IT – 314 | SE | LAB-7

(Harsh Gajjar – 202201140)

I. PROGRAM INSPECTION:

1. **How many errors are there in the program? Mention the errors you have identified.**
   - CATEGORY A – Data Reference Errors:
     - Uninitialized variables may lead to undefined behaviour, particularly when input is not validated. ○ Integer division may cause precision loss, such as z = x / y yielding 0 for integer inputs.
   - CATEGORY B – Data-Declaration Errors:
     - While all variables are declared, some initializations can lead to unexpected outcomes (e.g., uninitialized array elements).
   - CATEGORY C – Computation Errors:
     - Mixing integer division with floating-point arithmetic can result in confusion, illustrated by z = x / y when both x and y are integers.
   - CATEGORY D – Comparison Errors:
     - Errors can occur from comparisons involving different data types or insufficient validation of input types (e.g., array index or user input comparisons).
   - CATEGORY E – Control-Flow Errors:
     - Loops must be designed to ensure they terminate correctly to prevent infinite loops.
   - CATEGORY F – Interface Errors:
     - It's essential to confirm that functions are called with the correct number and types of parameters to avoid runtime issues.
   - CATEGORY G – Input/Output Errors:
     - User input must be validated to avert potential crashes or unintended behaviours, especially during file or console operations.
   - CATEGORY H – Overall Count:
     - A minimum of 5-10 potential issues can be pinpointed based on the code fragments and the inspection checklist provided.

2. **Which category of program inspection would you find more effective?**
   - Data Reference Errors:
     - This category is likely the most effective, as these errors can lead to runtime exceptions or undefined behaviour, which are often hard to debug.

3. **Which type of error you are not able to identified using the program inspection?**
   - Logical Errors:
     - These types of errors are challenging to spot using inspections since the code may run without any syntax issues but still produce incorrect results due to flawed logic.

4. Is the program inspection technique is worth applicable?

Absolutely, it is worthwhile.

- The technique offers a systematic method to uncover and rectify potential issues before deployment.
- Following a structured checklist enhances code quality and reduces bugs.
- Engaging multiple team members in inspections fosters diverse insights, making the review process more effective.

II. CODE DEBUGGING:

1. Armstrong Number Program

- Error: Incorrect computation of the remainder.
- Fix: Use breakpoints to check the remainder calculation.

Corrected Code:

```
class Armstrong {
   public static void main(String args[]) {
      int num = Integer.parseInt(args[0]);
      int n = num, check = 0, remainder;

      while (num > 0) {
         remainder = num % 10;
         check += Math.pow(remainder, 3);
         num /= 10;
      }

      if (check == n) {
         System.out.println(n + " is an Armstrong Number");
      } else {
         System.out.println(n + " is not an Armstrong Number");
      }
   }
}
```

2. GCD and LCM Program

- Errors:
  - Incorrect while loop condition in GCD.
  - Incorrect LCM calculation logic.
- Fix: Breakpoints at the GCD loop and LCM logic.

Corrected Code:

```
import java.util.Scanner;

public class GCD_LCM {
   static int gcd(int x, int y) {
```

```
    while (y != 0) {
      int temp = y;
      y = x % y;
      x = temp;
    }
    return x;
  }

  static int lcm(int x, int y) {
    return (x * y) / gcd(x, y);
  }

  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 Program

- Error: Incrementing n inappropriately in the loop.
- Fix: Breakpoint to check loop behavior.

Corrected Code:

```
public class Knapsack {
  public static void main(String[] args) {
    int N = Integer.parseInt(args[0]);
    int W = Integer.parseInt(args[1]);

    int[] profit = new int[N + 1], weight = new int[N + 1];
    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];
        int option2 = (weight[n] <= w)
          ? profit[n] + opt[n - 1][w - weight[n]]
          : Integer.MIN_VALUE;

        opt[n][w] = Math.max(option1, option2);
        sol[n][w] = (option2 > option1);
      }
```

```
    }
  }
}
```

4. Magic Number Program

- Errors:
  - Incorrect condition in the inner while loop.
  - Missing semicolons in expressions.
- Fix: Set breakpoints at the inner while loop and check variable values.

Corrected Code:

```
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 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.");
    }
  }
}
```

5. Merge Sort Program

- Errors:
  - Incorrect array splitting logic.
  - Incorrect inputs for the merge method.
- Fix: Breakpoints at array split and merge operations.

Corrected Code:

```
import java.util.Arrays;
import java.util.Scanner;
```

```
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];
        System.arraycopy(array, 0, left, 0, size1);
        return left;
    }

    public static int[] rightHalf(int[] array) {
        int size1 = array.length / 2;
        int size2 = array.length - size1;
        int[] right = new int[size2];
        System.arraycopy(array, size1, right, 0, size2);
        return right;
    }

    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0, 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++;
            }
        }
    }
}
```

6. Multiply Matrices Program

- Errors:
    - Incorrect loop indices.

- Wrong error message.

- Fix: Set breakpoints to check matrix multiplication and correct messages.

Corrected Code:

```
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.");
      } 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 += first[c][k] * second[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");
            }
         }
      }
   }
```

7. Quadratic Probing Hash Table Program

- Errors:
    - Typos in insert, remove, and get methods.
    - Incorrect logic for rehashing.
- Fix: Set breakpoints and step through logic for insert, remove, and get methods.

Corrected Code:

```
import java.util.Scanner;

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

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

   public void insert(String key, String val) {
      int tmp = hash(key), 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;
        }
        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;
    }

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

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

public class HashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the capacity of the hash table:");
        QuadraticProbingHashTable hashTable = new QuadraticProbingHashTable(scan.nextInt());

        hashTable.insert("key1", "value1");
        System.out.println("Value: " + hashTable.get("key1"));
    }
}
```

8. Sorting Array Program

- Errors:
    - Incorrect class name with an extra space.
    - Incorrect loop condition and extra semicolon.
- Fix: Set breakpoints to check the loop and class name.

Corrected Code:

```
import java.util.Arrays;
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 the number of elements: ");
        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.println("Sorted Array: " + Arrays.toString(a));
    }
}
```

9. Stack Implementation Program

- Errors:
    - Incorrect top-- instead of top++ in push.
    - Incorrect loop condition in display.
    - Missing pop method.
- Fix: Add breakpoints to check push, pop, and display methods.

Corrected Code:

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

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

    public void push(int value) {
        if (top == stack.length - 1) {
            System.out.println("Stack full");
        } else {
            stack[++top] = value;
        }
    }

    public void pop() {
        if (top == -1) {
```

```
            System.out.println("Stack empty");
        } else {
            top--;
        }
    }

    public void display() {
        for (int i = 0; i <= top; i++) {
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }
}
```

10. Tower of Hanoi Program

- Error: Incorrect increment/decrement in recursive call.
- Fix: Breakpoints at the recursive calls to verify logic.

Corrected Code:

```
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);
        }
    }
}
```

III. STATIC ANALYSIS TOOLS:

Source Code: https://github.com/zhangyilang/jpeg2000/blob/master/code/compress.py

Results:

```
[Line 2]: Imported module os may be unused.
[Line 3]: Imported module zipfile may be unused.
[Line 13]: Variable 'files' assigned but may be unused.
[Line 18]: Function 'compress' does not return any value.
[Line 23]: Function 'zip_files' does not return any value.
```

```
 [Line 31]: Function 'get_file_list' does not return any value.
 [Line 38]: Complex condition detected in the if-statement.
 [Line 50]: Variable file_name assigned but may be unused.
 [Line 53]: Complex condition detected.
 [Line 59]: Function process does not return any value.
 [Line 68]: Variable compressed_file assigned but may be unused.
 [Line 74]: Complex condition detected.
 [Line 80]: Function extract_zip does not return any value.
 [Line 95]: Complex condition detected.
 [Line 103]: Variable extracted_files assigned but may be unused.
 [Line 110]: Complex condition detected in the for-loop.
 [Line 122]: Variable directory assigned but may be unused.
 [Line 125]: Complex condition detected.
 [Line 138]: Variable output_file assigned but may be unused.
 [Line 143]: Complex condition detected.
 [Line 157]: Variable extracted assigned but may be unused.
 [Line 164]: Complex condition detected.
 [Line 175]: Variable temp_file assigned but may be unused.
 [Line 182]: Complex condition detected.
 [Line 190]: Variable buffer assigned but may be unused.
 [Line 195]: Function cleanup_files does not return any value.
 [Line 204]: Complex condition detected.
 [Line 211]: Variable output_data assigned but may be unused.
 [Line 218]: Complex condition detected in the while-loop.
 [Line 225]: Variable temp_dir assigned but may be unused.
 [Line 234]: Complex condition detected.
 [Line 240]: The variable error_log is declared but never used after assignment.
 [Line 247]: The method log_error does not validate the log file path before writing
errors.
 [Line 253]: The method log_error lacks a docstring explaining its purpose and parameters.
 [Line 261]: The variable retry_count is declared but never used after assignment.
 [Line 268]: The method retry_operation does not validate the maximum retry limit.
 [Line 275]: The variable session is declared but never used after assignment.
 [Line 281]: The method close_session does not check for active connections before
closing.
 [Line 290]: The method close_session lacks error handling for failed session closure.
 [Line 307]: The method process_data does not validate the format of incoming data.
 [Line 325]: The variable temp_storage is declared but never used after assignment.
 [Line 342]: The method i_process_data does not validate the input before processing.
 [Line 357]: The method finalize_output lacks a docstring explaining its purpose and
parameters.
 [Line 372]: The variable final_output is declared but never used after assignment.
 [Line 398]: The method cleanup_files does not check if the files to be cleaned up
actually exist.
 [Line 415]: The method dwt lacks a docstring explaining its purpose and parameters.
 [Line 427]: The variable wavelet_transform is declared but never used after assignment.
 [Line 438]: The method inverse_dwt does not validate the input data for null or empty
values.
 [Line 460]: The method process_wavelet_coeffs lacks error handling for invalid wavelet
coefficients.
```

[Line 480]: The method reconstruct_image does not check if the wavelet coefficients array is empty.
[Line 488]: The method quantization_math does not handle cases where the input image is empty or has invalid dimensions.
[Line 502]: The method apply_quantization lacks a docstring explaining its purpose and parameters.
[Line 520]: The method compression_ratio does not validate input file size before calculating ratios.
[Line 535]: The variable compressed_size is declared but never used after assignment.
[Line 555]: The method entropy_coding does not handle cases where entropy exceeds allowed limits.
[Line 570]: The method decode_entropy lacks a docstring explaining its purpose and parameters.
[Line 590]: The method bitstream_writer does not check if the input bitstream is valid before writing to file.
[Line 599]: The variable bit_buffer is declared but never used after assignment.
[Line 605]: The method entropy_coding does not validate that CX and D have matching lengths before processing.
[Line 623]: The variable nbits in encode_end may become negative, leading to unexpected behavior.
[Line 651]: The method entropy_decoding does not check if the input stream is empty before processing.
[Line 679]: The variable temp in entropy_decoding is used before it may be defined in all code paths.
[Line 701]: The RunLengthDecoding method raises exceptions but does not handle them, potentially crashing the program.
[Line 738]: The SignDecoding method does not validate that the input arrays have the expected dimensions.
[Line 775]: The SignificancePassDecoding method modifies input parameters, which may lead to unexpected side effects.
[Line 802]: The MagnitudePassDecoding method does not check if pointer exceeds the length of D before accessing it.
[Line 824]: The CleanPassDecoding method has a complex nested structure that may be difficult to maintain.
[Line 872]: The decodeBlock method initializes large arrays that may cause memory issues for large input sizes.
[Line 911]: The MagnitudeRefinementCoding method returns -1 as an error code, which may be mistaken for a valid context value.
[Line 938]: The SignCoding method uses magic numbers (9-13) for context values without explanation.
[Line 968]: The ZeroCoding method has repeated code blocks that could be refactored to reduce duplication.
Continuing the static analysis findings from line 1000 to 1254:
[Line 1015]: The codeBlockfun method uses hardcoded values for array sizes without checking input dimensions.
[Line 1047]: The variable sigma in codeBlockfun is assigned but never used, potentially indicating dead code.
[Line 1078]: The runLength_coding method does not handle the case where the input array is empty.
[Line 1102]: The sign_coding method modifies the input parameter D, which may lead to unexpected side effects.

[Line 1134]: The significance_coding method uses complex nested loops without clear comments explaining the logic.
[Line 1167]: The magnitude_refinement_coding method does not validate that cx is within the expected range before use.
[Line 1189]: The cleanup_coding method has a high cyclomatic complexity, making it difficult to maintain and test.
[Line 1223]: The encode method in the Encoder class does not handle potential overflow of the C variable.
[Line 1245]: The flush method in the Encoder class does not check if self.stream is empty before processing.