Дәріс 7. Task класы. Тапсырманы құру. Тапсырманың идентификаторы. Күту әдістері.

Дәрістің мақсаты: Студенттерде параллель орындалатын тапсырмалар жұмысын басқару туралы түсінік қалыптастыру.

Дәрісті меңгеру нәтижесінде студенттер келесі қабілеттерге ие болады:

- Тапсырманы құру және іске қосу синтаксисін түсіну;
- Тапсырманың идентификаторын пайдалану;
- Күту әдістерін пайдалану ерекшеліктерін түсіну.

Task класы

Task класы негізге алынды. Орындаудың қарапайым бірлігі Thread емес, Task класының құралдарымен жинақталады. Task класы Thread класынан асинхронды операцияны білдіретін абстракция болуымен ерекшеленеді.

Ал Thread сыныбында орындау ағыны инкапсулданады. Әрине, жүйелік деңгейде ағын бұрынғысынша операциялық жүйе құралдарымен жоспарлауға болатын қарапайым орындау бірлігі болып қала береді. Бірақ Task сыныбы объектісі данасының және орындау ағынының сәйкестігі міндетті түрде өзара-бір мәнді болып табылмайды.

Бұдан басқа, тапсырмаларды орындауды ағындар пулымен жұмыс істейтін тапсырмаларды жоспарлаушы басқарады. Бұл, мысалы, бірнеше тапсырмалар бір ағымды бөлісе алады дегенді білдіреді. Task сыныбы (және қалған TPL кітапханасы) System.Threading аттарының кеңістігінде анықталған.

Тапсырмаларды құру

Task сыныбының нысаны түрінде жаңа тапсырманы жасау және оны орындауды әр түрлі тәсілдермен бастауға болады. Алдымен Task түріндегі нысанды құрастырушының көмегімен жасаймыз және оны Start() әдісін шақырып іске қосамыз. Осы мақсатта Task класында бірнеше құрастырушы анықталған. Төменде біз пайдаланатын құрастырушы келтірілген:

public Task (Action әрекет)

Бұл жерде әрекет тапсырманы білдіретін кодқа кіру нүктесін көрсетеді, ал Action - System аттар кеңістігінде анықталған делегат. Біз пайдаланатын Action делегатының формасы мынадай.

public delegate void Action()

Осылайша, кіру нүктесі ешқандай параметрлерді қабылдамайтын және ешқандай мәндерді қайтармайтын әдіс болуы тиіс. Тапсырма жасалғаннан кейін оны Start() әдісін шақыру арқылы іске қосуға болады. Төменде оның пішіндерінің бірі келтірілген.

public void Start()

Start() әдісін шақырғаннан кейін тапсырмаларды жоспарлаушы тапсырманы орындауды жоспарлайды.

Төменде келтірілген бағдарламада жоғарыда айтылғандардың барлығы іс жүзінде көрсетіледі. Бұл бағдарламада жеке тапсырма MyTask() әдісі негізінде жасалады. Main ()

әдісі орындала бастағаннан кейін іс жүзінде тапсырма жасалады және орындауға іске қосылады. MyTask() және Main() екі әдісі қатар орындалады.

Тапсырма идентификаторын қолдану

Thread класына қарағанда; Task сыныбында тапсырма атауын сақтауға арналған Name сипаты жоқ. Бірақ оның орнына онда тапсырмаларды тануға болатын тапсырма идентификаторын сақтауға арналған Id сипаты бар. Id сипаты тек оқуға арналған және int түріне жатады. Ол мынадай түрде жарияланады.

public int Id { get; }

Әрбір тапсырма жасалғанда идентификаторды алады. Идентификаторлар мәндері бірегей, бірақ реттелмеген. Сондықтан бір тапсырма идентификаторы басқасының алдында көрінуі мүмкін, бірақ ол аз мәнге ие болмауы мүмкін.

Қазір орындалатын тапсырманың идентификаторын CurrentId сипаты арқылы анықтауға болады. Бұл сипат тек оқуға арналған, static түріне жатады және келесі түрде жарияланады.

public static Nullable<int> CurrentID { get; }

Егер шақырушы код тапсырма болмаса, ол орындалатын тапсырманы немесе бос мәнді қайтарады.

Төменде келтірілген бағдарламаның мысалында екі тапсырма жасалады және олардың қайсысы орындалатыны көрсетіледі.

Күту әдістерін қолдану

Жоғарыда келтірілген мысалдарда негізгі орындау ағыны, ал мәні бойынша Main() әдісі аяқталды, себебі мұндай нәтиже Thread.Sleep() әдісінің шақыруларына кепілдік берді. Бірақ мұндай тәсілді қанағаттанарлық деп санауға болмайды.

Task сыныбында арнайы ұсынылатын күту әдістерін қолдана отырып, тапсырмалардың аяқталуын күтуді неғұрлым жетілдірілген тәсілмен ұйымдастыруға болады. Олардың ішіндегі ең қарапайым Wait() әдісі болып есептеледі, шақырылатын міндет аяқталғанға дейін шақырушы ағынды орындауды тоқтата тұрады. Төменде осы әдісті хабарландырудың қарапайым пішіні келтірілген.

public void Wait()

Бұл әдісті орындау кезінде екі ерекшелік генерациялануы мүмкін. Олардың біріншісі ObjectDisposedException алып тастау болып табылады. Ол егер міндет Dispose() әдісін шақыру арқылы босатылған жағдайда жасалады. Ал екінші ерекшелік, AggregateException, егер міндет ерекшелікті өзі туындатса немесе алып тасталса генерацияланады. Әдетте, дәл осы ерекшелік бақыланады және өңделеді. Міндет бір ғана ерекшелікті жасай алатындықтан, мысалы, егер оның туындаған міндеттері болса, барлық осындай ерекшеліктер AggregateException түрін біртұтас алып тастауға жиналады. Шын мәнінде не болғанын анықтау үшін осы жиынтық ерекшелікке байланысты ішкі ерекшеліктерді талдау жеткілікті. Ал сол уақытқа дейін бұдан әрі келтірілген мысалдарда міндеттермен туындайтын кез келген ерекшеліктер орындау кезінде өңделетін болады.

Төменде Wait() әдісін іс жүзінде қолдануды көрсету мақсатында өзгертілген алдыңғы бағдарламаның нұсқасы келтірілген. Бұл әдіс екі tsk және tsk2 тапсырмалары аяқталғанға дейін оны орындауды тоқтата тұру үшін Main() әдісінің ішінде пайдаланылады.

```
using System;
using System.Threading;
using System.Threading.Tasks;
class DemoTask {
// Тапсырма ретінде атқарылатын әдіс.
static void MyTask() {
Console.WriteLine("MyTask() №" + Task.CurrentId + " іске қосылды");
for(int count = 0; count < 10; count++) {
Thread.Sleep(500);
Console.WriteLine("MyTask() #" + Task.CurrentId + "әдісінде, санауыш мәні " + count );
}
Console.WriteLine("MyTask №" + Task.CurrentId + " аяқталды");
}
static void Main() {
Console.WriteLine("Негізгі ағын іске қосылды.");
// Екі тапсырма объектісін құру
Task tsk = new Task(MyTask);
Task tsk2 = new Task(MyTask);
// Тапсырмаларды атқаруға жіберу
tsk.Start();
tsk2.Start();
Console.WriteLine("tsk тапсырмасының идентификаторы: " + tsk.Id);
Console.WriteLine("tsk2 тапсырмасының идентификаторы: " + tsk2.Id);
// tsk және tsk2 тапсырмалары аяқталғанға дейін
// Main() әдісінің атқарылуын уақытша тоқтату
tsk.Wait();
tsk2.Wait();
Console.WriteLine("Негізгі ағын аяқталды.");
}
}
}
```