**Assignment: Collections & Generics**

**1. Write a generic method printArray, which takes input of an array and prints each element of the array.**

**Print an array of Integer, Double, Character using the same method.**

**Answer:**

public class GenericMethod {

    public static void main(String[] args) {

        Integer[] intArray = {1, 2, 3, 4, 5};

        Double[] doubleArray = {1.1, 2.2, 3.3, 4.4};

        Character[] charArray = {'H', 'E', 'L', 'L', 'O'};

        printArray(intArray);

        printArray(doubleArray);

        printArray(charArray);

    }

    public static <E> void printArray(E[] inputArray) {

        for (E element : inputArray) {

            System.out.printf("%s ", element);

        }

        System.out.println();

    }

}

**2.Write a program that uses a stack to determine whether a string is a palindrome (i.e string is spelled identically backward and forward).The program should ignore spaces and punctuation.**

**Answer:**

import java.util.Scanner;

import java.util.Stack;

public class Palindrome {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        System.out.println("Enter a string: ");

        String s = input.nextLine();

        System.out.println("The string is " + (isPalindrome(s) ? "a palindrome" : "not a palindrome"));

    }

    public static boolean isPalindrome(String s) {

        Stack<Character> stack = new Stack<>();

        for (int i = 0; i < s.length(); i++) {

            if (Character.isLetter(s.charAt(i))) {

                stack.push(Character.toLowerCase(s.charAt(i)));

            }

        }

        for (int i = 0; i < s.length(); i++) {

            if (Character.isLetter(s.charAt(i))) {

                if (Character.toLowerCase(s.charAt(i)) != stack.pop()) {

                    return false;

                }

            }

        }

        return true;

    }

}

**3. Write a program that concatenates two linked list objects of characters. Class ListConcatenate should include a static method concatenate that takes references to both list objects as arguments and concatenates the second list to the first list.**

**Answer:**

import java.util.\*;

public class ListConcatenate {

    public static void main(String[] args) {

        List<Character> list1 = new LinkedList<>();

        List<Character> list2 = new LinkedList<>();

        list1.add('A');

        list1.add('B');

        list1.add('C');

        list2.add('D');

        list2.add('E');

        list2.add('F');

        concatenate(list1, list2);

        System.out.println(list1);

    }

    public static void concatenate(List<Character> list1, List<Character> list2) {

        list1.addAll(list2);

    }

}

**4. In the classic problem of the Towers of Hanoi, you have 3 towers and N disks of different sizes which can slide onto any tower. The puzzle starts with disks sorted in ascending order of size from top to bottom (i.e., each disk sites on top of an even larger one). You have the following constraints:**

**1. Only one disk can be moved at a time.**

**2. A disk is slid off the top of one tower onto the next tower.**

**3. A disk can only be placed on top of a larger disk.**

**Write program to move the disks from the first tower to the last using stacks.**

**Answer:**

import java.util.\*;

public class TowersOfHanoi {

public static int n = 3;

@SuppressWarnings("unchecked")

public static Stack<Integer>[] towers = new Stack[3];

static char[] ch=new char[]{'A','B','C'};

public static void main(String[] args) {

for (int i = 0; i < 3; i++) {

towers[i] = new Stack<Integer>();

}

for (int i = n; i > 0; i--) {

towers[0].push(i);

}

moveDisks(n, towers[0], towers[1], towers[2], 0, 1, 2);

}

public static void moveDisks(int n, Stack<Integer> origin, Stack<Integer> buffer, Stack<Integer> destination, int a, int b, int c) {

if (n <= 0) {

return;

}

moveDisks(n - 1, origin, destination, buffer, a, c, b); //`n-1` disks from origin to buffer, using destination as a buffer

destination.push(origin.pop());

System.out.println("Move disk " + n + " from " + ch[a] + " to " + ch[c]);

moveDisks(n - 1, buffer, origin, destination, b, a, c); //`n-1` disks from buffer to destination, using origin as a buffer

}

}