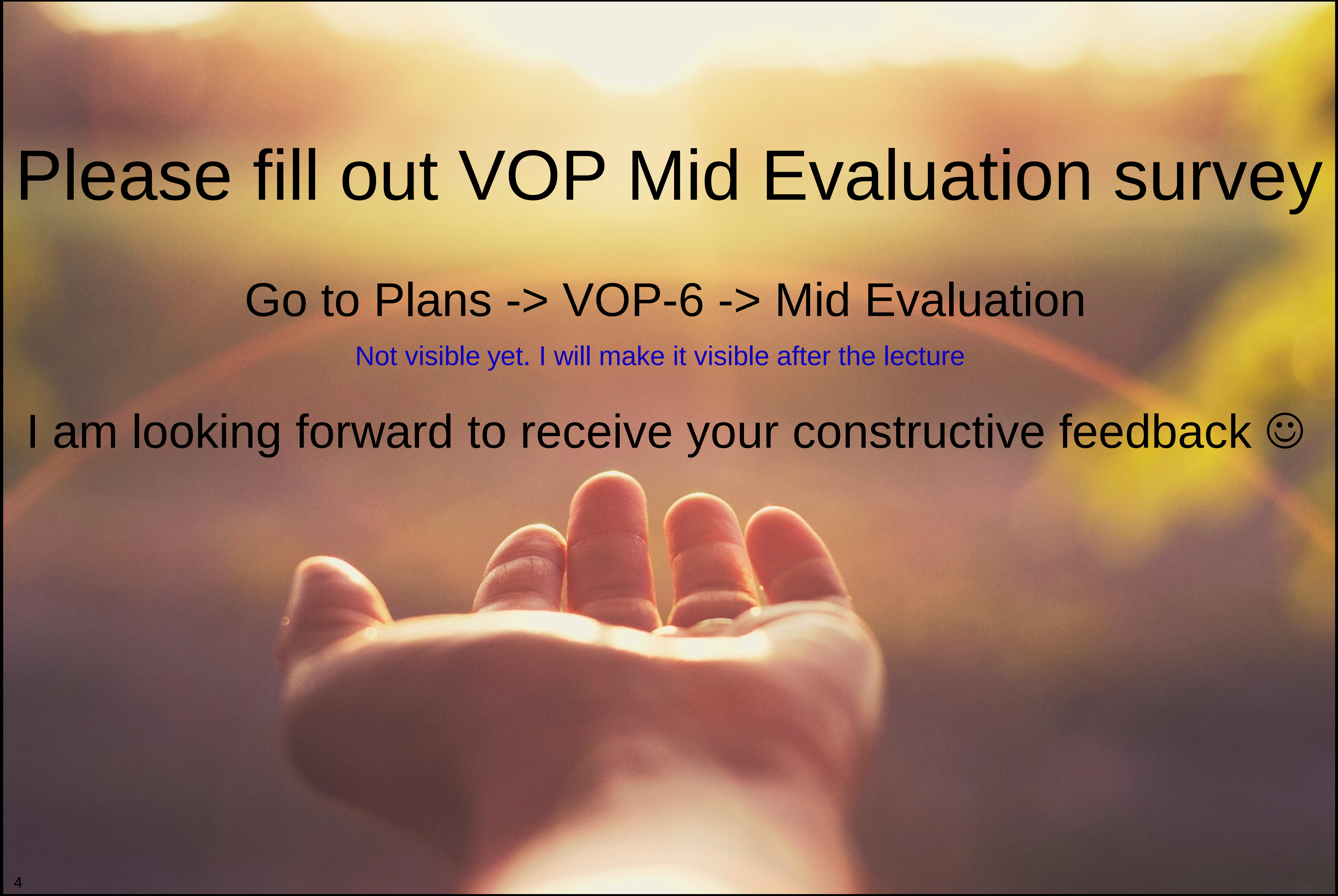
THE MAERSK MC-KINNEY MOLLER INSTITUTE

SDU

Lesson 6: Threads and Tasks

Agenda

- *Introduction to Multithreading*
  - ✓ *Single vs Multithreaded Model*
- *Ways to Implement Multithreading*
  - ✓ *Thread Class*
  - ✓ *ThreadStart Delegate*
  - ✓ *Task Class*
  - ✓ *Async and Await*



Please fill out VOP Mid Evaluation survey

Go to Plans -> VOP-6 -> Mid Evaluation

Not visible yet. I will make it visible after the lecture

I am looking forward to receive your constructive feedback ☺

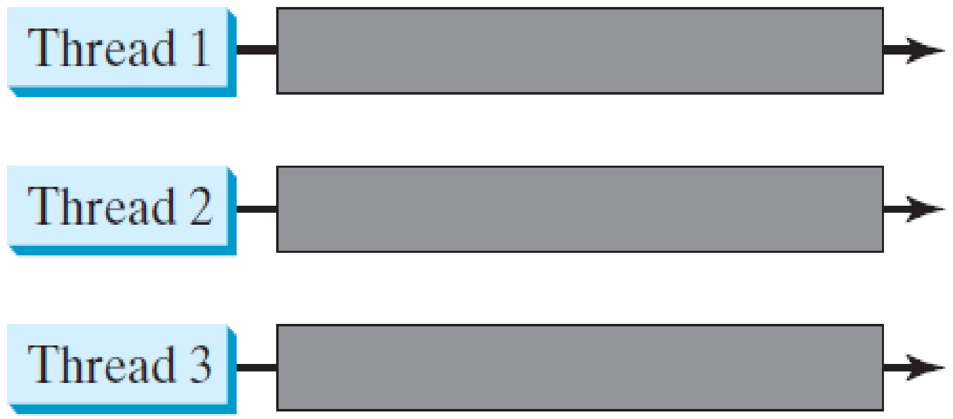
MultiThreading in C#

The slides from this lecture are taken from:
https://learn.microsoft.com/en-us/dotnet/api/system.threading.thread?view=net-8.0
https://www.geeksforgeeks.org/c-sharp-multithreading/
https://code-maze.com/csharp-async-vs-multithreading/

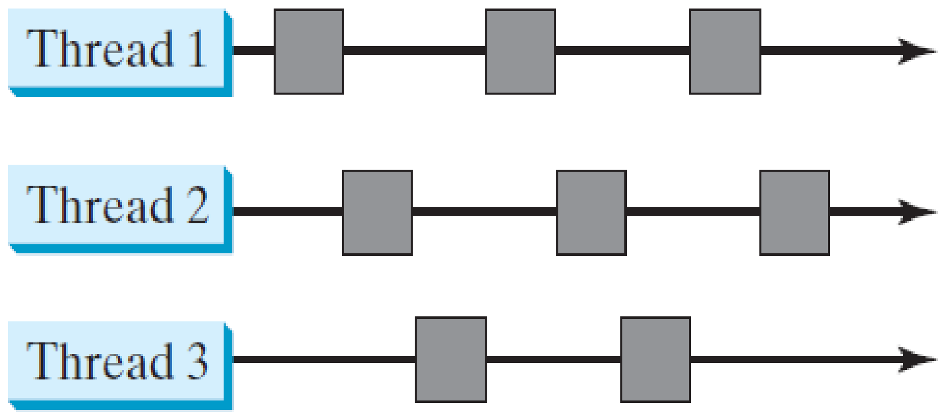
Threads

- Multithreading enables multiple tasks in a program to be executed concurrently.
- In a multi-processor system, threads are executed concurrently
- In a single-processor system, multiple threads share CPU time. This is called time sharing
- In a single-processor system, the OS is responsible for scheduling and allocating resources.

Multiple threads on multiple CPUs

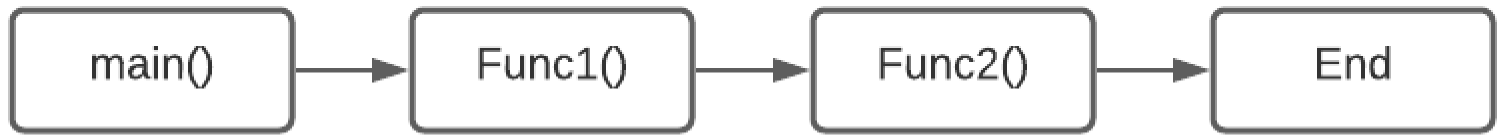


Multiple threads sharing a single CPU

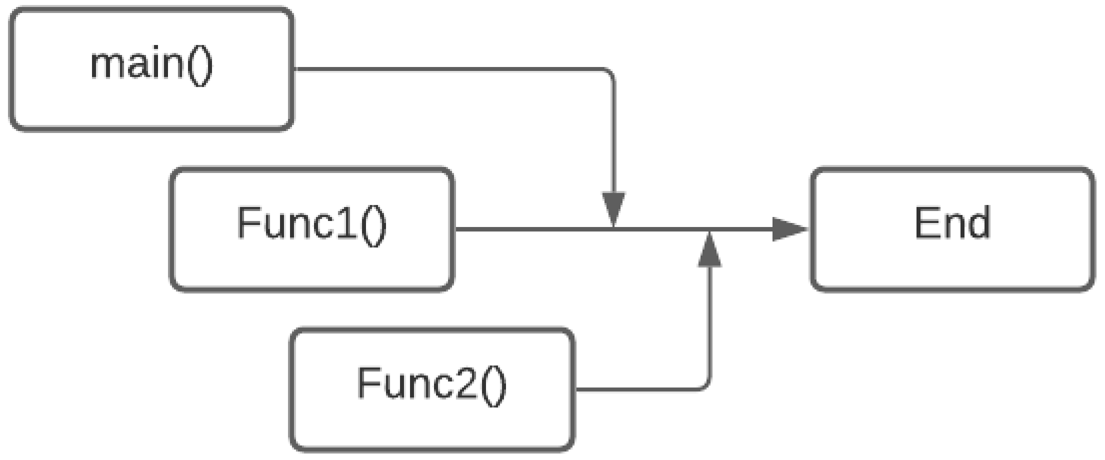


Flow of Control in C#

<u>Without MultiThreading</u>



<u>With MultiThreading</u>



Ways to implement MultiThreading

In C#, multithreading is supported through the **System.Threading** namespace.

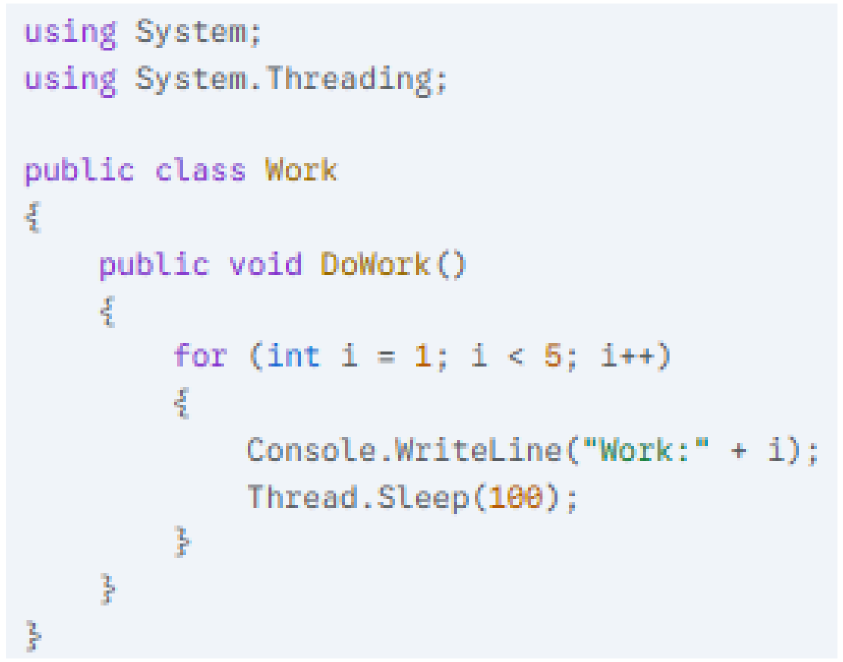
1. Thread Class
2. ThreadStart Delegate
3. Task Class
4. Async and Await

The slides from this lecture are taken from:
https://learn.microsoft.com/en-us/dotnet/api/system.threading.thread?view=net-8.0
https://www.geeksforgeeks.org/c-sharp-multithreading/

Creating and running Threads: Using Thread Class

- The Thread class is the most basic way to implement multithreading in C#.
- We can create a thread by instantiating a Thread object and passing a method that represents the task to be executed
- The Start() method of the Thread object tells the CLR that the thread is ready to run.

Method creation: Work.cs

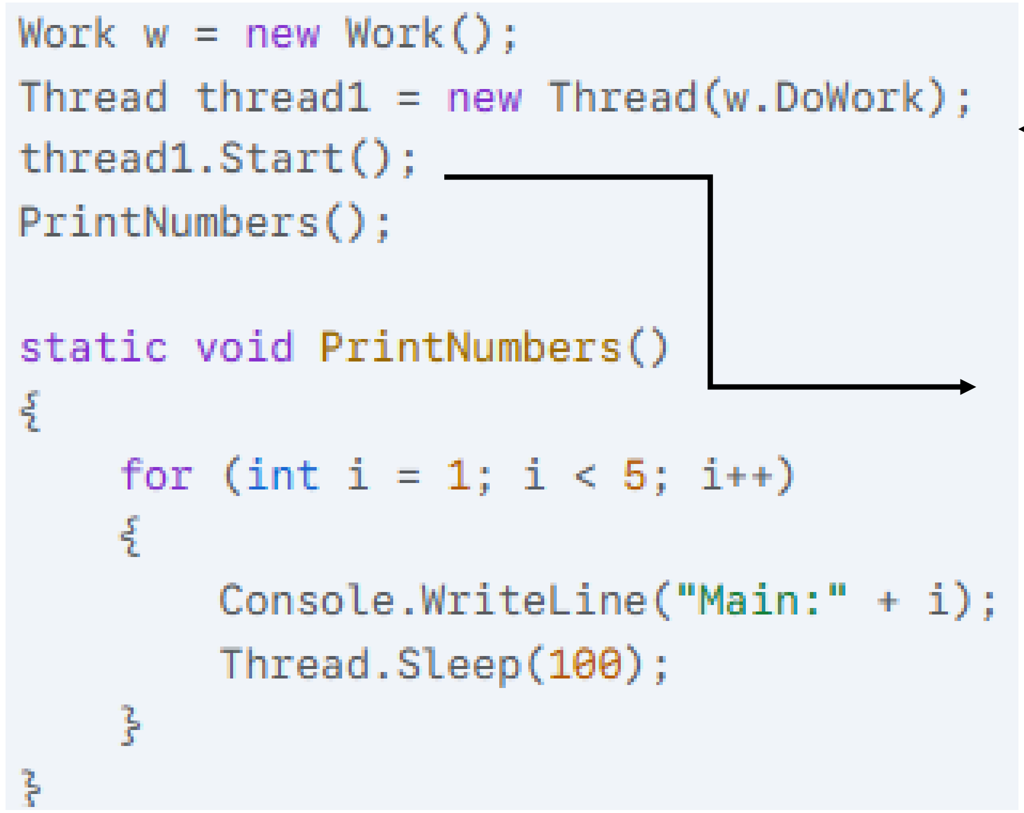


```csharp
using System;
using System.Threading;

public class Work
{
    public void DoWork()
    {
        for (int i = 1; i < 5; i++)
        {
            Console.WriteLine("Work:" + i);
            Thread.Sleep(100);
        }
    }
}
```

Program.cs

Creation & Execution of Thread

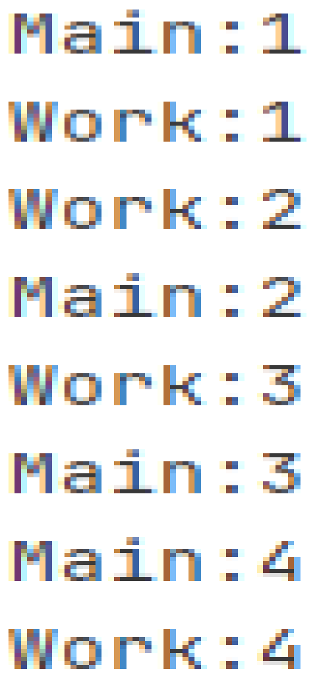


```csharp
Work w = new Work();
Thread thread1 = new Thread(w.DoWork);
thread1.Start();
PrintNumbers();

static void PrintNumbers()
{
    for (int i = 1; i < 5; i++)
    {
        Console.WriteLine("Main:" + i);
        Thread.Sleep(100);
    }
}
```

- A *foreground* thread by default.
- Runs independently of the main thread.

UsingThreadClass (MultiThreadApp): Output



```
Main:1
Work:1
Work:2
Main:2
Work:3
Main:3
Main:4
Work:4
```

$\longrightarrow$ Interleaved execution of Main and Work threads

$\longrightarrow$ Main thread finished, but Work thread continued

Thread Demo: Example

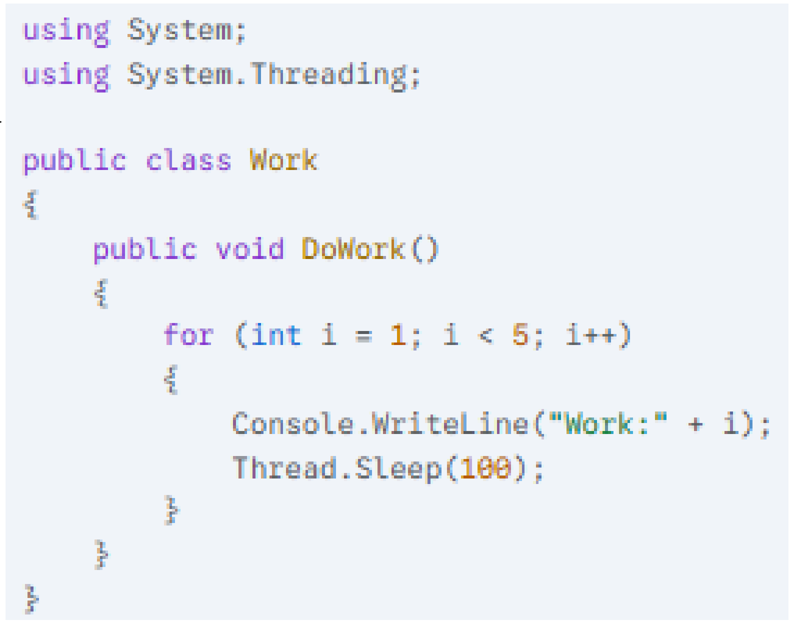
From Plans -> VOP-6 -> VOP-6 (Lecture) -> Resources and Activities -> MultiThreadApp.zip -> ThreadDemo

- ThreadDemo.PrintChar.cs
- ThreadDemo.PrintNum.cs
- ThreadDemo.Program.cs

Creating and running Threads: Using ThreadStart

- Methods are created using the `ThreadStart` delegate.
- The `ThreadStart` delegate contains the constructor, which takes a method that tells the system what your thread is going to run.
- The `Thread` Class contains a constructor that takes a `ThreadStart` delegate as a parameter and runs it as a thread.
- The `Start()` method of the `Thread` object tells the CLR that the thread is ready to run.

Method creation: Work.cs

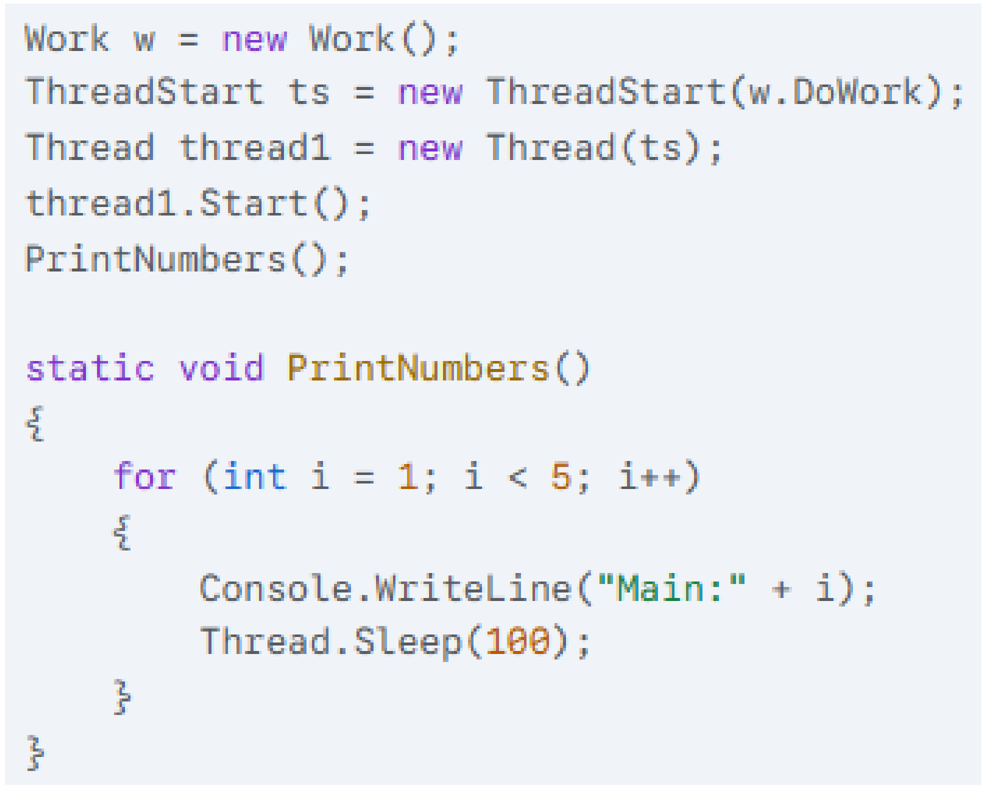


```
using System;
using System.Threading;

public class Work
{
    public void DoWork()
    {
        for (int i = 1; i < 5; i++)
        {
            Console.WriteLine("Work:" + i);
            Thread.Sleep(100);
        }
    }
}
```

Program.cs

Creation & Execution of Thread



```
Work w = new Work();
ThreadStart ts = new ThreadStart(w.DoWork);
Thread thread1 = new Thread(ts);
thread1.Start();
PrintNumbers();

static void PrintNumbers()
{
    for (int i = 1; i < 5; i++)
    {
        Console.WriteLine("Main:" + i);
        Thread.Sleep(100);
    }
}
```

SDU

Running a thread is simple in C# - follow these steps:

1. Write a class that has a method:

```
public class MyThread
{
        public void Run()
        {
                code statements
                . . .
        }
}
```

2. Create an object of your class:

```
MyThread mt = new MyThread();
```

3. Construct a ThreadStart delegate from your class with the method you want to execute:

```
ThreadStart ts = new ThreadStart(mt.Run);
```

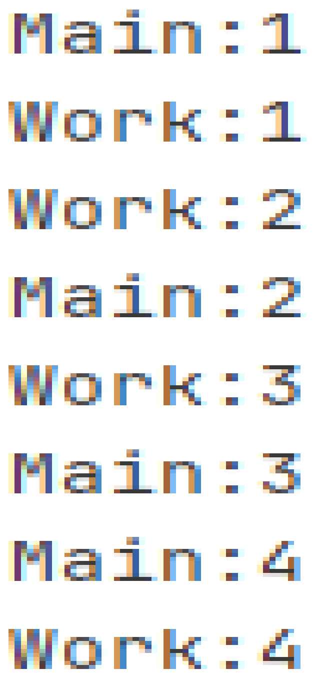
4. Construct a Thread object from the ThreadStart delegate:

```
Thread t = new Thread(ts);
```

5. Call the Start method to start the thread:

```
t.Start();
```

UsingThreadStart (MultiThreadApp) : Output



```
Main:1
Work:1
Work:2
Main:2
Work:3
Main:3
Main:4
Work:4
```

ThreadStartDemo: Example

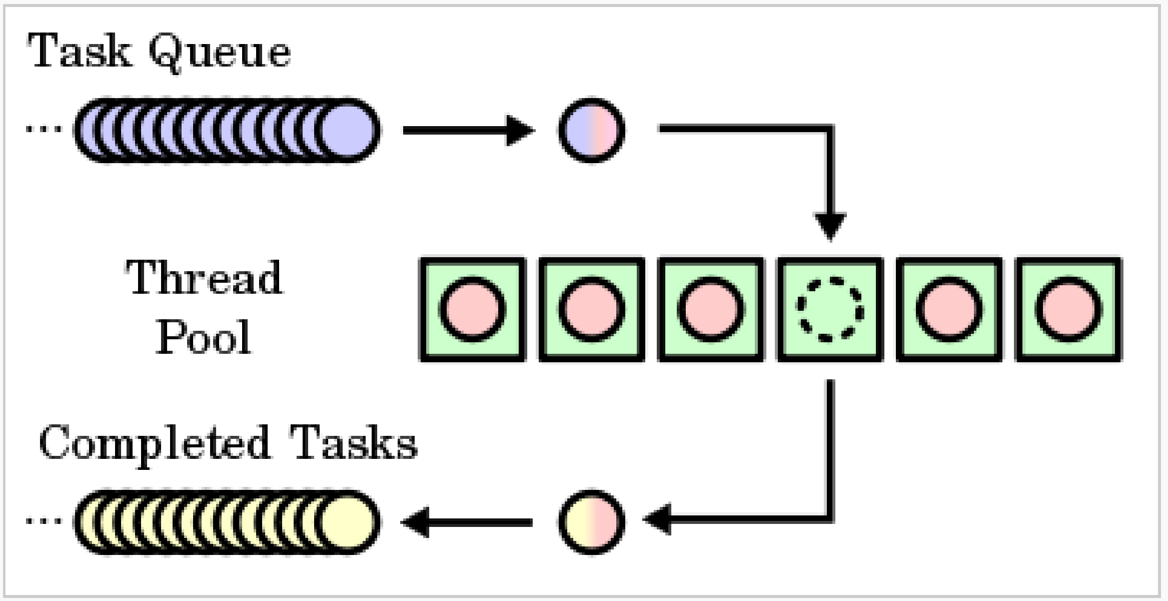
From Plans -> VOP-6 -> VOP-6 (Lecture) -> Resources and Activities -> MultiThreadApp.zip -> ThreadStartDemo

- ThreadStartDemo.PrintChar.cs
- ThreadStartDemo.PrintNum.cs
- ThreadStartDemo.Program.cs

Creating and running Tasks: Using Task Class

- Tasks in C# are objects that represent **asynchronous** operations, i.e., some work that should be done.
- They provide a higher-level abstraction than traditional threads.
- When you use Task.Run(), the task is typically executed on a thread pool thread.
- Thread pool threads are **background** threads.

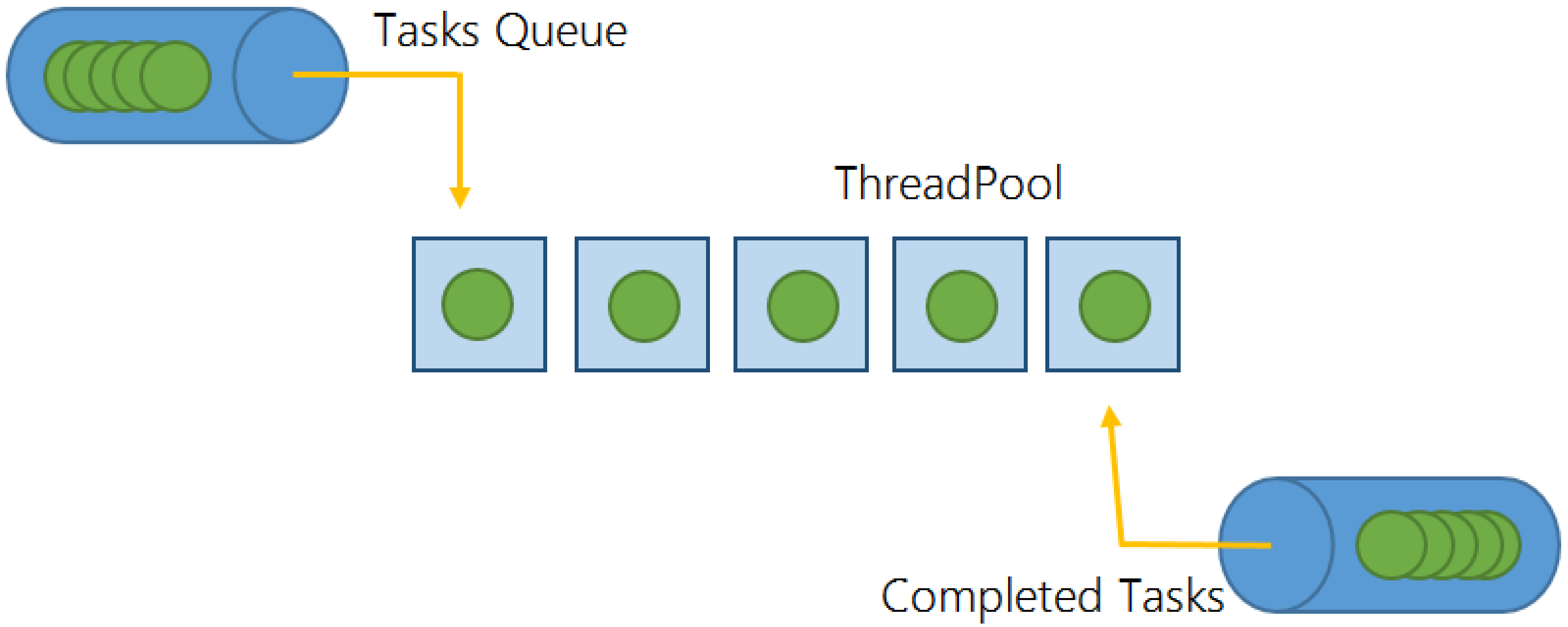
Tied to the Main Thread



A sample thread pool (green boxes) with waiting tasks (blue) and completed tasks (yellow)

ThreadPool

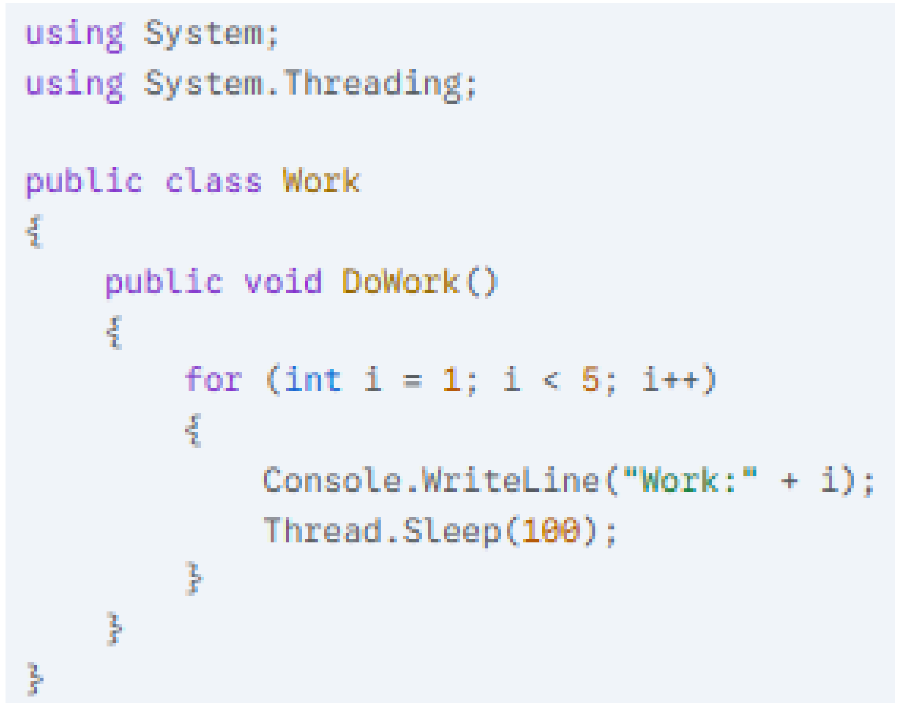
- The previous approach is not efficient for a large number of tasks because you have to create a thread for each task.
- A thread pool is ideal for managing the number of tasks executing concurrently
- C# provides the ThreadPool for executing tasks in a thread pool and for managing the tasks respectively



https://programmium.wordpress.com/2016/04/02/c-understanding-threads-and-threadpools/

Example: Using Task class

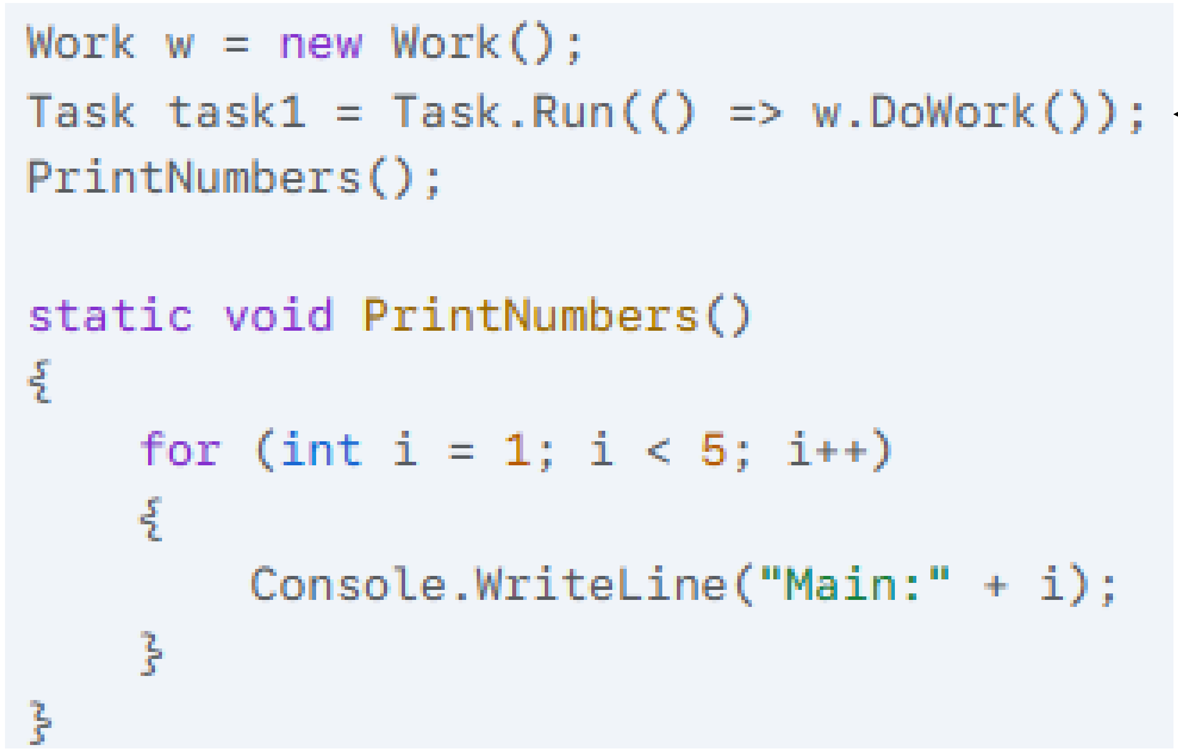
Method creation: Work.cs



```csharp
using System;
using System.Threading;

public class Work
{
    public void DoWork()
    {
        for (int i = 1; i < 5; i++)
        {
            Console.WriteLine("Work:" + i);
            Thread.Sleep(100);
        }
    }
}
```

Program.cs

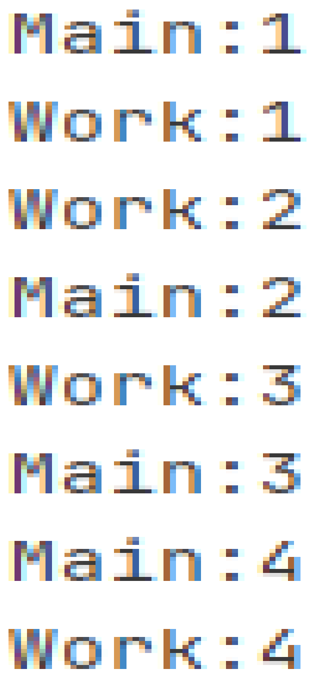


```csharp
Work w = new Work();
Task task1 = Task.Run(() => w.DoWork());
PrintNumbers();

static void PrintNumbers()
{
    for (int i = 1; i < 5; i++)
    {
        Console.WriteLine("Main:" + i);
    }
}
```

Creation & Execution of Task

UsingTaskClass (MultiThreadApp) : Output



```
Main:1
Work:1
Work:2
Main:2
Work:3
Main:3
Main:4
Work:4
```

TaskDemo: Example

From Plans -> VOP-6 -> VOP-6 (Lecture) -> Resources and Activities -> MultiThreadApp.zip -> TaskDemo

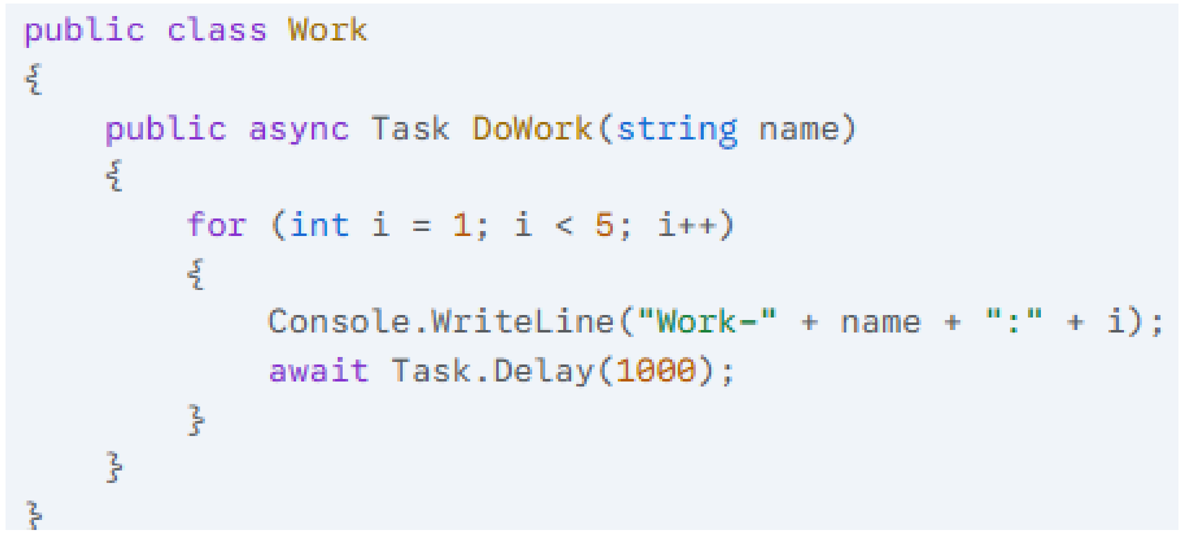
- TaskDemo.PrintChar.cs
- TaskDemo.PrintNum.cs
- TaskDemo.Program.cs

MultiTasking: Using async & await

- Asynchronous implementation is easy in a task by using:
- `async` and `await` keywords
- `async`: Declares a method as asynchronous, allowing it to use the await keyword.
- `await`: Suspends the execution of an asynchronous method until the awaited task completes, without blocking the calling thread

Example: Using async and await

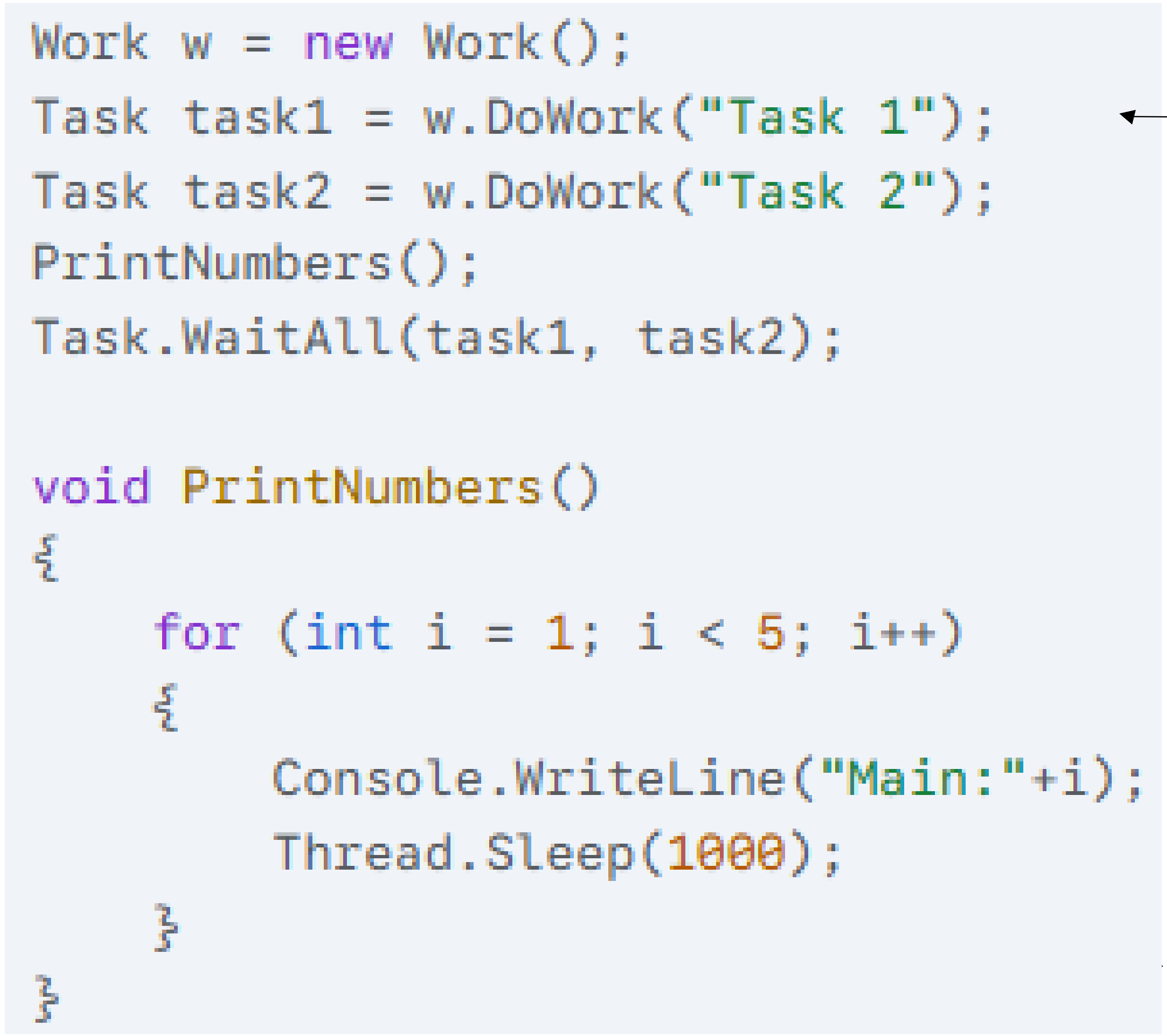
Async method creation: Work.cs



```
public class Work
{
    public async Task DoWork(string name)
    {
        for (int i = 1; i < 5; i++)
        {
            Console.WriteLine("Work-" + name + ":" + i);
            await Task.Delay(1000);
        }
    }
}
```

Program.cs

Execution of Async Tasks



```
Work w = new Work();
Task task1 = w.DoWork("Task 1");
Task task2 = w.DoWork("Task 2");
PrintNumbers();
Task.WaitAll(task1, task2);

void PrintNumbers()
{
    for (int i = 1; i < 5; i++)
    {
        Console.WriteLine("Main:"+i);
        Thread.Sleep(1000);
    }
}
```

UsingAsyncAwait (MultiThreadApp): Example

From Plans -> VOP-6 -> VOP-6 (Lecture) -> Resources and Activities -> MultiThreadApp.zip -> UsingAsyncAwait

- UsingAsyncAwait.Work.cs
- UsingAsyncAwait.Program.cs

MCQs Quiz

Go to Plans -> VOP-6 -> VOP-6 (Lecture) -> Lecture-6 Test

Good Luck ☺