(T30)比較 Thread(執行緒)、Async、Await
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92
==========
(T30)比較 Thread(執行緒)、Async、Await
==========

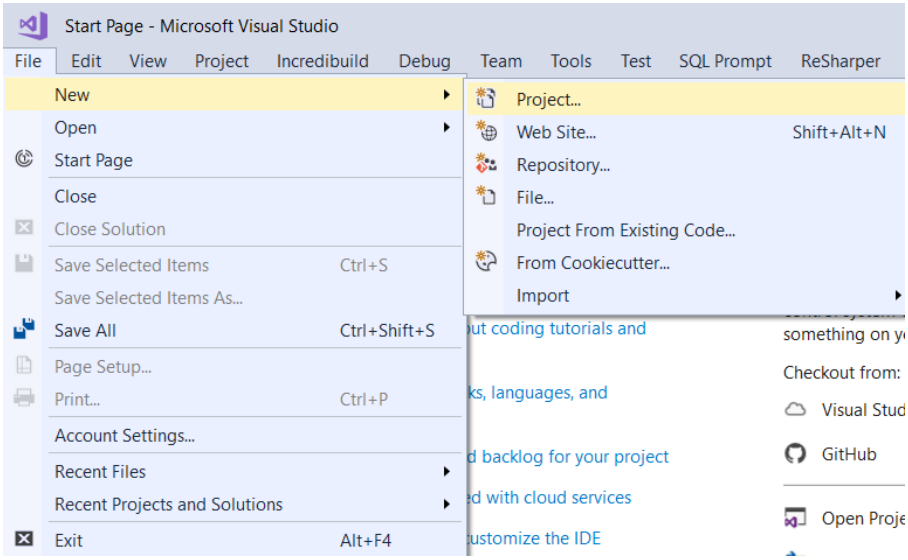
==========

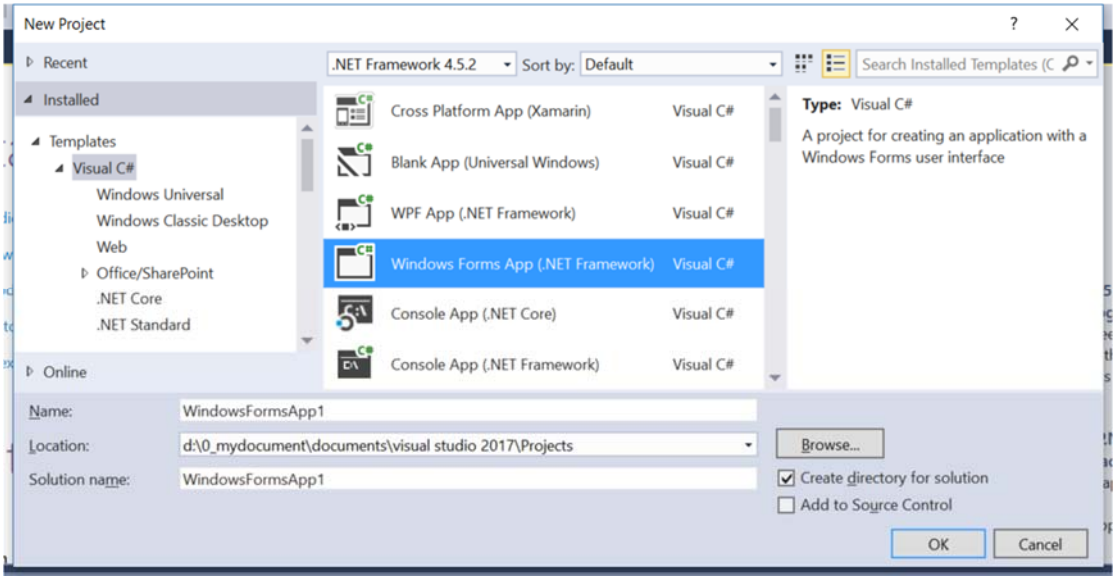
1. New Project

1.1. Create New Project : Sample

File --> New --> Project... -->

Visual C# --> **Windows Forms App (.Net Framework)** -->
Name: **WindowsFormsApp1**





==========

2. Form1.cs

2.1. Form1.cs [Design]

Drag and Drop 6 "Button"s on the Form and set the following properties
Name =btnThread1
Name =btnThread2
Name =btnThread3
Name =btnThread4
Name =btnThread5
Name =btnThread6

Drag and Drop one "Label" on the Form and set the following properties
Name = lbl1

2.2. Form1.cs

```
using System;
```

```
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            lbl1.Text = "";
            // 1. ================================================
            // No Thread, No Task problem.
            Console.WriteLine("1. No Thread, No Task problem. ======= ");
            btnThread1.Text = "btnThread1";
            btnThread1.Click += btnThread1_Click;
            // 2. ================================================
            // Thread fix the issues, but if cause new issues.
            Console.WriteLine("2. Thread fix the issues, but if cause new issues. ======= ");
            btnThread2.Text = "btnThread2";
            btnThread2.Click += btnThread2_Click;
            // 3. ================================================
            // Thread fix the issues, but if cause new issues.
            Console.WriteLine("3. Thread fix the issues, but if cause new issues. ======= ");
            btnThread3.Text = "btnThread3";
            btnThread3.Click += btnThread3_Click;
            // 4. ================================================
            // Thread fix the issues, but ...
            Console.WriteLine("4. Thread fix the issues, but ... ======= ");
            btnThread4.Text = "btnThread4";
            btnThread4.Click += btnThread4_Click;
            // 5. ================================================
            // Action and BeginInvoke(action) in Thread fix the issues, but it is too complex.
            Console.WriteLine("5. Action and BeginInvoke(action) in Thread fix the issues, but it is too
complex. ======= ");
            btnThread5.Text = "btnThread5";
            btnThread5.Click += btnThread5_Click;
            // 6. ================================================
            // Async and Await fix the issues, and it is easy.
            Console.WriteLine("6. Async and Await fix the issues, and it is easy. ======= ");
            btnThread6.Text = "btnThread6";
            btnThread6.Click += btnThread6_Click;
        }
        // 1. ================================================
        // No Thread, No Task problem.
        private int SlowFunc()
        {
            int outputInt = 0;
            Thread.Sleep(3000); // sleep for N millisecond.
            outputInt = 5000;
            return outputInt;
        }
        private void btnThread1_Click(object sender, EventArgs e)
        {
            lbl1.Text = "Processing...";
            int outputInt = SlowFunc();
```

```
    lbl1.Text = $"outputInt=={outputInt}.";
}
// 2 Problems.
// The application does not display the status message, "Processing...".
// While the application is busy processing,
// the UI windows form can not be re-sized or move.
// It is actually frozen
// 2. ================================================
// Thread fix the issues, but if cause new issues.
private void btnThread2_Click(object sender, EventArgs e)
{
    int outputInt = 0;
    Thread t1 = new Thread(() => { outputInt = SlowFunc(); });
    t1.Start();
    lbl1.Text = "Processing...";
    lbl1.Text = $"outputInt=={outputInt}.";
}
//1.
//Thread fix the issues, but if cause new issues.
//1.1.
//While the application is busy processing,
//the UI windows form can still be re-sized or move.
//It is not frozen any more.
//But
//The application still does not display the status message, "Processing...".
//It keep display "outputInt==0"
//1.2.
//While the Thread t1 is still working,
//but the UI main thread has already finished.
//Thus, UI Windows Form display "outputInt==0".
//But in fact, the outputInt will be updated after few millisecond.
// 3. ================================================
// Thread fix the issues, but if cause new issues.
private void btnThread3_Click(object sender, EventArgs e)
{
    int outputInt = 0;
    Thread t1 = new Thread(() => { outputInt = SlowFunc(); });
    t1.Start();
    lbl1.Text = "Processing...";
    // wait until Thread t1 finished then UI Thread can continue
    t1.Join();
    lbl1.Text = $"outputInt=={outputInt}.";
}
//1.
// The application display the status message, "Processing...".
// But while the application is busy processing,
// the UI windows form can not be re-sized or move.
// It is actually frozen.
// Because t1.Join(),
// the UI thread has to wait until t1 finished,
// then UI thread may continue.
// After the application finished its task,
// it correctly display "outputInt==N"
// N is the correct output of SlowFunc()
```

```
// 4. ================================================
// Thread fix the issues, but ...
private void btnThread4_Click(object sender, EventArgs e)
{
    int outputInt = 0;
    Thread t1 = new Thread(() =>
    {
        outputInt = SlowFunc();
        //1.
        //Error
        //System.InvalidOperationException
        //1.1.
        //it will throw System.InvalidOperationException
        //Cross-thread operation not valid:
        //Control 'lbl1' accessed from a thread
        //other than the thread it was created on.
        //1.2.
        //lbl1 was created by UI thread,
        //so only the UI thread can use lbl1
        //When other threads try to use lbl1,
        //it will throw System.InvalidOperationException
        lbl1.Text = $"outputInt=={outputInt}.";
    });
    t1.Start();
    lbl1.Text = "Processing...";
}
// 5. ================================================
// Action and BeginInvoke(action) in Thread fix the issues, but it is too complex.
private void btnThread5_Click(object sender, EventArgs e)
{
    int outputInt = 0;
    Thread t1 = new Thread(() =>
    {
        outputInt = SlowFunc();
        //If the working thread want to use parent thread, the UI thread,
        //then it need to use Action and BeginInvoke(action).
        //BeginInvoke() asks the UI thread,this,
        //to set the label.Text property in a type safe manner.
        Action action = () => lbl1.Text = $"outputInt=={outputInt}.";
        this.BeginInvoke(action);
    });
    t1.Start();
    lbl1.Text = "Processing...";
}
// The application displays the status message, "Processing...".
// While the application is busy processing,
// the UI windows form can be re-sized or move.
// It is not frozen any more.
// But it is too complex.
// 6. ================================================
// Async and Await fix the issues, and it is easy.
// If you want to "await", then the method must be "async".
private async void btnThread6_Click(object sender, EventArgs e)
{
    //Task<int> measn it will return an it when finish the Task.
    //This Task<int> task is actually pointing to SlowFunc() function
```

```
            Task<int> task = new Task<int>(SlowFunc);
            task.Start();
            lbl1.Text = "Processing...";
            //.Net will use some special algorism to avoid blocking and
            //Wait until the SlowFunc() task completes.
            int outputInt = await task;
            lbl1.Text = $"outputInt=={outputInt}";
        }
        // The application displays the status message, "Processing...".
        // While the application is busy processing,
        // the UI windows form can be re-sized or move.
        // It is not frozen any more.
        //Async and Await fix the issues, and it is easy.
    }
}
```