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

Мысал 1. Тапсырмадан мән қайтару

```
using System;
using System.Threading;
using System.Threading.Tasks;
class DemoTask
{
    static bool MyTask()
    {
        return true;
    }

    static int Sumlt(object v)
    {
        int x = (int)v;
        int sum = 0;
        for (; x > 0; x--)
            sum += x;
        return sum;
    }

    static void Main()
    {
        Console.WriteLine("Негизги агын иске косылды.");

        Task<bool> tsk = Task<bool>.Factory.StartNew(MyTask);
        Console.WriteLine("MyTask тапсырмасынын аткарылу натижеси: " + tsk.Result);

        Task<int> tsk2 = Task<int>.Factory.StartNew(Sumlt, 3);
        Console.WriteLine("Sumlt тапсырмасынын аткарылу натижеси: " + tsk2.Result);
        tsk.Dispose();
        tsk2.Dispose();
        Console.WriteLine("Негизги агын аякталды.");
    }
}
```

Мысал 2. Тапсырмадан бас тарту

```
    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 одан бас тартылғанын тексеру
            cancelTok.ThrowIfCancellationRequested();
            Console.WriteLine("MyTask() иске косылды");
            for (int count = 0; count < 10; count++)
            {
                // Тапсырмадан бас тартылғанын бақылау үшін сұрастыру тәсілі қолданылады
                if (cancelTok.IsCancellationRequested)
```

```
                {
                    Console.WriteLine("Тапсырманы жоюга сураныс тусти");
                    cancelTok.ThrowIfCancellationRequested();
                }
                Thread.Sleep(500);
                Console.WriteLine("MyTask() адисиндеги санауыш мани: " + count);
            }
            Console.WriteLine("MyTask аякталды");
        }

        static void Main()
        {
            Console.WriteLine("Негизги агын иске косылды");
            // Тапсырмадан бас тарту белгілерінің көзін құру
            CancellationTokenSource cancelTokSrc = new CancellationTokenSource();
            // Тапсырманы іске қосу, тапсырмаға және делегатқа бас тарту 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 тапсырмасынын орындалуынан бас тарту\n");
                Console.WriteLine(exc);
            }
            finally
            {
                tsk.Dispose();
                cancelTokSrc.Dispose();
            }
            Console.WriteLine("Негизги агын аякталды.");
        }
    }
```

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.");
}}
```

Мысал 3. Ағын жұмысындағы аластаманы сәтсіз өңдеу

```
    class Program
    {
        public static void Main()
        {
            try
            {
                new Thread(Go).Start();
            }
            catch (Exception)
            {
                // Бұл код орындалмайды
                Console.WriteLine("Аластама!");
            }
        }
        static void Go() { throw null; } //  NullReferenceException
       }
```

Мысал 4. Ағын жұмысындағы аластаманы сәтті өңдеу

```
class Program
    {
        public static void Main()
        {
            new Thread(Go).Start();
        }
        static void Go()
        {
            try
            {
                //...
                throw null; // NullReferenceException
                // ...
            }
            catch (Exception)
            {
                // аластама өңдеу әрекеттері
                Console.WriteLine("Alastama!");

            }
         }

     }
```

Мысал 5. Тапсырма аластамаларын өңдеу

```
static void Main()
        {
            Task task = Task.Run(() => { throw null; });
            try
            {
                task.Wait();
            }
            catch (AggregateException aex)
            {
                if (aex.InnerException is NullReferenceException)
                    Console.WriteLine("Null!");
                else
                    throw;
            }
        }
```

Мысал 6. Тапсырмадан бас тартудың тағы бір мысалы

```
class Program
    {
        static void Main()
        {
            var cts = new CancellationTokenSource();
            CancellationToken token = cts.Token;
            cts.CancelAfter(500);

            Task task = Task.Factory.StartNew(() =>
            {
                Thread.Sleep(1000);
                token.ThrowIfCancellationRequested();

            }, token);

            try { task.Wait(); }
            catch (AggregateException ex)
            {
                Console.WriteLine(ex.InnerException is TaskCanceledException);
                // a subclass of OperationCanceledException.
                Console.WriteLine(task.IsCanceled);
                Console.WriteLine(task.Status); // Canceled
            }

        }
```

Мысал 7. Parent-Child байланысы

```
static void Main()
        {
            Task parent = Task.Factory.StartNew(() =>
            {
                Console.WriteLine("I am a parent");
                Task.Factory.StartNew(() => // Detached task
                {

                    Console.WriteLine("I am detached");
                });
                Task.Factory.StartNew(() => // Child task
                {
                    Console.WriteLine("I am a child");
                }, TaskCreationOptions.AttachedToParent);
            });
              parent.Wait();
```

```
        }
```

```
static void Main()
        {
            TaskCreationOptions atp = TaskCreationOptions.AttachedToParent;
            var parent = Task.Factory.StartNew(() =>
            {
                Task.Factory.StartNew(() => // Child
                {
                    Task.Factory.StartNew(() => { throw null; }, atp); // Grandchild
                }, atp);
            });
            // The following call throws a NullReferenceException (wrapped
            // in nested AggregateExceptions):
            parent.Wait();
        }
```

Мысал 8. Басты тапсырманың жалғасы

```
static void Main()
        {
            TaskCreationOptions atp = TaskCreationOptions.AttachedToParent;
            Task.Factory.StartNew(() =>
            {
                Task.Factory.StartNew(() => { throw null; }, atp);
                Task.Factory.StartNew(() => { throw null; }, atp);
                Task.Factory.StartNew(() => { throw null; }, atp);
            })
            .ContinueWith(p => Console.WriteLine(p.Exception),
TaskContinuationOptions.OnlyOnFaulted);
            Console.ReadKey();

        }
```

Мысал 9. AggregateException өңдеу

```
class Program
    {
        static void Main()
        {
            var parent = Task.Factory.StartNew(() =>
            {
                // We'll throw 3 exceptions at once using 3 child tasks:
                int[] numbers = { 0 };
                var childFactory = new TaskFactory
                (TaskCreationOptions.AttachedToParent, TaskContinuationOptions.None);
                childFactory.StartNew(() => 5 / numbers[0]); // Division by zero
                childFactory.StartNew(() => numbers[1]); // Index out of range
                childFactory.StartNew(() => { throw null; }); // Null reference
            });
            try { parent.Wait(); }
            catch (AggregateException aex)
            {
                aex.Flatten().Handle(ex =>
                {
                    if (ex is DivideByZeroException)
                    {
                        Console.WriteLine("Divide by zero");
```

```
                    return true;
                }
                if (ex is IndexOutOfRangeException)
                {
                    Console.WriteLine("Index out of range");
                    return true;
                }
                return false; // All other exceptions will get rethrown
            });
        }
    }
}
```