package assignment2.problem1;  
  
import java.util.ArrayList;  
import java.util.Random;  
  
public class PrimeNumberThread extends Thread {  
  
 private int begin, end;  
 private int numberOfPrimeNumbers;  
 public ArrayList<Integer> primeNumbers;  
  
 public static Random *random* = new Random(); // used to simulate random waiting time  
  
 public PrimeNumberThread(String name, int begin, int end) {  
 setName(name);  
 this.begin = begin;  
 this.end = end;  
 primeNumbers = new ArrayList<>();  
 }  
  
 public int getNumberOfPrimeNumbers() { return numberOfPrimeNumbers; }  
  
 public void run() {  
 System.*out*.println(getName() + " [" + begin + "," + end + "] started!");  
 boolean isPrime;  
 // check all numbers in range  
 for (int i = Math.*max*(begin, 2); i <= end; i++) {  
 isPrime = true;  
 // calculate whether number is prime  
 for (int divisor = 2; divisor < i; divisor++) {  
 if (i % divisor == 0) {  
 isPrime = false;  
 break; // stop checking after already false  
 }  
 }  
 if (isPrime) {  
 numberOfPrimeNumbers += 1;  
 primeNumbers.add(i);  
 }  
 }  
 System.*out*.println(getName() + " [" + begin + "," + end + "] completed!");  
 }  
  
 public void print() {  
 System.*out*.println(getName() + " [" + begin + "," + end + "] : prime numbers found = " + primeNumbers.toString());  
 }  
  
}

package assignment2.problem1;  
  
public class Problem1 {  
  
 public static void main(String[] args) throws InterruptedException {  
  
 int from = 1;  
 int to = 20;  
  
 int numberOfThreads = 3;  
 int subLength = (to - from + 1) / numberOfThreads;  
  
 PrimeNumberThread[] pnthreads = new PrimeNumberThread[numberOfThreads];  
  
 // setup and create threads  
 for (int i = 0; i < numberOfThreads; i++) {  
 int subFrom = subLength \* i + from;  
 int subTo = (i < numberOfThreads - 1) ? subFrom + subLength - 1 : to;  
 pnthreads[i] = new PrimeNumberThread("Thread-" + i, subFrom, subTo);  
 }  
  
 // start threads  
 for (int i = 0; i < numberOfThreads; i++) {  
 pnthreads[i].start();  
 }  
  
 // wait for all threads to finish  
 for (int i = 0; i < numberOfThreads; i++) {  
 pnthreads[i].join();  
 }  
  
 int totalNumberOfPrimeNumbers = 0;  
  
 // add all amounts of primes  
 for (int i = 0; i < numberOfThreads; i++) {  
 totalNumberOfPrimeNumbers += pnthreads[i].getNumberOfPrimeNumbers();  
 }  
  
 // print out all numbers  
 System.*out*.println();  
 for (int i = 0; i < numberOfThreads; i++) {  
 pnthreads[i].print();  
 }  
  
 System.*out*.println("\nTotal number of prime numbers found: " + totalNumberOfPrimeNumbers);  
  
 }  
  
}  
*/\*\*  
 Thread-0 [1,6] started!  
 Thread-1 [7,12] started!  
 Thread-2 [13,20] started!  
 Thread-2 [13,20] completed!  
 Thread-0 [1,6] completed!  
 Thread-1 [7,12] completed!  
  
 Thread-0 [1,6] : prime numbers found = [2, 3, 5]  
 Thread-1 [7,12] : prime numbers found = [7, 11]  
 Thread-2 [13,20] : prime numbers found = [13, 17, 19]  
  
 Total number of prime numbers found: 8  
 \*/*

package assignment2.problem2;  
  
public class MatrixThread extends Thread {  
  
 private int[] matrixRow;  
 private int[] matrixCol;  
  
 private int result;  
  
 public MatrixThread(int[] matrixRow, int[] matrixCol) {  
 this.matrixRow = matrixRow;  
 this.matrixCol = matrixCol;  
 result = 0;  
 }  
  
 public int getResult() { return result; }  
  
 public void run() {  
 // calculate product of the matrixes' row and column  
 for (int i = 0; i < matrixRow.length; i++) {  
 result += matrixRow[i] \* matrixCol[i];  
 }  
 }  
  
}

package assignment2.problem2;  
  
public class Problem2 {  
  
 public static void main(String[] args) throws InterruptedException {  
  
 // initialize matrixes  
 int[][] matrixA = {  
 {3, 7},  
 {3, 2},  
 {6, 5},  
 {4, 8} };  
 int[][] matrixB = {  
 {3, 7, 2},  
 {3, 2, 9} };  
  
 // initialize MatrixThread array  
 MatrixThread[][] matrixThreadMatrix = new MatrixThread[matrixA.length][matrixB[0].length];  
 for (int row = 0; row < matrixThreadMatrix.length; row++) {  
 for (int col = 0; col < matrixThreadMatrix[row].length; col++) {  
 // get column of matrixB  
 int[] matrixBCol = new int[matrixA[row].length];  
 for (int i = 0; i < matrixBCol.length; i++) {  
 matrixBCol[i] = matrixB[i][col];  
 }  
 // initialize new MatrixThread with row and column  
 matrixThreadMatrix[row][col] = new MatrixThread(matrixA[row], matrixBCol);  
 }  
 }  
  
 // start all threads  
 for (int row = 0; row < matrixThreadMatrix.length; row++) {  
 for (int col = 0; col < matrixThreadMatrix[row].length; col++) {  
 matrixThreadMatrix[row][col].start();  
 }  
 }  
  
 // join all threads  
 for (int row = 0; row < matrixThreadMatrix.length; row++) {  
 for (int col = 0; col < matrixThreadMatrix[row].length; col++) {  
 matrixThreadMatrix[row][col].join();  
 }  
 }  
  
 // get product  
 int[][] product = new int[matrixThreadMatrix.length][matrixThreadMatrix[0].length];  
 for (int row = 0; row < matrixThreadMatrix.length; row++) {  
 for (int col = 0; col < matrixThreadMatrix[row].length; col++) {  
 product[row][col] = matrixThreadMatrix[row][col].getResult();  
 }  
 }  
  
 // print matrixes  
 *printMatrix*(matrixA, "A");  
 *printMatrix*(matrixB, "B");  
 *printMatrix*(product, "C");  
  
 }  
  
 public static void printMatrix(int[][] matrix, String name) {  
 System.*out*.println("Matrix " + name);  
 for (int row = 0; row < matrix.length; row++) {  
 for (int col = 0; col < matrix[row].length; col++) {  
 System.*out*.print(matrix[row][col] + "\t");  
 }  
 System.*out*.println();  
 }  
 System.*out*.println();  
 }  
  
}  
*/\*\*  
 Matrix A  
 3 7  
 3 2  
 6 5  
 4 8  
  
 Matrix B  
 3 7 2  
 3 2 9  
  
 Matrix C  
 30 35 69  
 15 25 24  
 33 52 57  
 36 44 80  
 \*/*

package assignment2.problem3;  
  
public class RowThread extends Thread {  
  
 private int[] row;  
  
 private int min;  
  
 public RowThread(int[] row) {  
 this.row = row;  
 min = row[0];  
 }  
  
 public int getMin() { return min; }  
  
 public void run() {  
 for (int i = 0; i < row.length; i++) {  
 if (row[i] < min) {  
 min = row[i];  
 }  
 }  
 }  
  
}

package assignment2.problem3;  
  
public class ColumnThread extends Thread {  
  
 public int[] col;  
  
 private int max;  
  
 public ColumnThread(int[] col) {  
 this.col = col;  
 max = col[0];  
 }  
  
 public int getMax() { return max; }  
  
 public void run() {  
 for (int i = 0; i < col.length; i++) {  
 if (col[i] > max) {  
 max = col[i];  
 }  
 }  
 }  
  
}

package assignment2.problem3;  
  
public class Problem3 {  
  
 public static void main(String[] args) throws InterruptedException {  
  
 int[][] matrix = {  
 {3, 7, 2, 3, 2},  
 {3, 2, 9, 2, 2},  
 {6, 5, 6, 4, 8},  
 {5, 2, 2, 3, 2} };  
  
 RowThread[] rowThreads = new RowThread[matrix.length];  
 ColumnThread[] columnThreads = new ColumnThread[matrix[0].length];  
  
 // initialize RowThreads  
 for (int i = 0; i < matrix.length; i++) {  
 rowThreads[i] = new RowThread(matrix[i]);  
 }  
  
 // initialize ColumnThreads  
 for (int i = 0; i < matrix[0].length; i++) {  
 int[] column = new int[matrix.length];  
 for (int j = 0; j < matrix.length; j++) {  
 column[j] = matrix[j][i];  
 }  
 columnThreads[i] = new ColumnThread(column);  
 }  
  
 // start threads  
 for (int i = 0; i < rowThreads.length; i++) {  
 rowThreads[i].start();  
 }  
 for (int i = 0; i < columnThreads.length; i++) {  
 columnThreads[i].start();  
 }  
  
 // join threads  
 for (int i = 0; i < rowThreads.length; i++) {  
 rowThreads[i].join();  
 }  
 for (int i = 0; i < columnThreads.length; i++) {  
 columnThreads[i].join();  
 }  
  
 System.*out*.println("Matrix M");  
 for (int row = 0; row < matrix.length; row++) {  
 for (int col = 0; col < matrix[row].length; col++) {  
 System.*out*.print(matrix[row][col] + "\t");  
 }  
 System.*out*.println();  
 }  
 System.*out*.println();  
  
 // loop through mins of each row and maxs of each col  
 boolean saddleExists = false;  
 for (int row = 0; row < rowThreads.length; row++) {  
 if (saddleExists) { break; }  
 for (int col = 0; col < columnThreads.length; col++) {  
 if (rowThreads[row].getMin() == columnThreads[col].getMax()) {  
 System.*out*.println("The saddle point is in M[" + row + "," + col + "] = " + rowThreads[row].getMin());  
 saddleExists = true;  
 break;  
 }  
 }  
 }  
  
 if (!saddleExists) {  
 System.*out*.println("No saddle point exists in Matrix M.");  
 }  
  
 }  
  
}  
*/\*\*  
 Matrix M  
 3 7 2 3 2  
 3 2 9 2 2  
 6 5 6 4 8  
 5 2 2 3 2  
  
 The saddle point is in M[2,3] = 4  
 \*/*