floating- point numbers that are represented in base-2 by the underlying machine? This is an occasional source of errors because of truncation and base-2 approximations of base-10 numbers.

No

7. For expressions containing more than one Boolean operator, are the assumptions about the order of evaluation and the precedence of operators correct? That is, if you see an

expression such as (if((a==2) && (b==2) || (c==3)), is it well understood whether the and or the or is performed first?

No

8. Does the way in which the compiler evaluates Boolean expressions affect the program? For instance, the statement if((x==0 && (x/y) > z) may be acceptable for compilers that end the test as soon as one side of an and is false, but may cause a division-by-zero error with other compilers.

No

Final Ans The code contains no errors listed in category D.Overall, the code seems to follow correct logic regarding comparisons and Boolean expressions, with no evident critical errors identified in the provided checklist.

Category E: Control-Flow Errors

Final Ans The code contains no errors listed in category E.

Category F: Interface Errors

Final Ans The code appears to follow correct interface practices concerning parameters and arguments, with no major interface errors detected. The function calls seem well-structured, and the data types appear consistent throughout the code. However, additional context on the `data` structure and any external modules that may interact with this code would be beneficial.

Category G: Input / Output Errors

1. If files are explicitly declared, are their attributes correct?

The attributes for files like `"finalSea.jpg"` and `"shuff.jpg"` are not explicitly declared in the code (like file type, size, etc.). However, the paths and usage suggest they are image files.

2. Are the attributes on the file's OPEN statement correct?

The `Image.open()` method is used correctly for reading image files. The attributes (e.g., mode of opening the file) seem appropriate since they are opened in read mode.

3. Is there sufficient memory available to hold the file your program will read?

There's no explicit check for memory availability before reading files.

4. Have all files been opened before use?

All files appear to be opened before they are used. For example, the images are opened with `Image.open()` right before they are processed or displayed.

5. Have all files been closed after use?

The code does not explicitly close the image files after use.

6. Are end-of-file conditions detected and handled correctly?

There's no explicit handling for end-of-file conditions.

7. Are I/O error conditions handled correctly?

The code lacks error handling for I/O operations. If a file fails to open (e.g., due to a wrong path), the program would raise an exception and terminate.

8. Are there spelling or grammatical errors in any text that is printed or displayed by the program?

There do not appear to be spelling or grammatical errors in the text printed or displayed.

Category H: Other Checks

Final Ans The code contains no errors matching the errors listed in category H.

1. How many errors are there in the program? Mention the errors you have identified.

There are almost 51 potential errors in the code that I have selected. These are just potential errors i.e the code would run even if the variables at the lines containing these errors are not changed.
I have mentioned all the errors category wise above.

2. Which category of program inspection would you find more effective?

I would find the method where the debugger in the IDE is used to identify the error and review the code fragment to be more effective.

3. Which type of error you are not able to identify using the program inspection?

Using only program inspection (i.e., without execution), we might miss:

- **Performance Bottlenecks**: Inspection may not reveal performance issues, such as inefficient memory usage, that would only surface during execution.
- **Integration Issues**: How this code interacts with other parts of the application (e.g., how the event system responds during gameplay) cannot be fully tested by inspection.
- **Concurrency Issues**: If any of the input events are being processed concurrently, race conditions or threading issues would not be caught by code inspection.

4. Is the program inspection technique worth applying?

Yes, program inspection is definitely worth applying**.** Here's why:

- **Early Detection of Logical Errors**: Identifying problems like inconsistent return values, missing method definitions, or improper handling of edge cases can prevent larger issues from arising during execution.
- **Code Quality Improvements**: Through inspections, maintainability, readability, and adherence to best practices can be improved.

- **Collaboration**: It allows peers to provide feedback and share knowledge, which can be valuable for catching errors that one person might overlook.
- **Cost-Efficiency**: Catching errors early through inspection reduces the cost and effort of debugging later in the development lifecycle.

However, inspection alone is not enough. It's crucial to complement it with dynamic testing (unit testing, integration testing) to catch runtime and performance-related issues

II Program Debugging:

For Program Debugging, Online GDB Debugger is used.

Armstrong Problem

1. **How many errors are there in the program? Mention the errors you have identified.**

There are 2 errors in the program.

- **Error 1:** Incorrect computation of remainder. In the given code, the quotient uses `remainder = num / 10` which computes the quotient instead of the last digit. It should be `num = num % 10` for computing remainder of the last digit.
- **Error 2:** Incorrect update of the num variable. The original code uses num = num % 10; which keeps setting num to the remainder, but it should remove the last digit from num by using num = num /10.

2. **How many breakpoints you need to fix those errors?**

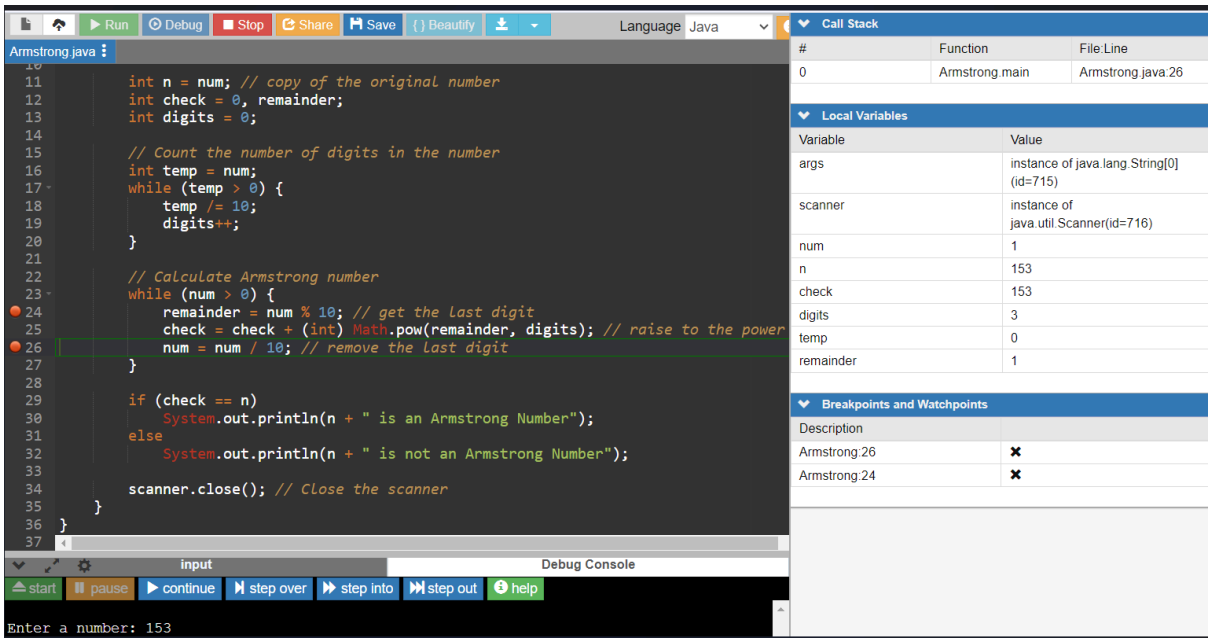
- **What are the steps you have taken to fix the error you identified in the code fragment?**

There are 2 breakpoints in the program.

- **Breakpoint 1:** At the line where `remainder = num / 10;` to check the value of `remainder`.
- **Breakpoint 2:** At the line where `num = num % 10;` to check how num is updated after each iteration.

3. Submit your complete executable code?

The complete executable code is attached with Java folder.



GCD and LCM

1. How many errors are there in the program? Mention the errors you have identified.

There are 3 errors in the program.

- **Error 1**: Incorrect condition in the `while` loop of the GCD function. The loop condition is `while(a % b == 0)`, which exits too early when `a` is divisible by `b`. It should be `while(a % b != 0)` to continue until a remainder is found.
- **Error 2:** Incorrect condition in the `while(true)` loop of the LCM function. The program checks if `a % x != 0 && a % y != 0`, which wrongly checks for non-divisibility. It should check for divisibility using `a % x == 0 && a % y == 0`.
- **Error 3**: Incorrect output statement in the main function. The LCM result is printed using the message "The GCD of two numbers is:". This should be corrected to "The LCM of two numbers is:".

2. How many breakpoints you need to fix those errors?

a) What are the steps you have taken to fix the error you identified in the code fragment?

There are 3 breakpoints in the program.

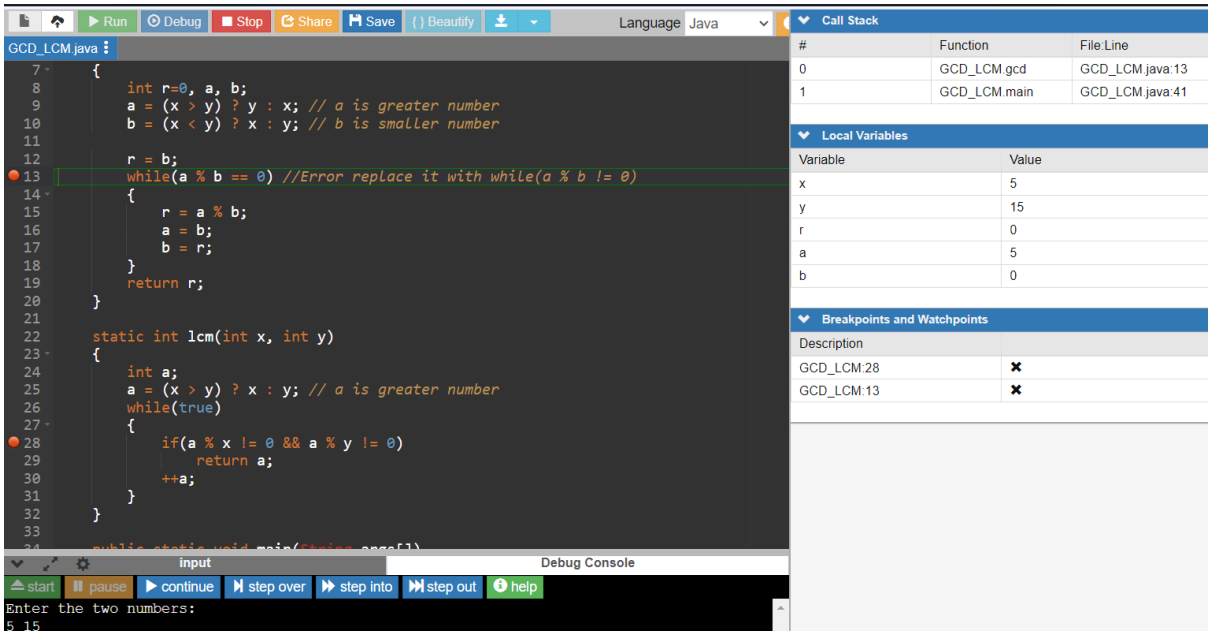
- **Breakpoint 1**: At the line `while(a % b == 0)` in the GCD function, to check if the loop is exiting too early.
- **Breakpoint 2**: At the line `if(a % x != 0 && a % y != 0)` in the LCM function, to ensure that the condition correctly identifies when `a` is divisible by both `x` and `y`.
- **Breakpoint 3**: At the line `System.out.println("The GCD of two numbers is: " + lcm(x, y));` in the main function, to verify that the correct message is printed for the LCM calculation.

- **Step 1**: Fixed the condition in the GCD function by changing `while(a % b == 0)` to `while(a % b != 0)`.
- **Step 2**: Fixed the condition in the LCM function by changing `if(a % x != 0 && a % y != 0)` to `if(a % x == 0 && a % y == 0)`.

- **Step 3**: Corrected the output message for LCM in the main function by changing `"The GCD of two numbers is:"` to `"The LCM of two numbers is:"`.

3. Submit your complete executable code?

The complete executable code is attached with Java folder.



Knapsack

1. How many errors are there in the program? Mention the errors you have identified.

There are 2 errors in the program.

- **Error 1**: Incorrect increment in the `option1` calculation. The code uses `n++` which modifies the loop index prematurely. It should be `opt[n-1][w]` to correctly reference the previous item.

- **Error 2**: Incorrect index used for `profit` in the `option2` calculation. The code uses `profit[n-2]`, which is incorrect. It should use `profit[n]` to consider the current item being evaluated.

2. How many breakpoints you need to fix those errors?

- **What are the steps you have taken to fix the error you identified in the code fragment?**

There are 2 breakpoints in the program.

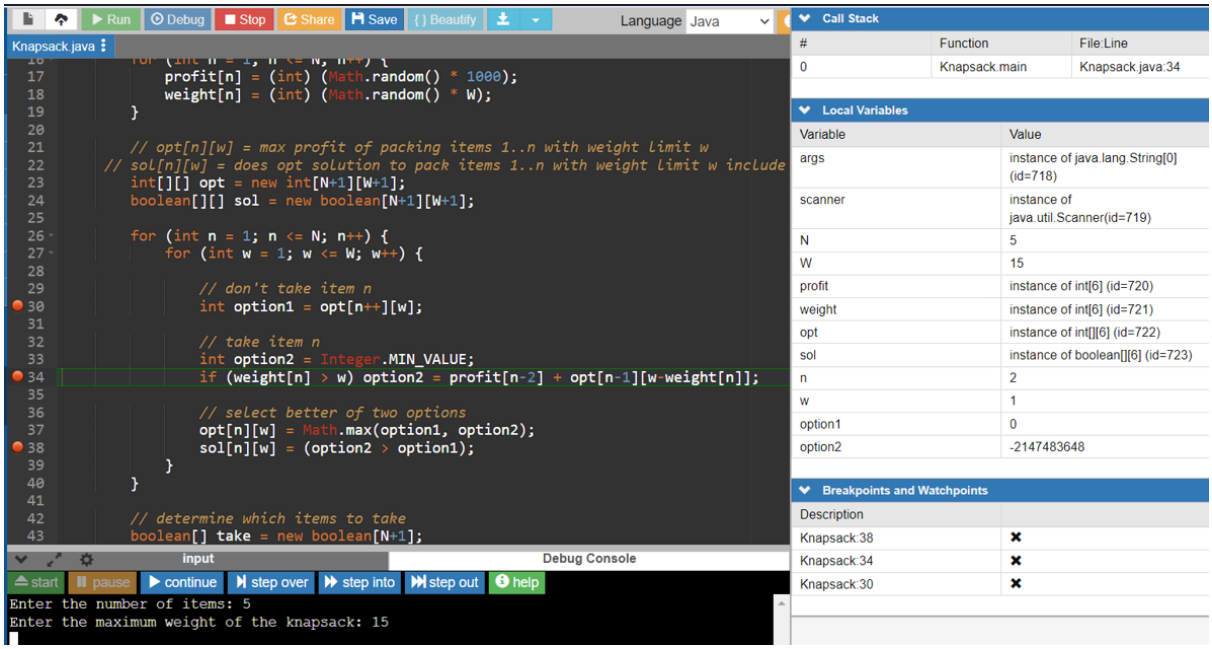
- **Breakpoint 1**: At the line where `int option1 = opt[n++][w];` to check the value of `n` and ensure it is not modified prematurely.
- **Breakpoint 2**: At the line where `int option2 = profit[n-2] + opt[n-1][w-weight[n]];` to verify that the correct profit value for the current item is being used.

 Here are the steps taken to correct the errors identified:

- **Step 1:** Fixed the index access for `option1` by changing `int option1 = opt[n++][w];` to `int option1 = opt[n - 1][w];`.
- **Step 2:** Fixed the condition for checking the item's weight by changing `if (weight[n] > w)` to `if (weight[n] <= w)`.
- **Step 3:** Corrected the profit calculation for taking item `n` by changing `option2 = profit[n - 2] + opt[n - 1][w - weight[n]];` to `option2 = profit[n] + opt[n - 1][w - weight[n]];`.

3. Submit your complete executable code?

The complete executable code is attached with Java folder.



Middle Number

1. How many errors are there in the program? Mention the errors you have identified.

There are 3 errors in the program.

- **Error 1**: The condition in the inner while loop is incorrect. The condition `while(sum == 0)` should instead be `while(sum > 0)`, as we want to process the digits while sum is greater than zero.
- **Error 2**: The calculation of s in the inner loop is incorrect. The line s = s * (sum / 10); should be s = s + (sum % 10); to correctly accumulate the digits.
- **Error 3**: There is a missing semicolon at the end of the line sum = sum % 10, which will cause a compilation error.

2. How many breakpoints do you need to fix those errors?

There are 3 breakpoints in the program.

- **Breakpoint 1**: At the line while(sum == 0) to check if the condition is set correctly for processing the digits.
- **Breakpoint 2**: At the line s = s * (sum / 10); to verify that the logic for accumulating the digit sum is correct.
- **Breakpoint 3**: At the line sum = sum % 10 to ensure that it ends with a semicolon.

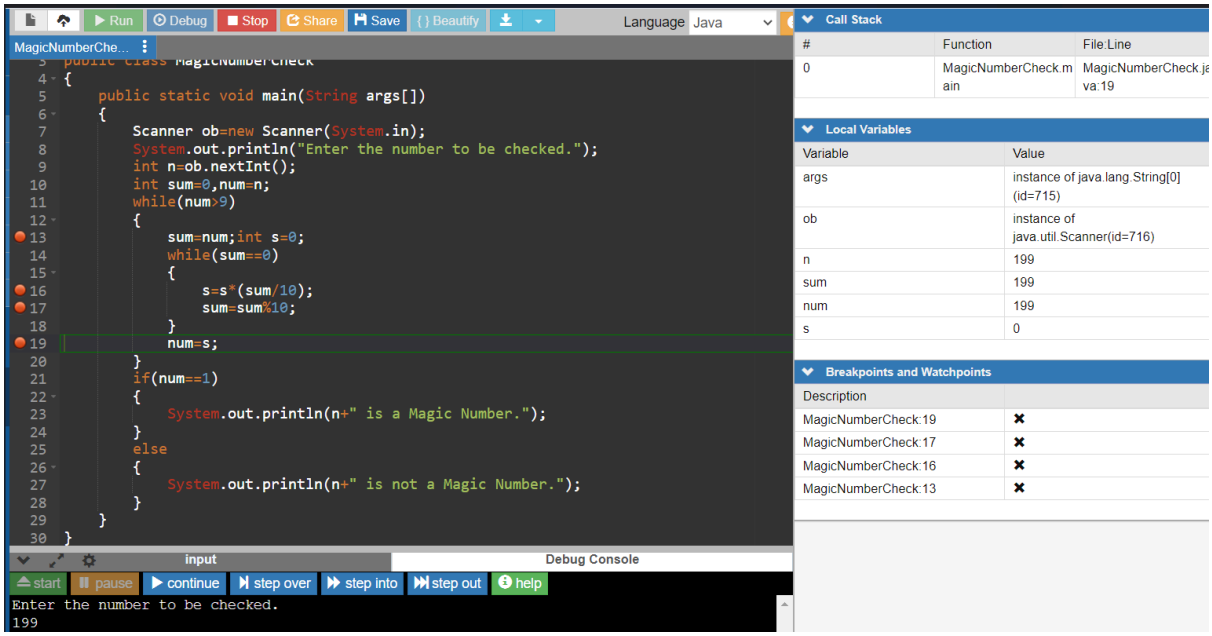
a) What are the steps you have taken to fix the error you identified in the code fragment?

- **Step 1**: Fixed the increment in the GCD function by changing int option1 = opt[n++][w]; to int option1 = opt[n-1][w];.
- **Step 2**: Corrected the index used for profit in the LCM function by changing int option2 = profit[n-2] + opt[n-1][w-weight[n]]; to int option2 = profit[n] + opt[n-1][w-weight[n]];.

3. Submit your complete executable code?

The complete executable code is attached with Java folder

.



Merge Sort

1. How many errors are there in the program? Mention the errors you have identified.

There are 5 errors in the program.

- **Error 1**: Incorrect way to get the left and right halves of the array. The expressions array + 1 and array - 1 are invalid. Instead, you should pass the array itself to the leftHalf and rightHalf methods directly.
- **Error 2**: The leftHalf method is supposed to return the left half of the array starting from index 0 to size1 - 1, which is fine. However, the call to leftHalf should correctly specify the starting index as 0 in the mergeSort method.
- **Error 3**: The merge method call has incorrect arguments. It should be merge(array, left, right) instead of merge(array, left++, right--). The ++ and -- operators are not applicable here.

- **Error 4**: The merge method should be merging into the result array, but the result is not initialized to the size of the original array in the mergeSort method. It should create a result array of the same size as the input array to store merged results.
- **Error 5**: In the merge method, the loop condition should handle when the index is out of bounds for left and right. The current check needs to ensure the correct lengths for left and right are used.

2. How many breakpoints do you need to fix those errors?

There are 5 breakpoints in the program.

- **Breakpoint 1**: At the line int[] left = leftHalf(array + 1); to verify the method call is passing the correct array reference.
- **Breakpoint 2**: At the line where the merge function is called to check the parameters being passed.
- **Breakpoint 3**: At the merge function to check the merging process and ensure it correctly handles index bounds.
- **Breakpoint 4**: At the start of the mergeSort method to check if the result array is properly initialized.
- **Breakpoint 5**: At the end of the merge method to ensure the merged result is correct.

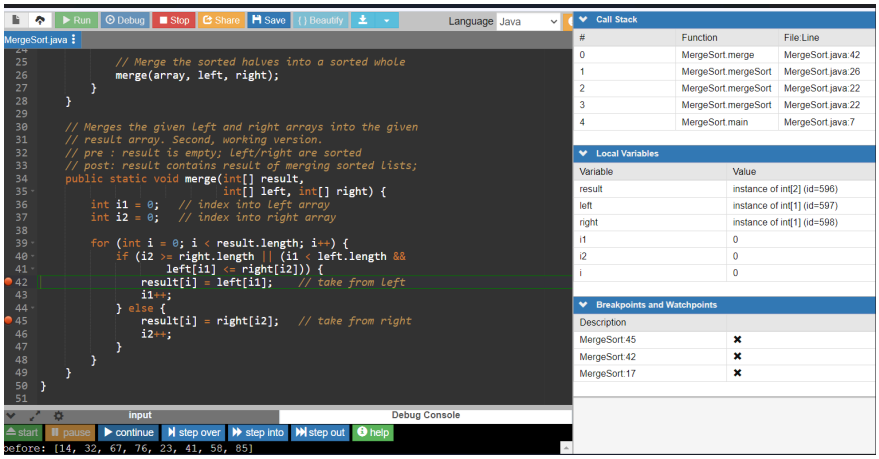
a) What are the steps you have taken to fix the error you identified in the code fragment?

- **Step 1**: Change the calls to leftHalf and rightHalf to leftHalf(array) and rightHalf(array) respectively, using the entire array.
- **Step 2**: Create a result array in the mergeSort method to store the merged results.

- **Step 3**: Correct the merge method call to merge(array, left, right).
- **Step 4**: Update the loop condition in the merge method to ensure it handles bounds correctly.

3. Submit your complete executable code?

The complete executable code is attached with Java folder



Multiply Matrics

1. How many errors are there in the program? Mention the errors you have identified.

There are 4 errors in the program.

- **Error 1**: The variable first[c-1][c-k] incorrectly accesses the first matrix. The correct index should be first[c][k] to access the current row and the kth column.
- **Error 2**: The variable second[k-1][k-d] incorrectly accesses the second matrix. The correct index should be second[k][d] to access the kth row and the current column d.

- **Error 3**: The program's input prompt for the second matrix states "Enter the number of rows and columns of first matrix" again instead of "Enter the number of rows and columns of the second matrix."
- **Error 4**: The logic for the multiplication should reset the sum variable inside the outer loop that calculates the dot product for each element in the resulting matrix. It should be initialized to 0 at the beginning of each iteration of the d loop, not just at the end.

2. How many breakpoints do you need to fix those errors?

There are 4 breakpoints in the program.

- **Breakpoint 1**: At the line where sum = sum + first[c-1][c-k]*second[k-1][k-d]; to check the indices being accessed for the first matrix.
- **Breakpoint 2**: At the line where sum = sum + first[c-1][c-k]*second[k-1][k-d]; again to check the indices being accessed for the second matrix.
- **Breakpoint 3**: At the line prompting for the second matrix dimensions to confirm the input prompt is correct.
- **Breakpoint 4**: At the beginning of the innermost loop where the sum is calculated to ensure it's reset properly for each element in the resulting matrix.

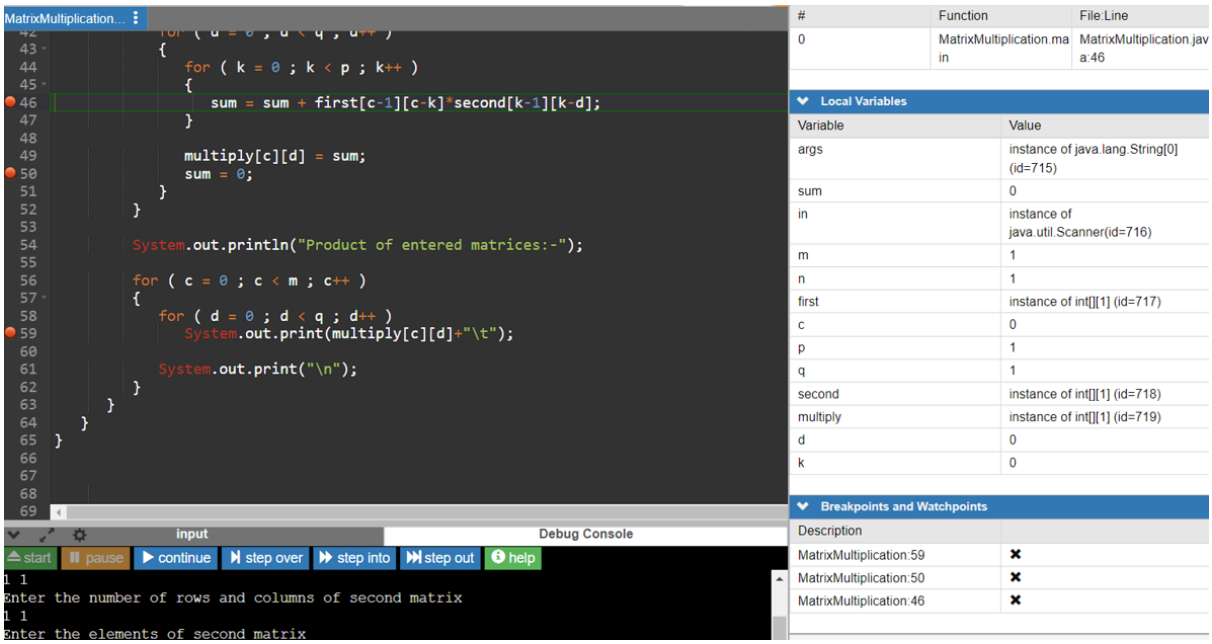
a) What are the steps you have taken to fix the error you identified in the code fragment?

- **Step 1**: Update the access in the multiplication logic from first[c-1][c-k] to first[c][k].
- **Step 2**: Update the access in the multiplication logic from second[k-1][k-d] to second[k][d].

- **Step 3**: Correct the input prompt from "Enter the number of rows and columns of first matrix" to "Enter the number of rows and columns of second matrix."
- **Step 4**: Move the initialization of sum to the beginning of the d loop to ensure it starts at 0 for each element in the resulting matrix.

3. Submit your complete executable code?

The complete executable code is attached with Java folder



Quadratic Probing

1. **How many errors are there in the program? Mention the errors you have identified.**

   There are several issues in the program:

   - **Error 1**: In the insert method, the expression I + = (I + h / h--) % maxSize; contains a syntax error due to an unnecessary

space in + =. The correct syntax should be I += (h * h) % maxSize; for proper quadratic probing.

- **Error 2**: In the get method, the expression I = (I + h * h++) % maxSize; causes the index I to increment incorrectly because h++ increments h after it is evaluated. It should be I = (I + h * h) % maxSize; followed by h++.
- **Error 3**: Similarly, in the remove method, the same issue exists with the increment of h. It should be fixed to I = (I + h * h) % maxSize; and h++.
- **Error 4**: In the remove method, the line keys[i] = vals[i] = null; should be replaced with two separate assignments: keys[i] = null; vals[i] = null; for clarity.
- **Error 5**: The comment /** maxSizeake object of QuadraticProbingHashTable **/ contains a typo and should be corrected to /** Make object of QuadraticProbingHashTable **/.

2. How many breakpoints do you need to fix those errors?

There are 5 breakpoints needed to fix the errors:

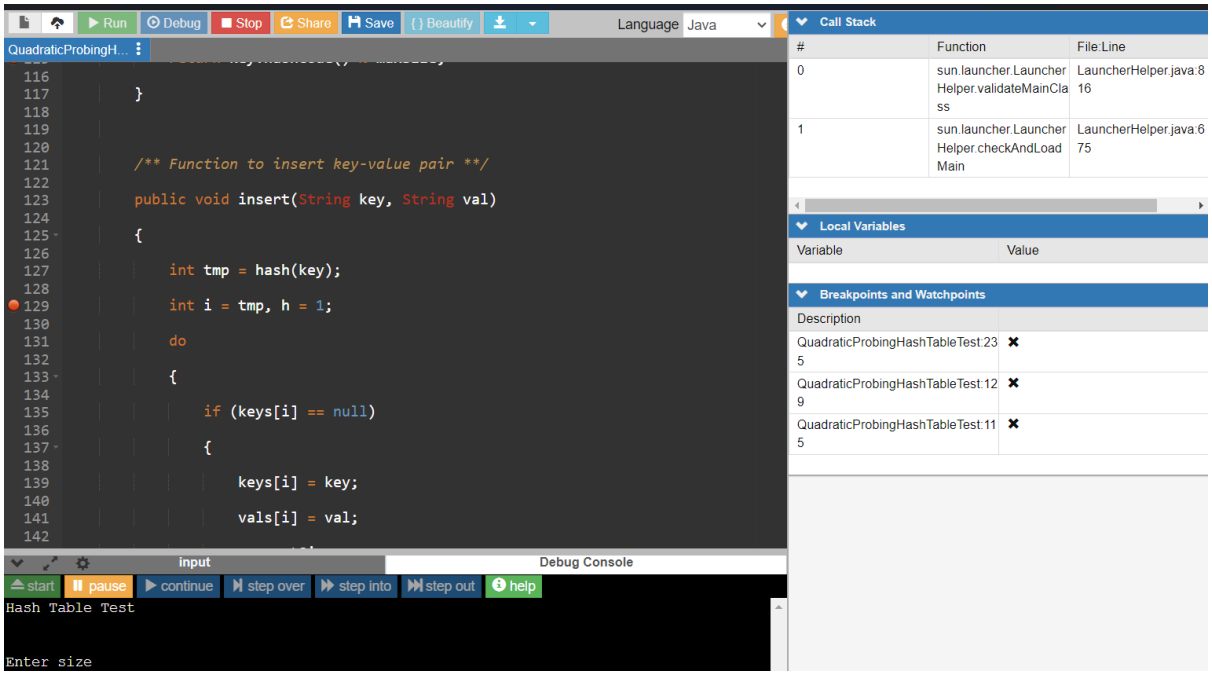
- **Breakpoint 1**: At the line I + = (I + h / h--) % maxSize; in the insert method to fix the syntax.
- **Breakpoint 2**: At the line I = (I + h * h++) % maxSize; in the get method to fix the increment logic.
- **Breakpoint 3**: At the line I = (I + h * h++) % maxSize; in the remove method to fix the increment logic.
- **Breakpoint 4**: At the line keys[i] = vals[i] = null; in the remove method to separate the assignments.
- **Breakpoint 5**: At the comment /** maxSizeake object of QuadraticProbingHashTable **/ for correction.

a) What are the steps you have taken to fix the error you identified in the code fragment?

- **Step 1**: Change I + = (I + h / h--) % maxSize; to I += (h * h) % maxSize; in the insert method.
- **Step 2**: Change I = (I + h * h++) % maxSize; to I = (I + h * h) % maxSize; in both the get and remove methods, and increment h after the assignment.
- **Step 3**: Replace keys[i] = vals[i] = null; in the remove method with two separate statements: keys[i] = null; vals[i] = null;.
- **Step 4**: Fix the comment /** maxSizeake object of QuadraticProbingHashTable **/ to /** Make object of QuadraticProbingHashTable **/.

3. Submit your complete executable code?

The complete executable code is attached with Java folder



Sorting Array

1. How many errors are there in the program? Mention the errors you have identified.

There are 2 errors in the program:

- **Error 1: Incorrect loop condition in the outer loop.**
  The condition in the outer loop is for (int i = 0; i >= n; i++), which is incorrect. It should be for (int i = 0; i < n; i++) to iterate over the array elements properly.
- **Error 2: Incorrect comparison logic in the inner loop.**
  The current code uses if (a[i] <= a[j]), which sorts the array in descending order instead of ascending order. It should be if (a[i] > a[j]) to achieve ascending order.

2. How many breakpoints do you need to fix those errors?

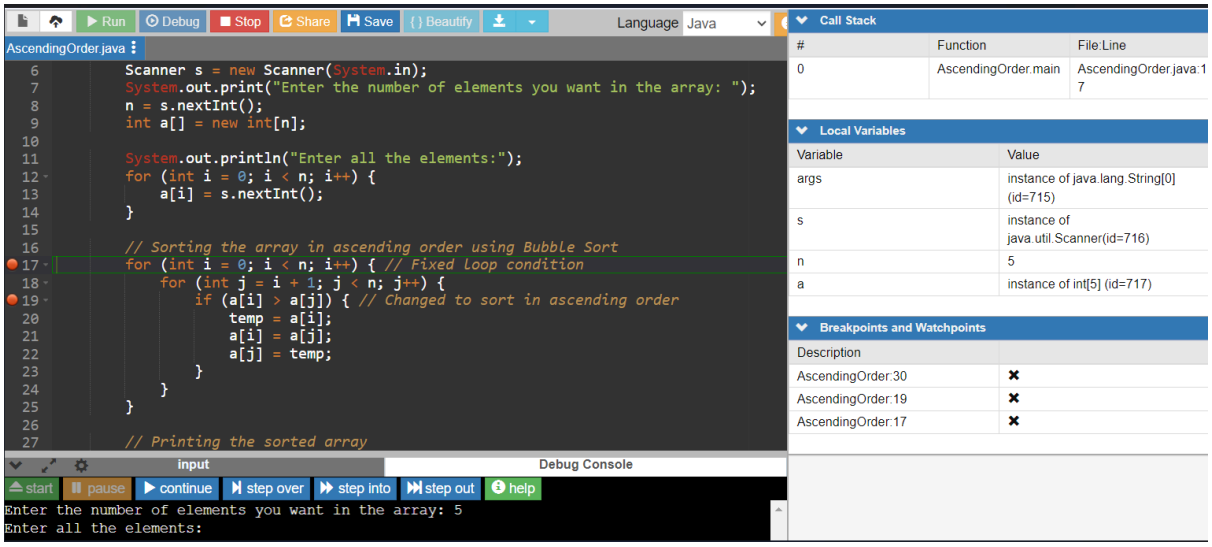
a) What are the steps you have taken to fix the error you identified in the code fragment?

There are 2 breakpoints needed in the program:

- **Breakpoint 1:** At the line where the outer loop condition is checked (for (int i = 0; i < n; i++)), to verify if the loop iterates correctly through all elements.
- **Breakpoint 2:** At the line where the comparison if (a[i] > a[j]) is made, to ensure the sorting logic is correctly comparing the elements for ascending order.

3. Submit your complete executable code?

The complete executable code is attached with Java folder



Stack Implementation

1. How many errors are there in the program? Mention the errors you have identified.

Solution) There are 4 errors in the program:

- **Error 1: Incorrect decrement in the push method.**
  The line top--; should be top++; because the top index should increase when a value is pushed onto the stack.

- **Error 2: Incorrect increment in the pop method.**
  The line top++; should be top--; since the top index should decrease when a value is popped from the stack.
- **Error 3: Incorrect loop condition in the display method.**
  The loop condition for(int i=0;i>top;i++) should be for(int i=0;i<=top;i++) to ensure that the loop iterates over the stack elements from index 0 to top.
- **Error 4: The stack is not initialized to store any values upon pushing.**
  After fixing the increment of top, the line stack[top]=value; should correctly set the value in the stack without causing an ArrayIndexOutOfBoundsException.

2. How many breakpoints do you need to fix those errors?

a) What are the steps you have taken to fix the error you identified in the code fragment?

Solution) There are 4 breakpoints needed in the program:

- **Breakpoint 1:** At the line stack[top] = value; in the push method to check the value of top and ensure it's within bounds.
- **Breakpoint 2:** At the line top--; in the pop method to check if top is correctly decremented.
- **Breakpoint 3:** At the loop condition in the display method to confirm that the loop iterates through the correct indices.
- **Breakpoint 4:** At the beginning of the push method to confirm that top is initialized correctly before adding elements.

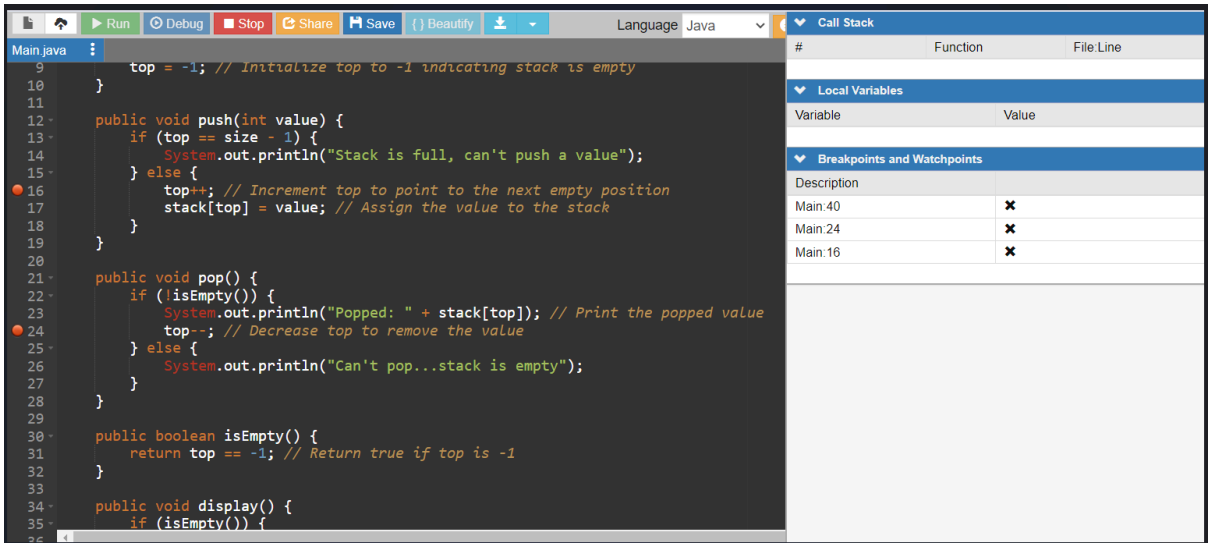
- **Steps Taken to Fix the Errors**

Here are the steps taken to correct the errors identified:

1. **Step 1:** Update the push method to increment top instead of decrementing it.
2. **Step 2:** Update the pop method to decrement top instead of incrementing it.
3. **Step 3:** Correct the loop condition in the display method from i > top to i <= top.
4. **Step 4:** Ensure that the stack operations correctly manage the top variable and that it does not exceed the bounds of the stack array.

3. Submit your complete executable code?

Solution) The complete executable code is attached with Java folder



Tower of Hanoi

1. How many errors are there in the program? Mention the errors you have identified.

Solution) There are 3 errors in the program:

- **Error 1:** The recursive call in the second call to doTowers uses topN++ and inter--. These expressions are incorrect because

they modify the values rather than passing the original ones. Instead, it should be topN - 1 and inter should remain unchanged.

- **Error 2:** The parameters passed to the recursive call are not correct. The characters cannot be incremented or decremented like integers. Instead, you should pass the same characters.
- **Error 3:** The first call to doTowers in the else block should not modify the variables; they should remain the same. The correct call should pass the from, inter, and to arguments properly.

2. How many breakpoints do you need to fix those errors?

a) What are the steps you have taken to fix the error you identified in the code fragment?

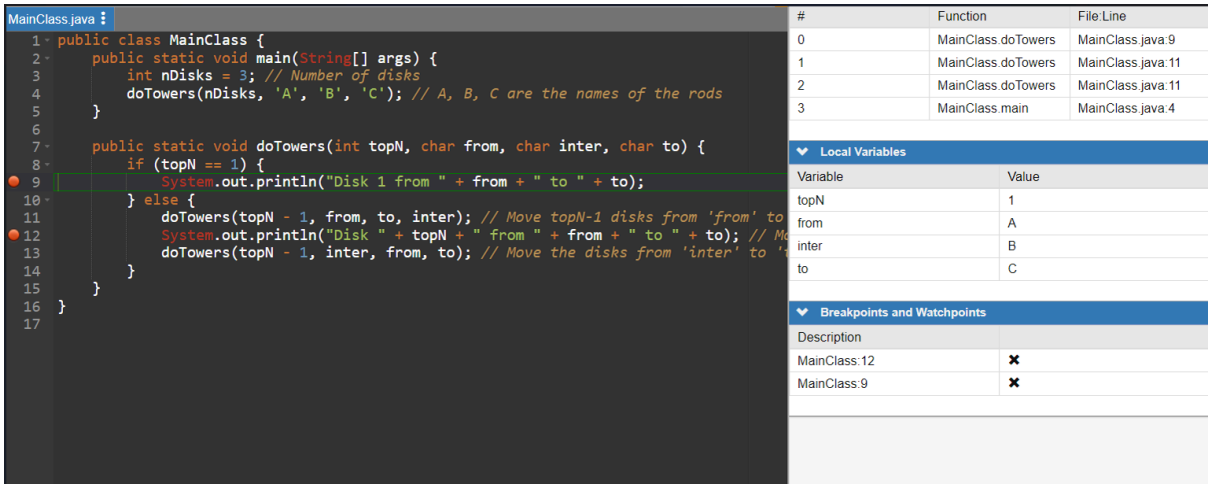
Solution) There are 2 breakpoints needed in the program:

- **Breakpoint 1:** At the first recursive call to doTowers to ensure the correct parameters are passed.
- **Breakpoint 2:** At the second recursive call to doTowers to check the parameters and confirm they are not modified incorrectly.

 Steps Taken to Fix the Errors

Here are the steps taken to correct the errors identified:

1. **Step 1:** Change the second call to doTowers from topN++ and inter-- to simply passing topN - 1, to, and from.
2. **Step 2:** Ensure that the recursive calls maintain the correct character parameters for the source, intermediate, and destination rods.
3. **Step 3:** Verify the flow of the program to ensure that the output matches the expected Tower of Hanoi solution.



III Static Analysis Tools:

I have used Pylint for static analysis of Python program used in section I.