Лекция 2. Көпағындық программалау негіздері. Ағынды құру және іске қосу. Join, Sleep әдістері.

Дәрістің мақсаты: Студенттерде көпағындық программалау негіздерін түсіну дағдыларын қалыптастыру.

Дәрісті меңгеру нәтижесінде студенттер келесі қабілеттерге ие болады:

- Көпағындық программалаудың қызметін түсіну;
- Ағынды құруға және іске қосуға арналған командаларды білетінін көрсету;
- Join, Sleep әдістерінің синтаксисі мен қызметін түсіну.

Көп ағынды бағдарлама қатар орындалатын екі және одан да көп бөліктен тұрады. Мұндай бағдарламаның әрбір бөлігі ағын деп аталады және пәрмендерді орындаудың жеке жолын анықтайды. Осылайша, көп ағынды өңдеу көп міндеттіліктің ерекше нысаны болып табылады.

Көп ағынды бағдарламалау осы мақсат үшін C# тілінің өзінде қарастырылған бірқатар қаражатқа, сондай-ақ .NET Framework ортасында анықталған сыныптарға сүйенеді. C # жүйесінде орнатылған көп ағынды өңдеуді қолдаудың арқасында басқа бағдарламалау тілдерінде көп ағынды өңдеуді ұйымдастыруға байланысты көптеген қиындықтар барынша азайтылады немесе мүлдем жойылады.

Одан әрі белгілі болғандай, C# 4.0 нұсқасын шығару арқылы көп ағынды өңдеуде қолдау нақты ұйымдастырылған және қарапайым түсіну үшін. .NET Framework ортасында көп ағынды қосымшаларға қатысы бар екі маңызды қосымша пайда болды. Олардың біріншісі - TPL (Task Parallel Library - Тапсырмаларды ашу кітапханасы), ал екіншісі - PLINQ (Parallel LINQ - Интеграцияланған сұраулардың параллель тілі). Екі толықтыру қатар бағдарламалауды қолдайды және көп процессорлы деректерді өңдеуге қатысты (көп ядролы) компьютерлермен. Бұдан басқа, кітапхана TPL жеңілдетеді көп ағынды қосымшалар жасау және оларды басқару.

Көп міндеттіліктің екі түрін ажыратады: процестер негізінде және ағындар негізінде. Осыған байланысты олардың арасындағы айырмашылықтарды түсіну маңызды. Процесс іс жүзінде орындалатын бағдарламаны білдіреді. Сондықтан процестер негізінде көп міндеттілік - бұл компьютерде екі және одан да көп бағдарлама қатар орындалатын құрал. Осылайша, процестер негізінде көп міндеттілік бір мезгілде мәтіндік редактордың бағдарламаларын, электрондық кестелерді және Интернеттегі мазмұнды қарап шығу. Процестер негізінде көп міндеттілікті ұйымдастыру кезінде бағдарлама тапсырмаларды жоспарлаушы үйлестіре алатын кодтың ең аз бірлігі болып табылады.

Ағын орындалатын кодтың үйлестірілетін бірлігін білдіреді. Бұл термин өзінің шығу тегі ретінде "орындау ағыны" ұғымына міндетті. Ұйымдастыру кезінде ағындар негізінде көп міндеттілік әрбір процесте кем дегенде бір ағын болуы тиіс, бірақ олар көп болуы мүмкін. Бұл бір бағдарламада бір мезгілде екі және одан да көп міндеттер шешілуі мүмкін. Мысалы, мәтін екі іс-әрекет екі бөлек ағымда орындалған жағдайда оны басып шығарумен бір мезгілде мәтін редакторында пішімделуі мүмкін.

Көп ағынды өңдеудің басты артықшылығы мынада: мүмкіндіктің арқасында өте тиімді жұмыс істейтін бағдарламаларды жазу орындау барысында сөзсіз туындайтын тоқтап тұру уақытын пайдалану тиімді бағдарламалардың көпшілігі. Белгілі болғандай, енгізу-шығару құрылғыларының көпшілігі, желілік порттарға қосылған құрылғылар, дискілердегі жинақтағыштар немесе орталық процессорға (ЦП) қарағанда әлдеқайда баяу жұмыс істейді. Сондықтан бағдарламаға өз уақытының көп бөлігін құрылғыға деректердің жіберілуін күтуге тура келеді ақпаратты енгізу-шығару немесе одан қабылдау. Ал көп ағынды

өңдеудің арқасында мәжбүрлеу кезінде қандай да бір басқа міндетті шешуі мүмкін тоқтап тұру. Мысалы, бағдарламаның бір бөлігі файлды арқылы жібереді Интернетпен қосылу, оның басқа бөлігі мәтіндік ақпаратты оқуды орындай алады, пернетақтадан енгізілетін, ал үшіншісі - кезекті пернетақтаны буферизациялауды жіберілетін деректер блогы.

Ағын бірнеше күйдің бірінде болуы мүмкін. Жалпы алғанда, ағын орындалатын болуға; уақытын және ресурстарын алғаннан кейін орындауға дайын ОП; тоқтатылған, т е. уақытша орындалмайтын; бұдан әрі жаңартылған; өзінің орындалуы үшін ресурстарды күтуге бұғатталған; сондай-ақ аяқталған, оның орындалуы аяқталған және қайта басталуы мүмкін болмаған жағдайда.

.NET Framework ортасында ағынның екі түрі анықталған: басым және фондық. Әдепкі құрылатын ағын автоматты түрде басымдыққа ие болады, бірақ оны фондық етуге болады. Басым ағындардың бірден-бір айырмашылығы фондық ағынның автоматты түрде аяқталатындығында, егер оның функцияларында барлық басым ағындар тоқтатылды.

Ағындар негізінде көп міндеттілікті ұйымдастыруға байланысты қажеттілік туындайды үйлестіру деп аталатын және үйлестіруге мүмкіндік беретін ерекше режимде ағындарды нақты түрде орындау.

Ағындарды құру және іске қосу

Көп ағынды өңдеу жүйесі орындау ағынын инкапсулациялайтын Thread класына негізделеді. Thread класы герметикалық болып табылады, яғни ол мұрагерлік ете алмайды. Thread класында ағынды басқаруға арналған бірқатар әдістер мен қасиеттер анықталған.

Ағынды жасау үшін Thread түріндегі нысанның, яғни System.Threading аттар кеңістігінде анықталған сыныптың данасын алу жеткілікті. Төменде Thread класының құрастырушысының қарапайым пішіні келтірілген:

public Thread (ThreadStart іске қосу)

мұнда *іске қосу* - ағынды орындауды бастау мақсатында шақырылатын әдістің аты, ал ThreadStart - көрсетілгендей, NET Framework ортасында анықталған делегат төменде.

public delegate void ThreadStart()

Демек, ағынға кіру нүктесі ретінде көрсетілетін әдістің қайтарылатын void түрі болуы және ешқандай дәлелдерді қабылдамауы тиіс. Жаңадан құрылған жаңа ағын орындалмайынша басталмайды
Thread сыныбында анықталатын оның Start() әдісі туындады. Екі пішін бар Start() әдісін жариялау. Төменде олардың біреуі келтірілген.

public void Start()

Бір күні басталғанда, ағын іске қосу көрсетілген әдістен қайтарылғанға дейін орындалады. Осылайша, одан қайтарылғаннан кейін бұл әдіс ағыны автоматты түрде тоқтатылады. Егер басталған ағын үшін Start() әдісін шақыруға әрекет жасасаңыз, бұл ерекшелік туындатады ThreadStateException.

Ағынның қашан аяқталатынын білу көбінесе пайдалы болып келеді. Осы мақсатқа арналған бағдарламалардың алдыңғы мысалдарында Count айнымалысының мәні қадағаланды.

Бірақ бұл жақсы емес және жалпылауға жарамды шешім емес. Рас, Thread класында ағынның аяқталу сәтін анықтау үшін басқа екі құрал бар. Осы мақсатта, ең алдымен, мынадай түрде анықталатын, тек оқу үшін қолжетімді IsAlive сипатын сұрауға болады.

public bool IsAlive { get; }

Аяқталу сәтін қадағалаудың тағы бір тәсілі Join() әдісін шақырудан тұрады. Төменде оның қарапайым пішіні келтірілген.

public void Join()

Join() әдісі ол шақырылған ағын аяқталғанын күтеді. Оның есімі шақырушы ағын шақырылған әдіске қосылғанға дейін күту қағидатын көрсетеді. Егер бұл ағын басталмаса, онда ThreadStateException алып тастау туындайды. Join() әдісінің басқа пішіндерінде күтілетін ең ұзақ уақыт кезеңін көрсетуге болады көрсетілген ағынды аяқтау.

Төменде келтірілген бағдарламаның мысалында Join() әдісі негізгі ағынның соңғысымен аяқталуы үшін пайдаланылады.

Мысал 1. Join() әдісін пайдалану.

```
using System;
using System.Threading;
class MyThread {
public int Count;
public Thread Thrd;  // ағындық типтегі обьект
public MyThread(string name) {
Count = 0;
Thrd = new Thread(this.Run); // ағынды құру
Thrd.Name = name;
Thrd.Start(); // ағынның атқарылуын бастау
}
// ағынға кіру нүктесі
void Run() {
Console.WriteLine(Thrd.Name + " басталды.");
do {
// ағынды уақытша тоқтату
Thread.Sleep(500);
Console.WriteLine(Thrd.Name + " ағынында, Count = " + Count);
Count++;
} While(Count < 10);
Console.WriteLine(Thrd.Name + " аяқталды.");
}
}
class JoinThreads {
static void Main() {
Console.WriteLine("Негізгі ағын басталды.");
// Үш ағын құру
MyThread mt1 = new MyThread("#1 буын");
MyThread mt2 = new MyThread("#2 буын");
MyThread mt3 = new MyThread("#3 буын");
// Join() әдісінің көмегімен ағындар аяқталғанша күтуді ұйымдастыру
mt1.Thrd.Join();
Console.WriteLine("#1 буын қосылды.");
mt2.Thrd.Join();
Console.WriteLine("#2 буын қосылды.");
mt3.Thrd.Join();
Console.WriteLine("#3 буын қосылды.");
Console.WriteLine("Негізгі ағын аяқталды.");
}
}
```