**Паралелно програмиране**

**Упражнение 2 – Нишки**

1. **Стартиране и приспиване на нишки**

**Thread(ThreadStart)**

Създава инстанция. Подава се делегат\* с метод, който да се изпълни при стартиране.

**Sleep(<miliseconds>)**

“Приспива” текущата нишка за указания брой милисекунди. Методът е статичен и блокира текущото изпълняваната нишка. След изтичането на зададения интервал, тя продължава работата си.

**Suspend()**

Ако нишката работи, я преустановява временно. Ако вече е преустановена, не се случва нищо. За разлика от Sleep(), чрез който нишка преустановява себе си за някакъв фиксиран интервал от време, Suspend() преустановява нишка за неопределено време и тя остава в това състояние до извикването на Resume(), който подновява изпълнението й.

**Resume()**

Подновява нишка, която е била преустановена (suspended). Ако нишката работи, не прави нищо.

public void DoTask1()

{

for (int j = 0; j < 100; j++)

{

Console.WriteLine("DoTask1 job {0},", j);

Thread.Sleep(100);

}

}

public void DoTask2()

{

for (int j = 0; j < 100; j++)

{

Console.WriteLine("DoTask2 job {0},", j);

Thread.Sleep(100);

}

static void Main(string[] args )

{

Thread thread1 = new Thread(() => DoTask1());

thread1.Name = "ThreadOne";

Thread thread2 = new Thread(() => DoTask2());

thread2.Name = "ThreadTwo";

thread1.Start();

thread2.Start();

thread1.Join();

thread2.Join();

}

1. **Стартиране и изчакване на нишка**

**Join()**

Извикващата нишка изчаква, докато извиканата приключи. Може да се зададе и таймаут.

**Abort()**

Хвърля ThreadAbortException в извиканата нишка, с което обикновено прекратява нишката. При определени условия, Abort() може и да не прекрати нишката.

**Interrupt()**

Ако нишката е в състояние WaitSleepJoin , хвърля ThreadInterruptedException. Нишката може да прихване това изключение и да продължи изпълнението си. Ако тя не го прихване, CLR го прихваща и прекратява нишката. Ако нишката не е в състояние WaitSleepJoin , извикването на Interrupt() не прави нищо.

1. **Синхронизиращи конструкции**

Синхронизират се само отделни участъци от кода - тези, които са рискови. Критична секция наричаме участък от кода, до който не трябва да бъде допускан едновременен достъп. За гарантиране безопасен достъп до критична секция може да използваме ключовата дума lock или класа Monitor.

lock (obj)

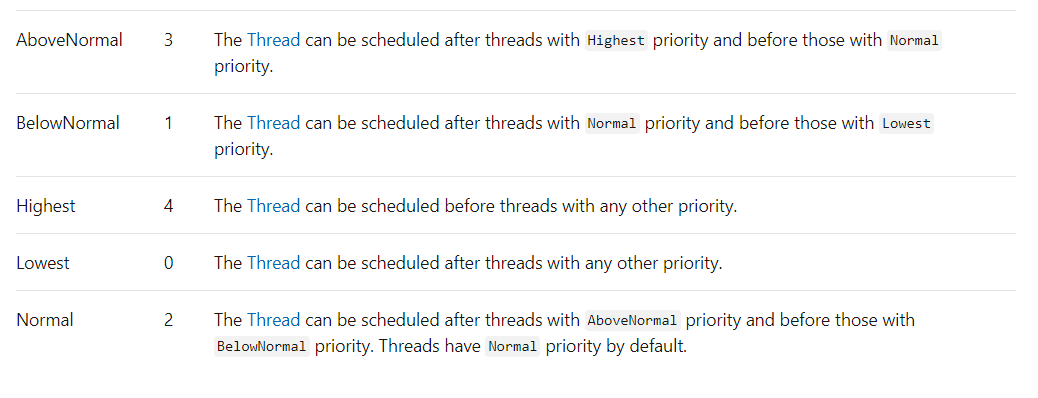
{

//code

}

1. **Приоритет на нишката**

public System.Threading.ThreadPriority Priority { get; set; }



\*Делегатите представляват .NET типове, които описват сигнатурата на даден метод (броя, типа и последователността на параметрите му) и връщания от него тип .Делегатите приличат на указателите към функции в C и C++ – съдържат силно-типизиран указател (референция) към метод.Те са структури от данни, които приемат като стойност методи, отговарящи на описаната от делегата сигнатура.Чрез тях се осъществяват "обратни извиквания" (callbacks).Могат да сочат както към статични методи, така и към методи на инстанция

**Задачи**

**Задача 1.** Имаме клас банкова сметка (Account), с член променлива mBalance -

текущият баланс по сметката и член метод Withdraw100(), който нямалява баланса

с 100. Проблемът настъпва, когато две нишки теглят едновременно пари от тази

банкова сметка, остатъкът в нея остава некоректен. Как може да се реши този проблем?

**Задача 2.** Да се създаде конзолно приложение на C#, което да сумира всeки от елементите на двумерен масив от тип double с число от същия тип. Масива да се инициализира със случайни числа. При обхождането на масива в сумиращия метод да се извежда съобщение в конзолата с името на нишката и стойностите на i и j само когато i и j се делят на 100.

Сумирането да се извършва в метод, който да се извиква от 2 нишки с различен приоритет. Да се измерят и сравнят резултатите от всяка нишка.