ТЕМА 1: СТВОРЕННЯ ПОТОКІВ. КЛАС THREAD. ПРІОРИТЕТНІ ТА ФОНОВІ ПОТОКИ. ПРІОРИТЕТИ ПОТОКІВ

Розрізняють два різновиди багатозадачності: на основі процесів і на основі потоків. У зв'язку з цим важливо розуміти відмінності між ними. Процес фактично являє собою виконувану програму. Тому багатозадачність на основі процесів - це засіб, завдяки якому на комп'ютері можуть паралельно виконуватися дві програми і більше. Так, багатозадачність на основі процесів дозволяє одночасно виконувати програми текстового редактора, електронних таблиць і перегляду вмісту в Інтернеті. При організації багатозадачності на основі процесів програма є найменшою одиницею коду, виконання якої може координувати планувальник задач.

Потік являє собою координовану одиницю виконуваного коду. При організації багатозадачності на основі потоків у кожного процесу повинен бути принаймні один потік, хоча їх може бути і більше. Це означає, що в одній програмі одночасно можуть вирішуватися дві задачі і більше. Наприклад, текст може форматуватися в редакторі тексту одночасно з його виведенням на друк, за умови, що обидві ці дії виконуються у двох окремих потоках.

Відмінності у багатозадачності на основі процесів і потоків можуть бути зведені до наступного: багатозадачність на основі процесів організовується для паралельного виконання програм, а багатозадачність на основі потоків - для паралельного виконання окремих частин однієї програми.

Головна перевага багатопотокової обробки полягає у тому, що вона дозволяє писати програми, які працюють дуже ефективно завдяки можливості вигідно використовувати час простою, що неминуче виникає у ході виконання більшості програм. А завдяки багатопотоковій обробці програма може розв'язувати якусь іншу задачу під час вимушеного простою.

Потік може перебувати в одному з декількох станів. В цілому, потік може бути таким, що виконуються; готовим до виконання, як тільки він отримає час і ресурси ЦП; призупиненим, тобто таким, що тимчасово не виконується; відновленим у подальшому; заблокованим в очікуванні ресурсів для свого виконання; а також завершеним, коли його виконання закінчене і не може бути відновлено.

У середовищі .NET Framework визначені два різновиди потоків: пріоритетний і фоновий. За замовчуванням потік, що створюється, автоматично стає пріоритетним, але його можна зробити фоновим. Єдина відмінність пріоритетних потоків від фонових полягає в тому, що фоновий потік автоматично завершується, якщо в його процесі зупинені всі пріоритетні потоки.

Класи, які підтримують багатопотокове програмування, визначені у просторі імен `System.Threading`. Тому будь-яка багатопотокова програма на C# включає в себе наступний рядок коду:

```
using System.Threading;
```

Система багатопотокової обробки ґрунтується на класі `Thread`, який інкапсулює потік виконання. Клас `Thread` є запечатаним, тобто він не може успадковуватися. У класі `Thread` визначено ряд методів і властивостей, призначених для керування потоками.

Створення і запуск потоку. Для створення потоку досить отримати екземпляр об'єкта типу `Thread`. Найпростіша форма конструктора класу `Thread`:

```
public Thread(ThreadStart Run),
```

де Run - це ім'я методу, що викликається з метою почати виконання потоку, а ThreadStart - делегат, що визначений в середовищі .NET Framework:

```
public delegate void ThreadStart()
```

Отже, метод, що вказується в якості точки входу в потік, повинен мати у якості типу, що повертається, void і не приймати жодних аргументів.

Новостворений потік не почне виконуватися до тих пір, поки не буде викликано його метод Start(), який визначається в класі Thread.

Розпочавшись одного разу, потік буде виконуватися до тих пір, поки не відбудеться повернення з методу, на який вказує Run. Таким чином, після повернення з цього методу потік автоматично припиняється. Якщо ж спробувати викликати метод Start() для потоку, який вже почався, це призведе до генерації винятку ThreadStateException.

Визначимо клас MyThread, призначений для створення другого потоку виконання. У методі Run() цього класу організовується цикл для підрахунку від 0 до 9.

```
class MyThread
{
      public int Count;
      string thrdName;

      public MyThread(string name)
      {
            Count = 0;
            thrdName = name;
      }

      //Точка входу у потік
      public void Run()
      {
            Console.WriteLine(thrdName + " starting.");

            do
            {
                  // Призупинення даного потоку
                  Thread.Sleep(500);
                  Console.WriteLine("In the thread "+thrdName+", Count="+Count);
                  Count++;
            } while (Count < 10);
            Console.WriteLine(thrdName + " is completed.");
      }
}
```

Метод Sleep() зумовлює призупинення того потоку, з якого він був викликаний, на визначений період часу, що вказується у мілісекундах. Коли призупиняється один потік, може виконуватися інший.

У методі Main() новий об'єкт типу Thread створюється за допомогою наступної послідовності операторів:

```
//Конструюємо об'єкт типу MyThread (поток)
MyThread mt=new MyThread("Thread №1");

//Конструюємо потік із методу RUN цього об'єкта
Thread newThrd =new Thread(mt.Run);
```

```
//Починаємо виконання цього потоку
newThrd.Start();
```

Спочатку створюється об'єкт типу `MyThread`. Потім цей об'єкт використовується для створення об'єкта типу `Thread`, для чого конструктору цього об'єкта в якості точки входу передається метод `mt.Run()`. І нарешті, виконання потоку починається з виклику методу `Start()`.Завдяки цьому метод `mt.Run()` виконується в своєму власному потоці. Після виклику методу `Start()` виконання основного потоку повертається до методу `Main()`, де починається цикл `do-while`. Обидва потоки продовжують виконуватися, спільно використовуючи ЦП, аж до закінчення циклу.

```
Main thread is beginning.
.Thread №1 starting.
.....In the thread Thread №1, Count=0
.....In the thread Thread №1, Count=1
....In the thread Thread №1, Count=2
.....In the thread Thread №1, Count=3
.....In the thread Thread №1, Count=4
.....In the thread Thread №1, Count=5
.....In the thread Thread №1, Count=6
.....In the thread Thread №1, Count=7
.....In the thread Thread №1, Count=8
.....In the thread Thread №1, Count=9
Thread №1 is completed.
Main thread is completed.
```

У цій програмі головний потік перевіряє лічильник і завершує своє виконання, коли його значення досягне 10.

Удосконалення багатопотокової програми. По-перше, можна зробити так, щоб виконання потоку починалося відразу ж після його створення. Для цього достатньо отримати екземпляр об'єкта типу `Thread` в конструкторі класу `MyThread`. І по-друге, в класі `MyThread` зовсім не обов'язково зберігати ім'я потоку, оскільки для цієї мети в класі `Thread` спеціально визначене властивість `Name`.

```
public string Name { get; set; }
```

```
class MyThread
{
	public int Count;
	public Thread newThrd;

	public MyThread(string name)
	{
		Count = 0;
		//Конструюємо потік з методу цього об'єкта
		newThrd = new Thread(this.Run);
		newThrd.Name = name;
		//Почати виконання потоку
		newThrd.Start();
	}

	//Точка входження у потік
	public void Run()
	{
		Console.WriteLine(newThrd.Name + " starting.");
		do
		{
			Thread.Sleep(500);
			Console.WriteLine("In the thread " + newThrd.Name + ", Count=" + Count);
			Count++;
		} while (Count<10);
		Console.WriteLine(newThrd.Name + " is completed.");
	}
```

```
}

class Program
{
	static void Main(string[] args)
	{
		Console.WriteLine("Main thread is beginning.");

		//Конструюємо об'єкт типу MyThread (поток)
		MyThread mt=new MyThread("Thread №1");

		do
		{
			Console.Write(".");
			Thread.Sleep(100);
		} while (mt.Count != 10);

		Console.WriteLine("Main thread is completed.");
		Console.ReadLine();
	}
}
```

Створення декількох потоків. У програмі можна породити стільки потоків, скільки буде потрібно.

```
	class Program
	{
		static void Main()
		{
			Console.WriteLine("Main thread is beginning.");

			//Конструюємо декілька потоків (об'єктів типу MyThread)
			MyThread mt1=new MyThread("Thread №1");
			MyThread mt2=new MyThread("Thread №2");
			MyThread mt3=new MyThread("Thread №3");

			do
			{
				Console.Write(".");
				Thread.Sleep(100);
			} while (mt1.Count < 10 || mt2.Count < 10 || mt3.Count < 10);

			Console.WriteLine("Main thread is completed.");
			Console.ReadLine();
		}
	}
```

```
Main thread is beginning.
Thread №1 starting.
Thread №2 starting.
.Thread №3 starting.
....In the thread Thread №1, Count=0
In the thread Thread №2, Count=0
.In the thread Thread №3, Count=0
....In the thread Thread №1, Count=1
In the thread Thread №2, Count=1
In the thread Thread №3, Count=1
.....In the thread Thread №1, Count=2
In the thread Thread №2, Count=2
In the thread Thread №3, Count=2
.....In the thread Thread №1, Count=3
In the thread Thread №2, Count=3
In the thread Thread №3, Count=3
.....In the thread Thread №1, Count=4
In the thread Thread №2, Count=4
In the thread Thread №3, Count=4
```

```
.....In the thread Thread №1, Count=5
In the thread Thread №2, Count=5
In the thread Thread №3, Count=5
.....In the thread Thread №1, Count=6
In the thread Thread №2, Count=6
In the thread Thread №3, Count=6
.....In the thread Thread №1, Count=7
In the thread Thread №2, Count=7
In the thread Thread №3, Count=7
.....In the thread Thread №1, Count=8
In the thread Thread №2, Count=8
In the thread Thread №3, Count=8
.....In the thread Thread №1, Count=9
Thread №1 is completed.
In the thread Thread №2, Count=9
Thread №2 is completed.
In the thread Thread №3, Count=9
Thread №3 is completed.
Main thread is completed.
```

Після того як всі три потоки почнуть виконуватися, вони будуть спільно використовувати ЦП.

Визначення моменту завершення потоку. Нерідко виявляється корисно знати, коли саме завершується потік. В попередніх прикладах програм для цієї мети відстежувалося значення змінної `Count`. У класі `Thread` є два інші засоби для визначення моменту закінчення потоку. З цією метою можна, перш за все, опитати доступну тільки для читання властивість `IsAlive`, що визначається наступним чином.

```
public bool IsAlive { get; }
```

Властивість `IsAlive` повертає логічне значення true, якщо потік, для якого вона викликається, продовжує виконується.

```
do
{
       Console.Write(".");
       Thread.Sleep(100);
} while (mt1.newThrd.IsAlive && mt2.newThrd.IsAlive && mt3.newThrd.IsAlive);
```

Ще один спосіб відстеження моменту закінчення полягає у виклику методу `Join()`.Метод `Join()` очікує до тих пір, поки потік, для якого він був викликаний, не завершиться.

```
class Program
       {
              static void Main(string[] args)
              {
                     Console.WriteLine("Main thread is beginning.");

                     //Сконструювати три потока
                     MyThread mt1=new MyThread("mt1");
                     MyThread mt2=new MyThread("mt2");
                     MyThread mt3=new MyThread("mt3");

                     //Приєднати перший потік
                     mt1.Thrd.Join();
                     Console.WriteLine("Thread №1 is joined");

                     //Блокувати викликаючий потік до завершення потоку
                     mt2.Thrd.Join();
                     Console.WriteLine("Thread №2 is joined");
```

```
                //Приєднати другий потік
                mt3.Thrd.Join();
                Console.WriteLine("Thread №3 is joined");

                Console.WriteLine("Main thread is completed");
                Console.ReadLine();
        }
}
```

Передача аргументу потоку. Аргумент може бути переданий потоку за допомогою використання інших форм методу `Start()`, конструктора класу `Thread`, а також методу, що служить точкою входу в потік.

Аргумент передається потоку в такій формі методу `Start()`.

```
public void Start(object параметр)
```

Отже, для того щоб передати аргумент потоку, досить передати його методу `Start()`.

Для застосування параметризованої форми методу Start () буде потрібна наступна форма конструктора класу Thread:

```
public Thread (ParameterizedThreadStart Run)
```

де `Run` позначає метод, що викликається з метою почати виконання потоку.

В цій формі конструктора `Run` має тип `ParameterizedThreadStart`, а не `Threadstart`, як в формі, що використовувалася в попередніх прикладах. В даному випадку `ParameterizedThreadStart` є делегатом, що декларується в такий спосіб:

```
public delegatfe void ParameterizedThreadStart (object obj)
```

Цей делегат приймає аргумент типу `object`. Тому для правильного застосування даної форми конструктора класу `Thread` у методу, що служить точкою входу в потік, повинен бути параметр типу `object`.

```
namespace PassArg
{
        class MyThread
        {
                public int Count;
                public Thread Thrd;

                public MyThread(string name, int num)
                {
                        Count = 0;
                        Thrd = new Thread(this.Run);
                        Thrd.Name = name;

                        //Передати параметр num методу Start()
                        Thrd.Start(num);
                }

                //У цій формі методу Run() вказується параметр типу object
                void Run (object num)
                {
                        Console.WriteLine(Thrd.Name + " exequting to number " + num);

                        do
                        {
                                Thread.Sleep(500);
                                Console.WriteLine("In thread " + Thrd.Name + ", Count = " + Count);
                                Count++;
                        } while (Count <= (int)num);
                        Console.WriteLine(Thrd.Name + " completed.");
                }
        }
        class Program
        {
                static void Main(string[] args)
```

```
        {
            Console.WriteLine("Main thread is beginning.");

            MyThread mt1=new MyThread("mt1",5);
            MyThread mt2=new MyThread("mt2",8);

            do
            {
                Console.Write(".");
                Thread.Sleep(100);
            } while (mt1.Thrd.IsAlive | mt2.Thrd.IsAlive);

            Console.WriteLine("Main Thread is completed.");
            Console.ReadLine();
        }
    }
}
```

Результат виконання програми:

```
Main thread is beginning.
mt2 exequting to number 8
.mt1 exequting to number 5
....In thread mt2, Count = 0
In thread mt1, Count = 0
.....In thread mt2, Count = 1
In thread mt1, Count = 1
.....In thread mt2, Count = 2
In thread mt1, Count = 2
.....In thread mt2, Count = 3
In thread mt1, Count = 3
.....In thread mt2, Count = 4
In thread mt1, Count = 4
.....In thread mt2, Count = 5
In thread mt1, Count = 5
mt1 completed.
.In thread mt2, Count = 6
.....In thread mt2, Count = 7
.....In thread mt2, Count = 8
mt2 completed.
Main Thread is completed.
```

Як випливає з наведеного вище результату, перший потік повторюється п'ять разів, а другий - вісім раз. Число повторень вказується в конструкторі класу `MyThread` і потім передається методу `Run()`, який служить в якості точки входу в потік, за допомогою параметризованої форми `ParameterizedThreadStart` методу `Start()`.

Властивість IsBackground. У середовищі .NET Framework визначені два різновиди потоків: пріоритетний і фоновий. Єдина відмінність між ними полягає в тому, що процес не завершиться до тих пір, поки не закінчиться пріоритетний потік, тоді як фонові потоки завершуються автоматично після закінчення всіх пріоритетних потоків. За замовчуванням потік, що створюється, стає пріоритетним. Але його можна зробити фоновим, використовуючи властивість `IsBackground`, визначену в класі `Thread`, в такий спосіб:

```
public bool IsBackground { get; set; }
```

Для того щоб зробити потік фоновим, досить присвоїти логічне значення `true` властивості `IsBackground`. А логічне значення `false` вказує на те, що потік є пріоритетним.

Пріоритети потоків. У кожного потоку є свій пріоритет, який певною мірою визначає, наскільки часто потік отримує доступ до ЦП. Взагалі кажучи, низькопріоритетні потоки отримують доступ до ЦП рідше, ніж високопріоритетні. Таким чином, протягом заданого проміжку часу низькопріоритетному потоку буде доступно менше часу ЦП, ніж високопріоритетному.

Слід мати на увазі, що, крім пріоритету, на частоту доступу потоку до ЦП впливають і інші фактори. Так, якщо високопріоритетний потік очікує на доступ до деякого ресурсу, наприклад для введення з клавіатури, він блокується, а замість нього виконується фоновий потік. Крім того, конкретне планування задач на рівні операційної системи також впливає на час ЦП, що виділяється для потоку.

Коли породжений потік починає виконуватися, він отримує пріоритет, встановлюваний за замовчуванням. Пріоритет потоку можна змінити за допомогою властивості `Priority`, що є членом класу `Thread`.

```
public ThreadPriority Priority{ get; set; }
```

де `ThreadPriority` позначає перерахування:

- `ThreadPriority.Highest`
- `ThreadPriority.AboveNormal`
- `ThreadPriority.Normal`
- `ThreadPriority.BelowNormal`
- `ThreadPriority.Lowest`

За замовчуванням для потоку встановлюється значення пріоритету `ThreadPriority.Normal`.

Розглянемо приклад, в якому виконуються два потоки: один з більш високим пріоритетом. Обидва потоки створюються в якості екземплярів об'єктів класу `MyThread`. У методі `Run()` організовується цикл, в якому підраховується певне число повторень. Цикл завершується, коли число досягає величини 1000000000 або коли статична змінна `stop` отримує логічне значення `true`. Спочатку змінна `stop` отримує логічне значення `false`. У першому потоці, де проводиться підрахунок до 1000000000, встановлюється логічне значення `true` змінної `stop`. В силу цього другий потік закінчується на наступному своєму інтервалі часу. На кожному кроці циклу рядок у змінній `currentName` перевіряється на наявність імені виконуваного потоку. Якщо імена потоків не збігаються, це означає, що сталося перемикання задач, що виконуються. Всякий раз, коли відбувається перемикання задач, ім'я нового потоку відображається і присвоюється змінній `currentName`. Це дає можливість відстежити частоту доступу потоку до ЦП. Після закінчення обох потоків відображається число повторень циклу в кожному з них.

```
namespace ThreadPriority
{
	class MyThread
	{
		public int Count;
		public Thread Thrd;

		static bool stop=false;
		static string currentName;

		//Сконструювати, але не починати виконання нового потоку
		public MyThread(string name)
		{
```

```
                Count = 0;
                Thrd = new Thread(Run);
                Thrd.Name = name;
                currentName = name;
        }

        //Почати виконання нового потоку
        void Run()
        {
                Console.WriteLine("Thread "+Thrd.Name+" is beginning.");
                do
                {
                        Count++;
                        if (currentName != Thrd.Name)
                        {
                                currentName = Thrd.Name;
                                Console.WriteLine("In thread " + currentName);
                        }
                } while (stop == false && Count < 1e9);
                stop = true;

                Console.WriteLine("Thread " + Thrd.Name + " is completed.");
        }
}
class Program
{
        static void Main()
        {
                MyThread mt1=new MyThread("With High priority.");
                MyThread mt2=new MyThread("with Low priority.");

                //Встановити пріоритети для потоків
                        //Вище середнього
                mt1.Thrd.Priority = System.Threading.ThreadPriority.AboveNormal;
                        //Нижче середнього
                mt2.Thrd.Priority = System.Threading.ThreadPriority.BelowNormal;

                //Почати потоки
                mt1.Thrd.Start();
                mt2.Thrd.Start();

                mt1.Thrd.Join();
                mt2.Thrd.Join();

                Console.WriteLine();
                Console.WriteLine("Thread " + mt1.Thrd.Name + " counted to " + mt1.Count);
                Console.WriteLine("Thread " + mt2.Thrd.Name + " counted to " + mt2.Count);
                Console.ReadLine();
        }
}
}
```

Результат може бути таким:

```
Thread With High priority. is beginning.
In thread With High priority.
Thread with Low priority. is beginning.
In thread With High priority.
In thread with Low priority.
In thread With High priority.
In thread With High priority.
In thread With High priority.
    ...
In thread With High priority.
In thread With High priority.
In thread with Low priority.
Thread with Low priority. is completed.
```

```
Thread With High priority. is completed.

Thread With High priority. counted to 1000000000
Thread with Low priority. counted to 36299971
```

Виходячи з результату, високопріоритетний потік отримав близько 96% всього часу, який був виділений для виконання цієї програми.

Завдання для виконання.

1. Повторити задачі, розв’язані у тексті.
2. Створити багатопотокову програму, задати різні пріоритети для потоків. Підрахувати розподіл процесорного часу між потоками у %.
   Варіант 1) Потоків – 3 шт. (пріоритети: середній, вище середнього, найвищий).
   Варіант 2) Потоків – 5 шт. (пріоритети: найвищий, найнижчий, вище середнього, середній, нижче середнього).
   Варіант 3) Потоків – 4 шт. (пріоритети: найнижчий, вище середнього, нижче середнього, найвищий).
   Варіант 4) Потоків – 6 шт. (пріоритети: середній, нижче середнього, вище середнього, найвищий, найнижчий, середній).

3. Високий рівень: Реалізувати програму із попереднього завдання з використанням графічного інтерфейсу та можливістю динамічного задання кількості та пріоритетності потоків; відображення ходу обчислень та виведення результатів.
4. Оформити звіт.