## 8 дәріс  Бір үдерістің әртүрлі тізбектерінің деректерді пайдалануы

### 8.1 Әртүрлі  тізбектердің жергілікті айнымалыларды пайдалануы

Алдыңғы дәрісте монитор экранының консольдік терезесіне «A» және «B» символдарын шығарған екі тізбектің жұмысын қарастырдық. Тізбектер бір-біріне тәуелсіз жұмыс істеді. C# тілінде әр объектіге деректердің өз стегі бөлінген  және жергілікті айнымалылар бөлек сақталғандықтан осылай болады.

Егер сандарды консольдік терезеге шығаратын әдісі бар қандай да бір клас үшін объект және жергілікті айнымалы құрса, онда әртүрлі тізбектердің жұмысы барысында осы кластың объектісінің жергілікті айнымалысын өзгертуге бола ма, жоқ па екендігін тексеруге мүкіндік бар.  Мұны біздің объектінің әдісін басты тізбекте және бір қосымша тізбекте қосып, ұйымдастаруға болады.

Бағдарламаның коды:

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Danie_1
    {
        class Pervaj_1
        {
            bool flag;
            public void Niti_1()
            {
                Console.WriteLine("Работает нить");
                if (!flag)
                {
                    flag = true;
                    for (int i = 0; i < 10; i++)
                        Console.Write(i + " ");
                    Console.WriteLine();
                }
            }
        }
        static void Main()
        {
            Pervaj_1 tt = new Pervaj_1();
            new Thread(tt.Niti_1).Start();
            tt.Niti_1();
            Console.ReadLine();
        }
    }
}
```

Результат работы программы:

Работает нить

Работает нить

0 1 2 3 4 5 6 7 8 9

или

Работает нить
0 1 2 3 4 5 6 7 8 9
Работает нить

Біздің бағдарламаның жұмысынан көргеніміздей, екі тізбекте **flag** жергілікті айнымалысымен жұмыс істейді. Сандардың тек бір реттілігі ғана шығарылады, ал басқасы блокталынады – тізбек **flag** айнымалысының мәнін «көреді».

Егер екі объект құрып, әр объекті үшін өз тізбегін іске қосса, онда әр объекті үшін (тиісінше әр тізбекке) өзінің **flag** жергілікті айнымалысы құрылатын болады және де әртүрлі тізбектердің жергілікті айнымалыны бірлесе пайдалануы тоқтатылады.

Бағдарлама коды:

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Danie_1
    {
        class Pervaj_1
        {
            bool flag;
            public void Niti_1()
            {
                Console.WriteLine("Работает нить");
                if (!flag)
                {
                    flag = true;
                    for (int i = 0; i < 10; i++)
                        Console.Write(i + " ");
                    Console.WriteLine();
                }
            }
        }
        static void Main()
        {
            Pervaj_1 tt = new Pervaj_1();
            new Thread(tt.Niti_1).Start();
            Pervaj_1 vv = new Pervaj_1();
            new Thread(vv.Niti_1).Start();
            //tt.Niti_1();
            Console.ReadLine();
        }
    }
}
```

Работает нить
Работает нить

0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
или

Работает нить
0 1 2 Работает нить
0 1 2 3 4 5 6 7 8 9
3 4 5 6 7 8 9
и т.д.

Сонымен, егер тізбектер бір объектіге тиесілі әдістерді немесе деректерді пайдаланса, онда әртүрлі тізбектердің деректерді бірлесе пайдалануы туралы айтуға болады.

Әртүрлі тізбектердің деректер мен әдістерді пайдалануы туралы теорияда, әдетте, тізбектердің «статикалық» элементтермен жұмыс істеуі қарастырылады, яғни **static** қатынау спецификаторы бар элементтермен тізбектің жұмыс істеуі сұрақтары қарастырылады. Біздің бағдарламада бір статикалық айнымалыны және бір әдісті жариялап, және де оларды әртүрлі тізбектерде қолданып көреміз.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
 class Danie_1
 {
  static int k = 0;
  static void cikl()
  {
   k++;
   Console.WriteLine("Работает поток " + k);
   int n1, n2;
   n1 = k * 10; n2 = (k + 1) * 10;
   for (int i = n1; i < n2; i++)
        Console.Write(i + " ");
   Console.WriteLine();
  }

  static void Main()
  {
   new Thread(cikl).Start();
   new Thread(cikl).Start();
   new Thread(cikl).Start();
   new Thread(cikl).Start();
   cikl();
   Console.ReadLine();
  }
 }
}
```

Работа программы:

Работает поток 1
Работает поток 3
50 Работает поток 2
50 51 52 53 54 55 56 57 58 59
50 51 52 53 54 55 56 57 58 59
51 52 53 54 55 56 57 58 59
Работает поток 4
Работает поток 5
50 51 52 53 54 55 56 57 58 59
50 51 52 53 54 55 56 57 58 59

Іске қосылған тізбектердің барлығы бір статикалық айнымалыны пайдаланады және бір әдіспен жұмыс істейді. Және де тізбектер жұмысының қауіпсіздігінің термині тиімділікті (рентабельдідік) пайдаланса, онда олардың барлығы тиімсіз (рентабельді емес) болады.

Тізбектер жұмысының қауіпсіздігін қамтамасыз етумен ортақ деректерді пайдаланудың нұсқаларының бірі оларда ағымдағы тізбектің жұмысы аяқталғанға дейін барлық қалған тізбектердің жұмысын блоктауға (іс жүзінде бүкіл бағдарламаны блоктау) мүмкіндік беретін **lock** операторын пайдалану болып табылады.

**lock** операторы C# тілінің арнайы блоктаушы конструкцияларға жатады.

C# тілінде бағдарламалау бойынша оқулықтардың әртүрлі авторлары **lock** операторын қалай анықтайтынын қарастырайық.

**lock** операторы **try/finally** блогымен **Monitir** класының **Enter** және **Exit** әдістерін шақырту үшін синтаксистік қысқарту болып табылады [Албахари б.743].

**lock** негізгі сөзі код блогын уақыттың әр мезетінде оны тек бір тізбек қана пайдалана алатындай етіп блоктауға мүмкіндік береді [Троелсен б.312].

Павловская Т.А. болса, **lock**-ты келесі жазба форматы бар оператор ретінде анықтайды

lock (өрнек)

{ операторлардың блогы}

Өрнек блоктауды қажет ететін объектіні анықтайды. Қарапайым әдістер үшін өрнек ретінде **this** негізгі сөзі, ал статикалық әдістер үшін – **typeof(**клас) пайдаланылады. Операторлардың блогы блоктауды қажет ететін кодтың қиын (күрделі, сыни, критический) секциясын береді [Павловская б. 242].

Бұл анықтамалардың барлығында **lock** көптеген тізбектер пайдалануы үшін блокталатын кодтың қандай да бір блогын енгізеді деген бірыңғай анықтама бар. Әдістемелік көзқарастан дәрістерімізде ұстанатын Т.А. Павловскаяның анықтамасы ең жақсы және түсінікті болып табылады, дәрістерімізде әрі қарай соны қолданамыз.

Сонымен, **lock** дегеніміз ол код секциясын уақыттың әр мезетінде оны тек бір ғана тізбек пайдаланытындай етіп блоктауға мүмкіндік беретін оператор.

Бағдарламаның коды:

```
using System;
```

```
using System.Threading;

namespace ConsoleApplication1
{
 class Danie_1
 {
  static int k = 0;
  static object t = typeof(object);
  static void cikl()
  {
   Console.WriteLine("Работает нить ---- " + k);
   lock (t)
   {
    k++;
    Console.WriteLine("Работает нить № " + k);
    int n1, n2;
    n1 = k * 10; n2 = (k + 1) * 10;
    for (int i = n1; i < n2; i++)
         Console.Write(i + " ");
    Console.WriteLine();
   }
  }

  static void Main()
  {
   new Thread(cikl).Start();
   new Thread(cikl).Start();
   new Thread(cikl).Start();
   new Thread(cikl).Start();
   cikl();
   Console.ReadLine();
  }
 }
}
```

Работа программы:
Работает нить ---- 0
Работает нить ---- 0
Работает нить ---- 0
Работает нить ---- 0
Работает нить ---- 0
Работает нить № 1
10 11 12 13 14 15 16 17 18 19
Работает нить № 2
20 21 22 23 24 25 26 27 28 29
Работает нить № 3
30 31 32 33 34 35 36 37 38 39
Работает нить № 4
40 41 42 43 44 45 46 47 48 49
Работает нить № 5
50 51 52 53 54 55 56 57 58 59

Біздің бағдарламада бір мезгілде барлық тізбектер іске қосылады, бірақ кезекті тізбектің орындалуы **lock** операторының көмегімен бағдарламаның қалған тізбектерінің орындалуын блоктауға әкеп соғады.

Бұл сияқты міндеттер тізбек жұмысы кезінде бағдарламаның басқа тізбектерімен де пайдаланылатын кейбір айнымалылардың мәндері тізбек жұмысының соңына дейін өзгермеу керек болғанда пайда болады. Мысалы, банк клиенттерінің шоттарымен операциялар атқарғанда.

Жұмысын аяқтаған тізбек қайтадан басынан басталмайды.

## 8.2 Тізбектерге деректерді жіберу

Тізбек объектілерін құру кезінде пайдаланылатын бізбен қарастырылған **ThreadStart** делегатында деректерді тізбектерге жіберу үшін қажетті формальді параметрлер жоқ – делегат іске қосылатын әдістің атауын ғана анықтай алады. Бірақ, **C#** тілінде тізбек конструкторы асыра жүктелген және тек объектіні құруға ғана қабілетті емес, сондай-ақ **ParameterizedThreadStart** деген қандай да бір параметрді жіберетін **.NET Framework** платформасының басқа да делегатын пайдалануға рұқсат етеді. Осы делегаттың келесі жазба форматы бар:

public delegate void ParameterizedThreadStart(object obj);

**ParameterizedThreadStart** әдісі **object** типіндегі тек бір параметрді қабылдауға қабілетті – **C#** тіліндегі кез келген объект **object**-ден туынды болып саналады.

Пайдаланылатын делегаттың (оқу мақсаттарында) атауын анық көрсететін және ол үшін тізбек қосылатын әдіске қандай да бір аргументті жіберетін оқу мысалын қарастырайық.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
 class Nit_Par
 {
  static void cikl(object ob)
  {
   int k = (int)ob;
   Console.WriteLine("Работает нить ---- " + k);
   k++;
   Console.WriteLine("Работает нить № " + k);
   for (int i = 0; i <= k; i++)
        Console.Write(i + " ");
   Console.WriteLine();
  }

  static void Main()
  {
      Thread[] name = new Thread[5];
      for (int j = 0; j < 5; j++)
      {
```

```
        name[j] = new Thread(new ParameterizedThreadStart(cikl));
        name[j].Start(j);
       }
      Console.ReadLine();
   }
  }
}
```

Работа программы:
Работает нить ---- 0
Работает нить № 1
0 1
Работает нить ---- 1
Работает нить ---- 2
Работает нить ---- 3
Работает нить № 4
0 1 2 Работает нить № 3
0 1 2 3
3 4
Работает нить ---- 4
Работает нить № 5
0 1 2 3 4 5
Работает нить № 2
0 1 2

Келтірілген мысалда тізбектік әдісті іске қосқан уақытта оған **object** типтіі бір параметрді жіберуге мүмкіндік беретін ParameterizedThreadStart(object obj) делегатының көмегімен **Thread** класының объектілерінің жиыны құрылады.

Тізбекте іске қосылатын әдіс алынған параметрдің бүтін санға анық түрленуін орындауға тиіс.

Біз атап кеткендей, **C#** тілінде тізбек конструкторы асыра жүктелген және бастапқыда берілгендей, яғни өзгеріссіз **.NET Framework** платформасының бірнеше делегатын пайдалануға жол береді. Сондықтанда, әдетте бағдарламада пайдаланылатын делегаттың атауы жазылмайды, мысалы, біздің бағдарламада **Main** әдісі келесідей жазылады:

```
static void Main()
  {
      Thread[] name = new Thread[5];
      for (int j = 0; j < 5; j++)
      {
          name[j] = new Thread(cikl);
          name[j].Start(j);
      }

     Console.ReadLine();
  }
```

Тізбек жұмысының реттілігін мысалы, **lock** операторының көмегімен жүргізуге болады.

Тізбектерге деректерді жіберудің басқа тәсілі, объектінің белгілі бір данасының объектісіз және оның өрістерінсіз пайдаланылатын статикалық әдісін емес, объектінің өрістерімен жұмыс істеуге мүмкіндік беретін әдісін тізбекте іске қосу болып табылады.

Объектінің таңдалған данасының қасиеттері тізбектердің тәртібін анықтайтын болады.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
 class Program
 {
  public static int kol=0;
  class T
  {
   private double data;
   private int c, n;
   public Thread thread1;

   public T()
   {
    thread1 = new Thread(run);
    thread1.Start(kol);
   }
   public void run(object ob)
   {
    lock (this)
    {
     kol++; n = kol;
     Console.WriteLine("Нить {0} стартовала ",kol);
     Console.WriteLine("Введите число");
     c = int.Parse(Console.ReadLine());
     if (kol == 4) data = c*40;
     if (kol < 4)
     {
      switch (kol)
      {
       case 1: data = Math.Cos(c); break;
       case 2: data = Math.Exp(c); break;
       case 3: data = c + 30; break;
      }
      T thr2 = new T();
      thr2.thread1.Join();
     }
     Console.WriteLine("Нить {0} завершена! Значение равно:
                                                    {1}",n,data);
    }
   }
  }
  static void Main()
  {
```

```
    T thr1 = new T();
    thr1.thread1.Join();
    Console.ReadKey();
   }
  }
}
```

Работа программы:

Нить 1 стартовала
Введите число
1
Нить 2 стартовала
Введите число
2
Нить 3 стартовала
Введите число
3
Нить 4 стартовала
Введите число
4
Нить 4 завершена! Значение равно: 160
Нить 3 завершена! Значение равно: 33
Нить 2 завершена! Значение равно: 7,38905609893065
Нить 1 завершена! Значение равно: 0,54030230586814

Біздің мысалда 4 тізбек іске қосылады, әрі олардың әрқайсысы **T** класының **run** әдісін пайдаланады. Параметр ретінде мәні қосылған тізбектің нөміріне сәйкес келетін **kol** айнымалысын алады. **Run** әдісі **T** класының әдісі болып табылады, сондықтанда ол осы кластың объектілерінің өрістерімен жұмыс істей алады, мысалы, әр тізбек үшін жеке берілген **c** өрісінің мәнін пайдаланып, қандай да бір өрнекті есептейді. Біздің бағдарламада әр тізбекте нәтижесі **data** нақты типті айнымалыға берілетін өз өрнегі қолданылған.

Бағдарламада кезекті тізбекті қосудың рекурсивтік тәсілі пайдаланылған – іске қосылатын тізбектердің санын шектеу if шартының көмегімен жүргізіледі (kol < 4). **kol** айнымалысы 0-ден бастап өзгереді. Әр қосылған тізбекте өзінің **T** объектісі құрылады thr2 = new T().

Құрылған объектінің ішінде шақыртқан объектіні оның аяқталуына дейін блоктайтын Join() (thr2.thread1.Join();) әдісі іске қосылады.

Көп тізбекті қосымшаларды пайдаланудың тиімділігі туралы - Интернеттен шолу (тізбекті кейді ағындар деп атайды).

### *Ағындарды қашан пайдаланған жөн*

Егерде орындалатын тапсырма көп уақыт алатын болса, онда көп ағындылықтың маңыздылығы болады, өйткені басқа компьютерден жауап тосу қажет, (қосымшаның сервері, деректер базасының немесе клиенттің сервері).

Көп ағынды типтік қосымша ұзақ есептеулерді фондық режімде атқарады. Басты ағын орындауды жалғастырады, сол уақытта жұмыс ағыны фондық тапсырманы орындайды.

Windows Forms қосымшаларында басты ағын ұзақ есептеулерді жүргізіп жатқанда, ол пернетақта мен тінтуірдің хабарламаларын өңдей алмайды және қосымша еш белгі бермейді. Осы себеп бойынша *"Жұмыс істеп тұрмын... Мархабат, тосыңыз"* деген жазбамен басты ағын пайдаланушыға модалдық диалогты көрсетсе де, жұмыс ағынында көп уақыт алатын тапсырмаларды іске қосу қажет, өйткені бағдарлама ағымдағы операция аяқталмаса, келесі операцияға ауыса алмайды. Осындай шешім, пайдаланушыны үдерісті үзуге итермелейтіндей, операциялық жүйе қосымшаны "Жауап бермейтін" деп белгілемейтіндігіне кепілдік береді. Осы жағдайда да модалдық диалог «Болдырмау» («Отмена») батырмасын ұсынуы мүмкін, өйткені осы форма тапсырма фондық ағында орындалып жатқанда, хабардарды алып жатады.

Бір ағынды web-сервер тек қана нашар емес, ол жай ғана мүмкін емес! Күйлерді (stateless) сақтамайтын қосымшалар серверінің жағдайында, көп ағындылық әдетте қарапайым түрде жүзеге асырылатыны қуантады. Статикалық айнымалылардағы деректерге қатынауды синхрондауда қиындықтар туындауы мүмкін.

### *Ағындар қашан қажет емес*

Көп ағындылықтың артықшылықтармен қатар, кемшіліктері де бар. Олардың ең бастысы – бағдарламалардың күрделілігін біршама арттыру. Күрделілікті қосымша ағындар емес, олардың өзара әрекеттесуін ұйымдастыру қажеттілігі арттырады. Әзірлеу циклының ұзақтығы, сондай-ақ бағдарламада кездейсоқ (анда-санда) пайда болатын және қиын анықталатын қателердің саны осы өзара әрекеттесу қаншалықты ниетті болғандығына байланысты болады. Сонымен, ағындардың өзара әрекеттестігінің дизайнын қарапайым етіп ұстау керек, немесе сіз кодты қайта жазуға немесе реттеуге деген табиғилыққа қарсы қабілеттілігіңіз болмаса, көп ағындылықты мүлдем пайдаланбау қажет.