Дәріс 9. Тапсырмадан мән қайтару. Ағындармен және тапсырмалармен жұмыста аластамаларды өңдеу.

Дәрістің мақсаты: Студенттерде ағындармен немесе тапсырмалармен жұмыс барысында туындайтын аластамалық жағдайлардың ерекшеліктері туралы түсінік қалыптастыру.

Дәрісті меңгеру нәтижесінде студенттер келесі қабілеттерге ие болады:

- Тапсырмадан мән қайтару тәсілін түсіну;
- Ағынның және/немесе тапсырманың жұмысындағы аластамалық жағдайды анықтау;
- Аластамалық жағдайдың ағындар/тапсырмалар жиынының жұмысына әсерін басқару.

Тапсырмадан мәнді қайтару

Тапсырма мәнді қайтаруы мүмкін. Бұл екі себеппен өте ыңғайлы. Біріншіден, бұл тапсырманың көмегімен кейбір нәтижені есептеуге болатынын білдіреді. Осыған ұқсас түрде қатар есептеулер қолданылады. Екіншіден, шақырушы процесс нәтиже шыққанша бұғатталады. Бұл нәтижені күтуді ұйымдастыру үшін ешқандай ерекше үндестіру қажет емес дегенді білдіреді.

Нәтижені тапсырмадан қайтару үшін Task<TResult> Tack сыныбының жалпыланған пішінін пайдалана отырып, осы тапсырманы жасау жеткілікті. Төменде Task сыныбының осы пішінінің екі құрастырушысы келтірілген:

public Task (Func < TResult > функция)

public Task (Func < Object, TResult > функция, Object күй)

функция орындалатын делегатты білдіреді. Оның Action емес, Func түрі болуы керектігіне назар аударыңыз. Func түрі тапсырма нәтижені қайтарған жағдайларда пайдаланылады. Бірінші құрастырушыда аргументсіз тапсырма жасалады, ал екінші құрастырушыда - күй ретінде берілетін Object түріндегі аргументті қабылдайтын тапсырма. Осы сыныптың басқа құрастырушылары да бар.

Күтілгендей, StartNew сыныбының жалпылама нысанында қол жетімді және нәтижені тапсырмадан қайтаруды қолдайтын TaskFactory<TResult>() әдісінің басқа да нұсқалары бар. Төменде осы әдістің жаңа қарастырылған Task сыныбының құрастырушыларымен қатар қолданылатын нұсқалары келтірілген.

public Task<TResult> StartNew(Func<TResult> функция)

public Task < TResult > StartNew (Func < Object, TResult > функция, Object күй)

Кез келген жағдайда тапсырма қайтаратын мән Task сыныбындағы Result сипатынан алынады, ол келесі түрде анықталады.

public TResult Result { get; internal set; }

set аксессоры осы сипат үшін ішкі болып табылады, сондықтан ол тек оқу үшін ғана сыртқы кодта қол жетімді болады. Демек, нәтиже алу тапсырмасы шақырушы кодты нәтиже есептелгенге дейін бұғаттайды.

Тапсырманы болдырмау және ерекшелікті өңдеу AggregateException

.NET Framework ортасының 4.0 нұсқасында тапсырманы болдырмаудың өте ыңғайлы тәсілі болғанымен, құрылымды қамтамасыз ететін жаңа кіші жүйе енгізілген. Бұл жаңа кіші жүйе алып тастау белгісі ұғымына негізделеді. Тоқтату белгілері Task класында, сонымен қатар фабрикалық StartNew() әдісінің көмегімен ұсталады.

Тапсырманы болдырмау, әдетте, мынадай түрде орындалады. Алдымен көзден алып тастау белгілерін алып тастау белгісі пайда болады. Содан кейін бұл белгі міндетке беріледі, содан кейін ол оны болдырмауға сұрау салуды алу мәніне бақылауға тиіс. (Бұл сұрау тек болдырмау белгілерінің көзінен ғана түсе алады.) Егер болдырмау сұрауы алынса, тапсырма аяқталуы тиіс. Кейбір жағдайларда бұл ешқандай қосымша әрекетсіз міндеттің жай тоқтатылуы үшін жеткілікті, ал басқаларында - міндеттен алып тастау белгісі үшін ThrowIfCancellationRequested() әдісі шақырылуы тиіс. Осының арқасында күшін жоятын кодта тапсырманың жойылғаны белгілі болады. Енді тапсырманы алып тастау процесін егжей-тегжейлі қарастырайық.

Болдырмау белгісі CancellationToken түріндегі нысанның, яғни System.Threading аттар кеңістігінде анықталған құрылымның данасы болып табылады. CancellationToken құрылымында бірнеше қасиеттер мен әдістер анықталған, бірақ біз олардың екеуін пайдаланамыз. Біріншіден, бұл тек оқу үшін қол жетімді IsCancellationRequested сипаты, ол келесі түрде жарияланады.

public bool IsCancellationRequested { get; }

Егер тапсырманы болдырмау шақырушы белгі үшін сұралған болса, ол логикалық мәнді true береді, ал басқаша - false логикалық мәні. Екіншіден, бұл былайша жарияланатын ThrowIfCancellationRequested() әдісі.

public void ThrowIfCancellationRequested()

Егер осы әдіс шақырылатын алып тастау белгісі алып тастауға сұрау салуды алса, онда осы әдісте OperationCanceledException алып тастау туындайды.

Олай болмаған жағдайда ешқандай әрекет орындалмайды. Болдырмау кодында тапсырманы болдырмау шын мәнінде болғанына көз жеткізу үшін аталған ерекшелікті қадағалауды ұйымдастыруға болады. Әдетте, осы мақсатта алдымен AggregateException алып тастау ұсталады, содан кейін оның ішкі ерекшелігі InnerException немесе InnerExceptions қасиетінің көмегімен талданады. (Сипат InnerExceptions ерекшеліктер жиынтығы болып табылады.

Мысал 1. Тапсырманың орындалуын болдырмау кезінде аластама өңдеу

```
using System;
using System.Threading;
using System.Threading.Tasks;
class DemoCancelTask {
// Тапсырма ретінде орындалатын әдіс
static void MyTask(Object ct) {
CancellationToken cancelTok = (CancellationToken) ct;
// Тапсырманы іске қосудың алдында оның жойылмағанын тексеру
cancelTok.ThrowIfCancellationRequested();
Console.WriteLine("MyTask() іске қосылды");
```

```
for(int count = 0; count < 10; count++) {
// Тапсырманың жойылғанын бақылау үшін сұрау тәсілі қолданылады
if(cancelTok.IsCancellationRequested) {
Console.WriteLine("Тапсырманы жоюға сұраныс түсті");
cancelTok.ThrowIfCancellationRequested();
}
Thread.Sleep(500);
Console.WriteLine("MyTask() әдісіндегі санауыш мәні: " + count );
}
Console.WriteLine("MyTask аяқталды");
}
static void Main() {
Console.WriteLine("Негізгі ағын іске қосылды");
// Тапсырманы жою белгілерінің көзі объектісін құру
CancellationTokenSource cancelTokSrc = new CancellationTokenSource();
// Тапсырманы іске қосу, тапсырмаға және делегатқа жою белгісін беру
Task tsk = Task.Factory.StartNew(MyTask, cancelTokSrc.Token, cancelTokSrc.Token);
// Тапсырманың күші жойылғанға дейін орындалуына мүмкіндік беру
Thread.Sleep(2000);
try {
// Тапсырманың орындалуынан бас тарту
cancelTokSrc.Cancel();
// tsk тапсырмасы аяқталғанға дейін Main() әдісінің орындалуын кідірту
tsk.Wait();
} catch (AggregateException exc) {
if(tsk.IsCanceled)
Console.WriteLine("\ntsk тапсырмасының орындалуынан бас тарту\n");
Console.WriteLine(exe);
} finally {
tsk.Dispose();
cancelTokSrc.Dispose();
}
Console.WriteLine("Негізгі ағын аяқталды.");
}
}
```

Мысал 2. Ағыннан жұмысын тоқтату кезіндегі аластама

```
using System;
using System.Threading;
class MyThread {
public Thread Thrd;
public MyThread(string name) {
Thrd = new Thread(this.Run);
Thrd.Name = name;
Thrd.Start();
}
// Ағынға кіру нүктесі.
void Run() {
try {
Console.WriteLine(Thrd.Name + " басталды.");
for (int i = 1; i <= 1000; i++) {
```

```
Console.Write(i + " ");
if((i%10)==0) {
Console.WriteLine();
Thread.Sleep(250);
}
}
Console.WriteLine(Thrd.Name + " қалыпты аяқталды.");
} catch(ThreadAbortException exc) {
Console.WriteLine("Ағын тоқтатылды, аяқтау коды " + exc.ExceptionState);
} }
}
class UseAltAbort {
static void Main() {
MyThread mt1 = new MyThread("Ағын 1");
Thread.Sleep(1000); // туынды ағынның жұмысына рұқсат беру
Console.WriteLine("Ағынды тоқтату.");
mt1.Thrd.Abort (100);
mt1.Thrd.Join(); // ағын тоқтатылуын күту
Console.WriteLine("Негізгі ағын тоқтатылды.");
}}
```