Day 9

The one link you need to recall

https://ddls.to/20483

Ready?

Do this every day BEFORE the class starts (takes about 15 minutes) (http://ddls.to/everyday)

1. Launch Lab01.
2. Login to Lab01 as **Admin**.
3. While in the Lab01 environment,
   i. run **cmd.exe** from the Windows Start button.
   ii. Run the command **git clone --depth 1** https://github.com/Mark-AIICT/CAD-2.git **C:\Users\Admin\Desktop\MarksFiles**
   iii. Navigate to **C:\Users\Admin\Desktop\MarksFiles\setups**, then right-mouse click **bootstrap.cmd** and **run as administrator**
   iv. While it's running, Sign in to Visual Studio on the Lab Environment. You can use any Microsoft account.
   v. When the script end it reboots the Virtual Machine. That's necessary.
   vi. Save the lab. (the save link is at the top right of the screen in the dropdown menu)

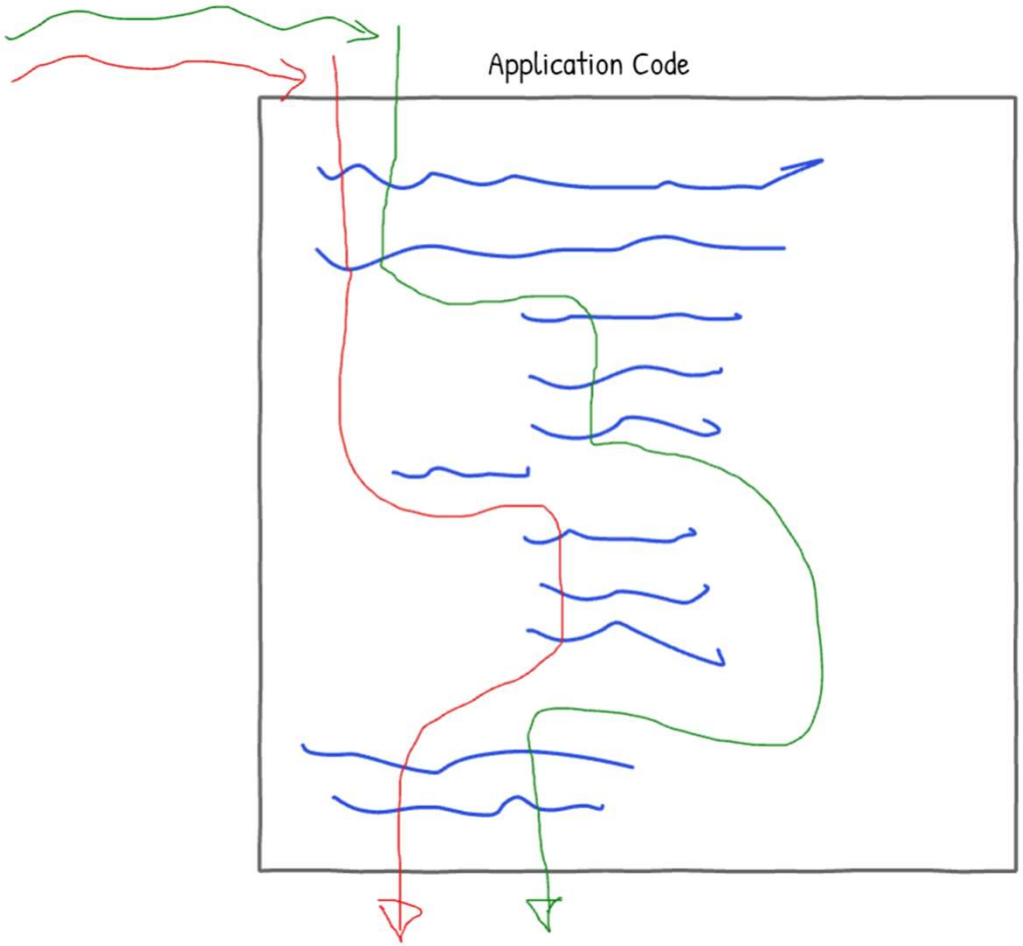
Course Outline

- Module 1: Review of Visual C# Syntax
- Module 2: Creating Methods, Handling Exceptions, and Monitoring Applications
- Module 3: Basic Types and Constructs of Visual C#
- Module 4: Creating Classes and Implementing Type-Safe Collections
- Module 5: Creating a Class Hierarchy by Using Inheritance
- Module 6: Reading and Writing Local Data
- Module 7: Accessing a Database
- Module 8: Accessing Remote Data (I'm replacing this with a better module)
- Module 9: Designing the User Interface for a Graphical Application
- Module 10: Improving Application Performance and Responsiveness
- Module 11: Integrating with Unmanaged Code
- Module 12: Creating Reusable Types and Assemblies
- Module 13: Encrypting and Decrypting Data

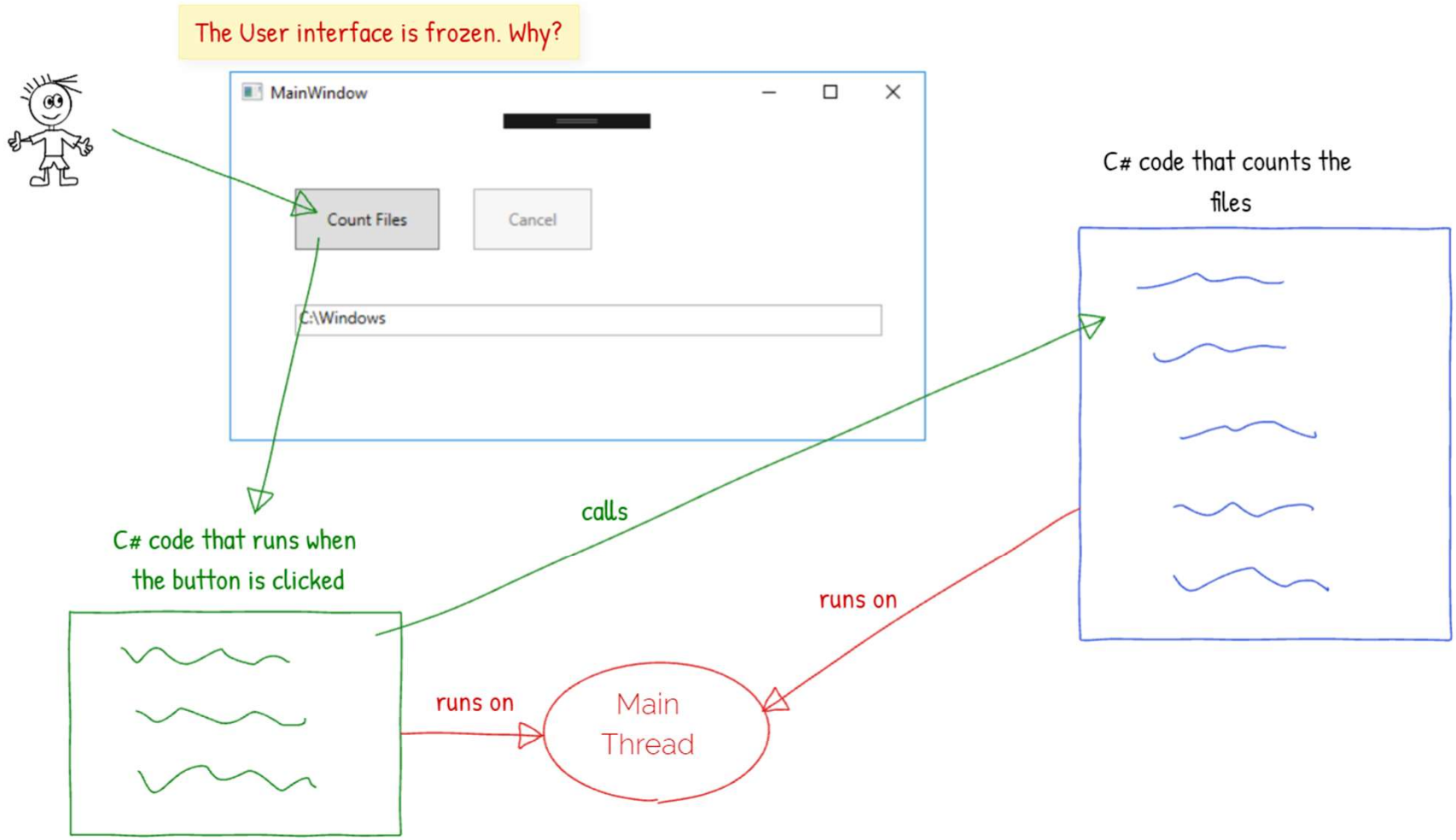
Multithreading

These are execution paths through the application. Can you see they look a bit like threads of cotton?

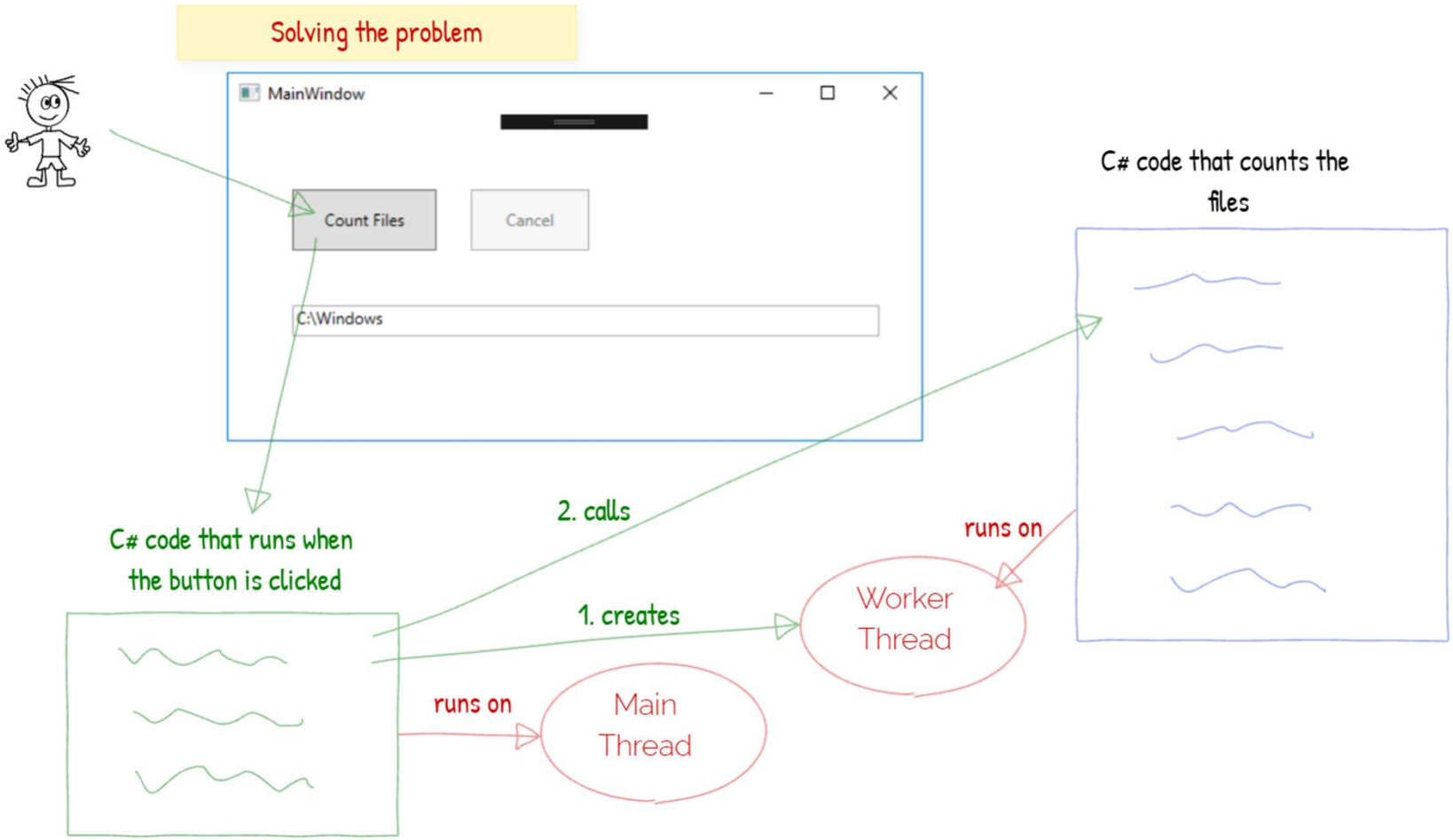
If the hardware on your device allows both paths to execute simulataneously then it is said to tbe muulti-threaded.



What's the problem we're trying to solve?



How will it be solved?



Before we look at Tasks…..

```
class Program
{
    //delegate void MyDelegate(int x, int y, string z);
    static void Main(string[] args)
    {
        //MyDelegate x = AddThem;

        Action<int, int, string> x = AddThem;
        x(3, 4, "5");

        Func<int, int, string, int> y = AddThem2;
        Console.WriteLine(y(5,6,"7"));


        Console.ReadLine();
    }

    static void AddThem(int x, int y, string z)
    {
        Console.WriteLine(x + y + Convert.ToInt32(z));
    }

    static int AddThem2(int x, int y, string z)
    {
        return x + y + Convert.ToInt32(z);
    }
}
```

A choice would be to define your own delegate data type and use it

You can write less code and use the built-in Action data type for methods that return void

If the called method returns a value then use Func rather than Action

Doing it with Lambdas results in less code

What is that?

```
class Program
{

    static void Main(string[] args)
    {
        Action<int, int, string> f = (x,  y,  z) =>
        {
            Console.WriteLine(x + y + Convert.ToInt32(z)); //if there's only one line of code you can get rid of the {}
        };
        f(3, 4, "5");

        Func<int, int, string, int> f2 = (x, y, z) => //if there's only one line of code you can get rid of the {}
        {
            return x + y + Convert.ToInt32(z);
        };
        Console.WriteLine(f2(5,6,"7"));

        Console.ReadLine();
    }


}
```

Lesson 1: Implementing Multitasking

- Creating Tasks
- Controlling Task Execution
- Returning a Value from a Task
- Cancelling Long-Running Tasks
- Running Tasks in Parallel
- Linking Tasks
- Handling Task Exceptions

Starting a task

```
static void Main(string[] args)
{
    Console.WriteLine("Type a number");
    string x = Console.ReadLine();


    Task T = new Task(p =>
    {
        Console.WriteLine("Calculation has started. Please wait....");
        DoSomethingThatTakesALongTime();
        Console.Write($"the cube of {x} is ");
        Console.WriteLine(Convert.ToInt32(p) * Convert.ToInt32(p) * Convert.ToInt32(p));
    }, x);


    T.Start();
    T.Wait();

    Console.WriteLine("Ok, we're done. press <enter> to end");
    Console.ReadLine();

}

private static void DoSomethingThatTakesALongTime()
{
    decimal result = 0;
    for (decimal i = 0; i < 100000000M; i++)
    {
        result += i;
    }
}
```

Starting a task that returns a value

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Type a number");
        string x = Console.ReadLine();


        Task<int> T = new Task<int>(p =>
        {
            DoSomethingThatTakesALongTime();
            return Convert.ToInt32(p) * Convert.ToInt32(p) * Convert.ToInt32(p);
        }, x);


        T.Start();
        T.Wait();

        Console.WriteLine($"The cube of {x} is {T.Result}");

        Console.WriteLine("Ok, we're done. press <enter> to end");
        Console.ReadLine();

    }

    private static void DoSomethingThatTakesALongTime()
    {
        decimal result = 0;
        for (decimal i = 0; i < 30000000M; i++)
        {
            result += i;
        }
    }
}
```

Parallel invoke

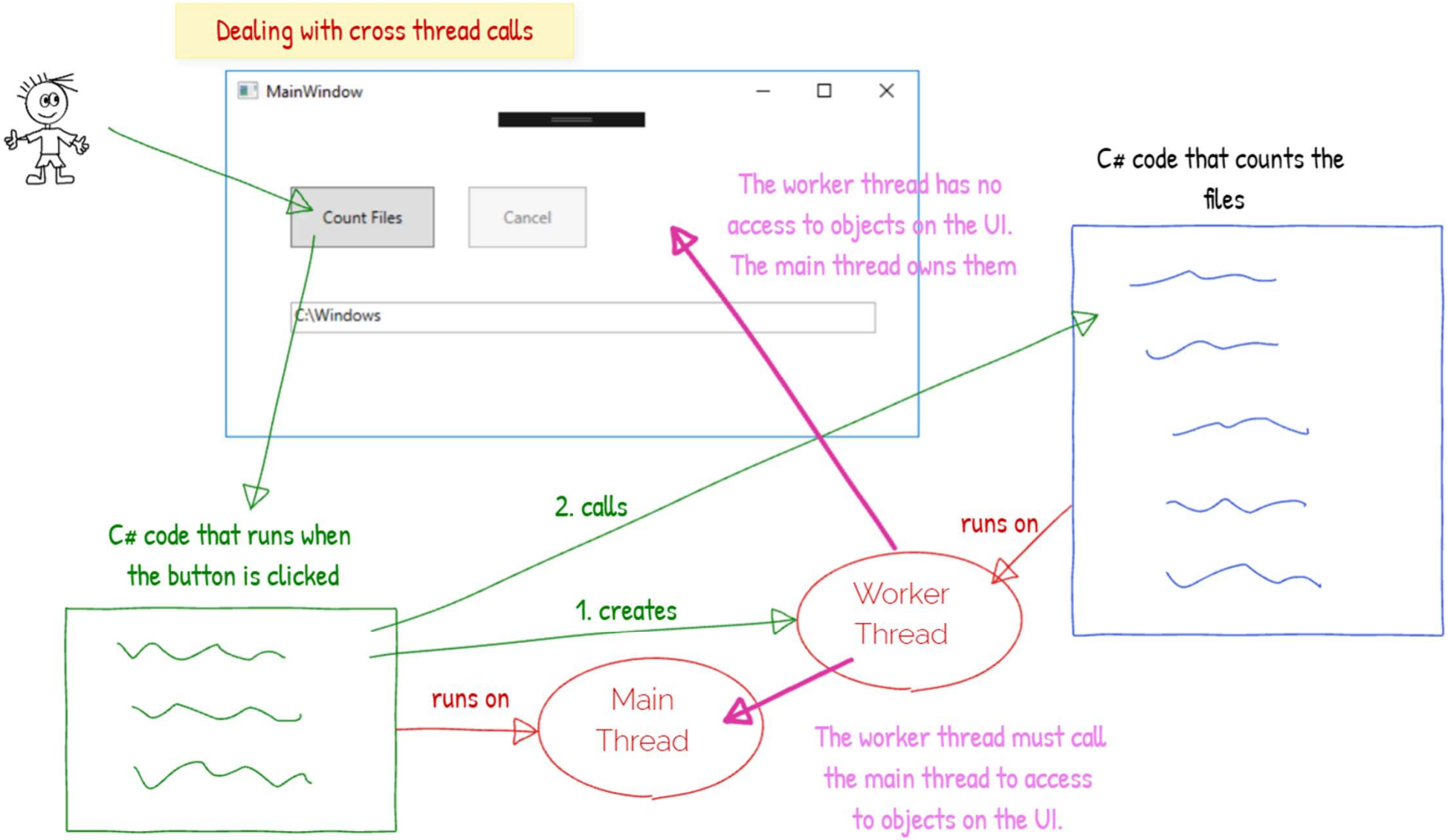
```
Parallel.Invoke(() =>
                {
                    decimal result = 0;
                    for (decimal i = 0; i < 40000000M; i++)
                    {
                        result += i;
                    }
                    MessageBox.Show(String.Format("40000000! is {0}", result));
                },

                () =>
                {
                    decimal result = 0;
                    for (decimal i = 0; i < 20000000M; i++)
                    {
                        result += i;
                    }
                    MessageBox.Show(String.Format("20000000! is {0}", result));
                }


                );
```

Task Continuation

```
static void Main(string[] args)
{
    // Create a task that returns a string.
    Task<string> firstTask = new Task<string>(() => "Hello");

    // Create the continuation task.
    // The delegate takes the result of the antecedent task as an argument.
    Task<string> secondTask = firstTask.ContinueWith((x) =>
                                    {
                                        DoSomethingThatTakesALongTime();
                                        return String.Format($"{ x.Result}, World!");
                                    }
    );

    // Start the antecedent task.
    firstTask.Start();
    secondTask.Wait();
    Console.WriteLine(secondTask.Result);
    Console.WriteLine("Done. Press enter to exit");
    Console.ReadLine();
}

private static void DoSomethingThatTakesALongTime()
{
    decimal result = 0;
    for (decimal i = 0; i < 10000000M; i++)
    {
        result += i;
    }

}
```

Lesson 2: Performing Operations Asynchronously

- Using the Dispatcher
- Using async and await
- Creating Awaitable Methods
- Creating and Invoking Callback Methods
- Handling Exceptions from Awaitable Methods


Dealing with cross thread calls
MainWindow
Count Files
Cancel
C:\Windows
The worker thread has no access to objects on the UI. The main thread owns them
C# code that counts the files
2. calls
runs on
C# code that runs when the button is clicked
1. creates
Worker Thread
runs on
Main Thread
The worker thread must call the main thread to access to objects on the UI.

Making a Cross thread call in WPF

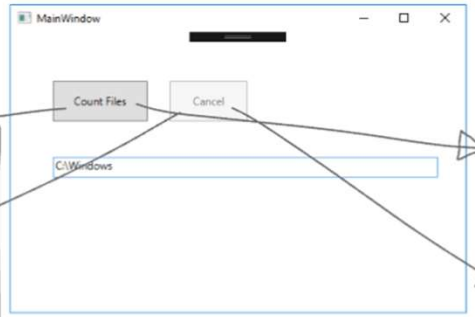
```
void MyFunction(object p)
{

    try
    {
        long result = CountOfFiles(p.ToString());
        MessageBox.Show(string.Format("There are {0} files in all directories below {1}",
                                      result, p));

        this.Dispatcher.BeginInvoke(new Action(()=>
        {
            NormalState();
        }));
    }
    catch (OperationCanceledException ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Using the Async Await pattern

Before async-await

MainWindow.Xaml.CS

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    cancellationTokenSource = new CancellationTokenSource();

    Task Tsk = new Task(MyFunction, T.Text);
    CountingState();
    Tsk.Start();
}

private void Button_Click_2(object sender, RoutedEventArgs e)
{
    cancellationTokenSource.Cancel();
    NormalState();
}

void MyFunction(object p)
{

    try
    {
        long result = CountOfFiles(p.ToString());
        MessageBox.Show(string.Format("There are {0} files in all directories below {1}",
                                      result, p));

        this.Dispatcher.BeginInvoke(new Action(()=>
        {
            NormalState();
        }));
    }
    catch (OperationCanceledException ex)
    {
        MessageBox.Show(ex.Message);
    }
}


long CountOfFiles(string path)
{
    long filesInPath=0;
```



With async-await

MainWindow.Xaml.CS

```
async private void Button_Click_1(object sender, RoutedEventArgs e)
{
    cancellationTokenSource = new CancellationTokenSource();
    CountingState();
    await MyFunction(T.Text);
}

private void Button_Click_2(object sender, RoutedEventArgs e)
{
    cancellationTokenSource.Cancel();
    NormalState();
}

async Task MyFunction(string p)
{
    try
    {
        Task<long> tsk = new Task<long>(x=>CountOfFiles(x.ToString()),p);
        tsk.Start();
        long result = await tsk;
        NormalState();
        MessageBox.Show(string.Format("There are {0} files in all directories below {1}",
                                      result, p));
    }
    catch (OperationCanceledException ex)
    {
        MessageBox.Show(ex.Message);
    }
}

long CountOfFiles(string path)
{
    long filesInPath=0;
```

Lesson 3: Synchronizing Concurrent Access to Data

- Using Locks
- Using Synchronization Primitives with the Task Parallel Library
- Using Concurrent Collections

Course Outline

- Module 1: Review of Visual C# Syntax
- Module 2: Creating Methods, Handling Exceptions, and Monitoring Applications
- Module 3: Basic Types and Constructs of Visual C#
- Module 4: Creating Classes and Implementing Type-Safe Collections
- Module 5: Creating a Class Hierarchy by Using Inheritance
- Module 6: Reading and Writing Local Data
- Module 7: Accessing a Database
- Module 8: Accessing Remote Data (I'm replacing this with a better module)
- Module 9: Designing the User Interface for a Graphical Application
- Module 10: Improving Application Performance and Responsiveness
- Module 11: Integrating with Unmanaged Code
- Module 12: Creating Reusable Types and Assemblies
- Module 13: Encrypting and Decrypting Data