Multi-threading and Parallel processing

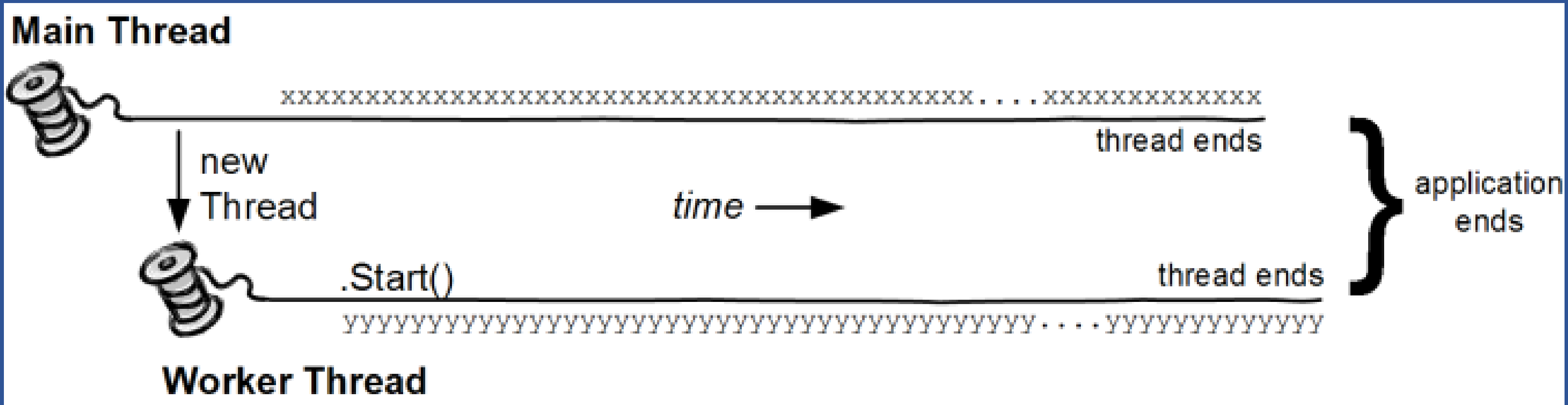
Multi-threading and Parallel processing Classes

- Thread
- Thread pool
- BackgroundWorker
- Task
- Mutex

- Starting with the .NET Framework 4, the recommended way to utilize multithreading is to use Task Parallel Library (TPL) and Parallel LINQ (PLINQ).
- Both TPL and PLINQ rely on the ThreadPool threads.

Thread

- A class in System.Threading.Thread
- Operations can execute on separate threads
- Known as multithreading or free threading
- Useful when;
  - More responsive to user input
  - Create scalable applications (add threads as the workload increases)


Main Thread
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx....xxxxxxxxxxxxxx
thread ends
new Thread
time
.Start()
thread ends
application ends
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy....yyyyyyyyyyyyyy
Worker Thread

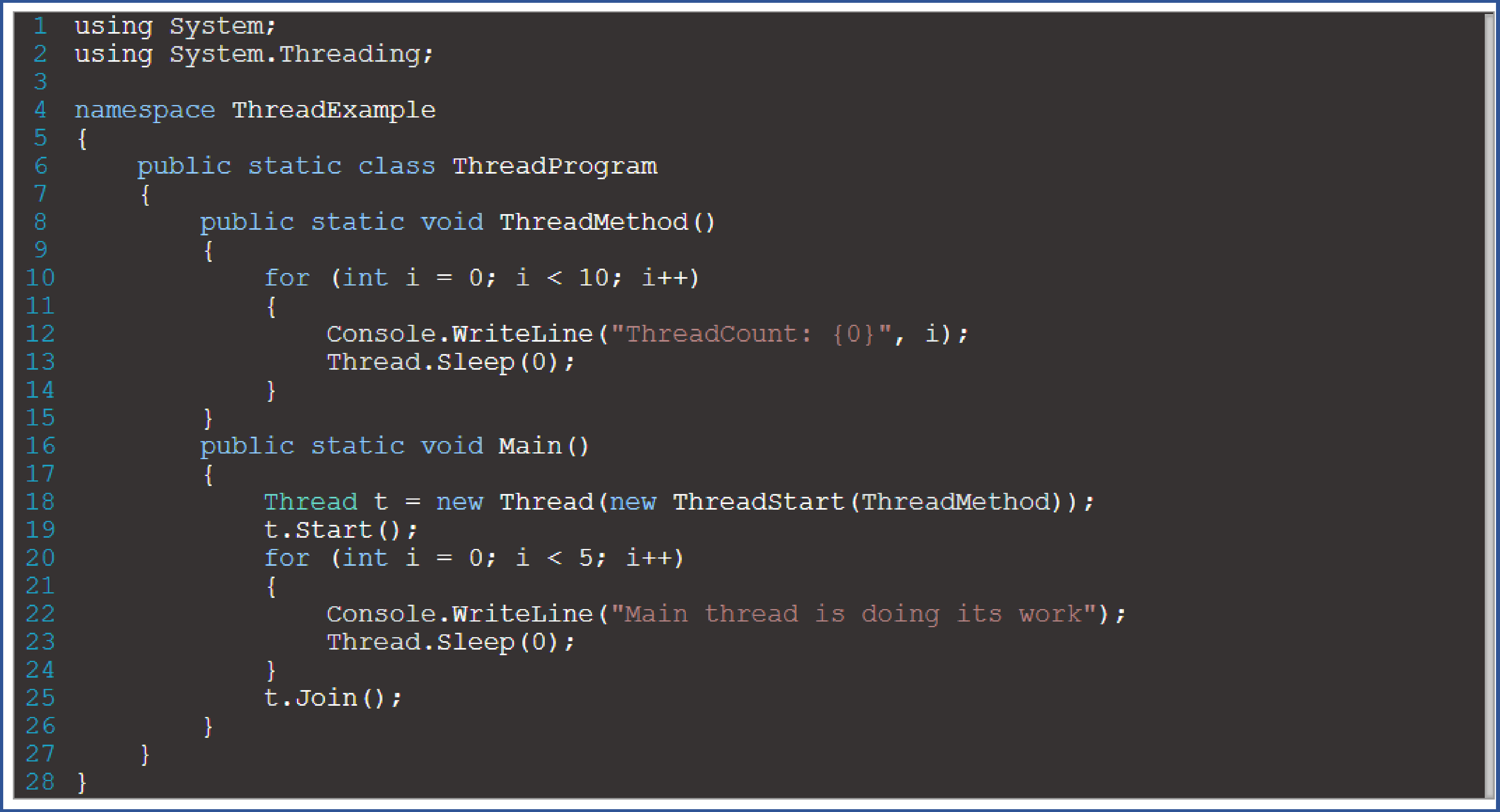
Properties of Thread

Property	Description
IsAlive	ReturnsTrue when the thread is started but not stopped
IsBackground	Returns whether the Thread is a Background Thread or not
Priority	Determines threads priority, i.e. highest,Normal,Lowest etc..
ThreadState	Returns the threads state, i.e. Aborted,Running,Stopped, Suspended,Background etc.

Methods in Thread Class

Methods	Purpose
Abort	To Terminate/stop the Thread
Join	It is called on the main thread to let it wait until the other thread finishes.
ResetAbort	Cancels an Abort for current thread
Sleep	Suspends Thread for specified amount of time
Start	Starts the Thread
Yield	Yields execution to another thread if one is ready to run

How to Create a Thread

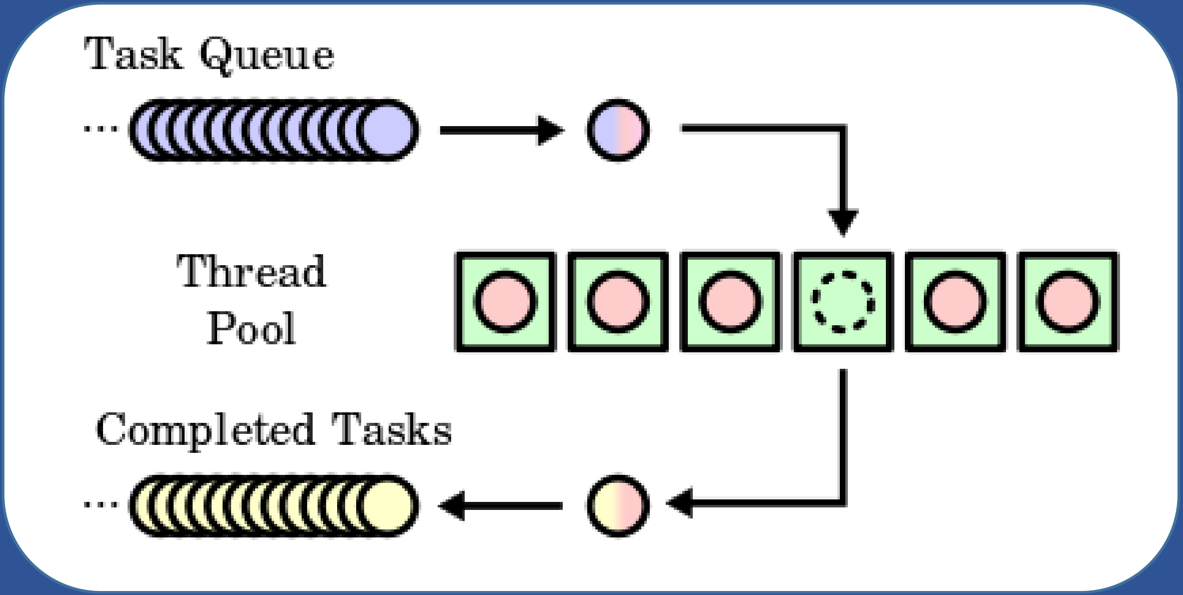


```
using System;
using System.Threading;

namespace ThreadExample
{
    public static class ThreadProgram
    {
        public static void ThreadMethod()
        {
            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine("ThreadCount: {0}", i);
                Thread.Sleep(0);
            }
        }
        public static void Main()
        {
            Thread t = new Thread(new ThreadStart(ThreadMethod));
            t.Start();
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine("Main thread is doing its work");
                Thread.Sleep(0);
            }
            t.Join();
        }
    }
}
```

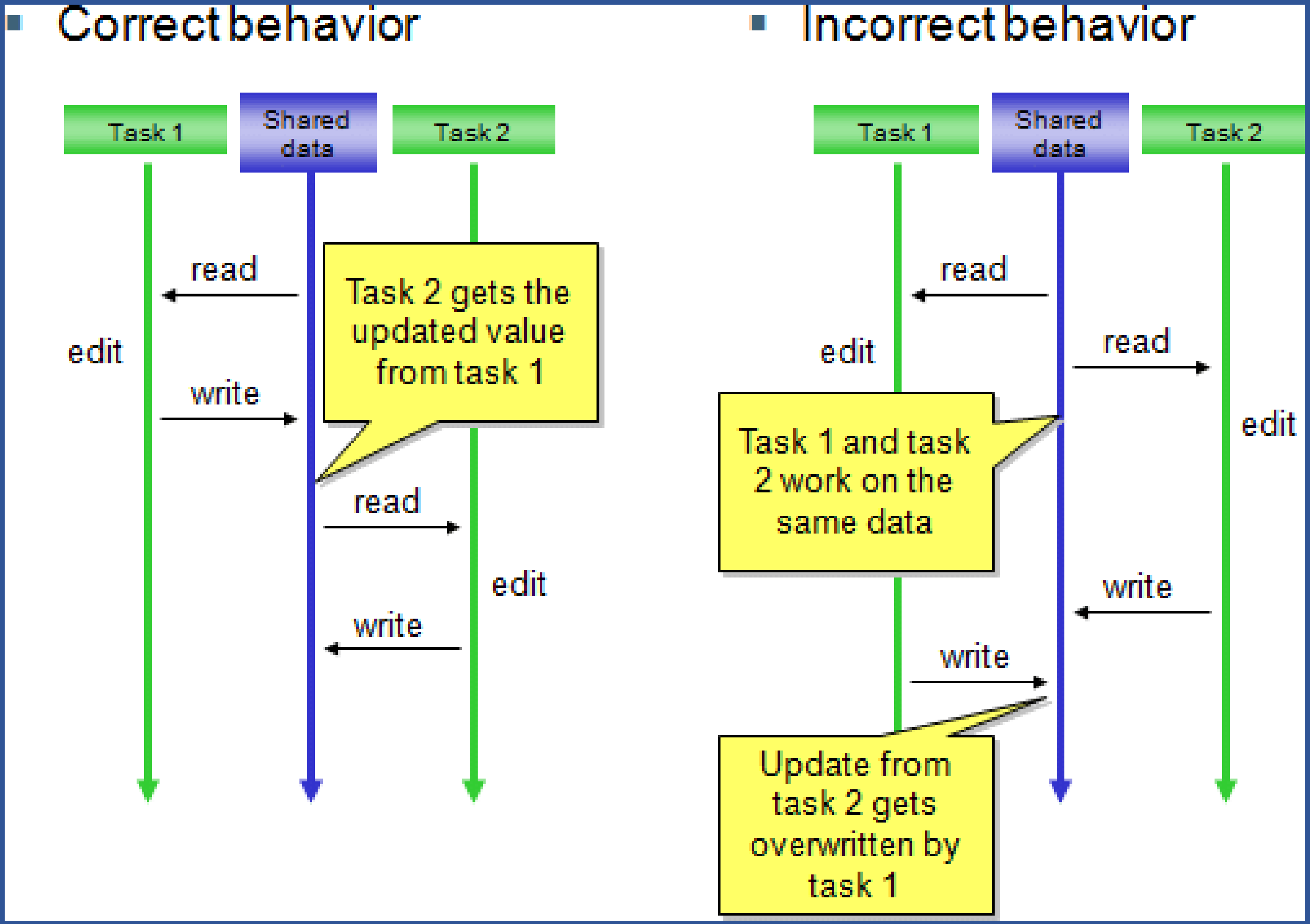
Thread pool

- A class in System.Threading.ThreadPool
- Design pattern for achieving concurrency of execution
- Also called a replicated workers or worker-crew model
- Maintains multiple threads
- waiting for tasks to be allocated for concurrent execution by the supervising program



Race conditions

Is a problem that can occur when a multithreaded program is not properly synchronized



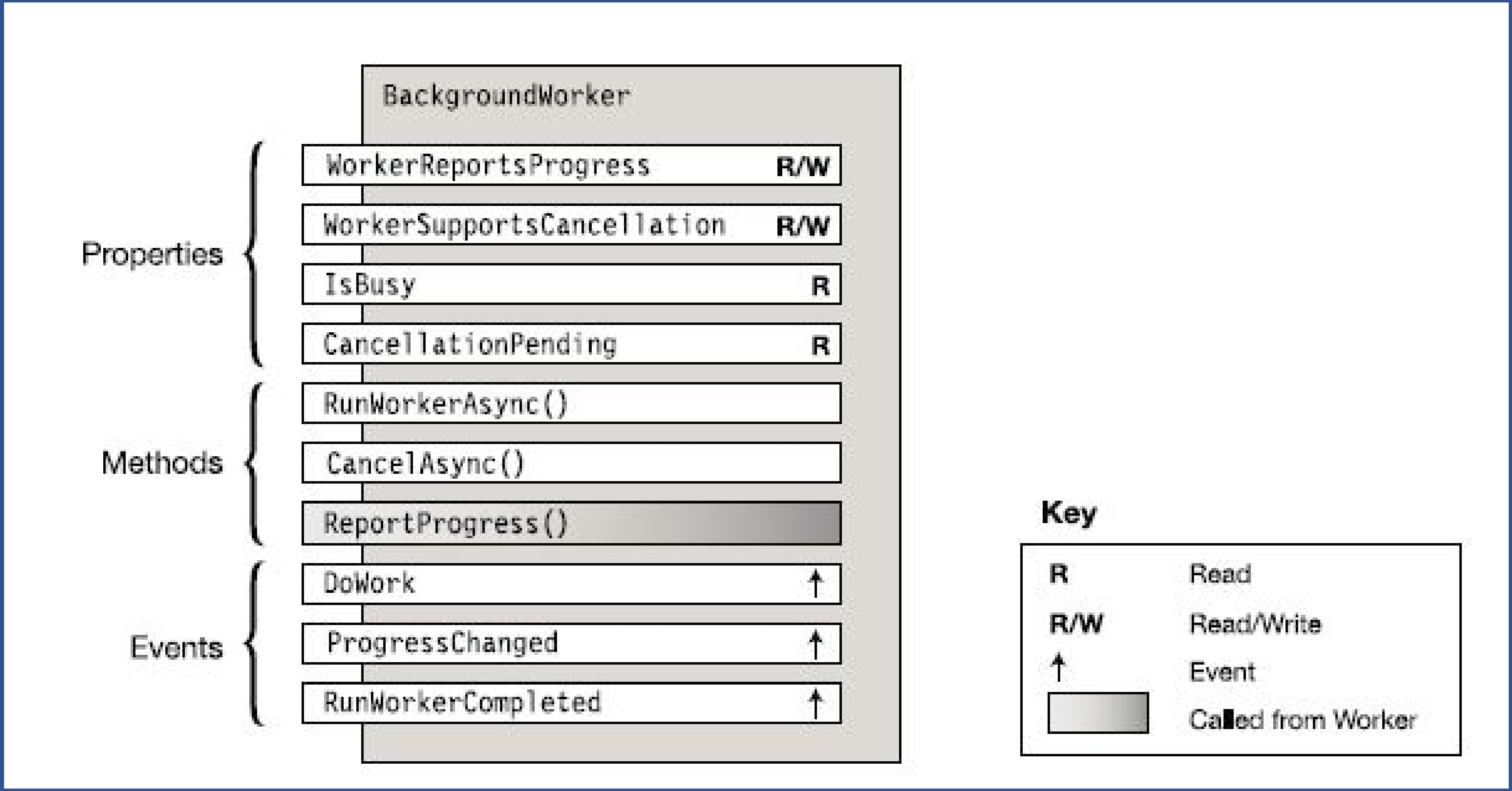
Avoiding race condition

Synchronization using;

- Thread.Join()
- Task.ContinueWith
- Lock
- Monitor Enter – Monitor Exit
- Mutex

Background worker

BackgroundWorker Class / Namespace: System.ComponentModel

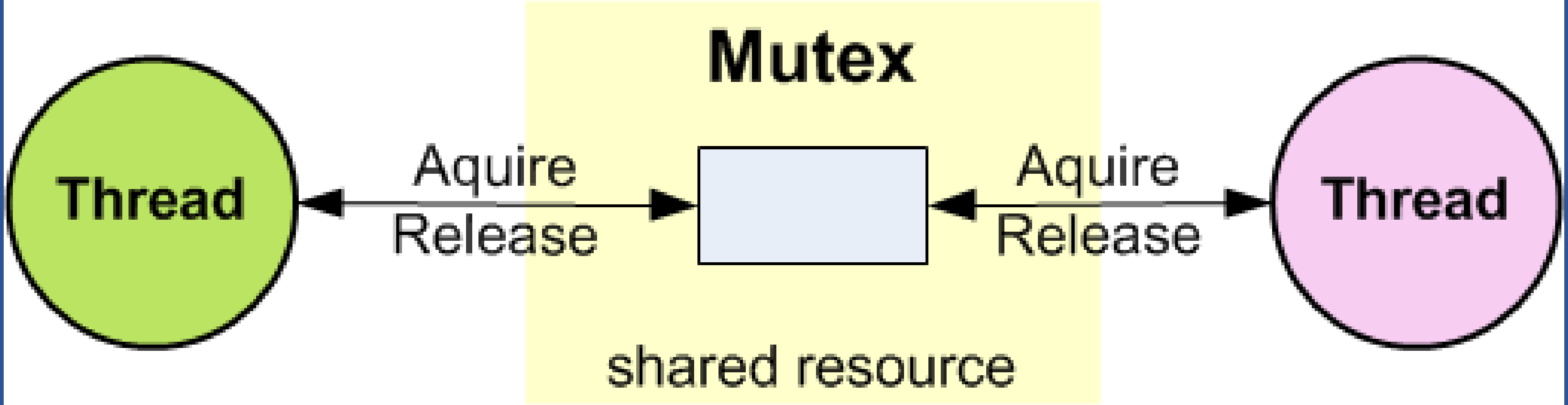


When to use BackgroundWorker?

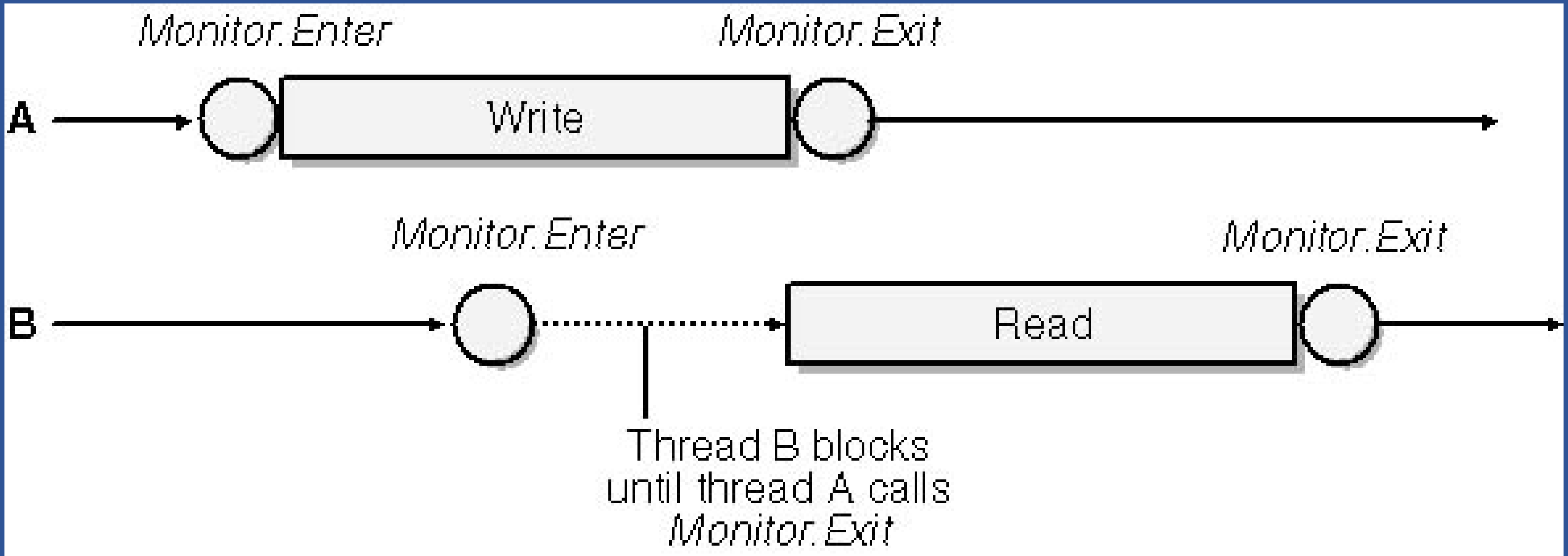
- Want to set up another thread
- Continuously running in the background
- Performing some work
- Occasionally communicating with the main thread

Mutex

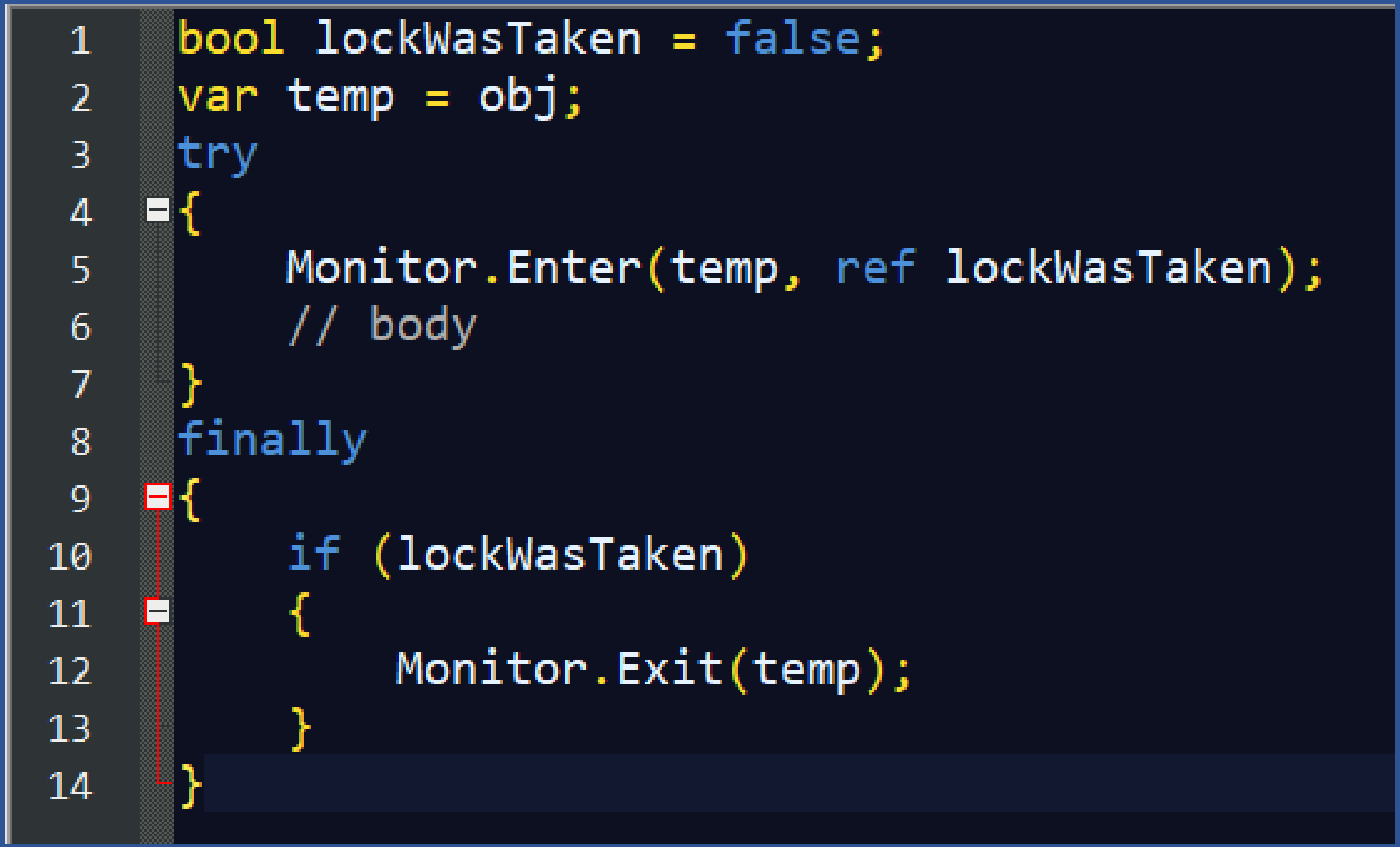
- A class in Namespace: System.Threading
- Manage multiple threads share the same resource



Monitor Enter – Monitor Exit



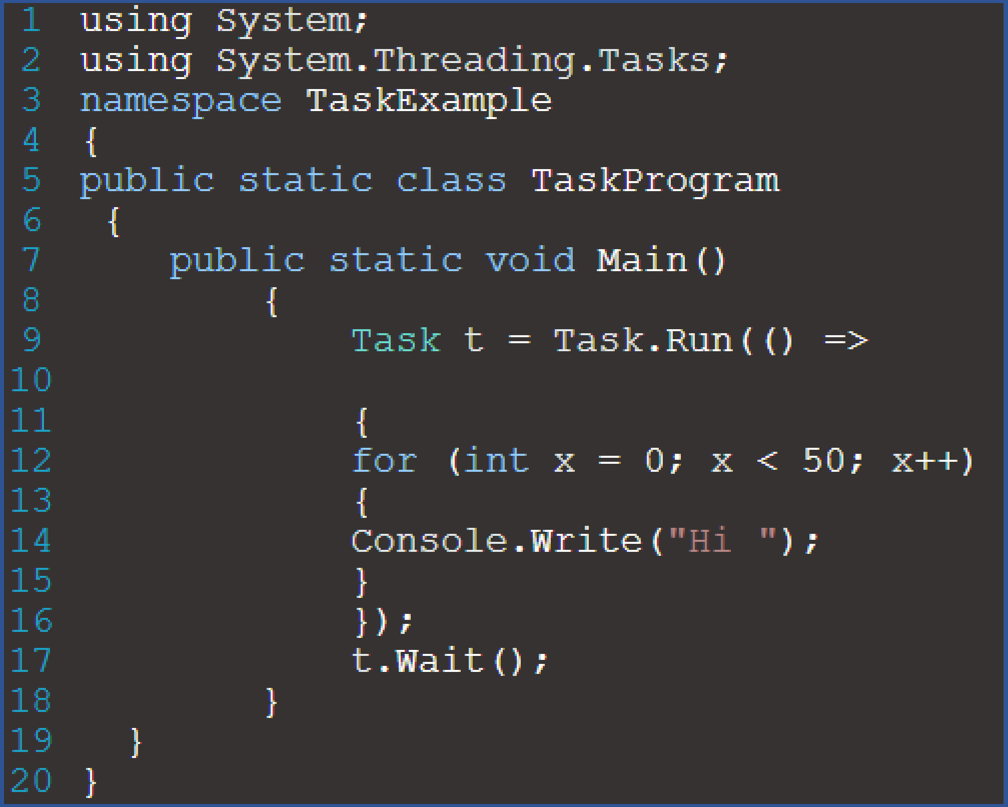
Monitor Enter – Monitor Exit



```
bool lockWasTaken = false;
var temp = obj;
try
{
    Monitor.Enter(temp, ref lockWasTaken);
    // body
}
finally
{
    if (lockWasTaken)
    {
        Monitor.Exit(temp);
    }
}
```

Task

- A class in System.Threading.Tasks.Task
- Provides parallel processing



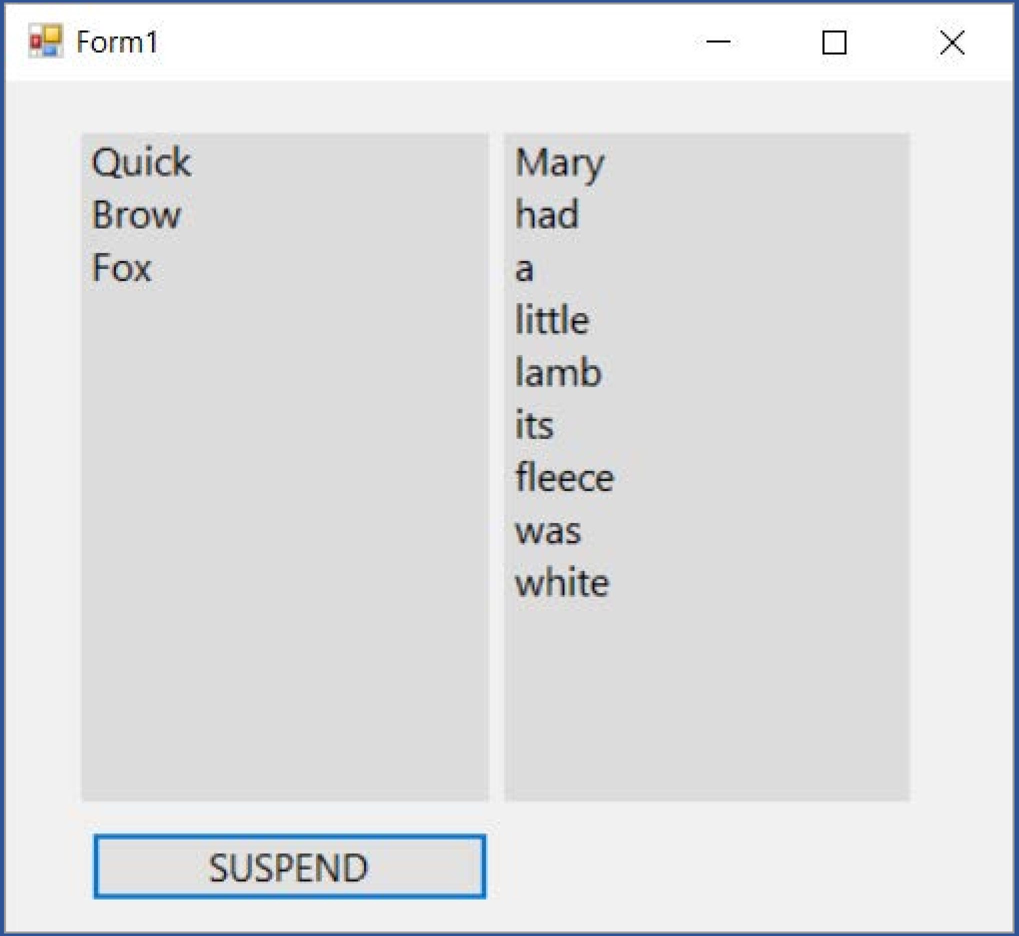
```
using System;
using System.Threading.Tasks;
namespace TaskExample
{
public static class TaskProgram
 {
    public static void Main()
        {
            Task t = Task.Run(() =>

            {
            for (int x = 0; x < 50; x++)
            {
            Console.Write("Hi ");
            }
            });
            t.Wait();
        }
  }
}
```

Task VS Thread

Thread	**Task**
Single or multiple processors	Guaranteed multiple processors
No Thread pool	Use thread pool internally
No return result	Have return result
Cannot be chained	Can be chained
No parent/child	Can have parent/child
Difficult to asynchronous	Easy asynchronous

Exercise 00100: Simple multi-thread. Suspend and resume

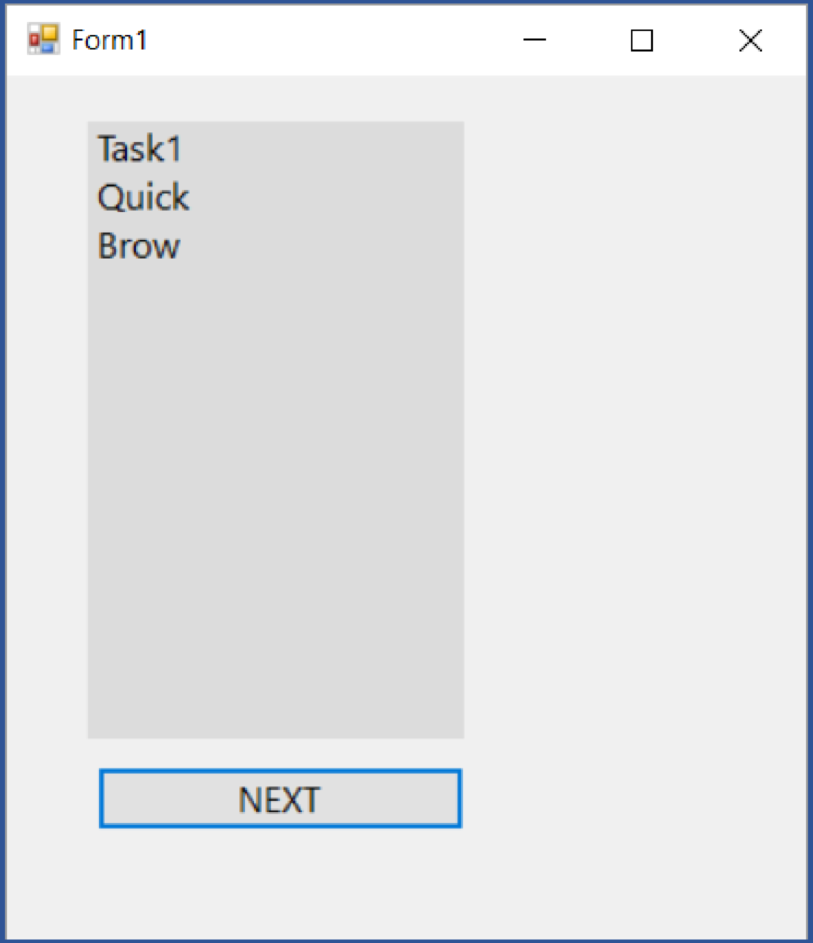


Quick
Brow
Fox

Mary
had
a
little
lamb
its
fleece
was
white

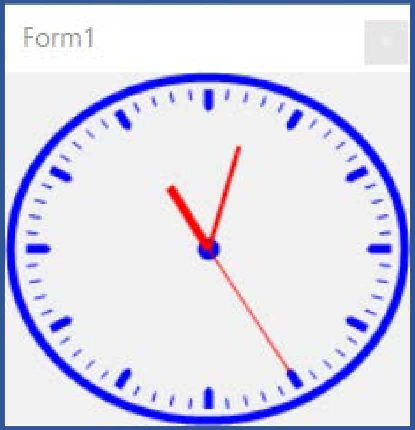
Exercise 00105: Thread and Queue

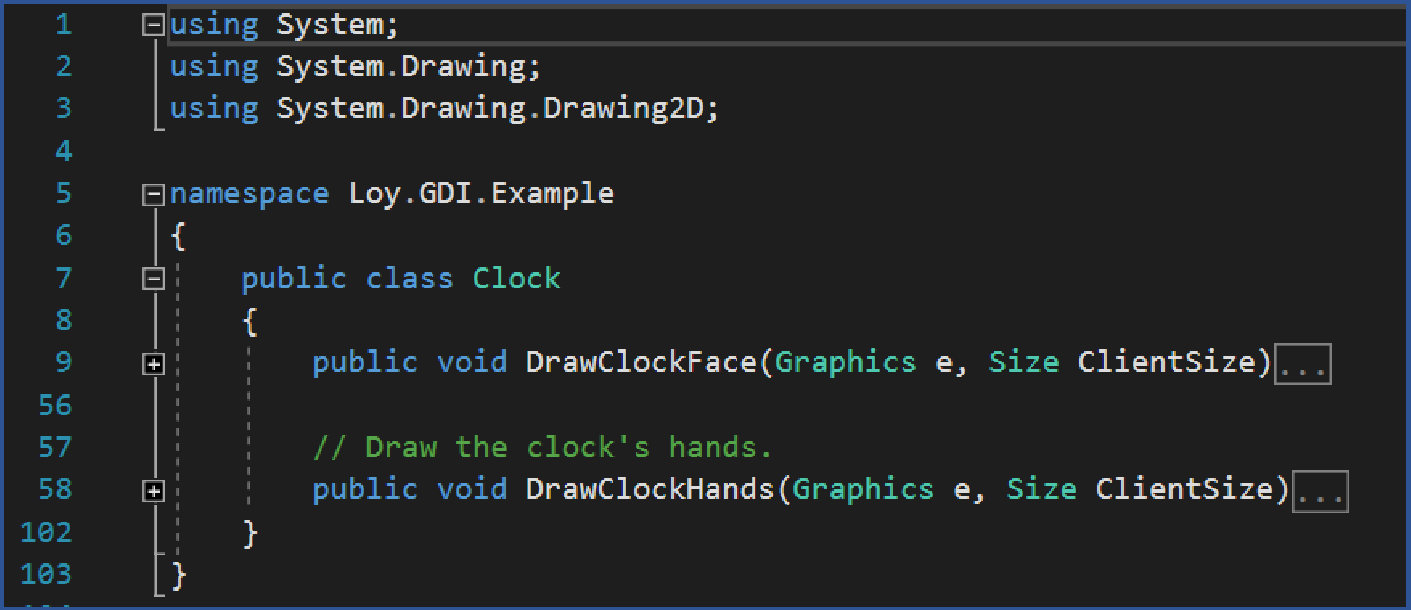
Put 2 thread in to Queue and dequque one when click at a button



Exercise 00110: Thread VS GDI+

Use thread to update clock hand





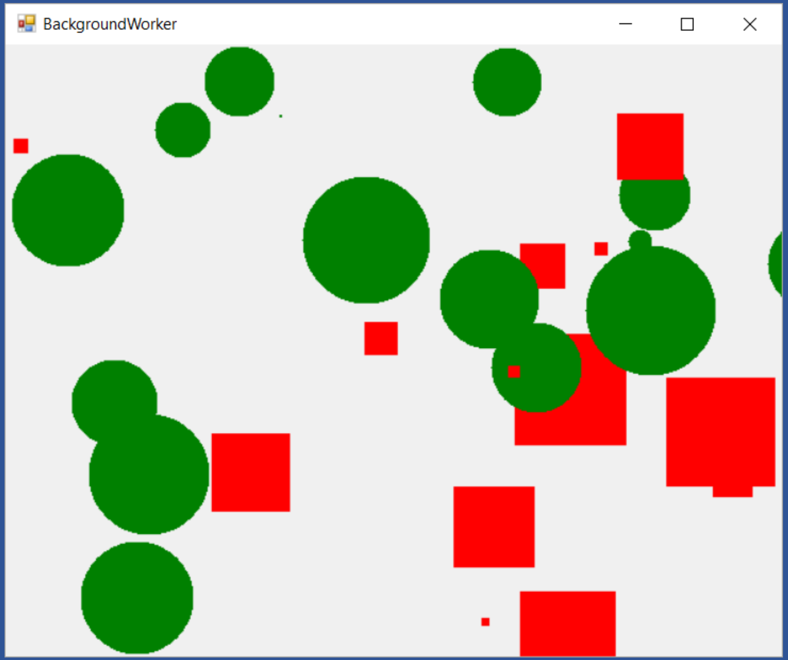
```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;

namespace Loy.GDI.Example
{
    public class Clock
    {
        public void DrawClockFace(Graphics e, Size ClientSize)...

        // Draw the clock's hands.
        public void DrawClockHands(Graphics e, Size ClientSize)...
    }
}
```

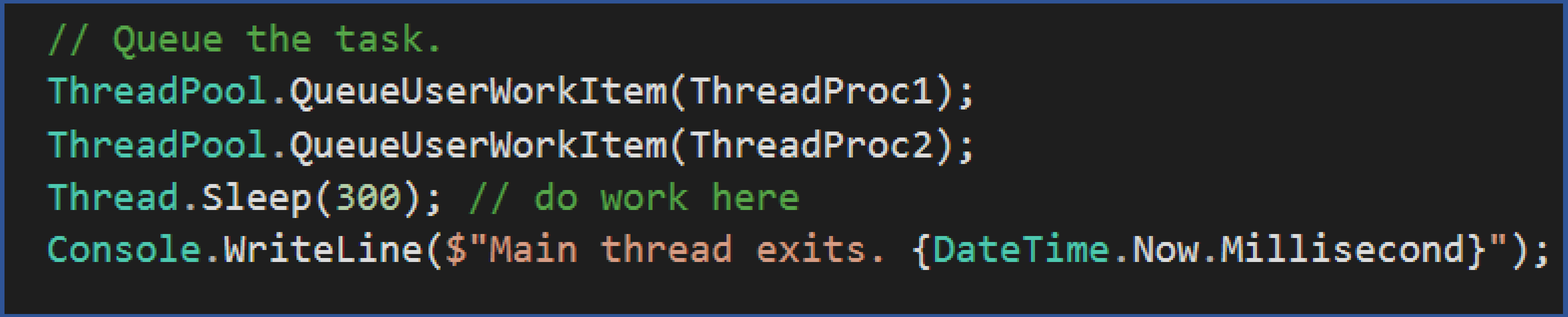
Exercise 00120: Thread VS GDI+

- 1st thread to draw rectangle randomly on form
- 2nd thread to draw circle randomly on form
- Both thread create and add shape to myList
- Use ManualResetEvent to prevent thread Accessing to myList during UI thread (Paint)



Exercise 00130: ThreadPool Class

- In the following example, the main application thread queues a method named ThreadProc to execute on a thread pool thread, sleeps for one second, and then exits.
- The ThreadProc method simply displays a message.

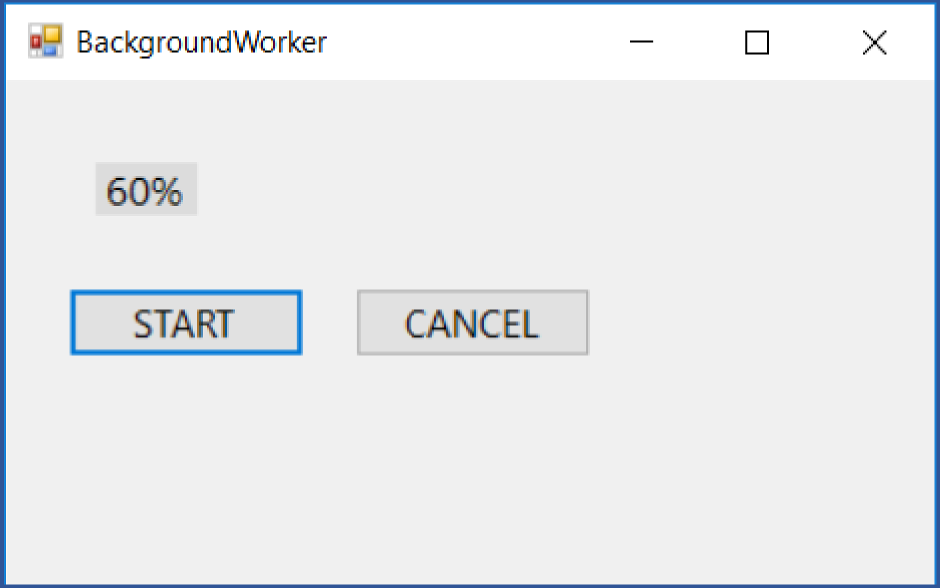


```
// Queue the task.
ThreadPool.QueueUserWorkItem(ThreadProc1);
ThreadPool.QueueUserWorkItem(ThreadProc2);
Thread.Sleep(300); // do work here
Console.WriteLine($"Main thread exits. {DateTime.Now.Millisecond}");
```

Output

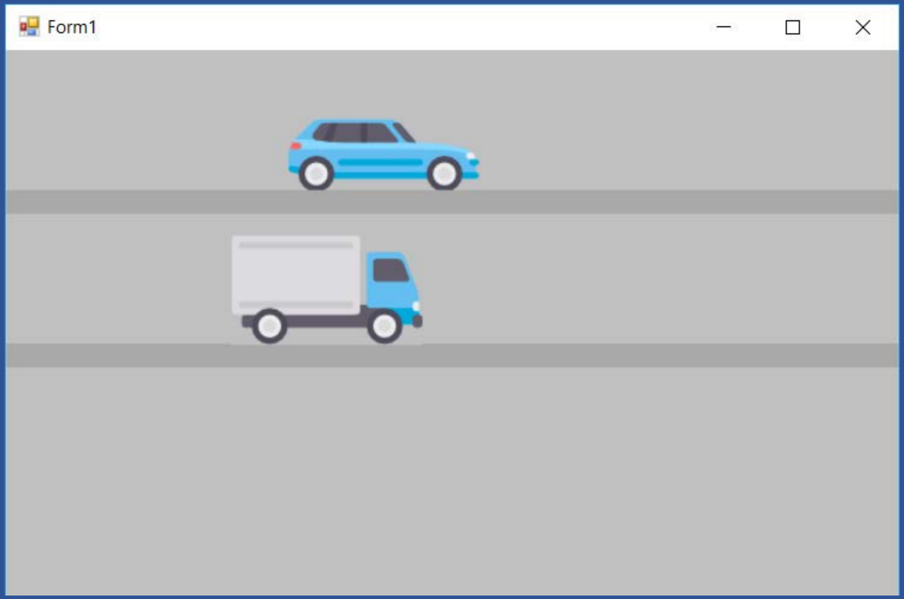
```
/*
Thread pool.proc1 567
Thread pool. proc2 668
Main thread exits. 767
*/
```

Exercise 02000: BackgroundWorker demo

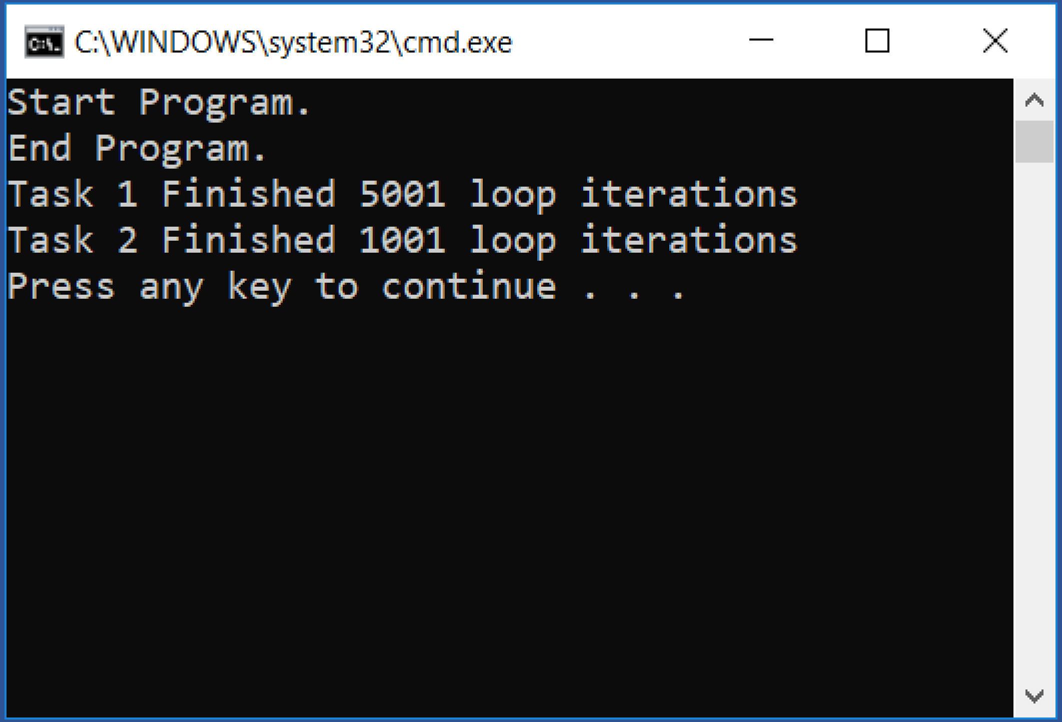


Exercise 02010: BackgroundWorker

1. Create New WinForm Project
2. Add Resource file
3. Add 2 pics to the resource file
4. Add 2 BackgroundWork to Main Form
5. Add 2 Picturebox
6. Write code to move pic from left to right

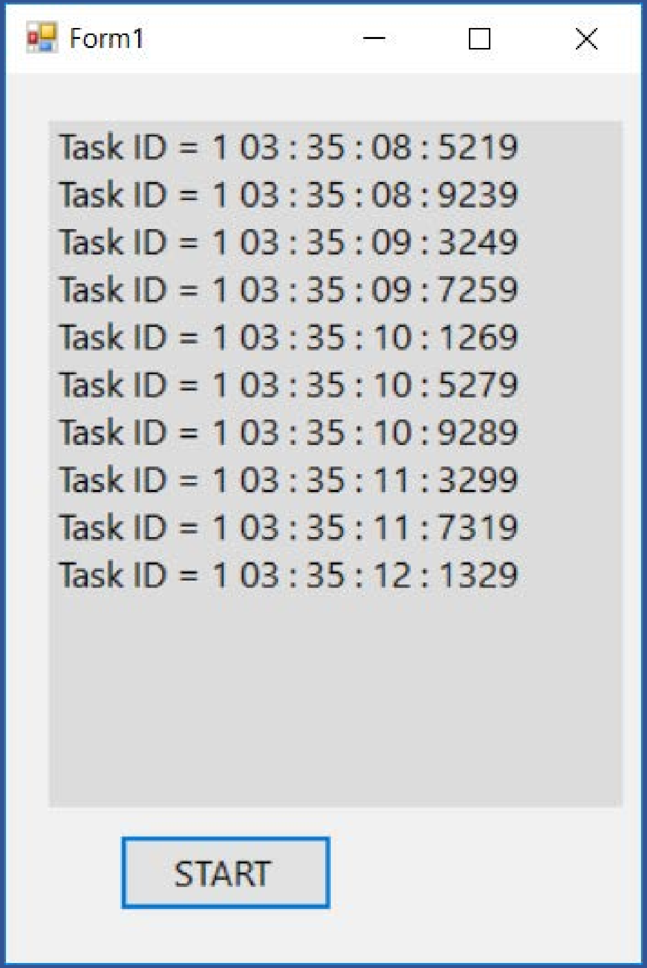


Exercise 02090: Task.Run vs Task.Factory.StartNew



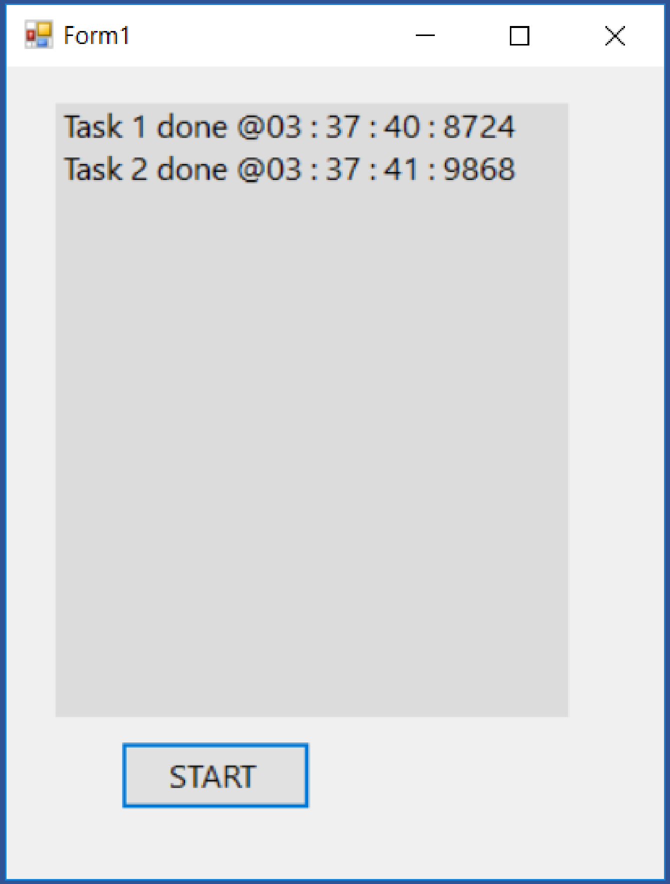
```
Start Program.
End Program.
Task 1 Finished 5001 loop iterations
Task 2 Finished 1001 loop iterations
Press any key to continue . . .
```

Exercise 03000: Creating and executing a task



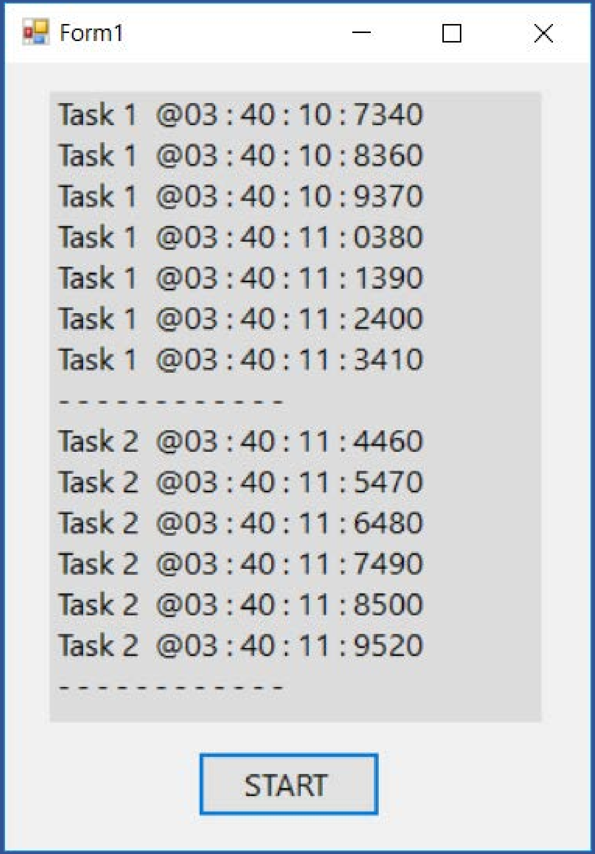
```
Task ID = 1 03 : 35 : 08 : 5219
Task ID = 1 03 : 35 : 08 : 9239
Task ID = 1 03 : 35 : 09 : 3249
Task ID = 1 03 : 35 : 09 : 7259
Task ID = 1 03 : 35 : 10 : 1269
Task ID = 1 03 : 35 : 10 : 5279
Task ID = 1 03 : 35 : 10 : 9289
Task ID = 1 03 : 35 : 11 : 3299
Task ID = 1 03 : 35 : 11 : 7319
Task ID = 1 03 : 35 : 12 : 1329
```

Exercise 03010: Using of Task.Wait()



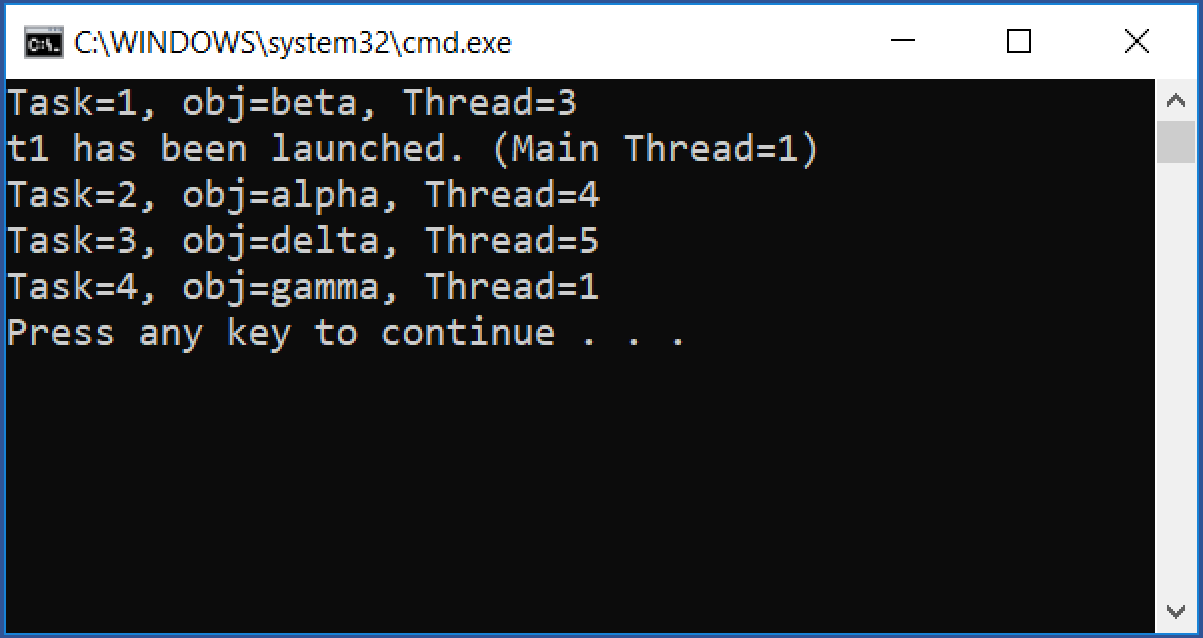
```
Task 1 done @03 : 37 : 40 : 8724
Task 2 done @03 : 37 : 41 : 9868
```

Exercise 03020: how to use ContinueWith()



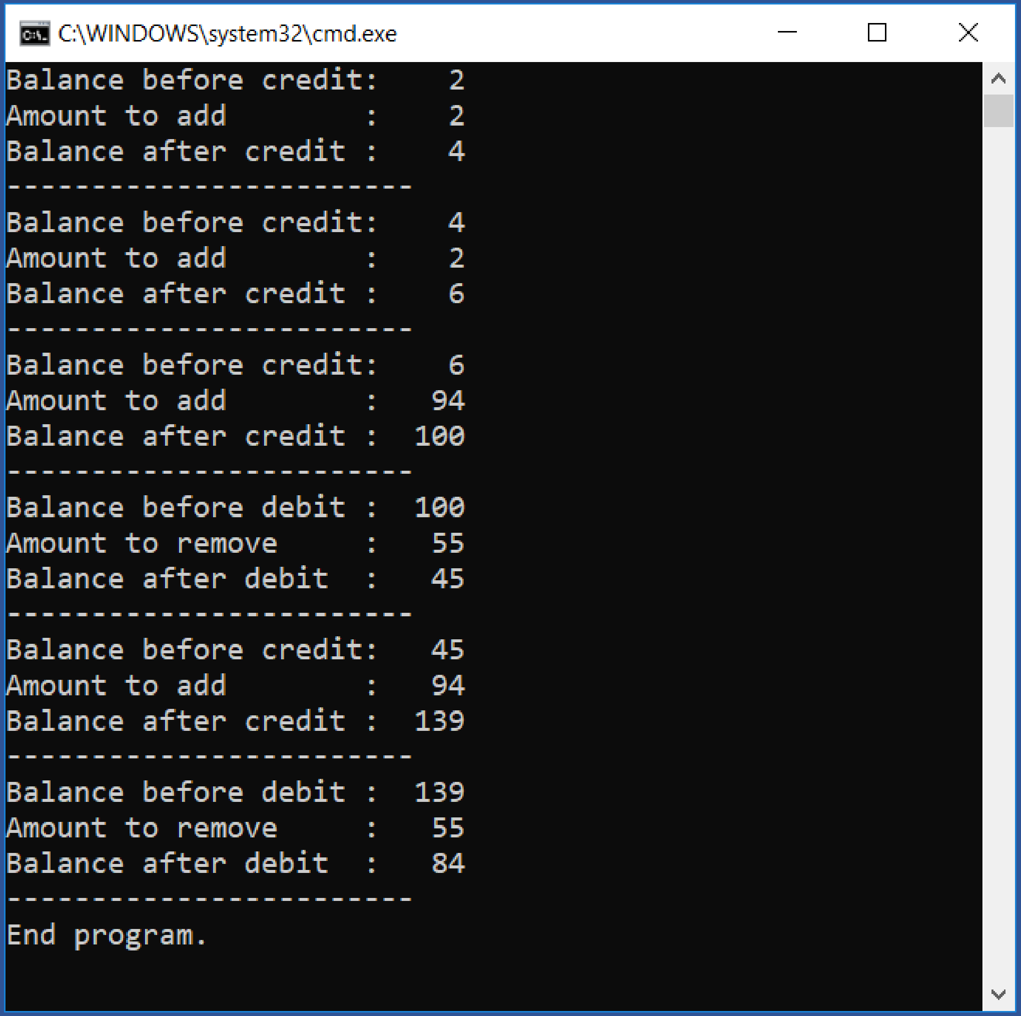
```
Task 1  @03 : 40 : 10 : 7340
Task 1  @03 : 40 : 10 : 8360
Task 1  @03 : 40 : 10 : 9370
Task 1  @03 : 40 : 11 : 0380
Task 1  @03 : 40 : 11 : 1390
Task 1  @03 : 40 : 11 : 2400
Task 1  @03 : 40 : 11 : 3410
- - - - - - - - - - - -
Task 2  @03 : 40 : 11 : 4460
Task 2  @03 : 40 : 11 : 5470
Task 2  @03 : 40 : 11 : 6480
Task 2  @03 : 40 : 11 : 7490
Task 2  @03 : 40 : 11 : 8500
Task 2  @03 : 40 : 11 : 9520
- - - - - - - - - - - -
```

Exercise 03030: Task instantiation



```
Task=1, obj=beta, Thread=3
t1 has been launched. (Main Thread=1)
Task=2, obj=alpha, Thread=4
Task=3, obj=delta, Thread=5
Task=4, obj=gamma, Thread=1
Press any key to continue . . .
```

Exercise 03030: lock statement



```
Balance before credit:    2
Amount to add        :    2
Balance after credit :    4
------------------------
Balance before credit:    4
Amount to add        :    2
Balance after credit :    6
------------------------
Balance before credit:    6
Amount to add        :   94
Balance after credit :  100
------------------------
Balance before debit :  100
Amount to remove     :   55
Balance after debit  :   45
------------------------
Balance before credit:   45
Amount to add        :   94
Balance after credit :  139
------------------------
Balance before debit :  139
Amount to remove     :   55
Balance after debit  :   84
------------------------
End program.
```