Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА № 1.3

з дисципліни «Паралельне програмування»
на тему «Потоки в мові Java»

ВИКОНАВ:
студент III курсу ФІОТ
групи ІО-25
Мамченко Д.О.
Залікова №2518

Київ – 2024

Тема: Потоки в мові Java.

Мета: Вивчення засобів мови Java для роботи с потоками.

Виконання роботи: Розробити програму, яка містить **паралельні потоки**, що реалізують відповідну функцію F1, F2, F3 з **Додатку Б** згідно отриманому варіанту.

Вимоги щодо створення потоків і завдання дослідження особливості виконання паралельної програми визначені в лабораторної роботі 1.1.

В потоках використати метод **join()**.

Необхідні теоретичні відомості: мова Java забезпечує програмування паралельних процесів (потоків) за допомогою потоків **(threads)**. Використовується клас **Thread** або інтерфейс **Runnable**.

Потоки визначають паралельне виконання Java програми в ПКС. Управління виконанням потоків можна здійснити за допомогою пріоритетів потоків (метод **setPriority()**).

Метод **join()** використовується для синхронізації основного метода з потоками, які він запускає на виконання.

Метод **run()** визначає дії потоку при виконанні.

Варіант: 18

1.11 *F1:* c = MAX(MA * MB) * (A * B)
2.29 *F2:* MF = (MG + MH) * (MK * ML) * (MG + ML)
3.7 *F3:* O = (P + R) * (MS * MT)

Загальний опис програми

Ця програма реалізує розв'язання задач паралельного програмування на мові Java з використанням багатопоточності. Вона складається з трьох паралельних потоків, які виконують обчислення заданих варіантом функцій. Завдання обчислюються на основі введених, зчитаних або згенерованих даних, таких як вектори та матриці.

Програма використовує механізм синхронізації за допомогою **CyclicBarrier**, що забезпечує одночасний початок обчислень всіма потоками після завершення підготовки даних.

Основний функціонал:

1. **Введення даних:**

   Програма дозволяє користувачу вибрати, звідки брати дані для обчислень: з консолі, з файлів, згенеровані випадково або встановлені за замовчуванням (1 для T1, 2 для T2, 3 для T3).

2. **Паралельні потоки:**

   Програма створює три потоки:

   T1: Виконує обчислення функції F1:
   $c = MAX(MA * MB) * (A * B)$

   T2: Виконує обчислення функції F2:
   $MF = (MG + MH) * (MK * ML) * (MG + ML)$

   T3: Виконує обчислення функції F3:
   $O = (P + R) * (MS * MT)$

3. **Операції з даними:**

   Операції з векторами та матрицями включають добуток, суму, максимальні значення елементів та інші математичні операції, які необхідні для розв'язання поставлених задач.

4. **Виведення результатів:**

   Після завершення обчислень кожен потік виводить свій результат. Для зручності виведення результатів вектори та матриці форматуються у зрозумілий для користувача вигляд.

Код програми:

T1.java

```
package ua.kpi.lab1.thread;

import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.CyclicBarrier;
import ua.kpi.lab1.utils.Data;
import ua.kpi.lab1.InputType;

/**
* Клас, що представляє потік T1 для обчислення формули F1: c = MAX(MA * MB) * (A * B).
*/
public class T1 extends Thread {
 private final InputType inputType;
 private final CyclicBarrier barrier;

 int[] A = new int[0];
 int[] B = new int[0];
 int[][] MA = new int[0][];
 int[][] MB = new int[0][];

 /**
  * Конструктор для створення потоку T1.
  *
  * @param name      ім'я потоку
  * @param priority  пріоритет потоку
  * @param stackSize розмір стека
  * @param inputType тип введення даних (CONSOLE, RANDOM, FILE, DEFAULT)
  * @param barrier   об'єкт для синхронізації потоків
  */
 public T1(String name, int priority, long stackSize, InputType inputType, CyclicBarrier
barrier) {
   super(null, null, name, stackSize);
   this.setPriority(priority);
   this.inputType = inputType;
   this.barrier = barrier;
 }
```

```java
/**
 * Метод, який виконується при запуску потоку T1.
 * Зчитує дані, чекає на інші потоки та обчислює результат формули F1.
 */
@Override
public void run() {
  initializeTensors();

  waitBarrier();

  long startTime = System.currentTimeMillis();
  int result = function1(MA, MB, A, B);
  long endTime = System.currentTimeMillis();

  System.out.println("Результат '" + getName() + "': " + result);
  System.out.println("Час виконання обчислень '" + getName() + "': " +
      (endTime - startTime) + " мс");
}

/**
 * Ініціалізує тензори (вектори та матриці) для потоку T1 на основі типу введення
даних.
 * Зчитує дані з консолі, файлу або генерує випадкові дані.
 */
private void initializeTensors() {
  String taskName = getName();
  switch (inputType) {
    case CONSOLE:
      A = Data.inputVectorFromConsole(taskName, "A");
      B = Data.inputVectorFromConsole(taskName, "B");
      MA = Data.inputMatrixFromConsole(taskName, "MA");
      MB = Data.inputMatrixFromConsole(taskName, "MB");
      break;
    case RANDOM:
      A = Data.randomVector("A");
      B = Data.randomVector("B");
      MA = Data.randomMatrix("MA");
      MB = Data.randomMatrix("MB");
      break;
    case FILE:
      A = Data.readVectorFromFile("A.txt", "A");
      B = Data.readVectorFromFile("B.txt", "B");
      MA = Data.readMatrixFromFile("MA.txt", "MA");
      MB = Data.readMatrixFromFile("MB.txt", "MB");
      break;
    default:
      A = Data.inputVector(1, "A");
      B = Data.inputVector(1, "B");
      MA = Data.inputMatrix(1, "MA");
      MB = Data.inputMatrix(1, "MB");
      break;
  }
```

```
 }

 /**
  * Чекає на інші потоки, використовуючи CyclicBarrier для синхронізації.
  * Виводить повідомлення про очікування інших потоків.
  */
 private void waitBarrier() {
   try {
     System.out.println("'" + getName() + "' має всі дані та чекає на інші потоки.");
     barrier.await();
   } catch (InterruptedException | BrokenBarrierException e) {
     throw new RuntimeException(e);
   }
 }

 /**
  * Обчислює результат формули F1: c = MAX(MA * MB) * (A * B).
  *
  * @param MA матриця MA
  * @param MB матриця MB
  * @param A вектор A
  * @param B вектор B
  * @return результат обчислення формули F1 у вигляді цілого числа
  */
 private int function1(int[][] MA, int[][] MB, int[] A, int[] B) {
   return Data.maxMatrixElement(Data.multiplyMatrices(MA, MB)) * Data.dotProduct(A, B);
 }
}
```

T2.java

```
package ua.kpi.lab1.thread;

import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.CyclicBarrier;
import ua.kpi.lab1.utils.Data;
import ua.kpi.lab1.InputType;

/**
* Клас, що представляє потік T2 для обчислення формули F2: MF = (MG + MH) * (MK * ML) *
(MG + ML).
*/
public class T2 extends Thread {

 private final InputType inputType;
 private final CyclicBarrier barrier;

 int[][] MG = new int[0][];
 int[][] MH = new int[0][];
 int[][] MK = new int[0][];
 int[][] ML = new int[0][];

 /**
```

```java
  * Конструктор для створення потоку T2.
  *
  * @param name      ім'я потоку
  * @param priority  пріоритет потоку
  * @param stackSize розмір стека
  * @param inputType тип введення даних (CONSOLE, RANDOM, FILE, DEFAULT)
  * @param barrier   об'єкт для синхронізації потоків
  */
 public T2(String name, int priority, long stackSize, InputType inputType, CyclicBarrier
barrier) {
   super(null, null, name, stackSize);
   this.setPriority(priority);
   this.inputType = inputType;
   this.barrier = barrier;
 }

 /**
  * Метод, який виконується при запуску потоку T2.
  * Зчитує дані, чекає на інші потоки та обчислює результат формули F2.
  */
 @Override
 public void run() {
   initializeTensors();

   waitBarrier();

   long startTime = System.currentTimeMillis();
   int[][] result = function2(MG, MH, MK, ML);
   long endTime = System.currentTimeMillis();

   System.out.println("Результат '" + getName() + "':\n" + Data.printMatrix(result,
true));
   System.out.println("Час виконання обчислень '" + getName() + "': " +
       (endTime - startTime) + " мс");
 }

 /**
  * Ініціалізує тензори (матриці) для потоку T2 на основі типу введення даних.
  * Зчитує дані з консолі, файлу або генерує випадкові дані.
  */
 private void initializeTensors() {
   String taskName = getName();
   switch (inputType) {
     case CONSOLE:
       MG = Data.inputMatrixFromConsole(taskName, "MG");
       MH = Data.inputMatrixFromConsole(taskName, "MH");
       MK = Data.inputMatrixFromConsole(taskName, "MK");
       ML = Data.inputMatrixFromConsole(taskName, "ML");
       break;
     case RANDOM:
       MG = Data.randomMatrix("MG");
       MH = Data.randomMatrix("MH");
       MK = Data.randomMatrix("MK");
```

```
        ML = Data.randomMatrix("ML");
        break;
      case FILE:
        MG = Data.readMatrixFromFile("MG.txt", "MG");
        MH = Data.readMatrixFromFile("MH.txt", "MH");
        MK = Data.readMatrixFromFile("MK.txt", "MK");
        ML = Data.readMatrixFromFile("ML.txt", "ML");
        break;
      default:
        MG = Data.inputMatrix(2, "MG");
        MH = Data.inputMatrix(2, "MH");
        MK = Data.inputMatrix(2, "MK");
        ML = Data.inputMatrix(2, "ML");
        break;
    }
  }

  /**
   * Чекає на інші потоки, використовуючи CyclicBarrier для синхронізації.
   * Виводить повідомлення про очікування інших потоків.
   */
  private void waitBarrier() {
    try {
      System.out.println("'" + getName() + "' має всі дані та чекає на інші потоки.");
      barrier.await();
    } catch (InterruptedException | BrokenBarrierException e) {
      throw new RuntimeException(e);
    }
  }

  /**
   * Обчислює результат формули F2: MF = (MG + MH) * (MK * ML) * (MG + ML).
   *
   * @param MG матриця MG
   * @param MH матриця MH
   * @param MK матриця MK
   * @param ML матриця ML
   * @return результат обчислення формули F2 у вигляді матриці
   */
  private int[][] function2(int[][] MG, int[][] MH, int[][] MK, int[][] ML) {
    return Data.multiplyMatrices(
        Data.multiplyMatrices(
            Data.sumMatrices(MG, MH),
            Data.multiplyMatrices(MK, ML)),
        Data.sumMatrices(MG, ML));
  }
}
```

T3.java

```
package ua.kpi.lab1.thread;

import java.util.concurrent.BrokenBarrierException;
```

```java
import java.util.concurrent.CyclicBarrier;
import ua.kpi.lab1.utils.Data;
import ua.kpi.lab1.InputType;

/**
* Клас, що представляє потік T3 для обчислення формули F3: O = (P + R) * (MS * MT).
*/
public class T3 extends Thread {

 private final InputType inputType;
 private final CyclicBarrier barrier;

 int[] P = new int[0];
 int[] R = new int[0];
 int[][] MS = new int[0][];
 int[][] MT = new int[0][];

 /**
  * Конструктор для створення потоку T3.
  *
  * @param name      ім'я потоку
  * @param priority  пріоритет потоку
  * @param stackSize розмір стека
  * @param inputType тип введення даних (CONSOLE, RANDOM, FILE, DEFAULT)
  * @param barrier   об'єкт для синхронізації потоків
  */
 public T3(String name, int priority, long stackSize, InputType inputType, CyclicBarrier
barrier) {
   super(null, null, name, stackSize);
   this.setPriority(priority);
   this.inputType = inputType;
   this.barrier = barrier;
 }

 /**
  * Метод, який виконується при запуску потоку T3.
  * Зчитує дані, чекає на інші потоки та обчислює результат формули F3.
  */
 @Override
 public void run() {
   initializeTensors();

   waitBarrier();

   long startTime = System.currentTimeMillis();
   int[] result = function3(P, R, MS, MT);
   long endTime = System.currentTimeMillis();

   System.out.println("Результат '" + getName() + "': " + Data.printVector(result,
true));
   System.out.println("Час виконання обчислень '" + getName() + "': " +
      (endTime - startTime) + " мс");
```

```
}

/**
 * Ініціалізує тензори (вектори та матриці) для потоку T3 на основі типу введення даних.
 * Зчитує дані з консолі, файлу або генерує випадкові дані.
 */
private void initializeTensors() {
  String taskName = getName();
  switch (inputType) {
    case CONSOLE:
      P = Data.inputVectorFromConsole(taskName, "P");
      R = Data.inputVectorFromConsole(taskName, "R");
      MS = Data.inputMatrixFromConsole(taskName, "MS");
      MT = Data.inputMatrixFromConsole(taskName, "MT");
      break;
    case RANDOM:
      P = Data.randomVector("P");
      R = Data.randomVector("R");
      MS = Data.randomMatrix("MS");
      MT = Data.randomMatrix("MT");
      break;
    case FILE:
      P = Data.readVectorFromFile("P.txt", "P");
      R = Data.readVectorFromFile("R.txt", "R");
      MS = Data.readMatrixFromFile("MS.txt", "MS");
      MT = Data.readMatrixFromFile("MT.txt", "MT");
      break;
    default:
      P = Data.inputVector(3, "P");
      R = Data.inputVector(3, "R");
      MS = Data.inputMatrix(3, "MS");
      MT = Data.inputMatrix(3, "MT");
      break;
  }
}

/**
 * Чекає на інші потоки, використовуючи CyclicBarrier для синхронізації.
 * Виводить повідомлення про очікування інших потоків.
 */
private void waitBarrier() {
  try {
    System.out.println("'" + getName() + "' має всі дані та чекає на інші потоки.");
    barrier.await();
  } catch (InterruptedException | BrokenBarrierException e) {
    throw new RuntimeException(e);
  }
}

/**
 * Обчислює результат формули F3: O = (P + R) * (MS * MT).
 *
```

```
   * @param P вектор P
   * @param R вектор R
   * @param MS матриця MS
   * @param MT матриця MT
   * @return результат обчислення формули F3 у вигляді вектора
   */
  private int[] function3(int[] P, int[] R, int[][] MS, int[][] MT) {
    return Data.multiplyVectorMatrix(
        Data.sumVectors(P, R),
        Data.multiplyMatrices(MS, MT));
  }
}
```

Data.java

```
package ua.kpi.lab1.utils;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Random;
import java.util.Scanner;

/**
* Клас Data містить методи для введення, генерації та обробки векторів і матриць.
* Він забезпечує введення даних з консолі, файлів або генерацію випадкових даних,
* а також виконання різних математичних операцій над векторами та матрицями.
*/
public class Data {
  public static int N = 3;
  private static final Object lock = new Object();

  /**
   * Вводить вектор з однаковими значеннями для кожного елемента.
   *
   * @param value значення для кожного елемента вектора
   * @param vectorName ім'я вектора
   * @return вектор з однаковими значеннями
   */
  public static int[] inputVector(int value, String vectorName) {
    int[] vector = new int[N];
    for (int i = 0; i < N; i++) {
      vector[i] = value;
    }
    System.out.printf("%s = %s%n", vectorName, printVector(vector));
    return vector;
  }

  /**
   * Вводить матрицю з однаковими значеннями для кожного елемента.
   *
   * @param value значення для кожного елемента матриці
```

```java
     * @param matrixName ім'я матриці
     * @return матриця з однаковими значеннями
     */
    public static int[][] inputMatrix(int value, String matrixName) {
      int[][] matrix = new int[N][N];
      for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
          matrix[i][j] = value;
        }
      }
      System.out.printf("%s = %n%s%n", matrixName, printMatrix(matrix));
      return matrix;
    }

    /**
     * Вводить вектор з консолі.
     *
     * @param taskName ім'я задачі
     * @param vectorName ім'я вектора
     * @return вектор, введений з консолі
     */
    public static int[] inputVectorFromConsole(String taskName, String vectorName) {
      synchronized (lock) {
        Scanner scanner = new Scanner(System.in);
        int[] vector = new int[N];
        System.out.printf("[%s] Введіть %d елементів для вектора '%s':%n",
            taskName, N, vectorName);
        for (int i = 0; i < N; i++) {
          vector[i] = scanner.nextInt();
        }
        System.out.printf("%s = %s%n", vectorName, printVector(vector));
        return vector;
      }
    }

    /**
     * Вводить матрицю з консолі.
     *
     * @param taskName ім'я задачі
     * @param matrixName ім'я матриці
     * @return матриця, введена з консолі
     */
    public static int[][] inputMatrixFromConsole(String taskName, String matrixName) {
      synchronized (lock) {
        Scanner scanner = new Scanner(System.in);
        int[][] matrix = new int[N][N];
        System.out.printf("[%s] Введіть елементи для %dx%d матриці '%s':%n",
            taskName, N, N, matrixName);
        for (int i = 0; i < N; i++) {
          for (int j = 0; j < N; j++) {
            matrix[i][j] = scanner.nextInt();
          }
        }
```

```
    System.out.printf("%s = %n%s%n", matrixName, printMatrix(matrix));
    return matrix;
  }
}

/**
 * Генерує випадковий вектор.
 *
 * @param vectorName ім'я вектора
 * @return випадковий вектор
 */
public static int[] randomVector(String vectorName) {
  Random rand = new Random();
  int[] vector = new int[N];
  for (int i = 0; i < N; i++) {
    vector[i] = rand.nextInt(100);
  }
  System.out.printf("%s = %s%n", vectorName, printVector(vector));
  return vector;
}

/**
 * Генерує випадкову матрицю.
 *
 * @param matrixName ім'я матриці
 * @return випадкова матриця
 */
public static int[][] randomMatrix(String matrixName) {
  Random rand = new Random();
  int[][] matrix = new int[N][N];
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      matrix[i][j] = rand.nextInt(100);
    }
  }
  System.out.printf("%s = %n%s%n", matrixName, printMatrix(matrix));
  return matrix;
}

/**
 * Зчитує вектор з файлу.
 *
 * @param filename ім'я файлу
 * @param vectorName ім'я вектора
 * @return вектор, зчитаний з файлу
 */
public static int[] readVectorFromFile(String filename, String vectorName) {
  int[] vector = new int[N];
  InputStream inputStream = Data.class.getClassLoader().getResourceAsStream(filename);
  if (inputStream == null) {
    throw new IllegalArgumentException("Input file not found: " + filename);
  }
  try (BufferedReader br = new BufferedReader(new InputStreamReader(inputStream))) {
```

```
    for (int i = 0; i < N; i++) {
      vector[i] = Integer.parseInt(br.readLine().trim());
    }
  } catch (IOException e) {
    throw new RuntimeException(e);
  }
  System.out.printf("%s = %s%n", vectorName, printVector(vector));
  return vector;
}

/**
 * Зчитує матрицю з файлу.
 *
 * @param filename ім'я файлу
 * @param matrixName ім'я матриці
 * @return матриця, зчитана з файлу
 */
public static int[][] readMatrixFromFile(String filename, String matrixName) {
  int[][] matrix = new int[N][N];
  InputStream inputStream = Data.class.getClassLoader().getResourceAsStream(filename);
  if (inputStream == null) {
    throw new IllegalArgumentException("Input file not found: " + filename);
  }
  try (BufferedReader br = new BufferedReader(new InputStreamReader(inputStream))) {
    for (int i = 0; i < N; i++) {
      String[] numbers = br.readLine().trim().split("\\s+");
      for (int j = 0; j < N; j++) {
        matrix[i][j] = Integer.parseInt(numbers[j]);
      }
    }
  } catch (IOException e) {
    throw new RuntimeException(e);
  }
  System.out.printf("%s = %n%s%n", matrixName, printMatrix(matrix));
  return matrix;
}

/**
 * Обчислює скалярний добуток двох векторів.
 *
 * @param A вектор A
 * @param B вектор B
 * @return скалярний добуток векторів
 */
public static int dotProduct(int[] A, int[] B) {
  int result = 0;
  for (int i = 0; i < N; i++) {
    result += A[i] * B[i];
  }
  return result;
}

/**
```

```java
 * Множить вектор на матрицю.
 *
 * @param vector вектор
 * @param matrix матриця
 * @return результат множення вектора на матрицю
 */
public static int[] multiplyVectorMatrix(int[] vector, int[][] matrix) {
  int[] result = new int[N];
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      result[i] += vector[j] * matrix[i][j];
    }
  }
  return result;
}

/**
 * Сумує два вектори.
 *
 * @param A вектор A
 * @param B вектор B
 * @return результат суми векторів
 */
public static int[] sumVectors(int[] A, int[] B) {
  int[] result = new int[N];
  for (int i = 0; i < N; i++) {
    result[i] = A[i] + B[i];
  }
  return result;
}

/**
 * Множить дві матриці.
 *
 * @param A матриця A
 * @param B матриця B
 * @return результат множення матриць
 */
public static int[][] multiplyMatrices(int[][] A, int[][] B) {
  int[][] result = new int[N][N];
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      for (int k = 0; k < N; k++) {
        result[i][j] += A[i][k] * B[k][j];
      }
    }
  }
  return result;
}

/**
 * Сумує дві матриці.
 *
```

```java
  * @param A матриця A
  * @param B матриця B
  * @return результат суми матриць
  */
 public static int[][] sumMatrices(int[][] A, int[][] B) {
   int[][] result = new int[N][N];
   for (int i = 0; i < N; i++) {
     for (int j = 0; j < N; j++) {
       result[i][j] = A[i][j] + B[i][j];
     }
   }
   return result;
 }

 /**
  * Знаходить максимальний елемент матриці.
  *
  * @param matrix матриця
  * @return максимальний елемент матриці
  */
 public static int maxMatrixElement(int[][] matrix) {
   int max = Integer.MIN_VALUE;
   for (int i = 0; i < N; i++) {
     for (int j = 0; j < N; j++) {
       if (matrix[i][j] > max) {
         max = matrix[i][j];
       }
     }
   }
   return max;
 }

 /**
  * Повертає рядкове представлення вектора.
  *
  * @param vector вектор
  * @param result чи є вивід вектора результатом
  * @return рядкове представлення вектора. Якщо вивід вектора не є результатом,
виводиться
  * скорочений вектор в разі перевищення довжини 50 символів. Інакше виводиться повний
вектор.
  */
 public static String printVector(int[] vector, boolean result) {
   StringBuilder sb = new StringBuilder();
   sb.append("(");
   for (int i = 0; i < N; i++) {
     sb.append(vector[i]);
     if (i < N - 1) {
       sb.append(", ");
     }
   }
   sb.append(")");
   if (result) {
```

```
      return sb.toString();
    } else {
      return sb.length() < 50 ? sb.toString() : sb.substring(0, 49) + "...";
    }
  }

  /**
   * Повертає рядкове представлення матриці.
   *
   * @param matrix матриця
   * @param result чи є вивід матриці результатом
   * @return рядкове представлення матриці. Якщо вивід матриці не є результатом,
виводиться
   * скорочена матриця в разі перевищення довжини 70 символів. Інакше виводиться повна
матриця.
   */
  public static String printMatrix(int[][] matrix, boolean result) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
        sb.append(matrix[i][j]).append("\t");
      }
      sb.append("\n");
    }
    if (result) {
      return sb.toString();
    } else {
      return sb.length() < 70 ? sb.toString() : sb.substring(0, 69) + "...";
    }
  }

  /**
   * Повертає рядкове представлення вектора.
   * Обгортка методу {@link #printVector(int[], boolean) printVector(vector, false)}.
   *
   * @param vector вектор
   * @return рядкове представлення вектора
   */
  public static String printVector(int[] vector) {
    return printVector(vector, false);
  }

  /**
   * Повертає рядкове представлення матриці.
   * Обгортка методу {@link #printMatrix(int[][], boolean) printMatrix(matrix, false)}.
   *
   * @param matrix матриця
   * @return рядкове представлення матриці
   */
  public static String printMatrix(int[][] matrix) {
    return printMatrix(matrix, false);
  }
}
```

TaskDetails.java

```java
package ua.kpi.lab1;

import lombok.AllArgsConstructor;
import lombok.Data;

/**
* Клас, що представляє деталі задачі (потоку).
* Містить інформацію про назву, пріоритет та розмір стека.
*/
@AllArgsConstructor
@Data
public class TaskDetails {

 /**
  * Назва задачі.
  */
 private String name;

 /**
  * Пріоритет задачі.
  */
 private int priority;

 /**
  * Розмір стека задачі (в байтах).
  */
 private long stackSize;
}
```

InputType.java

```java
package ua.kpi.lab1;

/**
* Перечислення для вибору типу введення даних.
* Використовується для визначення, звідки брати вхідні дані.
*/
public enum InputType {

 /**
  * Введення даних користувачем з консолі.
  */
 CONSOLE,

 /**
  * Встановлення за замовчуванням (стандартні значення).
  */
 DEFAULT,
```

```
 /**
  * Випадкова генерація даних у визначених межах.
  */
 RANDOM,

 /**
  * Читання даних з файлу.
  */
 FILE
}
```

Lab1.java

```
package ua.kpi.lab1;

import java.util.Scanner;
import java.util.concurrent.CyclicBarrier;
import ua.kpi.lab1.thread.T1;
import ua.kpi.lab1.thread.T2;
import ua.kpi.lab1.thread.T3;
import ua.kpi.lab1.utils.Data;

/**
* Паралельне програмування
* Лабораторна робота 1.3 "Потоки в мові Java"
* Варіант 18
* <ul>
*   <li>1.11 F1: c = MAX(MA * MB) * (A * B)</li>
*   <li>2.29 F2: MF = (MG + MH) * (MK * ML) * (MG + ML)</li>
*   <li>3.7 F3: O = (P + R) * (MS * MT)</li>
* </ul>
* Мамченко Д.О. IO-25
* Дата 22.09.2024
*/
public class Lab1 {
 private static final CyclicBarrier barrier = new CyclicBarrier(3);
 private static InputType inputType;

 /**
  * Головний метод програми, який запускає три потоки для обчислення формул F1, F2 та
F3.
  *
  * @param args аргументи командного рядка
  */
 public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);

   initializeData(scanner);

   TaskDetails t1Details = getTaskDetails(scanner, "T1");
   TaskDetails t2Details = getTaskDetails(scanner, "T2");
   TaskDetails t3Details = getTaskDetails(scanner, "T3");
```

```
    T1 t1 = new T1(t1Details.getName(), t1Details.getPriority(), t1Details.getStackSize(),
        inputType, barrier);
    T2 t2 = new T2(t2Details.getName(), t2Details.getPriority(), t2Details.getStackSize(),
        inputType, barrier);
    T3 t3 = new T3(t3Details.getName(), t3Details.getPriority(), t3Details.getStackSize(),
        inputType, barrier);

    t1.start();
    t2.start();
    t3.start();

    try {
      t1.join();
      t2.join();
      t3.join();
    } catch (InterruptedException e) {
      throw new RuntimeException(e);
    }
  }

  /**
   * Ініціалізує дані для обчислень, включаючи розмірність N та тип введення даних.
   *
   * @param scanner об'єкт Scanner для зчитування введення користувача
   */
  private static void initializeData(Scanner scanner) {
    System.out.print("Введіть розмірність N: ");
    Data.N = scanner.nextInt();

    System.out.print("Виберіть тип задання елементів (CONSOLE, DEFAULT, RANDOM, FILE): ");
    inputType = InputType.valueOf(scanner.next().toUpperCase());

    if (inputType == InputType.RANDOM) {
      System.out.println("Рандомні значення будуть згенеровані в діапазоні [0, 100)");
    }

    System.out.println();
  }

  /**
   * Збирає деталі задачі для потоку, включаючи ім'я, пріоритет та розмір стека.
   *
   * @param scanner об'єкт Scanner для зчитування введення користувача
   * @param taskLabel мітка задачі (наприклад, "T1", "T2", "T3")
   * @return об'єкт TaskDetails, що містить ім'я, пріоритет та розмір стека задачі
   */
  private static TaskDetails getTaskDetails(Scanner scanner, String taskLabel) {
    System.out.print("Вкажіть ім'я задачі " + taskLabel + ": ");
    String name = scanner.next();
    System.out.print("Встановіть пріоритет задачі " + name + " (1-10): ");
    int priority = scanner.nextInt();
    System.out.print("Задайте розмір стека задачі " + name + ": ");
    long stackSize = scanner.nextLong();
```

```
    System.out.println();
    return new TaskDetails(name, priority, stackSize);
  }
}
```

Результат виконання:

Консоль:

```
Введіть розмірність N: 3
Виберіть тип задання елементів (CONSOLE, DEFAULT, RANDOM, FILE): console

Вкажіть ім'я задачі T1: f1
Встановіть пріоритет задачі f1 (1-10): 1
Задайте розмір стека задачі f1: 0

Вкажіть ім'я задачі T2: f2
Встановіть пріоритет задачі f2 (1-10): 5
Задайте розмір стека задачі f2: 0

Вкажіть ім'я задачі T3: f3
Встановіть пріоритет задачі f3 (1-10): 9
Задайте розмір стека задачі f3: 0

[f3] Введіть 3 елементів для вектора 'P':
3 3 3
P = (3, 3, 3)
[f3] Введіть 3 елементів для вектора 'R':
3 3 3
R = (3, 3, 3)
[f3] Введіть елементи для 3x3 матриці 'MS':
3 3 3
6 6 6
9 9 9
MS =
3  3  3
6  6  6
9  9  9
[f3] Введіть елементи для 3x3 матриці 'MT':
1 2 3
3 2 1
3 3 3
MT =
1  2  3
3  2  1
3  3  3

'f3' має всі дані та чекає на інші потоки.
```

[f1] Введіть 3 елементів для вектора 'A':
1 1 1
A = (1, 1, 1)
[f1] Введіть 3 елементів для вектора 'B':
1 2 1
B = (1, 2, 1)
[f2] Введіть елементи для 3x3 матриці 'MG':
2 2 2
4 4 4
6 6 6
MG =
2 2 2
4 4 4
6 6 6

[f2] Введіть елементи для 3x3 матриці 'MH':
2 4 6
8 10 8
2 4 6
MH =
2 4 6
8 10 8
2 4 6

[f2] Введіть елементи для 3x3 матриці 'MK':
2 2 2
2 2 2
2 2 2
MK =
2 2 2
2 2 2
2 2 2

[f2] Введіть елементи для 3x3 матриці 'ML':
4 4 4
2 2 2
4 4 4

ML =
4 4 4
2 2 2
4 4 4

'f2' має всі дані та чекає на інші потоки.
[f1] Введіть елементи для 3x3 матриці 'MA':
1 1 1
1 1 1
1 1 1
MA =
1 1 1
1 1 1
1 1 1

[f1] Введіть елементи для 3x3 матриці 'MB':
1 1 2
1 1 3
1 1 4

MB =
1 1 2
1 1 3
1 1 4

'f1' має всі дані та чекає на інші потоки.
Результат 'f1': 36
Час виконання обчислень 'f1': 0 мс
Результат 'f3': (378, 756, 1134)
Час виконання обчислень 'f3': 0 мс
Результат 'f2':
7920 7920 7920
16720 16720 16720
13200 13200 13200

Час виконання обчислень 'f2': 0 мс

Значення за замовчуванням (T1 – 1, T2 – 2, T3 – 3):

Введіть розмірність N: *100*
Виберіть тип задання елементів (CONSOLE, DEFAULT, RANDOM, FILE): *Default*

Вкажіть ім'я задачі T1: *func1*
Встановіть пріоритет задачі func1 (1-10): *5*
Задайте розмір стека задачі func1: *2000000*

Вкажіть ім'я задачі T2: *func2*
Встановіть пріоритет задачі func2 (1-10): *1*
Задайте розмір стека задачі func2: *2000000*

Вкажіть ім'я задачі T3: *func3*
Встановіть пріоритет задачі func3 (1-10): *10*
Задайте розмір стека задачі func3: *2000000*

P = (3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...
A = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
R = (3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...
B = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
MG =
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2...
MS =
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3...
MA =
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 11 1 11 1 11 1 11 1...
MH =
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2...
MT =
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3...
MB =
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 11 1 11 1 11 1 11 1...
'func1' має всі дані та чекає на інші потоки.
MK =
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2...
'func3' має всі дані та чекає на інші потоки.
ML =
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2...
'func2' має всі дані та чекає на інші потоки.
Результат 'func1': 10000

Час виконання обчислень 'func1': 14 мс
Результат 'func3': (540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000,
540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000,
540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000,
540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000, 540000)
Час виконання обчислень 'func3': 14 мс
Результат 'func2':
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 6400
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 640
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 640
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 6400
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 640
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 640
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 6400

...

64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 640
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 640
64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000 64000000

Час виконання обчислень 'func2': 19 мс

Генерація випадкових елементів:

Введіть розмірність N: *1000*
Виберіть тип задання елементів (CONSOLE, DEFAULT, RANDOM, FILE): *RANDOM*
Рандомні значення будуть згенеровані в діапазоні [0, 100)

Вкажіть ім'я задачі T1: *f1*
Встановіть пріоритет задачі f1 (1-10): *1*
Задайте розмір стека задачі f1: *0*

Вкажіть ім'я задачі T2: *f2*
Встановіть пріоритет задачі f2 (1-10): *5*
Задайте розмір стека задачі f2: *0*

Вкажіть ім'я задачі T3: *f3*
Встановіть пріоритет задачі f3 (1-10): *10*
Задайте розмір стека задачі f3: *0*

Задайте розмір стека задачі f2: 0

Вкажіть ім'я задачі T3: *f3*
Встановіть пріоритет задачі f3 (1-10): *10*
Задайте розмір стека задачі f3: *0*

A = (3, 96, 19, 18, 45, 76, 65, 19, 31, 37, 9, 95, 47...
P = (12, 57, 71, 10, 50, 22, 54, 17, 79, 84, 69, 5, 8...
B = (41, 21, 0, 1, 12, 17, 44, 46, 7, 89, 98, 49, 52,...
R = (79, 28, 61, 48, 24, 32, 17, 45, 1, 27, 49, 68, 8...
MA =
25 34 64 89 9 63 30 68 5 99 67 39 70 29 23 58 20 15 92 83 30 89 37 4 ...
MG =
38 12 53 7 31 9 44 89 24 33 55 98 2 31 88 2 54 66 69 20 84 13 31 32 8...
MS =
51 2 9 6 71 85 65 51 88 49 38 50 76 36 16 70 56 47 91 23 41 92 61 1 8...
MT =
94 12 98 60 15 22 81 79 91 88 1 4 80 73 79 69 88 32 40 17 48 97 60 55...
'f3' має всі дані та чекає на інші потоки.
MB =
47 68 38 12 43 10 92 68 96 17 66 91 43 3 14 30 7 15 7 86 20 37 94 2 1...
'f1' має всі дані та чекає на інші потоки.
MH =
43 44 64 91 63 80 60 86 20 34 34 12 23 17 76 30 23 55 22 20 21 53 47 ...
MK =
24 11 27 88 29 8 36 41 86 84 29 45 5 78 45 59 93 37 56 80 88 64 50 2 ...
ML =
47 75 40 84 5 8 97 69 30 12 90 45 81 82 40 8 27 0 61 20 24 16 44 67 4...
'f2' має всі дані та чекає на інші потоки.

Результат 'f1': 163407134
Час виконання обчислень 'f1': 4024 мс
Результат 'f3': (1288931015, 601832024, 657846997, -1871594328, -599955251, 265918324, -2045771418, -278304532, 18250911, 97002073
-1181517542, -477602610, 1646486625, -1710082818, 1649527118, 720525137, 1401272135, -962440335, -1857249379, 2080474480, -2099
1474203179, 25592808, 294073011, 1216256448, -596035278, -660359648, 1225673003, 198370354, 101456934, -1476159923, 165917289
-714841257, 1497437625, -288881935, -1115837089, 1462044099, -712154635, 240276596, 749097481, 1843758676, 790923510, -5896860
201874972, -1104104795, -181615105, 266849056, 1652646737, -1278455938, -1332091721, -1519336288, -1739804977, -82075324, -13709
1296669161, -1647595147, -1365593980, -452371423, -1589952302, 1516822529, 1845408932, 1365757723, -1758469167, -424782128, -23
1618238903, -1186322316, 160895462, 1553495552, -1351195852, 1990136873, -655047572, 177426675, 176384656, -725210326, -135072
-807761512, -1960026915, 1240583261, 1887554652, -2135096589, -1412498962, 1745084663, 668671811, 296152675, 477687765, -13742

...

797211015, -1662628395, -85813169, 811919935, 219351264, -1041518250, 564603206, 89796760, 1980802964, 1066789
-264759865, 857879005, -1482002305, 577573051, 753327274, -2061437131, 1341327043, 958472080, 46282554, -2847
271219206, 214394479, 1490550871, -301754940, 1582846622, 1960508403, 1740193558, -1754964732, -1892714104, 16
-83830679, 1922356572, -977615642, 1697856655, -1563470943, -986797209, 1870392615, -1556466245, -755698260,
1715853878, -692943124, -221206094, 1600042477, -1201736099, 57246554, 1600427907, 1976242156, -1977825987, -1
-1381075658, 24740940, 979644969, 772944821, 345659097, 1452870347, -1091197490, 1052039912, -291884545, -4115
1342779408, 1977737300, 1285808245, -266608272, -1996776581, -140941257, 1651029380, -2139999775, 2106275573,
-1938373900, -24806853, 1526629931, -499022900, -2094041314, 97883536, -1628073686, -2125294875, 1243452211,
-699281855, -1872432904, 198842546, 2122746187, -1452564694, -1299964313, 1489151223, 147404553, -600239383,
1933590729, 1307828323, -1934549076, 1230156007, -1580780993, 1127421445, -747095166, -575503139, -488544660,
-326880345, -842212637, -1527551749, -1969808021, 1313636007, -1403546765, -789454016, 1167237013, 892003785,
-813366190, 913744157, -630093273, -1905283572, -1680835338, -306119439, -1280062240, -2142826826, 658953382,
1980049094, -628721391, 678897615, -121140415, -266523734, 122023738, -1783063994, -159717014, -1092346270, 800
Час виконання обчислень 'f3': 2668 мс

. . .

644324130 1751903538 -1722466042 1942556543 -906516996 1660247957 1622961765 -1025854549 -116193958 -1839159092 -418358102 2
692320941 1309619678 -1005458964 -223488207 -1377637810 -1066757347 63119483 -1070214697 2037733745 -596360399 824168639 -12
1833069718 -2141873342 1448994517 -529710631 933727914 708749332 -280056714 -2020024326 -69739249 -1951960929 -1497262025 -58
-171326657 1619155033 933032688 -1534452002 -1545940313 -1709471342 737119471 1305971279 878394216 -1642925727 -468551934 -19
-238808841 1120866446 -1399177903 959346405 1199051128 502003571 -634223528 1261160640 1189246321 -1809781032 -1006572124 -80
-466136011 -442984750 555845371 -1928977713 1428306758 -1751044445 224177957 391209644 2127102006 -1403450622 -804937209 1919
-1121057636 -1639336362 -1061997290 -470608225 -1796961385 48988032 -829415404 -1965016270 144784385 1678830779 -227870785 -9

Час виконання обчислень 'f2': 6649 мс

Зчитування з файлів:

Введіть розмірність N: *3*
Виберіть тип задання елементів (CONSOLE, DEFAULT, RANDOM, FILE): *file*

Вкажіть ім'я задачі T1: *function1File*
Встановіть пріоритет задачі function1File (1-10): *10*
Задайте розмір стека задачі function1File: *200000*

Вкажіть ім'я задачі T2: *function2FIle*
Встановіть пріоритет задачі function2FIle (1-10): *5*
Задайте розмір стека задачі function2FIle: *200000*

Вкажіть ім'я задачі T3: *function3File*
Встановіть пріоритет задачі function3File (1-10): *1*
Задайте розмір стека задачі function3File: *200000*

P = (7, 8, 9)
MG =
1 0 0
0 1 0
0 0 1

A = (1, 2, 3)
B = (4, 5, 6)
R = (10, 11, 12)
MH =
4 4 4
5 5 5
6 6 6

MK =
3 3 3
7 7 7
9 9 9

MA =
1 2 3
4 5 6
7 8 9

```
MS =
1  1  1
2  2  2
3  3  3

MT =
7  7  7
8  8  8
9  9  9

MB =
9  8  7
6  5  4
3  2  1

ML =
2  2  2
2  2  2
2  2  2

'function1File' має всі дані та чекає на інші потоки.
'function2FIle' має всі дані та чекає на інші потоки.
'function3File' має всі дані та чекає на інші потоки.
Результат 'function3File': (1368, 2736, 4104)
Час виконання обчислень 'function3File': 0 мс
Результат 'function1File': 4416
Результат 'function2FIle':
3318  3318  3318
4284  4284  4284
5166  5166  5166

Час виконання обчислень 'function1File': 0 мс
Час виконання обчислень 'function2FIle': 0 мс
```

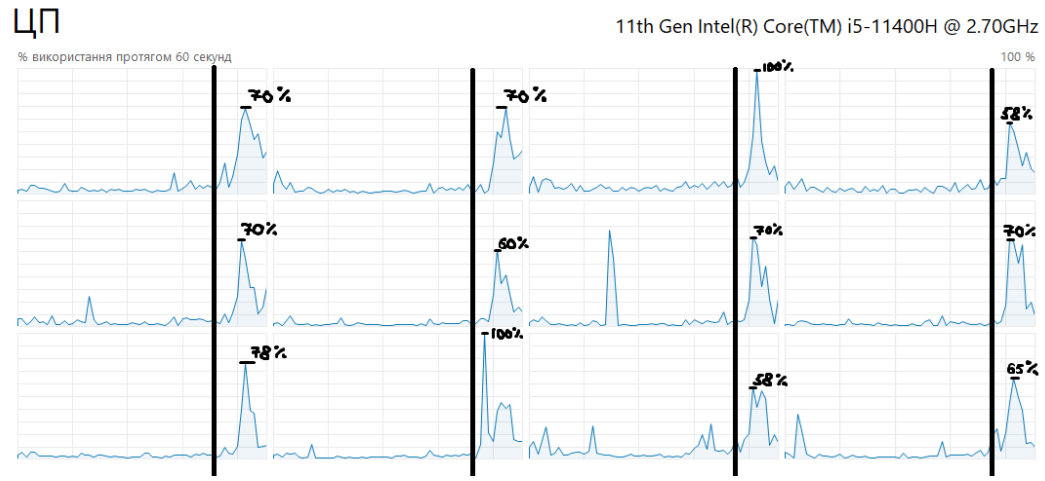
Аналіз результатів та висновок

Перше з чого хотілося б почати це проблеми введення даних з клавіатури та виведення результатів на екран. У багатопоточних програмах, коли кілька потоків виконують введення та виведення одночасно, результати можуть виводитися в некоректному порядку. На технічному рівні проблема змішаного введення/виведення виникає через те, що консоль (System.out або System.in в Java) **є глобальним ресурсом для всієї програми**. Якщо кілька потоків намагаються записати дані в консоль одночасно, операції виведення можуть переплітатися. Java не забезпечує автоматичної синхронізації при зверненні до консольного введення або виведення. Тому кожен потік може вводити/виводити дані без очікування завершення виведення іншими потоками. Введення та виведення даних в консолі не було синхронізоване, натомість лише зрозуміло розписаний текст, щоб

користувач розумів, звідки елементи появляються та куди елементи вводяться.

Надалі програма використовує **бар'єри** для синхронізації. У нашому випадку вони гарантують, що всі потоки досягнуть певної точки – закінчений ввід даних, перш ніж вони почнуть виконувати свої основні завдання – обчислення функцій.

Під час виконання багатопотокової програми для значення N = 1000 було проведено аналіз завантаження ядер процесора за допомогою Диспетчера задач ОС Windows. Що до завантаження процесора при всіх включених ядрах, то під час виконання обрахунків було помітно, що завантаження процесора було приблизно рівномірне, на здивування. Лише 2 ядра видали 100%. В середньому загруженість біля 70% на кожному ядру:



Результат при включених лише 3 ядрах був доволі очевидний. Всі загружені на 100%: