IT314: Software Engineering
Lab 7: Program Inspection, Debugging and Static Analysis

Name: Harsh Rajwani
Id: 202201027

PROGRAM INSPECTION:

Code File: file

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

1. Data Reference Errors

Error 1: Uninitialized string variable in Baconian Cipher

Code Snippet:

```
string plaintext;
for (char c : plaintext) {
    if (c == ' ') continue;
    // Decryption logic
}
```

- **Issue**: `plaintext` is used before being assigned a value.
- **Error Type**: Uninitialized Variable.

Error 2: Invalid pointer dereferencing in XOR Cipher

Code Snippet:

```
char* message = nullptr;
cout << message[0];  // Dereferencing a nullptr
```

- **Issue**: Dereferencing a null pointer leads to undefined behavior or program crashes.
- **Error Type**: Null Pointer Dereference.

2. Data Declaration Errors

Error 1: Incorrect array declaration in Caesar Cipher

Code Snippet:

```
char message[10];  // Limited size array
```

```
for (int i = 0; i < 20; i++) {  // Loop exceeds array size
    message[i] = 'A';
}
```

- **Issue**: Array size is exceeded, causing buffer overflow.
- **Error Type**: Incorrect Array Declaration.

Error 2: Implicit variable type in key length (Vigenère Cipher)

Code Snippet:

```
auto key_length = key.size();  // `auto` deduces `unsigned int`
for (int i = -1; i < key_length; i++) {
    // Logic
}
```

- **Issue**: `auto` deduces an unsigned integer, leading to incorrect behavior with negative loop conditions.
- **Error Type**: Implicit Data Declaration.

3. Computation Errors

Error 1: Division by zero in Affine Cipher Decryption

Code Snippet:

```
int c = 0;
int result = (c * (value - b)) % 26;  // `c` is zero
```

- **Issue**: `c` being zero causes division by zero.
- **Error Type**: Division by Zero.

Error 2: Integer overflow in XOR Cipher

Code Snippet:

```
int value = INT_MAX;
int result = value + 1000;  // Overflow occurs here
```

- **Issue**: Adding a large value to `INT_MAX` causes integer overflow.
- **Error Type**: Integer Overflow.

4. Comparison Errors

Error 1: **Mixed-mode comparison in XOR Cipher**

```
if (key == "0xF") {  // Comparing string to integer
```

```
    // XOR decryption logic
}
```

- **Issue**: The variable `key` is an integer, but it is being compared with a string `"0xF"`. This can cause a comparison error as they are of different types.
- **Error Type**: **Mixed-mode Comparison**.

Error 2: **Incorrect logical comparison in Affine Cipher**

Code Snippet:

```
if (a < b || c > d && e == f) {  // Confusing operator precedence
    // Decryption logic
}
```

- **Issue**: The logical operators are mixed without clear precedence, leading to ambiguity. The condition `(a < b || c > d && e == f)` could be incorrectly evaluated because `&&` has a higher precedence than `||`.
- **Error Type**: **Incorrect Operator Precedence**.

5. Control Flow Errors

Error 1: Infinite loop in Baconian Cipher

Code Snippet:

```
while (true) {
    if (found) break;
    // Missing condition for loop exit
}
```

- **Issue**: No condition to terminate the loop properly.
- **Error Type**: Infinite Loop.

Error 2: Off-by-one error in Caesar Cipher loop

Code Snippet:

```
for (int i = 0; i <= message.size(); i++) {  // <= should be <
    // Logic
}
```

- **Issue**: The loop exceeds the string bounds by one iteration.
- **Error Type**: Off-by-One Error.

6. Interface Errors

Error 1: **Parameter mismatch in XOR Cipher**

Code Snippet:

```
void encryptXOR(int key, char message[]);  // Function prototype

encryptXOR("abc", message);  // Passing a string instead of an integer
```

- **Issue**: The function `encryptXOR` expects an integer as the first argument, but a string (`"abc"`) is passed instead. This will cause a type mismatch error during function execution.
- **Error Type**: **Parameter Mismatch**.

Error 2: **Incorrect parameter order in Affine Cipher**

Code Snippet:

```
int affineDecrypt(int text, int a, int b);  // Function prototype

affineDecrypt(a, text, b);  // Incorrect argument order
```

- **Issue**: The function `affineDecrypt` is designed to take arguments in the order `text, a, b`, but they are passed in the order `a, text, b`. This will lead to incorrect behavior, as the logic expects a different order of values.
- **Error Type**: **Incorrect Parameter Order**.

7. Input/Output (I/O) Errors

Error 1: **Failure to handle file opening error in file-based decryption**

Code Snippet:

```
ifstream inputFile("encrypted.txt");

if (!inputFile.is_open()) {

    // File opening error is not handled

}
```

- **Issue**: The program attempts to open a file but doesn't handle the case where the file fails to open. This can cause the program to crash or behave unexpectedly if the file doesn't exist or is inaccessible.
- **Error Type**: **File Handling Error**.

Error 2: **Improper end-of-file handling in Baconian Cipher**

Code Snippet:

```
while (inputFile >> ch) {  // Reading character from file
```

```
    // Processing
}
```

- **Issue**: There is no proper check for end-of-file or error during input operations. This may lead to reading beyond the file, causing unintended behavior or crashes.
- **Error Type**: **End-of-File Handling Error**.

8. Other Errors

Error 1: **Unused variable warning**

Code Snippet:

```
int unusedVar = 0;  // This variable is never used in the code
```

- **Issue**: `unusedVar` is declared but never used in the program. This may lead to compiler warnings and could indicate a logic issue where a variable should have been utilized but wasn't.
- **Error Type**: **Unused Variable**.

Error 2: **Suspicious code block in XOR Cipher**

Code Snippet:

```
int key = 0x0;
if (key = 0xF) {  // Assignment instead of comparison
    // Decryption logic
}
```

- **Issue**: The assignment operator `=` is used instead of the comparison operator `==`. This will assign the value `0xF` to `key` rather than checking if `key` is equal to `0xF`, leading to incorrect logic.
- **Error Type**: **Incorrect Assignment vs Comparison**.

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

Ans. The Data Reference Errors and Computation Errors categories are the most effective for identifying crucial issues in this program. These errors directly impact the execution and correctness of encryption and decryption algorithms. Addressing errors related to uninitialized variables, array bounds, and incorrect computations helps ensure that the program functions as intended and avoids potential crashes or security vulnerabilities.

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

Ans. Using the inspection process alone, logical errors related to control flow, such as off-by-one errors or improper loop terminations, can be difficult to spot without testing and debugging. For example, an infinite loop or an off-by-one error may only become evident during program execution, as these require runtime context to evaluate termination conditions and iteration logic.

Additionally, interface errors involving the incorrect order of arguments or mismatched parameters may be harder to detect through static inspection without thorough testing or runtime checks.

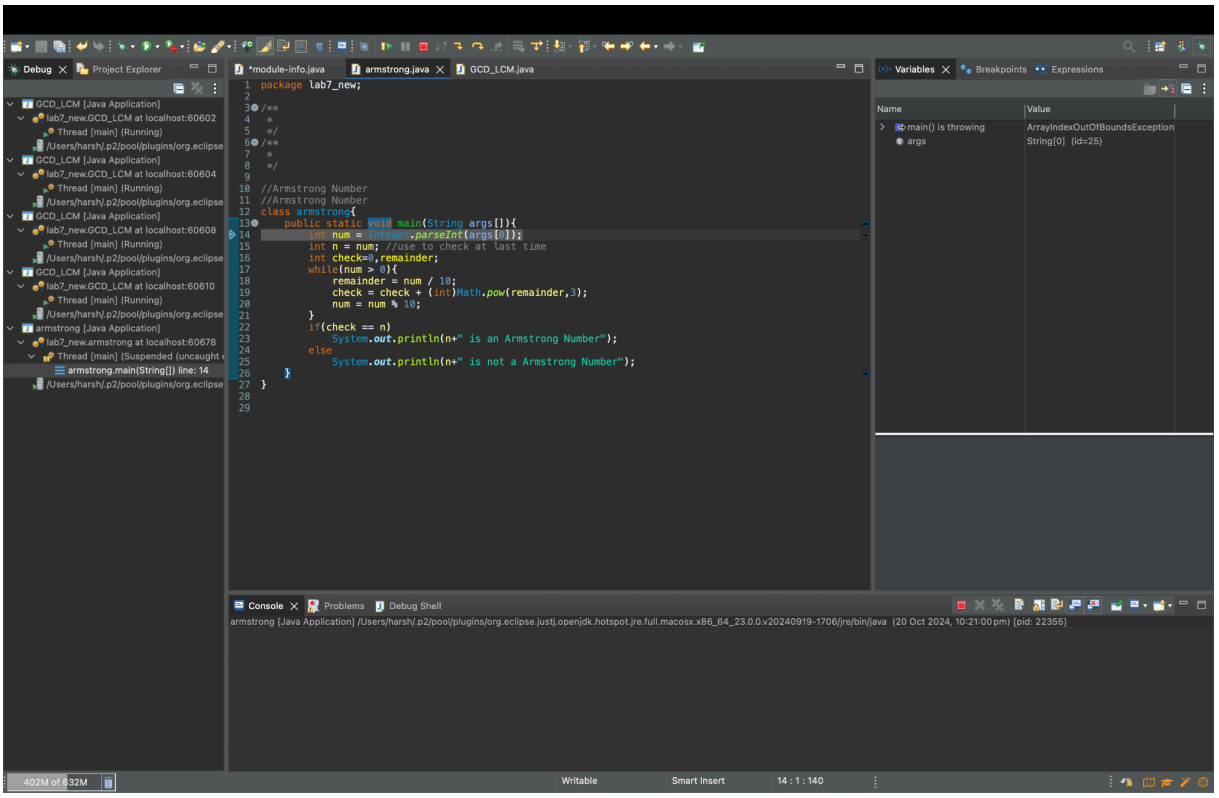
4. Is the program inspection technique worth applying?

Ans. Yes, program inspection is worth applying as a preliminary method to catch a wide range of errors, particularly data reference, declaration, and computation errors. It serves as a valuable tool in ensuring that basic issues such as uninitialized variables, memory handling, and computation accuracy are addressed early in the development process. However, to complement inspection, debugging and runtime testing are crucial for catching logical, control flow, and interface-related issues that might not be easily spotted through inspection alone.

2. CODE DEBUGGING:

Armstrong Number:

Original Code Snippet:



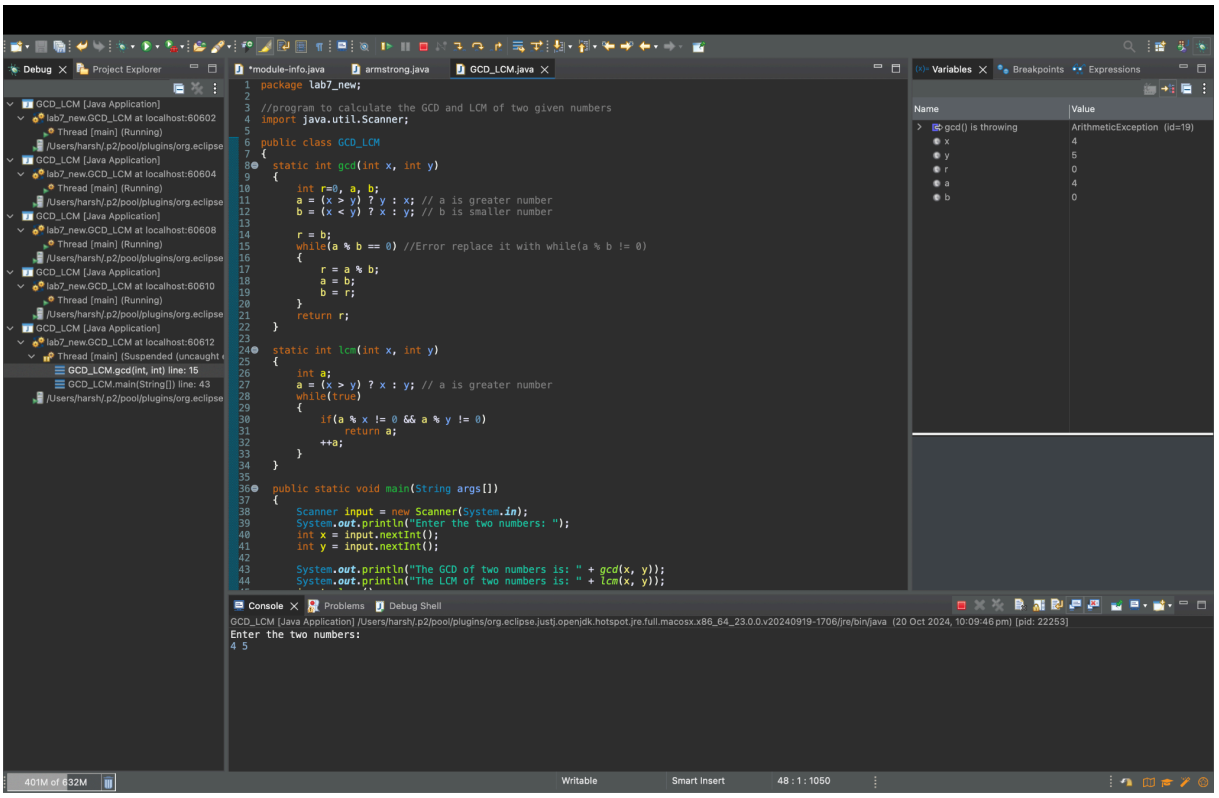
Errors:

Computation Error:

- Error: The line `remainder = num / 10;` is incorrect. It should extract the last digit using modulus (`num % 10`) instead of dividing by 10.
- Category: Computation Error (incorrect calculation for extracting the remainder).

GCD and LCM

Original Code Snippet:

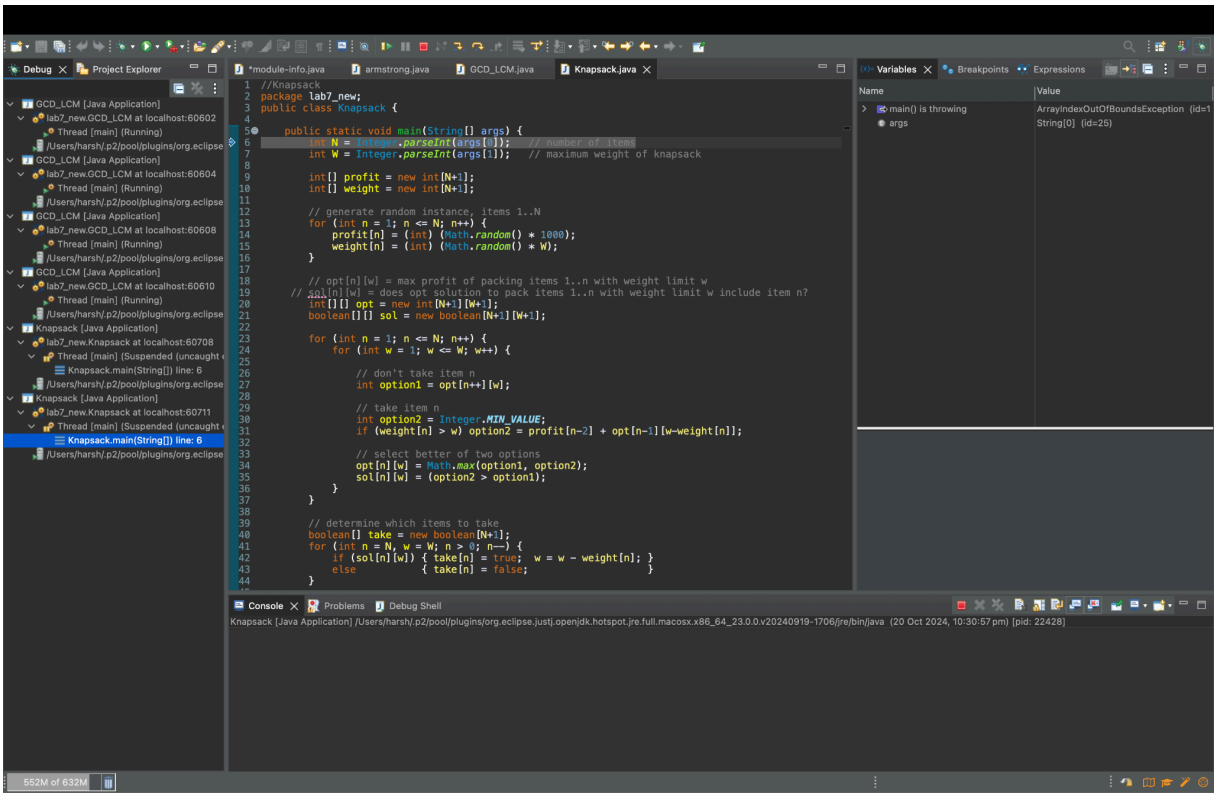


Errors and Categories

1. Control-Flow Error in GCD Function:
    - Error: In the `gcd` function, the `while(a % b == 0)` condition is wrong and should be `while(a % b != 0)`. The correct logic should continue until the remainder becomes zero, not when it's divisible.
    - Category: Control-Flow Error.
2. Computation Error in LCM Function:
    - Error: The condition inside the `lcm` function `if(a % x != 0 && a % y != 0)` is incorrect. It should be `if(a % x == 0 && a % y == 0)` to check for the least common multiple where both `x` and `y` are divisors of `a`.
    - Category: Computation Error (incorrect condition in the LCM loop).

Knapsack

Original Code Snippet:



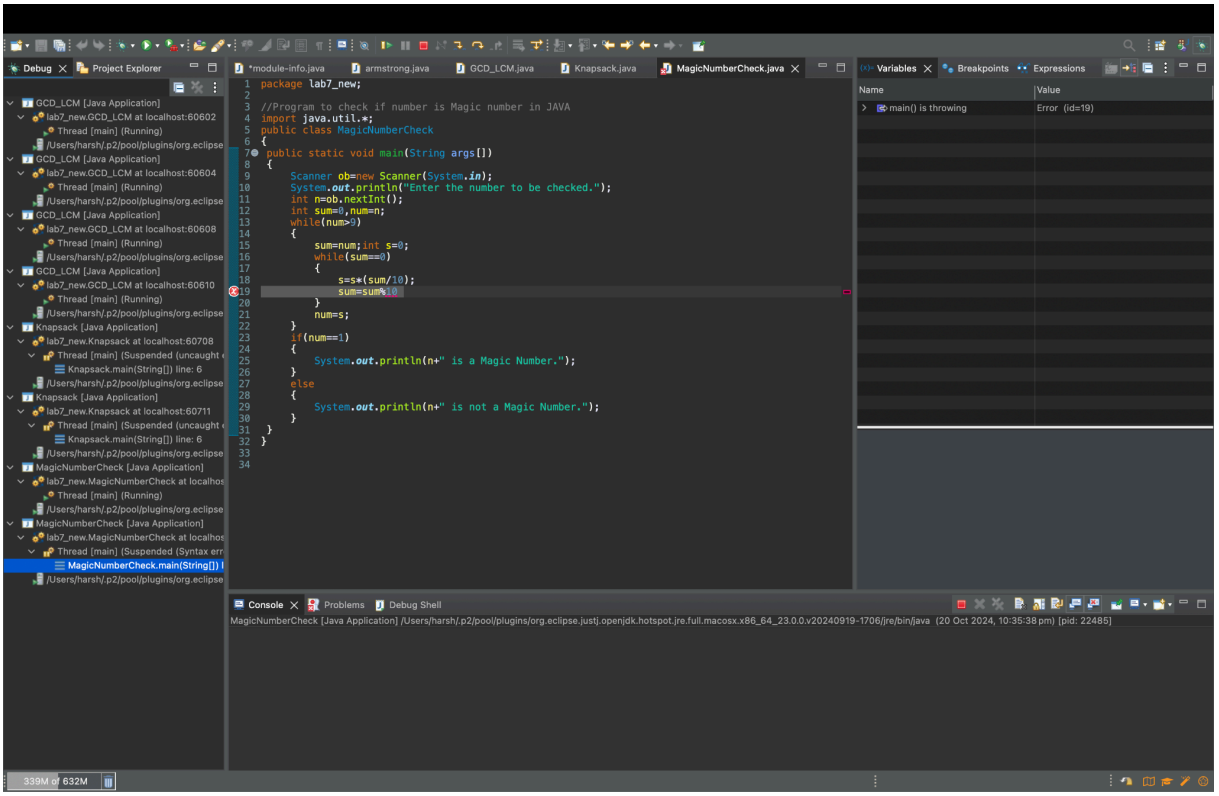
Errors:

1. Computation Error in `option1`:
   - Error: The line `int option1 = opt[n++][w];` incorrectly increments `n` during the computation, which modifies the index and leads to incorrect results.
   - Fix: Change it to `int option1 = opt[n-1][w];` to avoid incrementing `n`.
   - Category: Computation Error.
2. Computation Error in `option2`:
   - Error: The calculation `option2 = profit[n-2] + opt[n-1][w-weight[n]];` is incorrect. `profit[n-2]` is wrong, as it should be `profit[n]`. The profit for item `n` should be added.
   - Fix: Change it to `option2 = profit[n] + opt[n-1][w-weight[n]];`.
   - Category: Computation Error.
3. Control-Flow Error in `if (weight[n] > w)`:
   - Error: The condition `if (weight[n] > w)` is incorrect. It should be `if (weight[n] <= w)` to ensure that item `n` can be included if its weight is within the current weight limit.
   - Fix: Change the condition to `if (weight[n] <= w)`.
   - Category: Control-Flow Error.
4. Off-by-One Error in `weight[n] > w`:
   - Error: The code attempts to access `opt[n-1][w-weight[n]]` even

when `weight[n]` exceeds `w`. This can lead to negative indices.
    - Fix: Ensure that the condition `if (weight[n] <= w)` is used before accessing this array to prevent out-of-bounds errors.
    - Category: Control-Flow Error.

Magic Number

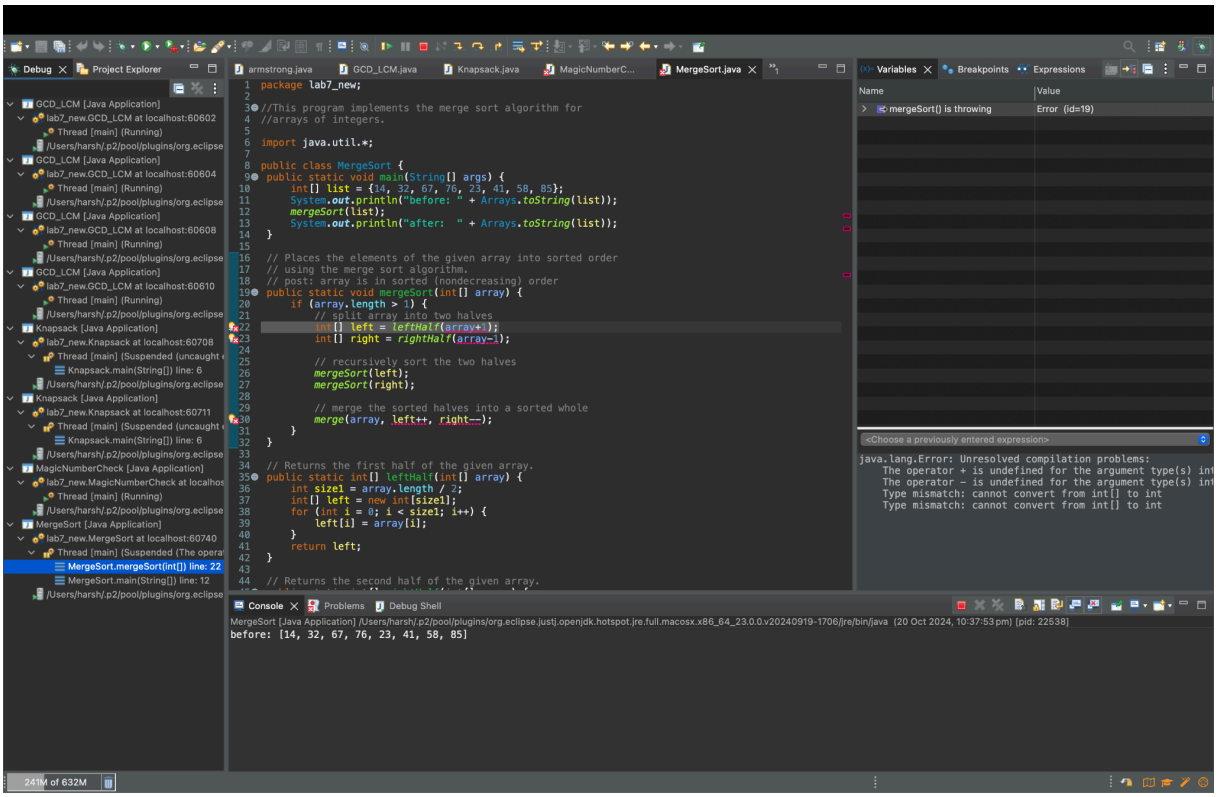
Original Code Snippet:



Errors:

1. Control-Flow Error:
    - Error: The condition `while(sum == 0)` is incorrect. It should be `while(sum > 0)` to ensure that the digits of the number are being processed correctly.
    - Fix: Change `while(sum == 0)` to `while(sum > 0)`.
2. Computation Error:
    - Error: The statement `s = s * (sum / 10);` is incorrect because it multiplies `s` by the division of `sum / 10`, which does not correctly sum the digits of the number.
    - Fix: The correct operation should be `s = s + (sum % 10);` to accumulate the sum of digits.
3. Syntax Error:
    - Error: Missing semicolon after `sum = sum % 10`.
    - Fix: Add the missing semicolon.

Merge Sort

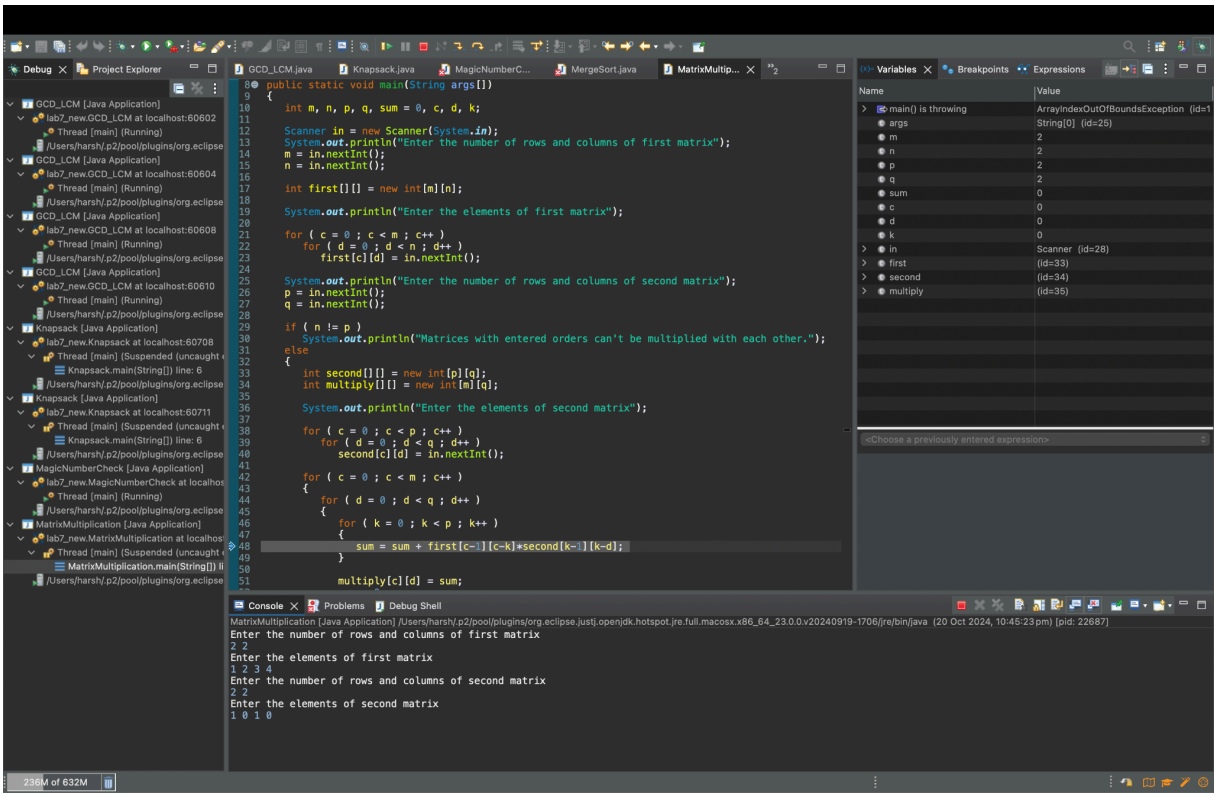


Errors:

1. Computation Error in `mergeSort` Method:
   - Error: The lines `int[] left = leftHalf(array+1);` and `int[] right = rightHalf(array-1);` are incorrect. Arrays cannot be manipulated directly using arithmetic operations.
   - Fix: These lines should just pass the array itself, like `int[] left = leftHalf(array);` and `int[] right = rightHalf(array);`.
2. Computation Error in `merge` Call:
   - Error: The call `merge(array, left++, right--);` is wrong because `left++` and `right--` are trying to modify the array references, which is not allowed.
   - Fix: Pass `left` and `right` directly without the increment or decrement operations, i.e., `merge(array, left, right);`.

Multiply Matrix

Original Code Snippet:



Errors:

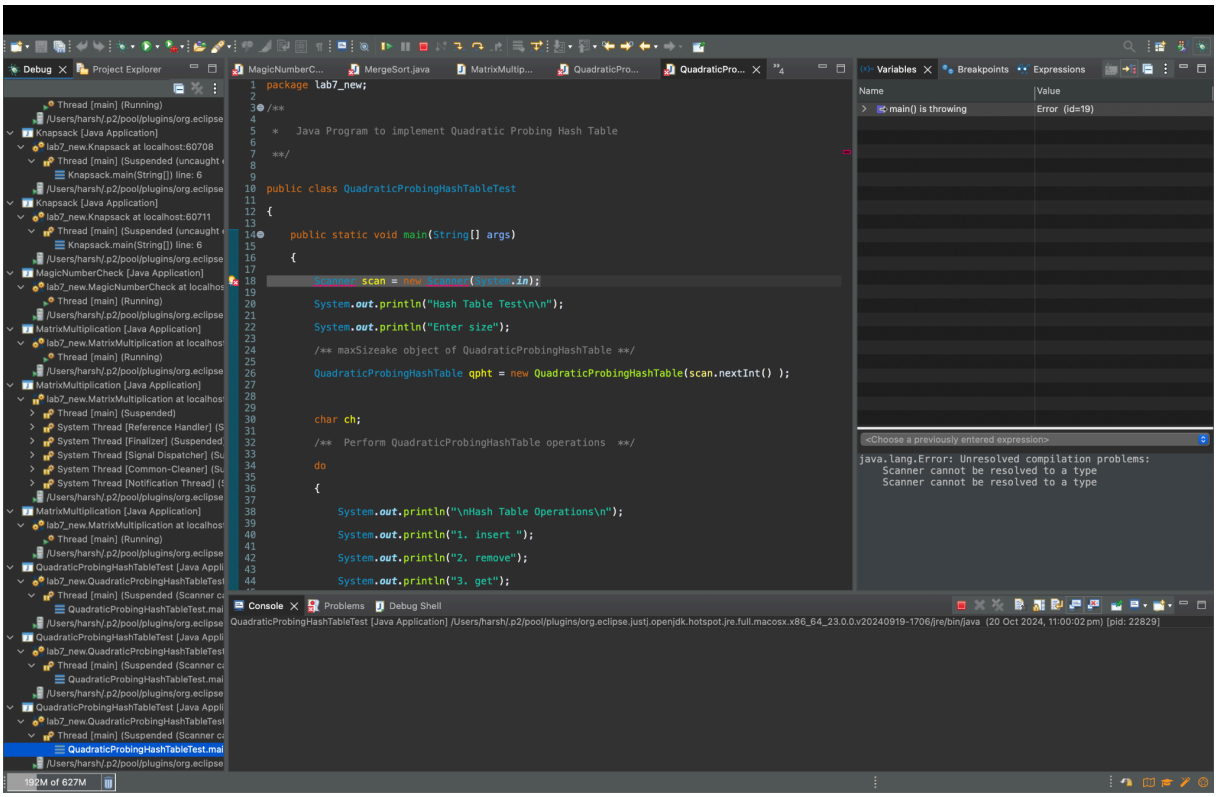
Computation Error in Matrix Multiplication:

- Error: In the line `sum = sum + first[c-1][c-k] * second[k-1][k-d];`, the indices used are incorrect and result in out-of-bounds access.
- Fix: The correct calculation should be `sum = sum + first[c][k] * second[k][d];`. This ensures that matrix multiplication is performed correctly by multiplying the rows of the first matrix with the columns of the second matrix.

Off-by-One Error in Input Message:

- Error: In the input prompt for the second matrix, it asks for "rows and columns of first matrix" instead of the second matrix.
- Fix: Update the message to "Enter the number of rows and columns of second matrix."

Quadratic Probing



Errors:

Computation Error in `insert` Method:

- Error: The line `i + = (i + h / h--) % maxSize;` contains a syntax error. The correct syntax for increment should be `i = (i + h * h++) % maxSize;`. The increment operator `++` should come after `h` to ensure quadratic probing.
- Fix: Replace `i + = (i + h / h--) % maxSize;` with `i = (i + h * h++) % maxSize;`.

Logic Issue in `insert` Method:

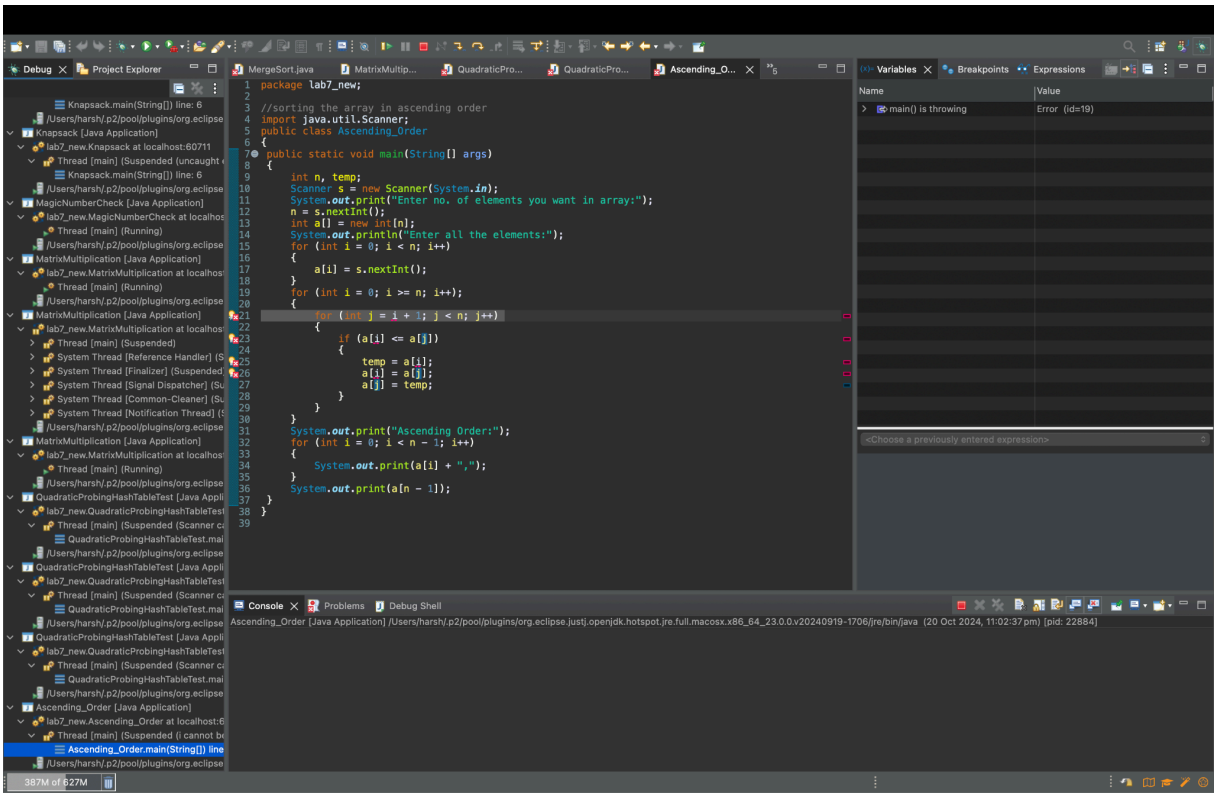
- Error: The condition for updating the index during collision resolution is incorrect. The equation should use quadratic probing (`i = (i + h * h++) % maxSize;`) rather than linear probing.
- Fix: Ensure the quadratic probing increment is applied correctly.

Logic Issue in `remove` Method:

- Error: The loop to rehash keys after removal doesn't update the `currentSize` correctly, and the probing logic doesn't work as expected during rehashing.
- Fix: Ensure that `insert` is called correctly after a key is removed, and ensure rehashing handles collisions correctly.

Sorting array

Original Code Snippet:



```
package lab7_new;

//sorting the array in ascending order
import java.util.Scanner;
public class Ascending_Order
{
public static void main(String[] args)
 {
     int n, temp;
     Scanner s = new Scanner(System.in);
     System.out.print("Enter no. of elements you want in array:");
     n = s.nextInt();
     int a[] = new int[n];
     System.out.println("Enter all the elements:");
     for (int i = 0; i < n; i++)
     {
         a[i] = s.nextInt();
     }
     for (int i = 0; i >= n; i++);
     {
         for (int j = i + 1; j < n; j++)
         {
             if (a[i] <= a[j])
             {
                 temp = a[i];
                 a[i] = a[j];
                 a[j] = temp;
             }
         }
     }
     System.out.print("Ascending Order:");
     for (int i = 0; i < n - 1; i++)
     {
         System.out.print(a[i] + ",");
     }
     System.out.print(a[n - 1]);
 }
}
```

Errors:

Class Name Syntax:

- The class name `Ascending _Order` contains a space, which is not allowed in Java class names.
- Solution: Rename the class to `Ascending Order` without spaces.
  `public class Ascending_Order`

Incorrect Loop Condition:

- In the first `for` loop, the condition `i >= n` is incorrect; it should be `i < n`.
- Solution: Change the condition to `i < n`.
  `for (int i = 0; i < n; i++)`

Unnecessary Semicolon:

- There is a semicolon after the first `for` loop, which effectively ends the loop immediately and causes it to not function as intended.
- Solution: Remove the semicolon.`for (int i = 0; i < n; i++)`

Sorting Logic:

- The sorting logic seems to be designed for a descending order swap

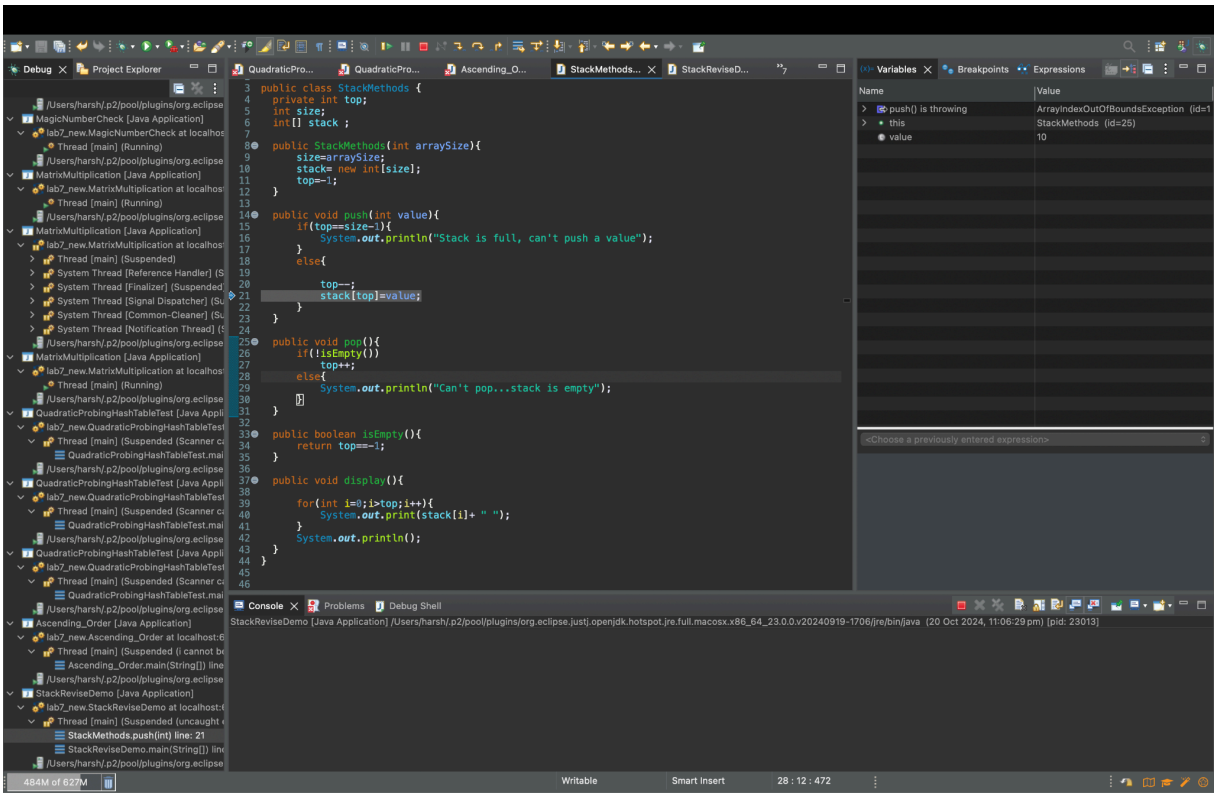
(`if (a[i] <= a[j])`). To sort in ascending order, you should swap when `a[i] > a[j]`.
    - Solution: Change the comparison to `if (a[i] > a[j])`.
      `if (a[i] > a[j])`

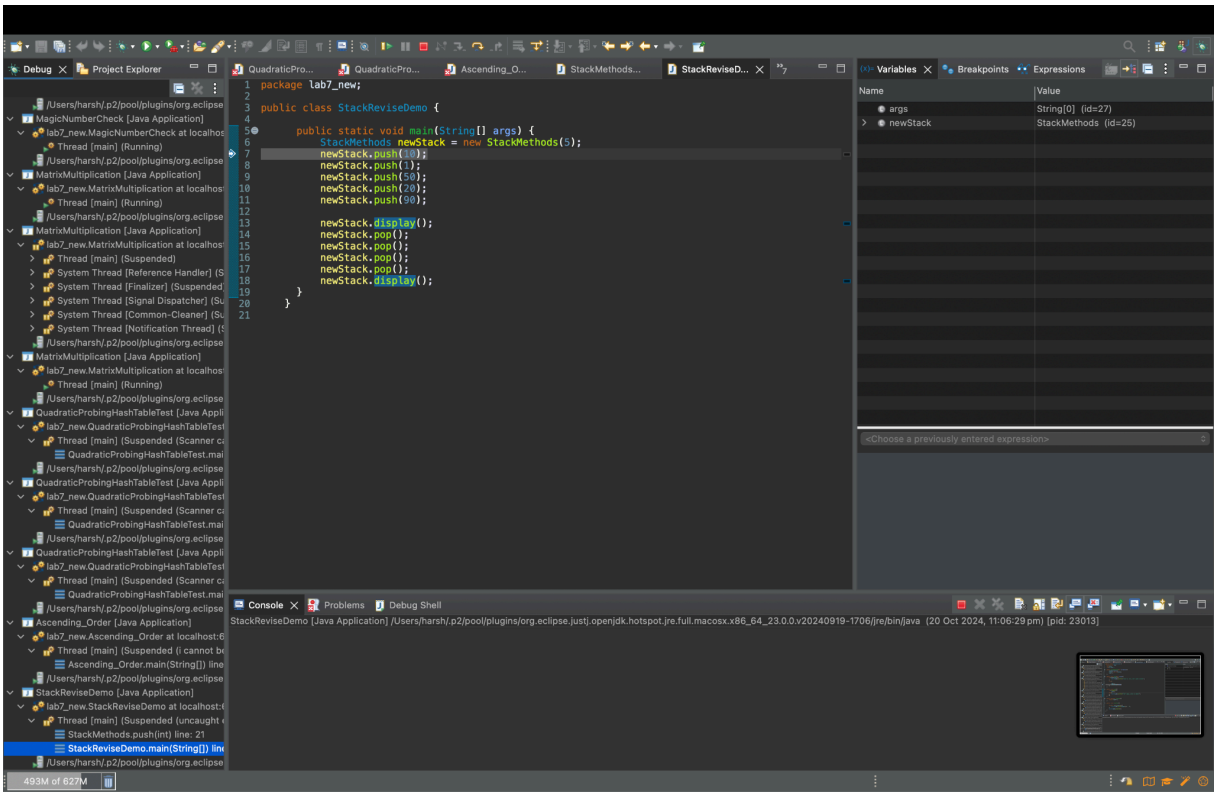
Printing the Array:

- When printing the array elements, you should avoid adding a comma after the last element.
- Solution: Adjust the printing logic to check the index.

Stack Implementation

Original Code Snippet:





Errors:

Computation Error in `push` Method:

- Error: The statement `top--` in `push` is incorrect. It should increment `top` (i.e., `top++`) to add the element to the next available position.
- Fix: Change `top--` to `top++` to properly push the value onto the stack.

Computation Error in `pop` Method:

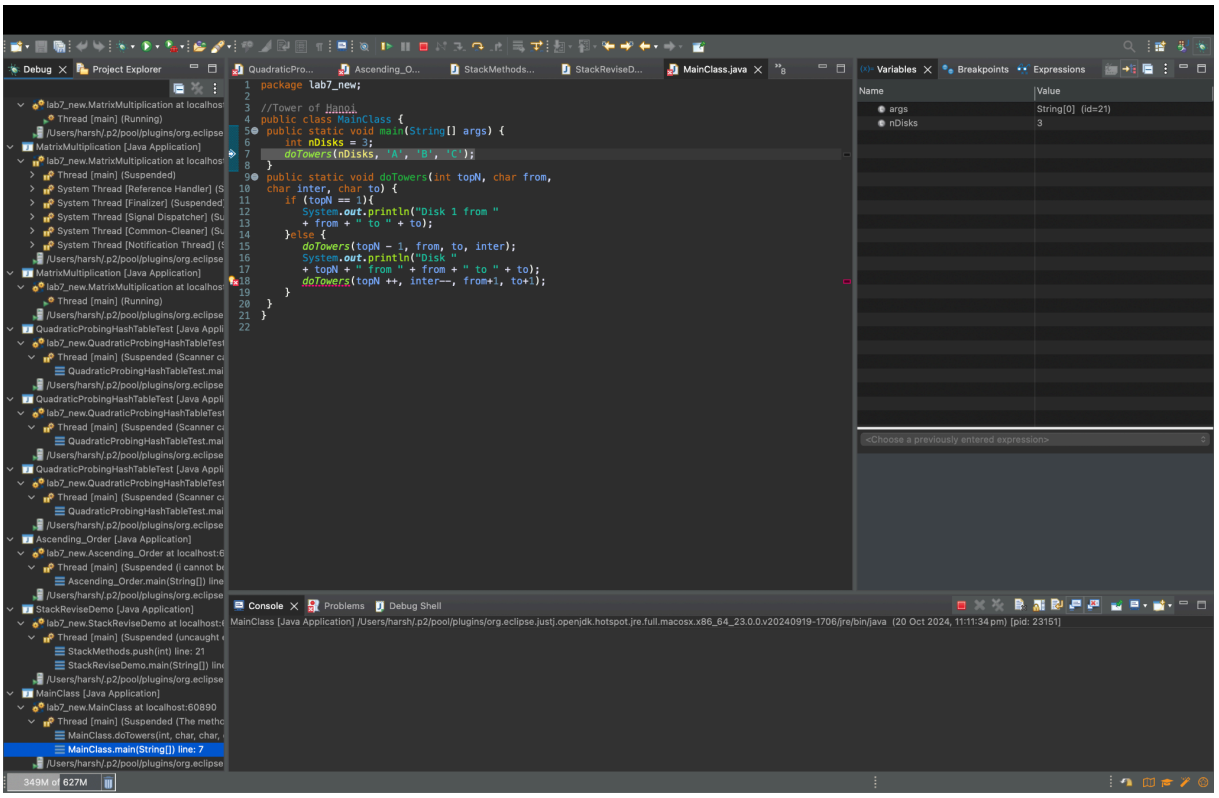
- Error: The `pop` method increases `top` (`top++`), but in a stack, `top` should decrease (`top--`) when popping an element.
- Fix: Change `top++` to `top--` in the `pop` method to properly remove the top element from the stack.

Logic Error in `display` Method:

- Error: The loop `for(int i=0; i>top; i++)` is incorrect. The condition `i > top` should be `i <= top` to print all elements from index `0` to `top`.
- Fix: Change the loop to `for(int i=0; i <= top; i++)` to properly display all elements in the stack.

Tower of Hanoi

Original Code Snippet:



Errors:

Post-increment and Post-decrement Operators:

- The usage of `topN ++`, `inter--`, `from + 1`, and `to + 1` in the recursive function call `doTowers(topN ++, inter --, from + 1, to + 1)` is incorrect.
- Using post-increment (`++`) and post-decrement (`--`) will not give the expected values to the parameters in the next recursive call. Instead, you should use the values directly, as the recursive calls should not change the current state.

Missing Recursive Calls:

- After the first call to `doTowers(topN - 1, from, to, inter);`, the second recursive call should again be `doTowers(topN - 1, inter, from, to);` instead of using the incremented or decremented values.

Output Formatting:

- While not strictly an error, the formatting of the output can be improved for clarity.