IT – 314
Software Engineering

Lab – 07 :

Program Inspection, Debugging and Static Analysis

Student Name - ID :

Dhruvin Akhaja - 202201172

Goal : For each of this code fragment given in ZIP File identify the following,

I. PROGRAM INSPECTION:

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

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

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

4. Is the program inspection technique is worth applicable?

II. CODE DEBUGGING:

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

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?

3. Submit your complete executable code?

• Armstrong Number :

I. PROGRAM INSPECTION:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified

- **Logical Error:** The code contains a logical error in the calculation of the Armstrong number. The variable `remainder` is computed using `num / 10`, which incorrectly extracts the leftmost digit instead of the rightmost digit. The correct operation should be `num % 10` to get the last digit. Additionally, the code updates `num` incorrectly; it should be `num = num / 10` to remove the last digit after processing it.

2. Effective Category

- **Code Review**: A code review is more effective in this context because it allows for a thorough examination of the logic and calculations used in the program. Reviewers can spot logical inconsistencies and suggest better approaches to problem-solving, enhancing overall code quality.

3. Unidentifiable Errors

- **Runtime Errors:** The program inspection cannot catch runtime errors, particularly those related to input validation. If the user provides a non-numeric argument or does not supply any arguments, the program will throw a `NumberFormatException`, causing it to fail at runtime rather than being caught during inspection.

4. Worth Applicability

- **Yes:** Program inspection is indeed a worthwhile technique. It helps identify logical and syntactical errors during the review process, which can lead to better coding practices and reduced bugs. However, it should be complemented with runtime testing and input validation to ensure robustness, as static inspection alone cannot identify all potential issues that may arise during execution.

II. CODE DEBUGGING:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified:

- **Logical Error in Remainder Calculation**: The line `remainder = num / 10` incorrectly calculates the remainder. It should be `remainder = num % 10` to extract the last digit of `num`.
- **Logical Error in Updating `num`** : The line `num = num % 10` incorrectly updates `num`. It should be `num = num / 10` to remove the last digit of `num`.

2. Number of Breakpoints Needed:

- One at the `remainder = num / 10;` line to inspect the remainder calculation and fix it.
- One at the `num = num % 10;` line to ensure that `num` is updated correctly.

2.a. Steps to Fix Errors:

- Fix the remainder calculation by replacing `remainder = num / 10` with `remainder = num % 10`.

- Correct the update of num by changing `num = num % 10` to `num = num / 10` so that it divides num by 10, effectively removing the last digit.

3. Complete Executable Code:

```
// Armstrong Number Program

class Armstrong {

    public static void main(String args[]) {

        int num = Integer.parseInt(args[0]);

        int n = num; // store original number

        int check = 0, remainder;


        while (num > 0) {

            remainder = num % 10; // fix: calculate remainder correctly

            check = check + (int) Math.pow(remainder, 3);

            num = num / 10; // fix: update num correctly by dividing it by 10

        }


        if (check == n)

            System.out.println(n + " is an Armstrong Number");

        else

            System.out.println(n + " is not an Armstrong Number");

    }

}
```

Input:

153

Output:

```
153 is an Armstrong Number
```

- **GCD and LCM :**

I. PROGRAM INSPECTION:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified

- **Logical Error in GCD Calculation:** In the `gcd` method, the condition in the while loop is incorrect. It should check for `while(a % b != 0)` instead of `while(a % b == 0)`. This error will cause the loop to terminate immediately if `a` is divisible by `b`, which is incorrect.
- **Inefficient LCM Calculation:** The `lcm` method incorrectly checks both conditions with `if(a % x != 0 && a % y != 0)`. It should use `if(a % x == 0 && a % y == 0)` to find the LCM properly. This would cause the loop to return incorrect values.

2. Effective Category

- **Code Review:** A code review would be the most effective category for this program. It allows for an in-depth examination of the logic, ensuring the algorithm's correctness and efficiency. Reviewers can also suggest optimizations for both GCD and LCM calculations.

3. Unidentifiable Errors

- **Runtime Errors:** Program inspection is unlikely to identify runtime errors such as input validation issues. For instance, if a user inputs a non-integer value or a negative number, the program will throw an exception or may behave unexpectedly.

4. Worth Applicability

- **Yes:** The program inspection technique is worth applying. It can effectively identify logical and syntactical issues early in the development process, promoting better coding practices. However, it should be complemented by additional testing, including unit tests and runtime validation, to ensure comprehensive coverage of potential issues.

II. CODE DEBUGGING:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified:

- **Error in the GCD Calculation Logic**: The while(a % b == 0) condition is incorrect. It should be while(a % b != 0) because the loop should continue until the remainder is 0, which indicates that b is the greatest common divisor.
- **Error in LCM Logic**: The if(a % x != 0 && a % y != 0) condition is incorrect. It should be if(a % x == 0 && a % y == 0) because the LCM is the smallest multiple of a that is divisible by both x and y.

2. Number of Breakpoints Needed:

- One at the while(a % b == 0) line to fix the incorrect condition.
- One at the if(a % x != 0 && a % y != 0) line to correct the LCM condition.

2.a. Steps to Fix Errors:

- Fix the while loop condition in the gcd method to while(a % b != 0) so that the loop continues until the remainder is zero.
- Fix the if condition in the lcm method to if(a % x == 0 && a % y == 0) to correctly identify the least common multiple.

3. Complete Executable Code:

```java
import java.util.Scanner;

public class GCD_LCM {
    static int gcd(int x, int y) {
        int r = 0, a, b;
        a = (x > y) ? x : y; // a is greater number
        b = (x < y) ? x : y; // b is smaller number

        r = b;
        while (a % b != 0) { // fix: change == to !=
            r = a % b;
            a = b;
            b = r;
        }
        return r;
    }

    static int lcm(int x, int y) {
        int a;
        a = (x > y) ? x : y; // a is greater number
        while (true) {
            if (a % x == 0 && a % y == 0) // fix: change != to ==
                return a;
            ++a;
        }
    }

    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the two numbers: ");
        int x = input.nextInt();
        int y = input.nextInt();

        System.out.println("The GCD of two numbers is: " + gcd(x, y));
        System.out.println("The LCM of two numbers is: " + lcm(x, y));
        input.close();
    }
}
```

Input:

4 5

Output:

The GCD of two numbers is: 1
The LCM of two numbers is: 20

• Knapsack :

I. PROGRAM INSPECTION:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified

- **Logical Error in Option Calculations:**
    - In the `opt` array updates, the statement `int option1 = opt[n++][w];` incorrectly modifies `n`, leading to skipped iterations and incorrect indexing. It should use `int option1 = opt[n][w];` to access the current item without incrementing `n`.
    - In the line `if (weight[n] > w) option2 = profit[n-2] + opt[n-1][w-weight[n]];`, the index `n-2` is incorrect. It should be `profit[n]` to consider the profit of the current item.
- **Initialization Issues**: The arrays `profit` and `weight` have an additional element, but the random generation loop starts from index `1`. The program will not handle the case for the item at index `0`, which can lead to unexpected behaviour or incorrect results.

2. Effective Category

- **Code Review**: A code review would be the most effective category for this program. It allows reviewers to catch logical errors and ensure the correct implementation of the dynamic programming approach for the knapsack problem.

3. Unidentifiable Errors

- **Logical Errors in Algorithm:** Some logical errors related to dynamic programming logic might not be caught through simple inspection, especially if they do not result in compile-time errors or immediate runtime exceptions. For instance, incorrect profit or weight calculations can produce misleading outputs without causing errors.

4. Worth Applicability

- **Yes**: The program inspection technique is worth applying. It helps identify potential issues early in the development process, improving code quality and correctness. However, it should be complemented with rigorous testing (e.g., unit tests, edge cases) to ensure the algorithm works as expected across various scenarios.

II. CODE DEBUGGING:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified:

- **Increment Issue:** In the line `int option1 = opt[n++][w];`, `n++` increments `n` before the next iteration, which is incorrect. It should be `n - 1` instead of `n++` to avoid skipping items.
- **Index Calculation in Profit Calculation:** In `option2 = profit[n-2] + opt[n-1][w-weight[n]];`, `profit[n-2]` incorrectly references previous profits. It should be `profit[n]` instead.
- **Bounds Issue:** The line `if (weight[n] > w)` should be `if (weight[n] <= w)` to ensure the item is taken only if its weight is within the current weight limit `w`.

2. Number of Breakpoints Needed:

- One to fix the increment issue on `opt[n++][w]` (should be `opt[n - 1][w]`).
- One to fix the incorrect array index on `profit[n-2]`.

3. Steps to Fix Errors:

- Change `opt[n++][w]` to `opt[n-1][w]` to avoid skipping an index while comparing.
- Change `profit[n-2]` to `profit[n]` to correctly calculate the profit.
- Adjust the condition in the `if` clause for weight to `weight[n] <= w`.

4. Complete Executable Code:

```
public class Knapsack {


    public static void main(String[] args) {

        int N = Integer.parseInt(args[0]);   // number of items

        int W = Integer.parseInt(args[1]);   // maximum weight of knapsack


        int[] profit = new int[N+1];

        int[] weight = new int[N+1];


        // generate random instance, items 1..N

        for (int n = 1; n <= N; n++) {

            profit[n] = (int) (Math.random() * 1000);

            weight[n] = (int) (Math.random() * W);

        }


        // opt[n][w] = max profit of packing items 1..n with weight limit w

        // sol[n][w] = does opt solution to pack items 1..n with weight limit w include item n?

        int[][] opt = new int[N+1][W+1];

        boolean[][] sol = new boolean[N+1][W+1];


        for (int n = 1; n <= N; n++) {

            for (int w = 1; w <= W; w++) {


                // don't take item n

                int option1 = opt[n-1][w]; // Fix: use n-1 instead of n++


                // take item n
                int option2 = Integer.MIN_VALUE;
                if (weight[n] <= w) // Fix: weight must be less than or equal to w
                    option2 = profit[n] + opt[n-1][w-weight[n]]; // Fix: profit[n],
not profit[n-2]

                // select better of two options
                opt[n][w] = Math.max(option1, option2);
                sol[n][w] = (option2 > option1);
            }
        }

        // determine which items to take
        boolean[] take = new boolean[N+1];
        for (int n = N, w = W; n > 0; n--) {
            if (sol[n][w]) {
                take[n] = true;
                w = w - weight[n];
            } else {
                take[n] = false;
            }
        }

        // print results
        System.out.println("item" + "\t" + "profit" + "\t" + "weight" + "\t" +
"take");
        for (int n = 1; n <= N; n++) {
            System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t" +
take[n]);
        }
    }
```

```
}
```

Input:

```
6 2000
```

Output (example):

```
item    profit    weight    take

1       336       784       false

2       674       1583      false

3       763       392       true

4       544       1136      true

5       14        1258      false

6       738       306       true
```

- Magic number :

I. PROGRAM INSPECTION:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified

- **Logical Errors:**
  - In the line `while(sum==0)`, it should be `while(sum>0)` to iterate over the digits of the number correctly.
  - The calculation `s=s*(sum/10);` should instead be `s += sum % 10;` to accumulate the sum of the digits correctly.
  - The line `sum=sum%10` should be `sum = sum / 10;` to reduce the number correctly for the next iteration.
- **Missing Semicolon:** There is a missing semicolon (`;`) at the end of the line `sum=sum%10` which will lead to a compilation error.

2. Effective Category

- **Code Review:** A code review is more effective for this program to catch logical mistakes and ensure the correctness of the algorithm, especially in the nested loops.

3. Unidentifiable Errors

- **Logic Errors:** Some logic errors, such as incorrect handling of number summation, may not be evident without thorough testing. For instance, the algorithm could produce incorrect results for inputs that require multiple iterations of summing digits.

4. Worth Applicability

- **Yes:** The program inspection technique is worth applying. It helps in identifying syntax and logical errors early in the development process, improving the overall quality of the code. Combining this with testing will lead to more robust code.

II. CODE DEBUGGING:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified:

- Incorrect Loop Condition: The inner `while` loop has a faulty condition `while(sum==0)`. It should check if `sum > 0` to sum up the digits of the number.
- Incorrect Summation Logic: The calculation `s=s*(sum/10);` is incorrect. It should sum the digits of the number, not multiply them.
- Missing Semicolon: In the line `sum=sum%10`, a semicolon is missing.

2. Number of Breakpoints Needed:

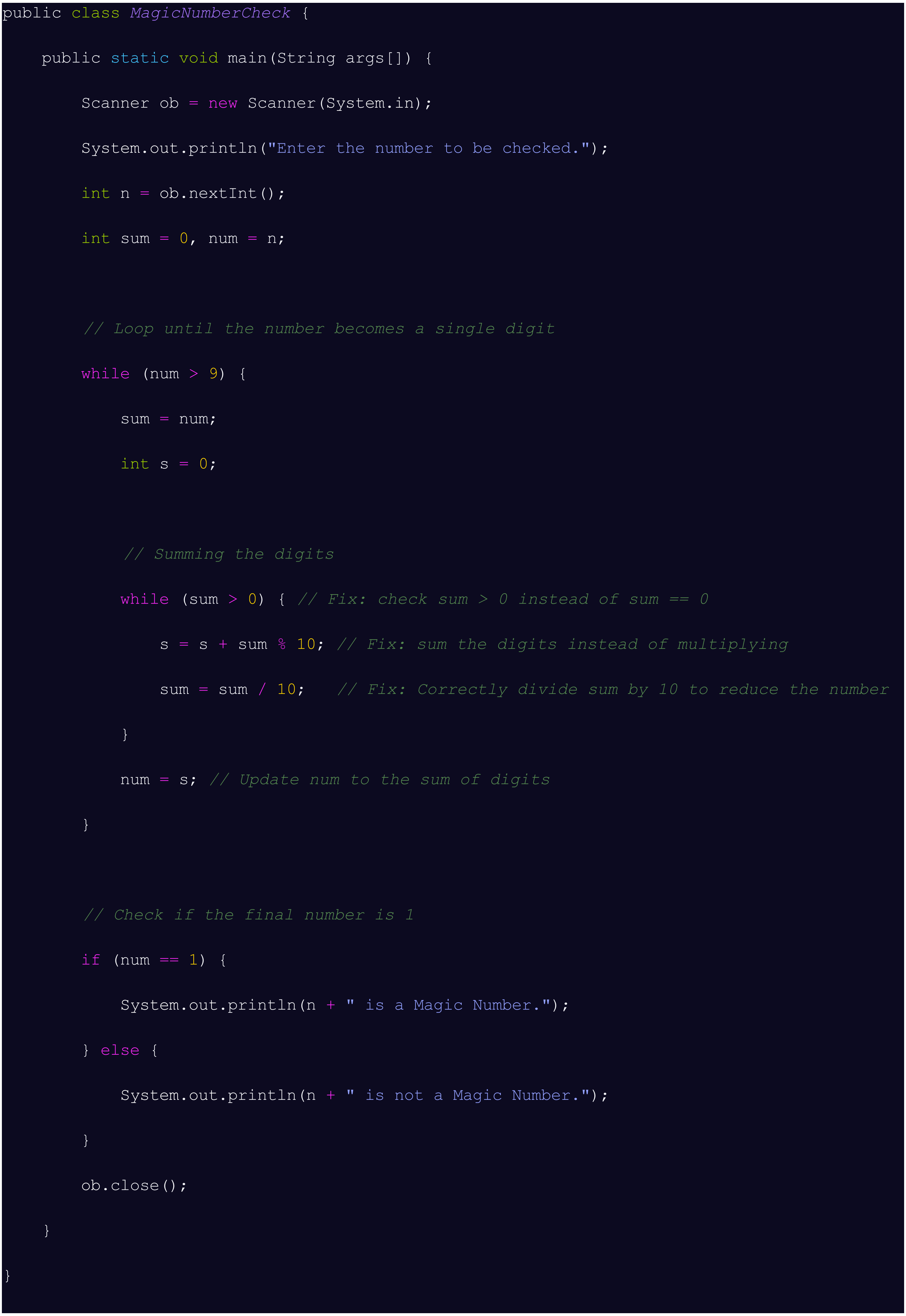
- To fix the loop condition (`while(sum > 0)` instead of `while(sum == 0)`).
- To fix the digit summation logic (`s = s + sum % 10` instead of `s = s * (sum / 10)`).

3. Steps to Fix Errors:

- Change the inner `while(sum == 0)` condition to `while(sum > 0)` to correctly sum the digits.
- Modify `s = s * (sum / 10)` to `s = s + sum % 10` to sum the digits rather than multiplying them.
- Add a semicolon in the line `sum = sum % 10`.

4. Complete Executable Code:

```
import java.util.*;
```



```java
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;

        // Loop until the number becomes a single digit
        while (num > 9) {
            sum = num;
            int s = 0;

            // Summing the digits
            while (sum > 0) { // Fix: check sum > 0 instead of sum == 0
                s = s + sum % 10; // Fix: sum the digits instead of multiplying
                sum = sum / 10;    // Fix: Correctly divide sum by 10 to reduce the number
            }
            num = s; // Update num to the sum of digits
        }

        // Check if the final number is 1
        if (num == 1) {
            System.out.println(n + " is a Magic Number.");
        } else {
            System.out.println(n + " is not a Magic Number.");
        }
        ob.close();
    }
}
```

Input:

```
Enter the number to be checked: 119
```

Output:

```
119 is a Magic Number.
```

Input:

```
Enter the number to be checked: 199
```

Output:

```
199 is not a Magic Number.
```

- ## Merge Sort :

I. PROGRAM INSPECTION:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified

- **Array Manipulation Errors:**
  - In the `mergeSort` method, `int[] left = leftHalf(array+1);` should pass `array` directly. You can't add 1 to an array in this context. It should be `int[] left = leftHalf(array);` and similarly for the right half, it should be `int[] right = rightHalf(array);`.
- **Incorrect Use of Increment/Decrement:**
  - The statement `merge(array, left++, right--);` is incorrect because `left++` and `right--` do not properly reference the arrays. It should simply be `merge(array, left, right);`.
- **Incorrect Array Length Handling in `merge`:**
  - The `merge` function should handle the length of the result array correctly. It currently assumes the result is the same size as `left` or `right` instead of being the full array.

2. Effective Category

- **Code Review:** This category is effective because the logical flow of array handling, recursive calls, and merging could be better analyzed by another programmer to spot errors early.

3. Unidentifiable Errors

- **Runtime Errors:** Some errors might only surface during execution, such as index out-of-bounds exceptions due to incorrect array lengths when merging or splitting arrays.

4. Worth Applicability

- **Yes:** Program inspection techniques are definitely worth applying here, as they can catch issues that may not be immediately visible during initial development. Thorough inspection, especially in recursive algorithms like merge sort, can help ensure the correctness of the implementation.

II. CODE DEBUGGING:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified:

- Incorrect Array Passing: In `mergeSort`, arrays are incorrectly passed. Instead of using `array + 1` and `array - 1`, we should pass the array itself for splitting into left and right halves.
- Incorrect Merging Indices: When merging, there are incorrect array access operations like `left++` and `right--`. These are invalid, as they modify the array reference and will cause runtime errors. They should be passed as `left` and `right` directly.

2. Number of Breakpoints Needed:

- To fix the incorrect splitting of the array (`array + 1`, `array - 1`).
- To correct the merging process (`left++`, `right--`).

3. Steps to Fix Errors:

- Modify the array splitting logic to correctly pass the array to `leftHalf` and `rightHalf` without using `array + 1` or `array - 1`.
- Pass the `left` and `right` arrays as they are in the `merge` function instead of modifying them with `left++` and `right--`.

4. Complete Executable Code:

```
import java.util.*;



public class MergeSort {
```

```java
    public static void main(String[] args) {
        int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
        System.out.println("before: " + Arrays.toString(list));
        mergeSort(list);
        System.out.println("after:  " + Arrays.toString(list));
    }

    // Places the elements of the given array into sorted order
    // using the merge sort algorithm.
    public static void mergeSort(int[] array) {
        if (array.length > 1) {
            // split array into two halves
            int[] left = leftHalf(array);
            int[] right = rightHalf(array);

            // recursively sort the two halves
            mergeSort(left);
            mergeSort(right);

            // merge the sorted halves into a sorted whole
            merge(array, left, right);
        }
    }

    // Returns the first half of the given array.
    public static int[] leftHalf(int[] array) {
        int size1 = array.length / 2;
        int[] left = new int[size1];
        for (int i = 0; i < size1; i++) {
            left[i] = array[i];
        }
        return left;
    }

    // Returns the second half of the given array.
    public static int[] rightHalf(int[] array) {
        int size1 = array.length / 2;
        int size2 = array.length - size1;
        int[] right = new int[size2];
        for (int i = 0; i < size2; i++) {
            right[i] = array[i + size1];
        }
        return right;
    }

    // Merges the given left and right arrays into the given
    // result array.
    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0;   // index into left array
        int i2 = 0;   // index into right array

        for (int i = 0; i < result.length; i++) {
            if (i2 >= right.length || (i1 < left.length &&
                    left[i1] <= right[i2])) {
                result[i] = left[i1];    // take from left
                i1++;
            } else {
                result[i] = right[i2];   // take from right
                i2++;
            }
        }
    }
}
```

Input:

```
before: [14, 32, 67, 76, 23, 41, 58, 85]
```

Output:

```
after:  [14, 23, 32, 41, 58, 67, 76, 85]
```

- ## Multiply matrices :

I. PROGRAM INSPECTION:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified

- **Incorrect Indexing in Matrix Multiplication:**
  - In the multiplication loop, the expressions `first[c-1][c-k]` and `second[k-1][k-d]` are incorrect. They should be `first[c][k]` and `second[k][d]` respectively. The `-1` offsets lead to out-of-bounds access in the arrays.
- **Sum Initialization:**
  - The variable `sum` is not properly initialized within the loop. It's declared outside the loops, which could lead to incorrect accumulation of values from previous iterations.
- **Matrix Size Validation:**
  - The prompt for the second matrix's dimensions incorrectly states "Enter the number of rows and columns of first matrix" instead of "Enter the number of rows and columns of second matrix".
- **Redundant Code:**
  - The `sum` variable can be declared inside the innermost loop instead of being a class member to prevent potential misuse in the outer loops.

2. Effective Category

- **Code Review:** This method is effective for catching logical errors and ensuring the correctness of indexing and mathematical operations, especially in nested loops.

3. Unidentifiable Errors

- **Logical Errors:** Some logical errors may not be identified until runtime, particularly if incorrect assumptions are made about the input matrices' sizes and the results generated, as the program could compile without any issues.

4. Worth Applicability

- **Yes:** The program inspection technique is worthwhile as it helps identify logical flaws and improves the overall robustness of the code. Matrix operations are prone to errors related to indexing and calculations, so thorough inspection can significantly enhance code quality.

II. CODE DEBUGGING:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified:

- Index Out of Bounds: The incorrect indexing occurs in the multiplication process. Specifically:
  - `first[c-1][c-k]` and `second[k-1][k-d]` will result in index out of bounds errors when `c` or `k` is 0.
- Incorrect Logic in Multiplication: The logic for accessing elements in both matrices during multiplication is incorrect. You should multiply `first[c][k]` by `second[k][d]` for matrix multiplication.

2. Number of Breakpoints Needed:

- Two Breakpoints:
  1. To fix the array index out-of-bounds errors during the matrix multiplication.
  2. To correct the matrix multiplication logic, ensuring correct element multiplication and accumulation.

3. Steps to Fix Errors:

- Replace `first[c-1][c-k]` with `first[c][k]` and `second[k-1][k-d]` with `second[k][d]` to avoid out-of-bounds errors.
- Ensure that the matrix multiplication logic iterates over `k` correctly and accumulates the product in `sum` for each element of the result matrix.

4. Complete Executable Code:

```java
import java.util.Scanner;

class MatrixMultiplication {
   public static void main(String args[]) {
      int m, n, p, q, sum = 0, c, d, k;

      Scanner in = new Scanner(System.in);
      System.out.println("Enter the number of rows and columns of the first matrix:");
      m = in.nextInt();
      n = in.nextInt();

      int first[][] = new int[m][n];

      System.out.println("Enter the elements of the first matrix:");

      for (c = 0; c < m; c++)
         for (d = 0; d < n; d++)
            first[c][d] = in.nextInt();

      System.out.println("Enter the number of rows and columns of the second matrix:");
      p = in.nextInt();
      q = in.nextInt();

      if (n != p) {
         System.out.println("Matrices with entered orders can't be multiplied with each other.");
      } else {
         int second[][] = new int[p][q];
         int multiply[][] = new int[m][q];
```

```
            System.out.println("Enter the elements of the second matrix:");

            for (c = 0; c < p; c++)
                for (d = 0; d < q; d++)
                    second[c][d] = in.nextInt();

            // Multiplication of the matrices
            for (c = 0; c < m; c++) {
                for (d = 0; d < q; d++) {
                    sum = 0;
                    for (k = 0; k < n; k++) {
                        sum += first[c][k] * second[k][d];
                    }
                    multiply[c][d] = sum;
                }
            }

            System.out.println("Product of the entered matrices:");

            for (c = 0; c < m; c++) {
                for (d = 0; d < q; d++)
                    System.out.print(multiply[c][d] + "\t");
                System.out.println();
            }
        }
        in.close();
    }
}
```

Input:

```
Enter the number of rows and columns of the first matrix:

2 2

Enter the elements of the first matrix:

1 2 3 4

Enter the number of rows and columns of the second matrix:

2 2

Enter the elements of the second matrix:

1 0 1 0
```

Output:

```
Product of entered matrices:

3   0

7   0
```

- ## Quadratic Probing :

I. PROGRAM INSPECTION:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified

- **Incorrect Syntax for Incrementing in Loop:**
  - In several places, expressions like `i + = (i + h / h--) % maxSize;` and `i = (i + h * h++) % maxSize;` have incorrect syntax. It should be `i += (h * h++);` for proper incrementation.
- **Potential Infinite Loop:**
  - The `do-while` loops in `insert`, `get`, and `remove` methods may lead to infinite loops if the table is full and there's no valid spot for a new key or while searching for a key that doesn't exist.
- **Improper Initialization of h:**
  - The variable h is not correctly reset after it is used, which could lead to incorrect index calculations.
- **Memory Leak:**

    - The `makeEmpty()` method resets the references to the keys and values without properly clearing the memory. In Java, this isn't typically a major issue due to garbage collection, but it's a bad practice.
- **Typo in Comments:**
    - The comments in the code have minor typos like "Fucntion" and "maxSizeake", which should be "Function" and "maxSize", respectively.

2. Effective Category

- **Code Review:** This approach helps in identifying logical and syntactical errors, ensuring that the algorithm behaves as expected, especially with the complex logic of hash table operations.

3. Unidentifiable Errors

- **Hash Collisions Handling:** Logical issues may arise with collision resolution if the hashing function and probing strategy do not distribute keys uniformly across the table. Such behaviour might not be apparent until the program is run with diverse input data.

4. Worth Applicability

- **Yes:** The program inspection is worthwhile, especially for data structure implementations like hash tables. Identifying logical and syntactical errors early can prevent runtime exceptions and improve the reliability of the code, especially in complex methods involving hashing and probing.

II. CODE DEBUGGING:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified

- The `hash` method might return a negative index, leading to ArrayIndexOutOfBoundsException.
- In the `insert` method, the line `i + = (i + h / h--) % maxSize;` has a syntax error; it should be `i = (i + h * h) % maxSize;`.
- The rehashing logic in the `remove` method does not correctly adjust the index when probing.
- The method `get` does not properly account for when `i` becomes out of bounds.

2. Breakpoints Needed

- Before the `hash` method call in the `insert` and `get` methods to check the value returned.

- After the `insert` method to check if the key-value pairs are inserted correctly.
- After the `get` method to check if it retrieves the correct value.
- Before and after the `remove` method to verify correct **deletion.**

3. Steps to Fix Errors

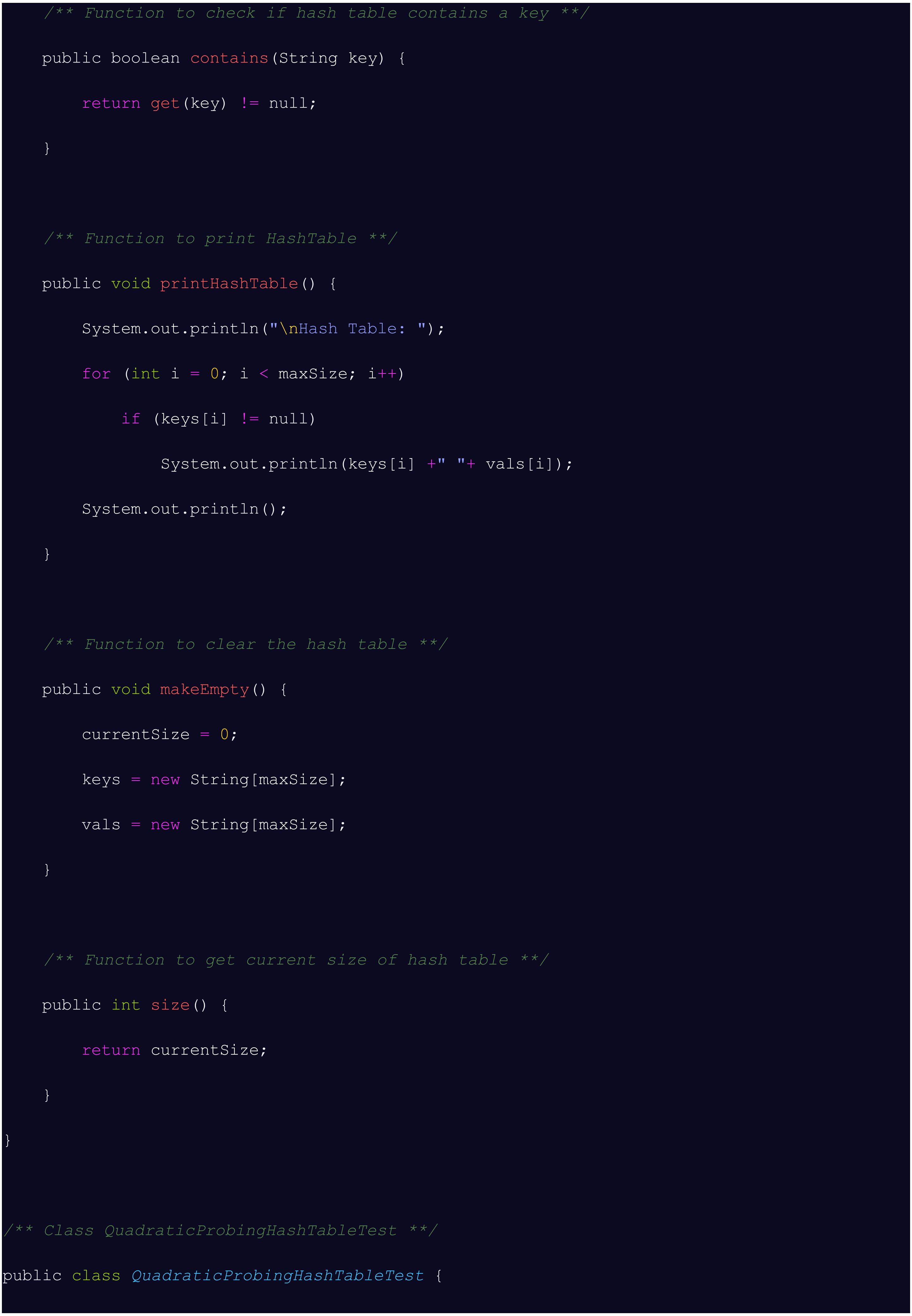
- Modify the `hash` method to ensure it returns a positive index using `Math.abs()`.
- Correct the syntax error in the `insert` method from + = to =.
- Adjust the probing logic in the `remove` method to correctly use `i = (i + h * h) % maxSize`.
- Ensure that all array accesses are within bounds in the `get` method.

4. Complete Executable Code

```
import java.util.Scanner;

/** Class QuadraticProbingHashTable **/
class QuadraticProbingHashTable {
    private int currentSize, maxSize;
    private String[] keys;
    private String[] vals;

    /** Constructor **/
    public QuadraticProbingHashTable(int capacity) {
        currentSize = 0;
        maxSize = capacity;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

    /** Function to get hash code of a given key **/
    private int hash(String key) {
        return Math.abs(key.hashCode()) % maxSize; // Ensure positive index
```

```java
    }

    /** Function to insert key-value pair **/
    public void insert(String key, String val) {
        if (key == null || val == null) return;  // Prevent null keys or values

        int tmp = hash(key);
        int i = tmp, h = 1;
        do {
            if (keys[i] == null) {
                keys[i] = key;
                vals[i] = val;
                currentSize++;
                return;
            }
            if (keys[i].equals(key)) {
                vals[i] = val;
                return;
            }
            i = (i + h * h) % maxSize; // Corrected probing logic
            h++;  // Increment h for the next probing
        } while (i != tmp);
    }

    /** Function to get value for a given key **/
    public String get(String key) {
        int i = hash(key), h = 1;
        while (keys[i] != null) {
            if (keys[i].equals(key))
```

```
                return vals[i];
            i = (i + h * h) % maxSize; // Corrected probing logic
            h++;  // Increment h for the next probing
        }
        return null;
    }

    /** Function to remove key and its value **/
    public void remove(String key) {
        if (!contains(key))
            return;

        /** Find position key and delete **/
        int i = hash(key), h = 1;
        while (!key.equals(keys[i]))
            i = (i + h * h) % maxSize;

        keys[i] = vals[i] = null;

        /** Rehash all keys **/
        for (i = (i + h * h) % maxSize; keys[i] != null; i = (i + h * h) % maxSize) {
            String tmp1 = keys[i], tmp2 = vals[i];
            keys[i] = vals[i] = null;
            currentSize--;
            insert(tmp1, tmp2);
        }
        currentSize--;
    }
```



```
    /** Function to check if hash table contains a key **/
    public boolean contains(String key) {
        return get(key) != null;
    }

    /** Function to print HashTable **/
    public void printHashTable() {
        System.out.println("\nHash Table: ");
        for (int i = 0; i < maxSize; i++)
            if (keys[i] != null)
                System.out.println(keys[i] +" "+ vals[i]);
        System.out.println();
    }

    /** Function to clear the hash table **/
    public void makeEmpty() {
        currentSize = 0;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

    /** Function to get current size of hash table **/
    public int size() {
        return currentSize;
    }
}

/** Class QuadraticProbingHashTableTest **/
public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Hash Table Test\n\n");
        System.out.print("Enter size: ");

        /** Create object of QuadraticProbingHashTable **/
        QuadraticProbingHashTable qpht = new QuadraticProbingHashTable(scan.nextInt());

        char ch;
        /**  Perform QuadraticProbingHashTable operations  **/
        do {
            System.out.println("\nHash Table Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. remove");
            System.out.println("3. get");
            System.out.println("4. clear");
            System.out.println("5. size");

            int choice = scan.nextInt();
            switch (choice) {
                case 1:
                    System.out.println("Enter key and value");
                    qpht.insert(scan.next(), scan.next());
                    break;
                case 2:
                    System.out.println("Enter key");
                    qpht.remove(scan.next());
                    break;
                case 3:
                    System.out.println("Enter key");
                    System.out.println("Value = " + qpht.get(scan.next()));
                    break;
                case 4:
                    qpht.makeEmpty();
                    System.out.println("Hash Table Cleared\n");
                    break;
                case 5:
                    System.out.println("Size = " + qpht.size());
                    break;
                default:
                    System.out.println("Invalid choice.");
            }
            qpht.printHashTable();
            System.out.println("Do you want to continue? (Y/N)");
            ch = scan.next().charAt(0);
        } while (ch == 'Y' || ch == 'y');

        scan.close();
    }
}
```

Input:

```
Hash Table Test

Enter size: 5

Hash Table Operations

1. insert
```

```
2. remove
3. get
4. clear
5. size
1
Enter key and value
c computer
1
Enter key and value
d desktop
1
Enter key and value
h harddrive
2
Enter key
d
3
Enter key
h
4
```

Output:

```
Value = desktop
Hash Table:
c computer
h harddrive
```

```
Do you want to continue? (Y/N)

y

Hash Table Operations

1. insert

2. remove

3. get

4. clear

5. size

5

Size = 2

Hash Table:

c computer

h harddrive

Do you want to continue? (Y/N)

n
```

- **Sorting the array :**

I. PROGRAM INSPECTION:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified

- **Incorrect Loop Condition:**
  - The first `for` loop uses `i >= n`, which should be `i < n`. This causes the loop to never execute, and hence no sorting occurs.
- **Unnecessary Semicolon:**
  - There is an unnecessary semicolon at the end of the first `for` loop (`for (int i = 0; i >= n; i++);`), which effectively terminates the loop prematurely.
- **Improper Sorting Logic:**

- The comparison logic in the inner loop is incorrect for sorting in ascending order. It should be `if (a[i] > a[j])` instead of `if (a[i] <= a[j])`.
- **Output Formatting:**
  - The output for printing the sorted array could lead to an extra comma after the last element if the size of the array is greater than one.

2. Effective Category

- **Code Review:** This category is effective as it helps in understanding the algorithm's correctness and ensuring that the sorting logic is implemented properly.

3. Unidentifiable Errors

- **Algorithm Efficiency:** Potential issues related to the algorithm's time complexity and performance might not be apparent through inspection, especially if the input size is large.

4. Worth Applicability

- **Yes:** Program inspection is worthwhile, especially for algorithms like sorting. Identifying logical errors early can lead to more efficient and correct implementations, and it also enhances code readability and maintainability.

II. CODE DEBUGGING:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Analysis of Errors :

- Class Name Formatting: The class name `Ascending _Order` contains a space, which is not valid in Java. It should be `AscendingOrder`.
- Loop Condition Mistake: The outer loop condition `for (int i = 0; i >= n; i++);` is incorrect. It should be `for (int i = 0; i < n; i++)`.
- Unnecessary Semicolon: The semicolon at the end of the outer loop declaration (`for (int i = 0; i >= n; i++);`) terminates the loop prematurely.
- Sorting Logic Error: The sorting condition `if (a[i] <= a[j])` should be `if (a[i] > a[j])` for ascending order.
- Print Formatting: The output formatting in the final print statement does not correctly display the array.

2. Breakpoints Needed :

- Check the initialization of the loop variable `i`.
- Validate the inner sorting condition.

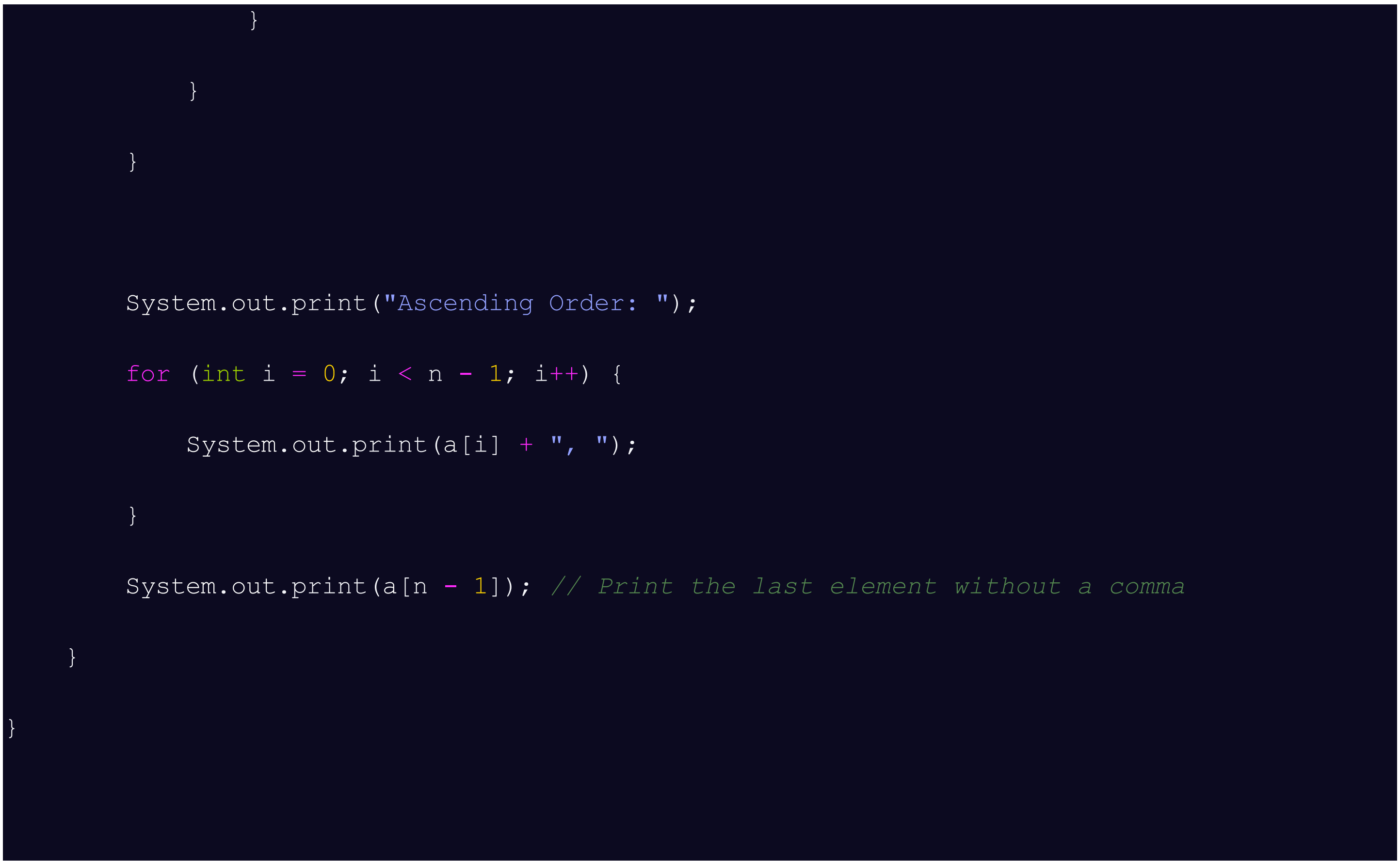
- Review the output formatting logic.

2.a. Steps to Fix the Errors :

- Change the class name to `AscendingOrder`.
- Correct the loop condition from >= to < in the outer loop.
- Remove the unnecessary semicolon at the end of the outer loop declaration.
- Update the sorting condition from <= to > to properly sort in ascending order.
- Fix the output formatting to ensure the correct display of all elements.

3. Complete Executable Code :

```
import java.util.Scanner;

public class AscendingOrder {
    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();
        }

        // Sorting the array in ascending order
        for (int i = 0; i < n; i++) { // Changed from >= to <
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) { // Changed from <= to >
                    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]); // Print the last element without a comma
    }
}
```

Input:

```
Enter no. of elements you want in array: 5

Enter all the elements:

1

12

2

9

7
```

Output:

```
Ascending Order: 1, 2, 7, 9, 12
```

- ## Stack implementation :

I. PROGRAM INSPECTION:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified

- **Incorrect Logic in push Method:**
  - In the push method, top should be incremented (i.e., top++) before assigning the value to stack[top]. The current logic incorrectly decrements top, which leads to an array index out of bounds or overwriting the wrong elements.
- **Incorrect Logic in display Method:**
  - The condition in the for loop (i > top) should be i <= top. As it stands, the loop will never execute, resulting in no output.
- **Pop Logic Needs Adjustment:**
  - The pop method should remove the top element by adjusting top (i.e., top--) instead of incrementing it. This will lead to incorrect behaviour when popping elements.

2. Effective Category

- **Code Review:** This is an effective category for identifying logical errors in stack implementations where the order of operations and index management are critical.

3. Unidentifiable Errors

- **Memory Management:** Potential memory issues such as stack overflow or underflow are not directly identifiable from inspection alone. They require dynamic testing with various input sizes.

4. Worth Applicability

- **Yes:** Program inspection is worthwhile, particularly in data structure implementations. It ensures that the logic for managing the stack's state (push, pop, and display) is correctly implemented, which is crucial for reliable functionality.

II. CODE DEBUGGING:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Analysis of Errors :

- Push Logic Error: The push method incorrectly decreases top before assigning a value, which results in an ArrayIndexOutOfBoundsException. The correct approach is to increment the top before assignment.

- Display Loop Condition Error: The loop in the `display` method uses `i > top`, which will never execute as `i` starts from `0`. It should be `i <= top` to display all elements in the stack.
- Pop Logic Error: In the `pop` method, `top` is incremented when popping an element, but it should decrement it instead to properly remove the element from the stack.
- Missing Return Value for `pop`: The `pop` method does not return the value being popped, which is generally expected in stack implementations.
- Output Formatting: The output format does not clearly separate the elements visually; adjustments can be made for better readability.

2. Breakpoints Needed :

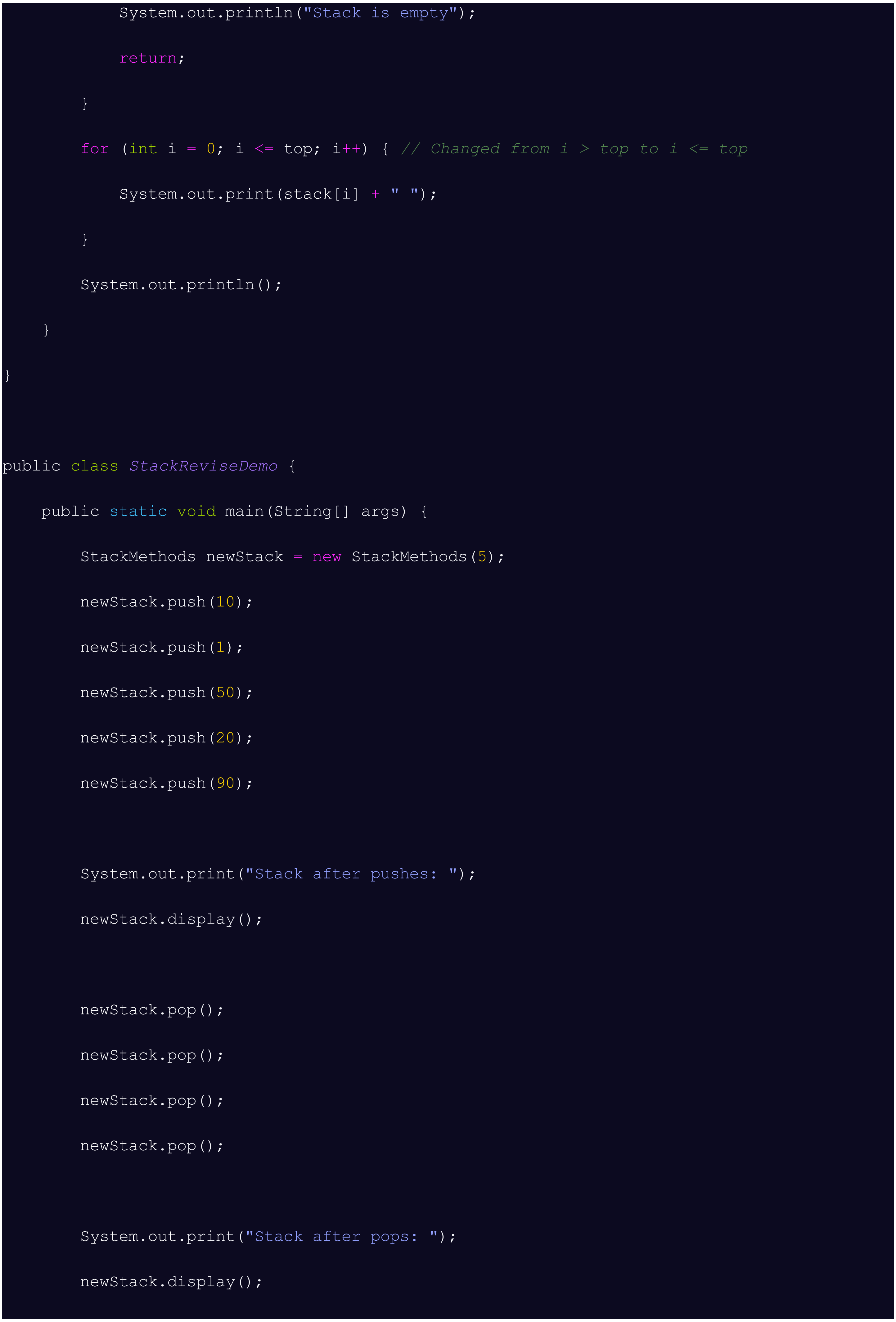
- Check the push method to ensure the value is added correctly.
- Validate the logic in the pop method to ensure it modifies the stack as expected.
- Review the display method for proper iteration over stack elements.

2.a. Steps to Fix the Errors :

- Update the `push` method to increment `top` before assigning the value.
- Change the loop condition in the `display` method from `i > top` to `i <= top`.
- Modify the `pop` method to decrement `top` correctly.
- Return the popped value from the `pop` method.
- Adjust the output formatting in the `display` method for clarity.

3. Complete Executable Code :

```java
// Stack implementation in Java
import java.util.Arrays;

public class StackMethods {
    private int top;
    int size;
    int[] stack;

    public StackMethods(int arraySize) {
        size = arraySize;
        stack = new int[size];
        top = -1;

    }



    public void push(int value) {

        if (top == size - 1) {

            System.out.println("Stack is full, can't push a value");

        } else {

            top++;

            stack[top] = value;

        }

    }



    public Integer pop() {

        if (!isEmpty()) {

            int value = stack[top];

            top--;

            return value;

        } else {

            System.out.println("Can't pop...stack is empty");

            return null; // Return null if stack is empty

        }

    }



    public boolean isEmpty() {

        return top == -1;

    }



    public void display() {

        if (isEmpty()) {
```



```java
            System.out.println("Stack is empty");
            return;
        }
        for (int i = 0; i <= top; i++) { // Changed from i > top to i <= top
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }
}

public class StackReviseDemo {
    public static void main(String[] args) {
        StackMethods newStack = new StackMethods(5);
        newStack.push(10);
        newStack.push(1);
        newStack.push(50);
        newStack.push(20);
        newStack.push(90);

        System.out.print("Stack after pushes: ");
        newStack.display();

        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();

        System.out.print("Stack after pops: ");
        newStack.display();
    }

}
```

Input:

```
No direct input from the user; elements pushed programmatically.
```

Output:

```
Stack after pushes: 10 1 50 20 90

Stack after pops: Stack is empty
```

Tower of Hanoi :

I. PROGRAM INSPECTION:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Errors Identified

- **Incorrect Logic in Recursive Call:**
  - In the recursive calls `doTowers(topN ++, inter--, from+1, to+1)`, the use of ++ and `--` will not behave as intended. The expressions should remain as `topN - 1`, `inter`, `from`, and `to` in the second recursive call.
- **Output Formatting:**
  - The code is expected to generate a sequence of moves for the Tower of Hanoi problem, but the way the recursion is structured currently can lead to infinite recursion or incorrect disk movements due to the improper handling of the parameters.
- **Missing Base Case Handling:**
  - The logic does not correctly handle the case where `topN` is not properly decremented, which can lead to unexpected results.

2. Effective Category

- **Logical Errors:** This is an effective category for identifying errors in recursive algorithms where state management and parameter handling are crucial.

3. Unidentifiable Errors

- **Performance Issues:** Potential performance issues, like stack overflow with large `nDisks`, aren't identifiable from code inspection alone; they require dynamic testing with large values.

4. Worth Applicability

- **Yes:** Program inspection is worthwhile in this context to ensure that the recursive logic is implemented correctly. Given that the Tower of Hanoi is a classic algorithm, accurate implementation is vital for educational purposes and understanding recursion.

II. CODE DEBUGGING:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Analysis of Errors

- Incorrect Increment/Decrement in Recursive Call: The expression `topN++`, `inter--`, `from+1`, and `to+1` in the recursive call are incorrect for the recursive logic. The variables should not be modified this way since they lead to incorrect values being passed to the next recursive calls.
- Improper Handling of Characters: The characters `from`, `to`, and `inter` should not be incremented or decremented, as they are of type `char`. Instead, we should maintain their original values.
- Output Clarity: The output does not clearly separate each move or stack state, but this is more of a formatting issue rather than an error.

2. Breakpoints Needed :

- Check the values being passed to the `doTowers` method to ensure they remain correct for each recursive call.
- Validate the output after each disk move to confirm the order of moves is correct.

2.a. Steps to Fix the Errors :

- Modify the recursive calls in the `doTowers` method to correctly pass the parameters without incrementing or decrementing the characters.
- Ensure the logic for moving disks follows the Tower of Hanoi rules without modifying the parameters incorrectly.

3. Complete Executable Code :

```
// 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); // Move topN-1 disks from source to intermediate

            System.out.println("Disk " + topN + " from " + from + " to " + to); // Move the bottom disk to target

            doTowers(topN - 1, inter, from, to); // Move the disks from intermediate to target

        }

    }

}
```

Input:

No direct user input; the number of disks is defined in the code (nDisks = 3).

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

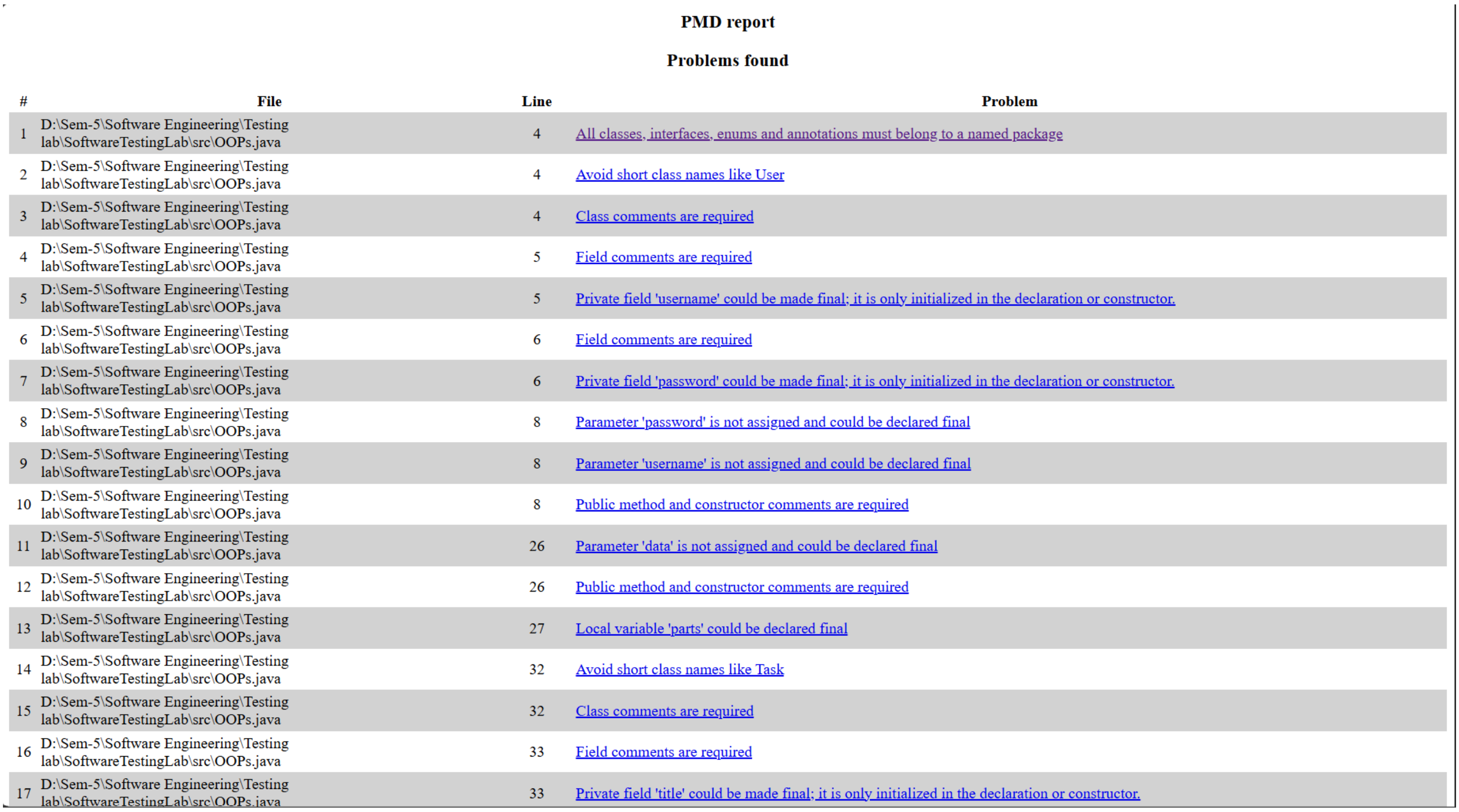
- ## Static Analysis Using PMD :

Code used for Static Analysis :

```
import java.io.*;
import java.util.*;

class User {
    private String username;
    private String password;
    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    @Override
    public String toString() {
        return username + "," + password;
    }

    public static User fromString(String data) {
        String[] parts = data.split(",");
        return new User(parts[0], parts[1]);
    }
}

class Task {
    private String title;
    private String description;
    private boolean completed;

    public Task(String title, String description) {
        this.title = title;
        this.description = description;
        this.completed = false;
    }

    public String getTitle() {
        return title;
    }

    public String getDescription() {
        return description;
    }

    public boolean isCompleted() {
        return completed;
    }

    public void complete() {
        this.completed = true;
    }
    @Override
    public String toString() {
        return title + "," + description + "," + completed;
    }

    public static Task fromString(String data) {
        String[] parts = data.split(",");
        Task task = new Task(parts[0], parts[1]);
        if (parts.length > 2) {
            task.completed = Boolean.parseBoolean(parts[2]);
        }
        return task;
    }
}

class UserManager {
    private List<User> users;
    private final String filename;

    public UserManager(String filename) {
        this.filename = filename;
        this.users = new ArrayList<>();
        loadUsers();
    }

    private void loadUsers() {
        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = br.readLine()) != null) {
                users.add(User.fromString(line));
            }
        } catch (IOException e) {
            System.out.println("Error loading users: " + e.getMessage());
        }
    }

    public void saveUsers() {
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(filename))) {
            for (User user : users) {
                bw.write(user.toString());
                bw.newLine();
            }
        } catch (IOException e) {
            System.out.println("Error saving users: " + e.getMessage());
        }
    }

    public void registerUser(String username, String password) {
        users.add(new User(username, password));
        saveUsers();
        System.out.println("User registered successfully.");
    }

    public boolean authenticateUser(String username, String password) {
        for (User user : users) {
            if (user.getUsername().equals(username) && user.getPassword().equals(password)) {
                return true;
            }
        }
        return false;
    }
}

class TaskManager {
    private List<Task> tasks;
    private final String filename;

    public TaskManager(String filename) {
        this.filename = filename;
        this.tasks = new ArrayList<>();
        loadTasks();
    }

    private void loadTasks() {
        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = br.readLine()) != null) {
                tasks.add(Task.fromString(line));
            }
        } catch (IOException e) {
            System.out.println("Error loading tasks: " + e.getMessage());
        }
    }

    public void saveTasks() {
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(filename))) {
            for (Task task : tasks) {
                bw.write(task.toString());
                bw.newLine();
            }
        } catch (IOException e) {

            System.out.println("Error saving tasks: " + e.getMessage());

        }

    }


    public void addTask(String title, String description) {

        tasks.add(new Task(title, description));

        saveTasks();

        System.out.println("Task added successfully.");

    }


    public void listTasks() {

        if (tasks.isEmpty()) {

            System.out.println("No tasks available.");

            return;

        }

        for (int i = 0; i < tasks.size(); i++) {

            Task task = tasks.get(i);

            String status = task.isCompleted() ? "✔️" : "❌";

            System.out.printf("%d. %s - %s (%s)\n", i + 1, task.getTitle(), status, task.getDescription());

        }

    }


    public void completeTask(int index) {

        if (index < 1 || index > tasks.size()) {

            System.out.println("Invalid task index.");

            return;

        }

        Task task = tasks.get(index - 1);

        task.complete();
        saveTasks();
        System.out.println("Task marked as completed.");
    }

    public void deleteTask(int index) {
        if (index < 1 || index > tasks.size()) {
            System.out.println("Invalid task index.");
            return;
        }
        tasks.remove(index - 1);
        saveTasks();
        System.out.println("Task deleted successfully.");
    }
}

public class TaskManagerApp {
    private static final String USER_FILE = "users.txt";
    private static final String TASK_FILE = "tasks.txt";
    private static UserManager userManager;
    private static TaskManager taskManager;
    private static String currentUser;

    public static void main(String[] args) {
        userManager = new UserManager(USER_FILE);
        taskManager = new TaskManager(TASK_FILE);
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\nTask Manager Application");
            System.out.println("1. Register");
            System.out.println("2. Login");
            System.out.println("3. Exit");

            String choice = scanner.nextLine();

            switch (choice) {
                case "1":
                    registerUser(scanner);
                    break;
                case "2":
                    if (loginUser(scanner)) {
                        userMenu(scanner);
                    }
                    break;
                case "3":
                    System.out.println("Exiting the application.");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid option. Please try again.");
            }
        }
    }

    private static void registerUser(Scanner scanner) {
        System.out.print("Enter username: ");
        String username = scanner.nextLine();
        System.out.print("Enter password: ");
        String password = scanner.nextLine();
        userManager.registerUser(username, password);
    }


    private static boolean loginUser(Scanner scanner) {
        System.out.print("Enter username: ");
        String username = scanner.nextLine();
        System.out.print("Enter password: ");
        String password = scanner.nextLine();

        if (userManager.authenticateUser(username, password)) {
            currentUser = username;
            System.out.println("Login successful. Welcome, " + currentUser + "!");
            return true;
        } else {
            System.out.println("Invalid username or password. Please try again.");
            return false;
        }
    }

    private static void userMenu(Scanner scanner) {
        while (true) {
            System.out.println("\nUser Menu");
            System.out.println("1. Add Task");
            System.out.println("2. List Tasks");
            System.out.println("3. Complete Task");
            System.out.println("4. Delete Task");
            System.out.println("5. Logout");

            String choice = scanner.nextLine();

            switch (choice) {
                case "1":
                    addTask(scanner);
                    break;
                case "2":
                    taskManager.listTasks();
                    break;
                case "3":
                    completeTask(scanner);
                    break;
                case "4":
                    deleteTask(scanner);
                    break;
                case "5":
                    System.out.println("Logging out.");
                    currentUser = null;
                    return;
                default:
                    System.out.println("Invalid option. Please try again.");
            }
        }
    }

    private static void addTask(Scanner scanner) {
        System.out.print("Enter task title: ");
        String title = scanner.nextLine();
        System.out.print("Enter task description: ");
        String description = scanner.nextLine();
        taskManager.addTask(title, description);
    }
```

```
  private static void completeTask(Scanner scanner) {

    taskManager.listTasks();

    System.out.print("Enter task number to complete: ");

    int taskNumber = Integer.parseInt(scanner.nextLine());

    taskManager.completeTask(taskNumber);

  }



  private static void deleteTask(Scanner scanner) {

    taskManager.listTasks();

    System.out.print("Enter task number to delete: ");

    int taskNumber = Integer.parseInt(scanner.nextLine());

    taskManager.deleteTask(taskNumber);

  }

}
```

Output of PMD(HTML) :



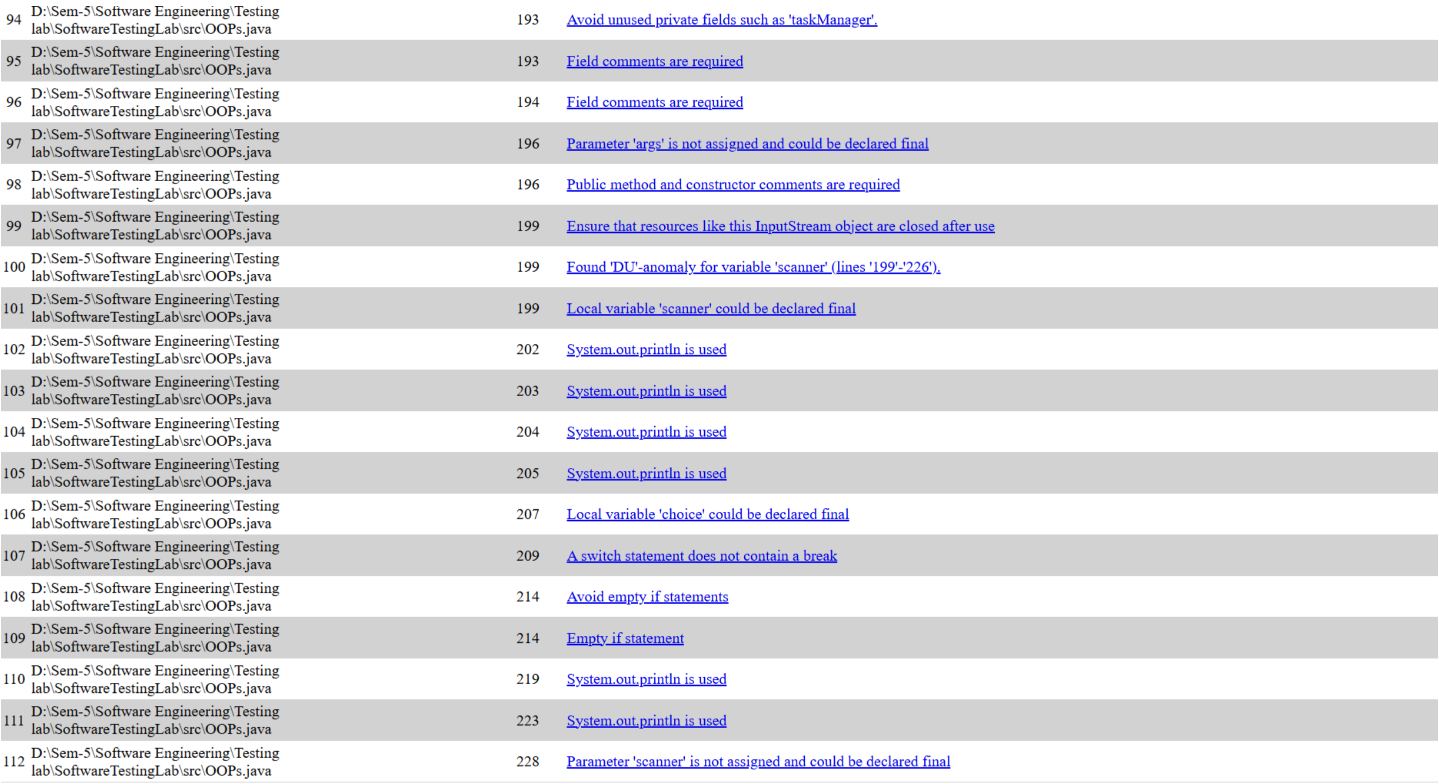
PMD report

Problems found

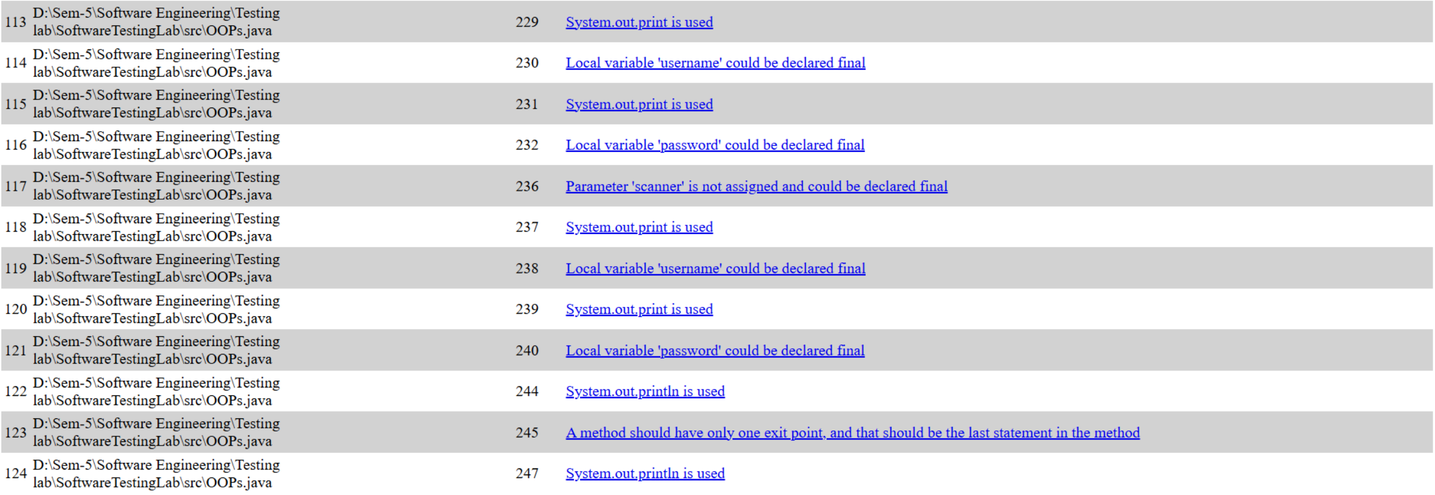
#	File	Line	Problem
1	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	4	All classes, interfaces, enums and annotations must belong to a named package
2	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	4	Avoid short class names like User
3	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	4	Class comments are required
4	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	5	Field comments are required
5	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	5	Private field 'username' could be made final; it is only initialized in the declaration or constructor.
6	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	6	Field comments are required
7	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	6	Private field 'password' could be made final; it is only initialized in the declaration or constructor.
8	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	8	Parameter 'password' is not assigned and could be declared final
9	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	8	Parameter 'username' is not assigned and could be declared final
10	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	8	Public method and constructor comments are required
11	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	26	Parameter 'data' is not assigned and could be declared final
12	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	26	Public method and constructor comments are required
13	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	27	Local variable 'parts' could be declared final
14	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	32	Avoid short class names like Task
15	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	32	Class comments are required
16	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	33	Field comments are required
17	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	33	Private field 'title' could be made final; it is only initialized in the declaration or constructor.

18	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	34	Field comments are required
19	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	34	Private field 'description' could be made final; it is only initialized in the declaration or constructor.
20	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	35	Field comments are required
21	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	37	Parameter 'description' is not assigned and could be declared final
22	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	37	Parameter 'title' is not assigned and could be declared final
23	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	37	Public method and constructor comments are required
24	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	55	Public method and constructor comments are required
25	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	64	Parameter 'data' is not assigned and could be declared final
26	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	64	Public method and constructor comments are required
27	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	65	Local variable 'parts' could be declared final
28	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	66	Local variable 'task' could be declared final
29	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	67	Avoid using Literals in Conditional Statements
30	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	74	Class comments are required
31	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	75	Field comments are required
32	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	75	Private field 'users' could be made final; it is only initialized in the declaration or constructor.
33	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	76	Field comments are required
34	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	78	Parameter 'filename' is not assigned and could be declared final
35	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	78	Public method and constructor comments are required
36	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	85	Avoid instantiating FileInputStream, FileOutputStream, FileReader, or FileWriter
37	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	85	Avoid variables with short names like br
38	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	87	Avoid assignments in operands
39	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	87	Found 'DU'-anomaly for variable 'line' (lines '87'-'93').
40	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	91	System.out.println is used
41	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	95	Public method and constructor comments are required
42	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	96	Avoid instantiating FileInputStream, FileOutputStream, FileReader, or FileWriter
43	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	96	Avoid variables with short names like bw
44	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	97	Local variable 'user' could be declared final
45	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	102	System.out.println is used
46	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	106	Parameter 'password' is not assigned and could be declared final
47	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	106	Parameter 'username' is not assigned and could be declared final
48	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	106	Public method and constructor comments are required
49	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	109	System.out.println is used
50	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	112	Parameter 'password' is not assigned and could be declared final
51	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	112	Parameter 'username' is not assigned and could be declared final
52	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	112	Public method and constructor comments are required
53	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	113	Local variable 'user' could be declared final
54	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	114	Potential violation of Law of Demeter (method chain calls)
55	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	114	Potential violation of Law of Demeter (method chain calls)

56	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	115	A method should have only one exit point, and that should be the last statement in the method
57	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	122	Class comments are required
58	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	123	Field comments are required
59	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	123	Private field 'tasks' could be made final; it is only initialized in the declaration or constructor.
60	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	124	Field comments are required
61	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	126	Parameter 'filename' is not assigned and could be declared final
62	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	126	Public method and constructor comments are required
63	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	133	Avoid instantiating FileInputStream, FileOutputStream, FileReader, or FileWriter
64	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	133	Avoid variables with short names like br
65	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	135	Avoid assignments in operands
66	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	135	Found 'DU'-anomaly for variable 'line' (lines '135'-'141').
67	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	139	System.out.println is used
68	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	143	Public method and constructor comments are required
69	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	144	Avoid instantiating FileInputStream, FileOutputStream, FileReader, or FileWriter
70	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	144	Avoid variables with short names like bw
71	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	145	Local variable 'task' could be declared final
72	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	150	System.out.println is used
73	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	154	Parameter 'description' is not assigned and could be declared final
74	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	154	Parameter 'title' is not assigned and could be declared final
75	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	154	Public method and constructor comments are required
76	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	157	System.out.println is used
77	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	160	Public method and constructor comments are required
78	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	162	System.out.println is used
79	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	167	Parameter 'index' is not assigned and could be declared final
80	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	167	Public method and constructor comments are required
81	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	169	System.out.println is used
82	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	172	Local variable 'task' could be declared final
83	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	173	Potential violation of Law of Demeter (object not created locally)
84	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	175	System.out.println is used
85	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	178	Parameter 'index' is not assigned and could be declared final
86	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	178	Public method and constructor comments are required
87	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	180	System.out.println is used
88	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	185	System.out.println is used
89	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	189	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.
90	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	189	Class comments are required
91	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	190	Field comments are required
92	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	191	Field comments are required
93	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	192	Field comments are required



94	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	193	Avoid unused private fields such as 'taskManager'.
95	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	193	Field comments are required
96	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	194	Field comments are required
97	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	196	Parameter 'args' is not assigned and could be declared final
98	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	196	Public method and constructor comments are required
99	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	199	Ensure that resources like this InputStream object are closed after use
100	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	199	Found 'DU'-anomaly for variable 'scanner' (lines '199'-'226').
101	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	199	Local variable 'scanner' could be declared final
102	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	202	System.out.println is used
103	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	203	System.out.println is used
104	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	204	System.out.println is used
105	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	205	System.out.println is used
106	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	207	Local variable 'choice' could be declared final
107	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	209	A switch statement does not contain a break
108	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	214	Avoid empty if statements
109	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	214	Empty if statement
110	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	219	System.out.println is used
111	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	223	System.out.println is used
112	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	228	Parameter 'scanner' is not assigned and could be declared final



113	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	229	System.out.print is used
114	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	230	Local variable 'username' could be declared final
115	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	231	System.out.print is used
116	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	232	Local variable 'password' could be declared final
117	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	236	Parameter 'scanner' is not assigned and could be declared final
118	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	237	System.out.print is used
119	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	238	Local variable 'username' could be declared final
120	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	239	System.out.print is used
121	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	240	Local variable 'password' could be declared final
122	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	244	System.out.println is used
123	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	245	A method should have only one exit point, and that should be the last statement in the method
124	D:\Sem-5\Software Engineering\Testing lab\SoftwareTestingLab\src\OOPs.java	247	System.out.println is used

You can see,

Full code here:

https://github.com/akhaja-D6255/IT314-Lab-Assignements/blob/main/OOPs.java

Full HTML output of PMD here :

https://github.com/akhaja-D6255/IT314-Lab-Assignements/blob/main/PMD%20report.html