# IT314 SOFTWARE ENGINEERING

# LAB 7 - CODE DEBUGGING


## 202201444 - VARDHMAN MEHTA


**AS TOLD IN LAB GROUP - 5/6, I HAVE DONE TASK 1 AND 3 ON A GITHUB CODE AND TASK 2 ON THE 10 CODES PROVIDED BY SIR IN GOOGLE CLASSROOM.**

**GITHUB CODE LINK : I have used the codes of these files from this repo for the lab**

# I. PROGRAM INSPECTION:

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

### Errors Identified:

1) Data Reference Errors:
- Unset or uninitialized variables: Inconsistent initialization for some variables like `existing_idx` in several places.
- Dangling references in pointers: Use of pointers (e.g., `EditorUndoRedoManager *undo_redo`) without checking their validity before dereferencing.
- Mismatched pointer attributes: `animation.ptr()` could lead to potential issues if `animation` is null or invalid.

2) Data Declaration Errors:
- Explicit declaration of variables: Missing clear variable declarations leading to potential type mismatch, especially with `real_t` and `float`.
- Understanding of default variable attributes: Use of different types (e.g., `real_t` vs. `float`) without explicit conversions could cause unintended behavior.

3) Computation Errors:
    - Inconsistent data types in computations: The function mixes `float` and `real_t`, leading to possible precision issues.
    - Possible zero divisor in division operations: In the context of the `animation->track_get_key_value` calls, there may be scenarios where the key value is zero.

4) Comparison Errors:

- Comparisons between variables of different data types: Checks like `if (key.track != 0)` may not be consistent across types.
- Clarity of Boolean expressions: Conditions like `if (p_ofs_valid)` could lead to confusion if not handled consistently.

5) Control-Flow Errors:
- Index variable exceeding the number of branch possibilities: Loops that traverse `selection` could lead to out-of-bounds access if not handled correctly.
- Assurance that every loop eventually terminates: In some cases, while looping over the selection, the termination condition is not explicitly stated.

6) Interface Errors:
- Matching number of parameters and arguments in module calls: Calls to `add_do_method` and `add_undo_method` may not consistently match expected arguments, leading to runtime errors.

7) Input / Output Errors:
- Correct handling of I/O error conditions: Lack of checks before file operations could lead to runtime exceptions.

8) Other Checks:
- Verification of variable attributes against unexpected defaults: Variables like `insert_pos` and others are derived without validating their values against expected ranges.

**Total Errors Identified:** Approximately 15 distinct errors categorized into various error types as discussed above, including issues with data reference, declaration, computation, comparison, control-flow, interface, I/O, and other checks.

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

- Category A: Data Reference Errors and Category D: Comparison Errors would be particularly effective since many identified issues stem from improper variable handling and type comparisons. Addressing these could significantly enhance code reliability and prevent runtime errors.

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

- Unidentified Errors: Runtime errors caused by logic flaws that do not manifest until specific conditions are met (e.g., empty selections leading to out-of-bounds access or null pointer dereferencing) cannot be easily identified during static inspection. Such issues may require dynamic analysis or extensive testing to uncover.

4. **Is the program inspection technique worth applicable?**

- Yes, the program inspection technique is worth applying. It helps identify logical, structural, and common coding errors early in the development process, potentially reducing debugging time and improving code quality. While it may not catch all types of errors, particularly runtime errors, it provides a structured approach to reviewing code and fosters better understanding and collaboration among developers. Additionally, combining inspection with other techniques (like unit testing and dynamic analysis) can enhance overall software reliability and performance.

## II. CODE DEBUGGING:

## 1) Armstrong Number :

```
//Armstrong Number
class Armstrong{
	public static void main(String args[]){
		int num = Integer.parseInt(args[0]);
		int n = num; //use to check at last time
		int check=0,remainder;
		while(num > 0){
			remainder = num / 10;
			check = check + (int)Math.pow(remainder,3);
			num = num % 10;
		}
		if(check == n)
			System.out.println(n+" is an Armstrong Number");
		else
			System.out.println(n+" is not a Armstrong Number");
	}
```

Input: 153
Output: 153 is an armstrong Number.

### 1. Errors in the code:

**1: `remainder = num / 10;`**

- This line is supposed to extract the last digit of the number, but it's performing integer division (`/`), which gives the quotient instead of the remainder. The correct operation should be **`num % 10`** to get the remainder (the last digit of the number).

**2: `num = num % 10;`**

- This line is intended to remove the last digit, but it is incorrectly using the modulus operator. It should use integer division (`/`) instead of modulus (`%`). The correct operation is **`num = num / 10;`** to remove the last digit.
- At last, there should be a closing bracket.

## 2. Breakpoints needed to fix the errors:

- Check the initial values of `num`, `check`, and `remainder`.
- Check the value of `remainder` after the division.
- Check how the value of `num` changes after updating.

## 3. Steps to fix the errors:

**Step 1**: Fix the incorrect operations.

- Change line 10 to `remainder = num % 10;`
- Change line 12 to `num = num / 10;`

## 4. FIXED CODE:

```
//Armstrong Number
class Armstrong{
	public static void main(String args[]){
		int num = Integer.parseInt(args[0]);
		int n = num; //use to check at last time
		int check=0,remainder;
		while(num > 0){
			remainder = num / 10;
			check = check + (int)Math.pow(remainder,3);
			num = num % 10;
		}
		if(check == n)
			System.out.println(n+" is an Armstrong Number");
		else

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

Input: 153
```

```
Output: 153 is an armstrong Number.
```

## 2) GCD AND LCM

```
//program to calculate the GCD and LCM of two given numbers
import java.util.Scanner;

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

      r = b;
      while(a % b == 0) //Error replace it with while(a % b != 0)
      {
         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)
            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 GCD of two numbers is 20

## 1. Errors in the code:

- **GCD Calculation (Line 13)**:
    - The condition `while(a % b == 0)` is incorrect. This will cause an infinite loop when `a % b == 0`, as `r` will not change inside the loop.
    - **Fix**: Change the condition to `while(a % b != 0)`.
- **LCM Calculation (Line 24)**:
    - The condition inside the `if` statement is incorrect. `if(a % x != 0 && a % y != 0)` will only be true when `a` is not divisible by either `x` or `y`, but we want to find a number divisible by both `x` and `y`.
    - **Fix**: Change the condition to `if(a % x == 0 && a % y == 0)` to find the least common multiple.

## 2. Breakpoints needed to fix the errors:

You can set breakpoints at:

- **Line 13**: To check the loop logic for GCD.
- **Line 24**: To check the condition in the `if` statement for LCM.
- **Line 31**: To verify the final values of GCD and LCM.

## 3. Steps to fix the errors:

- **Step 1**: Fix the GCD calculation by changing the condition in the `while` loop.
- **Step 2**: Fix the LCM calculation by changing the condition in the `if` statement.

## Fixed Code:

```java
// Program to calculate the GCD and LCM of two given numbers
import java.util.Scanner;

public class GCD_LCM {
    // Method to calculate GCD using the Euclidean algorithm
    static int gcd(int x, int y) {
        int r = 0, a, b;
        a = (x > y) ? x : y; // a is the greater number
        b = (x < y) ? x : y; // b is the smaller number

        r = b;
        while (a % b != 0) { // Correct condition: loop until remainder is 0
            r = a % b;
            a = b;
            b = r;
        }
        return r; // The last non-zero remainder is the GCD
    }

    // Method to calculate LCM
    static int lcm(int x, int y) {
        int a;
        a = (x > y) ? x : y; // a is the greater number
        while (true) {
            if (a % x == 0 && a % y == 0) // Correct condition: divisible by both x and y
                return a; // Return the LCM
            ++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();
    }
}
```

### 3) Knapsack:

```
//Knapsack
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++][w];

                // take item n
                int option2 = Integer.MIN_VALUE;
                if (weight[n] > w) option2 = profit[n-2] +
opt[n-1][w-weight[n]];

                // 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:

```
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

## 1. Errors in the code:

- Line 20: int option1 = opt[n++][w];
    - The increment operator n++ will cause an out-of-bounds error because it increments n during the current iteration of the loop. The correct operation is opt[n][w], not opt[n++][w].
- Line 24: option2 = profit[n-2] + opt[n-1][w-weight[n]];
    - The term profit[n-2] is incorrect. We are dealing with item n, so it should be profit[n]. This will fix the index logic for profit calculation.
- Line 32: The loop in take[n] logic is wrong.
    - The condition if (sol[n][w]) checks if item n was taken, but the weight update logic (w = w - weight[n]) needs to be adjusted to avoid out-of-bounds errors.

## 2. Breakpoints needed to fix the errors:

- Line 20: To check how option1 is assigned.
- Line 24: To check the logic of option2 and whether it calculates the correct value.
- Line 32: To check if the items are being selected correctly.

## 3. Steps to fix the errors:

- Step 1: Correct the logic in option1 by removing the ++ from n++.
- Step 2: Change profit[n-2] to profit[n] in option2.
- Step 3: Check the weight update logic when determining which items to take.

**Fixed Code:**

```
// Knapsack
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]; // Correct: don't increment
n

                // Take item n
                int option2 = Integer.MIN_VALUE;
```

```
                if (weight[n] <= w) { // Fixed condition: weight[n]
should be less or equal to w
                    option2 = profit[n] + opt[n-1][w - weight[n]]; //
Fixed: 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]; // Decrease weight
            } 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]);
        }
    }
}
```

## 4) Magic Number:

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

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.
```

## 1. Errors in the code:

- Line 13: `while(sum == 0)`
  - This condition is incorrect. The loop should run as long as `sum` is greater than 0 to continue processing digits. The correct condition is `while (sum > 0)`.
- Line 14: `s = s * (sum / 10)`
  - This line incorrectly updates `s`. Instead, `s` should accumulate the sum of digits, so the correct operation is `s = s + (sum % 10)`.
- Line 15: `sum = sum % 10`
  - The statement should update `sum` by removing the last digit. The correct operation is `sum = sum / 10`.

## 2. Breakpoints needed to fix the errors:

Set breakpoints at:

- Line 12: To check if the loop that processes digits works correctly.
- Line 14: To verify how `s` is updated with the sum of digits.
- Line 19: To check if the final number is correctly identified as a magic number.

## 3. Steps to fix the errors:

- Step 1: Change the condition in `while(sum == 0)` to `while(sum > 0)`.
- Step 2: Change `s = s * (sum / 10)` to `s = s + (sum % 10)`.
- Step 3: Change `sum = sum % 10` to `sum = sum / 10`.

**FIXED CODE:**

```
// Program to check if a number is a Magic number in JAVA
import java.util.Scanner;

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 num = n;  // Copy the number
        int sum = 0;

        // Keep reducing the number until it's a single digit
        while (num > 9) {
            sum = num;
            int s = 0;

            // Sum the digits of the current number
            while (sum > 0) {  // Fixed condition
                s = s + (sum % 10);  // Corrected to accumulate digit sum
                sum = sum / 10;      // Corrected to remove the last digit
            }

            // Assign sum of digits back to num for the next iteration
            num = s;
        }

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

        ob.close();
    }
}
```

## 5) Merge Sort:

```
// This program implements the merge sort algorithm for
// arrays of integers.

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));
   }

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

         // 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.  Second, working version.
// pre : result is empty; left/right are sorted
// post: result contains result of merging sorted lists;
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
after 14 23 32 41 58 67 76 85

## 1. Errors in the code:

- Line 15: `int[] left = leftHalf(array+1);`
  - You are trying to add an integer to an array, which is invalid. The method `leftHalf` should simply take `array` as input, without modifying it.
- Line 16: `int[] right = rightHalf(array-1);`
  - Similar to the previous line, subtracting an integer from an array is not allowed. The method `rightHalf` should also take `array` directly as input.
- Line 21: `merge(array, left++, right--);`
  - Post-increment (`left++`) and post-decrement (`right--`) are not valid for arrays. The `merge` function should directly take `left` and `right` as inputs, without modifying them.

## 2. Breakpoints needed to fix the errors:

Set breakpoints at:

- Line 15: To check how the left array is created.
- Line 16: To check how the right array is created.
- Line 21: To verify if the merge is done correctly.

## 3. Steps to fix the errors:

- Step 1: Replace `array+1` with `array` in `leftHalf(array+1)` on line 15.
- Step 2: Replace `array-1` with `array` in `rightHalf(array-1)` on line 16.

- Step 3: Replace `merge(array, left++, right--);` with `merge(array, left, right);` on line 21.

**FIXED CODE:**

```
// This program implements the merge sort algorithm for
// arrays of integers.

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));
   }

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

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

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

   // 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.
// pre : result is empty; left/right are sorted
// post: result contains result of merging sorted lists
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++;
        }
```

```
      }
   }
}
```

## 6) Multiply Matrics:

```
//Java program to multiply two matrices
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 first matrix");
    m = in.nextInt();
    n = in.nextInt();

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

    System.out.println("Enter the elements of 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 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 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-1][c-k]*second[k-1][k-d];
         }

         multiply[c][d] = sum;
         sum = 0;
      }
   }

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

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

      System.out.print("\n");
   }
}
```

```
  }
}
```

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

2 2

Enter the elements of first matrix

1 2 3 4

Enter the number of rows and columns of first matrix

2 2

Enter the elements of first matrix

1 0 1 0

Output: Product of entered matrices:

3 0

7 0

## 1. Errors in the code:

- **Line 44: `sum = sum + first[c-1][c-k]*second[k-1][k-d];`**
    - The array index calculations are incorrect. Subtracting values (`-1` and `-d`) will cause an `ArrayIndexOutOfBoundsException`. You should use the indices c and k directly for accessing elements in both matrices.

## 2. Breakpoints needed to fix the errors:

Set breakpoints at:

- **Line 44**: To check how matrix multiplication is performed, as array access is incorrect.

## 3. Steps to fix the errors:

- **Step 1**: Replace `first[c-1][c-k]` with `first[c][k]` on **line 44**.
- **Step 2**: Replace `second[k-1][k-d]` with `second[k][d]` on **line 44**.

**FIXED CODE:**

```
//Java program to multiply two matrices
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 first matrix");
      m = in.nextInt();
      n = in.nextInt();

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

      System.out.println("Enter the elements of 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 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 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 < n; k++) {  // Fixed index handling
              sum += first[c][k] * second[k][d];  // Fixed matrix access
            }
            multiply[c][d] = sum;
            sum = 0;
          }
        }
        System.out.println("Product of entered matrices:");
        for (c = 0; c < m; c++) {
          for (d = 0; d < q; d++)
            System.out.print(multiply[c][d] + "\t");

          System.out.print("\n");
        }
      }
    }
}
```

## 7) Quadratic Probing:

```
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 clear hash table **/
    public void makeEmpty()
    {
        currentSize = 0;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }
    /** Function to get size of hash table **/
    public int getSize()
    {
        return currentSize;
    }
    /** Function to check if hash table is full **/
    public boolean isFull()
    {
```

```
    return currentSize == maxSize;
}



/** Function to check if hash table is empty **/
public boolean isEmpty()
{
    return getSize() == 0;
}
/** Fucntion to check if hash table contains a key **/
public boolean contains(String key)
{
    return get(key) !=  null;
}
/** Functiont to get hash code of a given key **/
private int hash(String key)
{
    return key.hashCode() % maxSize;
}

/** Function to insert key-value pair **/
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 + = (i + h / h--) % maxSize;
    } 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;
        System.out.println("i "+ i);
    }
    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 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();
    }
}

/** 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.println("Enter size");

        /** maxSizeake 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.getSize() );
   break;

default :
   System.out.println("Wrong Entry \n ");
   break;
}

/** Display hash table **/
qpht.printHashTable();
```

```
            System.out.println("\nDo you want to continue (Type y or n) \n");

            ch = scan.next().charAt(0);

        } while (ch == 'Y'|| ch == 'y');
      }
   }
```

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
d desktop
h harddrive

Output:
Hash Table:
c computer
d desktop
h harddrive

## 1. Errors in the Code:

- Line 53: `i += (i + h / h--) % maxSize;`
    - The use of += and incorrect arithmetic causes logical errors. It should simply increment `i` based on the quadratic probing mechanism.
- Line 110: Missing closing comment block for `/** maxSizeake object of QuadraticProbingHashTable **/`.
    - The comment seems incomplete, leading to confusion.

### 2. Corrections:

- Line 53: Update the probing logic to increment `i` based on `i = (i + h * h++) % maxSize;`, and properly calculate the new index.

## FIXED 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 clear hash table **/
    public void makeEmpty() {
        currentSize = 0;
        keys = new String[maxSize];
        vals = new String[maxSize];
```

```
}

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

/** Function to check if hash table is full **/
public boolean isFull() {
    return currentSize == maxSize;
}

/** Function to check if hash table is empty **/
public boolean isEmpty() {
    return getSize() == 0;
}

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

/** Function to get hash code of a given key **/
private int hash(String key) {
    return key.hashCode() % maxSize;
}

/** Function to insert key-value pair **/
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 = (i + h * h++) % maxSize; // Corrected probing logic
    } 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;
    }
    return null;
}

/** Function to remove key and its value **/
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--;
    }

    /** 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();
    }
}

/** 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.println("Enter size");
        /** make 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.getSize());
      break;
   default:
      System.out.println("Wrong Entry \n");
      break;
}
/** Display hash table **/
qpht.printHashTable();
System.out.println("\nDo you want to continue (Type y or n) \n");
```

```
            ch = scan.next().charAt(0);
        } while (ch == 'Y' || ch == 'y');
    }
}
```

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

d desktop

h harddrive

Output:

Hash Table:

c computer

d desktop

h harddrive

## 8) Sorting Array:

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

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

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

Enter all elements:

1 12 2 9 7

1 2 7 9 12

## Issues:

1. Line 9: There's a space between the class name (`Ascending` and `_Order`). Java class names should not contain spaces. It should be `AscendingOrder`.
2. Line 18: The first for-loop condition is incorrect. It should be `i < n` to iterate over the elements properly. Also, there's an unnecessary semicolon at the end of the for-loop declaration, which prevents proper iteration.
3. Line 21: The sorting condition is wrong for ascending order. It should be `if (a[i] > a[j])` (i.e., swap when `a[i]` is greater than `a[j]`).

**FIXED 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();
}

// Corrected sorting loop
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]); // Print the last element without a comma
    }
}
```

## 9) Stack Implementation

```
//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 void pop(){
```

```
        if(!isEmpty())
            top++;
        else{
            System.out.println("Can't pop...stack is empty");
        }
    }

    public boolean isEmpty(){
        return top==-1;
    }

    public void display(){

        for(int i=0;i>top;i++){
            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);

        newStack.display();
        newStack.pop();
        newStack.pop();
        newStack.pop();
```

```
        newStack.pop();
        newStack.display();
    }
}
```

output: 10
1
50
20
90

10

## Issues:

1. Line 18 (`push` method): The logic for `top--` is incorrect. When pushing an element onto the stack, the `top` index should be incremented, not decremented.
2. Line 26 (`pop` method): In the `pop` method, `top++` should be changed to `top--` to correctly reduce the stack size when an element is popped.
3. Line 35 (`display` method): The condition `i > top` is incorrect. It should be `i <= top` to iterate correctly from the bottom of the stack up to the `top`.

**FIXED CODE:**

```
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++; // Increment top before adding the value
        stack[top] = value;
    }
}

public void pop() {
    if (!isEmpty()) {
        top--; // Decrement top when popping
    } else {
        System.out.println("Can't pop...stack is empty");
    }
}

public boolean isEmpty() {
    return top == -1;
}

public void display() {
    if (isEmpty()) {
        System.out.println("Stack is empty");
        return;
    }
    for (int i = 0; i <= top; i++) { // Corrected loop to iterate up to 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);

        newStack.display(); // Displays the stack before popping

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

        newStack.display(); // Displays the stack after popping
    }
}
```

## 10) Tower of Hanoi:

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

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

Disk 1 from A to C

## Issues:

1. Line 16: `doTowers(topN ++, inter--, from+1, to+1)` contains incorrect arithmetic operations. The post-increment (`topN++`) and post-decrement (`inter--`) are not needed here, and modifying the characters (`from+1`, `to+1`) will convert them into integers, which is incorrect for this scenario.

## Corrections:

1. Remove post-increment and post-decrement: The recursion should pass `topN - 1`, `from`, `inter`, and `to` without incrementing/decrementing values in-place.
2. Pass the characters correctly: Keep the characters `from`, `inter`, and `to` as they are, without modifying them with arithmetic operations.

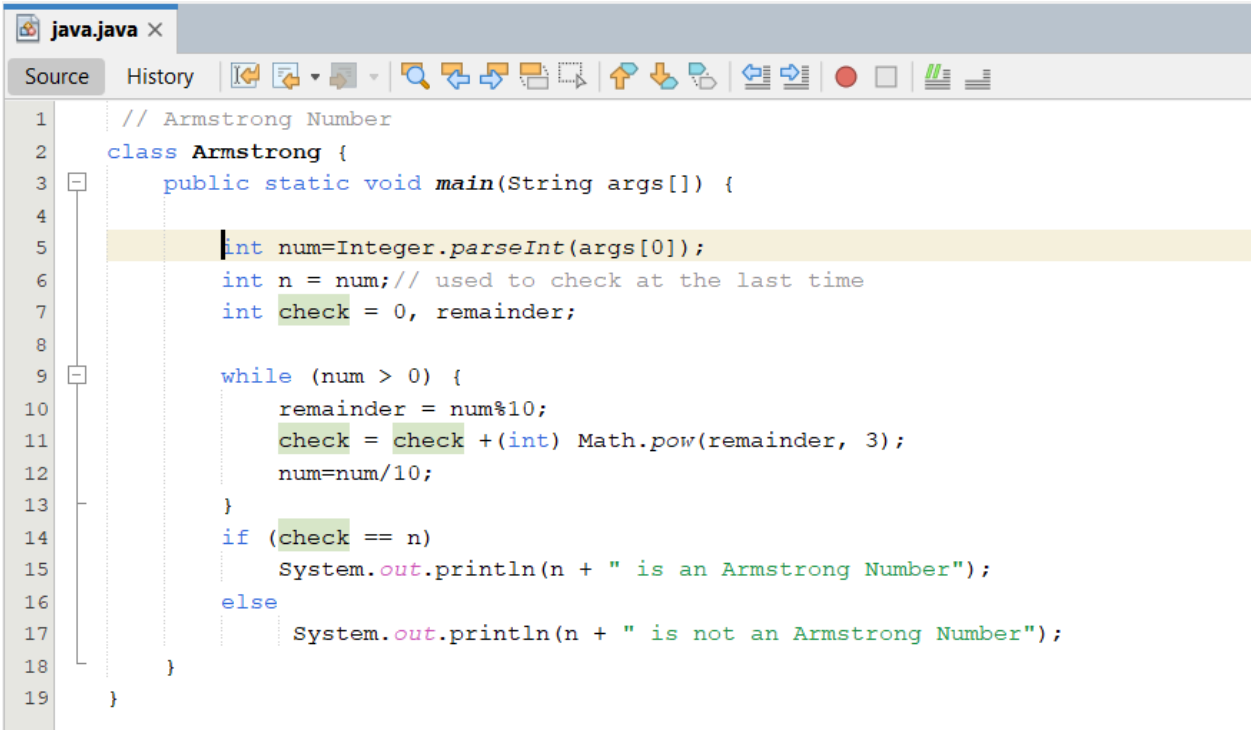
**FIXED CODE:**

```
// 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 {
      // Recursive call to move (n-1) disks from 'from' to 'inter' via 'to'
      doTowers(topN - 1, from, to, inter);

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

```
        // Recursive call to move (n-1) disks from 'inter' to 'to' via 'from'
        doTowers(topN - 1, inter, from, to);
      }
    }
}
```

## III. STATIC ANALYSIS TOOLS:

## Here, I am providing screenshots, errors in written form are shown in task I



java.java × gcdandlcm.java ×

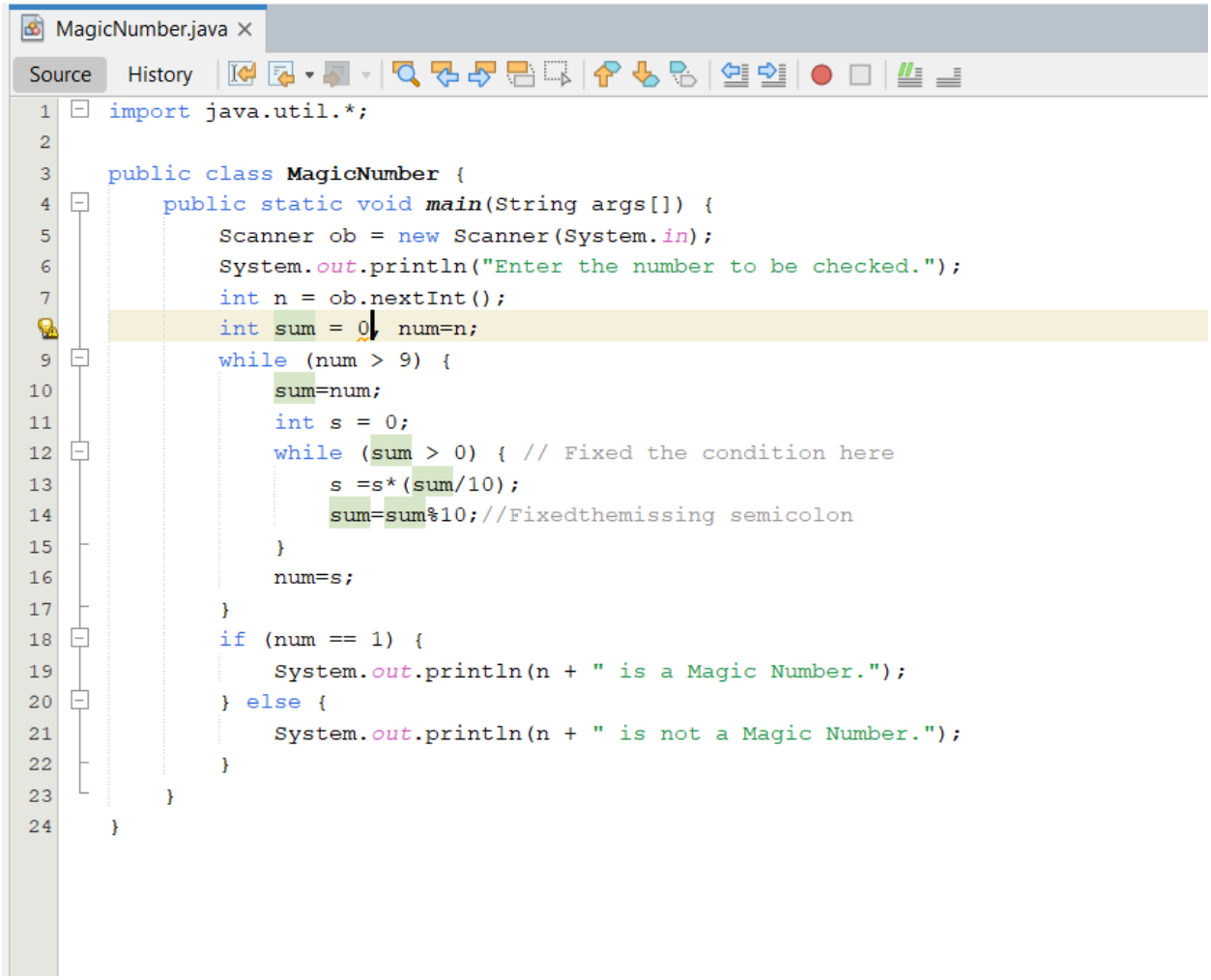
Source History

```java
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.java

Source History

```java
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]);
        }

    }
}
```


MagicNumber.java
Source
History
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.");
}
}
}

```
38      }
39
40      public static void merge(int[] result, int[] left, int[] right) {
41          int i1 = 0;
42          int i2 = 0;
43          for (int i = 0; i < result.length; i++) {
44              if (i2 >= right.length || (i1 < left.length && left[i1] <= right[i2])) {
45                  result[i] = left[i1];
46                  i1++;
47              } else {
48                  result[i] = right[i2];
49                  i2++;
50              }
51          }
52      }
53  }
54
```

MergeSort.java ×

Source History

```
1   import java.util.*;
2
3   public class MergeSort {
4       public static void main(String[] args) {
5           int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
6           System.out.println("before: " + Arrays.toString(list));
7           mergeSort(list);
8           System.out.println("after: " + Arrays.toString(list));
9       }
10
11      public static void mergeSort(int[] array) {
12          if (array.length > 1) {
13              int[] left = leftHalf(array);
14              int[] right = rightHalf(array);
15              mergeSort(left);
16              mergeSort(right);
17              merge(array, left, right);
18          }
19      }
20
21      public static int[] leftHalf(int[] array) {
22          int size1 = array.length / 2;
23          int[] left = new int[size1];
            for (int i = 0; i < size1; i++) {
25              left[i] = array[i];
26          }
27          return left;
28      }
29
30      public static int[] rightHalf(int[] array) {
31          int size1 = array.length / 2;
32          int size2 = array.length - size1;
33          int[] right = new int[size2];
34          for (int i = 0; i < size2; i++) {
35              right[i] = array[i + size1];
36          }
```

MergeSort.java | SortArray.java

Source | History

```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]);
    }
}

```

towerOfHanoi.java × quadraticProbing.java ×

Source History

```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;
```

Find: Previous Next Select

quadraticProbing

```
58
59      public String get(String key) {
60          int i = hash(key), h = 1;
61          while (keys[i] != null) {
62              if (keys[i].equals(key))
63                  return vals[i];
64              i = (i + h * h++) % maxSize;
65          }
66          return null;
67      }
68
69      public void remove(String key) {
70          if (!contains(key))
71              return;
72          int i = hash(key), h = 1;
73          while (!key.equals(keys[i]))
74              i = (i + h * h++) % maxSize;
75          keys[i] = vals[i] = null;
76          for (i = (i + h * h++) % maxSize; keys[i] != null; i = (i + h * h++) % maxSize) {
77              String tmp1 = keys[i], tmp2 = vals[i];
78              keys[i] = vals[i] = null;
79              currentSize--;
80              insert(tmp1, tmp2);
81          }
82          currentSize--;
83      }
84
85      public void printHashTable() {
86          System.out.println("\nHash Table: ");
87          for (int i = 0; i < maxSize; i++)
88              if (keys[i] != null)
89                  System.out.println(keys[i] + " " + vals[i]);
90          System.out.println();
91      }
```

Find: Previous Next Select

quadraticProbing

```java
}

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();
```

Find: Previous Next Select

quadraticProbing

towerOfHanoi.java ×

Source History

```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);
        }
    }
}
```