**CSA09 – Programming in Java**

**Day 3 Assignment Questions**

* Write a program in Java for dynamically changing the color of Text using Multithreading.

Solution:

import java.awt.Color;

public class DynamicTextColor {

public static void main(String[] args) {

final int DELAY = 1000; // Delay in milliseconds

final String MESSAGE = "Hello, world!"; // Message to display

// Start a new thread to change the text color every second

new Thread(() -> {

Color[] colors = {Color.RED, Color.ORANGE, Color.YELLOW, Color.GREEN, Color.BLUE, Color.MAGENTA};

int index = 0;

while (true) {

System.out.println("\033[38;2;" + colors[index].getRed() + ";" + colors[index].getGreen() + ";" + colors[index].getBlue() + "m" + MESSAGE);

index = (index + 1) % colors.length;

try {

Thread.sleep(DELAY);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}).start();

// Keep the main thread running so the program doesn't exit

while (true) {

try {

Thread.sleep(Long.MAX\_VALUE);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

* Differentiate Multiprocessing and Multithreading. Display Multiplication table for 5 and 10 using various stages of life cycle of the thread by generating a suitable code in Java.

Solution:

public class MultiplicationTable implements Runnable {

private int number;

public MultiplicationTable(int number) {

this.number = number;

}

@Override

public void run() {

System.out.println(Thread.currentThread().getName() + " started.");

// Display multiplication table

for (int i = 1; i <= 10; i++) {

System.out.println(number + " x " + i + " = " + (number \* i));

}

System.out.println(Thread.currentThread().getName() + " finished.");

}

public static void main(String[] args) throws InterruptedException {

// Create threads

Thread thread1 = new Thread(new MultiplicationTable(5));

Thread thread2 = new Thread(new MultiplicationTable(10));

// Start threads

thread1.start();

thread2.start();

// Wait for threads to finish

thread1.join();

thread2.join();

}

}

* An ugly number is a positive integer whose prime factors are limited to 2, 3, and 5.

Given an integer n, return true if n is an ugly number.

Example 1:

Input: n = 6

Output: true

Explanation: 6 = 2 × 3

Example 2:

Input: n = 1

Output: true

Explanation: 1 has no prime factors, therefore all of its prime factors are limited to 2, 3, and 5.

Example 3:

Input: n = 14

Output: false

Explanation: 14 is not ugly since it includes the prime factor 7.

Constraints:

-231 <= n <= 231 - 1

class Solution {

public:

bool isUgly(int n) {

}

}

Solution:

class Solution {

public boolean isUgly(int n) {

if(n <= 0) {

return false;

}

while(n % 2 == 0) {

n /= 2;

}

while(n % 3 == 0) {

n /= 3;

}

while(n % 5 == 0) {

n /= 5;

}

return n == 1;

}

}

* The Fibonacci numbers, commonly denoted F(n) form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

F(0) = 0, F(1) = 1

F(n) = F(n - 1) + F(n - 2), for n > 1.

Given n, calculate F(n).

Example 1:

Input: n = 2

Output: 1

Explanation: F(2) = F(1) + F(0) = 1 + 0 = 1.

Example 2:

Input: n = 3

Output: 2

Explanation: F(3) = F(2) + F(1) = 1 + 1 = 2.

Example 3:

Input: n = 4

Output: 3

Explanation: F(4) = F(3) + F(2) = 2 + 1 = 3.

Constraints:

0 <= n <= 30

class Solution {

public:

int fib(int n) {

}

}

Solution:

class Solution {

public int fib(int n) {

if(n == 0) {

return 0;

}

if(n == 1) {

return 1;

}

int[] fib = new int[n+1];

fib[0] = 0;

fib[1] = 1;

for(int i = 2; i <= n; i++) {

fib[i] = fib[i-1] + fib[i-2];

}

return fib[n];

}

}

* Removing duplicate elements in java : Find/Debug the errors and get output

class duplicate

{

// Function to remove duplicate elements

// This function returns new size of modified

// array.

static int removeDuplicates(int arr[], int n)

{

// Return, if array is empty

// or contains a single element

if (n==0 || n==1)

return n;

int[] temp = new int[n];

// Start traversing elements

int j = 0;

for (int j=0; i<n-1; i++)

// If current element is not equal

// to next element then store that

// current element

if (arr[i] != arr[i+1])

temp[j++] = arr[i];

// Store the last element as whether

// it is unique or repeated, it hasn't

// stored previously

temp[j++] = arr[n-1];

// Modify original array

for (int i=0; i<j; i++)

arr[i] = temp[i];

return j;

}

public static void main (String[] args)

{

it arr[] = {10, 20, 20, 30, 40, 40, 40, 50, 50};

int n = arr.length;

n = removeDuplicates(arr);

// Print updated array

for (int i=0; i<n; i++)

System.out.print(arr[i]+" ");

}

}

Solution:

class Duplicate {

// Function to remove duplicate elements

// This function returns new size of modified

// array.

static int removeDuplicates(int arr[], int n) {

// Return, if array is empty

// or contains a single element

if (n == 0 || n == 1) {

return n;

}

int[] temp = new int[n];

// Start traversing elements

int j = 0;

for (int i = 0; i < n - 1; i++) { // changed "j" to "i"

// If current element is not equal

// to next element then store that

// current element

if (arr[i] != arr[i + 1]) {

temp[j++] = arr[i];

}

}

// Store the last element as whether

// it is unique or repeated, it hasn't

// stored previously

temp[j++] = arr[n - 1];

// Modify original array

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

arr[i] = temp[i];

}

return j;

}

public static void main(String[] args) {

int arr[] = { 10, 20, 20, 30, 40, 40, 40, 50, 50 };

int n = arr.length;

n = removeDuplicates(arr, n); // passed n as second argument

// Print updated array

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

System.out.print(arr[i] + " ");

}

}

}