IT314 SOFTWARE ENGINEERING

LAB – 7 REPORT

LATHIGARA PRIYANSH – 202201449

PART – I: PROGRAM INSPECTION

CODE-1 (Magic text/numbers display)

1. **Errors Identified**:
    - **Category A (Data Reference Error)**:
        - cin.getline(str100) is incorrect, should be cin.getline(str, 100).
    - **Category C (Computation Error)**:
        - No specific computation error is visible.
    - **Category D (Comparison Error)**:
        - None
    - **Category E (Control-Flow Error)**:
        - The switch case should include a default case to handle unexpected inputs.
    - **Category G (I/O Error)**:
        - Redundant output with cout not structured optimally.
2. **Effective Category for Inspection**:
    - **Category A (Data Reference Errors)** for identifying issues in the cin.getline and ensuring proper input handling.
3. **Errors not identified by inspection**:
    - No GUI, memory, or file handling errors to identify.
4. **Applicability of Program Inspection**:
    - **Yes**, effective for catching data handling errors, but logic and efficiency issues would benefit from deeper testing.

REFRENCE: LINK

CODE-2 (Login and Registration System)

1. **Errors Identified**:
    - **Category A (Data Reference Error)**:
        - Variables like username password are missing a comma between them.
    - **Category F (Interface Error)**:
        - Inconsistent handling of the data file (data.txt) in various functions.
    - **Category G (I/O Error)**:
        - The ifstream and ofstream streams are not checked for proper opening.
2. **Effective Category for Inspection**:
    - **Category A (Data Reference Errors)**, especially around variable handling, initialization, and ifstream operations.
3. **Errors not identified by inspection**:
    - Complex logical flow errors in user management aren't easily detected.
4. **Applicability of Program Inspection**:
    - **Yes**, helps detect data structure inconsistencies and missing error-handling in file operations.

REFRENCE: LINK

CODE-3 (Hotel Management System)

1. **Errors Identified**:
    - **Category B (Data Declaration Error)**:
        - The size of phone number (char phone[10];) is not enough to store a standard phone number with formatting.
    - **Category F (Interface Error)**:
        - Issues with file handling like fout.write((char*)thissizeof(hotel)) which contains syntax errors.
    - **Category G (I/O Error)**:
        - File handling lacks error checking before reading or writing.
2. **Effective Category for Inspection**:
    - **Category G (I/O Errors)**, especially regarding file handling in add() and bill().
3. **Errors not identified by inspection**:
    - Logical flow errors in customer record handling may not be fully addressed with program inspection alone.
4. **Applicability of Program Inspection**:
    - **Yes**, program inspection helps in identifying memory and file handling problems but needs enhancement for logical flow checks.

REFRENCE: LINK

CODE-4 (Supermarket Billing System)

1. **Errors Identified**:
    - **Category B (Data Declaration Error)**:
        - Declaration of int k=7r=0flag=0; is incorrect due to missing commas between variables.
    - **Category E (Control-Flow Error)**:
        - Excessive use of goto statements makes control flow confusing and hard to follow.
    - **Category G (I/O Error)**:
        - Issues with fstream use, such as improper error checking for file openings.
2. **Effective Category for Inspection**:
    - **Category G (I/O Errors)**, which dominates in this code due to extensive file operations for the billing system.
3. **Errors not identified by inspection**:
    - Some logical flow errors, like improper use of goto, are harder to detect with inspection alone.
4. **Applicability of Program Inspection**:
    - **Yes**, critical for catching file handling errors, but deeper structural checks are needed for control flow.

REFRENCE: LINK

CODE-5 (GUI Calculator)

1. **Errors Identified**:
    - **Category B (Data Declaration Error)**:
        - Enum declarations like enum bool{falsetrue}; miss commas and proper formatting.
    - **Category E (Control-Flow Error)**:
        - Mouse event handling could lead to boundary condition failures, causing program crashes.
    - **Category G (I/O Error)**:
        - Lack of checks for the mouse pointer location and boundary limits may lead to runtime failures.
2. **Effective Category for Inspection**:
    - **Category E (Control-Flow Errors)**, particularly in GUI systems where flow control between event handling is crucial.
3. **Errors not identified by inspection**:
    - Errors in graphical rendering and complex input issues may not be covered.
4. **Applicability of Program Inspection**:
    - **Yes**, but it should be combined with runtime testing for GUI-based systems where user interaction is essential.

REFRENCE: LINK

PART – II: CODE DEBUGGING

Armstrong Number: Issues and Resolutions

1. **How many errors are there in the program?**
    - There are **two errors** identified in the program.
2. **How many breakpoints are required to fix those errors?**
    - You will need **two breakpoints** to address and fix these issues.

Steps Taken to Correct the Issues:

- **Issue 1:** The division and modulus operations are mistakenly reversed in the while loop.
    - **Solution:** Correct the operations by ensuring that the modulus retrieves the last digit, and the division reduces the number in each iteration for further processing.
- **Issue 2:** The accumulation of the check variable is incorrect.
    - **Solution:** Adjust the logic to accurately calculate the sum of each digit raised to the appropriate power, ensuring the check variable stores the correct total.

```
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.
```

GCD and LCM: Issues and Corrections

1. **How many errors are there in the program?**
    - **The program contains one error.**
2. **How many breakpoints are needed to resolve the error?**
    - **One breakpoint is required to correct this error.**

Steps Taken to Correct the Error:

- **Issue: The condition in the while loop within the GCD method is incorrect.**
- **Solution: Modify the condition from while (a % b == 0) to while (a % b != 0). This change ensures that the loop continues until the remainder becomes zero, which is essential for properly calculating the GCD.**

```java
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();
    }
}
```

Knapsack Problem: Issues and Resolutions

1. **How many errors are there in the program?**
    - There are **three errors** in the program.
2. **How many breakpoints are needed to resolve these errors?**
    - **Two breakpoints** are needed to correct these errors.

Steps Taken to Correct the Errors:

- **Issue 1:** The condition in the "take item n" case is incorrect.
    - **Solution:** Modify the condition from `if (weight[n] > w)` to `if (weight[n] <= w)` to ensure the profit is calculated only when the item can be included in the knapsack.
- **Issue 2:** The profit calculation is wrong.
    - **Solution:** Change `profit[n-2]` to `profit[n]` to ensure the correct profit value is used during the calculation.
- **Issue 3:** The indexing in the "don't take item n" case is incorrect.
    - **Solution:** Adjust `opt[n++][w]` to `opt[n-1][w]` to correctly reference the previous item's index during the calculation.

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

Magic Number Check: Issues and Corrections

1. **How many errors are there in the program?**
   - The program has **three errors**.
2. **How many breakpoints are needed to fix these errors?**
   - **One breakpoint** is enough to fix the errors.

Steps Taken to Correct the Errors:

- **Issue 1:** The condition in the inner `while` loop is wrong.
  - **Solution:** Change `while(sum == 0)` to `while(sum != 0)` so that the loop processes the digits until the sum becomes zero.
- **Issue 2:** The calculation of `s` inside the loop is incorrect.
  - **Solution:** Replace `s = s * (sum / 10)` with `s = s + (sum % 10)` to correctly add the individual digits.
- **Issue 3:** The operations inside the loop are in the wrong order.
  - **Solution:** Reorder the operations to `s = s + (sum % 10); sum = sum / 10;` to properly accumulate the sum of digits.


```java
// 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: Issues and Resolutions

1. **How many errors are there in the program?**
    - There are **three errors** in the program.
2. **How many breakpoints are needed to fix these errors?**
    - **Two breakpoints** are required to resolve these issues.

Steps Taken to Correct the Errors:

- **Issue 1:** Incorrect array indexing when splitting the array in `mergeSort`.
    - **Solution:** Update the array splitting logic by changing `int[] left = leftHalf(array+1)` to `int[] left = leftHalf(array)` and `int[] right = rightHalf(array-1)` to `int[] right = rightHalf(array)`, ensuring that the entire array is passed correctly.
- **Issue 2:** Incorrect increment and decrement in the `merge` function.
    - **Solution:** Remove the `++` and `--` from `merge(array, left++, right--)` and use `merge(array, left, right)` to pass the sub-arrays directly without altering their indices.
- **Issue 3:** The `merge` function is accessing the array out of bounds.
    - **Solution:** Adjust the array indexing to respect the array boundaries, preventing the function from accessing elements outside the valid range.


```java
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++;
            }
        }
```

```
    }
 }
```

Matrix Multiplication: Errors and Fixes

1. **How many errors are there in the program?**
   - There is **one error** in the program.
2. **How many breakpoints are needed to fix this error?**
   - **One breakpoint** is sufficient to resolve the issue.
3. **Steps taken to fix the error:**
   - **Error**: The matrix multiplication logic contains incorrect array indexing.
   - **Fix**: Update the indices from `first[c-1][c-k]` and `second[k-1][k-d]` to `first[c][k]` and `second[k][d]` respectively. This adjustment ensures that the matrix elements are correctly accessed during the multiplication process.

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

Quadratic Probing Hash Table: Errors and Fixes

1. **How many errors are there in the program?**
    - There is **one error** in the program.
2. **How many breakpoints are needed to fix this error?**
    - **One breakpoint** is sufficient to fix this issue.
3. **Steps taken to fix the error:**
    - **Error**: The insertion method contains a faulty line: i += (i + h / h--) % maxSize;.
    - **Fix**: The correct approach should be i = (i + h * h++) % maxSize;, which properly implements the quadratic probing logic to resolve collisions in the hash table.

```
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');
        }
```

```
    }
```

Sorting Array: Errors and Fixes

1. **How many errors are there in the program?**
    - There are **two errors** in the program.
2. **How many breakpoints are needed to fix these errors?**
    - **Two breakpoints** are needed to fix the errors.
3. **Steps taken to fix the errors:**
    - **Error 1**: The loop condition for (int i = 0; i >= n; i++); is incorrect.
    - **Fix 1**: Change it to for (int i = 0; i < n; i++) to ensure proper iteration over the array.
    - **Error 2**: The condition in the inner loop if (a[i] <= a[j]) should be reversed.
    - **Fix 2**: Modify it to if (a[i] > a[j]) to correctly sort 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: Errors and Fixes

1. **How many errors are there in the program?**
    - There are **two errors** in the program.
2. **How many breakpoints are needed to fix these errors?**
    - **Two breakpoints** are needed to fix the errors.
3. **Steps taken to fix the errors:**
    - **Error 1**: In the push method, the line top-- is incorrect.
    - **Fix 1**: Change it to top++ to correctly increment the stack pointer when pushing elements onto the stack.
    - **Error 2**: In the display method, the loop condition for (int i = 0; i > top; i++) is incorrect.
    - **Fix 2**: Modify it to for (int i = 0; i <= top; i++) to correctly display all elements in the stack.

```
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();
    }
}
```

Tower of Hanoi: Errors and Fixes

1. **How many errors are there in the program?**
   - There is **one error** in the program.
2. **How many breakpoints are needed to fix this error?**
   - **One breakpoint** is needed to fix the error.
3. **Steps taken to fix the error:**
   - **Error**: In the recursive call doTowers(topN++, inter--, from+1, to+1);, incorrect increments and decrements are applied to the variables.
   - **Fix**: Update the recursive call to doTowers(topN - 1, inter, from, to); to correctly follow the Tower of Hanoi logic and ensure proper recursion.

```
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)
      }
   }
}
```