1. Write a Java program to perform a runnable interface, take two threads t1 and t2 and fetch the names of the thread using getName() method.

import java.lang.Thread;

class MyRunnable implements Runnable {

String name;

MyRunnable(String name) {

this.name = name;

}

public void run() {

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

}

}

public class Main {

public static void main(String[] args) {

Thread t1 = new Thread(new MyRunnable("Thread-1"));

Thread t2 = new Thread(new MyRunnable("Thread-2"));

t1.start();

t2.start();

}

}

2.Given an integer N, the task is to write program to print the first N natural numbers in increasing order using two threads.

***Input:****N = 10****Output:****1 2 3 4 5 6 7 8 9 10*

***Input:****N = 18****Output:****1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18*

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.ReentrantLock;

class PrintNumbers {

private int n;

private volatile int counter = 1;

private ReentrantLock lock = new ReentrantLock();

private Condition oddCondition = lock.newCondition();

private Condition evenCondition = lock.newCondition();

PrintNumbers(int n) {

this.n = n;

}

void printOddNumbers() {

try {

lock.lock();

while (counter <= n) {

if (counter % 2 == 0) {

evenCondition.await();

} else {

System.out.println(counter);

counter++;

oddCondition.signal();

}

}

} catch (InterruptedException e) {

e.printStackTrace();

} finally {

lock.unlock();

}

}

void printEvenNumbers() {

try {

lock.lock();

while (counter <= n) {

if (counter % 2 == 1) {

oddCondition.await();

} else {

System.out.println(counter);

counter++;

evenCondition.signal();

}

}

} catch (InterruptedException e) {

e.printStackTrace();

} finally {

lock.unlock();

}

}

}

public class Main {

public static void main(String[] args) {

int N = 10; // Change this to your desired value of N

PrintNumbers numbers = new PrintNumbers(N);

Thread t1 = new Thread(() -> numbers.printOddNumbers());

Thread t2 = new Thread(() -> numbers.printEvenNumbers());

t1.start();

t2.start();

}

}

3.  Write a two-threaded program, where one thread finds all prime numbers (in 0 to 10) and another thread finds all palindrome numbers (in 10 to 50). Schedule these threads in a sequential manner to get the results.

Palindrome numbers from 10 to 50 : 11 22 33 44

Prime numbers from 0 to 10 : 2 3 5 7

import java.util.concurrent.ThreadLocalRandom;

public class SequentialThreads {

public static void main(String[] args) {

// Define ranges for prime and palindrome searches

int primeStart = 0;

int primeEnd = 10;

int palindromeStart = 10;

int palindromeEnd = 50;

// Create threads

Thread primeThread = new Thread(() -> findPrimes(primeStart, primeEnd));

Thread palindromeThread = new Thread(() -> findPalindromes(palindromeStart, palindromeEnd));

// Start threads sequentially

primeThread.start();

try {

primeThread.join(); // Wait for prime thread to finish before starting palindrome thread

} catch (InterruptedException e) {

e.printStackTrace();

}

palindromeThread.start();

}

private static void findPrimes(int start, int end) {

System.out.println("Prime numbers:");

for (int num = start; num <= end; num++) {

if (isPrime(num)) {

System.out.println(num);

}

}

}

private static boolean isPrime(int num) {

if (num <= 1) {

return false;

}

for (int i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) {

return false;

}

}

return true;

}

private static void findPalindromes(int start, int end) {

System.out.println("Palindrome numbers:");

for (int num = start; num <= end; num++) {

if (isPalindrome(num)) {

System.out.println(num);

}

}

}

private static boolean isPalindrome(int num) {

int reversedNum = 0, remainder, originalNum = num;

while (num != 0) {

remainder = num % 10;

reversedNum = reversedNum \* 10 + remainder;

num /= 10;

}

return originalNum == reversedNum;

}

}