VS2012 の秘密兵器、
async/await

BluewaterSoft

biac

自己紹介

biac
(山本 康彦)

BluewaterSoft

http://www.bluewatersoft.jp

- 1957、スプートニクの前に誕生
- 1983、名古屋大学工学部(修士)卒
- HONDA R&Dで自動車設計
- 1994～ ソフトウェア業界
- 2012～ BluewaterSoft

著作

- 著書
  - 「**速攻入門 C#**」 (2012/3) 技術評論社、共著
  - 「**ソフトな彼女とハードな彼氏。**」(2012/3) **アジャイルマインド**vol.1 掲載
- 記事
  - 連載中 「**C#でTDD入門**」CodeZine
  - 「**Metroスタイルアプリの開発者が知るべき3つのこと**」、他 **@IT** - .NET開発者中心

…etc.

わんくま同盟

わんくま同盟

名古屋勉強会で
TDD道場

次回**9月22日**、矢作紗友里誕生日

async / await - 辞書を引いてみる

- asynchronous [eisíŋkrənəs, æs-]
  非同期の

- await [əwéit]
  待ち受ける

※「プログレッシブ英和中辞典」より。

ごめん、タイトルは釣りです m(_`_)m

async / await は、
VS2012の機能じゃなく、

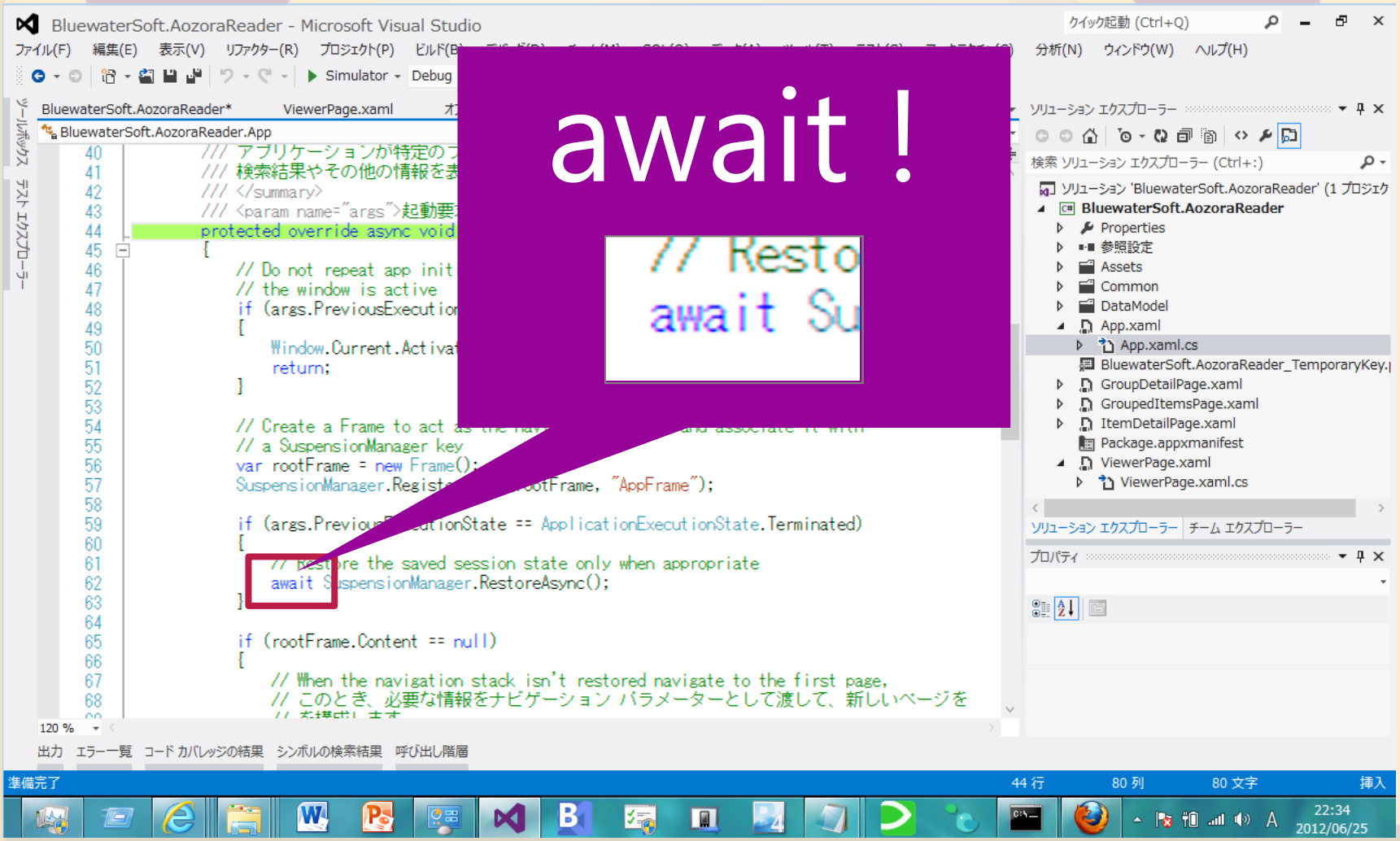
.NET 4.5 の新機能

今秋登場! Win8 & Windows Phone 8

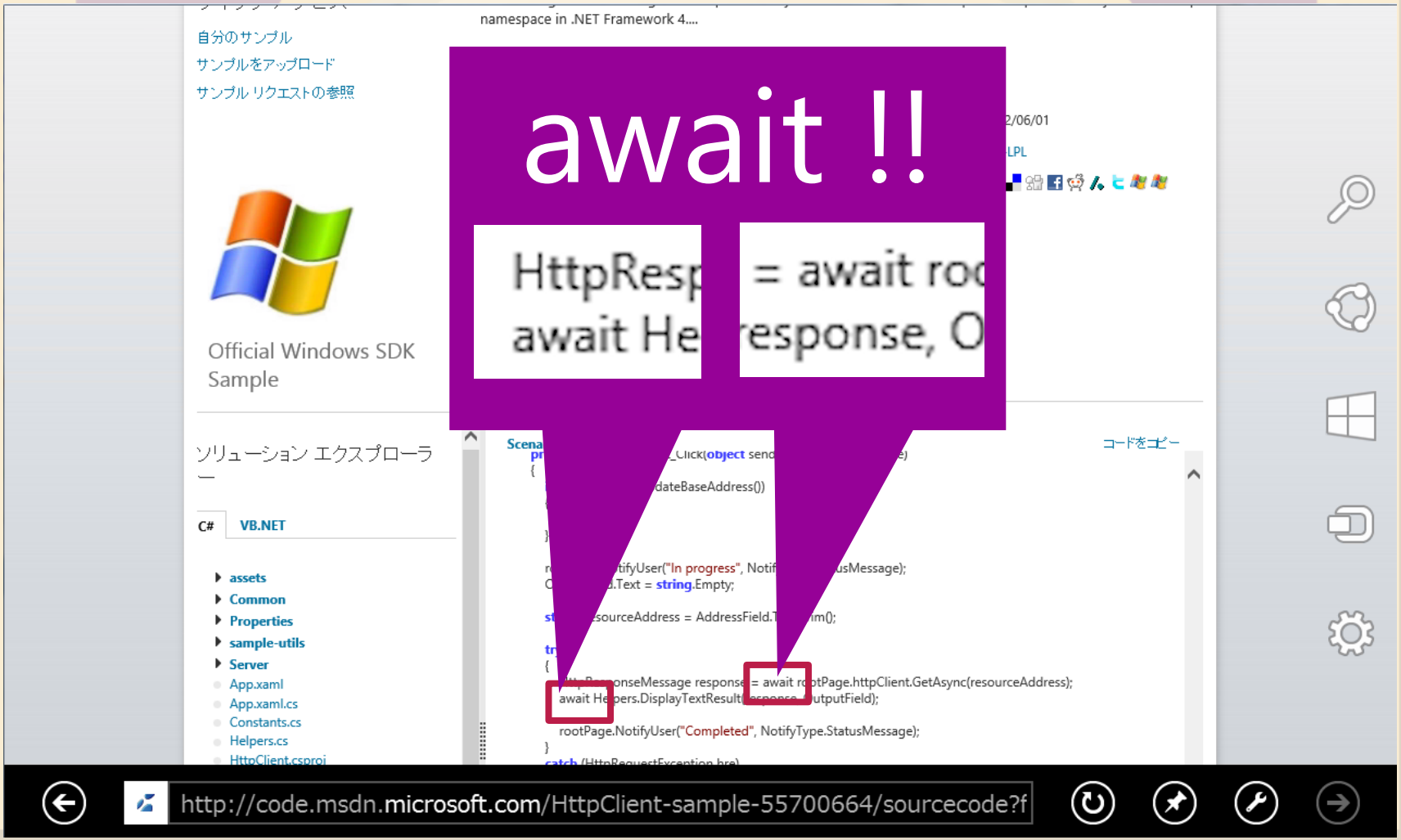
Win8ノートとタブレットは "Windows 8 Release Preview Product guide"
http://www.microsoft.com/about/mspreview/windows8/Windows8_RP_Product_guide.pdf
p.28の画像より。
MS製Win8タブレット"Surface" は MS News Center
http://www.microsoft.com/en-us/news/press/2012/jun12/06-18announce.aspx より。
WP8は、MS News Center Image Gallery http://www.microsoft.com/en-us/news/presskits/windowsphone/imageGallery.aspx より。


4G
12:38
Project development meeting
3:00 PM - 5:00 PM
Phone
Tue 31
Doug
Grandpa
Food
Luca's display show
Sat 9:00 AM
Sat 31
$20 for a three course japanese dinner with wine for two
Games
Yearly accounts review
Building 1940
Thursday to Friday
Calendar
Tue 31
ESPN
SCORECENTER
Maria
Missed call
Dave Barnett
AP
NEWS
People
Photos
College soccer game
West Mercer
Tomorrow 5:00 PM - 7:00 PM
Calendar
Tue 31
Phone
Photos
4:23
Adam
kindle
VOGUE
Book club at Sandra's
Seattle
Thursday 7:00 PM - 9:00 PM
3:10
Cherry vanilla chip muffins
Allrecipes
Phone
Messaging
52°/38°F
NEW YORK
AccuWeather.com
Photos

VS2012のテンプレートに



サンプルコードにも


namespace in .NET Framework 4....
自分のサンプル
サンプルをアップロード
サンプル リクエストの参照
await !!
HttpResp = await roo
await He esponse, O
Official Windows SDK Sample
ソリューション エクスプローラー
C# VB.NET
assets
Common
Properties
sample-utils
Server
App.xaml
App.xaml.cs
Constants.cs
Helpers.cs
HttpClient.csproj
コードをコピー
await Helpers.DisplayTextResult
rootPage.NotifyUser("Completed", NotifyType.StatusMessage);
http://code.msdn.microsoft.com/HttpClient-sample-55700664/sourcecode?f

Metroを覗くとawaitだらけ

- …ということは!?
- async / await を使いこなせないと、Win8 / WP8 に乗れない!!

今日、覚えて帰って欲しいこと

await は、

待たない!!

ちょっと 復習

- みんな! **yield return** は覚えているか!?

- 「ぇえと、たしか…」
- 「ループの途中で**いったん値を返して**、
  次に呼ばれたら、その続きを…」
  「そう、それだ!」
- 「yield returnの行で処理が止まるとは
  思わないよな。」

yield return の例

```
·[TestCase()]
·public·void·YieldReturnを使う()·{
···foreach·(int·n·in·YieldReturnのメソッド())
·····Console.Write("{0}¥r",·n);
·}

·private·IEnumerable<int>·YieldReturnのメソッド()·{
···for·(int·i·=·0;·i·<·10;·i++)
·····yield·return·i;
·}
```

yield return のように、await も待たない。
そこでメソッドから一旦抜けて、次の行から再開する。

では…

そろそろ

本題に入ろうか

画面がフリーズする

- 長時間掛かる処理をしていると、
  その間は画面がフリーズする。
  ので、少し気が利いてるアプリは 砂時計
  出したりする。

- これって嫌じゃね?

- つか、タッチパネルだと
  砂時計が出ないじゃん!!

画面がフリーズするのダメ、ゼッタイ

がんばって
非同期プログラミング
してみよっか

非同期プログラミングか～

- MSDN調べなきゃ!
  えっと、**BeginInvoke()** の使い方…

- いやそれ、レガシーコードだからw
  いつまで .NET 1.x やってるの!?

※ サンプルコードを書こうかと思ったけど、
長くてめんどくさいだけなので、省略! f(^^;

非同期プログラミングか～

- えっと…、じゃぁWinFormだから **BackgroundWorker** を…
- いやそれ、レガシーコードだからw いつまで .NET 2.0 やってるの!?

※ これのサンプルコードも省略! f(^^;

今どきは、Task クラス

- VS2010 (.NET Framework 4) なら、**Task** クラス使うっしょ!

```
private void buttonLongProcess_Click(object sender, EventArgs e) {
    var orgText = this.buttonLongProcess.Text;
    this.buttonLongProcess.Text = "長い処理中…";

    //UIスレッドで処理を実行するTaskScheduler
    var uiTaskScheduler =
        =TaskScheduler.FromCurrentSynchronizationContext();

    var t = Task.Factory.StartNew(() => {
            長い時間が掛かるメソッド();
        });
    t.ContinueWith(//非同期処理が終わったら、この中を継続して実行
        result => {
            this.buttonLongProcess.Text = orgText;
        },
        uiTaskScheduler //この継続処理は、UIスレッドで実行させる
    );
}

private void 長い時間が掛かるメソッド() {
    Thread.Sleep(10 * 1000);
}
```

Task.Factory.StartNew() / ContinueWith()

```
private void buttonLongProcess_Click(object sender, EventArgs e) {
    var orgText = this.buttonLongProcess.Text;
    this.buttonLongProcess.Text = "長い処理中…";

    //UIスレッドで処理を実行するTaskScheduler
    var uiTaskScheduler
        = TaskScheduler.FromCurrentSynchronizationContext();

    var t = Task.Factory.StartNew(() => {
            長い時間が掛かるメソッド();
        });
    t.ContinueWith(//非同期処理が終わったら、この中を継続して実行
        result => {
            this.buttonLongProcess.Text = orgText;
        },
        uiTaskScheduler //この継続処理は、UIスレッドで実行させる
    );
}

private void 長い時間が掛かるメソッド() {
    Thread.Sleep(10 * 1000);
}
```

非同期実行される部分

非同期実行が終わった後で、実行される部分

Task クラスで非同期処理を書いた

- やったぜ!
  これで画面がフリーズしないぞ!!

Taskクラス、すげ～っ!

- でも、しばらく使っていると、
  気が付いてしまう…

- なんで、呼び出す側が、
  毎回毎回、何度も何度も、
  同じことをダラダラ書かないと
  いけないの!? (--メ;

それって、つまり…?

```
···var·uiTaskScheduler·
·····=·TaskScheduler.FromCurrentSynchronizationContext();

···var·t·=·Task.Factory.StartNew(()·=>·{
·····長い時間が掛かるメソッド();
·····});
···t.ContinueWith(·//·非同期処理が終わったら、この中を継続して実行
·····result·=>·{
·····  ·this.buttonLongProcess.Text·=·orgText;
·····},
·····uiTaskScheduler·//この継続処理は、UIスレッドで実行させる
···);
```

```
···おまじない;長い時間が掛かるメソッド();
···this.buttonLongProcess.Text·=·orgText;
```

違うとこだけ
書けば済むように
してくれ〜!!

Microsoft 「よし、わかった!」

汝、.NET 4.5 からは async と await を使うべし

async / await は、Task… の省略記法

- 厳密には違うが…、まぁそう思ってくれ。

- 呼び出す側は楽になるぞ!
  呼び出される側は、
  ちぃっと面倒になるがな～w

2つの呪文: async と await

```
・・・おまじない・・長い時間が掛かるメソッド();
・・・this.buttonLongProcess.Text・=・orgText;
・
```

```
    private async void buttonLongProcess_Click2(object
sender, EventArgs e)
    {
        var orgText = this.buttonLongProcess.Text;
        this.buttonLongProcess.Text = "長い処理中…";

        await 長い時間が掛かるメソッドAsync();
        this.buttonLongProcess.Text = orgText;
    }

    private Task 長い時間が掛かるメソッドAsync() {
      return Task.Factory.StartNew(() =>
                Thread.Sleep(10 * 1000)
            );
    }
```

メソッド名の末尾に
"Async" を付ける慣習

**呼び出す側は
希望した通り!**

**呼び出される側に
StartNew()
とかが来た!**

メソッドの返値

```
[TestMethod]
public async Task Test01_返値を返すメソッドTest()
{
    int result1 = await 返値を返すメソッドAsync1(2);
    Assert.AreEqual<int>(3, result1);

    int result2 = await 返値を返すメソッドAsync2(2);
    Assert.AreEqual<int>(4, result2);
}

private Task<int> 返値を返すメソッドAsync1(int n)
{
    return Task.Factory.StartNew<int>(() => {
        Thread.Sleep(1000); return n + 1;
    });
}

private async Task<int> 返値を返すメソッドAsync2(int n) {
    await Task.Delay(1000);
    return n * 2; //←あれっ!? Task<int>型を返してないぞ!
}
```

Task<T>型にして
返してやる

await が
Task<T> から
T に戻してくれる

async が
Task<T> に
ラップしてくれる

await で一旦リターンするんだってば!!

```
private int testData = -1;

[TestMethod]
public void Test02_awaitで一旦returnしてくる() {
    WriteNowTime();
    Console.WriteLine("1. 呼び出し前: testData = {0}", testData);

    倍にしてメンバー変数testDataに代入するAsync(3);

    WriteNowTime(); //ここではまだ計算されていない!
    Console.WriteLine("2. 呼び出し後: testData = {0}", testData);

    Thread.Sleep(2000); WriteNowTime();
    Console.WriteLine("4. 2秒経過後: testData = {0}", testData);
}

private async Task 倍にしてメンバー変数testDataに代入するAsync(int n)
{
    await Task.Delay(1000); //ここで一旦returnされる!
    WriteNowTime(); Console.WriteLine("3. メソッド内で待機終了");
    testData = n * 2;
}
```

testData に 6 が
入るはずだが…

この計算が実行されるのは
何時だろう?

…ほら、await は待たずに return する!

すぐに return
している

1秒後に
計算が始まる

このことを忘れていると、
await 後の処理が実行されていないように見える!!

今の話、WinForm でもう一度

```
private int data;

private void buttonCalc_Click(object sender, EventArgs
{
    Calc();
    textBoxOutput.Text = this.data.ToString();
}

private async Task Calc()
{
    this.data = int.Parse(textBoxInput.Text); //入力値を
    //ほんとはDBアクセスとか時間の掛かることを非同期でしてると思ってくれ
    await Task.Delay(3000);

    //計算結果をメンバー変数に格納（←副作用！）
    this.data = this.data + 1; // この行が実行されないように思える
}
```

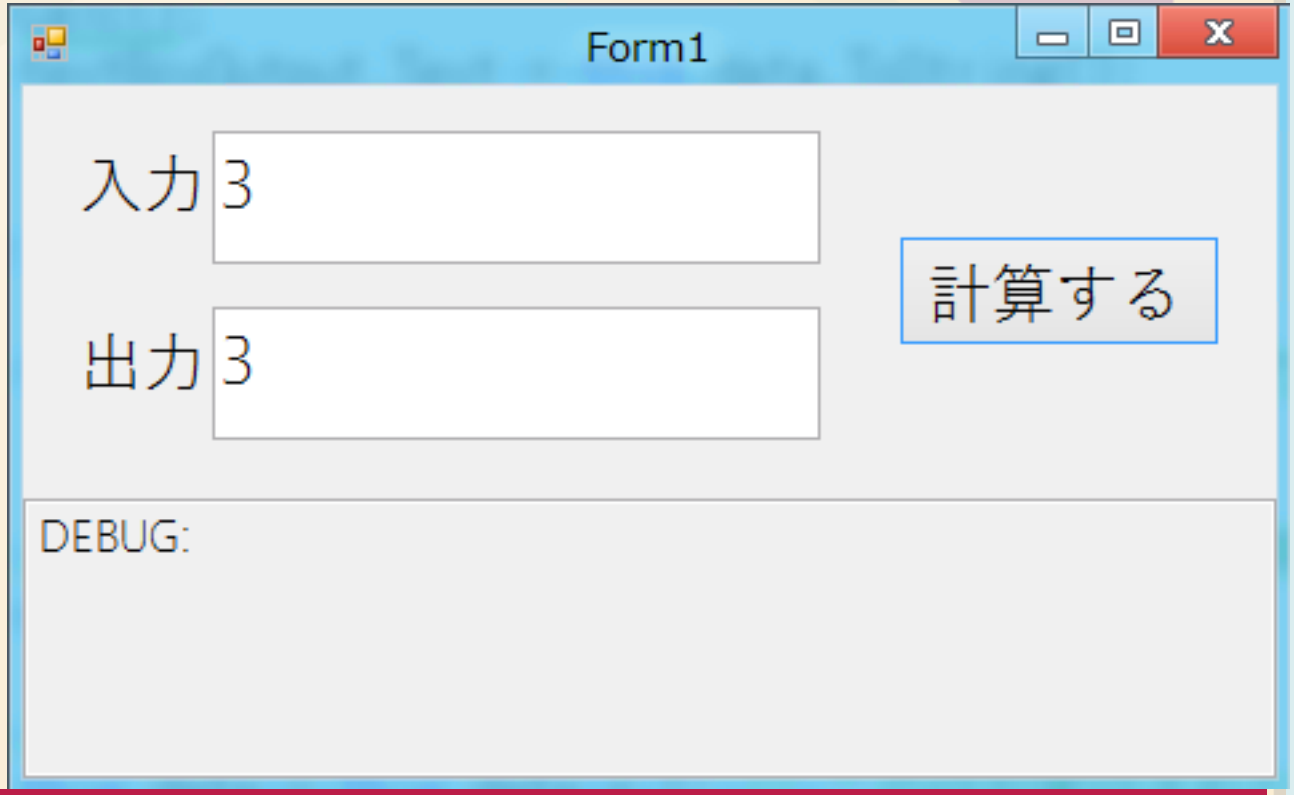
計算する
(1を足した)

計算結果を
TextBoxに表示

計算を実行する
前にリターン

この行が実行
されるのは何時?

足し算してくれてないよぉ～ (ToT)



これを直すには、
ちゃんと **await を付ける!!** あと、**副作用はヤバイ**って!

つまり…

- **await の後に副作用**を書いたら、

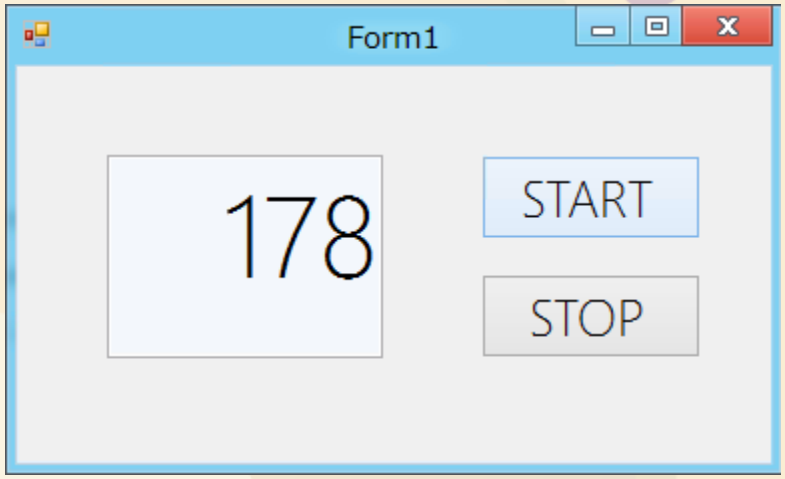
  死亡フラグ☠

  ってことだ f(^^;

- あと、Async なメソッドは
  必ず await で受けろ!

おまけ：タイマー割り込み代わりにも

- たとえば、こんなカウントダウンタイマー



- 精度を気にしないなら、async / await でさっくり書けたりする

async / await でタイマー代わり

```
private bool _continue;

private async void button1_Click(object sender, EventArgs e)
{
    _continue = true;
    for (int count = int.Parse(this.textBox1.Text); count >= 0; count--) {
        await Task.Delay(1000);

        if (!_continue)
            break;

        this.textBox1.Text = count.ToString();
    }
}

private void button2_Click(object sender, EventArgs e)
{
    _continue = false;
}
```

最後に、大事な事なので 2回言います。

await は、

待たずに 帰る!