# Software Engineering

IT – 314 | LAB – 7

Dharmik Godhani | 202201311

## Armstrong Code

### Program Inspection

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

There is a data reference error when string arguments are passed as command line inputs. The program doesn't handle cases where no input is provided. Also, there's a mistake in the for loop where the modulus and division operators are switched, causing wrong calculations of the remainder and quotient.

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

Category C, which deals with computational errors, was the most helpful since the code failed because of the wrong operators.

**Question 3. Which type of error were you unable to identify using the program inspection?**

In this case, all errors were found using the program inspection method.

**Question 4. Is the program inspection technique worth applying?**

Since the code is short and the errors are easy to spot, the program inspection method is useful and worth using here.

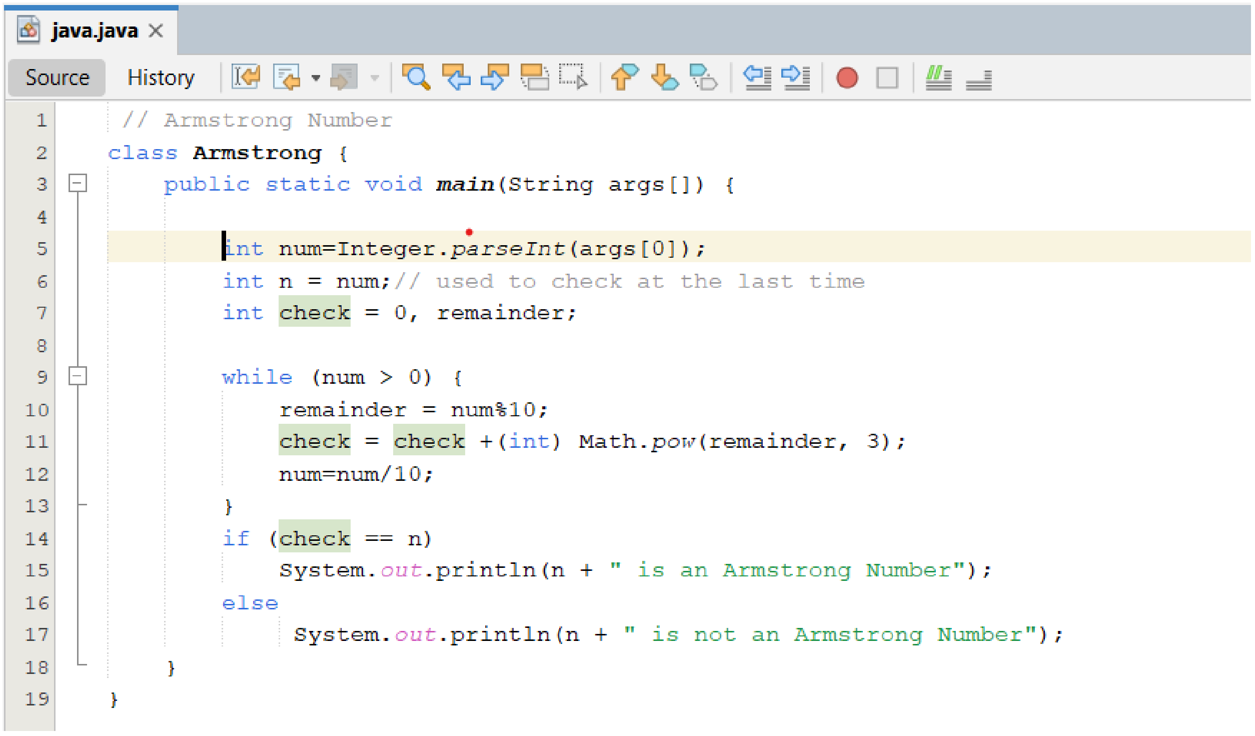
### Code Debugging

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

There are two errors in the program. Both the remainder and quotient calculations are incorrect because the operators were swapped.

**Question 2. How many breakpoints do you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?**

Using just one breakpoint at line 12, as shown in the figure, the error can be spotted on the first run of the while loop when the remainder and check values are incorrect. These errors can be fixed by replacing the wrong operators in the code to correctly compute the remainder, check, and num.



```
1   // Armstrong Number
2  class Armstrong {
3      public static void main(String args[]) {
4  
5          int num=Integer.parseInt(args[0]);
6          int n = num;// used to check at the last time
7          int check = 0, remainder;
8  
9          while (num > 0) {
10             remainder = num%10;
11             check = check +(int) Math.pow(remainder, 3);
12             num=num/10;
13         }
14         if (check == n)
15             System.out.println(n + " is an Armstrong Number");
16         else
17              System.out.println(n + " is not an Armstrong Number");
18     }
19 }
```

## GCD and LCM Code

### Program Inspection

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

There is a comparison error in determining which number, x or y, is larger or smaller, and in checking if the modulus is equal to or not equal to zero.

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

Category D, which focuses on comparison errors, was the most useful because the code failed due to incorrect comparisons.

**Question 3. Which type of error were you unable to identify using the program inspection?**

In this case, all errors were found using the program inspection method.

**Question 4. Is the program inspection technique worth applying?**

Since the code is short and the comparison errors are easy to spot, the program inspection technique is well-suited and worth applying here.

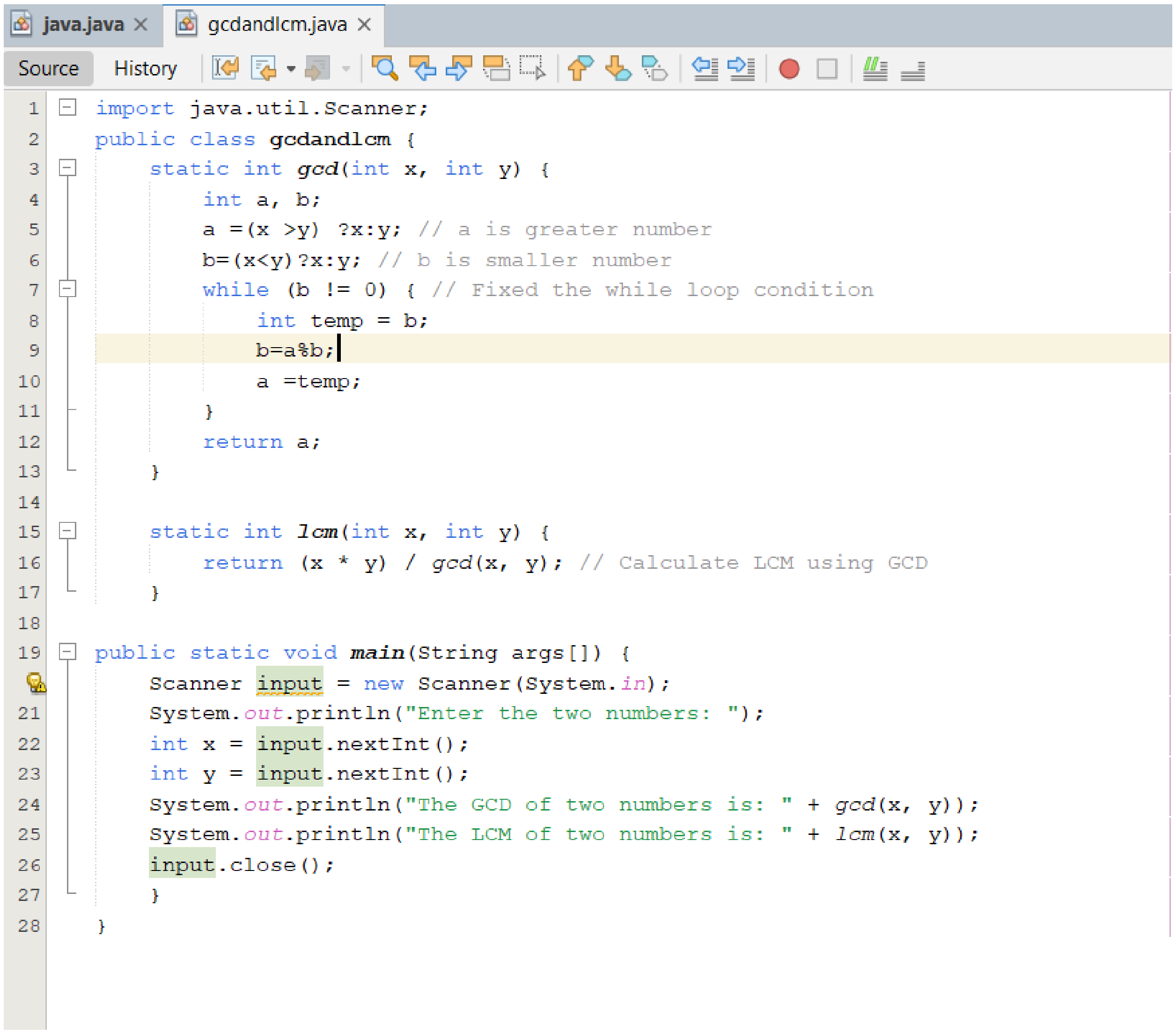
### Code Debugging

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

There are three errors in the program. First, the calculation of the larger value between x and y is incorrect. The other two errors involve the comparison with the modulus operator: for GCD, the loop should run until the modulus is not equal to zero, while for LCM, it should return when the modulus equals zero. Additionally, the code would fail if either number is zero, as the modulus operator isn't defined for zero.

**Question 2. How many breakpoints do you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?**

Three breakpoints are needed. One at the start and one at the end of the GCD loop helped identify both the mistake in the calculation of "a" and the error in the loop condition. Two more breakpoints are needed at the start and end of the LCM loop, where the error in the return condition for LCM can be detected.



```
import java.util.Scanner;
public class gcdandlcm {
    static int gcd(int x, int y) {
        int a, b;
        a =(x >y) ?x:y; // a is greater number
        b=(x<y)?x:y; // b is smaller number
        while (b != 0) { // Fixed the while loop condition
            int temp = b;
            b=a%b;
            a =temp;
        }
        return a;
    }

    static int lcm(int x, int y) {
        return (x * y) / gcd(x, y); // Calculate LCM using GCD
    }

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();
    }
}
```

## Knapsack Code

### Program Inspection

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

One error is the uninitialized vector opt, which should be initialized to 0 for cases where the bag's weight or the number of items is 0. Similarly, sol should be initialized to F, as no items are taken in those boundary conditions.

Another error is in the logic: option1 should be opt[n-1][w] instead of opt[n++][w], and option2 should be profit[n] + opt[n-1][w-weight[n]] rather than profit[n-2] + opt[n-1][w-weight[n]]. Additionally, the condition for option2 should be weight[n] <= w.

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

Category A, which focuses on Data Reference Errors, was the most effective, as the code failed due to incorrect value references and uninitialized variables.

**Question 3. Which type of error were you unable to identify using the program inspection?**

In this case, all errors were identified using the program inspection method.

**Question 4. Is the program inspection technique worth applying?**

Although the code was small, finding the errors was somewhat tedious using the program inspection method.

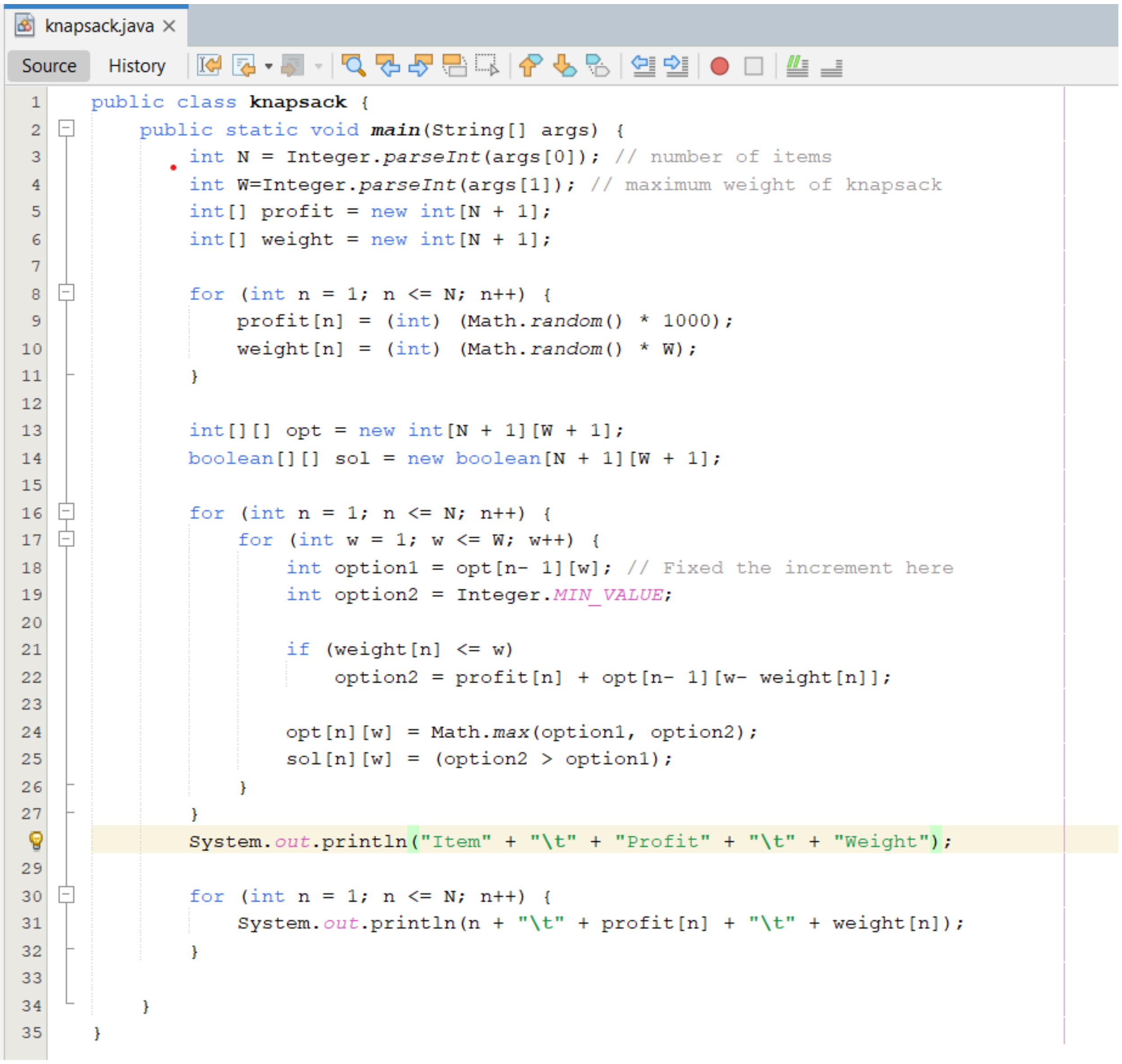
### Code Debugging

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

The main errors involved uninitialized 2D arrays `opt` and `sol`. While these don't cause an error in Java because they get initialized correctly, it's still good practice to explicitly initialize them. Another issue was in the logic for calculating both `option1` and `option2`, along with the conditions used.

**Question 2. How many breakpoints do you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?**

Breakpoints were added inside the double for loop, checking every 2 to 3 iterations. This made it easier to identify the mistakes. The errors were then fixed by initializing the 2D arrays and correcting the logic for the knapsack calculation.



knapsack.java ×

Source History

```
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];

        for (int n = 1; n <= N; n++) {
            profit[n] = (int) (Math.random() * 1000);
            weight[n] = (int) (Math.random() * W);
        }

        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++) {
                int option1 = opt[n- 1][w]; // Fixed the increment here
                int option2 = Integer.MIN_VALUE;

                if (weight[n] <= w)
                    option2 = profit[n] + opt[n- 1][w- weight[n]];

                opt[n][w] = Math.max(option1, option2);
                sol[n][w] = (option2 > option1);
            }
        }
        System.out.println("Item" + "\t" + "Profit" + "\t" + "Weight");

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

    }
}
```

## Magic Number Code

### Program Inspection

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

There is a syntax error on line 18, which can be detected by the compiler. Additionally, if 119 is meant to be a magic number, the logic in the code is flawed. The condition for the inner while loop is incorrect; it should run until `sum` is not equal to zero. The logic inside the loop also needs to be adjusted to properly determine whether the number is a magic number.

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

Category C, which focuses on computational errors, was the most useful here, as the code failed due to incorrect operators.

**Question 3. Which type of error were you unable to identify using the program inspection?**

In this case, all errors were identified using the program inspection method.

**Question 4. Is the program inspection technique worth applying?**

Given the short length of the code and the ease of spotting computational errors, the program inspection method is well-suited and worth applying here.

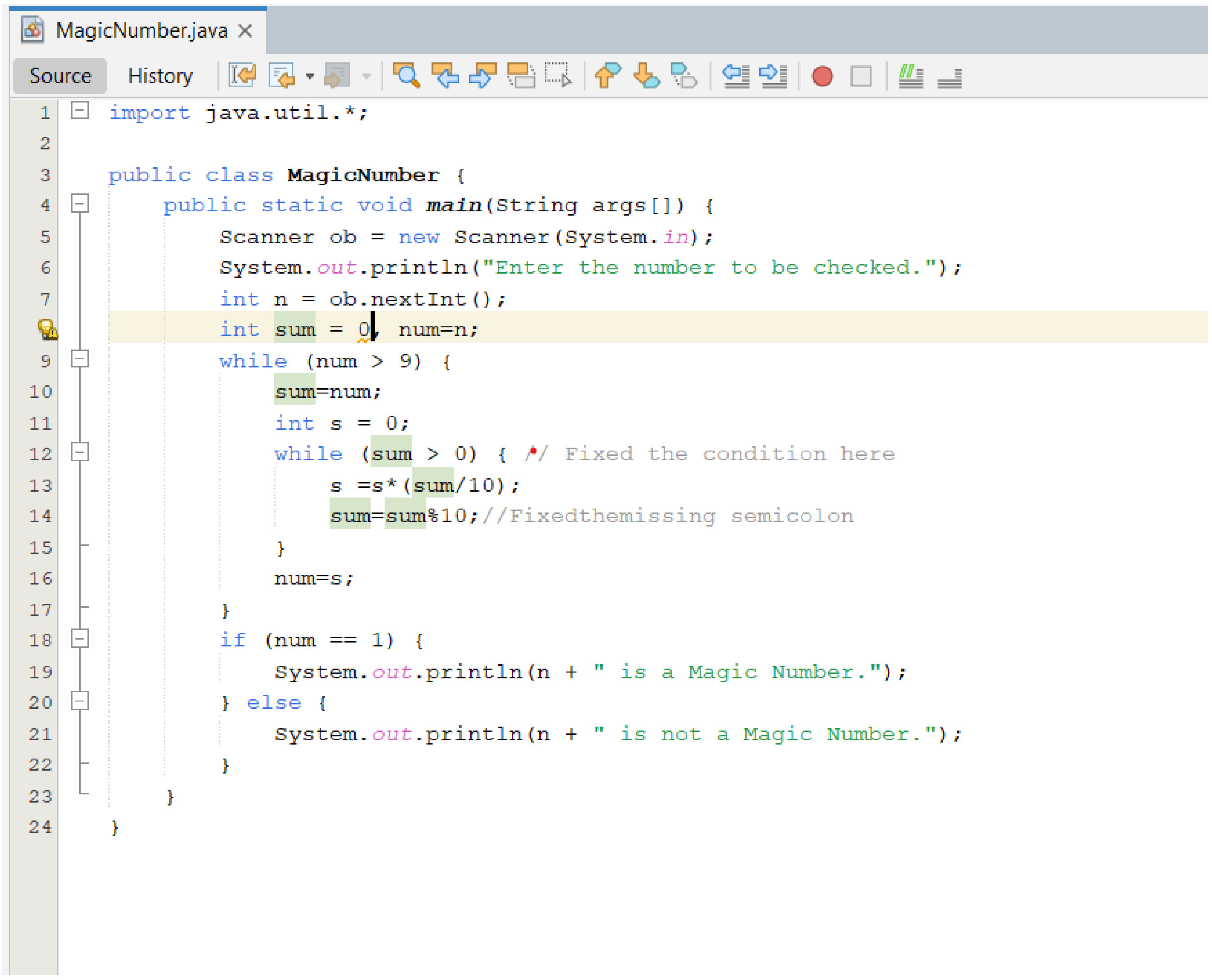
### Code Debugging

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

There were two errors in the program. After fixing the syntax error and updating the condition in the inner while loop, the logic for finding the magic number also needed to be corrected.

**Question 2. How many breakpoints do you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?**

A single breakpoint at line 18 was enough to track the iterations of the for loop and identify the errors. After fixing the mistakes, as shown below, the program ran successfully.



```java
import java.util.*;

public class MagicNumber {
    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) { // Fixed the condition here
                s =s*(sum/10);
                sum=sum%10;//Fixedthemissing semicolon
            }
            num=s;
        }
        if (num == 1) {
            System.out.println(n + " is a Magic Number.");
        } else {
            System.out.println(n + " is not a Magic Number.");
        }
    }
}
```

## Matrix Multiplication Code

### Program Inspection

**Question 1. How many errors are there in the program? Mention the errors you have identified.**
The program contained a single error related to the incorrect calculation of values in the final product matrix.

**Question 2. Which category of program inspection would you find more effective?**
In this case, Category C of the program inspection, which addresses computational errors, was the most effective, as the failure stemmed from incorrect arithmetic logic.

**Question 3. Which type of error were you unable to identify using the program inspection?**
In this instance, all errors were identifiable using the program inspection method.

**Question 4. Is the program inspection technique worth applying?**
Considering the short length of the code and the simplicity of finding computational errors, the program inspection technique is indeed applicable here.

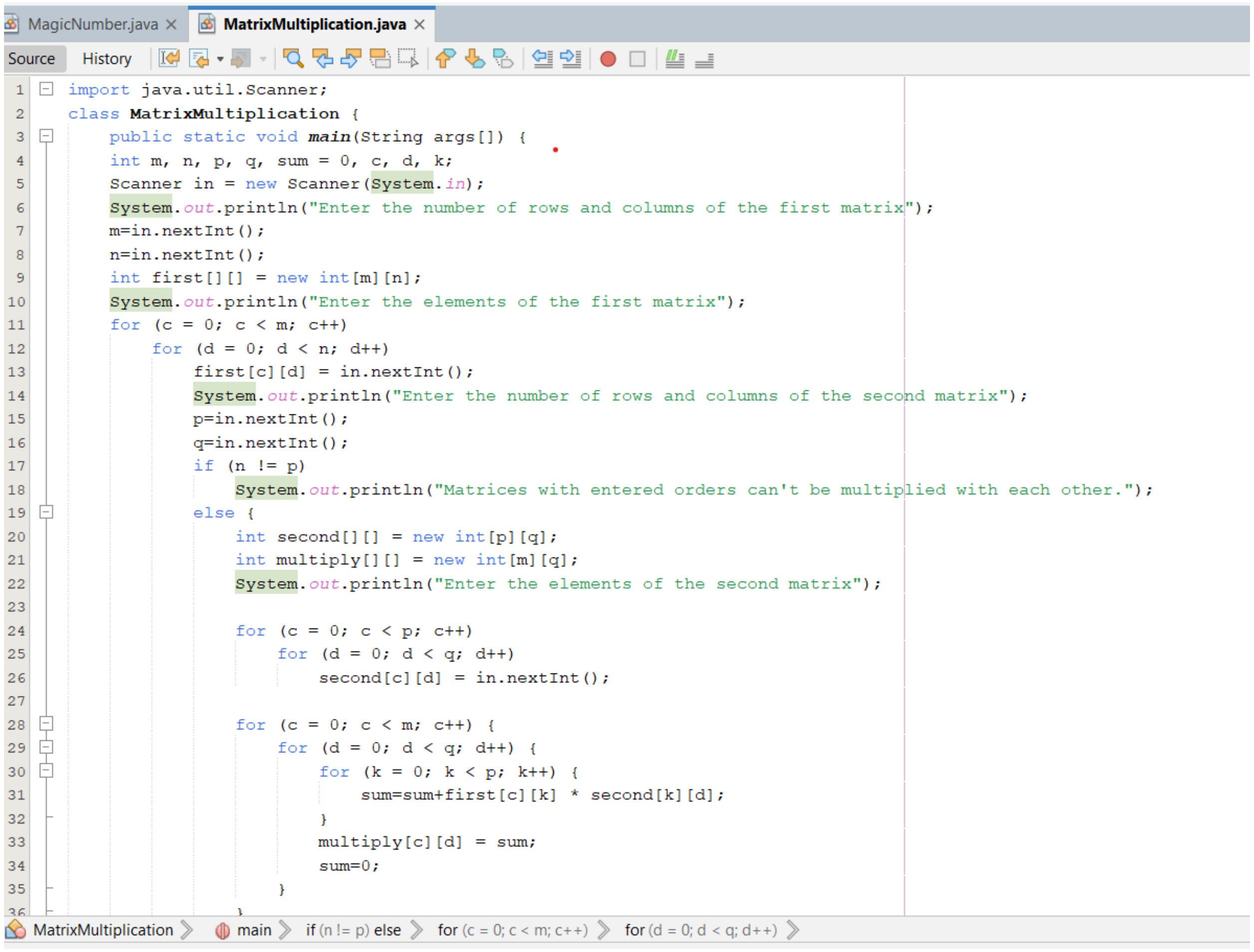
### Code Debugging

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

There was just one error in the program related to the calculation of the product matrix.

**Question 2. How many breakpoints do you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?**

By setting a breakpoint at line 46, we can go through each loop and examine the calculated values. This would reveal an array out-of-bounds error, indicating the incorrect use of indices in the sum calculation.


MagicNumber.java × MatrixMultiplication.java ×

Source History

```
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();

                for (c = 0; c < m; c++) {
                    for (d = 0; d < q; d++) {
                        for (k = 0; k < p; k++) {
                            sum=sum+first[c][k] * second[k][d];
                        }
                        multiply[c][d] = sum;
                        sum=0;
                    }
                }
```

MatrixMultiplication > main > if (n != p) else > for (c = 0; c < m; c++) > for (d = 0; d < q; d++) >

## Merge Sort Code

### Program Inspection

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

The only error in this code was a syntax error caused by the incorrect use of the left and right variables.

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

For this case, Category A, which addresses Data Reference Errors, was the most appropriate, as it focused on the issue with the wrongly referenced left and right variables.

**Question 3. Which type of error were you unable to identify using the program inspection?**

In this case, all the errors in the document were identifiable using the program inspection method.

**Question 4. Is the program inspection technique worth applying?**

Implementing the program inspection technique was challenging for this code. Due to its length, it was difficult to review every item on the checklist by analyzing the entire code.

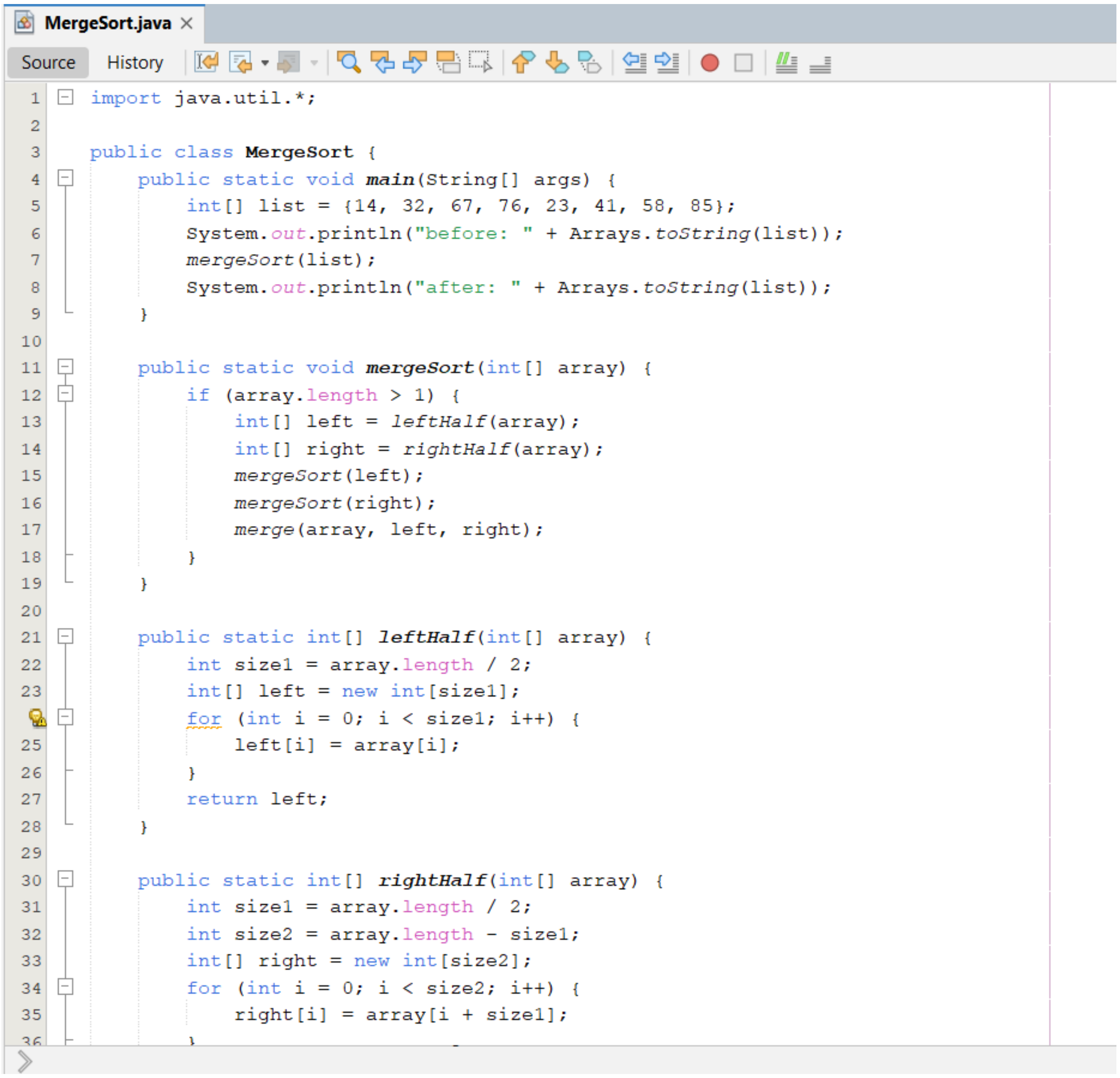
### Code Debugging

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

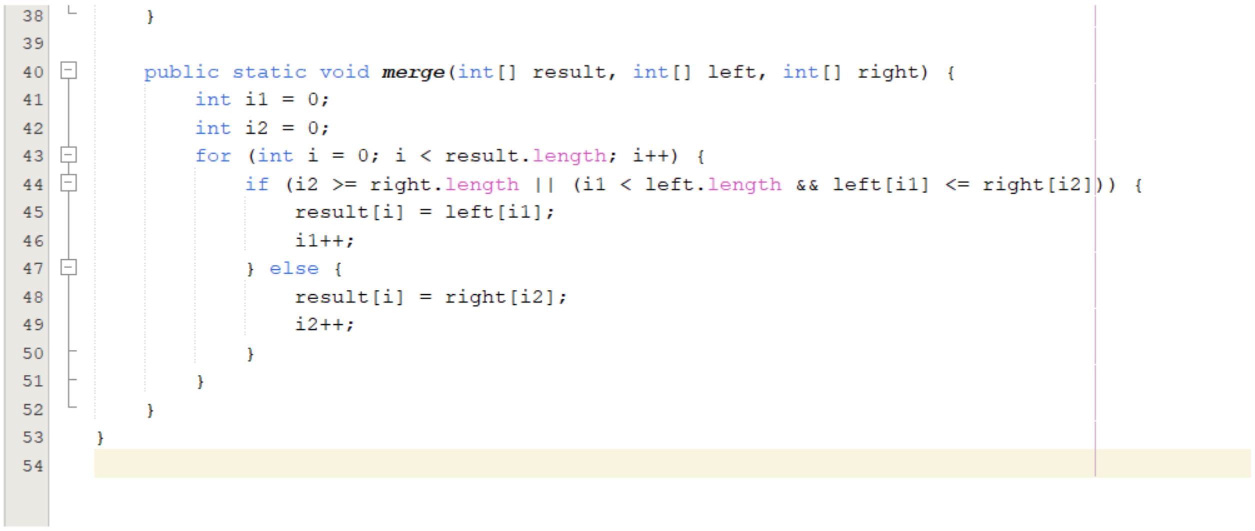
There were several syntax errors in the code, which were resolved by passing the complete array and then calculating the left and right sections of the array.

**Question 2. How many breakpoints do you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?**

With three breakpoints—one in the loop for calculating the left half, one in the loop for calculating the right half, and one in the loop for calculating the merge array—we can identify any mistakes during execution, one iteration at a time.



```
import java.util.*;

public class MergeSort {
    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));
    }

    public static void mergeSort(int[] array) {
        if (array.length > 1) {
            int[] left = leftHalf(array);
            int[] right = rightHalf(array);
            mergeSort(left);
            mergeSort(right);
            merge(array, left, right);
        }
    }

    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;
    }

    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];
```



```
    }

    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0;
        int i2 = 0;
        for (int i = 0; i < result.length; i++) {
            if (i2 >= right.length || (i1 < left.length && left[i1] <= right[i2])) {
                result[i] = left[i1];
                i1++;
            } else {
                result[i] = right[i2];
                i2++;
            }
        }
    }
}
```

# Sorting Array Code

## Program Inspection

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

There are several syntax errors in the code, particularly in the first double for loop, which is mistakenly closed with a semi-colon and has an incorrect boundary condition of greater than or equal to n instead of less than. Once the syntax errors are fixed, the condition for checking increasing elements is also incorrect, leading to output in decreasing order. Therefore, we adjust the loop's check condition.

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

For this question, Category D of the program inspection, which focuses on comparison errors, was the most effective, as the code's failure was due to improper comparisons between elements.

**Question 3. Which type of error were you unable to identify using the program inspection?**

In this case, all errors were identifiable using the program inspection method.

**Question 4. Is the program inspection technique worth applying?**

Given the short length of the code and the ease of identifying comparison errors, the program inspection technique is indeed applicable here.

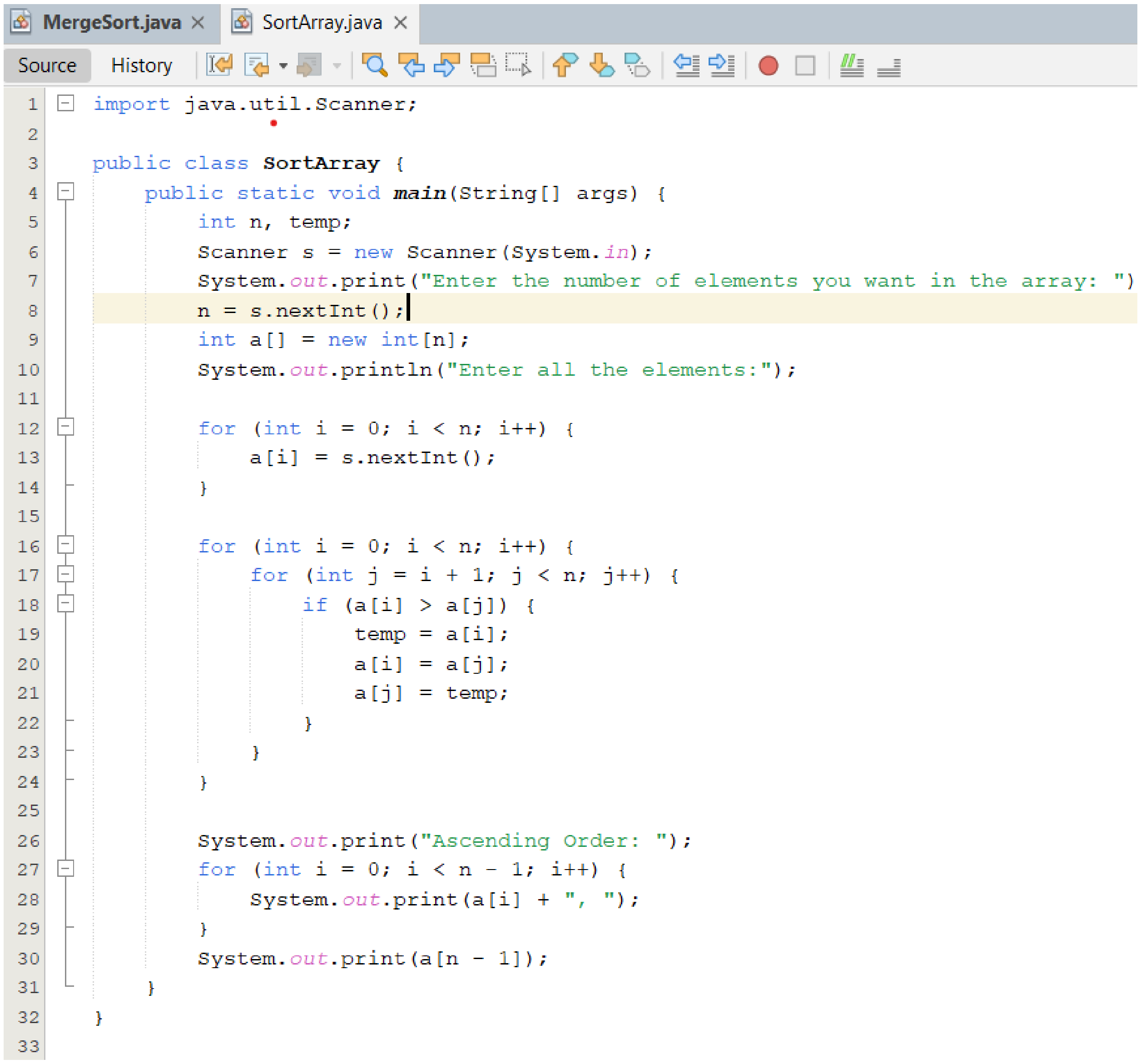
## Code Debugging

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

There were two errors in the program. Once the syntax errors were resolved, the remaining task was to adjust the check condition so that the output array would be sorted in ascending order instead of descending order.

**Question 2. How many breakpoints do you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?**

By placing a breakpoint on line 26, we discovered that the code was sorting the array in descending order instead of the required ascending order. We corrected this by modifying the check condition in the if loop.



```java
import java.util.Scanner;

public class SortArray {
    public static void main(String[] args) {
        int n, temp;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter the number of elements you want in the array: ")
        n = s.nextInt();
        int a[] = new int[n];
        System.out.println("Enter all the elements:");

        for (int i = 0; i < n; i++) {
            a[i] = s.nextInt();
        }

        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }

        System.out.print("Ascending Order: ");
        for (int i = 0; i < n - 1; i++) {
            System.out.print(a[i] + ", ");
        }
        System.out.print(a[n - 1]);
    }
}
```

## Stack Implementation Code

### Program Inspection

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

Once we identify the error in printing the stack within the for loop and correct the condition, we can easily fix the logic of the stack implementation, particularly the top variable of the stack.

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

In this case, Category A of the program inspection, which deals with Data Reference Errors, was the most effective, as the failure occurred due to incorrect access of stack data linked to errors in calculating the top variable's value.

**Question 3. Which type of error were you unable to identify using the program inspection?**

In this instance, all errors were identifiable using the program inspection method.

**Question 4. Is the program inspection technique worth applying?**

Given that the code was somewhat lengthy and contained many functions, it was a bit challenging to find the errors using the program inspection method.

### Code Debugging

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

There were two errors in this program: the for-loop condition at the end for printing and the updates to the top variable during the stack pop and push operations.

**Question 2. How many breakpoints do you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?**

By placing breakpoints at the end of the push and pop operations, as well as in the display function loop, we can observe the values of top and how the referenced value changes iteratively. Once we identify the calculation error, we can update the top variable correctly.

## Quadratic Probing

### Program Inspection

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

Three errors are there in this program. There's a typo in the insert method, in the line i += (i+h / h--). Then there is a logic error in the remove method. It should be i = (i + h*h++). In the get method, there is a logic error in the loop to find the keyl. It should be i = (i + h * h++)

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

The category of program inspection that would be most effective for this code is Category A: Syntax Errors and Category B: Semantic Errors, as there are both syntax errors and semantic issues in the code.

**Question 3. Which type of error were you unable to identify using the program inspection?**

In this case, all the errors were identifiable using the program inspection.

**Question 4. Is the program inspection technique worth applying?**

The program inspection technique is worth applying to identify and fix these errors, but it may not identify logical errors that affect the program's behavior.

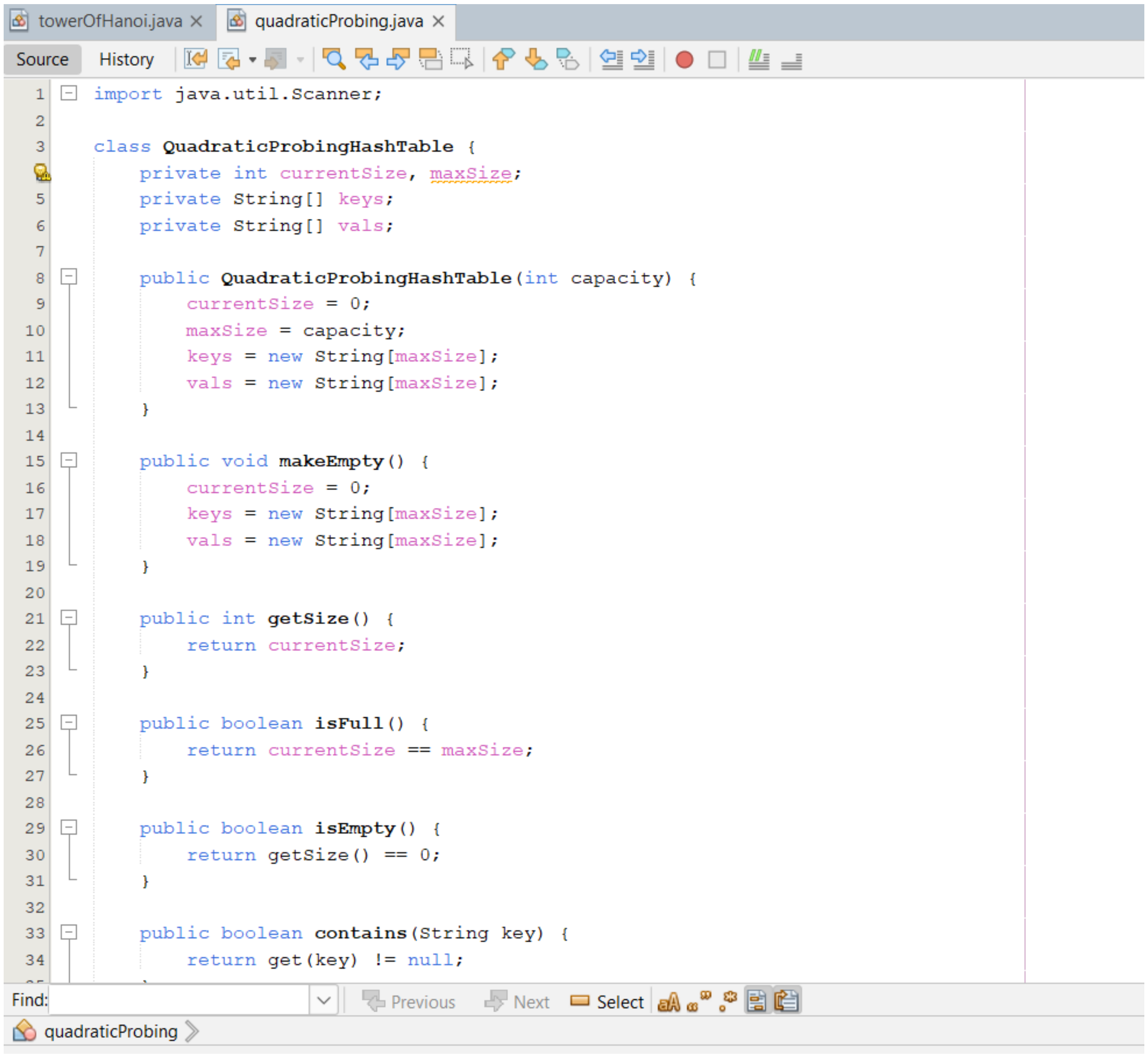
### Code Debugging

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

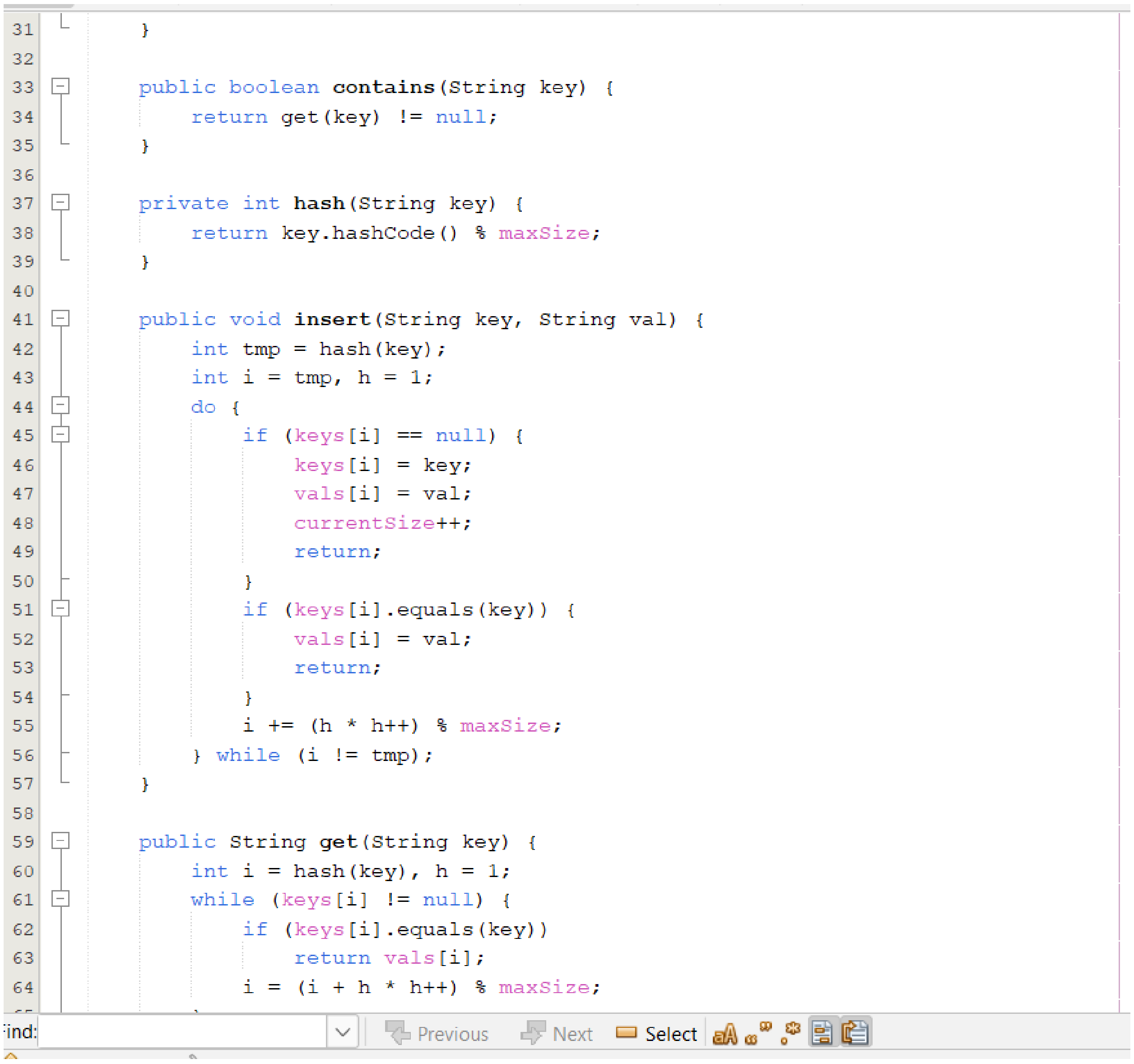
There are three errors in the program, as identified above.

**Question 2. How many breakpoints do you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?**

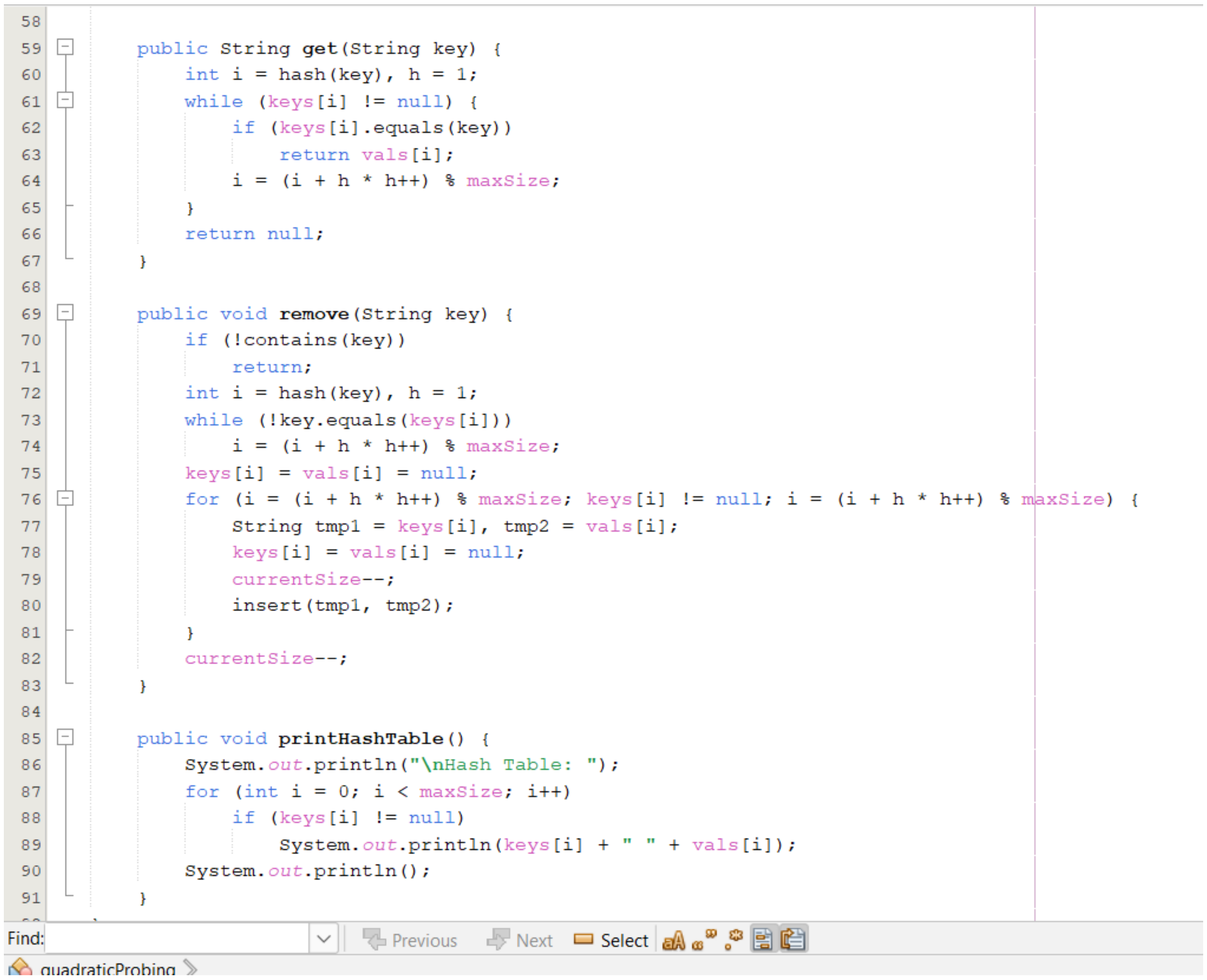
To fix these errors, you would need to set breakpoints and step through the code while examining variables like i, h, tmp1, and tmp2. You should pay attention to the logic of the insert, remove and get methods.



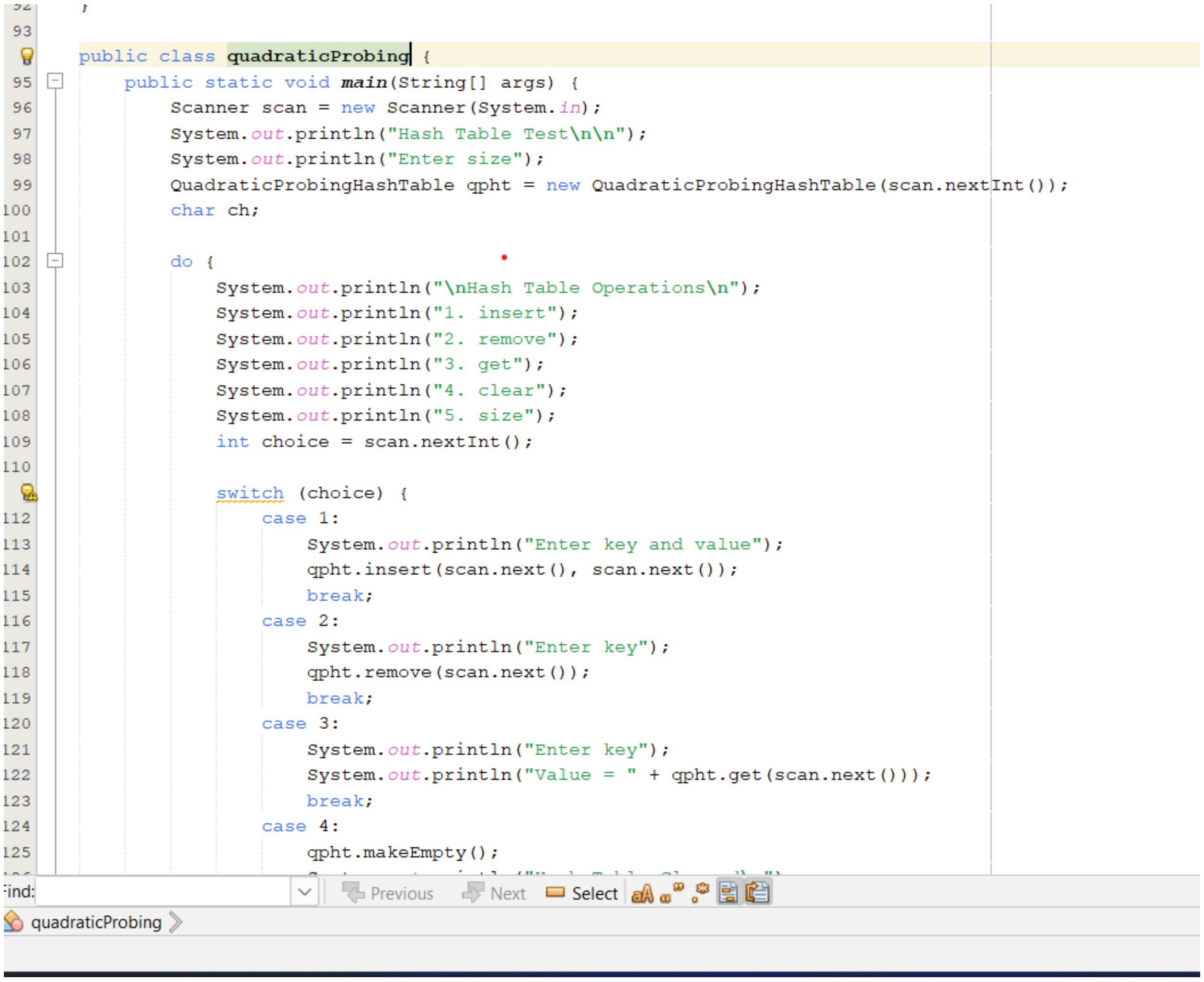
```java
import java.util.Scanner;

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

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

    public void makeEmpty() {
        currentSize = 0;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

    public int getSize() {
        return currentSize;
    }

    public boolean isFull() {
        return currentSize == maxSize;
    }

    public boolean isEmpty() {
        return getSize() == 0;
    }

    public boolean contains(String key) {
        return get(key) != null;
```



```
    }

    public boolean contains(String key) {
        return get(key) != null;
    }

    private int hash(String key) {
        return key.hashCode() % maxSize;
    }

    public void insert(String key, String val) {
        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 += (h * h++) % maxSize;
        } while (i != tmp);
    }

    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;
```



```java

    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;
        }
        return null;
    }

    public void remove(String key) {
        if (!contains(key))
            return;
        int i = hash(key), h = 1;
        while (!key.equals(keys[i]))
            i = (i + h * h++) % maxSize;
        keys[i] = vals[i] = null;
        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--;
    }

    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();
    }
```



```
}

public class quadraticProbing {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Hash Table Test\n\n");
        System.out.println("Enter size");
        QuadraticProbingHashTable qpht = new QuadraticProbingHashTable(scan.nextInt());
        char ch;

        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();
```

## Tower of Hanoi

### Program Inspection

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

There is one error in the line doTowers(topN ++, inter–, from+1, to+1), there are errors in the increment and decrement operators. It should be corrected to doTowers(topN- 1, inter, from, to).

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

The most effective category of program inspection would be Category B: Semantic Errors

**Question 3. Which type of error were you unable to identify using the program inspection?**

In this case, all the errors were identifiable using the program inspection.

**Question 4. Is the program inspection technique worth applying?**

The program inspection technique is worth applying to identify and fix semantic errors in the code.

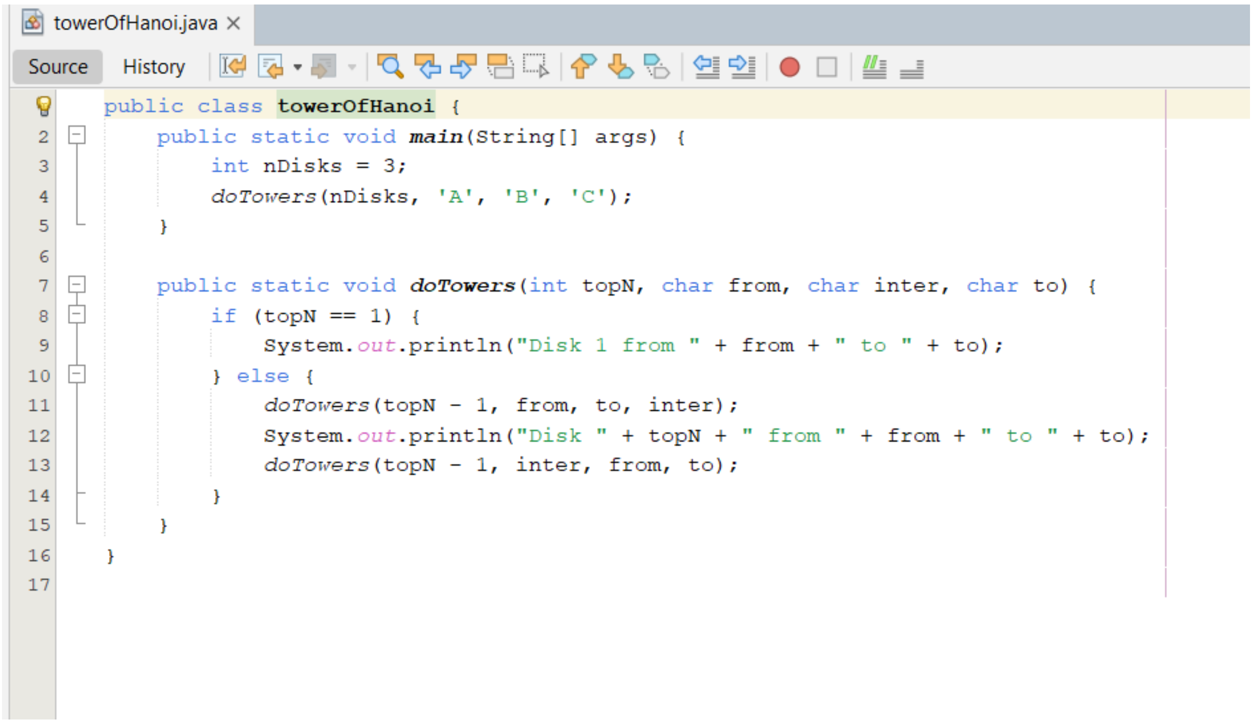
### Code Debugging

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

There are one errors in the program, as identified above.

**Question 2. How many breakpoints do you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?**

To fix this error, you need to replace the line: doTowers(topN ++, inter--, from+1, to+1); with the correct version: doTowers(topN- 1, inter, from, to);



```java
public class towerOfHanoi {
    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 - 1, inter, from, to);
        }
    }
}
```