Level UP

IT-образование
в Петербурге
и удалённо

Towards the .NET Junior Developer

The extremely solid course

Lesson 7

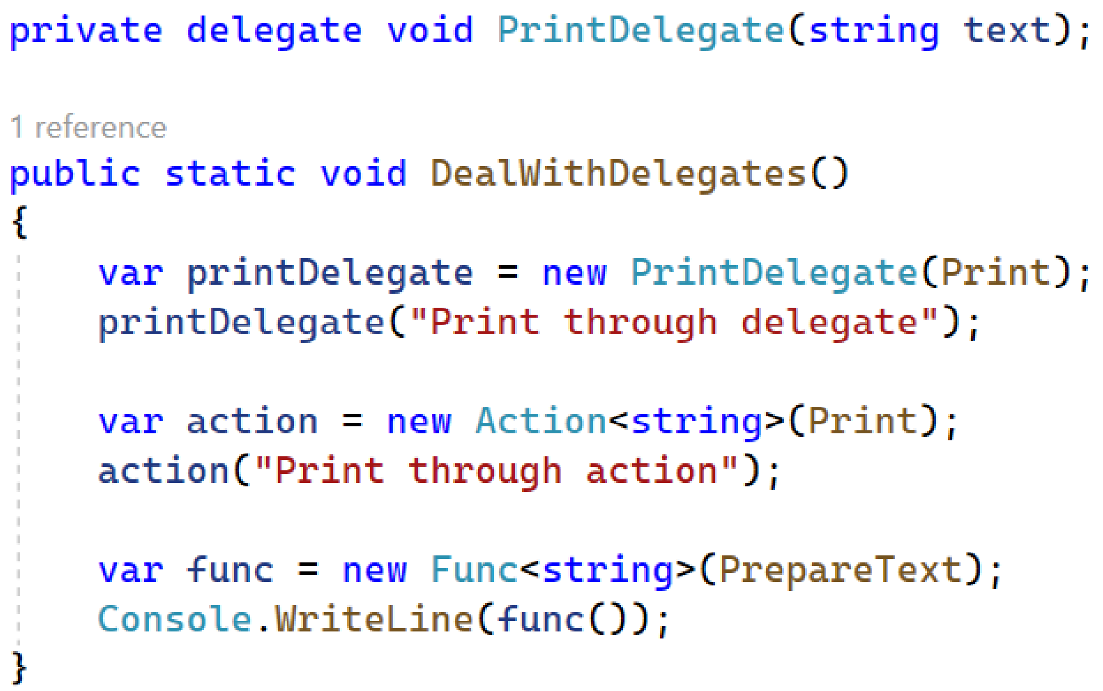
Asynchronous and multithreaded programming

Agenda

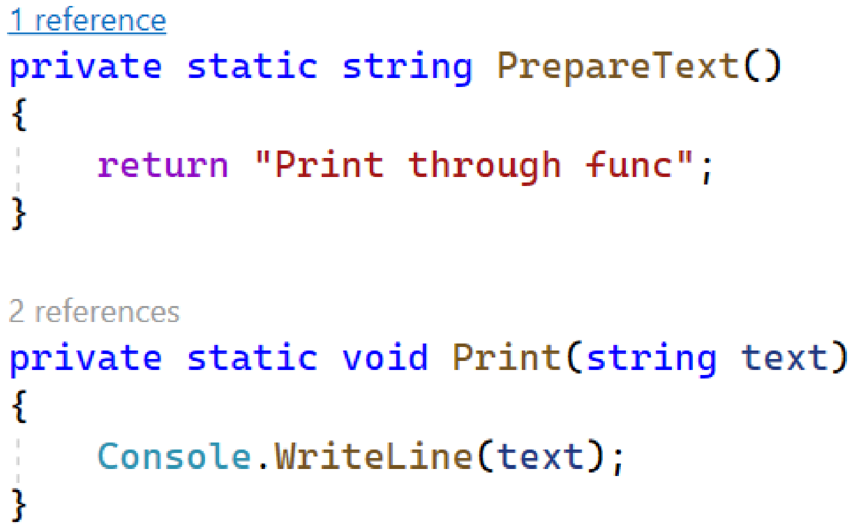
- Delegates and events
  - Delegates
  - Events
- Processes and threads
  - Process
  - Thread
  - Thread Pool
  - Tasks
- Asynchronous programming
  - Asynchronous execution
  - async/await
- Multithreaded programming
  - TPL
  - Threads synchronization
- Books of the day
- Links of the day
- Hometask

Delegates and events

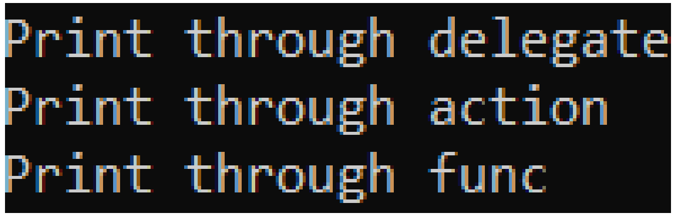
Delegates



```
private delegate void PrintDelegate(string text);

1 reference
public static void DealWithDelegates()
{
    var printDelegate = new PrintDelegate(Print);
    printDelegate("Print through delegate");

    var action = new Action<string>(Print);
    action("Print through action");

    var func = new Func<string>(PrepareText);
    Console.WriteLine(func());
}
```



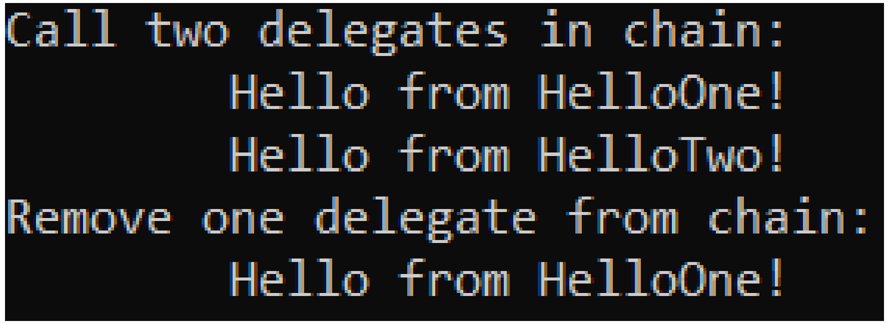
```
1 reference
private static string PrepareText()
{
    return "Print through func";
}

2 references
private static void Print(string text)
{
    Console.WriteLine(text);
}
```



```
Print through delegate
Print through action
Print through func
```

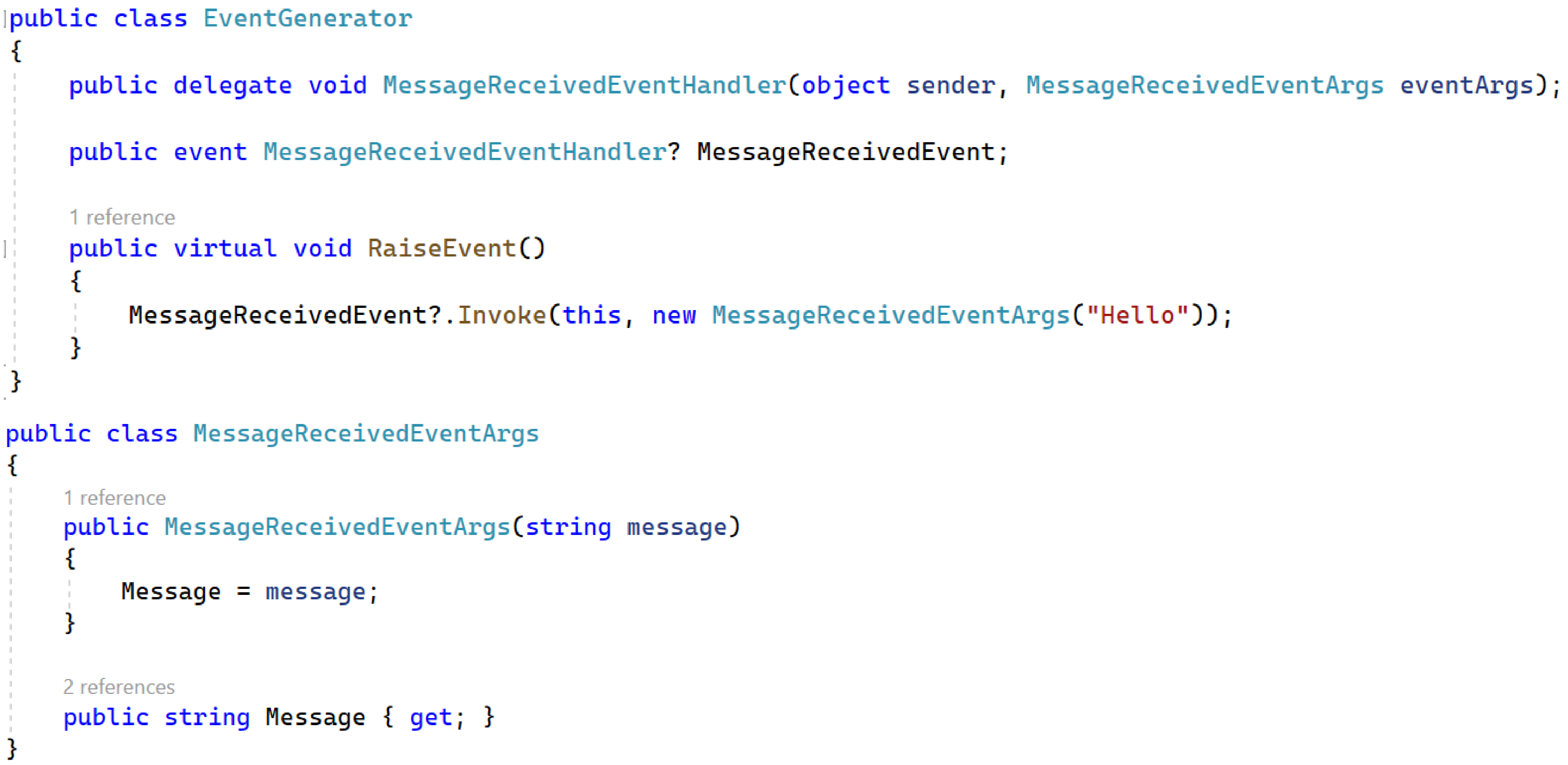
Delegates

```
var actionOne = new Action(HelloOne);
var actionTwo = new Action(HelloTwo);

var chain = actionOne + actionTwo;

Console.WriteLine("Call two delegates in chain:");
chain();

Console.WriteLine("Remove one delegate from chain:");
chain -= actionTwo;
chain();
```



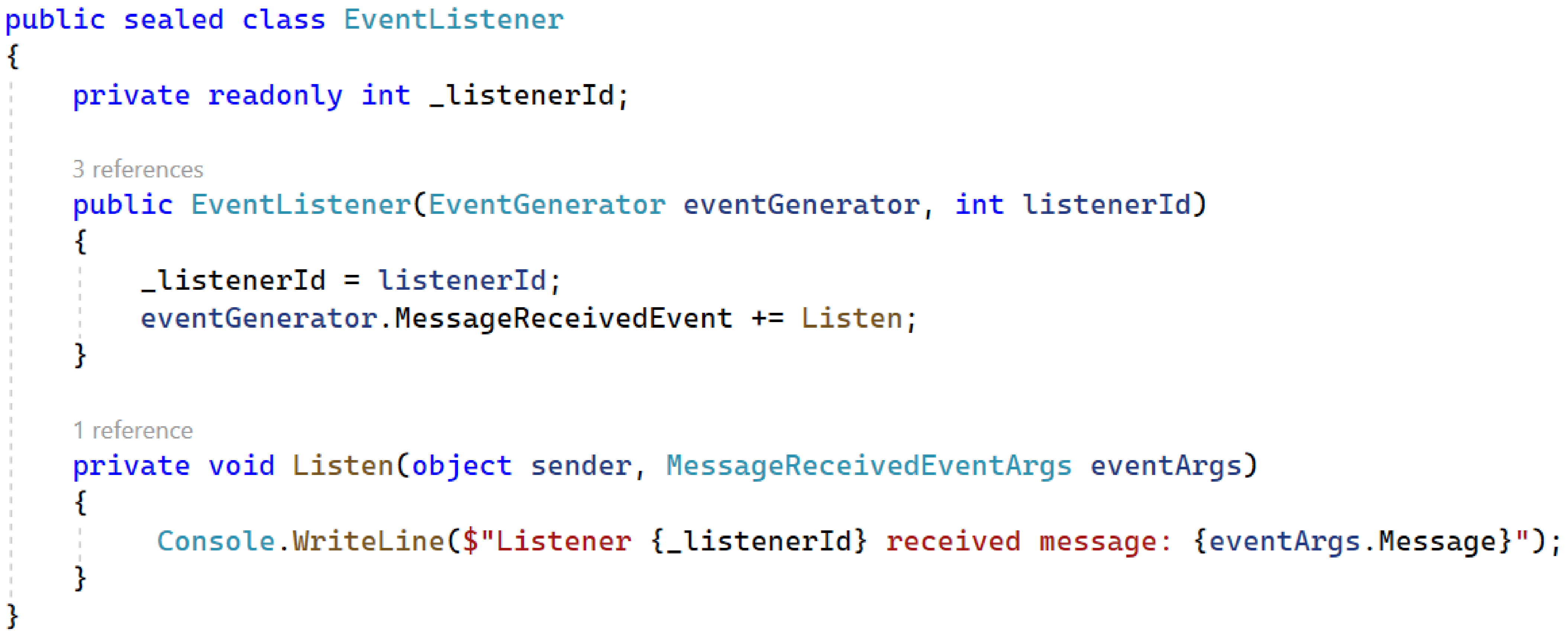
```
Call two delegates in chain:
        Hello from HelloOne!
        Hello from HelloTwo!
Remove one delegate from chain:
        Hello from HelloOne!
```

Events



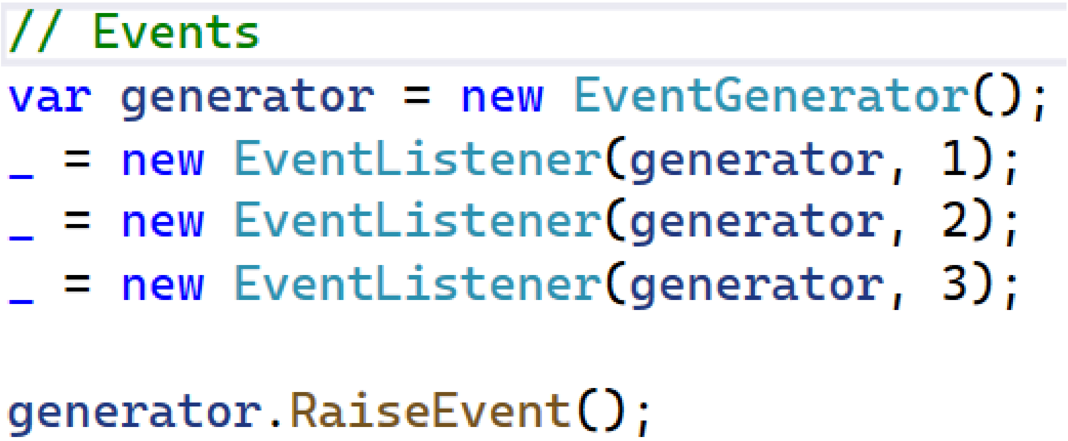
```
public class EventGenerator
{
    public delegate void MessageReceivedEventHandler(object sender, MessageReceivedEventArgs eventArgs);

    public event MessageReceivedEventHandler? MessageReceivedEvent;

    1 reference
    public virtual void RaiseEvent()
    {
        MessageReceivedEvent?.Invoke(this, new MessageReceivedEventArgs("Hello"));
    }
}

public class MessageReceivedEventArgs
{
    1 reference
    public MessageReceivedEventArgs(string message)
    {
        Message = message;
    }

    2 references
    public string Message { get; }
}
```

Events

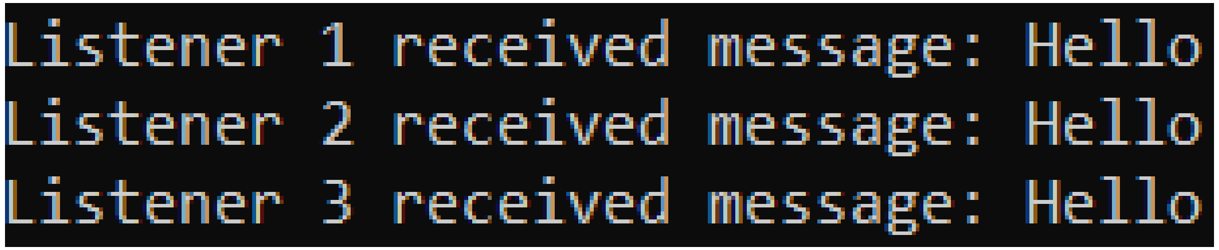


```
public sealed class EventListener
{
    private readonly int _listenerId;

    3 references
    public EventListener(EventGenerator eventGenerator, int listenerId)
    {
        _listenerId = listenerId;
        eventGenerator.MessageReceivedEvent += Listen;
    }

    1 reference
    private void Listen(object sender, MessageReceivedEventArgs eventArgs)
    {
         Console.WriteLine($"Listener {_listenerId} received message: {eventArgs.Message}");
    }
}
```

Events



```
// Events
var generator = new EventGenerator();
_ = new EventListener(generator, 1);
_ = new EventListener(generator, 2);
_ = new EventListener(generator, 3);

generator.RaiseEvent();
```

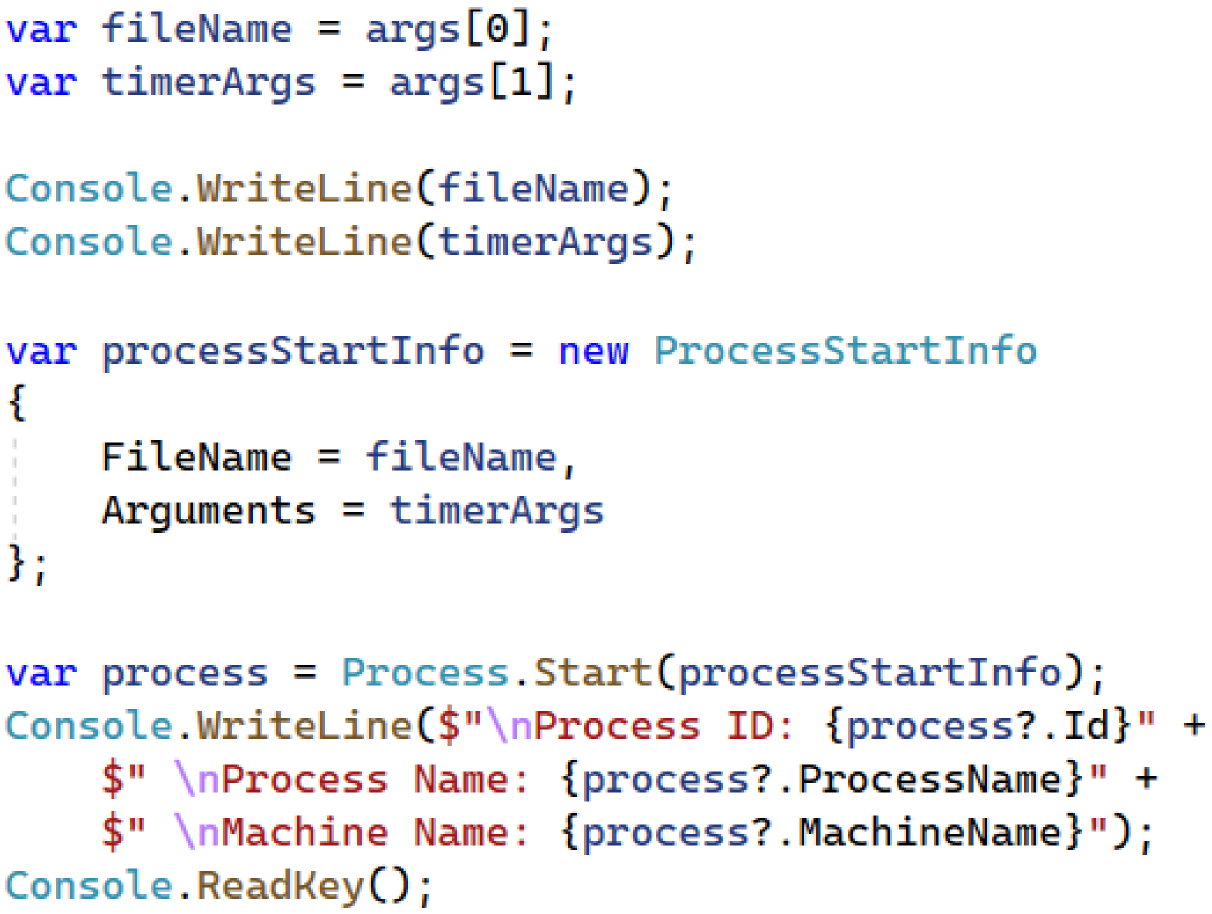


```
Listener 1 received message: Hello
Listener 2 received message: Hello
Listener 3 received message: Hello
```

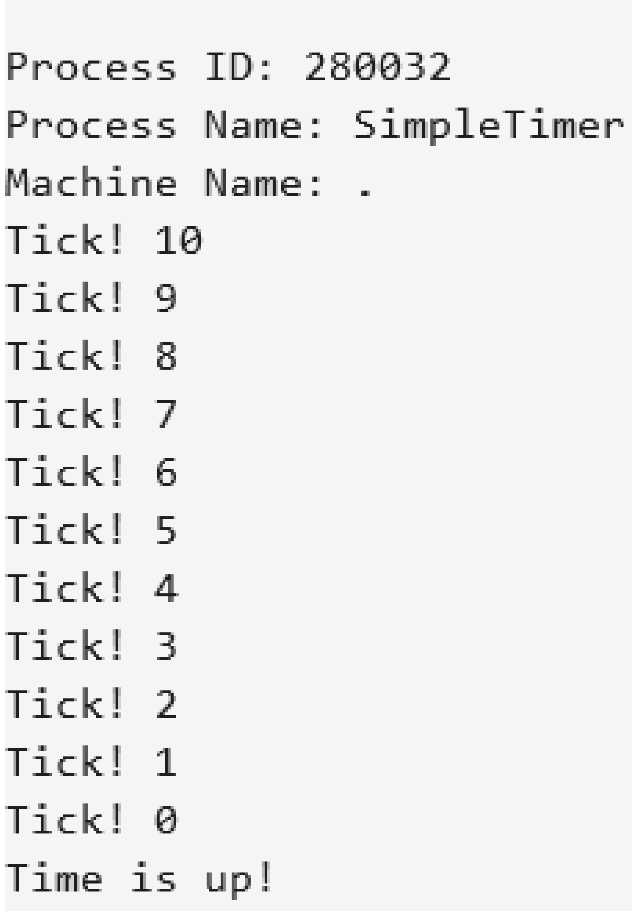
Delegates and events demo

Processes and threads

Process

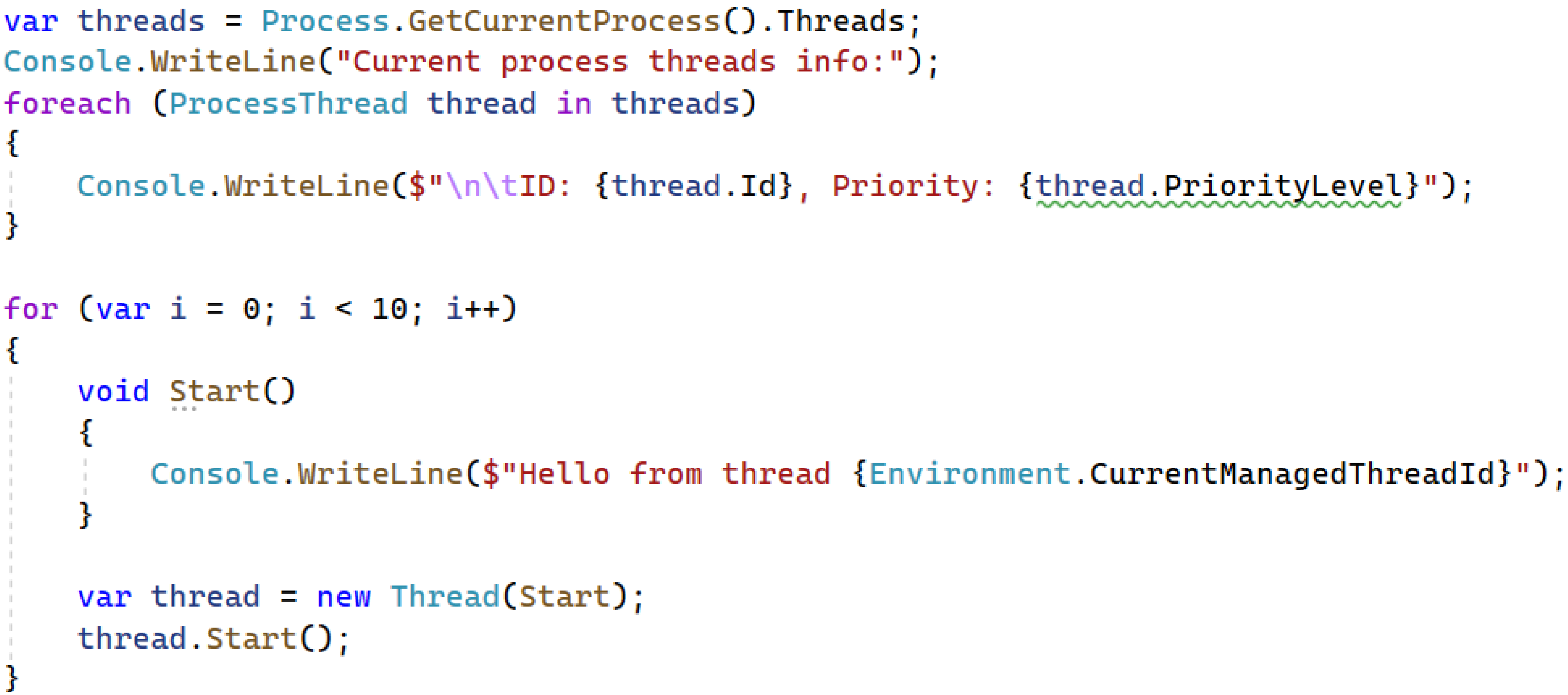


```
var fileName = args[0];
var timerArgs = args[1];

Console.WriteLine(fileName);
Console.WriteLine(timerArgs);

var processStartInfo = new ProcessStartInfo
{
    FileName = fileName,
    Arguments = timerArgs
};

var process = Process.Start(processStartInfo);
Console.WriteLine($"\nProcess ID: {process?.Id}" +
    $" \nProcess Name: {process?.ProcessName}" +
    $" \nMachine Name: {process?.MachineName}");
Console.ReadKey();
```

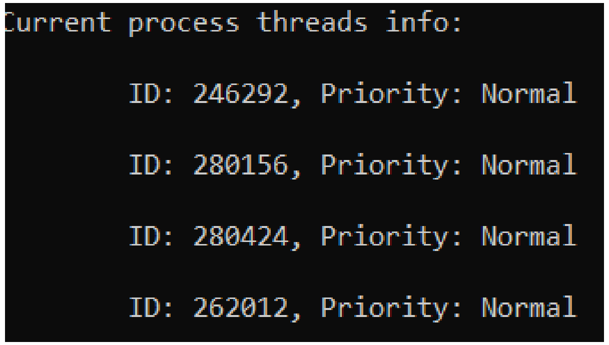


```
Process ID: 280032
Process Name: SimpleTimer
Machine Name: .
Tick! 10
Tick! 9
Tick! 8
Tick! 7
Tick! 6
Tick! 5
Tick! 4
Tick! 3
Tick! 2
Tick! 1
Tick! 0
Time is up!
```

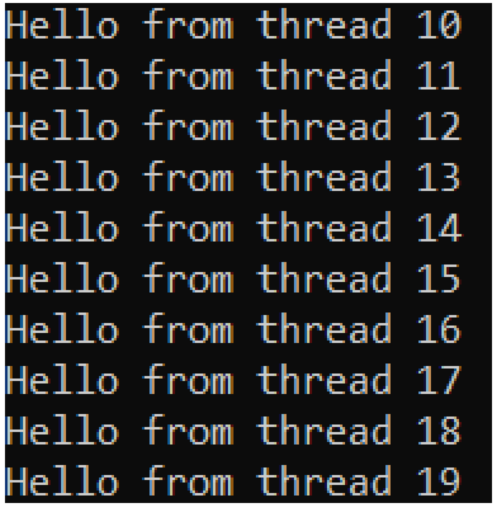
Thread



```
var threads = Process.GetCurrentProcess().Threads;
Console.WriteLine("Current process threads info:");
foreach (ProcessThread thread in threads)
{
    Console.WriteLine($"\n\tID: {thread.Id}, Priority: {thread.PriorityLevel}");
}

for (var i = 0; i < 10; i++)
{
    void Start()
    {
        Console.WriteLine($"Hello from thread {Environment.CurrentManagedThreadId}");
    }

    var thread = new Thread(Start);
    thread.Start();
}
```

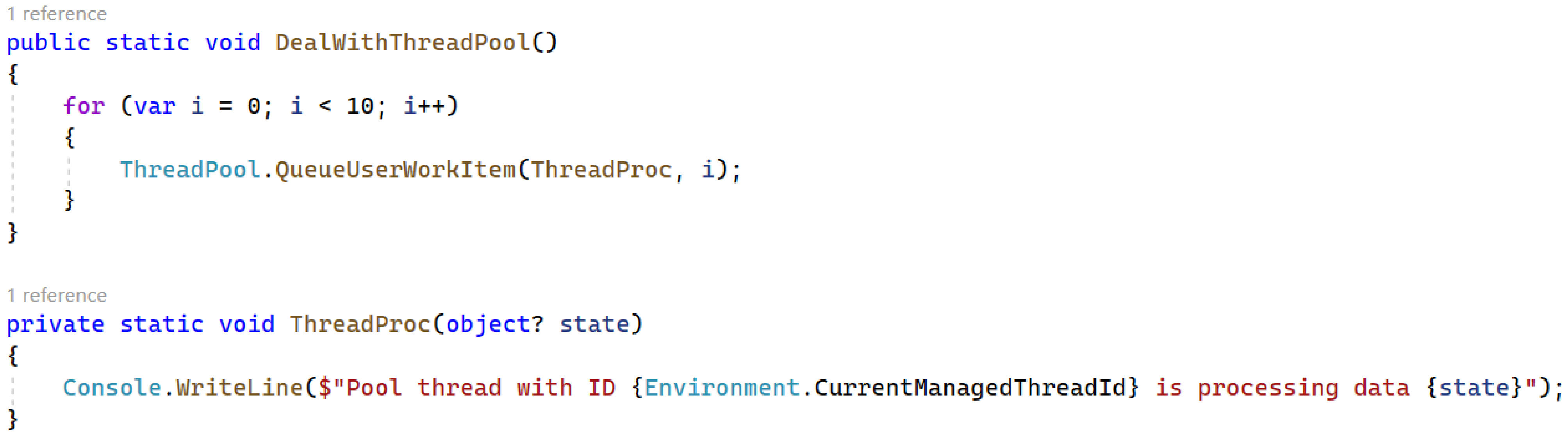


```
Current process threads info:

        ID: 246292, Priority: Normal

        ID: 280156, Priority: Normal

        ID: 280424, Priority: Normal

        ID: 262012, Priority: Normal
```

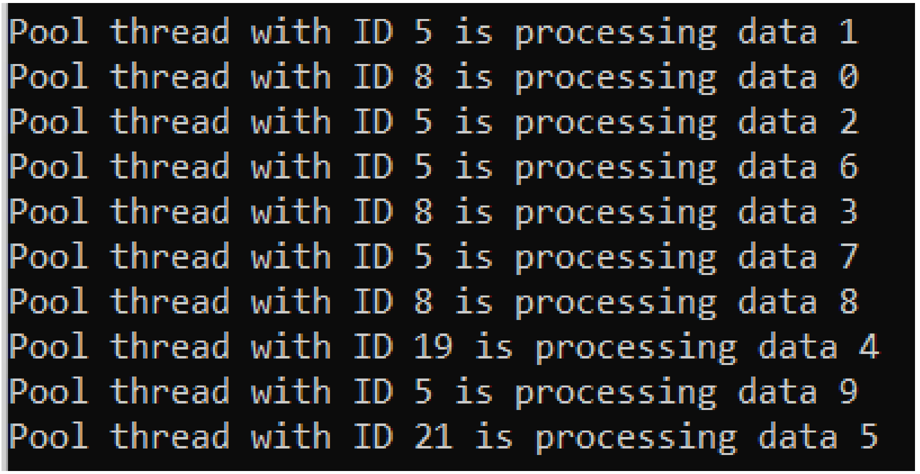


```
Hello from thread 10
Hello from thread 11
Hello from thread 12
Hello from thread 13
Hello from thread 14
Hello from thread 15
Hello from thread 16
Hello from thread 17
Hello from thread 18
Hello from thread 19
```

Thread Pool

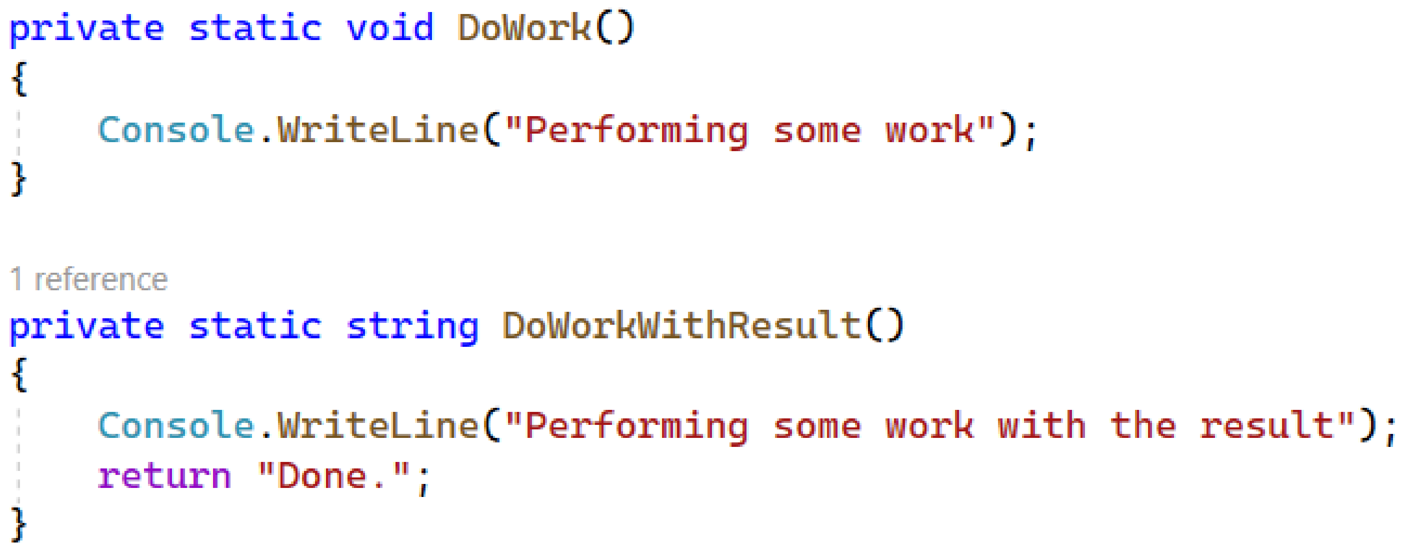


```
1 reference
public static void DealWithThreadPool()
{
    for (var i = 0; i < 10; i++)
    {
        ThreadPool.QueueUserWorkItem(ThreadProc, i);
    }
}

1 reference
private static void ThreadProc(object? state)
{
    Console.WriteLine($"Pool thread with ID {Environment.CurrentManagedThreadId} is processing data {state}");
}
```



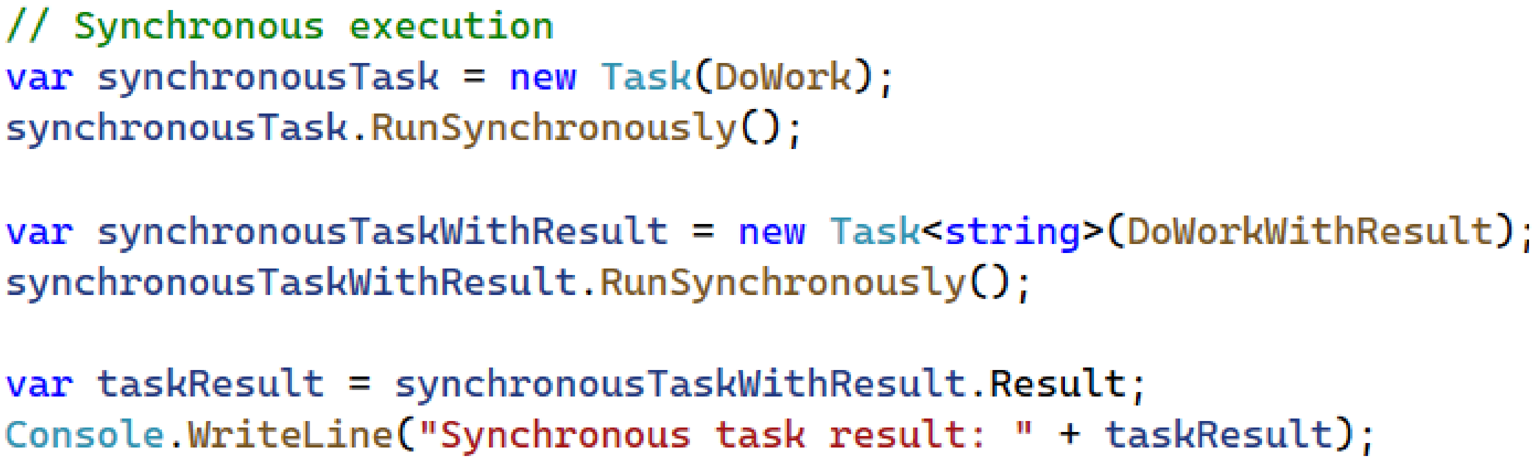
```
Pool thread with ID 5 is processing data 1
Pool thread with ID 8 is processing data 0
Pool thread with ID 5 is processing data 2
Pool thread with ID 5 is processing data 6
Pool thread with ID 8 is processing data 3
Pool thread with ID 5 is processing data 7
Pool thread with ID 8 is processing data 8
Pool thread with ID 19 is processing data 4
Pool thread with ID 5 is processing data 9
Pool thread with ID 21 is processing data 5
```

Tasks

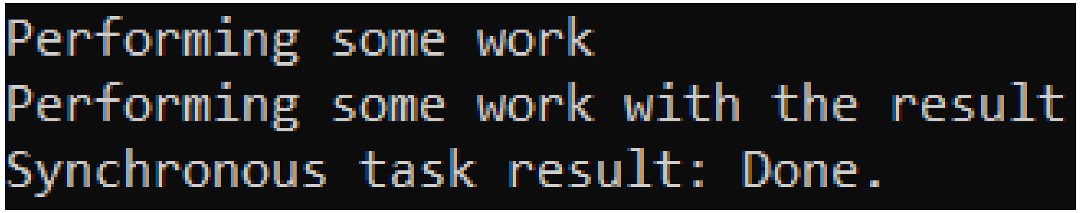


```
private static void DoWork()
{
    Console.WriteLine("Performing some work");
}

1 reference
private static string DoWorkWithResult()
{
    Console.WriteLine("Performing some work with the result");
    return "Done.";
}
```

Tasks – synchronous task execution

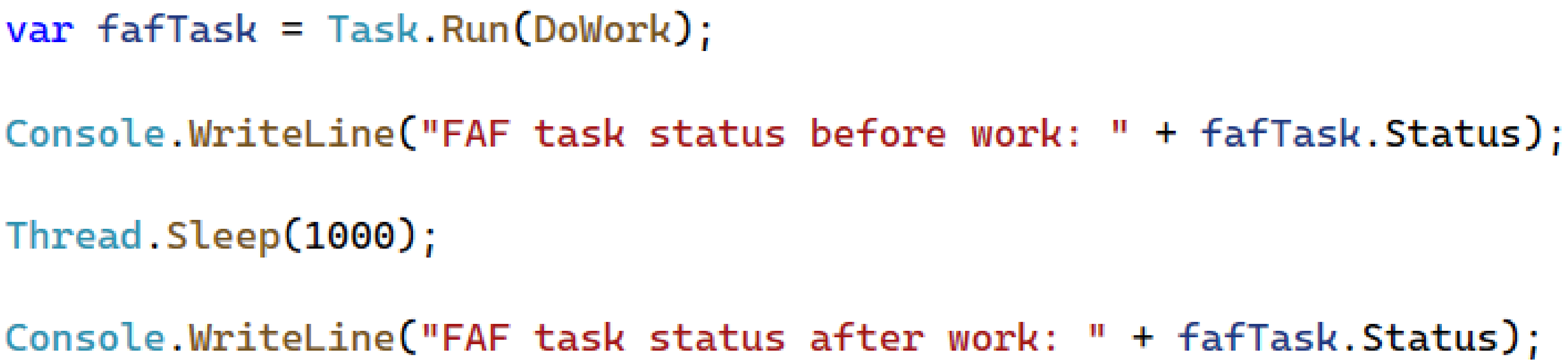


```
// Synchronous execution
var synchronousTask = new Task(DoWork);
synchronousTask.RunSynchronously();

var synchronousTaskWithResult = new Task<string>(DoWorkWithResult);
synchronousTaskWithResult.RunSynchronously();

var taskResult = synchronousTaskWithResult.Result;
Console.WriteLine("Synchronous task result: " + taskResult);
```

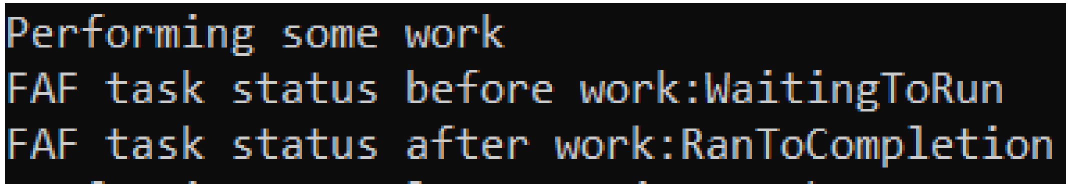


```
Performing some work
Performing some work with the result
Synchronous task result: Done.
```

Tasks – fire-and-forget

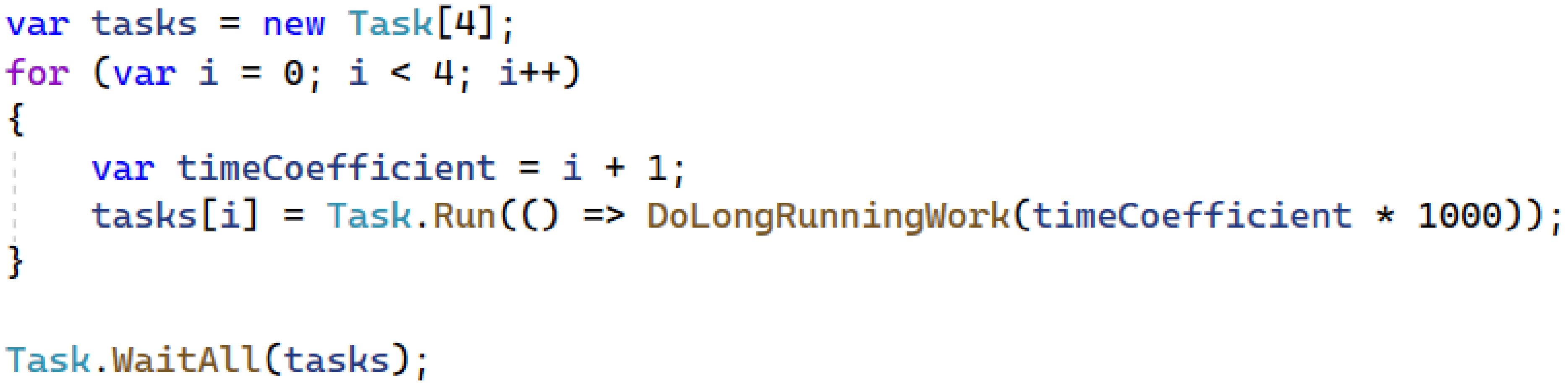


```
var fafTask = Task.Run(DoWork);

Console.WriteLine("FAF task status before work: " + fafTask.Status);

Thread.Sleep(1000);

Console.WriteLine("FAF task status after work: " + fafTask.Status);
```

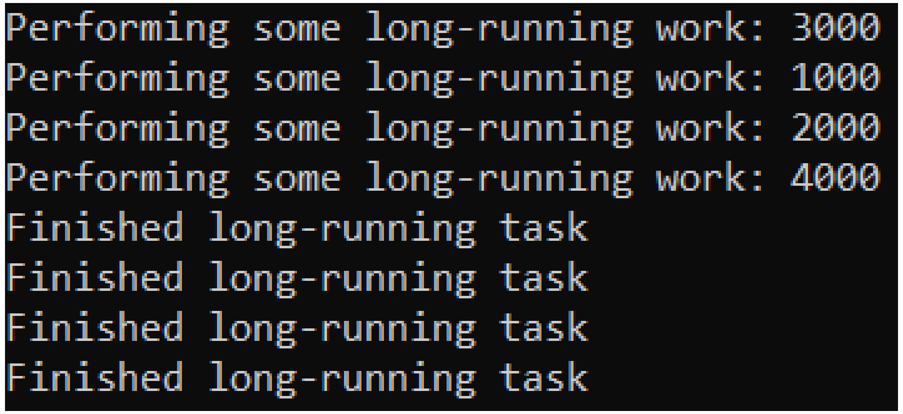


```
Performing some work
FAF task status before work:WaitingToRun
FAF task status after work:RanToCompletion
```

Tasks – waiting the group of tasks

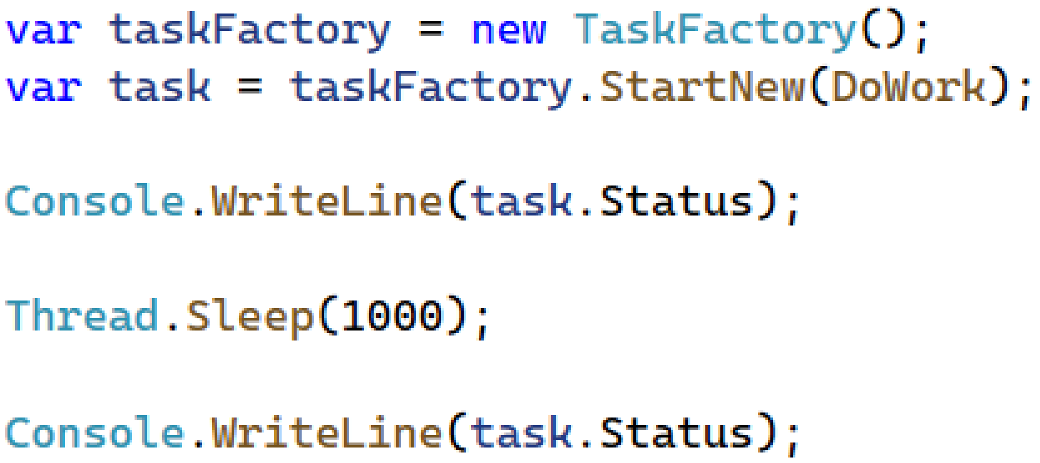


```
var tasks = new Task[4];
for (var i = 0; i < 4; i++)
{
    var timeCoefficient = i + 1;
    tasks[i] = Task.Run(() => DoLongRunningWork(timeCoefficient * 1000));
}

Task.WaitAll(tasks);
```

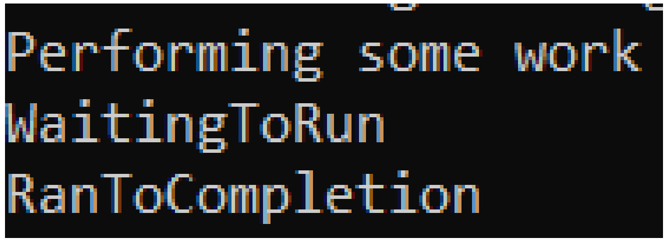


```
Performing some long-running work: 3000
Performing some long-running work: 1000
Performing some long-running work: 2000
Performing some long-running work: 4000
Finished long-running task
Finished long-running task
Finished long-running task
Finished long-running task
```

Tasks - TaskFactory

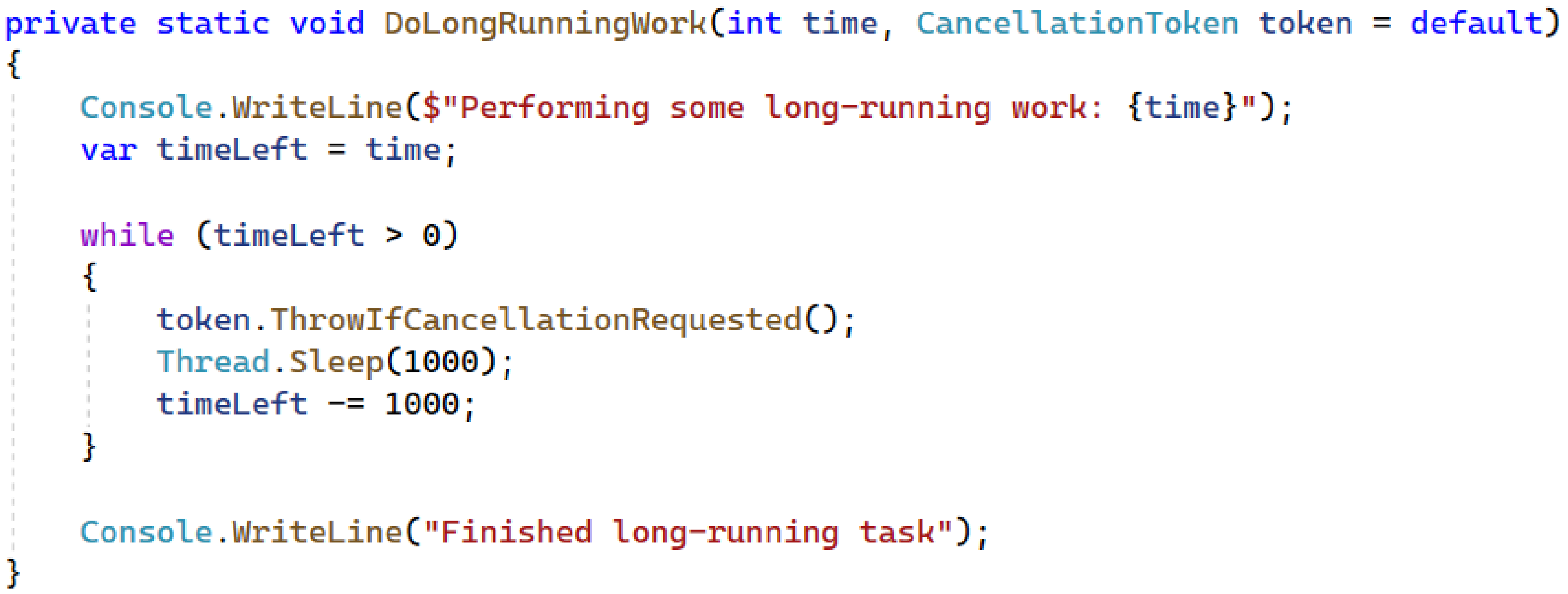


```
var taskFactory = new TaskFactory();
var task = taskFactory.StartNew(DoWork);

Console.WriteLine(task.Status);

Thread.Sleep(1000);

Console.WriteLine(task.Status);
```



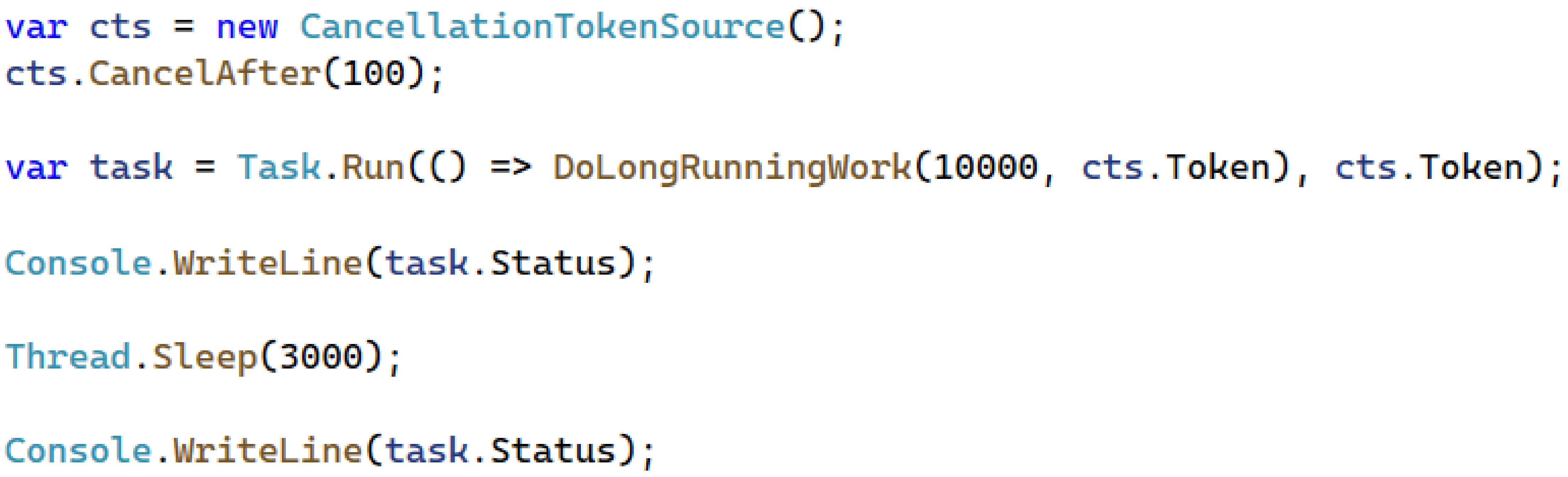
```
Performing some work
WaitingToRun
RanToCompletion
```

Tasks - cancellation

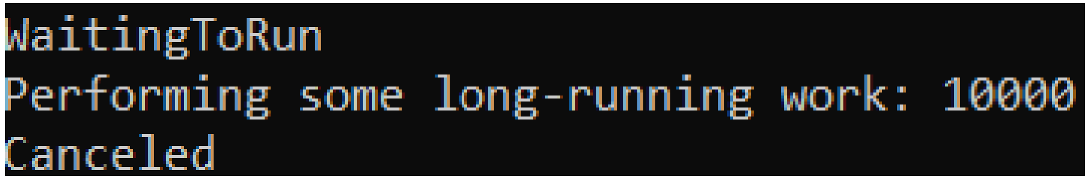


```
private static void DoLongRunningWork(int time, CancellationToken token = default)
{
    Console.WriteLine($"Performing some long-running work: {time}");
    var timeLeft = time;

    while (timeLeft > 0)
    {
        token.ThrowIfCancellationRequested();
        Thread.Sleep(1000);
        timeLeft -= 1000;
    }

    Console.WriteLine("Finished long-running task");
}
```

Tasks - cancellation



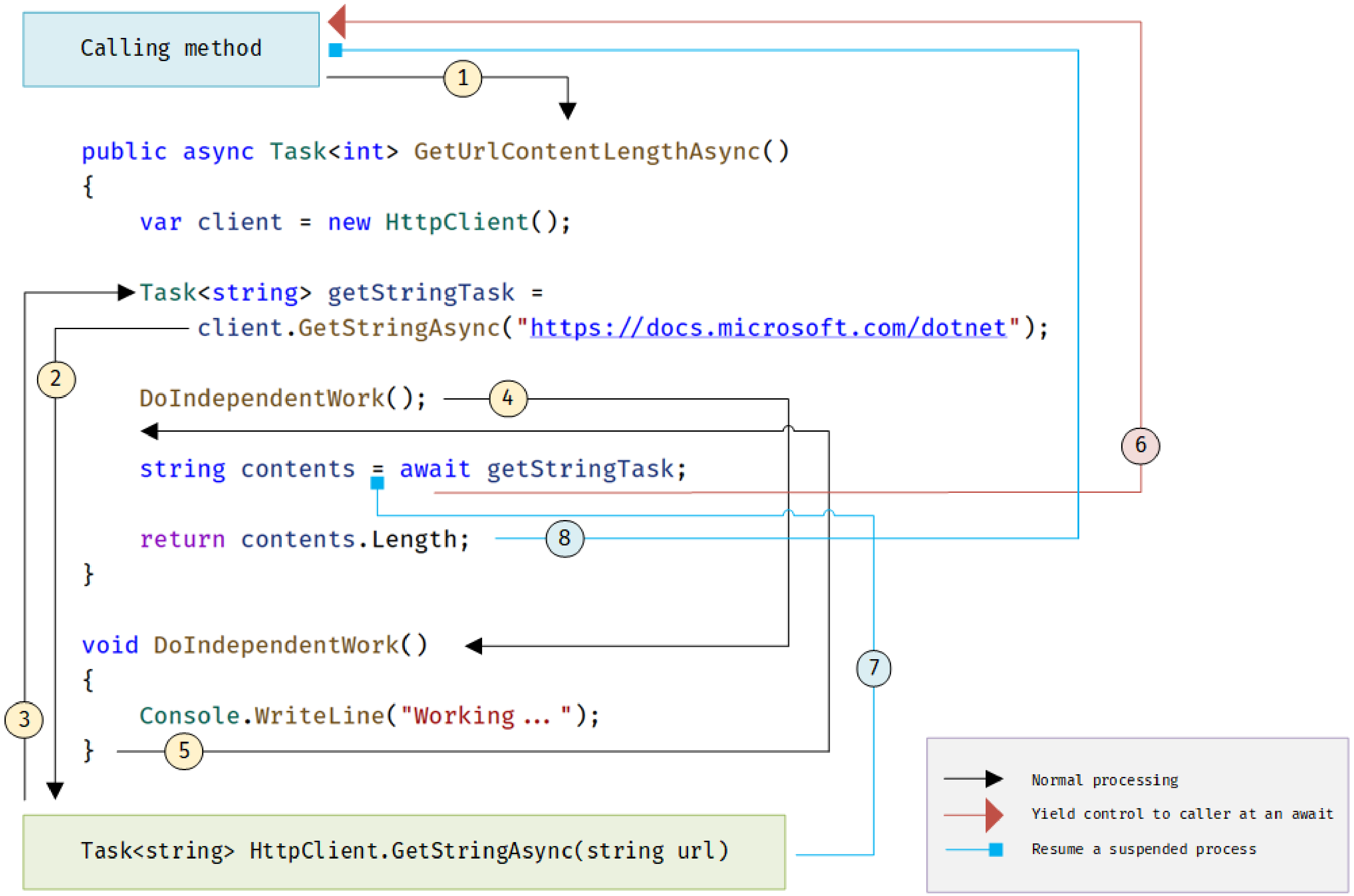
```
var cts = new CancellationTokenSource();
cts.CancelAfter(100);

var task = Task.Run(() => DoLongRunningWork(10000, cts.Token), cts.Token);

Console.WriteLine(task.Status);

Thread.Sleep(3000);

Console.WriteLine(task.Status);
```



```
WaitingToRun
Performing some long-running work: 10000
Canceled
```

Asynchronous programming

Asynchronous execution



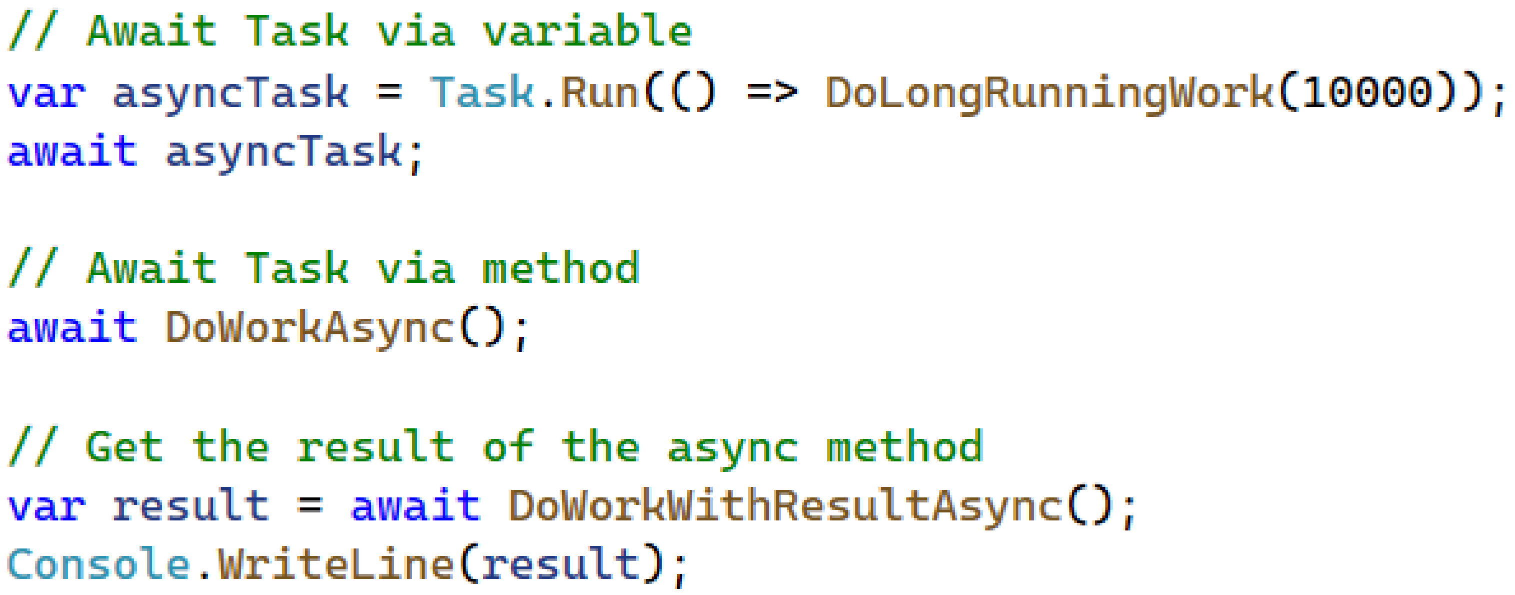
Calling method

```
public async Task<int> GetUrlContentLengthAsync()
{
    var client = new HttpClient();

    Task<string> getStringTask =
        client.GetStringAsync("https://docs.microsoft.com/dotnet");

    DoIndependentWork();

    string contents = await getStringTask;

    return contents.Length;
}

void DoIndependentWork()
{
    Console.WriteLine("Working . . . ");
}
```

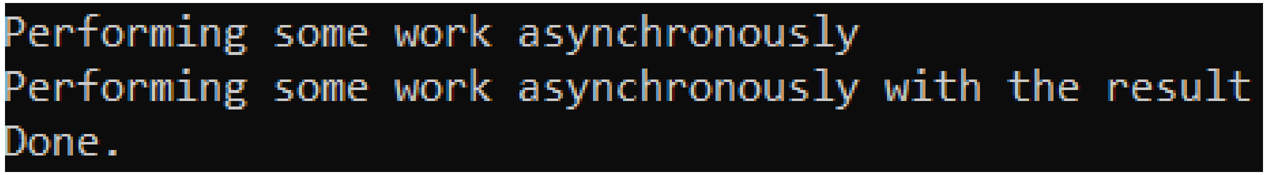
Task<string> HttpClient.GetStringAsync(string url)

- Normal processing
- Yield control to caller at an await
- Resume a suspended process

async/await



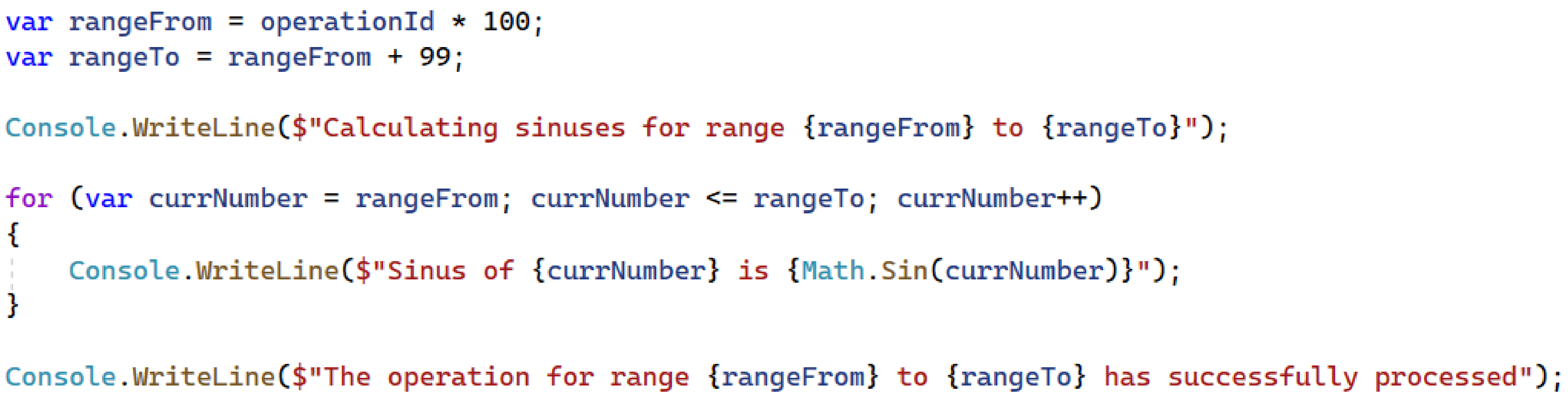
```
// Await Task via variable
var asyncTask = Task.Run(() => DoLongRunningWork(10000));
await asyncTask;

// Await Task via method
await DoWorkAsync();

// Get the result of the async method
var result = await DoWorkWithResultAsync();
Console.WriteLine(result);
```



```
Performing some work asynchronously
Performing some work asynchronously with the result
Done.
```

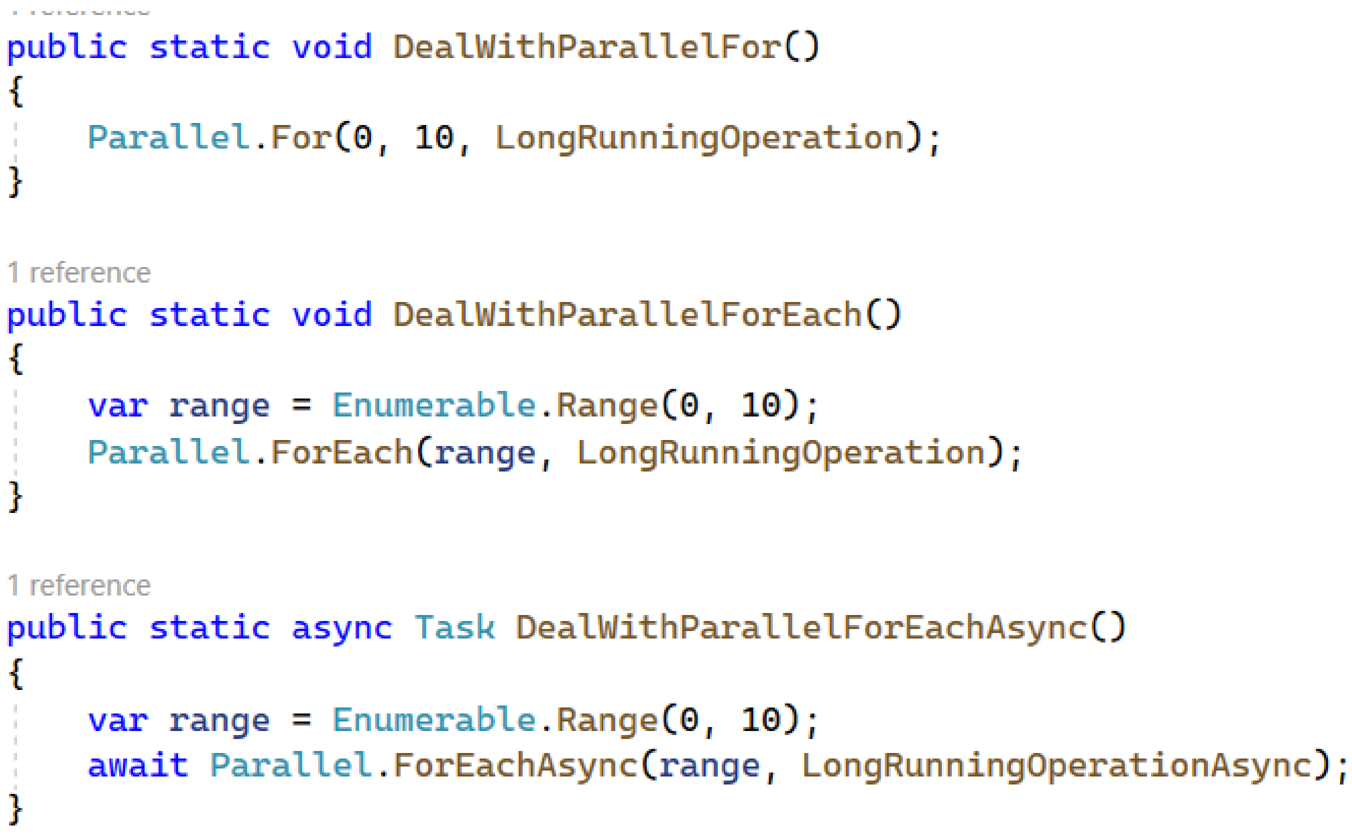
Multithreaded programming

TPL – Task Parallel Library

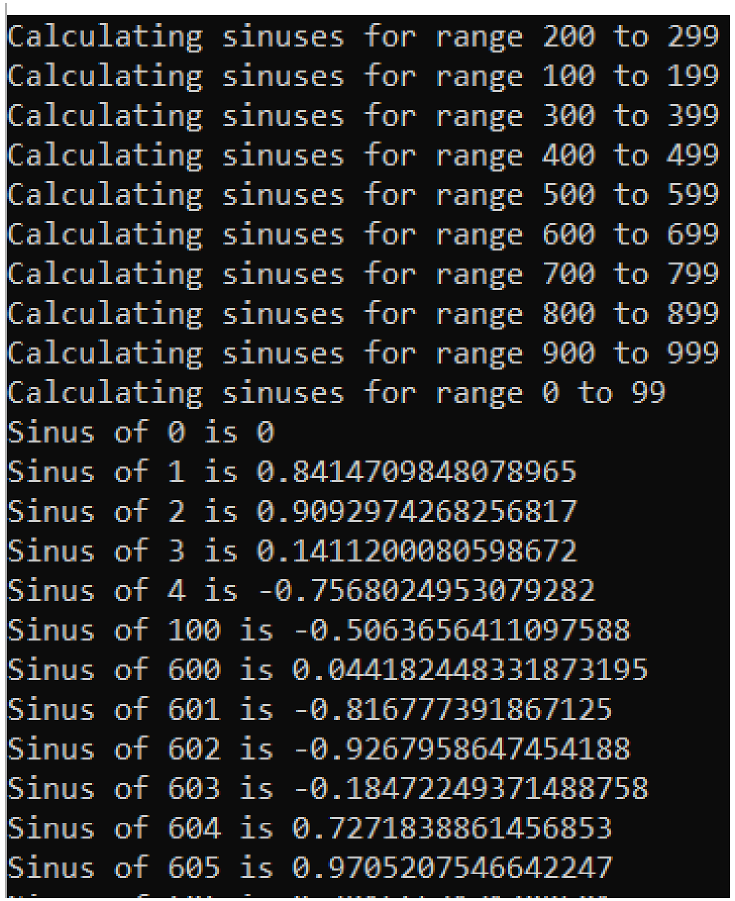


```
var rangeFrom = operationId * 100;
var rangeTo = rangeFrom + 99;

Console.WriteLine($"Calculating sinuses for range {rangeFrom} to {rangeTo}");

for (var currNumber = rangeFrom; currNumber <= rangeTo; currNumber++)
{
    Console.WriteLine($"Sinus of {currNumber} is {Math.Sin(currNumber)}");
}

Console.WriteLine($"The operation for range {rangeFrom} to {rangeTo} has successfully processed");
```

TPL – Task Parallel Library

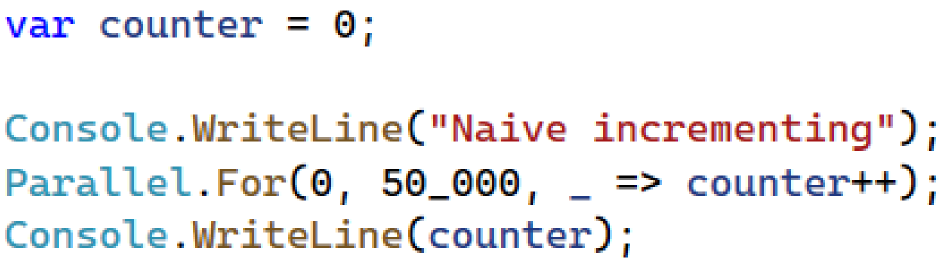


```
public static void DealWithParallelFor()
{
    Parallel.For(0, 10, LongRunningOperation);
}

1 reference
public static void DealWithParallelForEach()
{
    var range = Enumerable.Range(0, 10);
    Parallel.ForEach(range, LongRunningOperation);
}

1 reference
public static async Task DealWithParallelForEachAsync()
{
    var range = Enumerable.Range(0, 10);
    await Parallel.ForEachAsync(range, LongRunningOperationAsync);
}
```

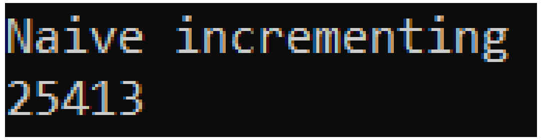


```
Calculating sinuses for range 200 to 299
Calculating sinuses for range 100 to 199
Calculating sinuses for range 300 to 399
Calculating sinuses for range 400 to 499
Calculating sinuses for range 500 to 599
Calculating sinuses for range 600 to 699
Calculating sinuses for range 700 to 799
Calculating sinuses for range 800 to 899
Calculating sinuses for range 900 to 999
Calculating sinuses for range 0 to 99
Sinus of 0 is 0
Sinus of 1 is 0.8414709848078965
Sinus of 2 is 0.9092974268256817
Sinus of 3 is 0.1411200080598672
Sinus of 4 is -0.7568024953079282
Sinus of 100 is -0.5063656411097588
Sinus of 600 is 0.044182448331873195
Sinus of 601 is -0.81677391867125
Sinus of 602 is -0.9267958647454188
Sinus of 603 is -0.18472249371488758
Sinus of 604 is 0.7271838861456853
Sinus of 605 is 0.9705207546642247
```

Threads synchronization - problem

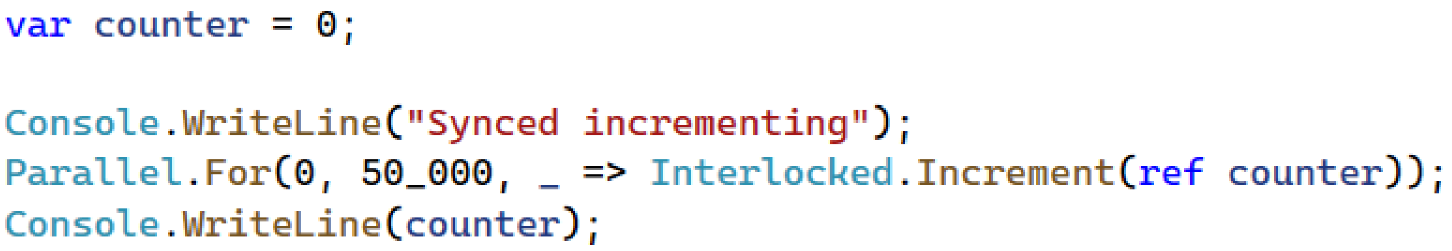


```
var counter = 0;

Console.WriteLine("Naive incrementing");
Parallel.For(0, 50_000, _ => counter++);
Console.WriteLine(counter);
```

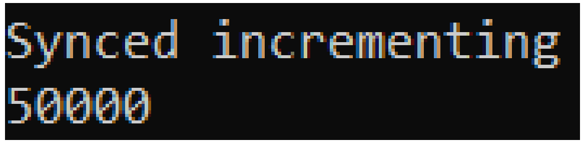


```
Naive incrementing
25413
```

Threads synchronization – Interlocked

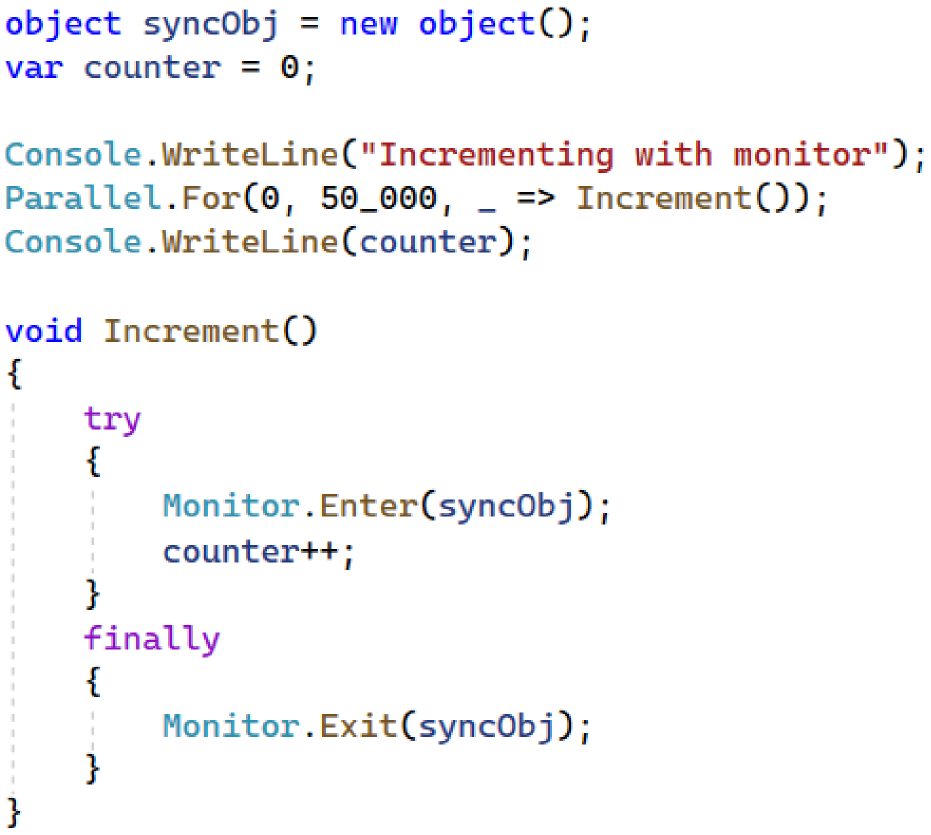


```
var counter = 0;

Console.WriteLine("Synced incrementing");
Parallel.For(0, 50_000, _ => Interlocked.Increment(ref counter));
Console.WriteLine(counter);
```

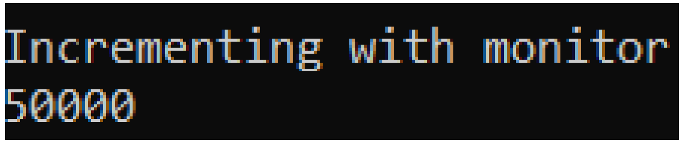


```
Synced incrementing
50000
```

Threads synchronization - Monitor

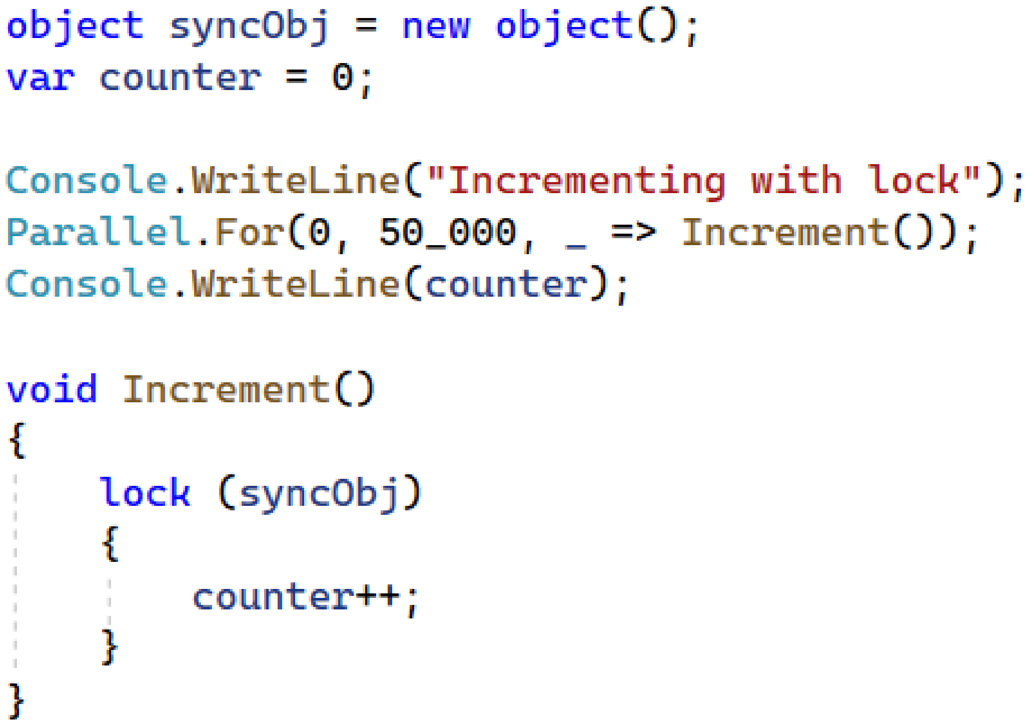


```
object syncObj = new object();
var counter = 0;

Console.WriteLine("Incrementing with monitor");
Parallel.For(0, 50_000, _ => Increment());
Console.WriteLine(counter);

void Increment()
{
    try
    {
        Monitor.Enter(syncObj);
        counter++;
    }
    finally
    {
        Monitor.Exit(syncObj);
    }
}
```

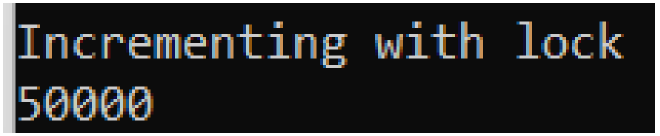


```
Incrementing with monitor
50000
```

Threads synchronization - lock

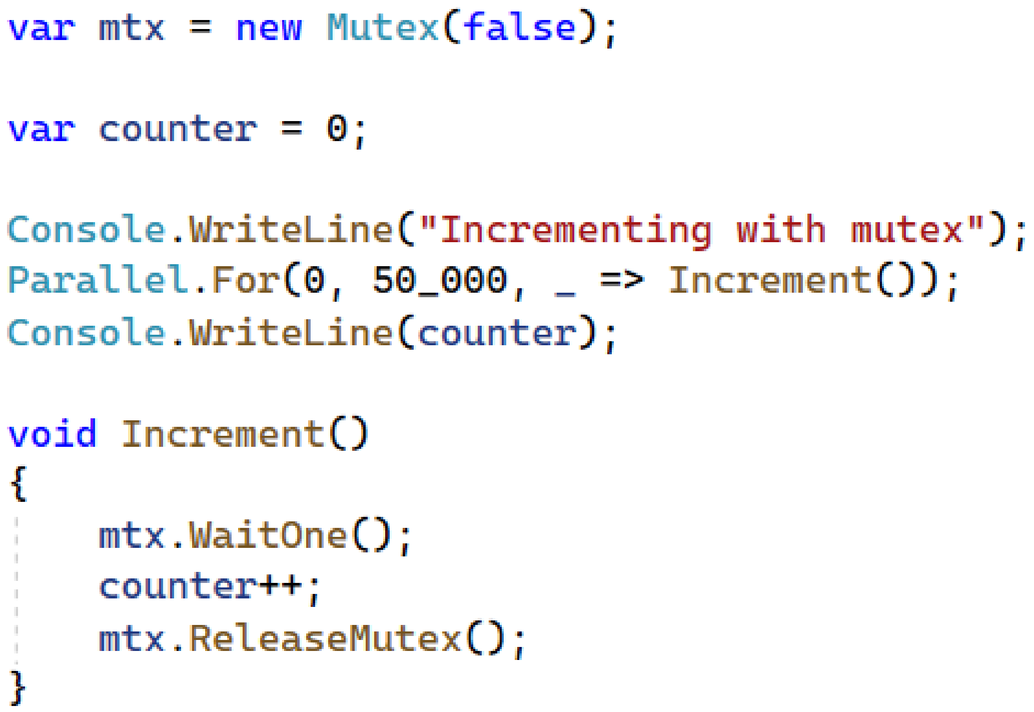


```
object syncObj = new object();
var counter = 0;

Console.WriteLine("Incrementing with lock");
Parallel.For(0, 50_000, _ => Increment());
Console.WriteLine(counter);

void Increment()
{
    lock (syncObj)
    {
        counter++;
    }
}
```

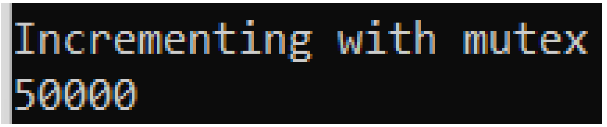


```
Incrementing with lock
50000
```

Threads synchronization - Mutex

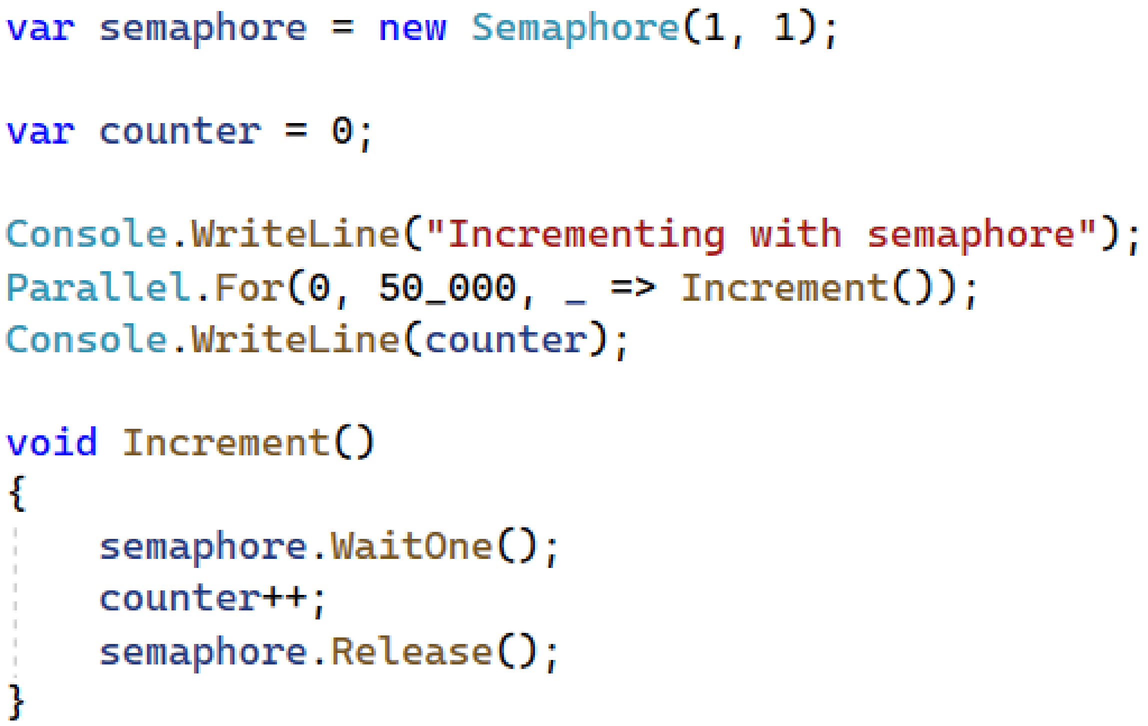


```
var mtx = new Mutex(false);

var counter = 0;

Console.WriteLine("Incrementing with mutex");
Parallel.For(0, 50_000, _ => Increment());
Console.WriteLine(counter);

void Increment()
{
    mtx.WaitOne();
    counter++;
    mtx.ReleaseMutex();
}
```

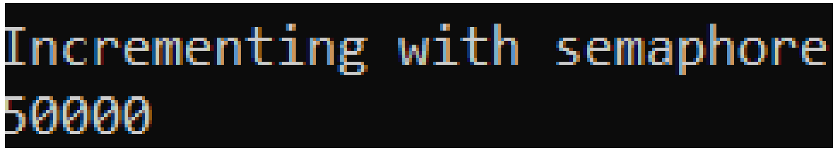


```
Incrementing with mutex
50000
```

Threads synchronization - Semaphore

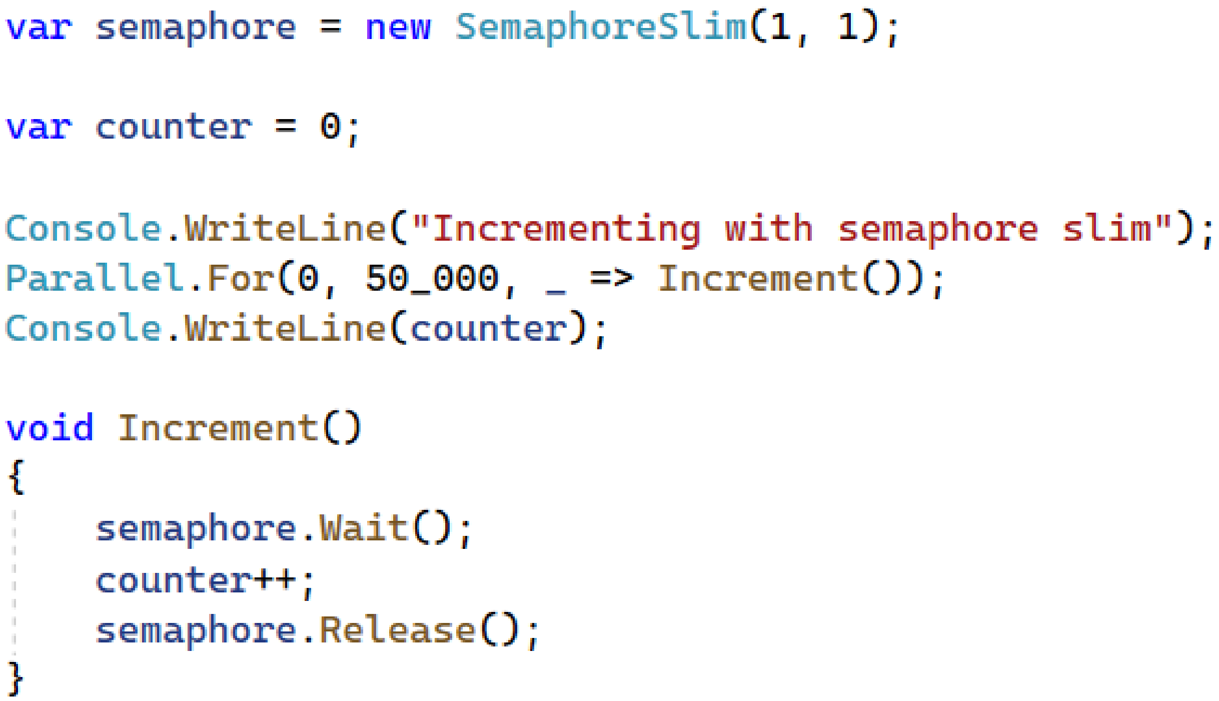


```
var semaphore = new Semaphore(1, 1);

var counter = 0;

Console.WriteLine("Incrementing with semaphore");
Parallel.For(0, 50_000, _ => Increment());
Console.WriteLine(counter);

void Increment()
{
    semaphore.WaitOne();
    counter++;
    semaphore.Release();
}
```

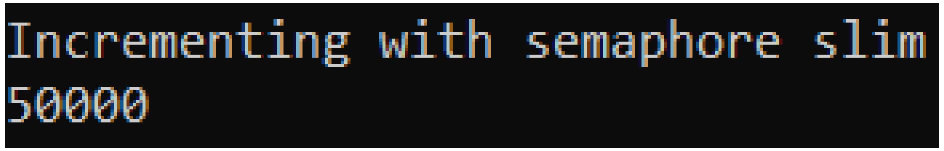


```
Incrementing with semaphore
50000
```

Threads synchronization - SemaphoreSlim



```
var semaphore = new SemaphoreSlim(1, 1);

var counter = 0;

Console.WriteLine("Incrementing with semaphore slim");
Parallel.For(0, 50_000, _ => Increment());
Console.WriteLine(counter);

void Increment()
{
    semaphore.Wait();
    counter++;
    semaphore.Release();
}
```



```
Incrementing with semaphore slim
50000
```

Books of the day

Cleary S. – Concurrency In C#

Terrell R. – Concurrency In .NET: Modern patterns of concurrent and parallel programming

Richter J. – CLR via C# 4th ed.

Links of the day

Albahari J. – Threading in C#

Task-based Asynchronous Programming model – TAP (MSDN)

Stephen Cleary (the blog)

ConfigureAwait Tips (Stephen’s Toub article translation on Habr)

Hometask

1. Create console application which would be able to calculate square roots for the numbers in range from M to N, where M and N – arguments what can be passed by the user. Print the pairs “number, square root” in the ascending order.

2. Create console application which would concatenate the content of all text files from the directory. Directory should be passed via console application argument. All text files from that directory should be copied in one one and saved in the same directory. Use asynchronous API (example).

That’s all for this time!