# Цель и задачи лабораторной работы

**Цель:** научиться использовать базовые возможности класса потоков Thread.

**Задачи:**

1. Научиться создавать потоки Thread;
2. Научиться использовать массивы потоков;
3. Научиться осуществлять мониторинг потоков.

# Реализация индивидуального задания

Согласно варианту задания, требуется:

Реализовать метод для запуска в отдельном потоке (в соответствии с индивидуальным вариантом).

Создать делегат для представления метода (если требуется).

В основной программе (функция Main( )) реализовать создание массива потоков. Метод, выполняющийся в параллельных потоках должен выводить информацию о ходе своего выполнения в консоль приложения.

В моём случае был метод преобразования матрицы случайных чисел: элементы кратные 3 заменяются на квадрат соответствующего элемента.

## Листинг программного кода

***namespace*** lab5

{

***class*** Program

    {

*// Метод для преобразования матрицы (работает в отдельном потоке)*

**static** **void** ProcessMatrixPart(**int**[,] matrix, **int** startRow, **int** endRow, **int** threadId)

        {

            Console.WriteLine($"Поток {threadId} начал обработку строк с {startRow **+** 1} по {endRow **+** 1}");

*for* (**int** i **=** startRow; i **<=** endRow; i**++**)

            {

*for* (**int** j **=** 0; j **<** matrix.GetLength(1); j**++**)

                {

*// Если элемент кратен 3, заменяем на квадрат*

*if* (matrix[i, j] **%** 3 **==** 0)

                    {

**int** original **=** matrix[i, j];

                        matrix[i, j] **=** original **\*** original;  *// Аве ссылкам*

                        Console.WriteLine($"Поток {threadId}: [{i},{j}] {original} -> {matrix[i, j]}");

                    }

*else*

                    {

*// Console.WriteLine($"Поток {threadId}: [{i},{j}] {matrix[i, j]} -> {matrix[i, j]}");*

                    }

                }

            }

            Console.WriteLine($"Поток {threadId} завершил работу");

        }

*// Метод для создания и запуска потоков*

**static** **void** TransformMatrixWithThreads(**int**[,] matrix, **int** threadCount)

        {

**int** rows **=** matrix.GetLength(0);

**int** rowsPerThread **=** rows **/** threadCount;

            Thread[] threads **=** **new** Thread[threadCount];

*for* (**int** i **=** 0; i **<** threadCount; i**++**)

            {

**int** startRow **=** i **\*** rowsPerThread;

**int** endRow **=** (i **==** threadCount **-** 1) **?** rows **-** 1 **:** startRow **+** rowsPerThread **-** 1;

**int** threadId **=** i **+** 1;

*// Создаем поток для обработки своей части матрицы*

                threads[i] **=** **new** Thread(() **=>** ProcessMatrixPart(matrix, startRow, endRow, threadId));

                threads[i].Start();

            }

*// Ждем завершения всех потоков*

*foreach* (Thread thread *in* threads)

            {

                thread.Join();

            }

        }

*// Вспомогательные методы*

**static** **void** PrintMatrix(**int**[,] matrix, **string** title)

        {

            Console.WriteLine($"\n{title}:");

*for* (**int** i **=** 0; i **<** matrix.GetLength(0); i**++**)

            {

*for* (**int** j **=** 0; j **<** matrix.GetLength(1); j**++**)

                {

                    Console.Write($"{matrix[i, j],4} ");

                }

                Console.WriteLine();

            }

        }

**static** **void** FillMatrixWithRandomNumbers(**int**[,] matrix, **int** min **=** 1, **int** max **=** 20)

        {

            Random rand **=** **new**();

*for* (**int** i **=** 0; i **<** matrix.GetLength(0); i**++**)

            {

*for* (**int** j **=** 0; j **<** matrix.GetLength(1); j**++**)

                {

                    matrix[i, j] **=** rand.Next(min, max);

                }

            }

        }

**public** **static** **void** Main(**string**[] args)

        {

*// Тут вполне можно было бы использовать Task.Run вместо потоков, которые больше используются для долгих операций / нужен foreground-поток*

**const** **int** ROWS **=** 5;

**const** **int** COLS **=** 5;

**const** **int** THREAD\_COUNT **=** 3;

*// Создаем и заполняем матрицу случайными числами*

**int**[,] matrix **=** **new** **int**[ROWS, COLS];

            FillMatrixWithRandomNumbers(matrix);

            PrintMatrix(matrix, "Исходная матрица");

*// Преобразуем матрицу с использованием потоков*

            Console.WriteLine("\n=== Начало преобразования матрицы ===");

            TransformMatrixWithThreads(matrix, THREAD\_COUNT);

            Console.WriteLine("=== Преобразование завершено ===\n");

            PrintMatrix(matrix, "Преобразованная матрица");

        }

    }

}

## Описание кода

**1. Многопоточная обработка матрицы**

* **Разделение матрицы** на части по строкам между потоками
* **3 потока** обрабатывают матрицу 5×5 параллельно
* **Каждый поток** получает свой диапазон строк

**2. Управление потоками**

* **Создание потоков:** new Thread(() => ProcessMatrixPart(...))
* **Запуск:** thread.Start()
* **Ожидание завершения:** thread.Join()

**3. Вспомогательные функции**

* FillMatrixWithRandomNumbers - заполнение случайными значениями
* PrintMatrix - форматированный вывод матрицы
* **Детальное логирование** процесса обработки

## Результат работы программы

Исходная матрица:

15 11 16 5 17

12 14 11 5 3

6 2 14 10 14

12 14 19 15 15

12 9 8 18 8

=== Начало преобразования матрицы ===

Поток 2 начал обработку строк с 2 по 2

Поток 2: [1,0] 12 -> 144

Поток 2: [1,4] 3 -> 9

Поток 2 завершил работу

Поток 1 начал обработку строк с 1 по 1

Поток 3 начал обработку строк с 3 по 5

Поток 1: [0,0] 15 -> 225

Поток 3: [2,0] 6 -> 36

Поток 3: [3,0] 12 -> 144

Поток 3: [3,3] 15 -> 225

Поток 3: [3,4] 15 -> 225

Поток 3: [4,0] 12 -> 144

Поток 3: [4,1] 9 -> 81

Поток 3: [4,3] 18 -> 324

Поток 3 завершил работу

Поток 1 завершил работу

=== Преобразование завершено ===

Преобразованная матрица:

225 11 16 5 17

144 14 11 5 9

36 2 14 10 14

144 14 19 225 225

144 81 8 324 8

# Контрольные вопросы

1. **В каком пространстве имен определен класс Thread? Поясните назначение класса Thread.** 
   1. Пространство имен: System.Threading
   2. Назначение:
      1. Создание и управление отдельными потоками выполнения
      2. Параллельное выполнение кода
      3. Контроль приоритетов и состояния потоков
2. **Как получить идентификатор текущего потока?** 
   1. int currentThreadId = Thread.CurrentThread.ManagedThreadId;
3. **Какой метод осуществляет запуск метода на выполнение в потоке?** 
   1. Thread thread = new Thread(MyMethod);  
      thread.Start();
4. **Для чего необходимы делегаты при использовании класса Thread?** 
   1. Указание метода, который будет выполняться в потоке
   2. Передача параметров через ParameterizedThreadStart
   3. Организация callback-ов при завершении потока
5. **Как организовать несколько потоков в программе?**
   1. Массив потоков
   2. Ожидание завершения