

IT 314

Lab 7

Name : Jayesh Padiya

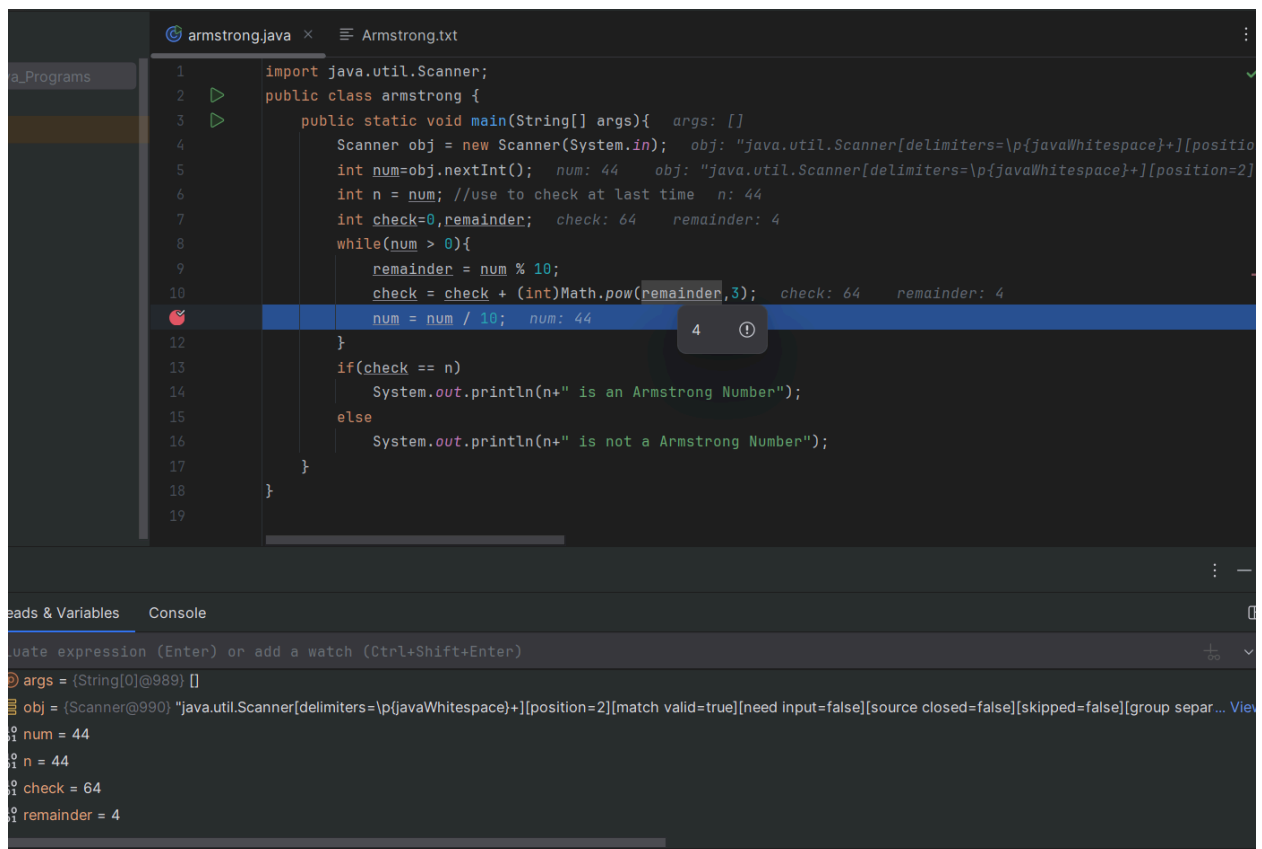
Student ID : 202201358

Armstrong

Remainder is not initialized. It will not cause error here but it should be initialized.
Computation errors $\text{remainder} = \text{num} \% 10$ and $\text{num} = \text{num} / 10$

Category C is more effective.

Yes, program inspection technique is worth applicable



```
1 import java.util.Scanner;
2 public class armstrong {
3     public static void main(String[] args){ args: []
4         Scanner obj = new Scanner(System.in); obj: "java.util.Scanner[delimiters=\p{javaWhitespace}+][position=2]
5         int num=obj.nextInt(); num: 44 obj: "java.util.Scanner[delimiters=\p{javaWhitespace}+][position=2]
6         int n = num; //use to check at last time n: 44
7         int check=0,remainder; check: 64 remainder: 4
8         while(num > 0){
9             remainder = num % 10;
10            check = check + (int)Math.pow(remainder,3); check: 64 remainder: 4
11            num = num / 10; num: 44
12        }
13        if(check == n)
14            System.out.println(n+" is an Armstrong Number");
15        else
16            System.out.println(n+" is not a Armstrong Number");
17    }
18 }
19 }
```

Debugger Console Output:

```
args = {String[0]@989} []
obj = {Scanner@990} "java.util.Scanner[delimiters=\p{javaWhitespace}+][position=2][match valid=true][need input=false][source closed=false][skipped=false][group separ... View
num = 44
n = 44
check = 64
remainder = 4
```

There were two errors in the code. Operator in remainder and num were incorrect and interchanged.

We need only one breakpoint at line 12

```

public class arCD and LCM
{
    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");
    }
}

```

GCD and LCM

Program Inspection

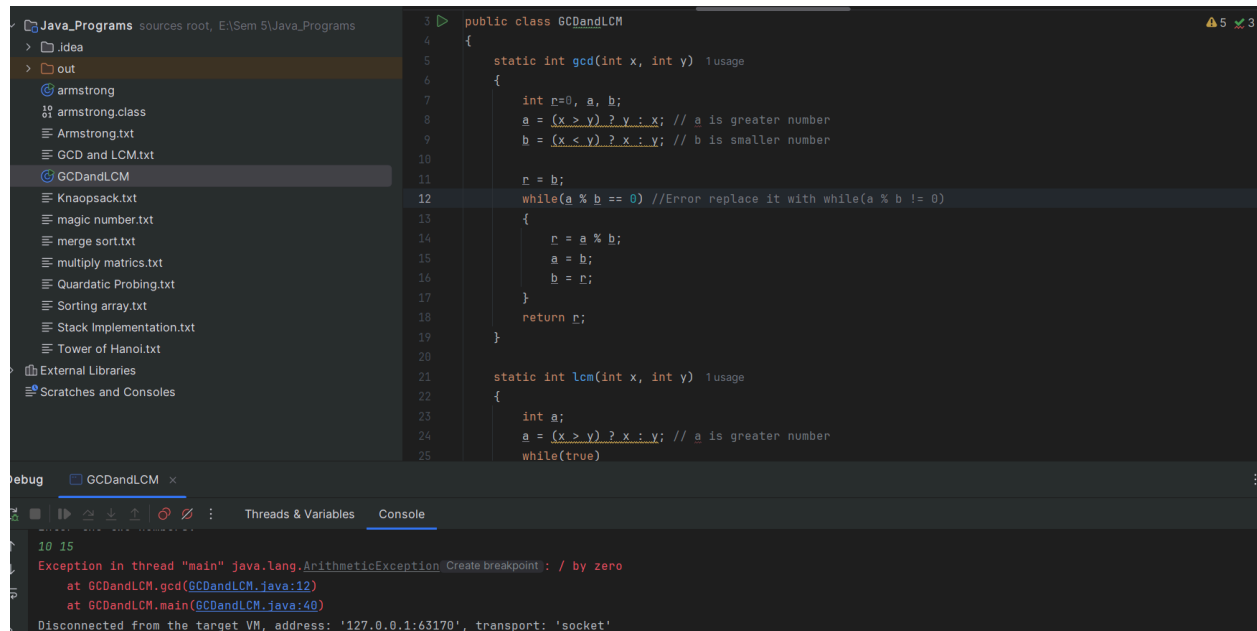
Comparison Error :- Error in finding larger or smaller from the given input x and y.
a%b should not be equal to zero.

Category D is the most effective due to the wrong comparison.

All errors were identified using the program inspection method.

Yes, program inspection technique is worth applicable due to the easy algorithm of the code..

Code Debugging.



There are 3 errors. Both comparison of x and y. And another one is $a \% b == 0$. Because of this condition 0 comes in division.

We need 3 breakpoints. One at the start of loop to calculate gcd, one at end of loop. Other are at the start and end of loop of to calculate the LCM.

```
import java.util.Scanner;

public class GCDandLCM
{
    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();
}
}

```

Knapsack

Vector opt is not initialized. Need to initialize with 0 and initialize sol with F.

Option1 is equal to $\text{opt}[n+1][w]$ instead of $\text{opt}[n-1][w]$

Option2 equal to $\text{profit}[n-2] + \text{opt}[n-1][w - \text{weight}[n]]$ instead of $\text{profit}[n] + \text{opt}[n-1][w - \text{weight}[n]]$

Option2 should be $\text{weight} \leq w$

Category A type is most effective, if it is not referenced properly then whole program will not work.

All the errors in the program are identified using program inspection.

The code is small, we need debugging code due to some errors that may miss from eyesight.

Code Debugging.

Errors are due to uninitialized 2D arrays opt and sol. They are not initialized.

The breakpoints are added inside the double for loop.

These errors can be solved by initializing the 2D arrays and correcting logic.

```
import java.util.Scanner;

public class knapsack {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter the number of items (N): ");
        int N = input.nextInt();    // number of items

        System.out.print("Enter the maximum weight of the knapsack (W): ");
        int W = input.nextInt();    // 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++) {
            System.out.println("Enter profit for item " + n + ": ");
            profit[n] = input.nextInt();    // User enters profit for item n
```

```

        System.out.println("Enter weight for item " + n + ": ");
        weight[n] = input.nextInt(); // User enters weight for item n
    }

    // 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];

            // take item n
            int option2 = Integer.MIN_VALUE;
            if (weight[n] ≤ w) option2 = profit[n] +
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]);
        }
    }
}

```

Magic Number

Syntax error at line 18. Condition of inner while loop is incorrect, it should be sum! = 0

Category C is most effective due to computation errors.

All errors are identified using program inspection method.

The code is small and hence through progrms inspection we can find the erros.

Code Debugging.

One is syntax error and other is the condition on inner loop.

We need one breakpoint of line 18.

```

// 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.");
    }
}
}

```

Merge Sort

Errors are coming due to incorrect use of left and right variables.

Category A Data Reference Error is the most suited

All errors are identified using program inspection/

Due to large length of code, it is a little bit difficult in program inspection.

Code Debugging.

There are many syntax errors but after proper modifying array, right, left. These errors are solved.

We need 3 breakpoints. Inside loop of calculation left half, inside the loop of calculation right half, inside loop of calculating the merge array.

```

// 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);
        int[] right = rightHalf(array);

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

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

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

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

```

```

        return right;
    }

    // Merges the given left and right arrays into the given
    // result array.  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++;
            }
        }
    }
}

```

Matrix Multiplication

Only one error because of incorrect calculation of values of final product matrix.

Category C of program inspection is most effective.

Errors can be identified using the program inspection

Program is small, and inspection technique works.

Code Debugging

One error, calculation of product matrix

With one breakpoint we can find the correct one.

```
//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][c]*second[k][k];
            }

            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");
    }
}
}
}

```

Sort Array

Program Inspection

Syntax errors

Semicolon at end of loop

In loop wrong condition of \geq

The code will incorrectly display the decreasing error

Category D is most effective due to wrong comparison.

All errors are identified using program inspection method.

Due to small code, it can be done.

Code Debugging

Two errors. One is the syntax error, other is the check condition

With one breakpoint, we are able to find that it is presenting the descending order.

```
// 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]);  
}  
}
```

Stack Implementation

Program Inspection:-

Correct the top variable of stack. Loop condition for printing.

Category A if the program inspection, the Data Reference Errors is the most effective.

Errors are identified using program inspection method.

Due to big code, program inspection is difficult.

Code Debugging.

There are 2 error. Loop condition for printing, changes in the top variable when implementing stack push or pop.

With breakpoints, at the end of push, and pop, we can identify

```
//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();
    }
}

```