# ОТЧЕТ ДЕНЬ 23

**«ЗАДАНИЯ МНОГОПОТОЧНОСТЬ»**

# Задание 1. Создание потокобезопасных методов вывода цветного текста в консоль, которые можно будет использовать в последующих упражнениях.

**Задание 1.1.** Создайте статический класс ColoredConsole. В этом классе создайте два статических метода: Write и WriteLine со следующими определениями:

public static void WriteLine(ConsoleColor color, String format, params object[] list)

public static void Write(ConsoleColor color, String format, params object[] list)

Реализация каждого метода должна устанавливать цвет текста в консоли в соответствии с первым параметром, а остальные параметры передавать без изменения одноименным методам из класса System.Console.

*Для проверки работы методов выведите на консоль несколько строк разным цветом.*

**Задание 1.2.** Создайте метод обратного вызова с одним параметром, соответствующий делегату ParameterizedThreadStart, то есть:

static void Имя(object o)

Этот метод должен выводить красным цветом слово Red (или Красный).

Используя класс System.Threading.Timer, создайте таймер, который будет запускать только что созданный метод каждые 200 миллисекунд.

Аналогично создайте еще один метод и таймер, чтобы каждые 200 миллисекунд выводилось слово Green или Зеленый.

Добавьте в конце метода Main какую-либо задержку, чтобы программа не завершалась немедленно после создания таймеров (можно использовать Thread.Sleep или Console.ReadLine).

Запустите программу на выполнение и наблюдайте за выводом. Временами должны появляться фрагменты с неправильным цветом текста:



*Примечание. Если фрагменты с неправильным цветом текста не появляются, можно попытаться изменить интервалы срабатывания таймеров.*

*Где в коде программы скрыт источник проблемы?*

**Задание 1.3.** Используйте критические секции в методах Write и WriteLine класса ColoredConsole, чтобы сделать их потокобезопасными (для этого можно воспользоваться классом Monitor или ключевым словом lock).

Убедитесь, что теперь все цвета отображаются правильно.

**Задание 1.4.** Замените все методы обратного вызова на один, принимающий цвет в виде параметра (второй параметр конструктора Timer). Так как теперь цвета отображаются всегда правильно, то выводить цвет необязательно. Заставьте метод обратного вызова выводить номер управляемого потока. Основной поток пусть выводит серые точки каждые 50 миллисекунд, пока не будет нажата любая клавиша.

**Задание 2.** Синхронизация доступа к ресурсам.

**Задание 2.1.** В классе Program создайте статическое свойство с именем Million100 типа int. Создайте и запустите на выполнение 20 фоновых потоков, каждый из которых 5 миллионов раз увеличивает значение поля Million100 (в конечном итоге значение этого поля должно стать сто миллионов). Выведите значение поля на экран (как вариант, можно выводить значение каждые 50 мс).

*Верный ли результат получен? Если нет, то соотнесите его с количеством ядер вашего процессора.*

**Задание 2.2.** Добавьте блокировку на основе критической секции, убедитесь в правильности результата вычислений.

**Задание 2.3.** Создайте метод, соответствующий делегату ParameterizedThreadStart, который каждую секунду (а также в начале и конце работы) выводит свое имя, продолжительность работы генерируется случайно в диапазоне 2-5 секунд. В качестве параметра метод принимает событие AutoResetEvent, которое он должен установить в сигнальное состояние в конце работы. Создайте три экземпляра потока и запустите их, передав в качестве параметра экземпляры класса AutoResetEvent. Программа должна сообщить, когда все три потока завершат работу.

**Задание 2.4.** Создайте метод, соответствующий делегату ThreadStart, который, как и в задании 3.3, каждую секунду (а также в начале и конце работы) выводит свое имя, продолжительность работы генерируется случайно в диапазоне 2-5 секунд. Работа должна выполняться последовательно, то есть, пока один поток не выведет в консоль все свои сообщения, никакой другой поток не может ничего выводить. Создайте три экземпляра потока, присвойте уникальные имена каждому потоку и запустите их.

Программа должна автоматически завершиться, когда все потоки отработают.

*Указание: использовать экземпляр класса System.Threading.Mutex.*

**Задание 2.5.** Измените программу из задания 2.4 так, чтобы только не более двух потоков могли одновременно выводить сообщения. Создайте три экземпляра потока, присвойте уникальные имена каждому потоку и запустите их. Программа должна автоматически завершиться, когда все потоки отработают.

*Указание: использовать экземпляр класса System.Threading.Semaphore.*

**Задание 3. Задачи.**

**Задание 3.1**. Создайте метод, соответствующий делегату Action (void имя()), который с интервалом в одну секунду выводит три сообщения о начале, продолжении и завершении работы задачи. Создайте три задачи и запустите их на выполнение.

**Задание 3.2.** Измените программу из задания 3.1, чтобы все три задачи запускались по очереди, то есть, завершение первой задачи должно автоматически запускать вторую и т. д.

**Задание 4.** Асинхронное выполнение методов.

**Задание 4.1.** Создайте метод, который принимает в качестве параметра строку со словом, находит в словаре соответствующую словарную статью и возвращает ее в виде одной строки. В начале и конце работы метода он должен выводить в консоль соответствующие сообщения.

*Примечание. На компьютерах с SSD-диском этот метод может выполняться слишком быстро, чтобы был заметен эффект. В этом случае придется добавить искусственную задержку с помощью метода Thread.Sleep.*

Объявите делегат, соответствующий этому методу, и создайте экземпляр делегата. Вызовите экземляр делегата синхронно. Для создания ощущения времени можно создать отдельный поток или задачу, которые будут выводить точки каждые 50 мс.

**Задание 4.2.** Вызовите экземляр делегата из задания 4.1 асинхронно.

Для определения готовности результата используйте свойство IAsyncResult.IsCompleted. Проверку свойства проводите раз в 50 мс, при неготовности выводите красный дефис, по готовности – сам результат.

**Задание 4.3.** Создайте асинхронную версию метода из задания 4.1. В имени метода используйте суффикс Async, тип возвращаемое значение Task<String>. Асинхронная версия метода создает, запускает задачу, выполняющую синхронную версию метода, и возвращает объект созданной задачи.

Снова выполните программу.

*В ОТЧЕТЕ ПРЕДСТАВИТЬ:*

Номер задания. Условие задания. Код программы (НЕ СКРИНОМ!).

Результат выполнения программы (скрин)