МИНИCTEPCTBO НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии

Департамент цифровых, робототехнических систем и электроники

Отчет к практической работе № 19

Тема: **«Создание и запуск задач»**

Дисциплина: **«Алгоритмы и структуры данных»**

|  |  |  |
| --- | --- | --- |
|  |  | **Выполнил:** |
|  |  | Студент группы ПИЖ-б-о-23-1, направление подготовки: 09.03.04 «Программная инженерия»  Панчешный Александр Алексеевич  **Проверил:**  Доцент ДЦРСИЭ  Николаев Евгений Иванович |

Ставрополь 2025

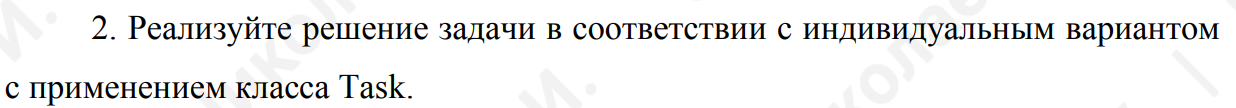
**Цель лабораторной работы:**

Изучить механизм работы с классами задач.

**Задачи лабораторной работы:**

1. Изучить возможности класса Task;
2. Научиться создавать и запускать задачи;
3. Научиться работать с массивами задач.

**Ход работы**

**Индивидуальное задание:**





**Исходный код:**

using System;

using System.Threading.Tasks;

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Программа подсчета четных элементов в матрице с использованием Task");

*// Создаем матрицу 5x5 со случайными значениями*

int[,] matrix = GenerateRandomMatrix(5, 5);

PrintMatrix(matrix);

*// Создаем и запускаем задачу*

Task<int> countTask = Task.Run(() => CountEvenElements(matrix));

*// Выводим сообщение о процессе выполнения*

Console.WriteLine("\nПодсчет четных элементов...");

*// Ожидаем завершения задачи и получаем результат*

countTask.Wait();

int evenCount = countTask.Result;

Console.WriteLine($"\nКоличество четных элементов: {evenCount}");

Console.WriteLine("Нажмите Enter для выхода...");

Console.ReadLine();

}

*// Метод подсчета четных элементов в матрице*

static int CountEvenElements(int[,] matrix)

{

int count = 0;

int rows = matrix.GetLength(0);

int cols = matrix.GetLength(1);

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

{

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

{

if (matrix[i, j] % 2 == 0)

{

count++;

}

}

}

return count;

}

*// Генерация матрицы случайных чисел*

static int[,] GenerateRandomMatrix(int rows, int cols)

{

Random rand = new Random();

int[,] matrix = new int[rows, cols];

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

{

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

{

matrix[i, j] = rand.Next(1, 100); *// числа от 1 до 99*

}

}

return matrix;

}

*// Вывод матрицы в консоль*

static void PrintMatrix(int[,] matrix)

{

Console.WriteLine("\nСгенерированная матрица:");

int rows = matrix.GetLength(0);

int cols = matrix.GetLength(1);

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

{

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

{

Console.Write($"{matrix[i, j],3} ");

}

Console.WriteLine();

}

}

}

**Контрольные вопросы**

**1. Что такое класс Task? Чем класс Task отличается от класса Thread?**

**Класс Task** - это абстракция для асинхронной операции в .NET, представляющая собой единицу работы, которая может выполняться асинхронно. Это часть TPL (Task Parallel Library).

**Отличия от Thread**:

* **Task** использует пул потоков, а **Thread** создает новый поток
* **Task** поддерживает продолжения (continuations) через ContinueWith
* **Task** имеет встроенную поддержку отмены через CancellationToken
* **Task** упрощает обработку ошибок (исключения агрегируются)
* **Task** более эффективен для коротких операций

Пример:

Task.Run(() => Console.WriteLine("Это задача")); // Использует пул потоков

new Thread(() => Console.WriteLine("Это поток")).Start(); // Создает новый поток

**2. Как получить идентификатор текущей задачи? Как получить идентификатор текущего потока?**

**Идентификатор задачи**:

int taskId = Task.CurrentId ?? -1; // Возвращает null для главного потока

**Идентификатор потока**:

int threadId = Thread.CurrentThread.ManagedThreadId;

Разница:

* Task.CurrentId уникален только в пределах текущего контекста выполнения
* ManagedThreadId уникален в пределах процесса

**3. Опишите возможные методы создания и запуска задач.**

Основные способы:

1. **Через Task.Run** (для пула потоков):

Task.Run(() => DoWork());

1. **Через конструктор Task + Start**:

var task = new Task(() => DoWork());

task.Start();

1. **Task.Factory.StartNew** (с дополнительными параметрами):

Task.Factory.StartNew(() => DoWork(),

CancellationToken.None,

TaskCreationOptions.None,

TaskScheduler.Default);

1. **Для возвращающих значение**:

Task<int> task = Task.Run(() => CalculateValue());

1. **Асинхронные методы**:

async Task MyMethodAsync()

{

await Task.Delay(1000);

}

**4. Какие параметры задачи может установить программист при реализации задачи? Опишите возможные варианты применения параметра типа TaskCreationOptions.**

**Основные параметры задачи**:

* CancellationToken - для отмены задачи
* TaskCreationOptions - флаги создания задачи
* TaskScheduler - планировщик выполнения
* **TaskCreationOptions**:
* None - поведение по умолчанию
* LongRunning - указывает, что задача будет долгой (не использовать пул потоков)
* PreferFairness - более честное планирование
* AttachedToParent - связь с родительской задачей
* DenyChildAttach - запрет дочерним задачам прикрепляться
* HideScheduler - игнорировать текущий планировщик
* RunContinuationsAsynchronously - продолжения выполняются асинхронно

Пример использования:

var task = new Task(() => DoLongWork(),

CancellationToken.None,

TaskCreationOptions.LongRunning);

Эти параметры позволяют оптимизировать выполнение задач под конкретные сценарии.