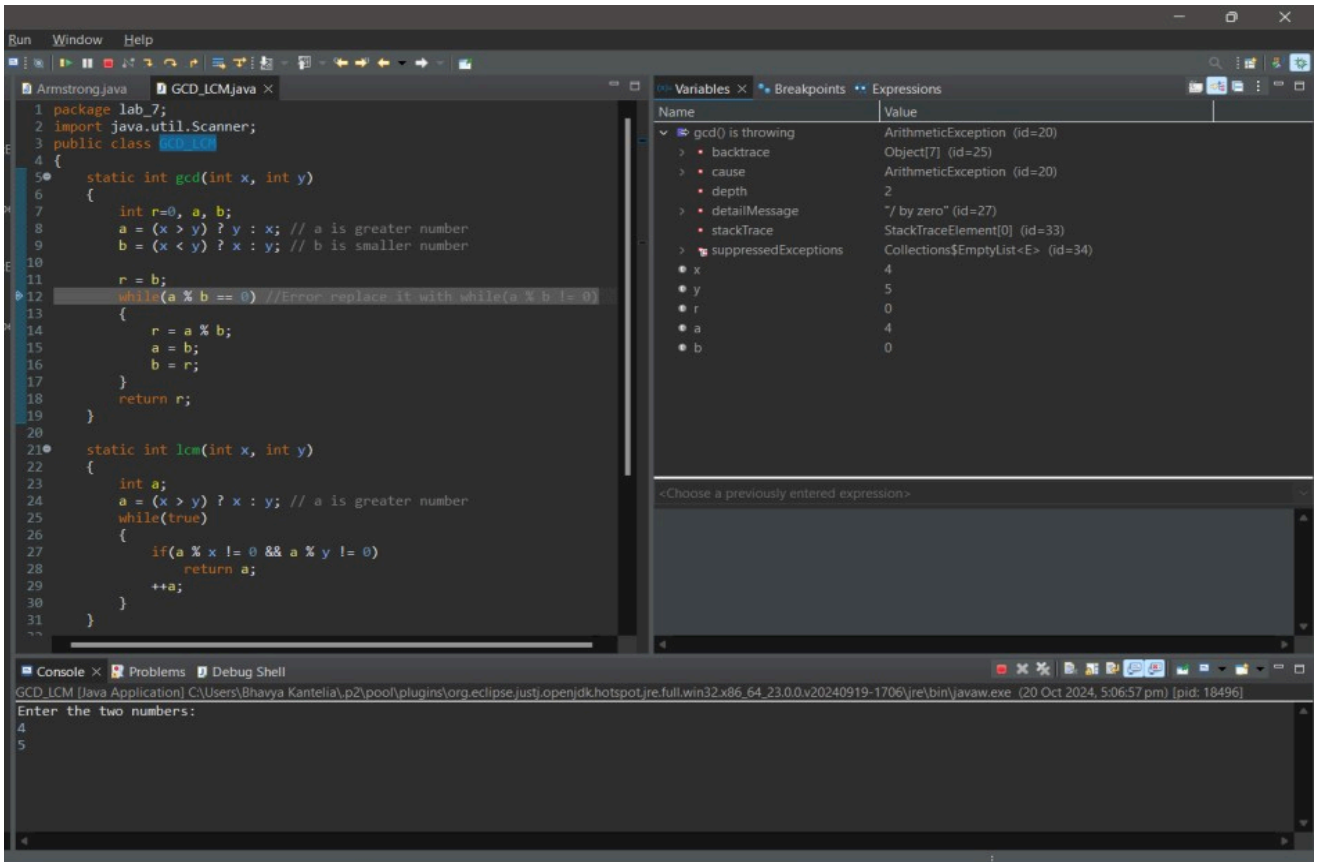
Lab Session-7
Program Inspection, Debugging and Static Analysis
(ID-202201054)

Part: Code debugging

- Debugging is the process of localizing, analyzing, and removing suspected errors in the code (Java code given in the .zip file)

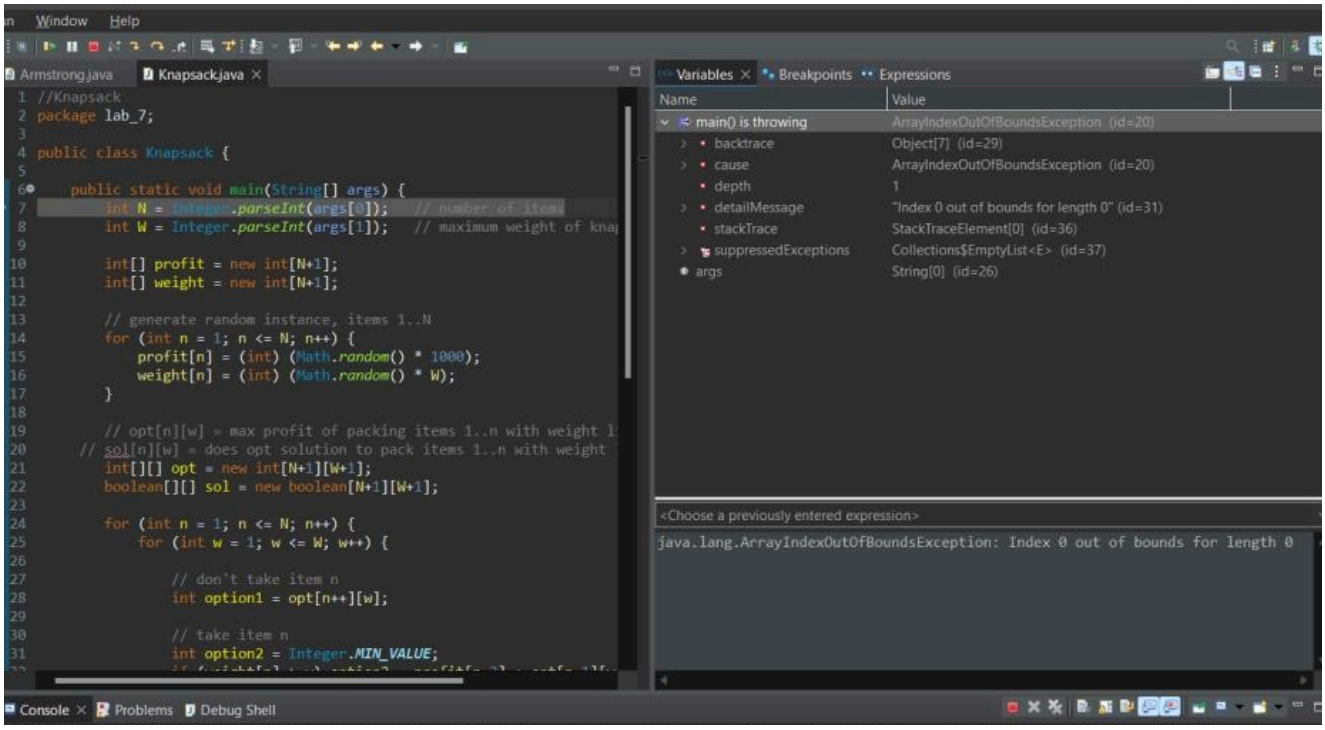
Instructions (Use Eclipse/Netbeans IDE, GDB Debugger)

1. GCD-LCM:



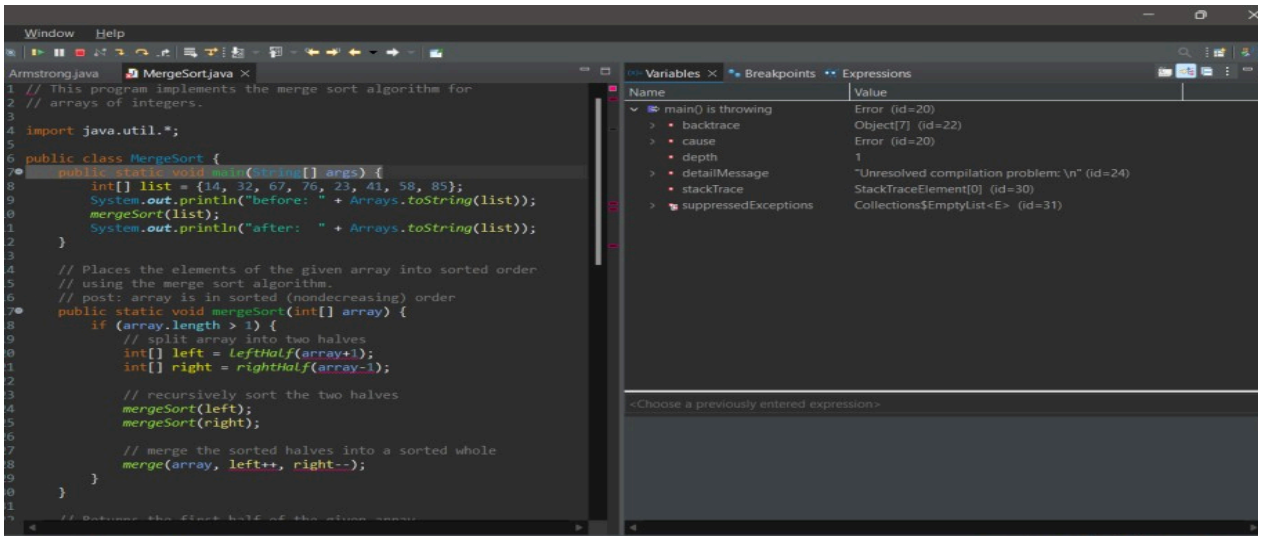
- Here the while loop should be a%b!=0 instead of a%b==0 and thus it throws Arithmetic Expression error.

2.KnapSack:



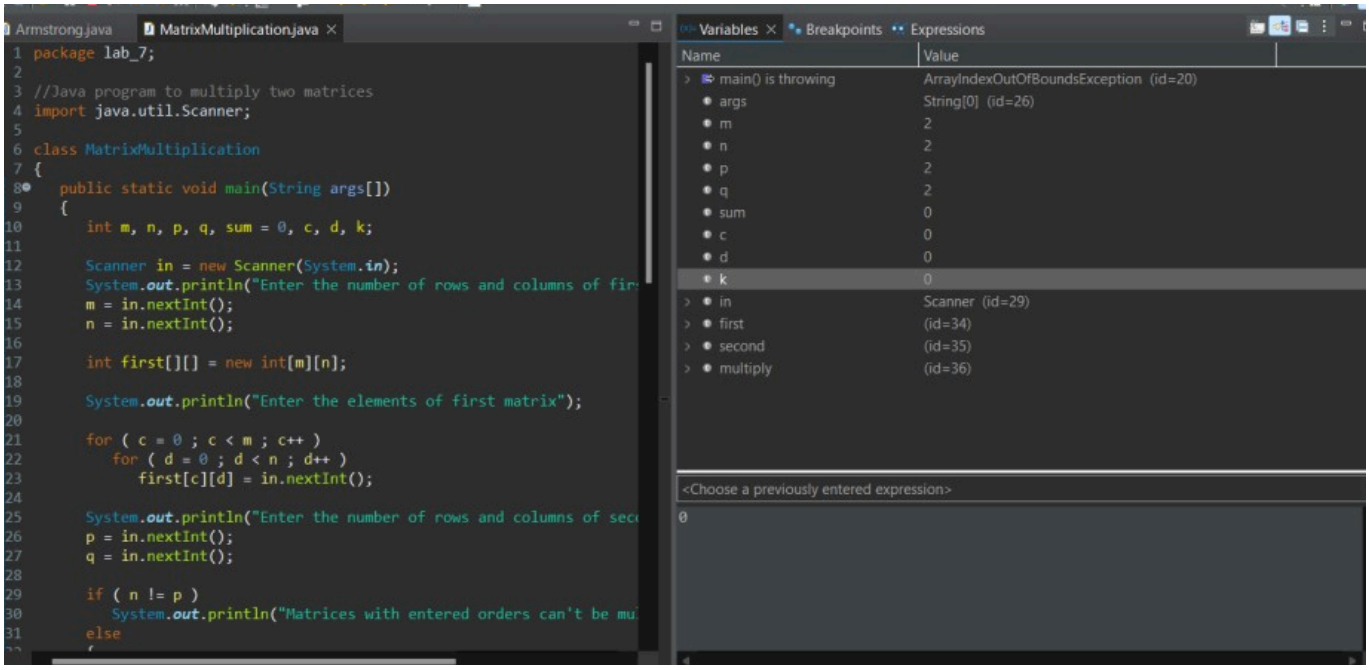
- Error here is an incorrect index update of n-1 which should be n++ which causes main out of bound error.

3. MergeSort:



- int[] left = leftHalf(array+1); should be int[] left = leftHalf(array);. The operation array+1 is invalid because you cannot add an integer to an array reference. int[] right = rightHalf(array-1); should be int[] right = rightHalf(array);. The operation array-1 is also invalid for the same reason. merge(array, left++, right--); should just be merge(array, left, right);. Increment (++) and decrement (--) operators are not needed here, as you're passing the entire array.

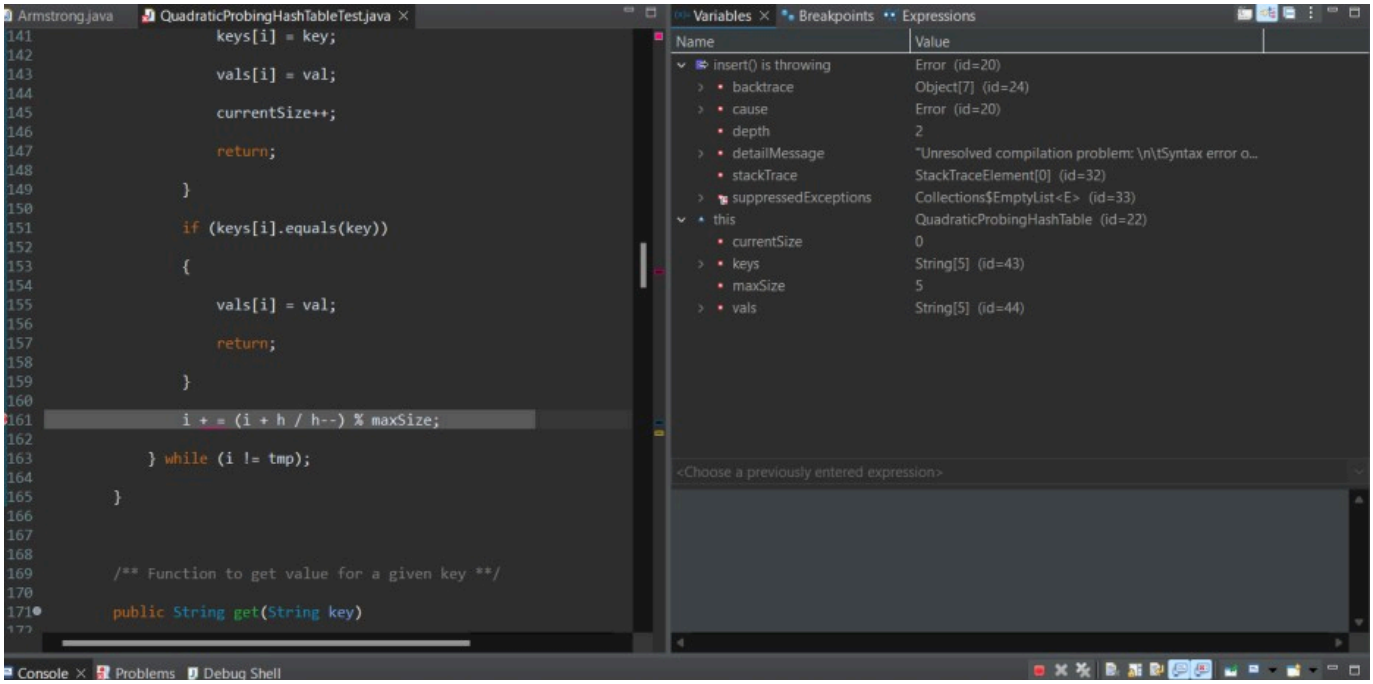
4. MatrixMultiplication:



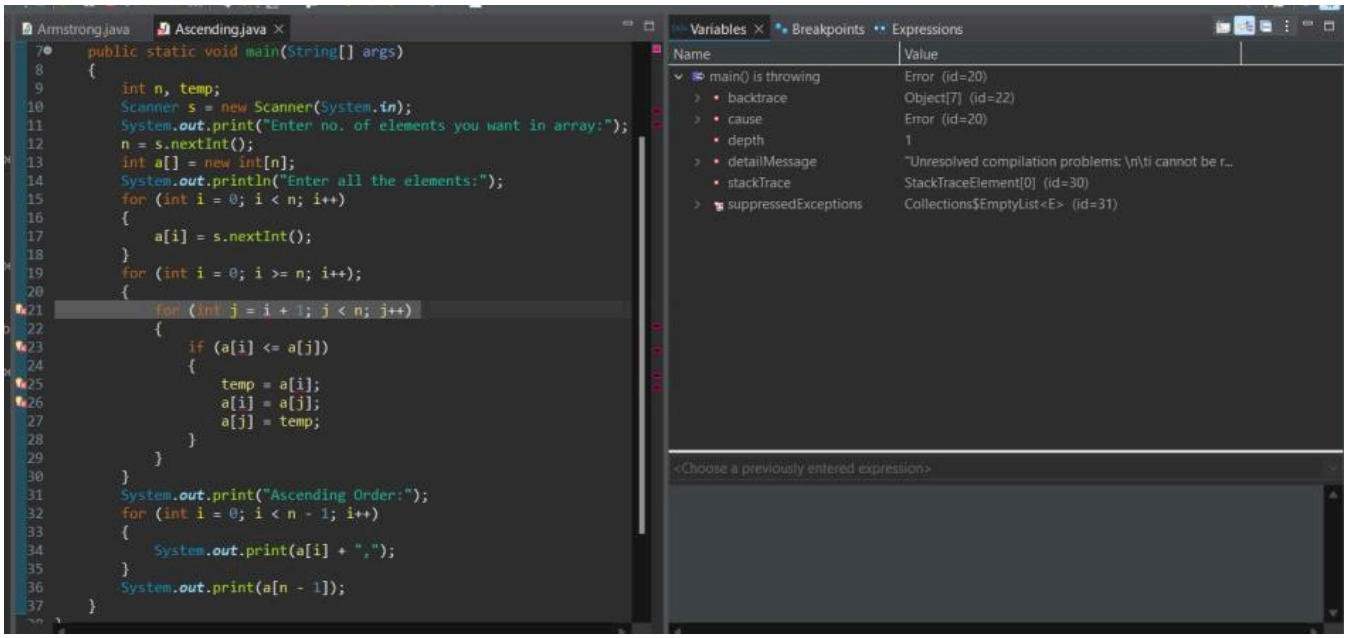
- In the loop where you are multiplying the matrices, you have incorrect indices for both matrices. first[c-1][c-k] and second[k-1][k-d] are wrong because you're trying to access invalid indices with negative values or incorrect references. The correct indices should be first[c][k] and second[k][d] because you need to multiply the corresponding elements of row c of the first matrix with column d of the second matrix.

5. QuadraticProbing:

- i + = (i + h / h--) % maxSize; is invalid syntax. It should be i = (i + h * h++) % maxSize;. The += operator should be properly placed, and the arithmetic operation should use * for quadratic probing, not /.



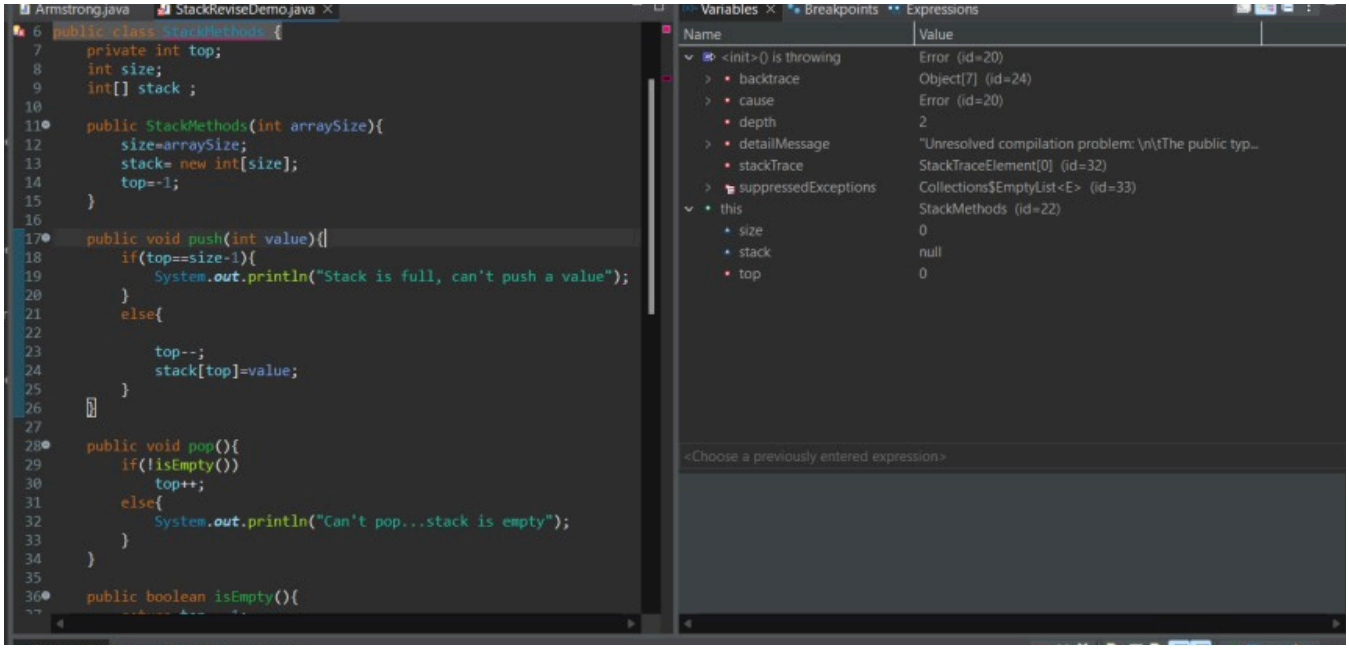
6. Ascending:



- The class name Ascending _Order has a space, which is not allowed in Java. The space should be removed or replaced with an underscore (_) if you want to separate words. The condition for (int i = 0; i >= n; i++); is incorrect because i >= n means the loop will never run, and there is an unnecessary semicolon (;) at the end of the loop. The correct condition

should be i < n. In the inner if condition, you are checking if (a[i] <= a[j]), which will sort the array in descending order.You should change it to if (a[i] > a[j]) to sort the array in ascending order. The final loop prints the array elements separated by commas but incorrectly leaves a trailing comma.

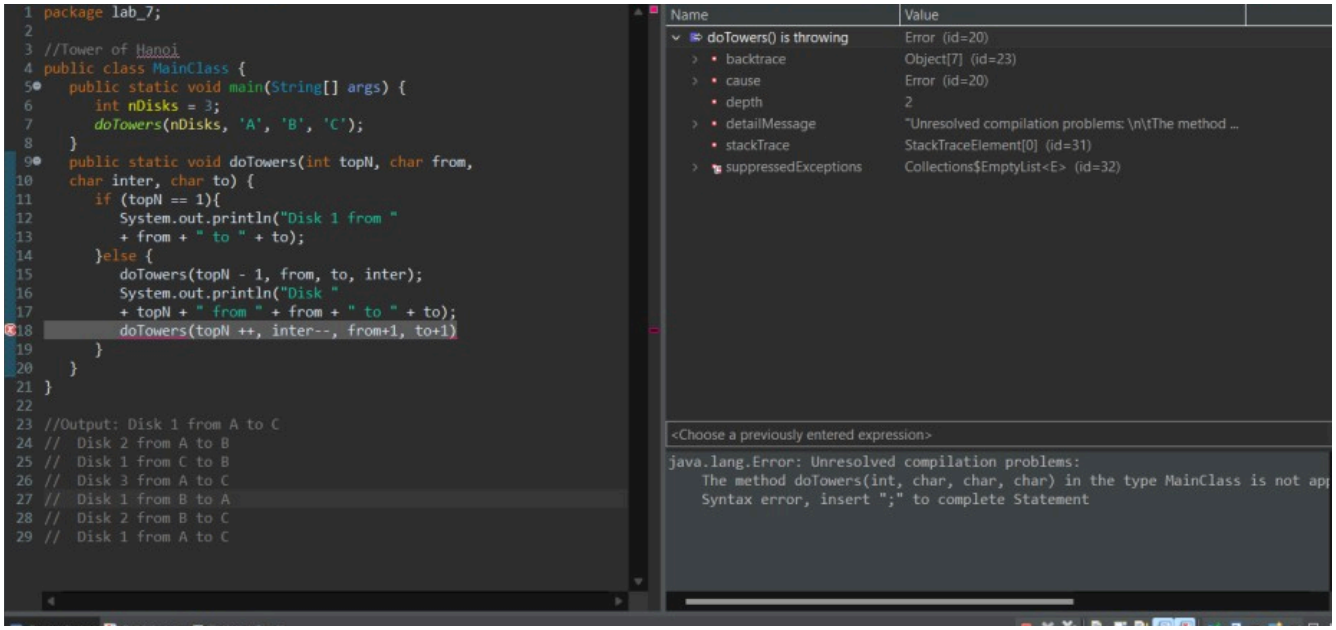
7. Stack:



- In the push method, top-- is used, which decrements top, but it should be top++ to increment the position for inserting a new value. In the display method, the condition for(int i=0;i>top;i++) is incorrect, as it will never execute. The condition should be i <= top to display all elements from index 0 to top. In the pop method, it only increments the top but doesn't actually remove the element or return it.For a correct stack implementation, you should return the popped value, and it should also decrement the top pointer

8. Tower Of Hanoi:

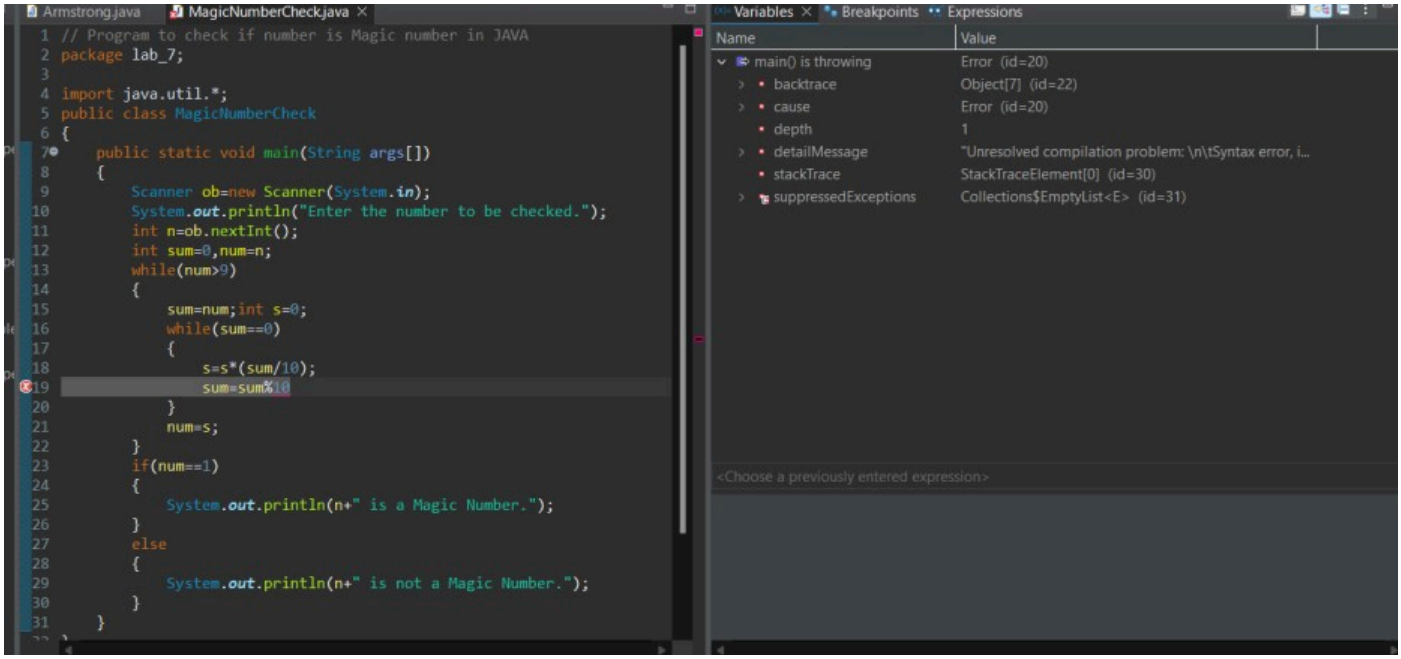
- The expressions topN++, inter--, from + 1, and to + 1 are incorrect in the context of the recursive calls. The parameters should be passed unchanged (no increment or decrement) to maintain the correct behavior of the algorithm. The topN decrement in the recursive calls should be topN - 1, not topN ++.



```
package lab_7;

//Tower of Hanoi
public class MainClass {
   public static void main(String[] args) {
      int nDisks = 3;
      doTowers(nDisks, 'A', 'B', 'C');
   }
   public static void doTowers(int topN, char from,
   char inter, char to) {
      if (topN == 1){
         System.out.println("Disk 1 from "
         + from + " to " + to);
      }else {
         doTowers(topN - 1, from, to, inter);
         System.out.println("Disk "
         + topN + " from " + from + " to " + to);
         doTowers(topN ++, inter--, from+1, to+1)
      }
   }
}

//Output: Disk 1 from A to C
//  Disk 2 from A to B
//  Disk 1 from C to B
//  Disk 3 from A to C
//  Disk 1 from B to A
//  Disk 2 from B to C
//  Disk 1 from A to C
```

```
Name                    Value
doTowers() is throwing  Error (id=20)
  backtrace             Object[7] (id=23)
  cause                 Error (id=20)
  depth                 2
  detailMessage         "Unresolved compilation problems: \n\tThe method ...
  stackTrace            StackTraceElement[0] (id=31)
  suppressedExceptions  Collections$EmptyList<E> (id=32)

<Choose a previously entered expression>
java.lang.Error: Unresolved compilation problems:
    The method doTowers(int, char, char, char) in the type MainClass is not ap
    Syntax error, insert ";" to complete Statement
```

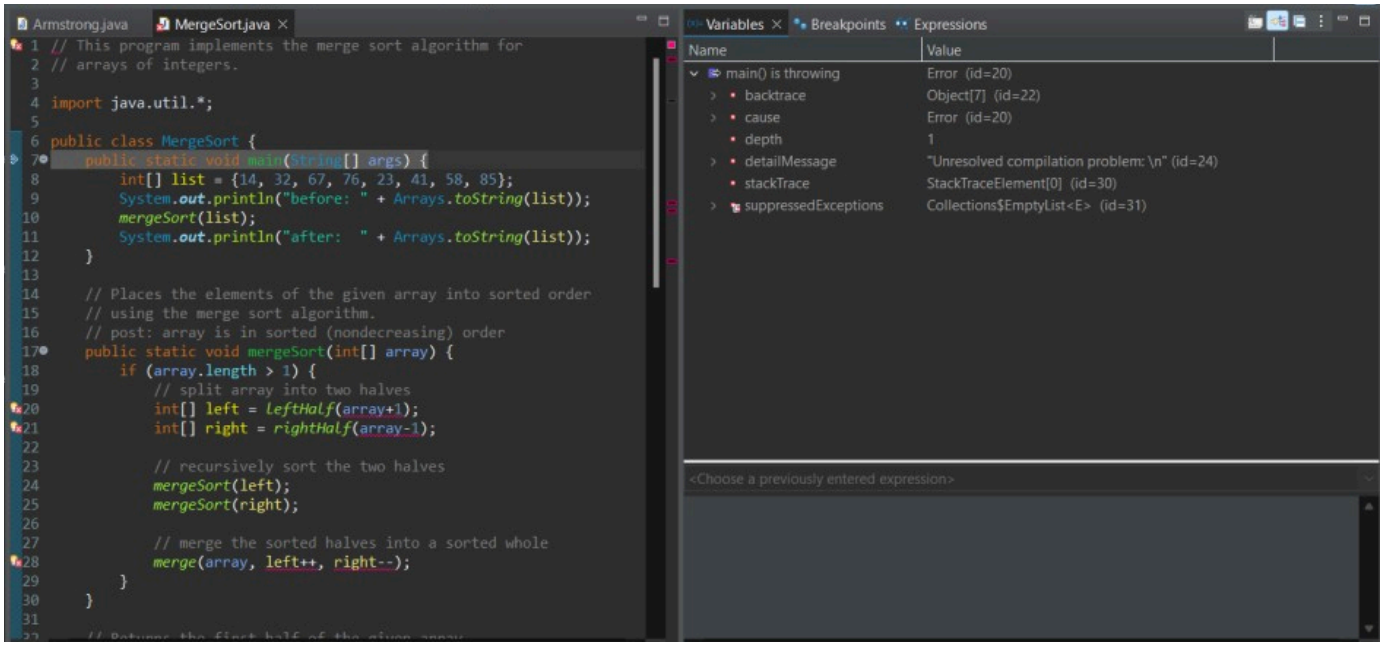
9. Magic Number Check:



```
// Program to check if number is Magic number in JAVA
package lab_7;

import java.util.*;
public class MagicNumberCheck
{
    public static void main(String args[])
    {
        Scanner ob=new Scanner(System.in);
        System.out.println("Enter the number to be checked.");
        int n=ob.nextInt();
        int sum=0,num=n;
        while(num>9)
        {
            sum=num;int s=0;
            while(sum==0)
            {
                s=s*(sum/10);
                sum=sum%10
            }
            num=s;
        }
        if(num==1)
        {
            System.out.println(n+" is a Magic Number.");
        }
        else
        {
            System.out.println(n+" is not a Magic Number.");
        }
    }
}
```

```
Name                    Value
main() is throwing      Error (id=20)
  backtrace             Object[7] (id=22)
  cause                 Error (id=20)
  depth                 1
  detailMessage         "Unresolved compilation problem: \n\tSyntax error, i...
  stackTrace            StackTraceElement[0] (id=30)
  suppressedExceptions  Collections$EmptyList<E> (id=31)

<Choose a previously entered expression>
```

- The condition while(sum == 0) is incorrect. You want to sum the digits of the number, so the condition should be while(sum != 0).
- Missing a semicolon (;) in sum=sum%10.



Part: Program Inspection:

the inspection of 2000 Lines of Code in pieces of 200. For each segment, there are category wise erroneous lines of code.

Category A: Data Reference Errors

- **Uninitialized Variables:**
  Variables such as `name`, `gender`, `age`, and `phone_no` are declared but may not always be initialized before they are referenced. Using these variables before assigning values can cause errors.
- **Array Bounds:**
  Arrays like `char specialization[100];` and `char name[100];` lack explicit bounds checking. This could result in buffer overflow if the array limit is exceeded.

Category B: Data Declaration Errors

- **Implicit Declarations:**
  Variables like `adhaar` and `identification_id` should be explicitly declared and assigned with the correct data types before use.
- **Array Initialization:**
  Arrays like `char specialization[100];` and `char gender[100];` should be initialized to prevent issues with uninitialized values.

Category C: Computation Errors

- **Mixed-mode Computations:**
  Strings like `phone_no` and `adhaar` are treated as numbers. Since these represent numeric strings, they should be handled as strings to avoid errors in numeric calculations.

Category E: Control-Flow Errors

- **Infinite Loops with `goto`:**
  Using `goto` statements for Aadhaar and mobile number validation (e.g., `goto C;`) is risky and can cause infinite loops. Using a `while` loop with proper exit conditions would be a safer alternative.

Category F: Interface Errors

- **Parameter Mismatch:**
  Functions like `add_doctor()` or `display_doctor_data()` should have matching parameters between the function declaration and the function call.

Category G: Input/Output Errors

- **File Handling:**
  Ensure that files such as `Doctor_Data.dat` are opened before use and closed properly after usage to avoid file access errors. Currently, there is no exception handling for file operations, which could cause runtime issues.

Control-Flow Suggestion: The `goto` statements used for Aadhaar and mobile number validation can lead to inefficient flow and hard-to-debug errors. Replacing them with loops would improve the structure and readability.

Second 200 Lines Review:

Category A: Data Reference Errors

- **File Handling:**
  Files like `Doctor_Data.dat` and `Patient_Data.dat` are used without proper error handling, which could lead to issues like file not found or

access errors. Ensure proper mechanisms for file handling are implemented to avoid crashes.

Category B: Data Declaration Errors

- **Strings and Arrays:**
  Variables like `name[100]`, `specialization[100]`, and `gender[10]` may face buffer overflow issues if input exceeds the defined length. It's important to handle such cases carefully.

Category C: Computation Errors

- **Vaccine Stock Calculation:**
  The function `display_vaccine_stock()` calculates the total vaccines across different centers without checking for negative values or overflow. Handling such cases will prevent incorrect calculations.

Category E: Control-Flow Errors

- **Repetitive Use of `goto`:**
  Functions such as `add_doctor()` and `add_patient_data()` use multiple `goto` statements for validation (e.g., Aadhaar or mobile number). Replacing these with proper loops like `while` or `do-while` will improve the readability and control flow of the code.

Category F: Interface Errors

- **Incorrect Data Type Comparisons:**
  In the `search_doctor_data()` function, comparisons between strings like `identification_id` and `sidentification_id` use `.compare()`. Careful management of string comparisons is necessary to avoid potential errors.

Category G: Input/Output Errors

- **Missing File Closing:**
  Files opened in functions like `search_center()` and `display_vaccine_stock()` should be closed properly after reading data to prevent memory leaks or file lock issues.

Third 200 Lines Review:

Category A: Data Reference Errors

- **File Handling:**
  In functions like `add_vaccine_stock()` and `display_vaccine_stock()`, ensure proper error checking when opening files (e.g., `center1.txt`, `center2.txt`). Verify that files open correctly before performing any operations.

Category B: Data Declaration Errors

- **Inconsistent Data Types:**
  Variables like `adhaar` and `phone_no` are expected to be numeric strings, but they are inconsistently handled across functions. Ensure they are treated consistently as strings, not as integers.

Category C: Computation Errors

- **Vaccine Stock Summation:**
  In `display_vaccine_stock()`, ensure all vaccine stock variables are initialized before summing them to avoid errors caused by negative values or uninitialized variables.

Category E: Control-Flow Errors

- **Use of `goto`:**
  Functions like `search_doctor_data()` and `add_doctor()` continue to use `goto` statements, which complicate logic. Replacing these with loops (e.g., `while` or `for`) would enhance code readability and prevent infinite loops.

Category F: Interface Errors

- **Parameter Mismatch:**
  Functions like `search_by_aadhar()` should ensure that the parameters, such as `adhaar`, are passed consistently and correctly throughout the code.

Category G: Input/Output Errors

- **File Access Without Proper Closing:**
  Files like `Doctor_Data.dat` are opened without being closed in some branches. Always ensure that files are properly closed after use to prevent resource leakage.

Fourth 200 Lines Review:

Category A: Data Reference Errors

- **Uninitialized Variables:**
  In functions like `update_patient_data()`, `show_patient_data()`, and `applied_vaccine()`, variables like `maadhaar` and file streams should be explicitly initialized to avoid using uninitialized data.

Category B: Data Declaration Errors

- **Array Length Issues:**
  Character arrays such as `sgender[10]` and `adhaar[12]` are at risk of buffer overflows. Validate input length against the array size to avoid exceeding limits.

Category C: Computation Errors

- **Vaccine Doses:**
  In `update_patient_data()`, directly incrementing the `dose` with `dose++` may lead to invalid counts if not properly validated. Ensure that dose values remain within valid ranges.

Category E: Control-Flow Errors

- **Improper Use of `goto`:**
  Functions like `search_doctor_data()` and `add_patient_data()` rely on `goto` statements for control flow, which makes the code difficult to maintain. Replacing these with loops will improve readability and logic control.

Category F: Interface Errors

- **Incorrect String Comparisons:**
  String comparisons, like in `search_by_aadhar()`, use direct comparisons such as `adhaar.compare(sadhaar)`, which may not

always handle all cases correctly. Ensure that string matching is done consistently and correctly.

Category G: Input/Output Errors

- **File Handling Issues:**
  Functions like `add_patient_data()` open files like `Patient_Data.dat` and `Doctor_Data.dat` without proper error handling after opening. Lack of error handling could lead to runtime issues if the file operation fails.

Fifth 200 Lines Review:

Category A: Data Reference Errors

- **Uninitialized Variables:**
  In functions like `update_patient_data()` and `search_doctor_data()`, variables such as `maadhaar` should be explicitly initialized to prevent the use of uninitialized data.

Category B: Data Declaration Errors

- **Array Boundaries:**
  Arrays like `sgender[10]` are susceptible to buffer overflows if input exceeds the defined limit. Make sure that input length is validated to prevent such issues.

Category C: Computation Errors

- **Patient Dose Incrementation:**
  In `update_patient_data()`, the `dose` is incremented directly with `dose++` without any validation. This can cause incorrect dose counts if limits are not enforced.

Category E: Control-Flow Errors

- **Repetitive Use of `goto`:**
  Functions like `search_doctor_data()` and `add_doctor()` continue to use multiple `goto` statements, complicating the control flow. Use loop structures like `while` or `for` for better readability and maintenance.

Category F: Interface Errors

- **Parameter Mismatch:**
  Functions like `search_by_aadhar()` should ensure that parameters are passed consistently and in the expected format across all subroutines.

Category G: Input/Output Errors

- **File Handling:**
  Files such as `Patient_Data.dat` and `Doctor_Data.dat` are opened but sometimes not closed correctly in certain parts of the code. This could lead to resource leakage. Adding proper exception handling and closing mechanisms will prevent such issues.

Final 1000 Lines Review:

Category A: Data Reference Errors

- **File Handling:**
  Files such as `center1.txt`, `center2.txt`, and `center3.txt` are used in functions like `add_vaccine_stock()` and `display_vaccine_stock()` without proper error handling. It's important to implement error-handling mechanisms in case the files fail to open or encounter access issues.

Category B: Data Declaration Errors

- **Data Initialization:**
  Variables like `sum_vaccine_c1`, `sum_vaccine_c2`, and `sum_vaccine_c3`, used for displaying vaccine stock, should be explicitly initialized to avoid unintended behavior caused by uninitialized variables.

Category C: Computation Errors

- **Vaccine Stock Calculation:**
  In functions such as `add_vaccine_stock()`, ensure that stock values are always positive and properly validated. Incorrect handling of negative or invalid values can cause errors, particularly during operations like subtraction in `display_vaccine_stock()`.

Category E: Control-Flow Errors

- **Excessive Use of `goto` Statements:**
  Functions like `add_doctor()` and `add_patient_data()` contain many `goto` statements, which complicate the control flow. These statements should be replaced with loop structures like `while` or `for` to improve code readability and maintainability.

Category G: Input/Output Errors

- **Inconsistent File Closing:**
  In multiple sections of the code, files are opened without being consistently closed in all branches of execution. It's crucial to ensure that every file opened is properly closed after use to prevent resource leaks and potential memory issues.