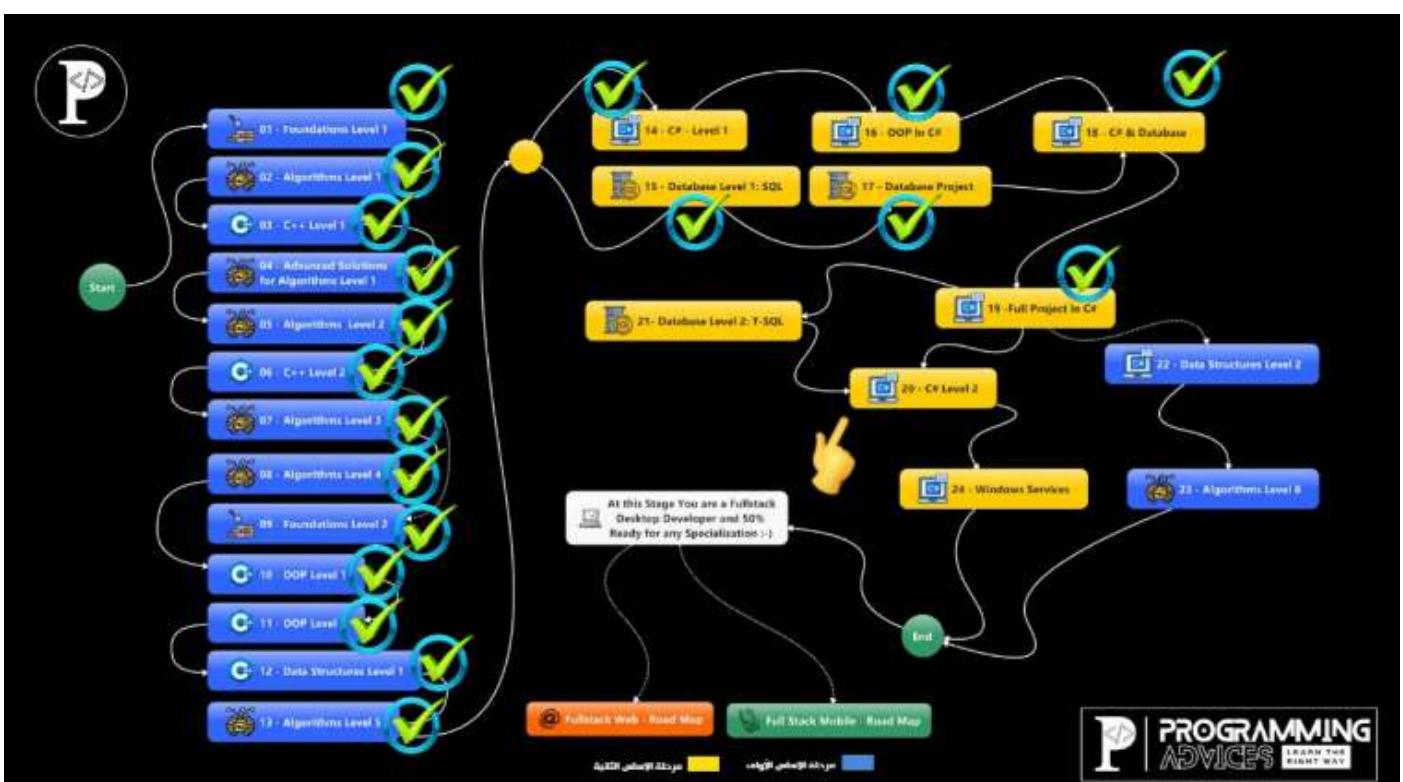


About This Course



مهم جدا

يجب ان تكون قد انهيت جميع الكورسات المشار اليها بالاخضر



Telegram Group for This Course

رابط المجموعة على التلجرام

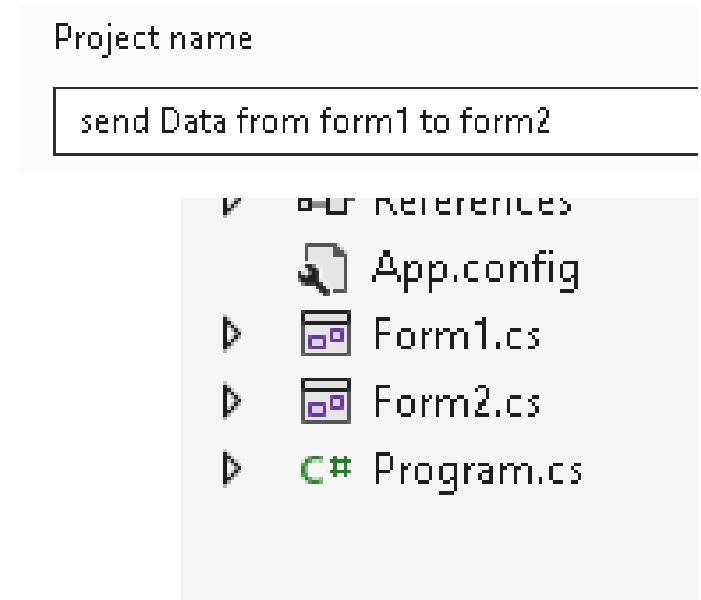
https://t.me/+WbWn3Ghti_05ZWU0

Send Data To Form

الأول لما بنكون عاملين فورم والفورم ده بيفتح فورم تاني وكنا عاوزين نبعثه داتا بنبعته عن طريق التعديل عال constructor

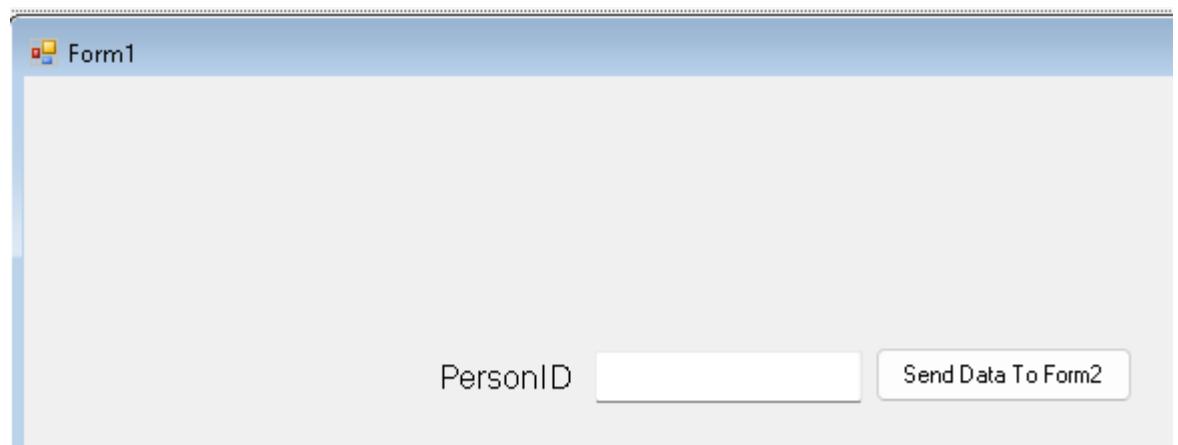
تعالي نشوف هوا عايز يقول ايه

تعالي نعمل تطبيق جديد ونعمل اتنين فورم



اول فورم هنحط فيه LABEL و text box و زرار

تاني فورم هنحط فيه اتنين label



PersonID Recieved Is

label2

بعدين نروح عالكود بتاع form2 واعرف متغير واضيفه عال constructor بتاع الفورم
وبعدين في ال event بتاع الفورم اللي اسمه load بعدد ال label

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace send_Data_from_form1_to_form2
{
    public partial class Form2 : Form
    {
        private int _ID = -1;
        public Form2(int iD)
        {
            InitializeComponent();
            _ID = iD;
        }

        private void Form2_Load(object sender, EventArgs e)
        {
            lblRecieved.Text = _ID.ToString();
        }
    }
}
```

كده جهزنا form2 عشان يستقبل داتا
نروح للزرار اللي في form1 ونبعد منه الداتا

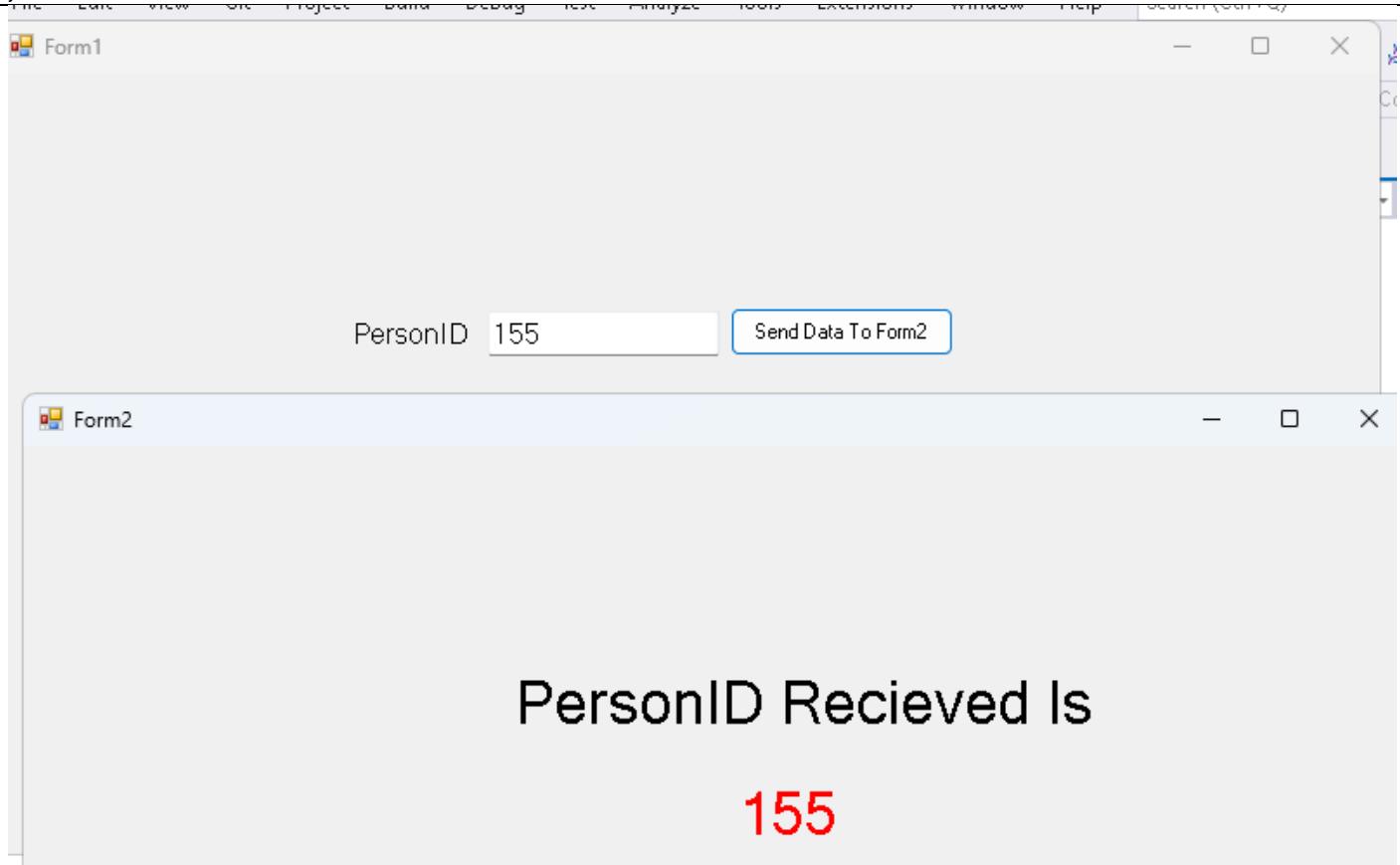
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```

namespace send_Data_from_form1_to_form2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnSend_Click(object sender, EventArgs e)
        {
            Form frm = new Form2(int.Parse(textBox1.Text));
            frm.ShowDialog();
        }
    }
}

```



Send Data Back To Form Using Delegate

ومعناها انك بتخلي غيرك يعمل شغلك وفي البرمجه بيقولك ال delegation هيا عباره عن method pointer على الكوبي

تعرف الكوبي ؟

اه

طب ايه وظيفته ؟

انه بيكون اختصار بياخد الناس من مكان بيوديهم مكان تاني

بيتعمل امتى ؟

لما بيكون فيه صعوبه في التنقل بين المكانين دول

طيب ليه بقولك الكلام ده؟

عشان ده هوا نفس المبدأ بتاع ال delegate

هشرحلك مثل في الاندرويد ستوديو بلغة الجافا وبعدين نيجي نطبق بال C#

في الاندرويد ستوديو كانت ظهرت عندي مشكلة شبيهه بدرس النهارده وهيا اني كان عندي كلاس فيه بيانات معينه وعندي شاشة

فكت عايز وانا واقف في الكود بتاع الشاشه اكتب كود معين الكود ده بيستخدم داتا موجوده في الكلاس
الداتا دي مش هعرف اوصلها من بره

فكان حل المشكلة دي هوا ال delegation
طب كان ايه الحل برضه؟

الحل كان اني اعمل interface وال method ده جواه
ال method كانت بتاخذ parameters من نفس نوع ال parameters اللي هستعملها
طب اشمعنا ؟ interface

لان ال object هيجرني اني اكتب implementation لما اجي اخد منه
وده كود ال interface

```
package com.example.database;
```

```
public interface on_recycler_view_item_click_listener
{
    void on_item_click(int car_id);
}
```

تاني خطوه واني اجي جوه الكلاس اللي فيه الداتا واحد object من ال interface
واجي في ال constructor بتاع الكلاس ده واحظ parameter من نوع ال

```
private on_recycler_view_item_click_listener listener;
public car_adapter(on_recycler_view_item_click_listener listener) {
    this.listener=listener;
}
```

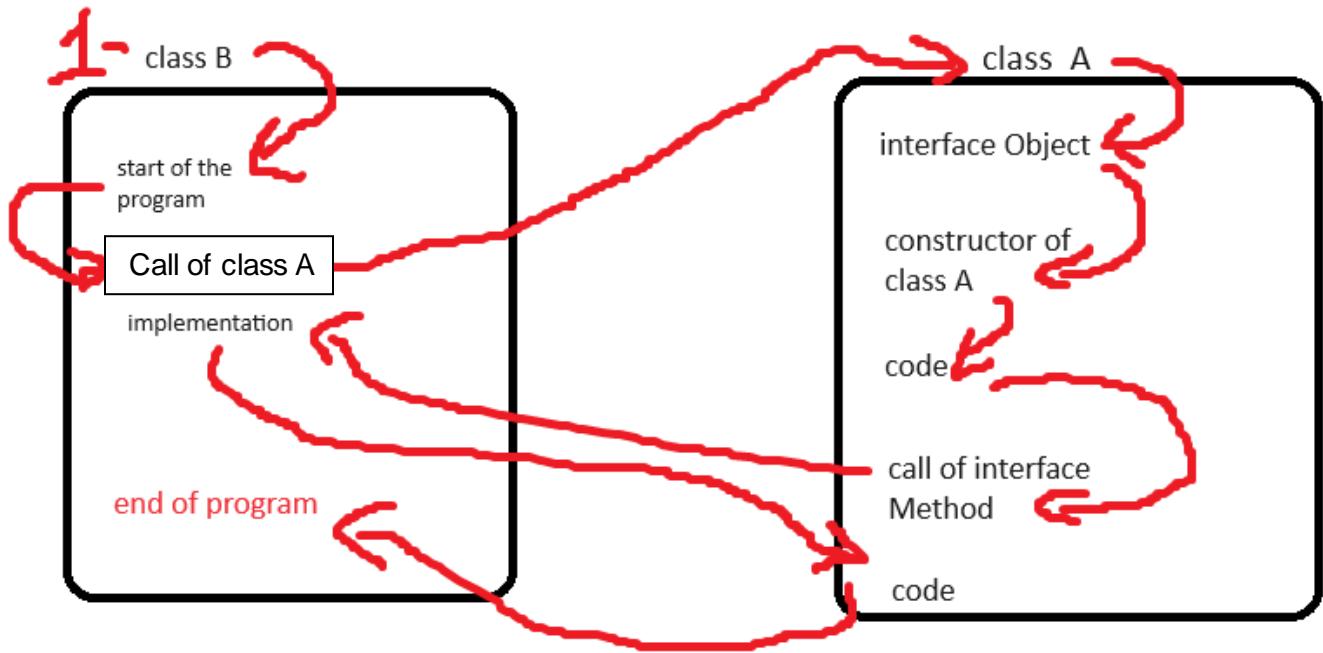
كده يبقى أي شخص هبيجي يستخدم الكلاس بتاعي لازم يديله object من ال interface وبناللي مضطر يعمل ال method بتاع ال implementation و ساعتها هيظهر له ال parameters اللي في ال method اللي انا أصلا عايز انقله الله عشان يكتب الكود بتاعه طب بالنسبة للكلاس؟

بالنسبة للكلاس فانا خزنت فيه ال object بتاع ال interface وال object اصبح جاهز وكل اللي تحتاج اعمله اني اجي استدعي ال method اللي في ال object ده

خد مثال تاني وافهمه بالراحه

<pre>public interface Printer { public void print(String message); }</pre>	<p>ده interface اسمه printer وفيه function</p>
<pre>public class PrinterDelegator { private Printer printer; public PrinterDelegator(Printer printer) { this.printer = printer; } public void print(String message) { printer.print(message); } }</pre>	<p>وده كلاس بيأخذ فيه object من ال constructor وبيعبيه عن طريق ال</p>
<pre>public class Main { public static void main(String[] args) { PrinterDelegator printerDelegator = new PrinterDelegator(new Printer(){ @Override public void print(String message) { //Write the implementation here } });</pre>	<p>وبعدين بيستدعي ال function عشان ينفذها هنا بقى في ال main بيأخذ object من الكلاس اللي اسمه printer delegator فيطلب منه بال interface من ال object بتاع ال implementation method اللي في interface</p>

ودي الخطوات اللي بيمشي فيها البرنامج



اظن كده فهمنا الشغل بيتم ازاي

الشغل بال #C# بيتم بنفس المنطق بس بطريقه مختلفه

ازاي ؟

قالك انك بدل ماتقدر تعمل في interface وتعمل get set وحوارات عمالك حاجه ابسط اسمها
delegate

ايه ال delegate دي؟

زي ما الحنا عارفين ان ال pointer عباره عن مخزن لـ address بتاع المتغير او ال object

قالك بقى ان ال delegate هيا عباره عن pointer عادي جدا بس بيشاور على method بدل ما يشاور على object

يعني بدل ماتعمل interface لا اعمل pointer يشاور عال method نفسها

وطريقتها انك بتكتب function من غير parameter اللي انت عايزه يتبعت بره الكلاس بس بتبعت معاه object بيكون عباره عن كلمة this واللي معناها انك بتبعت الكلاس اللي انت فيه عشان اللي هيسخدم ال delegate بتاعك يعرف هو جاي منين

يعني بتعمل function من غير implementation وبتجي قبل ال return type و بتكتب كلمة delegate

زي كده

```
[access modifier] delegate [return type] [delegate name]([parameters])
```

```
public delegate void DataBackEventHandler(Object sender,int PersonID);
```

بالسطر ده انت كده عرفت delegate على pointer بيشاور على method بس من غير implementation

(اعتبر نفسك عملت interface وال method بقائه لو هتقارنه بالمثال بتاع الاندرويد اللي شرحتهوك فوق)

طب ليه؟

لأن ال implementation هيكتب من بره الكلاس اصحي للكلام
بعدين هتاخذ object من ال ده pointer
(برضه اعتبر نفسك عرفت object من ال interface في المثال بتاع الاندرويد)

```
public DataBackEventHandler DataBack;
```

نافق حاجتين

اول حاجه انك تستدعي ال method في المكان اللي انت عايزة في الكلاس بتاعك وده بيتم عن طريق
انك تكتب السطر ده ومكان ال sender بتدليها الكلاس اللي انت فيه وبتبعد المتغير اللي انت عايزة
تبعنه

```
DataBack?.Invoke(this,PersonID);
```

طب يعني ايه invoke
اصير علي رزقك

ثاني حاجه ال implementation بتاعت ال method في الكلاس الثاني
هتعمل function عاديه هتحط فيها الكود اللي عايزة تنفذه وال parameters اللي كنت محددها سمي
ال method أي اسم المهم ان ال parameters تكون نفس ال parameters اللي حددتها فوق

```
private void Form2_DataBack(object sender, int PersonID)
{
    textBox1.Text = PersonID.ToString();
}
```

بس لما تيجي تشغيل الفورم الثاني هتكتب السطر ده واللي معناه انك تستدعي ال object اللي عملته من
ال method pointer وبتقوله بالله عليك خد ال

```
Form2 frm = new Form2();
frm.DataBack += Form2_DataBack;
frm.ShowDialog();
```

طيب لسه ماعرفناش ال invoke دي ايه وظيفتها
هنا بيقولك ان ال delegate دي معموله عشان يتم استدعائها من اكتر من مكان فلما بتكتب السطر ده

```
frm.DataBack += Form2_DataBack;
```

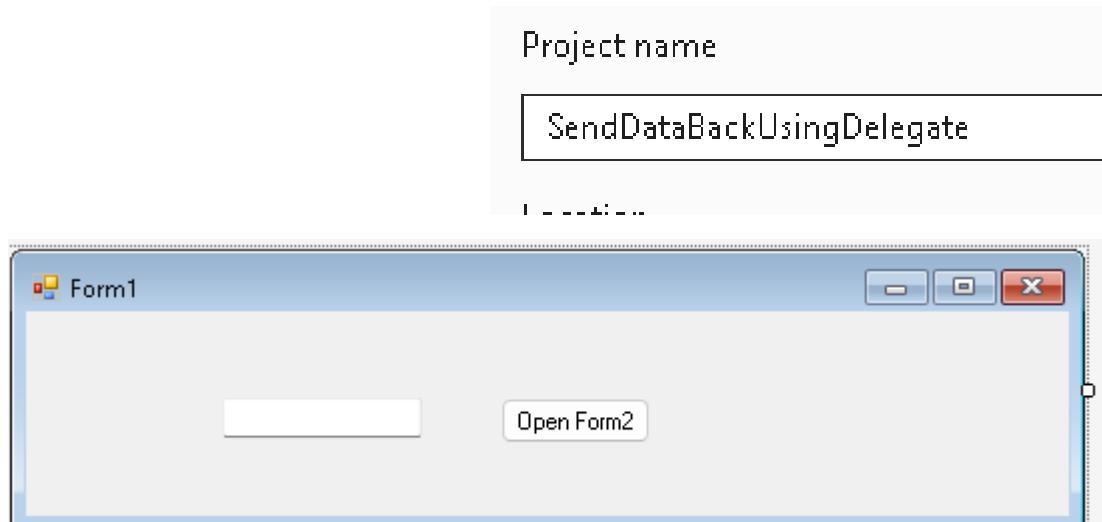
ال delegate بيضم ال method والاكراد اللي فيها لقايمة ال methods اللي عايزه تتنفذ

شوف الصورة اللي جايه دي انا عملت debug عشان اظهرلك اللي متخزن في ال object هتلقيه مخزن اسم ال method اللي هينفذها ولو ماكنتش زودت السطر اللي فوق قبل مافتح الفورم ماكنتش ال method هتنضاف وبالتالي ال delegate مش هيلاقى حاجه ينفذها فيأخذ اجازه عشان يرتاح

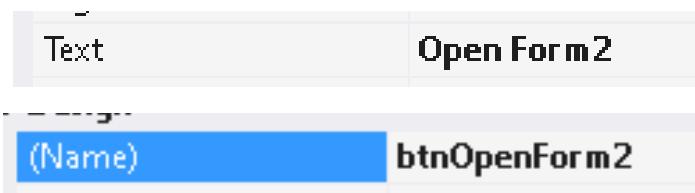
```
▶ public event DataBackEventHandler DataBack;
    ▶ DataBack {Method = {Void Form2_DataBack(System.Object, Int32)}} □
```

وبالتالي لما بستدعي ال invoke بيروح يدور علي كل ال methods اللي متخزنه في ال object وينفذها

تعالي نعمل تطبيق جديد باتنين فورم
اول فور هنحط فيه زرار و text box
وتاني فورم هنحط فيه label و زرار text box



دي خصائص الزرار





دي الخصائص بتاعت ال text box

Design

(Name)

txtPersonID

ConnectionString

Trans

دي الخصائص بتاعت الزرار

Text	SendDataBack
------	---------------------

وده الكود بتاع form2

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SendDataBackUsingDelegate
{
    public partial class Form2 : Form
    {
        public delegate void DataBackEventHandler(Object sender,int PersonID);
        public DataBackEventHandler DataBack;

        public Form2()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int PersonID = int.Parse(txtPersonID.Text) ;
            DataBack?.Invoke(this,PersonID);

            this.Close();
        }
    }
}

```

وده الكود بتاع form1

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SendDataBackUsingDelegate
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnOpenForm2_Click(object sender, EventArgs e)
        {

            Form2 frm = new Form2();
            frm.DataBack += Form2_DataBack;
            frm.ShowDialog();
        }

        private void Form2_DataBack(object sender, int PersonID)
        {
            textBox1.Text = PersonID.ToString();
        }
    }
}

```

User Controls

انت دلوقتي عارف ال controls اللي في ال windows app زي ال text box وال combo وغيره

هنا بقى بيقولك انك ممكن تعمل control جديد بيضم controls من اللي موجودين يعني مثلاً تعمل control بيكون فيه خمسه text box واربع ازرار وبيأخذ بيانات معينه بيعرضها بيفيدك الموضوع ده لو عندك اكتر من شاشه بتعرض فيها نفس المعلومات بتحطها في control وبتستدعيها في كل الأماكن زي كده

Person ID : 1

Name: Mohammed Sager Mussa Abu-Hadoud

National No: 12345 Date Of Birth: 11/6/1977

Gendor: Male Phone: 999876

Email: Msager@gmail.com Country: Jordan

Address: Amman Jubaiha

Person ID : 2

Name: Eslam

National No: 1234566 Date Of Birth: 12/31/2000

Gendor: Female Phone: 6543211

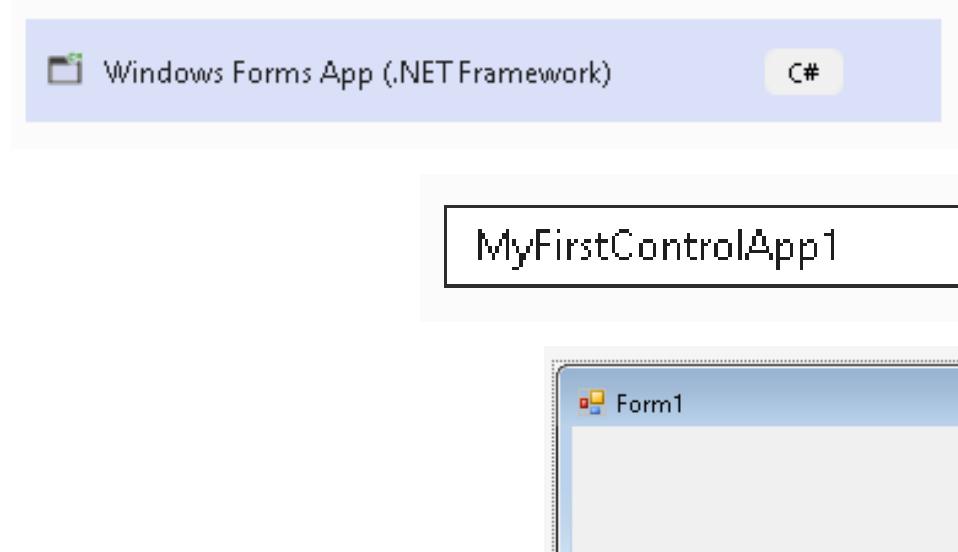
Email: e1 Country: Jordan

Address: address2

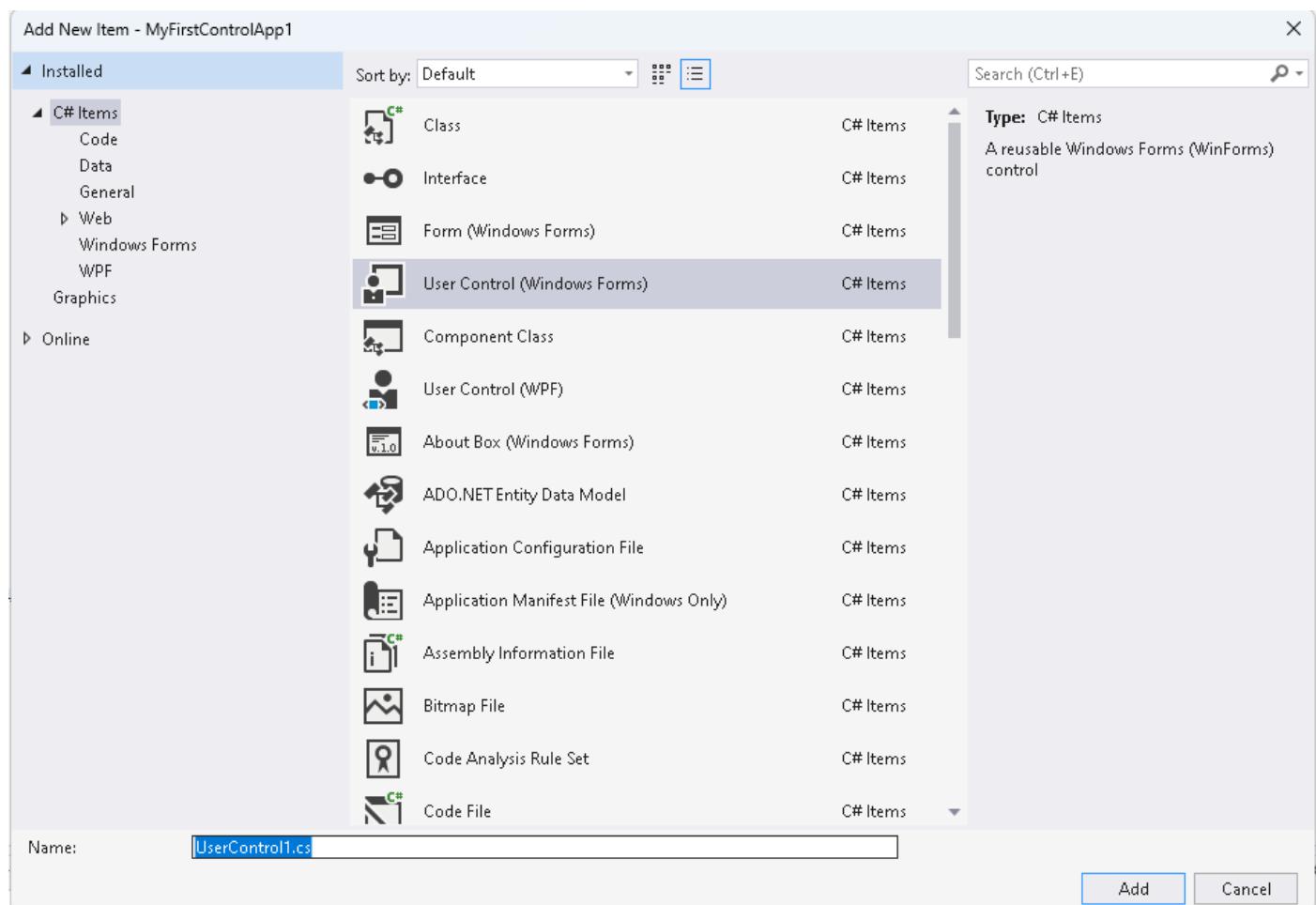
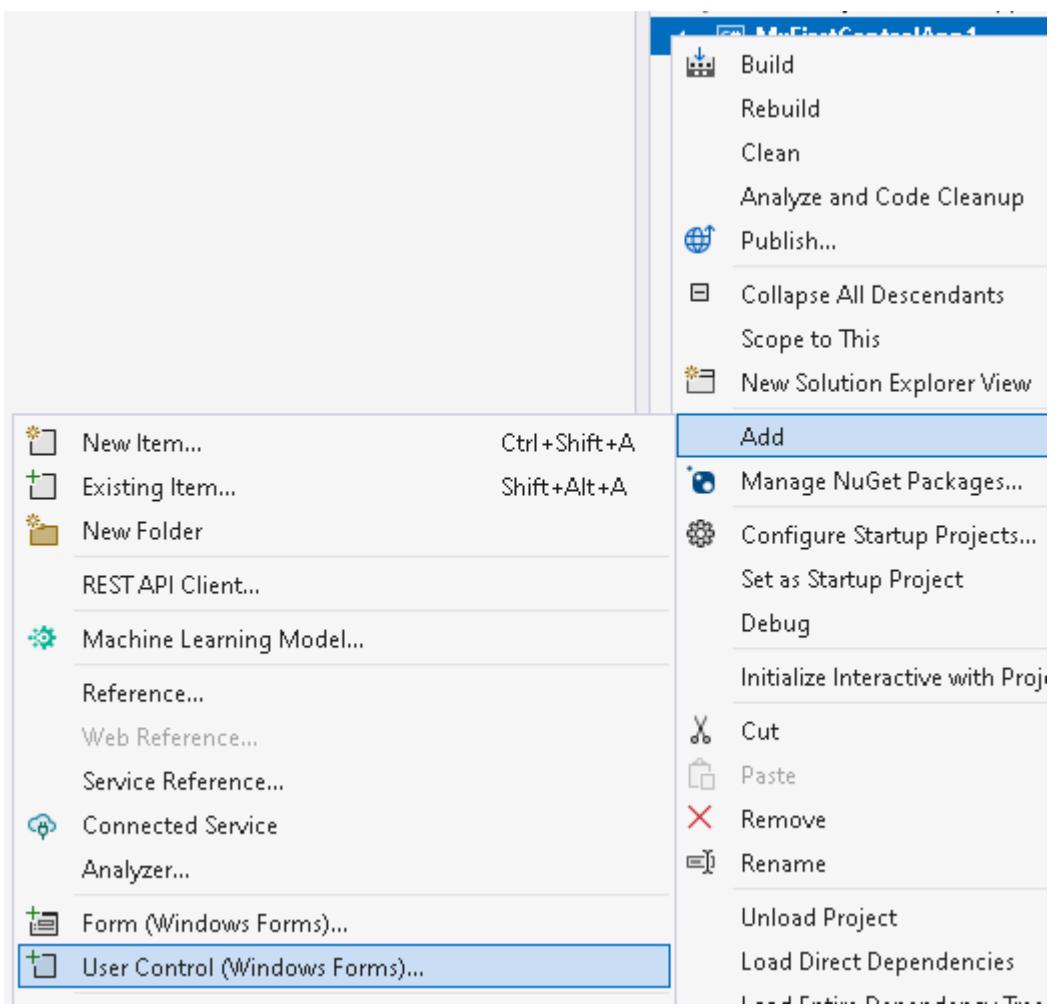
ال نوعين controls :

١ - وده بتسخدمه في برنامج واحد User Control
٢ - وده بتسخدمه في اكتر من مشروع Custom control

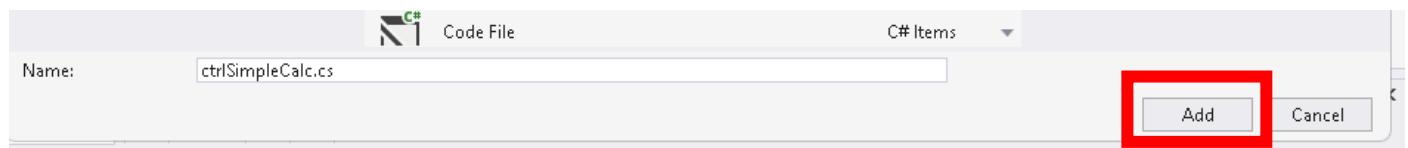
تعالي نعمل فورم جديد



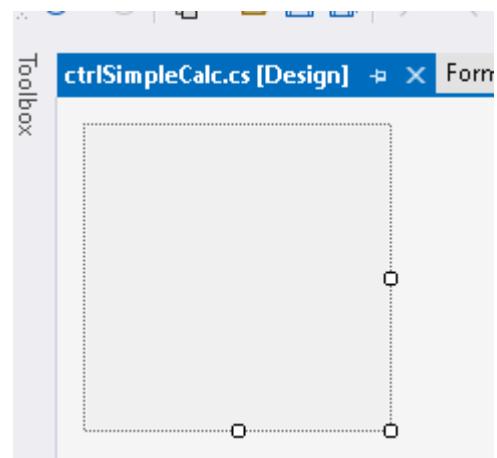
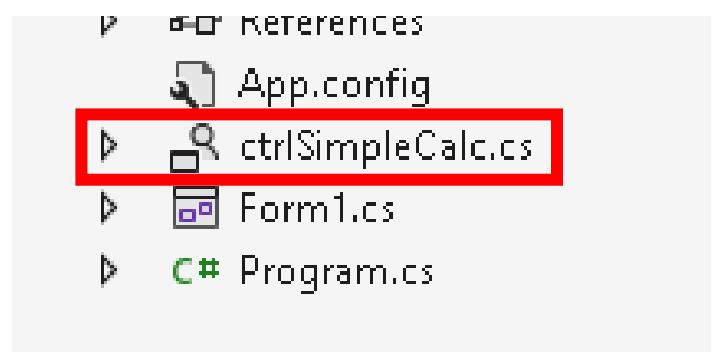
وبعدين هنضيف CONTROL جديد



ctrlSimpleCalc.cs



هتلacieh ضافهولك و هنتعامل معاه كأنه فورم عادي وتضيف فيه controls زي مانت عاوز

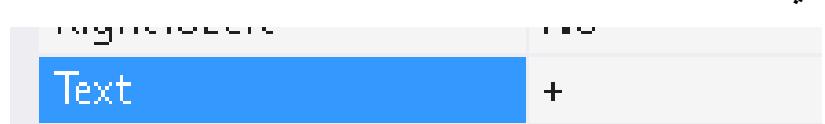


هنخلي حجم الخط ١٢

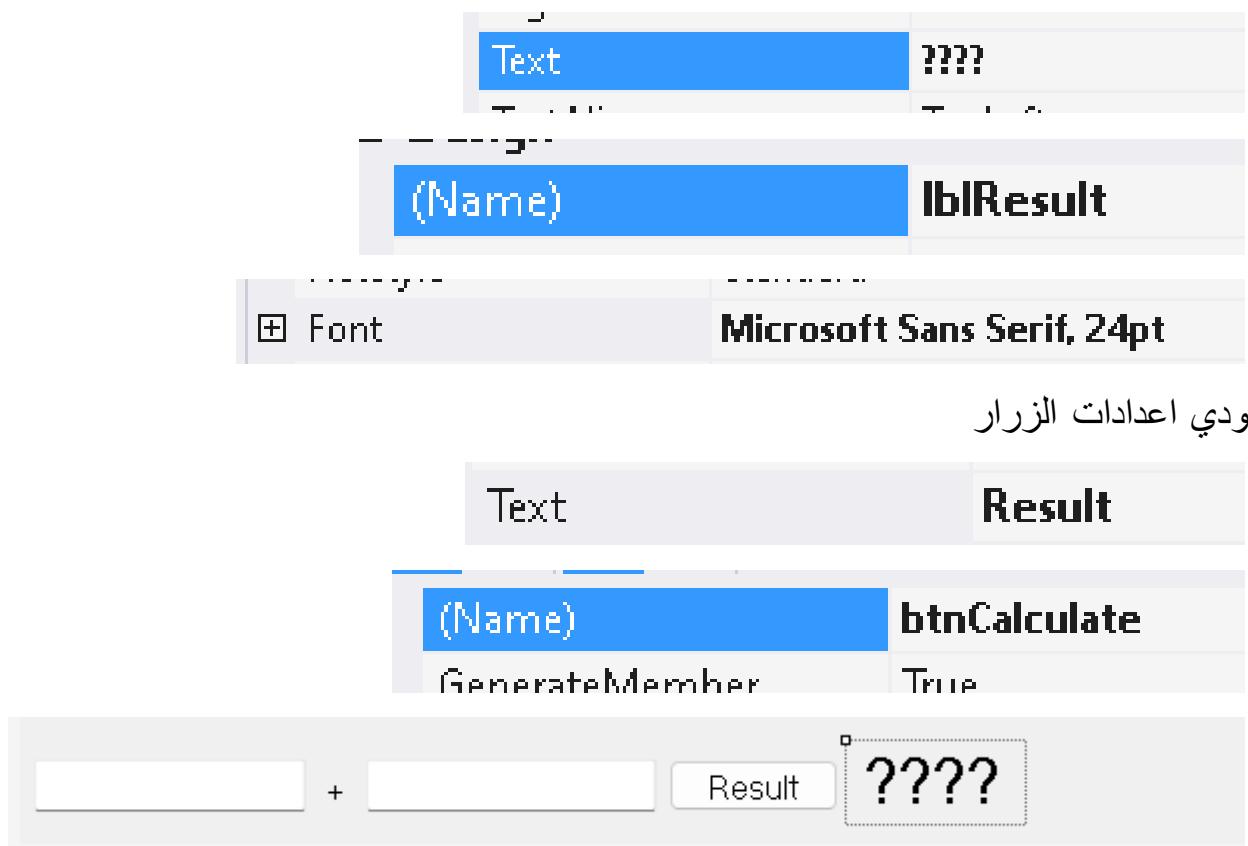
وبعدين هنضيف اتنين text box وزرار واتنين label



دي اعدادات label1



دي اعدادات label2



وده الكود بتاع ال CONTROL يدوب عملت EVENT عالزرار

```

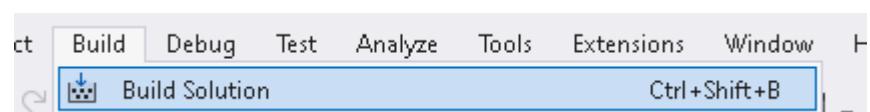
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MyFirstControlApp1
{
    public partial class ctrlSimpleCalc : UserControl
    {
        public ctrlSimpleCalc()
        {
            InitializeComponent();
        }

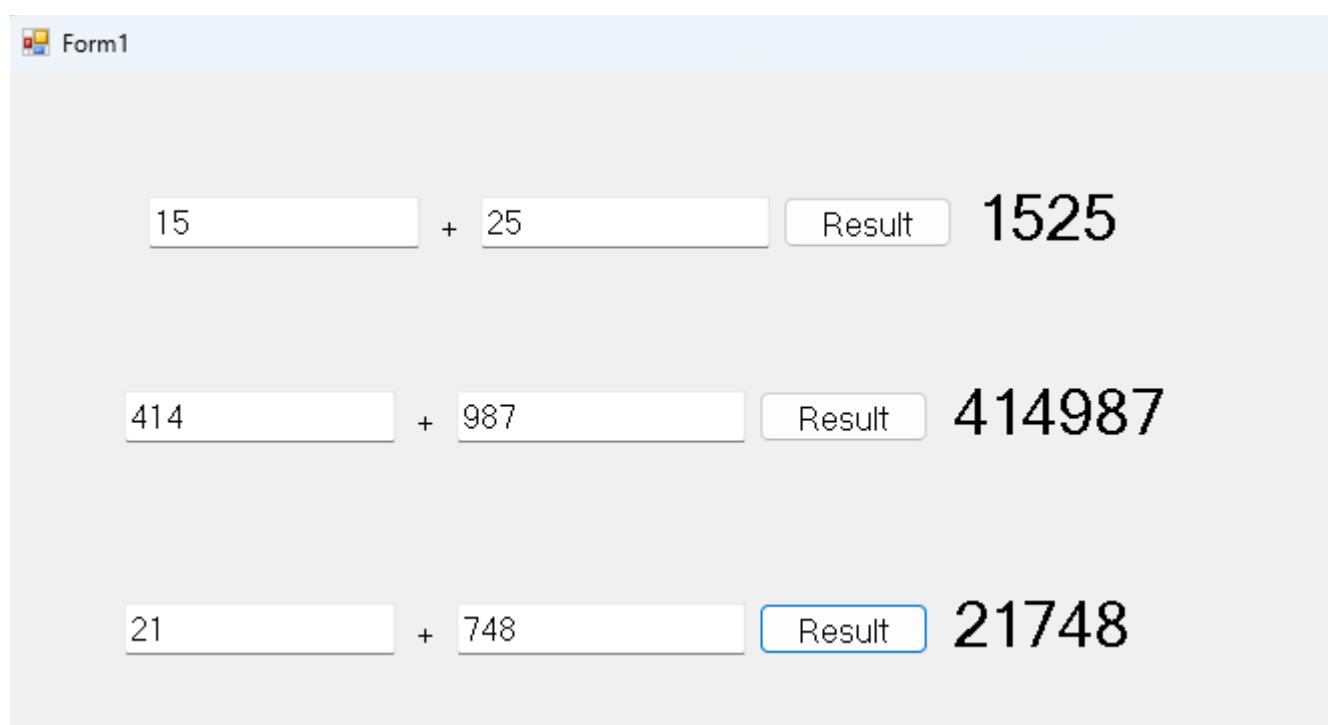
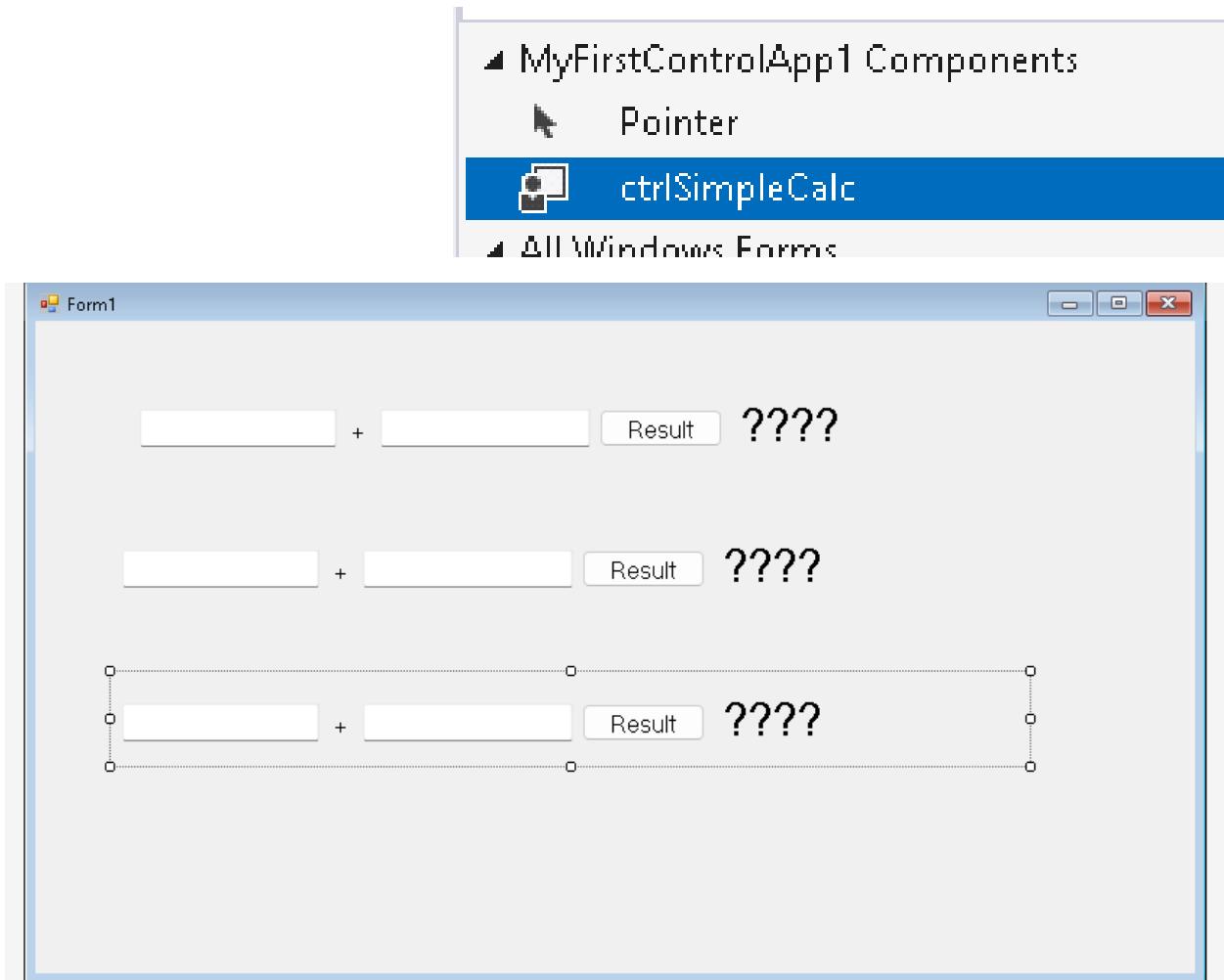
        private void btnCalculate_Click(object sender, EventArgs e)
        {
            lblResult.Text = int.Parse(textBox1.Text) + int.Parse(textBox2.Text).ToString();
        }
    }
}

```

هنروح عالفورم ونعمل build



هتلقيها في ال tool box



تعالی نعمل زرار یخفي ال control

_____ + _____

Result

????



_____ + _____

Result

????

_____ + _____

Result

????

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MyFirstControlApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            ctrlSimpleCalc1.Visible = false;
        }
    }
}
```

User Controls Expose Property

هنا بيقولك لو عايز توصل لاي عنصر جوه ال control بتاعك بتسخدم ال get وال set

طيب انا دلوقتي في المشروع بنطاع الدرس اللي فات عايز الزرار اللي بيحفي ال control عايزه يظهرلي message box فيها قيمة ال lblResult اللي جوه ال control

فهروح لل control واعمل متغير واعمله get وجوه ال هجيب القيمه اللي انا عايزها

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```

using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MyFirstControlApp1
{
    public partial class ctrlSimpleCalc : UserControl
    {
        public ctrlSimpleCalc()
        {
            InitializeComponent();
        }

        public float Result { get { return (float)Convert.ToDouble(lblResult.Text); } }

        private void btnCalculate_Click(object sender, EventArgs e)
        {
            lblResult.Text = int.Parse(textBox1.Text) + int.Parse(textBox2.Text).ToString();
        }
    }
}

```

بعدين هروح للفورم واعمل اللي انا عايزه

```

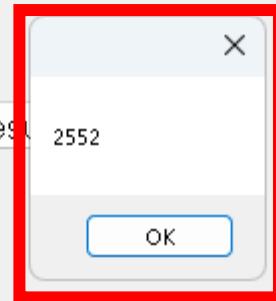
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MyFirstControlApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show(ctrlSimpleCalc1.Result.ToString());
        }
    }
}

```

25 + 52 Result 2552



button1

Simple Event With Parameter

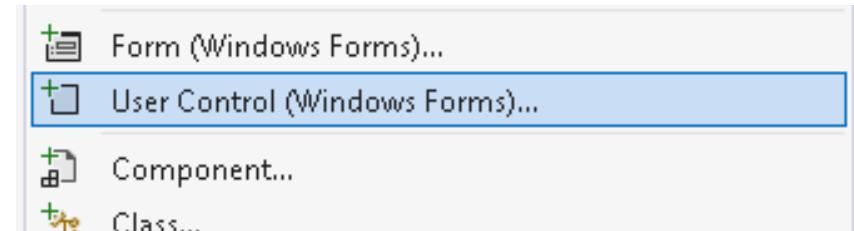
How to make a simple event in user control and send parameter with it?

الدرس اللي فات اتعلمـنا فيه ازاي نستقبل داتا من فورم بعد ما نفتحه
دلوقي احنا عاوزين نعمل EVENT زي اللي موجودين في الفورم زي الـ on load وـ on click
تعالي الأول نعمل تطبيق جديد

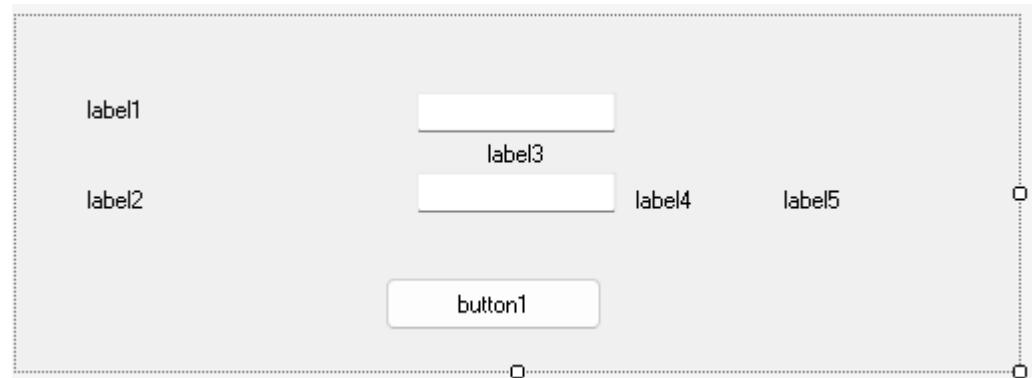
Project name

SimpleEventWithParameters

وبعدـين هنعمل control جديد



هنضيف اتنـيين text box و أربـعة label و زـرار



دي خصائص ال control

Appearance	
BackColor	ActiveCaption
Font	Microsoft Sans Serif, 12pt
Size	505, 162
Width	505
Height	162

ودي الخصائص بتاعت 1 label

Text	
Location	11, 17
X	11
Y	17
Margin	3, 0, 3, 0

ودي خصائص 2 label

Text	
Location	11, 81
Margin	3, 0, 3, 0

ودي خصائص 3 label

Text	
Font	Microsoft Sans Serif, 15.7
Location	228, 45
Margin	3, 0, 3, 0

ودي خصائص 4 label

Text	
Font	Microsoft Sans Serif, 15.7

Text	=
Text	Text
+ Location	400, 76
+ Margin	3, 0, 3, 0

وهي خصائص label5

Font	Microsoft Sans Serif, 21.7...
ForeColor	ControlText
Text	???
+ Location	430, 75
+ Margin	3, 0, 3, 0
Design	
(Name)	lblResult

وهي خصائص ال textBox1

(Name)	textNumber1
+ Location	105, 17
+ Margin	4, 5, 4, 5
+ MaximumSize	0, 0
+ MinimumSize	0, 0
+ Size	272, 26

وهي خصائص textbox2

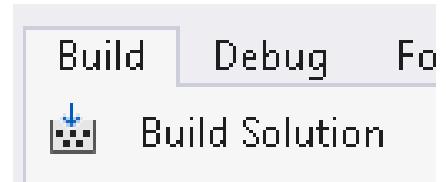
(Name)	textNumber2
+ Location	105, 75
+ Margin	4, 5, 4, 5
+ MaximumSize	0, 0
+ MinimumSize	0, 0
+ Size	272, 26

وهي خصائص الزرار

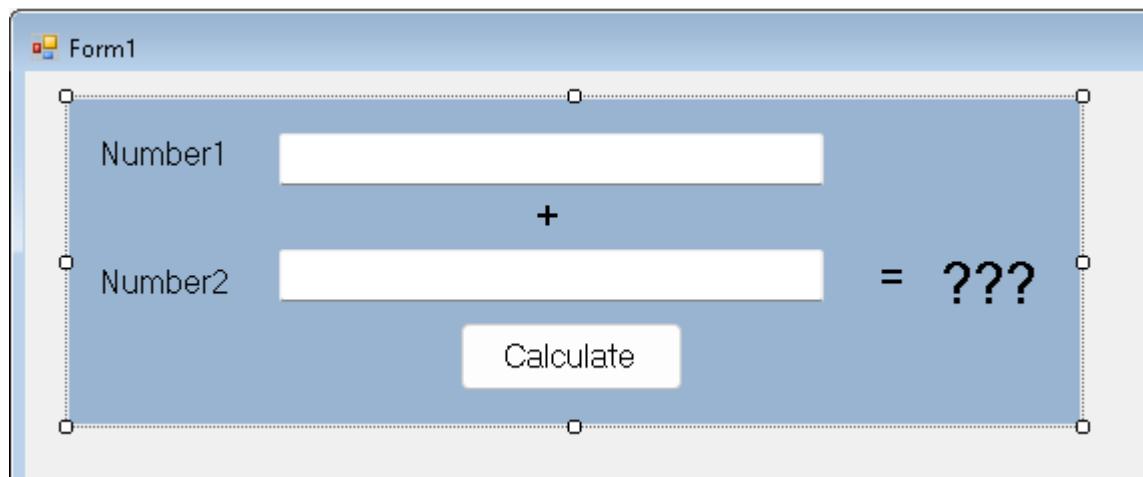
Text	Calculate
Text	Calculate

(Name)	btnCalculate
Location	195, 111
Margin	4, 5, 4, 5
MaximumSize	0, 0
MinimumSize	0, 0
Padding	0, 0, 0, 0
Size	112, 35

بعدين هنعمل build



ونروح للفورم نضيف فيه ال control ده



دلوقي احنا عاوزين نعمل event بحيث انه لما يخلص العملية الحسابيه بتاعته اكتب كود معين زي اني اطلع رساله

بس الكود ده هكتبه في الفورم

يعني عايز ال control اللي عملته يكون ليه event خاص بيه زي ما ال button ليه events كده الموضوع هنا زيه زي موضوع ال delegate مجرد pointer بيشاور علي method معينه ال迪 ال implementation method بتاعها المسؤول عنه هو اللي هيستخدم ال control مش اللي عمل ال control

اللي هتعرف تعمله في ال event هتعرف تعمله في ال delegate لأن الاتنين واحد الفرق بس انهم جابوا ال delegate وقعدول يضيفوا ويعدلوا عليه بحيث ان استخدامه يكون اسهل وأمن وعشان ماحدش ييجي يلعب في ال delegate بيوظها

ال event بتشتغل زي قنوات الراديو لو حابب تسمع قناة معينة بتضبط التردد عليها

فال event هو عباره عن notification يتم ارساله لكل الناس اللي مشتركين فيه والكلas اللي بيبيعut ال notification دي اسمه publisher يبعنهم داتا معينه في وقت معين وكل الناس المشتركين فيه واللي اسمهم subscribers لو حد فيهم عاييز يعمل حاجه بيعملها

طيب عشان اعرف event لازم اعرف delegate الأول لانه مبني عليه في ال C# هما عاملين حاجه اسمها built in delegates ودي عباره عن مجموعة من ال delegates هما عرفوها مسبقا عشان يسهلوا عليك الموضوع اللي يهمنا منهم واحد اسمه action وبيأخذ منك data type عشان يحدد ال data type اللي في ال parameters

وكل اللي عمله اني هكتب public event action وبعدين هحدد datatype من نفس نوع الداتا اللي عاييز انقلها بره اللي هيستخدم ال control يعني هعرف event من النوع و هكتب اسم لل event ده

public event Action<int> onCalculationComplete;	ده تعريف ال event
---	-------------------

من التسهيلات اللي بيسهلها عليك ال event انه بيخليك بدل ما تستدعي ال invoke لا انت تستدعي اسم ال event نفسه وتديله ال parameter اللي انت عاييزها

if (onCalculationComplete != null) { onCalculationComplete(Result); }	وهنا يستدعيه واديله القيمه لو مش ب null
---	---

بس كده ده كل الحوار
وده كود ال control

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
  
namespace SimpleEventWithParameters  
{  
    public partial class MyUserControl : UserControl  
    {  
        public MyUserControl()  
        {  
            InitializeComponent();  
        }  
  
        public event Action<int> onCalculationComplete;  
  
        private void btnCalculate_Click(object sender, EventArgs e)
```

```

    {
        int Result = Convert.ToInt32(textNumber1.Text) + Convert.ToInt32(textNumber2.Text);
        lblResult.Text = Result.ToString();

        if (onCalculationComplete != null) {
            onCalculationComplete(Result);
        }
    }
}

```

وده كود ال form

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SimpleEventWithParameters
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void myUserControl1_onCalculationComplete(int obj)
        {
            int Results = obj;
            MessageBox.Show("Results = " + obj);
        }
    }
}

```

هنا الأستاذ عمل تعديل بسيط وهو انه بدل مايستدعي ال event على طول في ال control لا هوa عمل استدعاء جواه بحيث انه مايشغلش ال event عمال على بطاطا

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SimpleEventWithParameters
{
    public partial class MyUserControl : UserControl
    {

        // Define a custom event handler delegate with parameters
    }
}

```

```

public event Action<int> OnCalculationComplete;
// Create a protected method to raise the event with a parameter
protected virtual void CalculationComplete(int PersonID)
{
    Action<int> handler = OnCalculationComplete;
    if (handler != null)
    {
        handler(PersonID); // Raise the event with the parameter
    }
}

public MyUserControl()
{
    InitializeComponent();
}

private void btnCalculate_Click(object sender, EventArgs e)
{
    int Result = Convert.ToInt32(txtNumber1.Text) + Convert.ToInt32(txtNumber2.Text);
    lblResult.Text = Result.ToString();

    if (OnCalculationComplete != null)
        // Raise the event with a parameter
        CalculationComplete(Result);
}
}

```

لاحظ انك لو فتحت ال properties بتاعت ال control لما ضيفته للفورم هتلافي ال event بتاعك



Simple Event With Parameters Using Arguments

دلوقي احنا عازين نعمل standard event يعني event زي اللي بنلاقيه في أي ال control بتاعته بيكون فيها source واللي بيقولك ال event ده جاي منين وال e اللي هيا ال method بتاعته بيكون فيها شوية متغيرات بتقدر تستخدمها في الكود بتاعك event argument

في نفس المشروع اللي فات انا مش عايز ابعث ال result بس لا انا عايز ابعث الرقم الأول والرقم الثاني والنتيجه وكمان عايز ابعث وأقول ال event ده مين اللي استدعااه او جاي منين

تعالي ناخد الموضوع واحده واحده

احنا دلوقي عازين نبعث اكتر من متغير الحل اتنا نجمع كلهم في كلاس واحد

فهنعمل كلاس عادي جدا جوه الكلاس بتاعنا وهنعمل constructor و get لكل متغير من المتغيرات اللي فيه

الكلas ده لازم نخليه يورث من كلاس تاني اسمه event args

```
public class CalculationCompleteEventArgs : EventArgs {
```

```
    public int Result { get; }
    public int Val1 { get; }
    public int Val2 { get; }

    public CalculationCompleteEventArgs(int Results, int Val1, int Val2) {
        this.Result = Results;
        this.Val1 = Val1;
        this.Val2 = Val2;
    }
}
```

بعدين هنعمل event عادي جدا بس ال delegate بتاعه مش هيكون action هيكون data type هندلره اسم الكلاس اللي عملناه

```
public event EventHandler<CalculationCompleteEventArgs> OnCalculationComplete;
```

وبعدين ؟

وبعدين بتكمel الكود بتاعك عادي بس فيه ملاحظه صغيره
وهيأ انك هتحتاج تستعدي ال method من خلال ال invoke زي ماكنت بتعمل مع ال delegate العاديه

فالحنا حطينا الخطوه دي في method لوحدها

```
protected virtual void RaiseOnCalculationComplete(CalculationCompleteEventArgs e)
{
    OnCalculationComplete?.Invoke(this, e);
}

public void RaiseOnCalculationComplete(int Results, int Val1, int Val2) {
    RaiseOnCalculationComplete(new CalculationCompleteEventArgs(Results, Val1, Val2));
}

private void btnCalculate_Click(object sender, EventArgs e)
{
    int Val1, Val2;
    Val1 = Convert.ToInt32(textNumber1.Text);
    Val2 = Convert.ToInt32(textNumber2.Text);

    int Result = Val1 + Val2;

    lblResult.Text = Result.ToString();

    if (OnCalculationComplete != null) {
        RaiseOnCalculationComplete(Result, Val1, Val2);
    }
}
```

بس بعد كده هنروح للفورم نستدعي ال button زى ماكنا بنعمل مع ال event

```
using System.Windows.Forms;

namespace SimpleEventWithParameters
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void myUserControl1_CalculationComplete(object sender, MyUserControl.CalculationCompleteEventArgs e)
        {
            MessageBox.Show($"Results={e.Result},val1= {e.Val1},val2= {e.Val2}");
        }
    }
}
```

وده كود ال control كله

```
using System;
using System.Windows.Forms;

namespace SimpleEventWithParameters
{
    public partial class MyUserControl : UserControl
    {
        public MyUserControl()
        {
            InitializeComponent();
        }

        public class CalculationCompleteEventArgs : EventArgs
        {

            public int Result { get; }
            public int Val1 { get; }
            public int Val2 { get; }

            public CalculationCompleteEventArgs(int Results,int Val1,int Val2)
            {
                this.Result = Results;
                this.Val1 = Val1;
                this.Val2 = Val2;
            }
        }

        public event EventHandler<CalculationCompleteEventArgs> OnCalculationComplete;

        protected virtual void RaiseOnCalculationComplete(CalculationCompleteEventArgs e)
        {
            OnCalculationComplete?.Invoke(this, e);
        }
    }
}
```

```

public void RaiseOnCalculationComplete(int Results, int Val1, int Val2) {
    RaiseOnCalculationComplete(new CalculationCompleteEventArgs(Results, Val1, Val2));
}

private void btnCalculate_Click(object sender, EventArgs e)
{
    int Val1, Val2;
    Val1 = Convert.ToInt32(textNumber1.Text);
    Val2 = Convert.ToInt32(textNumber2.Text);

    int Result = Val1 + Val2;

    lblResult.Text = Result.ToString();

    if (OnCalculationComplete != null) {
        RaiseOnCalculationComplete(Result, Val1, Val2);
    }
}
}
}

```

Temperature Change Event Example

هنعمل تطبيق بسيط عال event بال console application
تعالي نعمل واحد فاضي



بص كده عالكلاس ده

```

public class Thermostat {
    private double OldTemperature;
    private double CurrentTemperature;

    public void SetTemperature(double NewTemperature) {

        if (NewTemperature != CurrentTemperature) {
            OldTemperature = CurrentTemperature;
            CurrentTemperature = NewTemperature;
        }
    }
}

```

احنا هنا عملنا كلاس اسمه thermostat الكلاس ده فيه درجة الحرارة القديمه ودرجة الحراره الجديد
و فيه property set بتخليني اقدر اعدل علي درجات الحراره بحيث لما استدعي ال set واديها درجه حراره اخلي ال old degree تخزن في ال current degree والدرجة اللي انا اديتها يخزنها في ال current

ايوه عاييز ايه يعني؟

انا دلوقتي عاييز لما اغیر في درجة الحرارة اعرض رسالة عالشاشة تقولي درجة الحرارة قبل التعديل وبعد التعديل والفرق بين الدرجتين

بس عاييز الرسالة دي تظهر لو درجة الحرارة فعلاً اتغيرت يعني لو كانت ال new لاتساوي ال current بس يعني لو درجة الحرارة الحاليه كانت ١٠ وانا قولته غيرها يخليها ١٠ مايظهرش الرسالة وعايز اعمل الرسالة دي بره الكلاس بتاع ال thermostat

اقدر اعمل ده بال if statement عادي بس افرض انه في البرنامج بتاعنا فيه اكتر من مكان واكتر من كود بيعتمدوا على تغير درجة الحرارة هقدر اعمل if statement كل شوية؟

اما اعمل ايه؟

اعمل event امال هما اختروعها ليه

طب يلا بینا نشوف هنعملها ازاي

اول حاجه احنا عشان نعرض الرسالة بتاعت الدوجه القديمه والجديده احنا عاززين نوصل للقيم بتاعهم
عشان كده هنعمل كلاس يجمعهم كلهم والكلاس ده هنخليه يورث من event args

```
public class TemperatureChangedEventArgs:EventArgs {  
    public double OldTemperature { get; }  
    public double NewTemperature { get; }  
    public double Difference { get; }  
  
    public TemperatureChangedEventArgs(double OldTemperature, double NewTemperature) {  
        this.OldTemperature = OldTemperature;  
        this.NewTemperature = NewTemperature;  
        this.Difference = NewTemperature - OldTemperature;  
    }  
}
```

كده أي حد هيأخذ object من الكلاس هيحتاج يدخل درجات الحرارة وهو هيحسب لوحده الفرق
بعدين هنعرف event في كلاس ال thermostat وال event ده هيستغل محطة راديو بتبث موجات
الاذاعه

```
public event EventHandler<TemperatureChangedEventArgs>  
TemperatureChanged;
```

وبعدين هنعمل method وظيفتها انها تشغيل event من خلال ال invoke ال هتطلب
مننا parameter من نوع الكلاس هنقوم نخلي ال method تطلبه لك

```
private void OnTemperatureChanged(TemperatureChangedEventArgs e) {  
    TemperatureChanged?.Invoke( this,e);  
}
```

احنا عملناها **private** عشان مش عايزيين حد يوصلها بالطريقة دي

اماال هنوصلها ازاي ؟

هنعمل **method** تانيه بتطلب درجات الحراره وتسديعها زي كده

```
protected virtual void OnTemperatureChanged(double OldTemperature, double CurrentTemperature) {  
    OnTemperatureChanged(new TemperatureChangedEventArgs(OldTemperature, CurrentTemperature));  
}
```

بعدين هنشغل ال **event** في ال **property set**

```
public void SetTemperature(double NewTemperature) {  
  
    if (NewTemperature != CurrentTemperature) {  
        OldTemperature = CurrentTemperature;  
        CurrentTemperature = NewTemperature;  
  
        OnTemperatureChanged(OldTemperature, CurrentTemperature);  
    }  
}
```

كده أي حد هييجي يعدل علي درجة الحرارة الحاليه هيشفوف هل الدرجه اللي جت مختلفه عن الدرجة الحاليه ولا لال لا لو مختلفه هيعمل تعديل عالمتغيرات وي العمل بث مباشر

ازاي هيعمل بث مباشر ؟

اماال احنا كنا بنشرح فيه أي ؟

ماالنا قولنا ان ال **pointer** هيبدأ يلف علي كل **address** يشاور عليه ويشغله

طب ازاي هنخلي ال **pointer** يشاور علي ال **address** ؟

ده اللي جاي بقى

دلوقي احنا عايزيين نعمل كلاس كل وظيفته انه يعرض رسالة معينه لما يحصل التغيير في درجة الحرارة

فكل اللي احنا محتاجينه اتنا نعمل **method** خاصه بعرض الرساله وعاوزين نخلي ال **pointer** اللي في الكلاس بتاع ال **thermostat** يشاور علي ال **method** دي

ده كود ال **method** وخليل بالك انها لازم تأخذ نفس ال **parameters** اللي بيأخذها ال **event**

وطبعاً بالك انك هنا بتعمل **implementation** مش بتشغل ال **method** لأن التشغيل جاي منين ؟

ايوه التشغيل جاي من كلاس ال **thermostat** يعني زرار ال **power** في ال **thermostat**

```
public void HandleTemperatureChange(object sender, TemperatureChangedEventArgs e) {  
    Console.WriteLine("\n\nTemperature changed:");  
    Console.WriteLine($"Temperature changed from {e.OldTemperature}°C");
```

```

Console.WriteLine($"Temperature changed to {e.NewTemperature}°C");
Console.WriteLine($"Temperature Difference is {e.Difference}°C");
}

```

وهنا احنا بنخلي ال pointer يشاور علي ال method

```

public void Subscribe(Termostat thermostat) {
    thermostat.TemperatureChanged += HandleTemperatureChange;
}

```

هنجي في ال main ناخد object من كلاس ال thermostat و بعدين هنسندعي ال subscribe عشان يخلي ال pointer عال subscribe عشان يشاور علي ال method

وبعدين من كلاس ال thermostat هنقدر نغير في الحرارة براحتنا ولو كانت درجات الحرارة مختلفه فعلا هيعرض الرسالة

```

public class Program
{
    static void Main(string[] args)
    {
        Thermostat thermostat = new Thermostat();
        Display display = new Display();
        display.Subscribe(thermostat);

        thermostat.SetTemperature(5);
        thermostat.SetTemperature(10);
        thermostat.SetTemperature(10);
        thermostat.SetTemperature(10);
        thermostat.SetTemperature(50);

        Console.ReadLine();
    }
}

```

وده الكود كله

```

using System;

namespace TemperatureChangeEventExample
{

    public class TemperatureChangedEventArgs : EventArgs {
        public double OldTemperature { get; }
        public double NewTemperature { get; }
        public double Difference { get; }

        public TemperatureChangedEventArgs(double OldTemperature, double NewTemperature) {
            this.OldTemperature = OldTemperature;
            this.NewTemperature = NewTemperature;
            this.Difference = NewTemperature - OldTemperature;
        }
    }

    public class Thermostat {
        public event EventHandler<TemperatureChangedEventArgs> TemperatureChanged;
    }
}

```

```
private double OldTemperature;
private double CurrentTemperature;

private void OnTemperatureChanged(TemperatureChangedEventArgs e) {
    TemperatureChanged?.Invoke( this,e);
}

protected virtual void OnTemperatureChanged(double OldTemperature, double CurrentTemperature) {
    OnTemperatureChanged(new TemperatureChangedEventArgs(OldTemperature,CurrentTemperature));
}

public void SetTemperature(double NewTemperature) {

    if (NewTemperature != CurrentTemperature){
        OldTemperature = CurrentTemperature;
        CurrentTemperature = NewTemperature;

        OnTemperatureChanged(OldTemperature, CurrentTemperature);
    }
}

}

public class Display {

    public void Subscribe(Thermostat thermostat) {
        thermostat.TemperatureChanged += HandleTemperatureChange;
    }

    public void HandleTemperatureChange(object sender, TemperatureChangedEventArgs e) {
        Console.WriteLine("\n\nTemperature changed:");
        Console.WriteLine($"Temperature changed from {e.OldTemperature}°C");
        Console.WriteLine($"Temperature changed to {e.NewTemperature}°C");
        Console.WriteLine($"Temperature Difference is {e.Difference}°C");
    }
}

public class Program
{
    static void Main(string[] args)
    {
        Thermostat thermostat = new Thermostat();
        Display display = new Display();
        display.Subscribe(thermostat);

        thermostat.SetTemperature(5);
        thermostat.SetTemperature(10);
        thermostat.SetTemperature(10);
        thermostat.SetTemperature(10);
        thermostat.SetTemperature(50);

        Console.ReadLine();
    }
}
```

احنا كان ممكن نعمل تعديل بحيث ان ال **thermostat** يكون جوه كلاس ال **object** بتاع ال **display** زى كده بس الطريقة اللي فوق dynamic اكتر هتلاحظ ده في الدرس الجاي وطبعا انت بتحدد الطريقة حسب احتياجاتك

```
public class Display {  
    public Display() {  
        thermostat.TemperatureChanged += HandleTemperatureChange;  
    }  
  
    Thermostat thermostat = new Thermostat();  
    public void SetTemperature(double NewTemperature) {  
  
        thermostat.SetTemperature(NewTemperature);  
  
    }  
  
    public void HandleTemperatureChange(object sender, TemperatureChangedEventArgs e) {  
        Console.WriteLine("\n\nTemperature changed:");  
        Console.WriteLine($"Temperature changed from {e.OldTemperature}°C");  
        Console.WriteLine($"Temperature changed to {e.NewTemperature}°C");  
        Console.WriteLine($"Temperature Difference is {e.Difference}°C");  
    }  
}  
  
public class Program {  
    static void Main(string[] args) {  
        Display display = new Display();  
  
        display.SetTemperature(5);  
        display.SetTemperature(10);  
        display.SetTemperature(10);  
        display.SetTemperature(10);  
        display.SetTemperature(50);  
  
        Console.ReadLine();  
    }  
}
```

بس كده على مكان السطر ده في الكود اللي فوق

```
thermostat.TemperatureChanged += HandleTemperatureChange;
```

لو لاحظت هتلافقيني حاطه في ال **constructor** بتاع الكلاس

طيب ليه ماسبيتهوش في الشارع في الكلاس

عشان قالك انه لازم يتحط في **method** عشان يضمن ان الكود هيتنفذ ال **method** دي تكون **constructor** او تكون **setter** او تكون **constructor** لوحدها كده تكون زي ماتكون المهم انها تتحط في مكان يضمن تنفيذها

طيب ليه ماحطيناهاش مع ال **set temperature** ؟

لأنه في كل مره هستدعي ال `set temperature` هيتم اضافه ال `pointer` وبالتالي
هيتم تنفيذ الكود بتاعي اكتر من مره زي كده

شوف التعديل ده

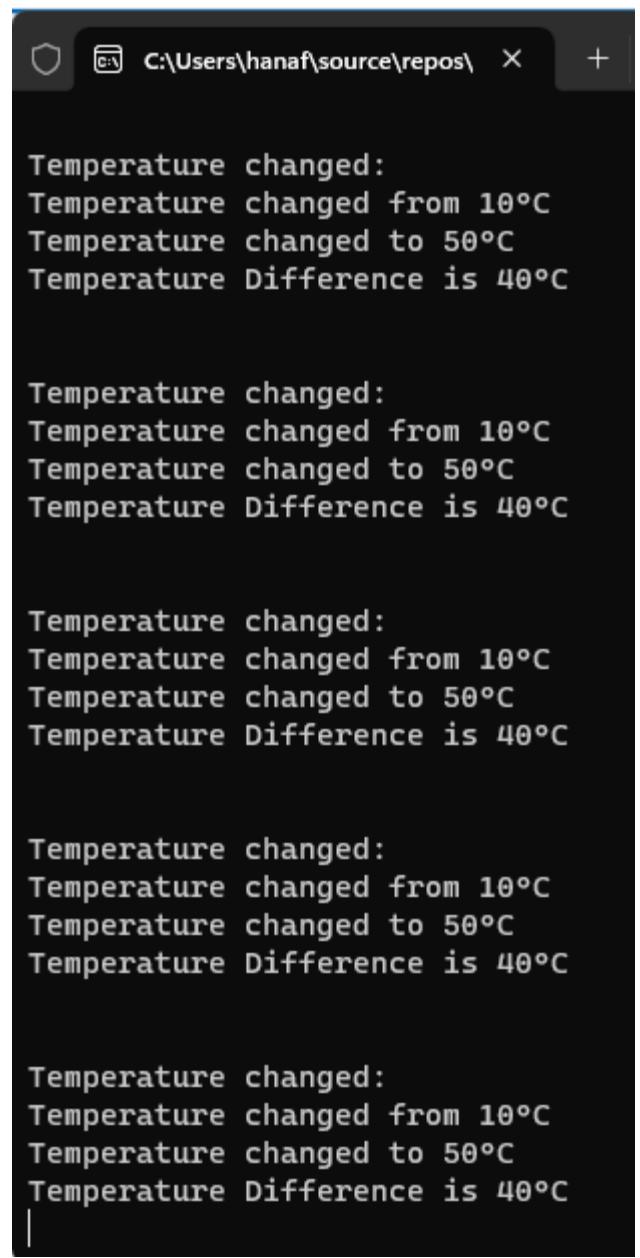
```
public class Display {
    public Display() {
        //thermostat.TemperatureChanged += HandleTemperatureChange;
    }

    Thermostat thermostat = new Thermostat();
    public void SetTemperature(double NewTemperature) {
        thermostat.TemperatureChanged += HandleTemperatureChange;
        thermostat.SetTemperature(NewTemperature);

    }

    public void HandleTemperatureChange(object sender, TemperatureChangedEventArgs e) {
        Console.WriteLine("\n\nTemperature changed:");
        Console.WriteLine($"Temperature changed from {e.OldTemperature}°C");
        Console.WriteLine($"Temperature changed to {e.NewTemperature}°C");
        Console.WriteLine($"Temperature Difference is {e.Difference}°C");
    }
}
```

وشوف نتيجته



```
Temperature changed:  
Temperature changed from 10°C  
Temperature changed to 50°C  
Temperature Difference is 40°C  
  
Temperature changed:  
Temperature changed from 10°C  
Temperature changed to 50°C  
Temperature Difference is 40°C  
  
Temperature changed:  
Temperature changed from 10°C  
Temperature changed to 50°C  
Temperature Difference is 40°C  
  
Temperature changed:  
Temperature changed from 10°C  
Temperature changed to 50°C  
Temperature Difference is 40°C  
  
Temperature changed:  
Temperature changed from 10°C  
Temperature changed to 50°C  
Temperature Difference is 40°C
```

عشان كده حطيناه مع ال `constructor` بحيث انه يتنفذ مره واحدة اثناء انشاء ال `object` الموضوع ده مافكركش بحاجه؟

افكرك انا

في الشرح بتاع الكورس اللي فات انا كنت قولتلك ان الفورم اللي بنعمله مقسم نصين نص بنكتب فيه الكود بتاعنا والنص الثاني خاص بالديزاین وتعريف ال `controls` والخواص بتاعتتها

من ضمن الخواص كان السطر بتاع ال `event`

اللي لما كنا بنحذف الكود بتاع ال `event` يطلعنا الخطأ ده

To prevent possible data loss before loading the designer, the following errors must be resolved:

1 Error

The designer cannot process unknown name 'btnClose_Click' at line 157. The code within the method 'InitializeComponent' is generated by the designer and should be changed and try opening the designer again.

Ignore and Continue

Instances of this error (1)

1. DVLD frmAddEditPerson.Designer.cs Line:157 Column:1 Show Call Stack

Help with this error

Could not find an associated help topic for this error. [Check Windows Forms Design-Time error list](#)

Forum posts about this error

[Search the MSDN Forums for posts related to this error](#)

وكنا بندوس على الخطأ عشان يفتحانا كود نحذف منه السطر ده

```
154     this.btnClose.Text = "Close";
155     this.btnClose.TextAlign = System.Drawing.ContentAlignment.MiddleRight;
156     // ...
157     this.btnClose.Click += new System.EventHandler(this.btnClose_Click);
158     // ...
159     // lblRemove
```

طب فاكر اني قولتلك ان المكان ده هو ال **method implementation** اللي اسمها **initialize components** اللي هي دي؟

```
public frmAddEditPerson(int PersonID)
{
    InitializeComponent();
    this.PersonID = PersonID;

    if (PersonID>0) { this.Mode = enMode.Update; } else { this.Mode = enMode.AddNew; }
}
```

يعني عايز تقول ايه قررتني؟

عايز اقولك انه ال **constructor** بيتاً اضافتها في ال **events** بتابع الفورم عشان يسهل عليك الاستخدام وعشان ده المكان اللي هيضمن فيه ان ال **events** كلها هيتم اضافتها وهيتاً اضافتها مره واحده بس

شوف ملف الشرح بتاعي بتاع الكورس اللي فات عشان تفهم اكتر هتلاقيه في درس ال main form

News Publisher Subscriber Example

دلوقي عازين نعمل خدمه إخبارية مكونه من **publisher** و **subscriber** واي حد حابب يفعل الاشعارات بيفعها واي حد عايز يلغى الاشتراك من ناشر معين يلغيه

اول حاجه هيا اننا نعمل كلاس خاص بالخبر نفسه بس المرادي مش هنخليه يورث من عشان عازين نبعث الكلاس كله علي بعضه

```
public class NewsArticle
{
    public string Title { get; }
    public string Content { get; }
```

```

public NewsArticle(string Title, string Content)
{
    this.Title = Title;
    this.Content = Content;
}

```

بعدين هنعمل كلاس ال publisher عباره عن event و method عادي جدا

```

public class NewsPublisher
{
    public event EventHandler<NewsArticle> NewNewsPublished;

    protected virtual void OnNewsPublished(NewsArticle News)
    {
        NewNewsPublished?.Invoke(this, News);
    }

    public void PublishNews(string Title, string Content)
    {
        NewsArticle article = new NewsArticle(Title, Content);
        OnNewsPublished(article);
    }

}

```

بعدين هنعرف subscriber هنخزن فيه اسم لـ subscriber وهنزود method لالغاء الاشتراك مع الناشر

```

public class NewsSubscriber
{
    public string Name { get; }
    public NewsSubscriber(string Name)
    {
        this.Name = Name;
    }

    public void Subscribe(NewsPublisher Publisher)
    {
        Publisher.NewNewsPublished += HandleNewNews;
    }

    public void Unsubscribe(NewsPublisher Publisher)
    {
        Publisher.NewNewsPublished -= HandleNewNews;
    }

    public void HandleNewNews(object sender, NewsArticle article)
    {
        Console.WriteLine($"{Name} Received a new news article:");
        Console.WriteLine($"Title: {article.Title}");
        Console.WriteLine($"Content: {article.Content}");
        Console.WriteLine();
    }
}

```

وفي ال main اقدر اعمل اكتر من ناشر واكتر من مشترك وكل مشترك يقدر يتحكم في الاشعارات اللي بتوصله من كل ناشر

انت بقى ممكن تعمل كلاس تاني تسميه archive ه يكون هوا كلاس ال subscriber بس الفرق انه بدل مايعرض رساله لا ده يأخذها يخزنها في الداتابيز باسم الناشر والوقت والتاريخ

```
using System;
using System.Security.Policy;

namespace TemperatureChangeEventExample
{

    public class NewsArticle
    {
        public string Title { get; }
        public string Content { get; }

        public NewsArticle(string Title, string Content)
        {
            this.Title = Title;
            this.Content = Content;
        }
    }

    public class NewsPublisher
    {
        public event EventHandler<NewsArticle> NewNewsPublished;

        protected virtual void OnNewsPublished(NewsArticle news)
        {
            NewNewsPublished?.Invoke(this, news);
        }

        public void PublishNews(string Title, string Content)
        {
            NewsArticle article = new NewsArticle(Title, Content);
            OnNewsPublished(article);
        }

    }

    public class NewsSubscriber
    {
        public string Name { get; }
        public NewsSubscriber(string name)
        {
            this.Name = name;
        }

        public void Subscribe(NewsPublisher publisher)
        {
            publisher.NewNewsPublished += HandleNewNews;
        }
    }
}
```

```

public void Unsubscribe(NewsPublisher Publisher)
{
    Publisher.NewNewsPublished -= HandleNewNews;
}

public void HandleNewNews(object sender, NewsArticle article)
{
    Console.WriteLine($"Name: {Name} Received a new news article:");
    Console.WriteLine($"Title: {article.Title}");
    Console.WriteLine($"Content: {article.Content}");
    Console.WriteLine();
}
}

public class Program
{
    static void Main(string[] args)
    {
        NewsPublisher publisher = new NewsPublisher();
        NewsSubscriber subscriber1 = new NewsSubscriber("Subscriber 1");

        subscriber1.Subscribe(publisher);

        NewsSubscriber subscriber2 = new NewsSubscriber("Subscriber 2");
        subscriber2.Subscribe(publisher);

        publisher.PublishNews("Breaking News", "A significant event just happened!");
        publisher.PublishNews("Tech Update", "New Gadgets are hitting the market.");

        subscriber1.Unsubscribe(publisher);
        publisher.PublishNews("Weather Forecast", "Expected sunny weather for the weekend.");

        subscriber2.Unsubscribe(publisher);
        publisher.PublishNews("Final Edition", "Last news update for today.");

        Console.ReadLine();
    }
}
}

```

New Order Event Example

دلوقي احنا عايزين نعمل كلاس لل **order** ولما العميل يطلب **order** معين بيعت بالايميل وال **sms** ويبلغ شركة الشحن

الكود مافيهوش حاجه جديه هوا الفكره انك بدل ماتعمل **subscriber** واحد لا بتعمل اكتر من واحد وبنفس الطريقه

هنا بيقولوك انك كان ممكن تعمل ال **object** بتابع ال **order** داخل ال **subscriber** نفسه زي ما شرحتلك فوق بس ممكن تعمل تعديل معين او انك مثلا عندك اكتر من **object** لنفس الاوردر تلاقى نفسك عشان تعدل عالاوردر انت هتدخل جوه ال **subscriber** نفسه وتعديل

فالطريقه دي بيقولك انها بتخلي الكود loosely coupled يعني مش معتمد على بعضه وكل جزء بيكون قائم بذاته

```
using System;
using System.Security.Policy;

namespace TemperatureChangeEventExample
{

    public class OrderEventArgs : EventArgs {
        public int OrderID { get; }
        public int OrderTotalPrice { get; }
        public string ClientEmail { get; }

        public OrderEventArgs(int orderId, int OrderTotalPrice, string clientEmail)
        {
            this.OrderID = orderId;
            this.OrderTotalPrice = OrderTotalPrice;
            this.ClientEmail = clientEmail;
        }
    }

    public class Order {
        public event EventHandler<OrderEventArgs> OnOrderCreated;
        public void Create(int orderId, int OrderTotalPrice, string ClientEmail) {
            Console.WriteLine("New order created; now will notify everyone by raising the event.\n");
            if(OnOrderCreated!=null){
                OnOrderCreated(this,new OrderEventArgs(orderId,OrderTotalPrice,ClientEmail));
            }
        }
    }

    public class EmailService {

        public void Subscribe(Order order) {
            order.OnOrderCreated += HandleNewOrder;
        }

        public void UnSubscribe(Order order)
        {
            order.OnOrderCreated -= HandleNewOrder;
        }

        public void HandleNewOrder(object sender,OrderEventArgs e) {
            Console.WriteLine($"-----Email Service-----");
            Console.WriteLine($"Email Service Object Recieved a new order event");
            Console.WriteLine($"Order ID: {e.OrderID}");
            Console.WriteLine($"Order Price: {e.OrderTotalPrice}");
            Console.WriteLine($"Email: {e.ClientEmail}");
            Console.WriteLine($"{`\nSend an Email`}");
            Console.WriteLine($"-----");
            // write the code to send email
            Console.WriteLine();
        }
    }

    public class SMSService {
        public void Subscribe(Order order)
```

```

{
    order.OnOrderCreated += HandleNewOrder;
}

public void UnSubscribe(Order order)
{
    order.OnOrderCreated -= HandleNewOrder;
}

public void HandleNewOrder(object sender, OrderEventArgs e)
{
    Console.WriteLine($"-----SMS Service-----");
    Console.WriteLine($"SMS Service Object Recieved a new order event");
    Console.WriteLine($"Order ID: {e.OrderID}");
    Console.WriteLine($"Order Price: {e.OrderTotalPrice}");
    Console.WriteLine($"Email: {e.ClientEmail}");
    Console.WriteLine($"{Environment.NewLine}Send an SMS");
    Console.WriteLine($"-----");
    // write the code to send SMS
    Console.WriteLine();
}

public class ShippingService {
    public void Subscribe(Order order)
    {
        order.OnOrderCreated += HandleNewOrder;
    }

    public void UnSubscribe(Order order)
    {
        order.OnOrderCreated -= HandleNewOrder;
    }

    public void HandleNewOrder(object sender, OrderEventArgs e)
    {
        Console.WriteLine($"-----Shipping Service-----");
        Console.WriteLine($"Shipping Service Object Recieved a new order event");
        Console.WriteLine($"Order ID: {e.OrderID}");
        Console.WriteLine($"Order Price: {e.OrderTotalPrice}");
        Console.WriteLine($"Email: {e.ClientEmail}");
        Console.WriteLine($"{Environment.NewLine}Send an Shipping");
        Console.WriteLine($"-----");
        // write the code to send Shipping
        Console.WriteLine();
    }
}

public class Program
{
    static void Main(string[] args)
    {
        Order order = new Order();
        EmailService emailService = new EmailService();
        SMSService smsService = new SMSService();
        ShippingService shippingService = new ShippingService();
    }
}

```

```

emailService.Subscribe(order);
smsService.Subscribe(order);
shippingService.Subscribe(order);

order.Create(10,540,"Ahmed@gmail.com");
order.Create(10,540,"Ali@gmail.com");

Console.ReadLine();
}
}
}

```

الشغله دي اسمها publisher subscriber design pattern

What we learned?

Publisher Subscriber Design Pattern ☺

Delegation Concept

الفكره فيه انك بتفصل المهام عن بعض وبيزيل التعقيد في الكود

Logger Example

ال `log` معناها انك عايز تسجل معلومه معينه تسجلها عالشاشة او علي ملف او في أي مكان
 مثل عليها هو ال `error` اللي بيطلع ممكن تظهره باي طريقه او تخزنها
 فاكر لما كنا بنعمل `enum` وكنا بعد كده بنعمل عليها `switch` هنا بقى انت مش هتعمل `switch` كل
 اللي هتعمله انك هتبعت ال `function` بتاع ال `address` اللي عايز تنفذها

تعالي كده واحده واحده ومن البداييه خالص
 احنا اتكلمنا عن ال `event` وعرفنا انه أصلا مبني عال `delegate`
 ايه هو ال `delegate` ?

هو `pointer` بس بدل مايشاور علي `object` او متغير لا هيشاور علي `method`

بیشتعل ازای؟

بیشتعل کویری انک دلوقتی موجود فی کلاس اسمه محمد و عایز تنفذ کود معین الكود ده بیعتمد علی داتا معینه و فی وقت معین

الداتا دی موجوده فی کلاس اسمه احمد والوقت المعین ده بیکون فی مرحله معینه من تشغیل الکلاس اللي اسمه احمد بررضه

فبنجي لاحمد ونعرف فيه delegate (ال pointer اللي هیشاور عال method)

ولما تيجي اللحظه المناسبه بنقول لاحمد شغلي ال method اللي ال delegate بتاعك بیشاور عليه أيا كان بقی هوا مین او موجود فین اهم حاجه ان ال parameters وال return type بتوع ال delegate يكونوا نفسهم بتوع ال delegate

طيب ازای بنعرف ال delegate

کانک بتعرف method بالظبط بس بتزود کلمة delegate

طيب ازای بشغل ال methods اللي بیشاور عليها؟

لو هنشاور علی method واحده بس

بناخد object من ال delegate وبنخزن فيه ال method

تعالی ناخد مثل

هنعمل logger

هنعمل کلاس عادي کل وظیفته انه بیعرف delegate و بیشغله

ادی کلاس

```
public class Logger {
```

وبعدین هنعرف delegate و نسمیه logAction و هنخليه ياخذ رساله

الرساله دی احنا ملناش دعوه فيها ايه ولا عایزین نعمل بیها ايه

دي بس وظیفتها انها بتحدلنا ال parameters بتاعت ال method اللي هنشاور عليها وفيما بعد لما تشغلي ال method هتطلب مننا داتا عشان تمررها لل logAction

```
public delegate void LogAction(string Message);
```

وبعدین هناخد object من ال delegate اللي عملناه ده

```
private LogAction _logAction;
```

طيب هنجي في ال constructor بناء الكلاس ونطلب فيه object من نفس نوع ال object اللي عملناه ونخزنه في ال object اللي عندنا

```
public Logger(LogAction action) {  
    this._logAction = action;  
}
```

بعد كده هنشغل ال logger هيطلب منا الداتا هنقوله ياخدها من الشخص اللي هيستخدم ال logger بعدين

```
public void Log(string message) {  
  
    this._logAction(message); }
```

طيب تعالى بقى نعمل method ونشغلها
هنعمل اتنين واحده بتطبع عالشاشة والثانويه بتكتب في ملف text عادي جدا بس بتأخذ نفس
ال parameters

```
public static void LogToScreen(string Message) {  
    Console.WriteLine(Message);  
}  
  
public static void LogToFile(string Message) {  
    string FileName = "log.txt";  
    using (StreamWriter writer = new StreamWriter(FileName, true)) {  
        writer.WriteLine(Message);  
    }  
}
```

نيجي في ال main

هناخد فيه object من ال logger فال constructor يطلب منا action هنيدله اسم ال

```
static void Main(string[] args)  
{  
    Logger ScreenLogger = new Logger(LogToScreen);  
    Logger FileLogger = new Logger(LogToFile);
```

هل كده اشتغل؟

لا انت كده يدوب خليته يشاور عال method

طب عايزيين نشغلها

خذ شغلها براحتك

```
ScreenLogger.Log("The message will be displayed on the screen.");
FileLogger.Log("The message will be logged to a File.");
```

طبعاً انت ممكن تعمل **method** جوه ال **logger** عان تقدر تغير ال **method** اللي عاوز تشاور عليها زي كده

```
public void ChangeMethod(LogAction action)
{
    this._logAction = action;
}
```

```
FileLogger.ChangeMethod(LogToScreen);
FileLogger.Log("The message will be displayed on the screen.");
```

بس كده وده الكود كله

```
using System;
using System.IO;

namespace TemperatureChangeEventExample
{

    public class Logger {

        public delegate void LogAction(string Message);

        private LogAction _logAction;

        public Logger(LogAction action) {
            this._logAction = action;
        }

        public void ChangeMethod(LogAction action)
        {
            this._logAction = action;
        }

        public void Log(string message){ this._logAction(message); }
    }
}
```

```
public class Program
{

    public static void LogToScreen(string Message) {
        Console.WriteLine(Message);
    }

    public static void LogToFile(string Message) {
        string FileName = "log.txt";
        using (StreamWriter writer = new StreamWriter(FileName, true)) {
            writer.WriteLine(Message);
        }
    }
}
```

```

        }

static void Main(string[] args)
{
    Logger ScreenLogger = new Logger(LogToScreen);
    Logger FileLogger = new Logger(LogToFile);

    ScreenLogger.Log("The message will be displayed on the screen.");
    FileLogger.Log("The message will be logged to a File.");

    FileLogger.ChangeMethod(LogToScreen);
    FileLogger.Log("The message will be displayed on the screen.");

    Console.ReadLine();
}
}
}

```

What is Multicast Delegate?

هناخد الكود اللي فات عشان هنعمل عليه تعديل بسيط تقدر من خلله تخلي ال pointer او ال method الواحد يشاور علي اكتر من delegate

ده التعديل في ال logger هشيل ال constructor و هعدل عال method اللي بتغير ال logger اللي بنشاور عليها و هنخليها تضيف بدل ماتغير

```

public class Logger {

    public delegate void LogAction(string Message);

    private LogAction _logAction;

    public void AddMethod(LogAction action)
    {
        this._logAction += action;
    }

    public void Log(string message) { this._logAction(message); }
}

```

وهاجي في ال main هاخد object من الكلاس logger و هضيف علي نفس ال اتنين methods

بحيث انك تشغلي كل ال methods باستدعاء ال logger.log مرة واحدة

```

static void Main(string[] args)
{
    Logger Logger = new Logger();

    Logger.AddMethod(LogToScreen);
    Logger.AddMethod(LogToFile);

    Logger.Log("The message will be displayed on the screen.{115}");
}

```

```
        Console.ReadLine();
    }
```

You already know it :-), many methods can subscribe to a delegate, and when delegate is called it will call all subscribers, this is called **multicast delegate**.

Multicast Delegate:

In C#, a multicast delegate is a special type of delegate that can **reference multiple methods and invoke them in a single call**.

Delegates are used to encapsulate and reference methods, and a multicast delegate extends this concept by allowing you to combine multiple method references into a single delegate object.

You can create a multicast delegate by using the `+=` and `-=` operators to add or remove method references to the delegate. When you invoke a multicast delegate, it will call all the referenced methods in the order they were added.

Here's a simple example:

```
using System;

public delegate void MyDelegate(string message);

class Program
{
    static void Main()
    {
        MyDelegate myDelegate = Method1;
        myDelegate += Method2;

        myDelegate("Hello, world!");

        myDelegate -= Method1;
        myDelegate("Another message.");
    }
}
```

```

static void Method1(string message)
{
    Console.WriteLine("Method1: " + message);
}

static void Method2(string message)
{
    Console.WriteLine("Method2: " + message);
}

```

In this example, the `MyDelegate` delegate is a multicast delegate that references both `Method1` and `Method2`. When you invoke the delegate with `myDelegate("Hello, world!");`, both `Method1` and `Method2` are called, and their output is displayed.

You can also use multicast delegates for scenarios like event handling, where multiple event handlers need to be called when an event is raised. Multicast delegates are commonly used in C# for implementing the observer pattern and event-driven programming.

Func Delegate

ال `func` ممكن ماتستخدمهاش عادي بس زيادة معرفة وهيا عباره عن انها بتخليلك تعرف `delegate`
بيرجع داتا مش بيرجع `void` زي ماكنا بنعمل

ده كود بسيط لـ `delegate` الجديد فيه اني بدل كلمة `void` حطيت `int` عشان اعرفه اني اوي
هيشاور عليها هترجع `int` `function`

```

using System;
using System.IO;

namespace TemperatureChangeEventExample
{
    public class Program
    {
        delegate int SquareDelegate(int x);

        static int SquareMethod(int x) {
            return x * x;
        }
    }
}
```

```

    }

    static void Main(string[] args)
    {
        SquareDelegate square = new SquareDelegate(SquareMethod);

        int result = square(5);

        Console.WriteLine("The Square of 5 is: " + result);
        Console.ReadLine();
    }
}

```

هنا بقى قالك بدل ماتكتب السطر ده لا عملك اختصار وهو ببساطه انك بتكتب func و بتكتب بعديه اكتر من data type بتقدر تضيف لحد ١٦ واحد فبهم لازم يعبر عن ال اللي ال method هترجعها

شوف الفرق في الكود اللي جاي

```

using System;
using System.IO;

namespace TemperatureChangeEventExample
{
    public class Program
    {
        // delegate int SquareDelegate(int x);

        static Func<int,int> square = SquareMethod;

        static int SquareMethod(int x) {
            return x * x;
        }

        static void Main(string[] args)
        {
            // SquareDelegate square = new SquareDelegate(SquareMethod);

            int result = square(5);

            Console.WriteLine("The Square of 5 is: " + result);
            Console.ReadLine();
        }
    }
}

```

What is Func Delegates?

In simple words: it's a shortcut for normal delegate..

The `Func` delegate is a predefined delegate type in C# that represents a method that takes zero or more input parameters and returns a value. It's part of the `System` namespace and is often used to define and pass around functions or methods that return a value.

The `Func` delegate is defined in various forms, depending on the number of input parameters and the return value:

- `Func<TResult>`: Represents a method that takes no parameters and returns a result of type `TResult`.
- `Func<T, TResult>`: Represents a method that takes one parameter of type `T` and returns a result of type `TResult`.
- `Func<T1, T2, TResult>`: Represents a method that takes two parameters of types `T1` and `T2` and returns a result of type `TResult`.
- So on, `Func<T1, T2, ..., Tn, TResult>` for methods with n parameters.

Here are a few examples of how to use the `Func` delegate:

1. `Func<int, int>`: Represents a method that takes an `int` as input and returns an `int`.
2. `Func<string, int, bool>`: Represents a method that takes a `string` and an `int` as input and returns a `bool`.
3. `Func<double, double, double, double>`: Represents a method that takes three `double` parameters and returns a `double`.

Action Delegate

زي ماقولنا قبل كده ال `action` هوا `delegate` جاهز تحتاج منك بس تدخل ال `data types` بتاعت
ال `parameters` وممكن تضيف لحد ١٦ `parameter`

ال `void` هوا `delegate` بيرجع `void` بيقي لازم ال `methods` اللي بيشاور عليها ترجع

```
using System;
using System.IO;

namespace TemperatureChangeEventExample
{
```

```

public class Program
{
    static void Main(string[] args)
    {
        Action ParameterLessAction = ParamameterLessMethod;
        Action<int> ActionWith_int_Parameter = MethodWith_int_Parameter;
        Action<string ,int> ActionWithMultipleParameters = MethodWithMultipleParameters;

        ParameterLessAction();
        ActionWith_int_Parameter(42);
        ActionWithMultipleParameters("Hi",100);
        Console.ReadLine();
    }

    static void ParamameterLessMethod()
    {
        Console.WriteLine("ParameterLessAction");
    }

    static void MethodWith_int_Parameter(int num)
    {
        Console.WriteLine($"ParamameterLessMethod {num}");
    }

    static void MethodWithMultipleParameters(string z, int x)
    {
        Console.WriteLine($"{z} {x}");
    }
}

```

Action Delegate:

In simple words: it's a shortcut for normal delegate like the Func Delegate but without returning value.

In C#, a delegate is a type that represents a reference to a method, allowing you to treat methods as first-class objects. Delegates are often used for implementing callback functions, event handling, and other scenarios where you want to pass a method as a parameter to another method.

An **Action** delegate is a predefined delegate type in C# that represents a method that takes zero or more parameters and does not return a value (i.e., it has a **void** return type). It is part of the **System** namespace. Actions are typically used when you need to perform some action or operation without needing to return a value.

Predicate Delegate

ال predicate delegate هو delegate واحد بس ويرجع واحد parameter عادي بيأخذ

```

using System;
using System.IO;

```

```

namespace TemperatureChangeEventExample
{
    public class Program
    {
        static Predicate<int> isEvenPredicate = isEven;

        static bool isEven(int x)
        {
            return x % 2 == 0;
        }

        static void Main(string[] args)
        {
            bool result = isEvenPredicate(5);

            Console.WriteLine(result);

            Console.ReadLine();
        }
    }
}

```

In simple words: it's a shortcut for a delegate function that takes one parameter and return boolean.

In C#, a predicate is a delegate that represents a method that takes one or more parameters and returns a Boolean value. Specifically, the `Predicate<T>` delegate is a built-in generic delegate in the `System` namespace, and it's commonly used for defining conditions or filters.

The `Predicate<T>` delegate is defined as follows:

```
public delegate bool Predicate<in T>(T obj);
```

Here, `T` is the type of the parameter that the method takes, and the delegate returns a Boolean value.

Lambda Expression

ال `lambda` كنا بنعملها اسمها `name functions` فيه نوع تانيه اسمه `methods`
 وده يعتبر اختصار مش اكتر لكن مش هيأثر عالسرعه
 ومعموله عشان يخلوا ال `oop` تدعم ال `functional programming` وهيا عادة بتسخدم مع ال
`delegates`

بص ال `method` بتاعت ال `square` في الكود اللي جاي ده

```
using System;
using System.IO;
```

```

namespace TemperatureChangeEventExample
{
    public class Program
    {
        // delegate int SquareDelegate(int x);

        static Func<int, int> square = SquareMethod;

        static int SquareMethod(int x)
        {
            return x * x;
        }

        static void Main(string[] args)
        {
            // SquareDelegate square = new SquareDelegate(SquareMethod);

            int result = square(5);

            Console.WriteLine("The Square of 5 is: " + result);
            Console.ReadLine();
        }
    }
}

```

تقدر تستغني عن الجزء ده

```

static int SquareMethod(int x)
{
    return x * x;
}

```

بالجزء ده

```
static Func<int, int> square = x=>x*x;
```

الجزء ده $x \Rightarrow x^*$ اسمه **lambda expression** هيا العلامه دي
وال **lambda expression** بيسموها **anonymous function** يعني لو عايز تعمل حاجه
عالسريع وال **scope** بتاعها محدود استخدمها بدل ماتكتب في سطور كتير عشان تعمل حاجه بسيطه
ال **lambda** بتكتب ازاي؟

اول حاجه هتسأله هل ال **method** اللي عايز تعملها دي بتأخذ **parameters**
لو بتأخذ بتحطهم بين قوسين ولو مش بتأخذ بتحط القوسين فاضيين

(input-parameters)

بعدين بتحط علامه يساوي وبعدها اكتر من

=>

تالت حاجه هيا ال **implementation** بتعال **function** هل هو مجرد **expression** يعني مثلا عباره عن عملية جمع او طرح او مقاره او الحاجات دي؟

لو اه بتكتبها علي طول

فيصبح عندك ال **expression** بالشكل ده

(input-parameters) => expression

طيب لو كان ال **implementation** عباره عن سطر كود معين خفيف لطيف؟
هتحطه بين قوسين فيصبح ال **expression** بالمنظار ده

(input-parameters) => { Your Code }

شوف المثال اللي جاي ده

```
using System;
using System.IO;

namespace TemperatureChangeEventExample
{
    public class Program
    {
        // delegate int SquareDelegate(int x);

        static Func<int, int> square = x=>x*x;

        static void Main(string[] args)
        {
            // SquareDelegate square = new SquareDelegate(SquareMethod);

            int result = square(5);

            Console.WriteLine("The Square of 5 is: " + result);
            Console.ReadLine();
        }
    }
}
```

ده الكود اللي كنا عاملينه بتعال **action delegate**

```
using System;
using System.IO;

namespace TemperatureChangeEventExample
{
    public class Program
    {

        static void Main(string[] args)
        {
            Action ParameterLessAction = ParameterLessMethod;
            Action<int> ActionWith_int_Parameter = MethodWith_int_Parameter;
```

```

Action<string, int> ActionWithMultipleParameters = MethodWithMultipleParameters;

ParameterLessAction();
ActionWith_int_Parameter(42);
ActionWithMultipleParameters("Hi", 100);
Console.ReadLine();
}

static void ParameterLessMethod()
{
    Console.WriteLine("ParameterLessAction");
}

static void MethodWith_int_Parameter(int num)
{
    Console.WriteLine($"ParameterLessMethod {num}");
}

static void MethodWithMultipleParameters(string z, int x)
{
    Console.WriteLine($"{z} {x}");
}
}

```

لو عملناه بال lambda هيكون كده

```

using System;
using System.IO;

namespace TemperatureChangeEventExample
{
    public class Program
    {

        static void Main(string[] args)
        {
            Action ParameterLessAction = ()=> {
                Console.WriteLine("ParameterLessAction");

                Action<int> ActionWith_int_Parameter = (num) => {
                    Console.WriteLine($"ParameterLessMethod {num}"); };

                Action<string, int> ActionWithMultipleParameters = (z,x) => {
                    Console.WriteLine($"{z} {x}");
                };

                ParameterLessAction();
                ActionWith_int_Parameter(42);
                ActionWithMultipleParameters("Hi", 100);
                Console.ReadLine();
            }
        }
    }
}

```

بص الكود ده

```

using System;
using System.IO;

namespace TemperatureChangeEventExample
{
    public class Program
    {
        delegate int Operation(int x,int y);

        static void ExcuteOperation(int x, int y,Operation operation) {
            int Result = operation(x, y);
            Console.WriteLine("Result: "+Result);
        }

        static int Add(int x, int y) { return x + y; }

        static int Sub(int x, int y) { return x - y; }

        static void Main(string[] args)
        {
            Operation AddOp = Add;
            Operation SubOp = Sub;

            ExcuteOperation(5,10,AddOp);
            ExcuteOperation(5,10,SubOp);

            Console.ReadLine();
        }
    }
}

```

ممكن نعمله کده

```

using System;
using System.IO;

namespace TemperatureChangeEventExample
{
    public class Program
    {
        delegate int Operation(int x,int y);

        static void ExcuteOperation(int x, int y,Operation operation) {
            int Result = operation(x, y);
            Console.WriteLine("Result: "+Result);
        }

        static void Main(string[] args)
        {
            Operation AddOp = (x,y)=>x+y;
            Operation SubOp = (x, y) =>x-y;

            ExcuteOperation(5,10,AddOp);
            ExcuteOperation(5,10,SubOp);

            Console.ReadLine();
        }
    }
}

```

} و ممکن نخی ال func تاخد parameter ک بحیث انک تکب کل execute operation لوحدها

```
using System;
using System.IO;

namespace TemperatureChangeEventExample
{
    public class Program
    {
        static void ExecuteOperation(int x, int y, Func<int,int,int> Operation)
        {
            int Result = Operation(x, y);
            Console.WriteLine("Result: " + Result);
        }

        static void Main(string[] args)
        {
            Func<int,int,int> Add = (x,y)=>x+y;
            Func<int, int, int> Sub = (x, y) =>x-y;

            ExecuteOperation(5,10,Add);
            ExecuteOperation(5,10,Sub);

            Console.ReadLine();
        }
    }
}
```

Lambda expressions are often preferred over normal (named) functions in specific scenarios because they offer some advantages, such as **conciseness, inline definition, and the ability to create anonymous functions**. Here are a few reasons why you might choose to use lambda expressions over normal functions:

- Conciseness:** Lambda expressions are typically more concise than defining a separate function, especially for small, simple operations. This leads to more readable and less verbose code.
- Anonymous Functions:** Lambda expressions allow you to create anonymous functions on the fly without needing to declare a separate named function. This is particularly useful when you need a function for a one-time or specific purpose.
- Inline Definitions:** Lambda expressions can be defined inline within a statement or method call, making it easy to pass functions as arguments to methods or use them in LINQ queries without defining separate functions.
- Readability:** In some cases, a lambda expression's compact and in-place definition can improve code readability, as it keeps the code closer to the point where the function is used.

5. **Functional Programming:** Lambda expressions align with functional programming principles, which can lead to more expressive and declarative code in certain contexts, such as LINQ queries and event handling.
6. **Flexibility:** Lambda expressions can capture variables from their surrounding scope (known as closures), allowing you to create more flexible and context-aware functions.

While lambda expressions offer these advantages, it's important to note that there are cases where defining a separate named function is more appropriate. You would typically use named functions when you need to reuse the same logic in multiple places, promote code organization, or when the function is more complex and requires detailed documentation.

In summary, lambda expressions are a valuable tool in C# for creating concise, **inline**, and **anonymous functions**, which can improve code readability and maintainability in specific scenarios, such as LINQ queries, event handlers, and small, one-off operations.

Named Functions vs. Lambda Expressions when to use and which is faster?

Named Functions vs. Lambda Expressions

Let's talk about when to use named functions and when to use lambda expressions, and also touch on the performance aspect.

Named Functions:

1. Readability and Reusability:

- Use named functions when the logic is complex or when the operation needs to be reused in multiple places. Named functions can enhance code readability and maintainability.

```
int Square(int num)
{
    return num * num;
}

// Usage
int result = Square(5);
Console.WriteLine("Square of number: " + result);
```

2 .Clear Intent:

- If the function's purpose is clear from its name, using a named function is often a good choice. It makes your code self-documenting.

Lambda Expressions:

- Conciseness:
 - Use lambda expressions for short, simple operations, especially when the logic is straightforward and doesn't need a separate named function. They shine in scenarios where brevity is essential.

```
// Lambda expression for squaring a number
Func<int, int> square = (int num) => num * num;
```

```
// Usage
int result = square(5);
Console.WriteLine("Square of number: " + result);
```

- Inline Usage: If the function is used inline, for example, in LINQ queries or event handling, lambda expressions can be more convenient.

```
// Using lambda in LINQ
var evenNumbers = numbers.Where(n => n % 2 == 0);

// Using lambda in event handling
button.Click += (sender, e) => Console.WriteLine("Button clicked!");
```

Performance Considerations:

In terms of performance, the difference between named functions and lambda expressions is usually negligible. Both can be optimized by the compiler. The choice between them should primarily be based on readability, maintainability, and code organization.

In terms of performance, the difference between using a lambda expression and a declared function (method) is usually negligible. Both approaches can be optimized by the compiler, and the generated IL (Intermediate Language) code may end up being quite similar.

The choice between a lambda expression and a declared function often depends on factors like readability, code organization, and whether the function is used in a single location or needs to be reused in multiple places.

In simple cases, using a lambda expression with the `Func<int, int>` type is concise and suitable for a simple operation like squaring a number. If your logic becomes more complex or you need to reuse the operation in multiple places, declaring a separate method might make your code more modular and maintainable.

Ultimately, for performance considerations, the difference between these two approaches is likely to be minimal. Choose the one that fits best with your coding style and the overall structure of your program.

Conclusion:

- Named functions: Ideal for complex logic, reuse, and when code readability is crucial.
- Lambda expressions: Great for concise, one-off operations, and when brevity is a priority.

Remember, the performance difference is often minimal, and your choice should be driven by the specific needs of your code and the principles of clean and maintainable coding.

When should named functions be used?

When the logic is complex

When the operation needs to be reused in multiple places

When the function's purpose is clear from its name

When brevity is essential

When should lambda expressions be used?

For short, simple operations

When the logic is complex

When code readability is crucial

What is a benefit of using named functions?

Enhances code readability

Improves performance

Allows inline usage

Enables brevity

What is a benefit of using lambda expressions?

Conciseness

Readability and reusability

Clear intent

Performance optimization

Allows inline usage

Enables brevity

C# using - To Import Library

كلمة using ليها اربع استخدامات اول حاجه وهيا اننا نعمل import للمكتبات

C# using - To Import Library

In C#, we use the using keyword to import external resources (namespaces, classes, etc) inside a program. For example,

```
// using System namespace  
using System;  
  
namespace Program {  
  
    class Program1 {  
        static void Main(string[] args) {  
            Console.WriteLine("Hello World!");  
        }  
    }  
}
```

Output

```
Hello World!
```

In the above example, notice the line

```
using System;
```

Here, we are importing the **System** namespace inside our program. This helps us to directly use the classes present in the **System** namespace.

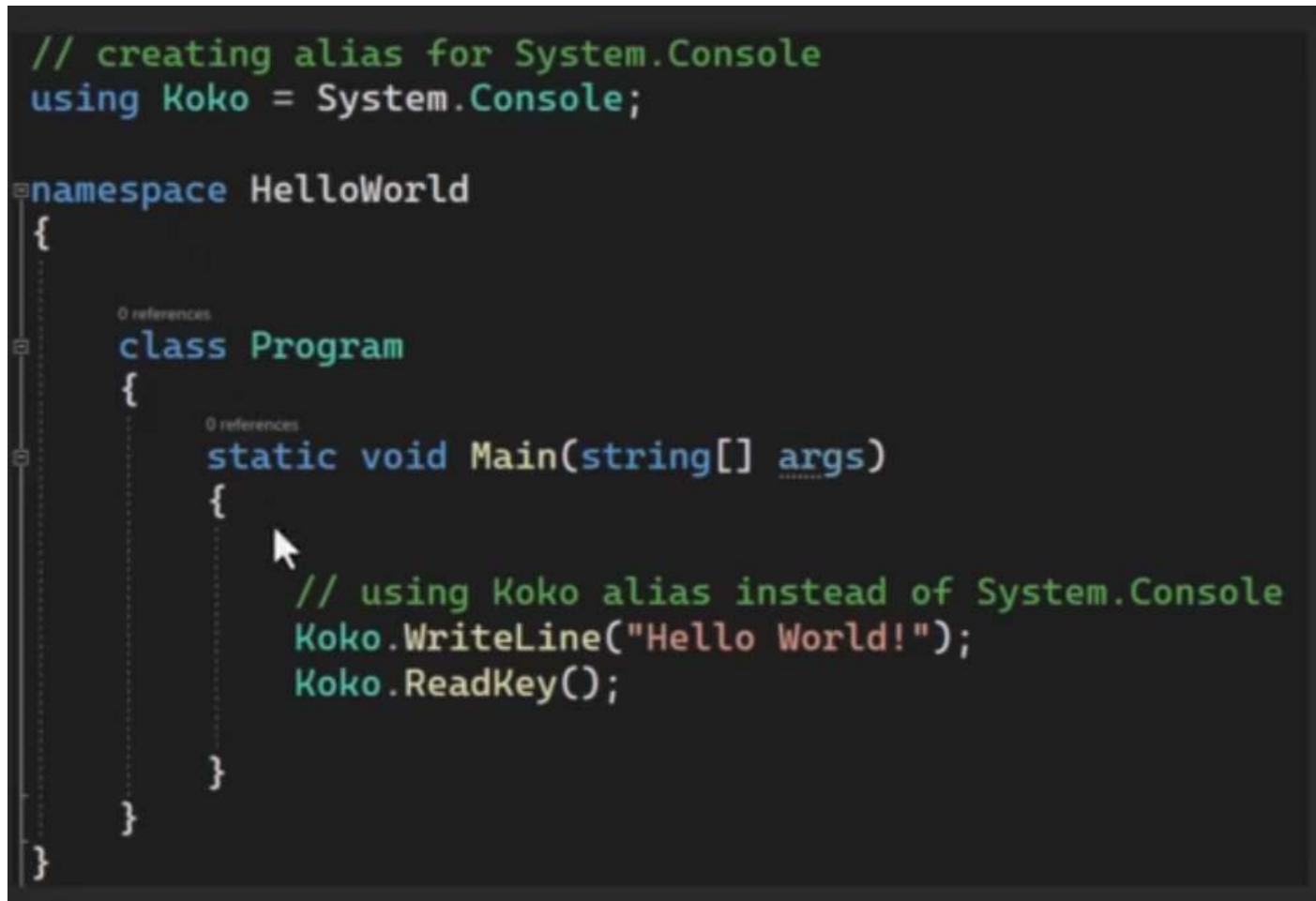
Also, because of this, we don't have to write the fully qualified name of the print statement.

```
// full print statement  
System.Console.WriteLine("Hello World!");  
  
// print statement with using System;  
Console.WriteLine("Hello World!");
```

C# using to create an alias

تاني استخدام لكلمة `using` هو لما يكون عندك كود طويل ممكن تعمله اختصار بكلمة `using` يعني بدل ما كنت بتكتب مثلا `a.b.c.getname` لا انت ممكن تكتب في الكلاس من فوق `a.b` بحيث انك تيجي في الكود تكتب `c.getname` على طول

زي في ال `C++` لما كنا بنستخدم ال `using` عشان مانفضلش كل شوية نكتب `std` في المثال اللي جاي بدل مانفضل كل شوية نكتب `System.Console.WriteLine` لا احنا حطيناه في كلمة `Koko` وبقينا نستخدم كلمة `Koko` لدلالة علي `System.console`



The screenshot shows a C# code editor with the following code:

```
// creating alias for System.Console
using Koko = System.Console;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            // using Koko alias instead of System.Console
            Koko.WriteLine("Hello World!");
            Koko.ReadKey();
        }
    }
}
```

A cursor is visible over the `Koko` alias in the `Main` method.

C# using to create an alias

We can also create aliases with the help of `using` in C#. For example,

```
// creating alias for System.Console
using Koko = System.Console;

namespace HelloWorld {

    class Program {
        static void Main(string[] args) {
```

```
// using Koko alias instead of System.Console  
Koko.WriteLine("Hello World!");  
Koko.ReadKey();  
}  
}  
}
```

Output

```
Hello World!
```

In the above program, we have created an alias for `System.Console`.

```
using Koko = System.Console;
```

This allows us to use the alias Koko instead of `System.Console`.

```
Koko.WriteLine("Hello World!");
```

Here, Koko will work just like `System.Console`.

C# using static directive

لما كنا بنكتب كلمة `using` وبعدها اسم `namespace` معين بنقدر بعد كده نستخدم أي عنصر جوه ال `namespace` بدون مانذكر اسمه

طيب انا هستخدمه عشان اشاور علي عناصر معينه جوه كلاس معين جوه ال `namespace` والعناصر دي من النوع `static` بس

خلاص اكتب كلمة `static` بعد كلمة `using`

```
using System;
using static directive
using static System.Math;

namespace Program
{
    class Program1
    {
        public static void Main(string[] args)
        {

            double n = Sqrt(9);
            Console.WriteLine("Square root of 9 is " + n);
            Console.ReadKey();
        }
    }
}
```

C# using static directive

In C#, we can also import classes in our program. Once we import these classes, we can use the static members (fields, methods) of the class.

We use the `using static` directive to import classes in our program.

Example: C# using static with System.Math

```
using System;

// using static directive
using static System.Math;

namespace Program {

    class Program1 {
        public static void Main(string[] args) {

            double n = Sqrt(9);
            Console.WriteLine("Square root of 9 is " + n);
        }
    }
}
```

```
    }
}
}
```

Output

```
Square root of 9 is 3
```

In the above example, notice the line,

```
using static System.Math;
```

Here, this line helps us to directly access the methods of the **Math** class.

```
double n = Sqrt(9);
```

We have used the **Sqrt()** method directly without specifying the **Math** class.

If we don't use the **using static System.Math** in our program, we have to include the class name **Math** while using **Sqrt()**. For example,

```
using System;

namespace Program {

    class Program1 {
        public static void Main(string[] args) {

            // using the class name Math
            double n = Math.Sqrt(9);
            Console.WriteLine("Square root of 9 is " + n);
        }
    }
}
```

Output

```
Square root of 9 is 3
```

In the above example, notice the line,

```
double n = Math.Sqrt(9);
```

Here, we are using `Math.Sqrt()` to compute the square root of **9**. This is because we haven't imported the `System.Math` in this program.

C# using for Resource Management

فيه حاجه اسمها **unmanaged code** و **managed code**
ال **managed code** بيقي ليه مميزات انه بيكون فيه **garbage collection** وغيرها من
المميزات اللي مش موجوده في ال **C++**
طيب احنا عارفين انه ال **C#** مبنيه على **C++** وبالتالي فيه اكواد معموله بال **C#** وبالتالي تكون
و فيه مكتبات بتعتمد على ال **C++** وبالتالي تكون **unmanaged code**
احجه زي لما كنا بنفتح اتصال ب **file** او قاعدة بيانات كنا لازم نخلی بالننا اننا نغلق ال **connection**
عشان مايعلش مشاكل
هنا بقى بيقولك انه ال **using** بتخلی ال **connection** يتغفل من غير ما ننت تقول يعني بتخلی الكود
بتاعك **managed**

For Resource Management

هنا لما بنجي نفتح ملف بتحط ال **connection** اللي عايز تفتحه بين قوسين بس يكون الكلاس اللي
هتستخدمه يكون بيستخدم اسمه **disposable interface** زي ال **stream reader**
وبتكتب الكود بتاعك من غير ما تحتاج انك تقفله لانه بمجرد ما ال **compiler** يخرج من الجزء ده هيقف
الاتصال مع نفسه

1- File I/O - Reading and writing to a text file:



```
using (var reader = new StreamReader("example.txt"))
{
    string line;
    while ((line = reader.ReadLine()) != null)
    {
        Console.WriteLine(line);
    }
}
// The StreamReader will be automatically closed and resources released.
```

وده مثال تاني عالداتا بيز

2-Database Connection - Connecting to a SQL Server database using SqlConnection :

```
using (var connection = new SqlConnection(connectionString))
{
    connection.Open();
    // Execute SQL commands
}
// The SqlConnection will be automatically closed and resources released.
```



وذه مثال عال network

3- Network Resources - Working with a network stream:

```
using (var client = new TcpClient("example.com", 80))
{
    using (var stream = client.GetStream())
    {
        // Read and write to the network stream
    }
}
// Both TcpClient and NetworkStream will be automatically closed and resources released.
```

لو انت عملت كلاس بيعمل implementation disposable interface | بتقدر تستخدم ال using

4- Working with IDisposable Objects - Using custom objects that implement IDisposable :

```
using (ResourceType resource = new ResourceType())
{
    // Code that uses the resource
    // The resource will be automatically disposed when the block is exited
}
```



بالنسبة لنا في الداتا بيز بنستخدم ال using مع ال connection وال command وال data reader من غير ماقوله افتح واقفل

```
using System;
using System.Data.SqlClient;

class Program
{
    static void Main()
    {
        string connectionString = "Server=your_server;Database=your_database;User
Id=your_username;Password=your_password;";

        try
        {
```

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();

    string query = "SELECT FirstName, LastName FROM Employee";

    using (SqlCommand command = new SqlCommand(query, connection))
    {
        using (SqlDataReader reader = command.ExecuteReader())
        {
            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    string firstName = reader["FirstName"].ToString();
                    string lastName = reader["LastName"].ToString();
                    Console.WriteLine($"Name: {firstName} {lastName}");
                }
            }
            else
            {
                Console.WriteLine("No rows found.");
            }
        }
    }
}

catch (SqlException ex)
{
    Console.WriteLine("Database connection error: " + ex.Message);
}
}
```

In C#, the `using` statement is also used for **Resource Management**: The `using` statement is commonly used for resource management, such as working with objects that implement the `IDisposable` interface. The `IDisposable` interface provides a method called `Dispose` that allows you to release unmanaged resources or perform cleanup operations. By using the `using` statement, you can ensure that the `Dispose` method is called automatically when the block of code is exited, even if an exception is thrown.

Here's an example of how the `using` statement is used for resource management:

```
using (var resource = new DisposableResource())
{
    // Use the resource
    // The Dispose method will be automatically called when this block is exited.
}
```

Examples of using the Using statement:

1- File I/O - Reading and writing to a text file:

```
using (var reader = new StreamReader("example.txt"))
{
    string line;
    while ((line = reader.ReadLine()) != null)
    {
        Console.WriteLine(line);
    }
}
// The StreamReader will be automatically closed and resources released.
```

2-Database Connection - Connecting to a SQL Server database using SqlConnection:

```
using (var connection = new SqlConnection(connectionString))
{
    connection.Open();
    // Execute SQL commands
}
// The SqlConnection will be automatically closed and resources released.
```

3- Network Resources - Working with a network stream:

```
using (var client = new TcpClient("example.com", 80))
{
    using (var stream = client.GetStream())
    {
        // Read and write to the network stream
    }
}
// Both TcpClient and NetworkStream will be automatically closed and resources released.
```

4- Working with IDisposable Objects - Using custom objects that implement IDisposable:

```
using (ResourceType resource = new ResourceType())
{
    // Code that uses the resource
    // The resource will be automatically disposed when the block is exited
}
```

If you have a custom class that needs to be used with the `using` statement, make sure it implements `IDisposable` and properly handles resource cleanup in its `Dispose` method, as demonstrated in the previous response.

In all these examples, the `using` statement ensures that the resources are properly released and disposed of when they go out of scope, making the code cleaner and more reliable.

Here are a few reasons why using `using` is recommended for database connections:

1. Resource Cleanup: Database connections often involve unmanaged resources, such as network connections or file handles. The `Dispose` method is where you release these resources. If you don't properly release these resources, it can lead to issues like connection leaks, where the database server runs out of available connections.
2. Timely Release of Resources: By using the `using` statement, you guarantee that the `Dispose` method is called when the code block is exited. This ensures that resources are released in a timely manner, even if an exception occurs within the block.
3. Easier Code Maintenance: Using `using` makes the code cleaner and more readable. It clearly indicates the scope in which the resource is used, making it easier for developers to understand when the resource is acquired and when it is released.

Nullable Data Types

في الكروس اللي فات كنا بنتجنب الأخطاء الناتجه عن ان ال object تكون قيمته null عن طريق ال if statement دلوقتي بقى بيقولك انك ممكن تعرف object او متغير معين انه nullable ممكن يكون null بدل ماتقدر تكتب if كل شوية طريقة تعريف المتغير انه nullable ليه طريقتين اول طريقة انك تكتب nullable وتحدد ال data type

```
Nullable<int> nullableInt1 = null;
```

وتاني طريقة واسمها short hand notation يدوب بتحط علامة استفهام

```
int? nullableInt2 = null;
```

وبتقدير تشيك ان كان فيها قيمة ولا لا

```
nullableInt1.HasValue
```

وبتقدير تعين قيمة بحيث انه المتغير بتاعك مايظهرش ب null ابدأ تجربا للاخطاء وده بيتم بعلامتين استفهام

```
nullableInt2 = nullableInt2 ?? 0;
```

```
int result = nullableInt2 ?? 0;
```

وتقدر تحوله ل string بالطريقة دي انك تحط علامة استفهام وبعدها tostring

```
string stringValue = nullableInt2?.ToString();
```

In C#, a nullable data type is a value type that can also be assigned a value of null. This is achieved by using the Nullable<T> structure or its shorthand notation T?, where T is the underlying value type. This is particularly useful when dealing with database values or scenarios where a variable may legitimately have no value.

Here's an example using Nullable<T> with an int:

```
using System;
```

```
class NullableExample
{
    static void Main()
    {
        // Declare a nullable int using Nullable<T>
        Nullable<int> nullableInt1 = null;

        // Shorthand notation using int?
        int? nullableInt2 = null;

        // Check if the nullable ints have values
        if (nullableInt1.HasValue)
        {
            Console.WriteLine("nullableInt1 has a value: " + nullableInt1.Value);
        }
        else
        {
            Console.WriteLine("nullableInt1 is null.");
        }

        if (nullableInt2.HasValue)
        {
            Console.WriteLine("nullableInt2 has a value: " + nullableInt2.Value);
        }
        else
        {
            Console.WriteLine("nullableInt2 is null.");
        }

        // Using the null-coalescing operator
        int result = nullableInt2 ?? 0;
        Console.WriteLine("Using null-coalescing operator: " + result);

        // Using the null-conditional operator
        string stringValue = nullableInt2?.ToString();
    }
}
```

```

        Console.WriteLine("String representation: " + (stringValue ?? "null"));
        Console.ReadKey();
    }

}

```

In this example:

- `Nullable<int>` and `int?` are both used to declare nullable integers.
- `HasValue` property is used to check if the nullable value has a value.
- The null-coalescing operator (`??`) is used to provide a default value if the nullable value is `null`.
- The null-conditional operator (`?.`) is used to access a property or method only if the nullable value is not `null`.

Using `int?` is more common and convenient for declaring nullable integers in C#. You can apply the concept of nullable types to other value types as well, such as `DateTime?`, `float?`, etc., depending on your requirements.

Nullable Datatypes Example 2

```

using System;

class NullableExample
{
    static void Procedure1(string name, Nullable<int> Age) {
        Console.WriteLine(name);

        if (Age.HasValue) {
            Console.WriteLine(Age.ToString());
        } else { Console.WriteLine("Age is null"); }
    }

    static void Procedure2(string name, int? Age) {
        Console.WriteLine(name);

        if (Age.HasValue)
        {
            Console.WriteLine(Age.ToString());
        }
        else { Console.WriteLine("Age is null"); }
    }
}

```

```

static void Main()
{
    Procedure1("Mohamed", null);
    Procedure1("Aly", 35);

    Procedure2("Mohamed", null);
    Procedure2("Ali", 35);

    Console.ReadKey();
}
}

```

Serialization / Deserialization

ال serialization هيا عباره عن انك بتاخذ object موجود في ال memory وتحوله لصيغه معينه من خلال الصيغه دي تقدر تبعته لاي برنامج تاني او ملف تاني يقدر يتعامل معاه ويرجعه object تاني وهيا طريقة من خلالها بتقدر تبعت object كله علي بعضه من برنامج لبرنامج تاني يعني من الآخر بدل ماكنا مثلاً واحداً بنتعامل مع الداتابيز كنا بنفك ال object ونطلع المتغيرات اللي فيه ونبعتها ولما نيجي نستلمها بنستلهمها متفرطه ونرجع نجمعها تاني في object هنا لا قالك حول ال object لصيغه معينه وبعدين ابتعته كله علي بعضه

انك تحول ال object لصيغه معينه ده اسمه serialization لكن لو عايز تعمل العكس ده اسمه deserialization يعني تحول الصيغه ل object زي مكان فيه أنواع من الصيغ اللي تقدر تحول ليها ال object بتابعك :-

١ - **Binary serialization** :- وهيا انك تحول ال object binary ودي اسرع طريقة لو هتسعمل ال serialization في نفس البرنامج وعشان تستخدمنا بتسدعي مكتبه اسمها كده بس عيبه انك ماتقدر تفتحه وتقراء

System.Runtime.Serialization.Formatters.Binary

٢ - **Xml serialization** :- انك بتحط ال object بصيغة xml وده بيكون مفروء يعني تقدر تقراء وكمان كل ال operating systems بتقدر تقراء ويفضل انك تتعامل بيها لو هتبعت ال object من نظام تشغيل لنظام تشغيل تاني زي من ويندوز ل os مثل دى المكتبه بتابعته

System.Xml.Serialization

٣ - **JSON serialization** :- وده زيه زي ال xml بس اخف منه وده استخدامه شائع مع ال AJAX وال web APIs

دي المكتبه بتابعته

System.Runtime.Serialization.Json

What is **Serialization**?

- **Serialization** in C# refers to the process of converting an object or a data structure into a format that can be easily stored, transmitted, or reconstructed.

Purpose of **Serialization**?

- The primary purpose of serialization is to persistently store the state of an object or to send it over a network.

What is **Deserialization**?

- The reverse process, which involves reconstructing the object from its serialized form, is called deserialization.

Serialization Formats in C#:

- **Binary Serialization**: This format is efficient but not human-readable.
- It's suitable for saving object state within the same platform.
- **Library to use**:
`System.Runtime.Serialization.Formatters.Binary`

Serialization Formats in C#:

- **XML Serialization**: Objects are serialized into XML format, which is both human-readable and platform-independent.
- XML serialization is commonly used when interoperability with other systems is required.
- **Library to use**:
`System.Xml.Serialization`

Serialization Formats in C#:

- **JSON Serialization**: Similar to XML, JSON is a human-readable and lightweight data interchange format.
- It is commonly used for web APIs and AJAX requests.
- **Library to use**:
`System.Runtime.Serialization.Json`

When to use Serialization?

- Data Persistence
- Communication Between Applications
- Cross-Language Communication
- Web Development
- Many other places.

Serialization in C# refers to the process of converting an object or a data structure into a format that can be easily stored, transmitted, or reconstructed. The primary purpose of serialization is to persistently store the state of an object or to send it over a network. The reverse process, which involves reconstructing the object from its serialized form, is called deserialization.

Key Points about Serialization in C#:

- Format:
 - Serialization typically involves converting an object into a format such as XML, JSON, binary, or other custom formats.
- Object State:
 - During serialization, the entire state of an object, including its data members and their values, is saved. This allows the object to be reconstructed later with the same state.
- Interfaces:

- Objects that need to be serialized often implement the `Serializable` interface. In C#, this is achieved by marking a class with the `[Serializable]` attribute.

```
[Serializable]  
public class MyClass  
{  
    // Class members and methods  
}
```

- **Serialization Formats in C#:**

- **Binary Serialization:** This format is efficient but not human-readable. It's suitable for saving object state within the same platform.
- **XML Serialization:** Objects are serialized into XML format, which is both human-readable and platform-independent. XML serialization is commonly used when interoperability with other systems is required.
- **JSON Serialization:** Similar to XML, JSON is a human-readable and lightweight data interchange format. It is commonly used for web APIs and AJAX requests.

- **Serialization Libraries:**

- In C#, the `System.Runtime.Serialization` namespace provides classes for binary and XML serialization. The `DataContractJsonSerializer` class is commonly used for JSON serialization.

```
// Binary Serialization  
using System.Runtime.Serialization.Formatters.Binary;  
  
// XML Serialization  
using System.Xml.Serialization;  
  
// JSON Serialization  
using System.Runtime.Serialization.Json;
```

Serialization is a crucial aspect of many applications, especially when working with distributed systems, databases, or any scenario where object state needs to be persisted or transmitted between different parts of a system.

When to use Serialization?

Serialization is used in various scenarios to persistently store object state, transmit data over a network, or share data between different parts of an application. Here are common situations where serialization is beneficial:

- Data Persistence:
 - Use Case: Storing and retrieving object state from storage, such as databases or files.
 - Example: Saving application settings, user preferences, or any data that needs to persist between program executions.
- Communication Between Applications:
 - Use Case: Transmitting data between different applications or services over a network.
 - Example: Web API communication, inter-process communication (IPC), or sending data between a client and server.
- Object Copying:
 - Use Case: Creating deep copies of objects.
 - Example: Cloning an object to preserve its state at a specific point in time or duplicating complex data structures.
- Cross-Language Communication:
 - Use Case: Communicating between programs written in different languages.
 - Example: Exchanging data between a C# application and a Python application or any other combination of languages.
- Caching and Memorization:
 - Use Case: Storing the result of expensive computations to avoid redundant calculations.
 - Example: Caching the results of database queries or complex algorithmic computations.
- Web Development:
 - Use Case: Sending data between the client and server in web applications.

- Example: Serializing data to JSON for transmission between the browser and the server in AJAX requests or web API interactions.
- Message Queues:
 - Use Case: Placing messages in a queue for asynchronous communication between components.
 - Example: Serializing messages before placing them in a message queue for processing by other parts of the system.
- State Management in Distributed Systems:
 - Use Case: Managing state in distributed systems or microservices.
 - Example: Serializing and deserializing messages or data when passing information between different components in a distributed architecture.
- Remoting and RPC (Remote Procedure Call):
 - Use Case: Invoking methods on objects located on a remote machine.
 - Example: Remoting in .NET or using technologies like gRPC for remote procedure calls.
- Versioning and Migration:
 - Use Case: Updating applications while preserving compatibility with previously stored data.
 - Example: Serializing data in a way that allows for versioning and handling changes in the object structure over time.
- Testing:
 - Use Case: Creating consistent test data for unit testing or mocking.
 - Example: Serializing and deserializing objects to ensure that the test data is representative of real-world scenarios.

While serialization provides many benefits, it's essential to consider factors such as performance, security, and interoperability when choosing a serialization method. Additionally, be aware of potential issues, such as versioning challenges when updating object structures.

What is serialization in C#?

Converting an object into a format that can be easily stored, transmitted, or reconstructed

Converting an object into a format that can only be stored

Converting an object into a format that can only be transmitted

Converting an object into a format that can only be reconstructed

What is the primary purpose of serialization?

To persistently store the state of an object

To convert an object into a readable format

To convert an object into a binary format

To transmit an object over a network

Which interface is commonly implemented by objects that need to be serialized?

Serializable

Deserializable

Storable

Transmittable

What are some common serialization formats in C#?

Binary, XML, JSON

Binary, CSV, JSON

XML, CSV, JSON

Binary, XML, YAML

Which namespace provides classes for binary and XML serialization in C#?

System.Runtime.Serialization

System.Xml.Serialization

System.Json.Serialization

System.Binary.Serialization

Which format is suitable for saving object state within the same platform?

Binary Serialization

XML Serialization

JSON Serialization

YAML Serialization

Which format is commonly used when interoperability with other systems is required?

Binary Serialization

XML Serialization

JSON Serialization

YAML Serialization

Which format is commonly used for web APIs and AJAX requests?

Binary Serialization

XML Serialization

JSON Serialization

YAML Serialization

What are some common use cases for serialization?

Data persistence, communication between applications, object copying, cross-language communication

Data persistence, object reconstruction, communication between applications, object copying

Communication between applications, object persistence, object copying, cross-language communication

Data persistence, communication between applications, object reconstruction, cross-language communication

XML Serialization Example

هعمل الأول كلاس person بسيط

```
public class clsPerson {  
    public string Name { get; set; }  
    public int Age { get; set; }  
}
```

تعالي ناخد منه object

```
clsPerson person = new clsPerson {Name="Mohammed",Age=40};
```

عاوزين نخزن ال object ده في ملف صيغته xml ونحط الملف ده عالجهاز بتاعتنا وبعدين نرجع نقر على الملف ونحو اللي فيه ل object تاني

طيب فيه حاجتين لازم تعملهم قبل ماتبدأ التحويل
اول حاجه انك تستدعى المكتبه بتاعت التحويل

```
using System.Xml.Serialization;
```

تاني حاجه انك تعرف الكلاس انه هيتحوال عشان يجهز نفسه طب ازاي؟

قالك فيه حاجه اسمها attribute عباره عن كلمه بين قوسين بتحطها عشان تحدد خاصيه معينه لسه هنيجي لشرح بتاعها بعدين
المهم انك بتطلع الكلمه دي فوق الكلاس

```
[Serializable]  
public class clsPerson {
```

كده انت جاهز للتحويل
عشان نكتب في ملف معين كنا بنستخدم stream writer وعشان نقرأ في ملف بنستخدم stream reader

```
using (TextWriter writer = new StreamWriter("person.xml")) {}  
using (TextReader reader=new StreamReader("person.xml")) {}
```

طيب عشان نعمل عملية التحويل بناخد object من كلاس اسمه XmlSerializer وال
بتاعه بيطلب منا ال constructor datatype

```
XmlSerializer Serializer = new XmlSerializer(typeof(clsPerson));
```

عشان نحول ل xml بنستعدي `serialize` اسمها وعشان نحول من xml بنستعدي `deserialize`

بس كده خلصنا

```
Serializer.Serialize(writer,person);
```

```
clsPerson DeserializedPerson=(clsPerson)Serializer.Deserialize(reader);
```

```
using System;
using System.IO;
using System.Xml.Serialization;
```

```
[Serializable]
public class clsPerson {
    public string Name { get; set; }
    public int Age { get; set; }
}
```

```
class NullableExample
{
    static void Main()
    {
        clsPerson person = new clsPerson {Name="Mohammed",Age=40};

        XmlSerializer Serializer = new XmlSerializer(typeof(clsPerson));

        using (TextWriter writer = new StreamWriter("person.xml"))
        {
            Serializer.Serialize(writer,person);
        }

        using (TextReader reader=new StreamReader("person.xml"))
        {
            clsPerson DeserializedPerson=(clsPerson)Serializer.Deserialize(reader);
            Console.WriteLine($"Name: {DeserializedPerson.Name}, Age: {DeserializedPerson.Age}");
        }

        Console.ReadLine();
    }
}
```

XML Serialization is another common way to serialize objects in C#. Here's the same example using XML Serialization:

```
using System;
using System.IO;
using System.Xml.Serialization;
```

```
[Serializable]
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}

class Program
{
    static void Main()
    {
        // Create an instance of the Person class
        Person person = new Person { Name = "Mohammed Abu-Hadhoud", Age = 46 };

        // XML serialization
        XmlSerializer serializer = new XmlSerializer(typeof(Person));
        using (TextWriter writer = new StreamWriter("person.xml"))
        {
            serializer.Serialize(writer, person);
        }

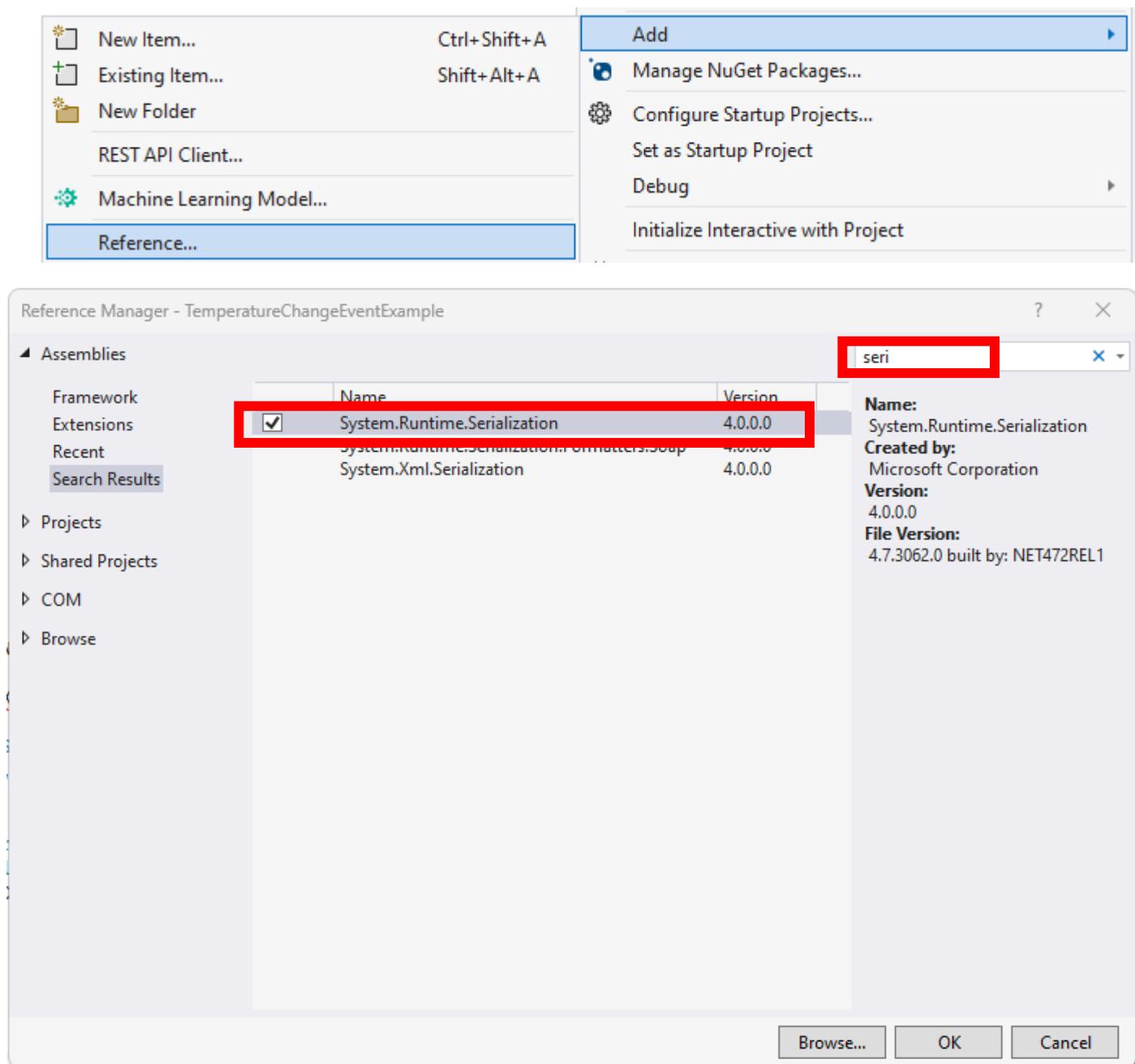
        // Deserialize the object back
        using (TextReader reader = new StreamReader("person.xml"))
        {
            Person deserializedPerson = (Person)serializer.Deserialize(reader);
            Console.WriteLine($"Name: {deserializedPerson.Name}, Age: {deserializedPerson.Age}");
        }
    }
}
```

In this example, the `XmlSerializer` class from the `System.Xml.Serialization` namespace is used to serialize and deserialize the `Person` object. The `XmlSerializer` automatically generates XML tags based on the properties of the object.

Remember that for XML Serialization to work, the class must have a parameterless constructor (either explicitly defined or provided by default) and all properties to be serialized must be public. Additionally, the class or its members can be annotated with attributes to control the serialization process.

Json Serialization Example

عشان تستخدم المكتبه بتاعت ال JSON بتروح تعملها reference الأول



```
using System.Runtime.Serialization.Json;
```

وده نفس السطر بس تغير اسم الكلاس

```
DataContractJsonSerializer Serializer = new  
DataContractJsonSerializer(typeof(clsPerson));
```

وهنا لما بتحول ال object بتحوله في ال memory الأول قبل ما يحطه في ملف

```
using (MemoryStream stream = new MemoryStream()) {  
    Serializer.writeObject(stream, person);  
    string jsonString = System.Text.Encoding.UTF8.GetString(stream.ToArray());  
    File.WriteAllText("person.json", jsonString);  
}
```

والقراءة عادي مافيهاش حاجه

```
using (FileStream stream=new FileStream("person.json", FileMode.Open)) {  
    clsPerson DeserializedPerson=(clsPerson)Serializer.ReadObject(stream);  
    Console.WriteLine($"Name: {DeserializedPerson.Name}, Age: {DeserializedPerson.Age}");  
}
```

JSON Serialization is commonly used in web development and APIs. Here's the same example using JSON Serialization in C#:

```
using System;  
using System.IO;  
using System.Runtime.Serialization.Json;  
  
[Serializable]  
public class Person  
{  
    public string Name { get; set; }  
    public int Age { get; set; }  
}  
  
class Program  
{  
    static void Main()  
    {  
        // Create an instance of the Person class
```

```

Person person = new Person { Name = "Mohammed Abu-Hadhoud", Age = 30 };

// JSON serialization
DataContractJsonSerializer serializer = new DataContractJsonSerializer(typeof(Person));
using (MemoryStream stream = new MemoryStream())
{
    serializer.WriteObject(stream, person);
    string jsonString = System.Text.Encoding.UTF8.GetString(stream.ToArray());

    // Save the JSON string to a file (optional)
    File.WriteAllText("person.json", jsonString);
}

// Deserialize the object back
using (FileStream stream = new FileStream("person.json", FileMode.Open))
{
    Person deserializedPerson = (Person)serializer.ReadObject(stream);
    Console.WriteLine($"Name: {deserializedPerson.Name}, Age: {deserializedPerson.Age}");
}
}

```

In this example, the `DataContractJsonSerializer` class from the `System.Runtime.Serialization.Json` namespace is used to serialize and deserialize the `Person` object. The resulting JSON string is UTF-8 encoded, and you can save it to a file or send it over the network.

Just like with XML Serialization, ensure that the class is marked with the `[Serializable]` attribute, and the properties to be serialized are public. If needed, you can customize the serialization process using attributes like `[DataMember]` for properties. Also, consider using third-party libraries like Newtonsoft.Json (Json.NET) for more advanced JSON serialization scenarios.

Binary Serialization Example

عسان تحول ل `binary` بتسخدم `formatter.serialize` و `deserialize` مفيهاش حاجه صعبه

```

using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Runtime.SerializationJson;

[Serializable]
public class clsPerson {
    public string Name { get; set; }
    public int Age { get; set; }
}

class NullableExample
{

    static void Main()
    {
        clsPerson Person = new clsPerson {Name="Mohammed",Age=40};

        BinaryFormatter formatter = new BinaryFormatter();
        using (FileStream stream = new FileStream("person.bin", FileMode.Create))
        {
            formatter.Serialize(stream, Person);
        }

        // Deserialize the object back
        using (FileStream stream = new FileStream("person.bin", FileMode.Open))
        {
            clsPerson deserializedPerson = (clsPerson)formatter.Deserialize(stream);
            Console.WriteLine($"Name: {deserializedPerson.Name}, Age: {deserializedPerson.Age}");
            Console.ReadKey();
        }
        Console.ReadLine();
    }
}

```

Here's the example using Binary Serialization in C#:

```

using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

[Serializable]
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}

```

```

class Program
{
    static void Main()
    {
        // Create an instance of the Person class
        Person person = new Person { Name = "Mohammed Abu-Hadhoud", Age = 46 };

        // Binary serialization
        BinaryFormatter formatter = new BinaryFormatter();
        using (FileStream stream = new FileStream("person.bin", FileMode.Create))
        {
            formatter.Serialize(stream, person);
        }

        // Deserialize the object back
        using (FileStream stream = new FileStream("person.bin", FileMode.Open))
        {
            Person deserializedPerson = (Person)formatter.Deserialize(stream);
            Console.WriteLine($"Name: {deserializedPerson.Name}, Age: {deserializedPerson.Age}");
            Console.ReadKey();
        }
    }
}

```

In this example, the `BinaryFormatter` class from the `System.Runtime.Serialization.Formatters.Binary` namespace is used for both serialization and deserialization. The resulting binary file (`person.bin`) contains the serialized data of the `Person` object.

Remember that binary serialization is more efficient in terms of size and speed compared to XML or JSON serialization, but the resulting file is not human-readable. Additionally, be cautious when using binary serialization in scenarios where the serialized data might need to be shared across different platforms or languages, as binary formats are less interoperable in such cases.

Attributes In C#

ال attributes هيا طريقة بتمكنك انك تضيف معلومات زياده عالكلاس او ال method او المتغيرات المعلمات زياده دي اسمها meta data

طريقتها انك بتحط القوسين دول [] فوق الكلاس او ال method او أيها كان وفيها منها نوعين custom و built in

ال object زى ال serializable اللي كنا بنستعملها في تحويل صيغة ال built in فيه برضه conditional obsolete وفيه compiler للتاله compiler دول بيستفيد منهم ال attributes

What are Attributes?

- In C#, attributes provide a way to add metadata to your code.
- They are used to provide additional information about elements in your code, such as classes, methods, properties, and so on.
- Attributes are defined using square brackets [] and are placed above the code element they are associated with.

Example

```
[Serializable]
public class MyClass
{
    [Obsolete("This method is deprecated.")]
    public void DeprecatedMethod()
    {
        // Deprecated method implementation
    }

    [Conditional("DEBUG")]
    public void DebugMethod()
    {
        // Code to be executed only in debug mode
    }
}
```

Why Attributes

- **Attributes play a crucial role in:**
- **Enhancing code readability.**
- **Providing additional information (Meta Data)**
- **Enabling frameworks and tools to understand and process your code more effectively.**
- **They are widely used in areas like serialization, documentation, testing, and more.**

In C#, attributes provide a way to add metadata to your code. They are used to provide additional information about elements in your code, such as classes, methods, properties, and so on. Attributes are defined using square brackets [] and are placed above the code element they are associated with.

Here's a basic example of using attributes in C#:

```
[Serializable]
public class MyClass
{
    [Obsolete("This method is deprecated. Use NewMethod instead.")]
    public void DeprecatedMethod()
    {
        // Deprecated method implementation
    }

    [Conditional("DEBUG")]
    public void DebugMethod()
    {
        // Code to be executed only in debug mode
    }
}
```

In this example:

- The `Serializable` attribute is applied to the `MyClass` class, indicating that instances of this class can be serialized.
- The `Obsolete` attribute is applied to the `DeprecatedMethod` method, marking it as deprecated and providing a message that suggests using the `NewMethod` instead.
- The `Conditional` attribute is applied to the `DebugMethod` method, indicating that the method should only be called if the symbol `DEBUG` is defined during compilation.

Attributes play a crucial role in enhancing code readability, providing additional information, and enabling frameworks and tools to understand and process your code more effectively. They are widely used in areas like serialization, documentation, testing, and more.

Serialization Attributes

اتكلمنا قبل كده عن ال `serialize` وانه بيسمح لك انك تحول صيغة ال `object` لكتاب معين وفيه `attribute` جوه الكلاس ده مش عايزة يدخل في ال `non serialized` بكتب فوقيه `serialization`

Serialized Attribute?

- In C#, the term "serialized attribute" refers to the concept of serialization or deserialization.
- **Serialization** is the process of converting an **object** or **data structure** into a format that can be easily stored, transmitted, or reconstructed later.
- **Deserialization** is the reverse process, where the serialized data is converted back into an object.
- **Serialization attributes** are often used to control how objects are serialized or deserialized by indicating how certain members should be treated during the process.

[Serializable] Attribute

[Serializable] Attribute:

The [Serializable] attribute is applied to a class to indicate that its instances can be serialized.

```
[Serializable]
public class MyClass
{
    // Class Code
}
```

[NonSerialized] Attribute

[NonSerialized] Attribute:

The [NonSerialized] attribute Applied to a field to indicate that it should not be serialized.

```
[Serializable]
public class MyClass
{
    // Will be serialized
    public int SerializedField;

    // Will not be serialized
    [NonSerialized]
    public int NonSerializedField;
}
```

In C#, the term "serialized attribute" refers to the concept of serialization attributes used with classes and objects when performing serialization or deserialization.

Serialization is the process of converting an object or data structure into a format that can be easily stored, transmitted, or reconstructed later. Deserialization is the reverse process, where the serialized data is converted back into an object.

Attributes in C# are used to provide metadata about the code elements like classes, methods, or properties. Serialization attributes are often used to control how objects are serialized or deserialized by indicating how certain members should be treated during the process.

Here are some commonly used serialization attributes in C#:

[Serializable] Attribute:

- - The [Serializable] attribute is applied to a class to indicate that its instances can be serialized.
- Example:

```
[Serializable]  
public class MyClass  
{  
    // Class members  
}
```

[NonSerialized] Attribute:

- - Applied to a field to indicate that it should not be serialized.
- Example:

```
[Serializable]  
public class MyClass  
{  
    // Will be serialized  
    public int SerializedField;  
  
    // Will not be serialized  
    [NonSerialized]  
    public int NonSerializedField;  
}
```

These attributes help customize the serialization and deserialization process to meet specific requirements, such as excluding certain fields, renaming elements, or controlling the order of serialization. Depending on the serialization framework or library used (e.g., XML serialization, JSON serialization, binary serialization), different attributes might be employed.

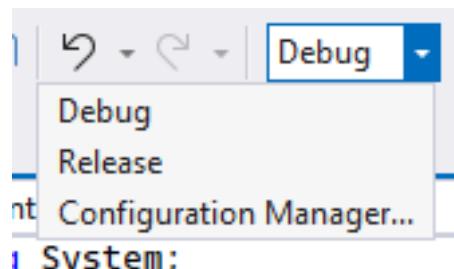
Note: there are more advance attributes for serialization, at your level now you dont need to know them.

Conditional Attribute Examples

ال لو انت فيه method معينه مش عايزها تشتعل غير في ال release لكن في ال debug time مش هتشتعل شايف السطر ده؟

[Conditional("DEBUG")]

لو كتبته فوق mode debug وال mode release هيستغل لو كان debug mode مش هيستغل
ال mode بتعرفه من هنا



لو غيرت ال METHOD وعملته RELEASE مش هتشتعل غير ال العادي
لو جيت اخترت RELEASE وطلعتك الرسالة دي
اخтар اللي علمتك عليه

Just My Code Warning

You are debugging a Release build of TemperatureChangeEventExample.exe. Using Just My Code with Release builds using compiler optimizations results in a degraded debugging experience (e.g. breakpoints will not be hit).

→ Stop Debugging

→ Disable Just My Code and Continue

→ Continue Debugging

→ Continue Debugging (Don't Ask Again)

```
using System;
using System.Diagnostics;
using System.IO;

public class MyClass
{
    [Conditional("DEBUG")]
    public void DebugMethod()
    {
        Console.WriteLine("Debug method excuted.");
    }

    public void NormalMethod()
    {
        Console.WriteLine("Normal Method excuted.");
    }
}

class NullableExample
{
    static void Main()
    {
        MyClass cls = new MyClass();
        cls.NormalMethod();
        cls.DebugMethod();
        Console.ReadLine();
    }
}
```

فاکر محمد هنیدی فی فول الصين العظيم (يوم ورا يوم حببي ماجالي نوم)؟

اهو هوا ده ال conditional

بتحط كلمة سر ومش هيتم تنفيذ ال method غير لما تقول كلمة السر دي

بعض ال method دي حطينا عليها كلمة السر

[Conditional("YOM_WARA_YOM")]

```

public static void LogTrace(string Message)
{
    Console.WriteLine($"[TRACE] {Message}");
}

```

لو عملت قرد الكود ده مش هيتنفذ غير لما تعرف كلمة السر او تشيل ال **CONDITIONAL** دي
طيب هعرف ال **conditional** ازاي؟
قالك فيه طريقتين

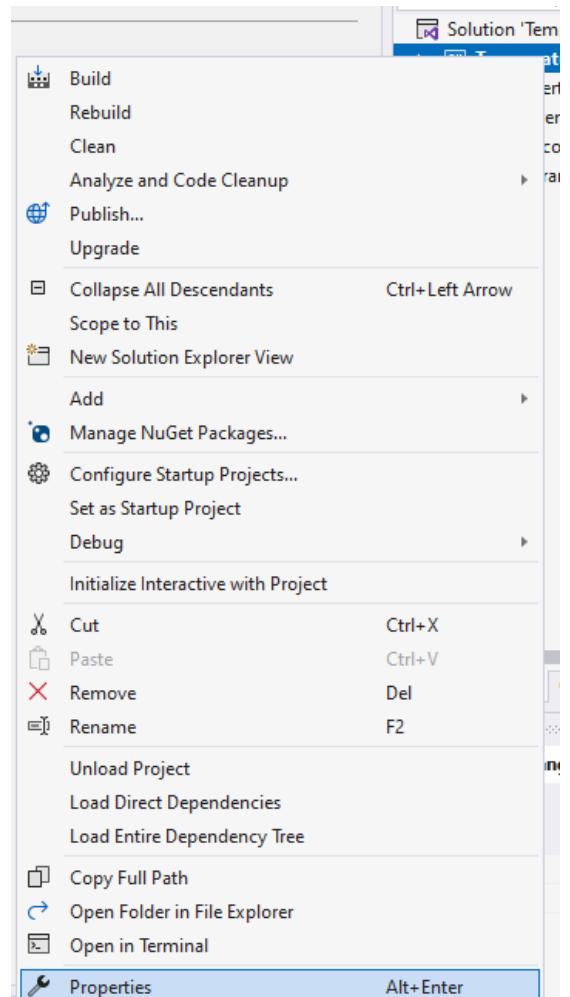
اول طريقة انك تيجي في اول الكود وتكلب **define** وبعدها كلمة السر

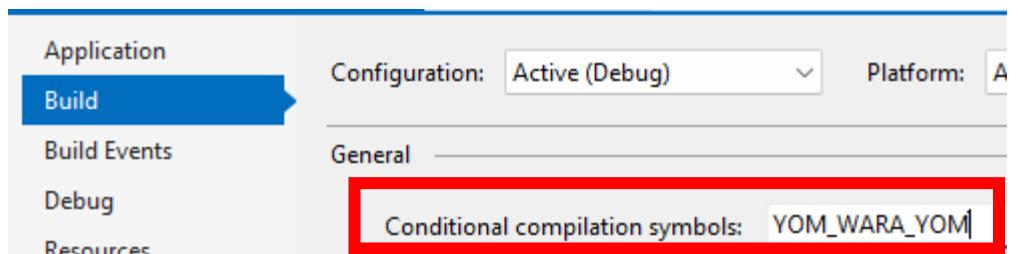
```

#define YOM_WARA_YOM
using System;

```

تاني طريقة انك تعرفها في ال **properties** بتاعت ال **project**





```
#define YOM_WARA_YOM
using System;
using System.Diagnostics;
using System.IO;

public class TraceExample
{
}

class NullableExample
{
    [Conditional("YOM_WARA_YOM")]
    public static void LogTrace(string Message)
    {
        Console.WriteLine($"[TRACE] {Message}");
    }

    static void Main()
    {
        LogTrace("this trace message will be included only if YOM_WARA_YOM is defined");
        Console.WriteLine("Rest of the program");

        Console.ReadLine();
    }
}
```

Conditional Attributes?

In C#, the term "conditional attribute" is commonly associated with the `[Conditional]` attribute. The `[Conditional]` attribute is used in C# to conditionally include or exclude methods from compilation based on the specified compilation symbols. It doesn't affect the runtime behavior of the code but rather influences whether or not a particular method is included in the compiled output.

Below is a simple C# program that demonstrates the use of the `Conditional` attribute. In this example, the `DebugMethod` will only be compiled and executed if the `DEBUG` compilation symbol is defined.

```
using System;
using System.Diagnostics;
```

```
public class MyClass
{
    [Conditional("DEBUG")]
    public void DebugMethod()
    {
        Console.WriteLine("Debug method executed.");
    }

    public void NormalMethod()
    {
        Console.WriteLine("Normal method executed.");
    }
}

class Program
{
    static void Main()
    {
        MyClass myClass = new MyClass();

        // Call the methods
        myClass.DebugMethod(); // This will only be executed in DEBUG builds
        myClass.NormalMethod(); // This will always be executed

        Console.ReadLine();
    }
}
```

Here's how you can compile and run this program:

- In a development environment like Visual Studio, you can set the build configuration to "Debug" or "Release" to see the difference. The `DebugMethod` will be included only in the "Debug" build.

- Alternatively, you can use the `#define` directive to manually define the `DEBUG` symbol in your code, like this:

```
#define DEBUG
```

Place the `#define DEBUG` line at the beginning of your code file, and then both `DebugMethod` and `NormalMethod` will be included in the compilation.

Remember that the `Conditional` attribute affects the inclusion of the method at compile-time, not runtime. So, if the `DEBUG` symbol is not defined during compilation, the `DebugMethod` calls will not be present in the compiled code.

Custom Symbol Example:

Example of using the `Conditional` attribute in combination with trace statements, you might be interested in incorporating conditional compilation of trace statements based on the presence of a compilation symbol. Here's a simple example:

```
#define TRACE_ENABLED

using System;
using System.Diagnostics;

public class TraceExample
{
    [Conditional("TRACE_ENABLED")]
    public static void LogTrace(string message)
    {
        Console.WriteLine($"[TRACE] {message}");
    }

    public static void Main()
    {
```

```

        LogTrace("This trace message will only be included if TRACE_ENABLED is defined.");
        Console.WriteLine("Rest of the program.");

        Console.ReadLine();
    }
}

```

In this example:

- The `TRACE_ENABLED` compilation symbol is defined at the beginning of the file using `#define`.
- The `LogTrace` method is marked with the `Conditional("TRACE_ENABLED")` attribute. This means that the method calls will only be included in the compiled code if the `TRACE_ENABLED` symbol is defined during compilation.
- In the `Main` method, a trace message is logged using `LogTrace`, and it will only be included if `TRACE_ENABLED` is defined.

You can experiment with this example by commenting out or removing the `#define TRACE_ENABLED` line. When the symbol is not defined, the trace messages will be excluded during compilation.

Remember, the `Conditional` attribute is a compile-time directive, so it affects what gets compiled into the executable based on the presence or absence of the specified compilation symbols.

Obsolete Attribute

ال `obsolete attribute` دى خاصه بالمبرمجين انه لو فيه حد بيستخدم `method` معينه في الكلاس بتاعي وبعدين عملت تحديثات وعايز اشيل ال `method` دى وابدلها بواحده تانيه مش بشيلها على طول لا بعمل زي `hint` او تحذير ان ال `method` دى هلغتها في التحديثات الجايه واحسنلاك تستخدموه تانيه حصلت معاي كتير في الاندرويد لما كنت بذاكر من كورس قديم وكان فيه `methods` معينه جوجل لغت الدعم عنها لأنها عملت حاجات افضل فكان بيحط عليها خط لأنها مشطوبه ال `methods` القديمه دى بنقول عليها `deprecated` يعني تعالى نشوف بتعمل ازاي هنا

```

using System;

public class MyClass
{

```

```

[Obsolete("this Method will be deprecated")]
public void Method1() {}
public void Method2() {}
}

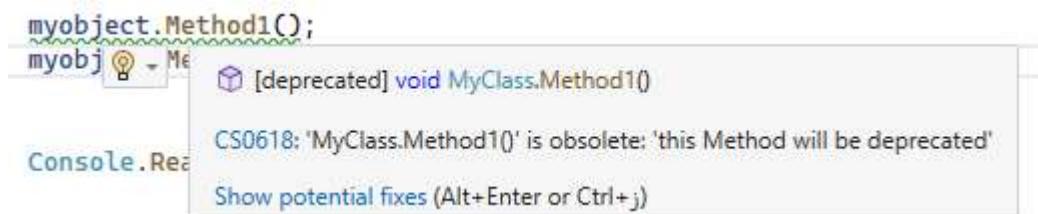
class NullableExample
{
    static void Main()
    {
        MyClass myobject=new MyClass();

        myobject.Method1();
        myobject.Method2();

        Console.ReadLine();
    }
}

```

هذا يظهر لك خط اخضر



The `Obsolete` attribute in C# is used to mark program entities (such as classes, methods, properties, etc.) that are considered obsolete or deprecated. This attribute informs developers that the marked entity should not be used because it is outdated or will be removed in future versions of the code. It also allows you to provide a custom message to suggest an alternative or explain the reason for deprecation.

Here's an example of using the `Obsolete` attribute in a C# program:

```

using System;

public class MyClass
{
    [Obsolete("This method is marked as obsolete, and will be deprecated in the future.")]
    public void Method1()
    {
        Console.WriteLine("This method is marked as obsolete, and will be deprecated in the
future.");
    }
}

```

```

public void Method2()
{
    Console.WriteLine("This is the recommended method to use.");
}

}

class Program
{
    static void Main()
    {
        MyClass myObject = new MyClass();

        // Deprecated method usage
        myObject.Method1(); // Generates a compiler warning

        // New method usage
        myObject.Method2();

        Console.ReadLine();
    }
}

```

In this example:

- The `Method1` in the `MyClass` class is marked with the `Obsolete` attribute. The attribute includes a custom message indicating that the method is obsolete and suggesting the use of `Method2` instead.
- When you compile the program, calling `Method1` will generate a compiler warning. This warning serves as a notification to developers that the method is deprecated, and they should consider using the recommended alternative.
- The `Method2` is introduced as a replacement for the deprecated method, and it can be used without generating any warnings.

Keep in mind that while the `Obsolete` attribute helps communicate to developers that certain code is deprecated, it's ultimately up to the development team to manage the deprecation process and migrate to newer alternatives. It's a good practice to provide clear and informative messages when marking code as obsolete to guide developers on how to proceed.

Quiz

What is the role of attributes in C#?

To enhance code readability

Enabling frameworks and tools to understand and process your code more effectively

To provide additional information and enhance code readability

To enable frameworks and tools to process code more effectively

All of the above

How are attributes defined in C#?

Using curly braces {}

Using angle brackets <>

Using square brackets []

Using parentheses ()

What does the [Serializable] attribute indicate?

The class is obsolete

The class can be serialized

The class is only executed in debug mode

The class is not included in the compiled output

What does the [NonSerialized] attribute indicate?

The field should be excluded from serialization

The field can be serialized

The field is deprecated

The field should only be executed in debug mode

What does the [Conditional] attribute indicate in the given example?

The method is deprecated

The method is only executed in debug mode

The method is not included in the compiled output

The method is conditionally included in the compiled output if it's symbol is defined

What does the [Obsolete] attribute indicate in the given example?

The method is deprecated

The method is only executed in debug mode

The method is not included in the compiled output

Instances of the class can be serialized

Custom Attribute

خلصنا من ال built in attributes نيجي لـ **custom attribute** هنا بيقولك تقدر تعمل ال attribute الخاص بيك عشان تستخدمه في أي وقت او أي مكان انت عايزه عشان تعمل ال attribute بتعمل كلاس بيورث من **System.Attribute** تعالى نعمل كلاس خفي

```
public class MyCustomAttribute : Attribute
{
    public string Description { get; }

    public MyCustomAttribute(string description)
    {
        Description = description;
    }
}
```

كلاس اهو مافيهوش حاجة

ناقص بس حاجة واحد و هيا انك تحدد الاستخدام بتاعه فبتيجي فوق الكلاس وبتحدد هل هتستخدمه لل classes ولا لل methods ولايه بالضبط عن طريق انك تكتب AttributeTargets وبعدها نقطه وتختر اللي انت عايزه وبعدين لو هتستخدمه مره واحد ولا اكتر من مره على نفس العنصر

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = true)]
```

ود الكود كله

```
using System;

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = true)]
public class MyCustomAttribute : Attribute
{
    public string Description { get; }
```

```
public MyCustomAttribute(string description)
{
    Description = description;
}
```

```
[MyCustom("This is a class attribute")]
class MyClass
{
    [MyCustom("This is a method attribute")]
    public void MyMethod()
    {
        // Method implementation
    }
}
```

```
class NullableExample
{
    static void Main()
    {
        Console.ReadLine();
    }
}
```

In C#, a custom attribute is a user-defined metadata that you can apply to elements in your code, such as classes, methods, properties, or parameters. Attributes provide a way to add declarative information to your code, which can be used by the runtime, tools, or other code to perform specific actions or make decisions.

To define a custom attribute, you create a class that inherits from the `System.Attribute` class or one of its derived classes. The attribute class can then be applied to various elements in your code using square brackets.

Here's a simple example of a custom attribute:

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = true)]
public class MyCustomAttribute : Attribute
{
    public string Description { get; }

    public MyCustomAttribute(string description)
    {
        Description = description;
    }
}
```

```
}
```



```
[MyCustom("This is a class attribute")]
class MyClass
{
    [MyCustom("This is a method attribute")]
    public void MyMethod()
    {
        // Method implementation
    }
}
```

In this example, `MyCustomAttribute` is a custom attribute that takes a description as a parameter. The attribute can be applied to classes and methods. The `AttributeUsage` attribute is used to specify where the custom attribute can be applied (`AttributeTargets.Class` and `AttributeTargets.Method` in this case) and whether it can be applied multiple times to the same element (`AllowMultiple = true`).

You can then use reflection or other mechanisms to access and utilize the information provided by these custom attributes at runtime. Custom attributes are often used for various purposes such as code generation, documentation, or influencing the behavior of frameworks and libraries.

Note When we study reflection we will explain it.

What is Reflection?

ال reflection هيا مكتبات بتمكنا انك تدخل علي كود اي حد وتشوف كل حاجه فيه تقدر تشووف ال parameters وال datatypes وال methods في ال run time

اعتبره هاكر بيكتشفلك الكود بتاع الشخص

زي مثلا في ال serialization لما كان بيتدعني ال object.serialize اللي جوه ال object ويتعامل

ال kod اللي بتكتشفه ده انت مش عارف عنه حاجه في ال compile time بتعرف في ال runtime

لو انت مثلا بتعمل reference لملف dll بتلاقيه كشفاك كل اللي فيه ده بيتم عن طريق ال reflection هتلaci التطبيقات بتاعته تحت

هنا بيقولك انك لازم تخلي بالك وانت بتسخدم ال reflection لانه بيأثر عالسرعه وممكن في ال datatype يضرب بسبب runtime

Let us think ...

- How did they do the serialization?
 - They needed a way to navigate through the given object ☺
 - This way using the ...

Reflection

What is Reflection?



- Reflection in C# refers to the ability of a program to inspect, own structure, metadata, and behavior at runtime.
- Reflection in C# refers to the ability of a program to inspect, query, and interact with the metadata of types and members in an assembly at runtime.
- With reflection, you can obtain information about types, fields, properties, methods, and other members of your code dynamically, without knowing them at compile time.
- With reflection, you can dynamically load assemblies, examine and create types, and invoke methods or access properties, all without knowing them at compile-time. It provides a way to manipulate types, objects, and members dynamically.

Key aspects of reflection in C# include:

1. Type Information: You can retrieve information about a type such as its name, namespace, methods, properties, fields, events, and more.
2. Instantiation: You can create an instance of a type dynamically at runtime using the `Activator.CreateInstance` method.
3. Method Invocation: You can invoke methods on an object dynamically, even if you don't know the method at compile time, using the `Invoke` method.
4. Property and Field Access: You can get and set the values of properties and fields dynamically.



Practical uses of reflection

Serialization

Development of Frameworks

Development of unit testing frameworks

Data access frameworks

Generate UI

extensibility mechanisms – Plugins

Validation Frameworks

Certain Design Patterns

And many other uses

Reflection in C# refers to the ability of a program to inspect its own structure, metadata, and behavior at runtime.

Reflection in C# refers to the ability of a program to inspect, query, and interact with the metadata of types and members in an assembly at runtime. With reflection, you can obtain information about types, fields, properties, methods, and other members of your code dynamically, without knowing them at compile time.

With reflection, you can dynamically load assemblies, examine and create types, and invoke methods or access properties, all without knowing them at compile-time. It provides a way to manipulate types, objects, and members dynamically.

Key aspects of reflection in C# include:

1. Type Information: You can retrieve information about a type, such as its name, namespace, methods, properties, fields, events, and more.
2. Instantiation: You can create an instance of a type dynamically at runtime using the `Activator.CreateInstance` method.
3. Method Invocation: You can invoke methods on an object dynamically, even if you don't know the method at compile time, using the `Invoke` method.
4. Property and Field Access: You can get and set the values of properties and fields dynamically.

Reflection is powerful but comes with some trade-offs:

- Performance: Reflection can have a performance overhead compared to statically typed code because it bypasses some of the optimizations performed by the compiler.
- Type Safety: Since reflection allows dynamic interaction with types, there's a risk of runtime errors if the code is not carefully designed.

Practical uses of reflection:

- One practical use of reflection in C# is in the development of frameworks and tools where the structure of types is not known at compile time. Reflection enables these frameworks to dynamically discover and interact with types provided by user code. One such example is in the development of dependency injection containers.
- Another practical use of reflection in C# is in the development of serialization and deserialization frameworks. These frameworks convert objects into a format that can be easily stored, transmitted, or reconstructed, and reflection is often used to dynamically inspect and manipulate the structure of objects.

- Another practical use of reflection in C# is in the development of unit testing frameworks. Unit testing frameworks often utilize reflection to dynamically discover and execute test methods within test classes.
- Another practical use of reflection in C# is in the development of data access frameworks or Object-Relational Mapping (ORM) tools. Reflection can be employed to dynamically map object properties to database fields, allowing for flexible and generic data access without explicit knowledge of the underlying database schema.
- Another practical use of reflection in C# is in the development of user interface frameworks or libraries that need to dynamically generate or bind UI elements based on the properties of data objects.
- Another practical use of reflection in C# is in the implementation of extensibility mechanisms, where the application can dynamically discover and load extensions or plugins at runtime without having to know about them at compile time.
- Another practical use of reflection in C# is in the development of attribute-based validation frameworks. By utilizing custom attributes and reflection, you can create a system where validation rules are associated with class properties, making it easier to define and enforce data validation rules.
- Another practical use of reflection in C# is in the development of code generation tools or frameworks. Reflection allows you to inspect and analyze the structure of types at runtime, and this capability is often leveraged in code generation scenarios.
- Another practical use of reflection in C# is in certain design patterns.
- Reflection can also be utilized in the context of documentation generation, where you dynamically extract information about types, methods, properties, and other elements in your codebase to generate documentation automatically.
- and may other uses :-)

Quiz

What does reflection in C# refer to?

The ability of a program to inspect its own structure, metadata, and behavior at runtime.

The ability of a program to query and interact with the metadata of types and members in an assembly at runtime.

The ability of a program to dynamically load assemblies, examine and create types, and invoke methods or access properties at runtime.

The ability of a program to manipulate types, objects, and members dynamically.

All of the above

What is one key aspect of reflection in C#?

Type Information

Instantiation

Method Invocation

Property and Field Access

All of the above

What are some trade-offs of using reflection?

Performance overhead

Type safety risk

Lack of compile-time knowledge

All of the above

Which of the following is a practical use of reflection in C#?

Development of serialization and deserialization frameworks

Development of unit testing frameworks

Used in certain design patterns

All of the above

What is one practical use of reflection in C#?

Development of data access frameworks or ORM tools

Development of user interface frameworks or libraries

Implementation of extensibility mechanisms

All of the above

'Type' Class

ال reflection بت تكون من مكتبات

فيه كلاس أساسى اسمه type الكلاس ده بيخلبك توصل لمعلومات ال نفسه يعني مثلا عندك كلاس اسمه person تقدر توصل للمعلومات اللي فيه عن طريق انك تخزنها في ال object من الكلاس type

What is Type Class?



- The Type class is a central class in reflection.
- It representing a type in C#.
- You can use it to get information about a type, such as its methods, properties, fields, and events.

```
Type type = typeof(string);
```

```
Console.WriteLine("Type Information:");
```

```
Console.WriteLine($"Name: {type.Name}");
Console.WriteLine($"Full Name: {type.FullName}");
Console.WriteLine($"Is Class: {type.IsClass}");
```

Type Class:

The `Type` class is a central class in reflection, representing a type in C#. You can use it to get information about a type, such as its methods, properties, fields, and events.

Example 1: Getting Type Information

```
using System;

class Program
{
    static void Main()
    {
        Type myType = typeof(string);

        Console.WriteLine("Type Information:");
        Console.WriteLine($"Name: {myType.Name}");
        Console.WriteLine($"Full Name: {myType.FullName}");
        Console.WriteLine($"Is Class: {myType.IsClass}");

        Console.ReadLine();
    }
}
```

What is the purpose of the Type class in C#?

To represent a type in reflection

To create new types in C#

To store data about objects in C#

To perform mathematical operations in C#

Navigate String Library Using Reflection Example

هناخد ال Assembly assembly نخزنها في string library ونطبعها عالشاشة
اول حاجه هناخد object من النوع Assembly وده اللي بيتحط فيه المكتبات
وبحدد فيه ال data type اللي عايز استكشفها

```
Assembly assembly = typeof(string).Assembly;
```

بعدين هناخد object من Assembly ونخزن فيه ال datatype اللي عاوزين
نستكشفه

```
Type type = assembly.GetType("System.String");
```

هستدعى المكتبه linq عشان تسهل عليا عمليات ال sorting

```
using System.Linq;
```

بعدين لو كان ال type مش فاضي هاتلي كل ال public methods وكل ال اللي بتظهرلي لما باخد object وتقدر تحدد أي حاجه تانيه عن طريق ال binding.flag وبعدين برتبهم بالاسم

```
var Methods=type.GetMethods(BindingFlags.Public|BindingFlags.Instance)  
.OrderBy( method => method.Name );
```

بعدين هلف عليهم واطبعهم

```
foreach (var method in Methods)  
{  
    Console.WriteLine($"{method.ReturnType} {method.Name}({GetParameterList(method.GetParameters())});  
}  
} else { Console.WriteLine("System.String type not found."); }
```

ودي function عشان تضبط بيانات ال methods في سطر واحد

```
static string GetParameterList(ParameterInfo[] parameters)  
{  
    return string.Join(", ", parameters.Select(parameter => $"{parameter.ParameterType} {parameter.Name}"));  
}
```

```
using System;  
using System.Reflection;  
using System.Linq;  
  
class NullableExample  
{
```

```

public class Person {
    public string Name { get; set; }

    public void Method() {}

}

static void Main()
{
    Assembly assembly = typeof(string).Assembly;

    Type type = assembly.GetType("System.String");

    if (type!=null) {

        var Methods=type.GetMethods(BindingFlags.Public|BindingFlags.Instance)
            .OrderBy( method => method.Name );

        foreach (var method in Methods)
        {
            Console.WriteLine($"{method.ReturnType} {method.Name}({GetParameterList(method.GetParameters())})");
        }
    } else { Console.WriteLine("System.String type not found."); }

    Console.ReadLine();
}
static string GetParameterList(ParameterInfo[] parameters)
{
    return string.Join(", ", parameters.Select(parameter => $"{parameter.ParameterType} {parameter.Name}"));
}
}

```

انا جيت عمل السطر ده بدون ال assembly واشتعل عادي

Type type = typeof(string);

Navigating the entire .NET string library using reflection is not a straightforward task, as it involves exploring the assemblies, types, methods, and properties related to strings. However, I can provide you with a simple example that uses reflection to list all methods of the `System.String` class in the `mscorlib` assembly.

```

using System;
using System.Linq;
using System.Reflection;

class Program
{
    static void Main()
    {

```

```
// Get the assembly containing the System.String type
Assembly mscorlib = typeof(string).Assembly;

// Get the System.String type
Type stringType = mscorlib.GetType("System.String");

if (stringType != null)
{
    Console.WriteLine($"Methods of the System.String class:\n");

    // Get all public methods of the System.String class
    var stringMethods = stringType.GetMethods(BindingFlags.Public | BindingFlags.Instance)
        .OrderBy(method => method.Name);

    foreach (var method in stringMethods)
    {
        Console.WriteLine($"{method.ReturnType}{method.Name}({GetParameterList(method.GetParameters())})");
    }
    else
    {
        Console.WriteLine("System.String type not found.");
    }

    Console.ReadKey();
}

static string GetParameterList(ParameterInfo[] parameters)
{
    return string.Join(", ", parameters.Select(parameter => $"{parameter.ParameterType}{parameter.Name}"));
}
```

```
    }  
}
```

This example does the following:

1. Retrieves the `mscorlib` assembly, which contains fundamental types like `System.String`.
2. Obtains the `System.String` type.
3. Lists and prints the names and signatures of all public methods of the `System.String` class.

Keep in mind that this example only scratches the surface, and the .NET Framework has a comprehensive string library with many methods, properties, and functionalities. The methods obtained by reflection include not only commonly used methods but also internal methods and methods that are less commonly used. Additionally, newer versions of .NET may introduce new members to the `System.String` class.

Reflection Example 2 - Dynamically Create Objects and Invoke Methods.

مثال تاني

```
using System;  
using System.Reflection;  
using System.Linq;  
using MyNameSpace;  
  
namespace MyNameSpace {  
    public class MyClass {  
        public int Property1 { get; set; }  
  
        public void Method1() {  
            Console.WriteLine("\tMethod1 is Executed");  
        }  
  
        public void Method2(int Value1, string Value2)  
        {  
            Console.WriteLine($"\\tMethod2 is Executed with parameters: {Value1},{Value2}");  
        }  
    }  
}  
class NullableExample  
{  
    static string GetParameterList(ParameterInfo[] parameters)  
    {  
        return string.Join(", ", parameters.Select(parameter => $"{parameter.ParameterType} {parameter.Name}"));  
    }  
  
    static void Main()
```

```

{
    Type type = typeof(MyClass);

    Console.WriteLine($"Type Name: {type.Name}");
    Console.WriteLine($"Type Full Name: {type.FullName}");
    Console.WriteLine("\nProperties:");

    foreach (var property in type.GetProperties()) {
        Console.WriteLine($"{property.Name}, {property.PropertyType}");
    }

    foreach (var method in type.GetMethods())
    {
        Console.WriteLine($"{method.ReturnType}, {method.Name}({GetParameterList(method.GetParameters())})");
    }

    object myClassInstance=Activator.CreateInstance(type);

    type.GetProperty("Property1").SetValue(myClassInstance,42);
    Console.WriteLine("\nSet Property1 to 42 using reflection:");

    Console.WriteLine("\nGetting Property1 value using reflection:");
    int PropertyValue=(int)type.GetProperty("Property1").GetValue(myClassInstance, null);
    Console.WriteLine($"{PropertyValue}");

    Console.WriteLine("\nExcuting Methods using reflection:\n");
    type.GetMethod("Method1").Invoke(myClassInstance, null);

    object[] Prameters = { 100, "Mohamed" };
    type.GetMethod("Method2").Invoke(myClassInstance, Prameters);
    Console.ReadLine();
}

}

```

Reflection with Custom Attributes

عرفنا attribute عادي كلاس وزودنا فوقية سطر

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = true)]
public class MyCustomAttribute : Attribute
{
    public string Description { get; }

    public MyCustomAttribute(string description)
    {
        Description = description;
    }
}
```

دلوقي ال constructor بتابع ال attribute ده هيطلب قيمة من النوع string فاي كلاس او method هتتيجي تضيقه ال attribute ده لازم يكتب فيه أي حاجه

احنا عاملينه عشان تقدر تحط فيه description للكود اللي بتكتبه سواء كان كلاس او method لو هتعمل documentation مثلًا تقدر تستخدم ال reflection في انك تستدعي بيانات الكلاس بما فيهن ال description وتحطهم في ملف word او text حتى

هنا هنعمل بقى كلاس ونضيف له ال attribute

```
[MyCustom("This is a class attribute")]
class MyClass
{
    [MyCustom("This is a method attribute")]
    public void MyMethod()
    {
        // Method implementation
    }
}
```

في ال main بلف عال attributes وبطبعها

```
Type type = typeof(MyClass);

// Get class-level attributes
object[] classAttributes = type.GetCustomAttributes(typeof(MyCustomAttribute), false);

foreach (MyCustomAttribute attribute in classAttributes)
{
    Console.WriteLine($"Class Attribute: {attribute.Description}");
}
```

وهنا بجيبي ال attributes بتاعت المmethod وطبعها ممكن بقى لو عندك اكتر من تلف عليهم وتنادي ال attributes بتاعهم

```
// Get method-level attributes
MethodInfo MethodInfo = type.GetMethod("MyMethod");

object[] methodAttributes = MethodInfo.GetCustomAttributes(typeof(MyCustomAttribute), false);

foreach (MyCustomAttribute attribute in methodAttributes)
{
    Console.WriteLine($"Method Attribute: {attribute.Description}");
}
```

In C#, a custom attribute is a user-defined metadata that you can apply to elements in your code, such as classes, methods, properties, or parameters. Attributes provide a way to add declarative information to your code, which can be used by the runtime, tools, or other code to perform specific actions or make decisions.

Using custom attributes in C# involves applying them to elements in your code and then retrieving or utilizing that metadata at runtime. Here's a step-by-step guide on how to use custom attributes:

Step 1: Define the Custom Attribute

Create a class that inherits from `System.Attribute` or one of its derived classes. Add any properties or fields that you need to store metadata. Here's an example:

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = true)]
public class MyCustomAttribute : Attribute
{
    public string Description { get; }

    public MyCustomAttribute(string description)
    {
        Description = description;
    }
}
```

In this example, `MyCustomAttribute` is a custom attribute that takes a description as a parameter. The attribute can be applied to classes and methods. The `AttributeUsage` attribute is used to specify where the custom attribute can be applied (`AttributeTargets.Class` and `AttributeTargets.Method` in this case) and whether it can be applied multiple times to the same element (`AllowMultiple = true`).

Step 2: Apply the Custom Attribute

Apply the custom attribute to the desired elements in your code using square brackets:

```
[MyCustom("This is a class attribute")]
class MyClass
{
    [MyCustom("This is a method attribute")]
    public void MyMethod()
    {
        // Method implementation
    }
}
```

In this example, the `MyCustomAttribute` is applied to both the `MyClass` class and the `MyMethod` method.

Step 3: Retrieve Attribute Information at Runtime

You can use reflection to retrieve information about the custom attribute at runtime. Here's an example:

```
using System;
using System.Reflection;

class Program
{
    static void Main()
    {
        // Get the type of MyClass
        Type type = typeof(MyClass);

        // Get class-level attributes
        object[] classAttributes = type.GetCustomAttributes(typeof(MyCustomAttribute), false);
        foreach (MyCustomAttribute attribute in classAttributes)
        {
            Console.WriteLine($"Class Attribute: {attribute.Description}");
        }

        // Get method-level attributes
        MethodInfo methodInfo = type.GetMethod("MyMethod");
        object[] methodAttributes = methodInfo.GetCustomAttributes(typeof(MyCustomAttribute), false);
        foreach (MyCustomAttribute attribute in methodAttributes)
        {
            Console.WriteLine($"Method Attribute: {attribute.Description}");
        }
    }
}
```

In this example, we use reflection to get information about the `MyClass` type and its `MyMethod` method. We then retrieve and print the custom attribute information.

Step 4: Compile and Run

Compile your code and run it. The output should display the descriptions provided in the custom attributes.

This is a basic example, and in real-world scenarios, custom attributes can be more complex and used for various purposes, such as configuration, validation, or documentation. The key is to apply them to the appropriate elements in your code and use reflection to access the attribute information at runtime.

Custom Attributes for Validation Example

هنسخدم ال `custom attribute` عشان نعمل ال `validation`

دلوقي عملنا `attribute` بياخد منك رقمين وبتقدر تعمل فيه `error message` بحيث انك تقدر بعد كده تعمل متغير وتضيفله معلومات زي اقل رقم واكتر رقم يقدر يوصله ورسالة الخطأ الخاصه بيها كانك بتعمل كلاس مرتبط بمتغير

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
public class RangeAttribute : Attribute {
    public int Min { get; set; }
    public int Max { get; set; }
    public string ErrorMessage { get; set; }
    public RangeAttribute(int min, int max) { Min = min; Max = max; }
}
```

وهنا عملنا كلاس `Person` وعملنا `attribute` للعمر واديناله القيم

```
public class Person {
    [Range(18,99,ErrorMessage = "Age must be between 18 and 99.")]
    public int Age { get; set; }
    public string Name { get; set; }
}
```

طيب نيجي لل `validation`

الفكره هنا انك بتلف علي كل المتغيرات اللي في الكلاس تشو夫 هل فيها متغير معين تمت إضافة ال `attribute` اللي انا عامله ليه ولا لا؟

لو تمت اضافه ال `attribute` ليه بمسك ال `attribute` اخزنه

وبعدين بمسك القيمه بتاعت المتغير واخزنها برضه

بعدين اقارنهم ببعض واطلع بنتيجه

هنا الفكره انك بتقدر تضيف نفس attribute لاي متغير في الكلاس وتغير حاجات صغير فيه و تعمل واحده تشيك فيها على كل المتغيرات وال method دي كبيره شوية بس احسن ماتقدر تكتب اكتر من method او حتى تعمل switch statement على ١٠٠ متغير مثلا

```
static bool ValidatePerson(Person person)
{
    Type type = typeof(Person);

    foreach (var property in type.GetProperties())
    {
        if (Attribute.IsDefined(property, typeof(RangeAttribute)))
        {
            var rangeAttribute = (RangeAttribute)Attribute.GetCustomAttribute(property, typeof(RangeAttribute));
            int value = (int)property.GetValue(person);

            if (value < rangeAttribute.Min || value > rangeAttribute.Max)
                Console.WriteLine($"Validation failed for property '{property.Name}': {rangeAttribute.ErrorMessage}");
                return false;
            }
        }
    }
    return true;
}
```

في ال main بقى احنا مش حاسين بحاجه

```
static void Main()
{
    Person person = new Person{Age=25,Name="Mohamed"};

    if (ValidatePerson(person))
        Console.WriteLine("Valid");
    else
        Console.WriteLine("not Valid");
    Console.ReadKey();
}
```

وده الكود كله

```
using System;
using System.Reflection;
using System.Linq;
using System.Net.Cache;

[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
public class RangeAttribute : Attribute
{
    public int Min { get; set; }
    public int Max { get; set; }
    public string ErrorMessage { get; set; }
    public RangeAttribute(int min, int max) { Min = min; Max = max; }
}

public class Person
{
    [Range(18,99,ErrorMessage = "Age must be between 18 and 99.")]
    public int Age { get; set; }
    public string Name { get; set; }
}
```

```

}

class NullableExample
{
    static bool ValidatePerson(Person person)
    {
        Type type = typeof(Person);

        foreach (var property in type.GetProperties())
        {
            if (Attribute.IsDefined(property, typeof(RangeAttribute)))
            {

                var rangeAttribute = (RangeAttribute)Attribute.GetCustomAttribute(property, typeof(RangeAttribute));
                int value = (int)property.GetValue(person);

                if (value < rangeAttribute.Min || value > rangeAttribute.Max)
                    Console.WriteLine($"Validation failed for property '{property.Name}': {rangeAttribute.ErrorMessage}");
                return false;
            }
        }
        return true;
    }

    static void Main()
    {
        Person person = new Person{Age=25,Name="Mohamed"};

        if (ValidatePerson(person))
            Console.WriteLine("Valid");
        else { Console.WriteLine("not Valid"); }
        Console.ReadKey();
    }
}

```

Special Comments In C#

ال special comments لما بتجي تقف على built in method مثلا بطلعلك الرساله دي



دي بيقولك بتعمل ب comment بس بيكتب بطريقة ال xml انك تفتح tag وتكتب جواه اللي انت عايزه

ده كلاس اهو عاوزين نعمل فيه الحوار ده

```
public class Calculator {
```

```
public int Add(int a,int b) { return a + b; }

public int Sub(int a, int b) { return a - b; }

}
```

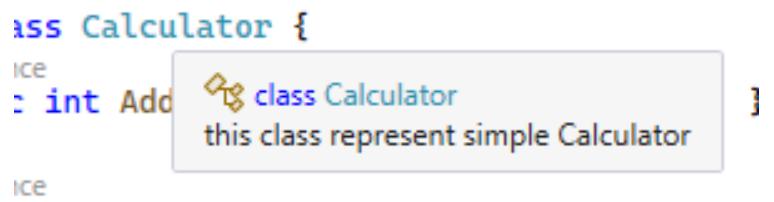
ال **comment** العادي كنا بنعمله ب // ده هنعمله ب //

انت اول ماتحط ال ٣ شرطات هيطلعلك المنظر ده بتكتب اللي انت عايزه في النص

```
/// <summary>
///
/// </summary>
```

```
/// <summary>
/// this class represent simple Calculator
/// </summary>
```

لو جيت وقفت عالكلاس بالماوس هتطلعلك دي



```
class Calculator {
    int Add
}
```

في ال **method** برضه اول ماكتبتي ال ٣ شرطات ظهرت دي

```
/// <summary>
///
/// </summary>
/// <param name="a"></param>
/// <param name="b"></param>
/// <returns></returns>
public int Add(int a,int b) { return a + b; }
```

تقدير تكتب اللي انت عايزه كده

```
/// <summary>
/// Adds two numbers
/// </summary>
/// <param name="a">the first member to be added</param>
/// <param name="b">the second member to be added</param>
```

```
/// <returns>the sum of two numbers</returns>
```

```
using System;
using System.Reflection;
using System.Linq;
using System.Net.Cache;

/// <summary>
/// this class represent simple Calculator
/// </summary>
public class Calculator {

    /// <summary>
    /// Adds two numbers
    /// </summary>
    /// <param name="a">the first member to be added</param>
    /// <param name="b">the second member to be added</param>
    /// <returns>the sum of two numbers</returns>
    public int Add(int a,int b) { return a + b; }

    /// <summary>
    /// Subtracts two numbers
    /// </summary>
    /// <param name="a">the number from which to subtract</param>
    /// <param name="b">the number to be subtracted</param>
    /// <returns>the result</returns>
    public int Sub(int a, int b) { return a - b; }
}

class NullableExample
{

    static void Main()
    {
        Calculator myCalculator = new Calculator();

        int sum = myCalculator.Add(5, 3);

        Console.WriteLine("Sum: " + sum);

        int difference = myCalculator.Sub(8, 3);

        Console.WriteLine("Difference: " + difference);

        Console.ReadKey();
    }
}
```

In C#, method descriptions are typically added using XML comments, which are special comments that provide information about the code and can be used to generate documentation. Here's an example of how you can add descriptions to methods in C# using XML comments:

```
using System;
```

```
/// <summary>
/// This class represents a simple calculator.
/// </summary>
public class Calculator
{
    /// <summary>
    /// Adds two numbers and returns the result.
    /// </summary>
    /// <param name="a">The first number to be added.</param>
    /// <param name="b">The second number to be added.</param>
    /// <returns>The sum of the two numbers.</returns>
    public int Add(int a, int b)
    {
        return a + b;
    }

    /// <summary>
    /// Subtracts the second number from the first number and returns the result.
    /// </summary>
    /// <param name="a">The number from which to subtract.</param>
    /// <param name="b">The number to subtract.</param>
    /// <returns>The result of subtracting the second number from the first number.</returns>
    public int Subtract(int a, int b)
    {
        return a - b;
    }
}

class Program
{
    static void Main()
    {
        // Create an instance of the Calculator class
        Calculator myCalculator = new Calculator();

        // Example usage of the Add method
    }
}
```

```

    int sum = myCalculator.Add(5, 3);

    Console.WriteLine("Sum: " + sum);

    // Example usage of the Subtract method
    int difference = myCalculator.Subtract(8, 3);
    Console.WriteLine("Difference: " + difference);

    Console.ReadKey();
}

}

```

In the example above, the `<summary>` tags provide a brief description of the class and each method. The `<param>` tags are used to describe the parameters of the methods, and the `<returns>` tag is used to describe the return value.

- These comments can be processed by tools like Visual Studio to generate documentation for your code. To view the documentation, you can hover over a method or class, or use the IntelliSense feature in Visual Studio. Additionally, you can use tools like Sandcastle or DocFX to generate more formal documentation from these XML comments.

When you use an Integrated Development Environment (IDE) like Visual Studio, these XML comments are often displayed as tooltips or in the IntelliSense suggestions, providing developers with information about the method while they are writing code.

To generate documentation from these XML comments, you can use tools like Sandcastle or Visual Studio's built-in features. This documentation can be valuable for anyone using your code, as it provides clear and concise information about the purpose and usage of your methods.

Mutable and Immutable Types

Mutable يعني قابل للتغيير و immutable يعني غير قابل للتغيير

ال objects او المتغيرات هل تقدر تعدل فيهم في ال runtime ولا لا
فيه حاجات زي ال string دي مابتغيرش لو جيت غيرتها في ال runtime هوا بيعمل نسخه منه
بالقيمه الجديده لكن القيمة الاصلية مابتغيرش

ال memory mutable بيكون اوفر لل mutable لأنك بتغير عالقيمه الاصليه

من عيوبها انك ممكن لو فيه threads بتسعمل نفس المتغير وتغير على قيمته الاصلية وتبدأ تحصل
مشاكل وده من مميزات ال immutable انه امن

من عيوب ال immutable انك بتعمل اكتر من نسخه لنفس المتغير فبتاكل من المساحة
بيقولك الأفضل انك تستخد ال immutable types عشان دي بتقلل ال bugs بس مش في كل
الأوقات بتقدر تستخدمها فعادي يعني

Mutable and Immutable Types

- In C#, types are classified as either `mutable` or `immutable` based on whether their instances can be modified after they are created.
- Understanding the difference between `mutable` and `immutable` types is crucial for designing robust and maintainable code.

Mutable Types:

- `Mutable types` are types whose `instances` can be modified after they are created.
 - Characteristics: Properties or fields of a `mutable` type can be changed.
 - Changes to an instance affect the state of that instance.
 - Examples include classes, arrays, and custom objects where properties can be modified.

Immutable Types:

- Immutable types are types whose instances cannot be modified after they are created.
 - Characteristics: Properties or fields of an immutable type cannot be changed after the instance is created.
 - Any operation that appears to modify the instance actually returns a new instance with the desired changes.
 - Examples include strings, tuples, and some built-in value types.
 - Another Example a Person Class that has everything as ready only.

Pros and Cons:

Mutable Types:

- Pros:
 - More flexible for certain scenarios.
 - Can be more memory-efficient if state changes frequently.
- Cons:
 - Prone to unintended side effects.
 - May require additional effort to maintain consistency.

Immutable Types:

- Pros:
 - Safer and less error-prone since instances cannot be modified.
 - Easier to reason about and maintain.
- Cons:
 - Creating a new instance for each modification
 - Can be less memory-efficient for certain scenarios.

Guidelines:

- **Favor Immutability:** Whenever possible, prefer using immutable types to reduce bugs related to unintended state changes.
- **Use Mutability When Necessary:** There are scenarios where mutability is more appropriate, such as when frequent state changes are expected or when performance is a critical concern.

In C#, types are classified as either mutable or immutable based on whether their instances can be modified after they are created. Understanding the difference between mutable and immutable types is crucial for designing robust and maintainable code.

Mutable Types:

- Definition: Mutable types are types whose instances can be modified after they are created.
 - Characteristics: Properties or fields of a mutable type can be changed.
 - Changes to an instance affect the state of that instance.
 - Examples include classes, arrays, and custom objects where properties can be modified.

Example of a mutable class in C#:

```
public class MutablePerson
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

Immutable Types:

- Definition: Immutable types are types whose instances cannot be modified after they are created.
 - Characteristics: Properties or fields of an immutable type cannot be changed after the instance is created.
 - Any operation that appears to modify the instance actually returns a new instance with the desired changes.
 - Examples include strings, tuples, and some built-in value types.

Example of an immutable class in C#:

```
public class ImmutablePerson
{
    public string Name { get; }
    public int Age { get; }
```

```
public ImmutablePerson(string name, int age)
{
    Name = name;
    Age = age;
}
```

Pros and Cons:

Mutable Types:

- - Pros: More flexible for certain scenarios.
 - Can be more memory-efficient if state changes frequently.
 - Cons: Prone to unintended side effects.
 - May require additional effort to maintain consistency.

Immutable Types:

- - Pros: Safer and less error-prone since instances cannot be modified.
 - Easier to reason about and maintain.
 - Cons: Creating a new instance for each modification can be less memory-efficient for certain scenarios.

Guidelines:

- - Favor Immutability: Whenever possible, prefer using immutable types to reduce bugs related to unintended state changes.
 - Use Mutability When Necessary: There are scenarios where mutability is more appropriate, such as when frequent state changes are expected or when performance is a critical concern.

Example Usage:

```
// Mutable example
```

```
MutablePerson person1 = new MutablePerson { Name = "Alice", Age = 30 };
person1.Age = 31; // Mutable state change

// Immutable example
ImmutablePerson person2 = new ImmutablePerson("Bob", 25);
// person2.Age = 26; // Compiler error - immutable type
ImmutablePerson newPerson = new ImmutablePerson(person2.Name, 26); // Creating a new instance
with the desired change
```

In practice, the choice between mutable and immutable types depends on the specific requirements and constraints of your application and the use case at hand. Immutable types are often preferred in scenarios where predictability and avoiding unintended side effects are crucial.

Quiz

What is the definition of mutable types?

Types whose values can be modified after the object is created.

Types whose values cannot be changed after the object is created.

Types that are only used in concurrent programming.

Types that are only used in functional programming.

Which of the following is an example of a mutable type?

Strings

Arrays

What are the characteristics of mutable types?

Modifications can be made in place and they are safer for multithreaded environments.

Modifications can be made in place and they are flexible for scenarios where dynamic changes to data are required.

Modifications cannot be made in place and they eliminate the need for locks in multithreaded environments.

Modifications cannot be made in place and they promote predictability and ease of reasoning about code.

What is the definition of immutable types?

Types whose values can be modified after the object is created.

Types whose values cannot be changed after the object is created.

Types that are only used in concurrent programming.

Types that are only used in functional programming.

Which of the following is an example of an immutable type?

Classes

Arrays

Strings

What are the characteristics of immutable types?

Values remain constant once the object is created and they are safer for multithreaded environments.

Values remain constant once the object is created and they promote predictability and ease of reasoning about code.

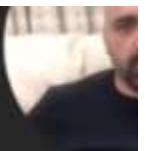
Values can be modified in place and they eliminate the need for locks in multithreaded environments.

Values can be modified in place and they are flexible for scenarios where dynamic changes to data are required.

What are Generics?

ال generic هو نفسه ال template في ال C++ انك تقدر تقدر تخلي اللي هيستخدم الكلاس بتاعك هو اللي يحدد ال data type

What are Generics?



- Generics in C# provide a powerful way to create reusable, type-safe components.
- They allow you to define classes, interfaces, methods, and other constructs with placeholders for the types they will work with, enabling you to write code that can work with any data type.

Generics in C# provide a powerful way to create reusable, type-safe components.

They allow you to define classes, interfaces, methods, and other constructs with placeholders for the types they will work with, enabling you to write code that can work with any data type.

Generics provide a flexible and efficient way to create reusable code that works with a variety of data types while maintaining type safety. They are widely used in the .NET Framework and in C# programming.

Generic Methods

عشان تخلی ال method قبل متغير من النوع generic بتيجي قبل اسما ال method و بتكتب T وبعد الاسم بتكتب <T> وبعدين بتستخدم ال T ك data type و بتتعامل

```
using System;

public class Utility
{
    public static T Swap<T>(ref T first, ref T second)
    {
        T temp = first;
        first = second;
        second = temp;
        return temp;
    }
}

class NullableExample
{
    static void Main()
    {
        int a = 5, b = 10;
        Console.WriteLine($"Before swap: a = {a}, b = {b}");
        Utility.Swap(ref a, ref b);
        Console.WriteLine($"After swap: a = {a}, b = {b}");
        Console.WriteLine();

        string x = "Hello", y = "World";
        Console.WriteLine($"Before swap: x = {x}, y = {y}");
        Utility.Swap(ref x, ref y);
        Console.WriteLine($"After swap: x = {x}, y = {y}");

        Console.ReadKey();
    }
}
```

Here's a complete C# program that includes the Utility class with the generic Swap method and demonstrates its usage:

```
using System;

public class Utility
{
    public static T Swap<T>(ref T first, ref T second)
    {
        T temp = first;
```

```
        first = second;
        second = temp;
        return temp;
    }
}

class Program
{
    static void Main()
    {
        // Usage with integers
        int a = 5, b = 10;
        Console.WriteLine($"Before swap: a = {a}, b = {b}");
        Utility.Swap(ref a, ref b);
        Console.WriteLine($"After swap: a = {a}, b = {b}");
        Console.WriteLine();

        // Usage with strings
        string x = "Hello", y = "World";
        Console.WriteLine($"Before swap: x = {x}, y = {y}");
        Utility.Swap(ref x, ref y);
        Console.WriteLine($"After swap: x = {x}, y = {y}");
    }
}
```

In the `Utility` class, the `Swap` method is declared as a static method that takes two parameters (`first` and `second`) by reference using the `ref` keyword. The method uses a temporary variable (`temp`) to perform the swap operation.

In the usage section, you demonstrate how this generic `Swap` method can be used with both integers (`int`) and strings (`string`). The method is called with the `ref` parameters, and after the call, the values of the variables are swapped.

This is a great illustration of the flexibility and type safety provided by generics in C#. The same `Swap` method can be used for different types without the need for code duplication.

In this program:

1. The `Utility` class is defined with the generic `Swap` method.
2. The `Program` class contains the `Main` method, which demonstrates the usage of the `Swap` method with both integers and strings.
3. The program prints the values before and after the swap to illustrate the effect of the `Swap` method.

When you run this program, you should see output similar to the following:

Before swap: a = 5, b = 10

After swap: a = 10, b = 5

Before swap: x = Hello, y = World

After swap: x = World, y = Hello

This confirms that the `Swap` method is working as expected, swapping the values of the variables for both integers and strings.

Generic Classes

عسان تعمل `generic class` بتحط `<T>` بعد اسمه

```
using System;

public class GenericBox<T> {
    private T content;

    public GenericBox(T value) {
        this.content = value;
    }

    public T GetContent() {
```

```

        return content;
    }

}

class NullableExample
{

    static void Main()
    {
        GenericBox<int> intBox = new GenericBox<int>(42);

        Console.WriteLine("Content of intBox: " + intBox.GetContent());

        GenericBox<string> stringBox = new GenericBox<string>("hello");
        Console.WriteLine("Content of stringBox: " + stringBox.GetContent());

        Console.ReadKey();
    }

}

```

A generic class is a class that has one or more type parameters. Type parameters are placeholders for the actual types that the class will work with.

Here's a complete C# program that includes the `GenericBox<T>` class and demonstrates its usage:

```

using System;

public class GenericBox<T>
{
    private T content;

    public GenericBox(T value)
    {
        content = value;
    }

    public T GetContent()
    {
        return content;
    }
}

```

```
class Program
{
    static void Main()
    {
        // Usage:
        GenericBox<int> intBox = new GenericBox<int>(42);
        Console.WriteLine("Content of intBox: " + intBox.GetContent()); // Outputs: 42

        GenericBox<string> stringBox = new GenericBox<string>("Hello, World!");
        Console.WriteLine("Content of stringBox: " + stringBox.GetContent()); // Outputs:
Hello, World!
    }
}
```

In this complete program:

1. The `GenericBox<T>` class is defined with a generic parameter `T`.
2. The `Program` class contains the `Main` method, which demonstrates the usage of the `GenericBox<T>` class with both `int` and `string` types.
3. Instances of `GenericBox<int>` and `GenericBox<string>` are created, and the content is printed using the `GetContent` method.

When you run this program, you should see output similar to the following:

```
Content of intBox: 42
Content of stringBox: Hello, World!
```

This demonstrates how the generic class `GenericBox<T>` can be instantiated with different types, providing flexibility and reusability.

Quiz

What is the purpose of generics in C#?

To create reusable, type-safe components

To work only with integers

Which of the following constructs can be defined with generics?

Classes

Interfaces

Methods

All of the above

What is the benefit of using generics in the Swap method?

It allows the method to work with any data type

It improves the performance of the method

It reduces the number of lines of code

It restricts the method to work only with integers

What does the GenericBox class demonstrate?

The usage of generic parameters

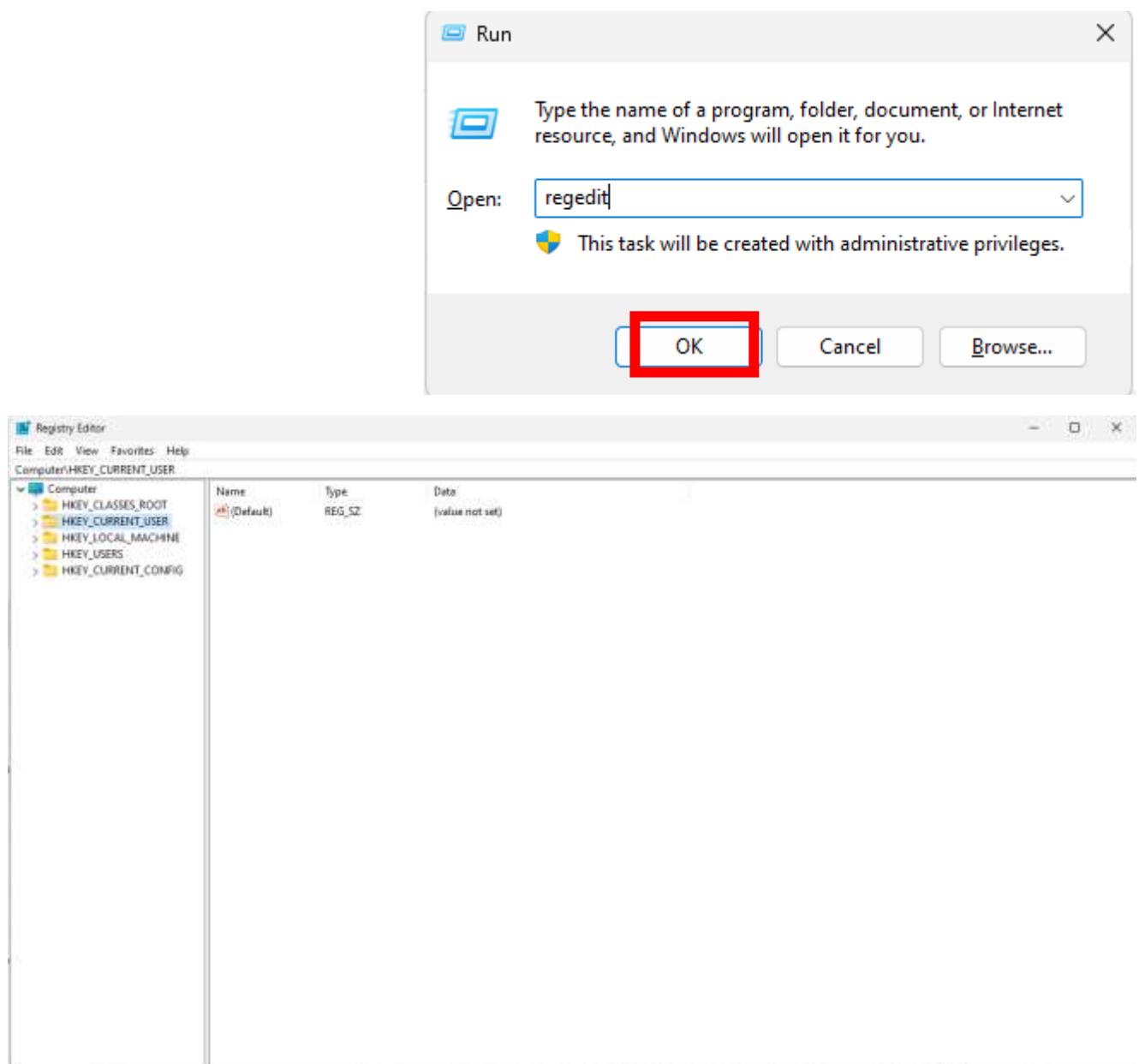
The usage of static methods

The usage of ref parameters

What is Windows Registry and Why?

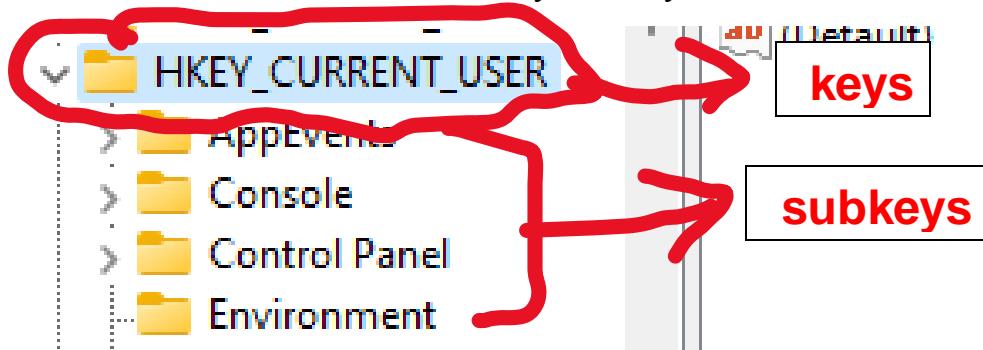
قبل مانبدا عايز اقولك ان الدواء فيه سم قاتل

ما تلعيش كتير في اللي جاي ده عشان ماتبواطنش برامج عندك
لما تدوس على زرار الويندوز مع حرف ال r هتلطلك شاشة ال run اكتب regedit هتلطلك شاشة
ال registry



طيب ايه هوa ال registry ؟

قالك هوa عبارة عن داتابيز فيها keys و sub keys



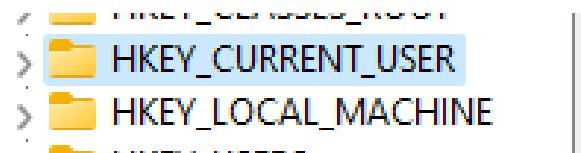
ولما بتتص في الجنب بتلاقي حاجات ملونه دي اسمها values عباره عن اسم ال value والنوع والقيمه

Name	Type	Data
ab (Default)	REG_SZ	(value not set)
ab Path	REG_SZ	C:\Program Files\7-Zip\
ab Path64	REG_SZ	C:\Program Files\7-Zip\

فال مكان بتقدر التطبيقات انها تخزن فيه معلومات خاصه بيه زي مثلا ال product key

تقدر كمان تجي في التطبيق بناء الكورس اللي فات وتحفظ اسم المستخدم وكلمة المرور هنا بدل ال text file

دول اهم keys



ال local machine القيم اللي فيه هيا مشتركه بين كل المستخدمين بتوع الجهاز

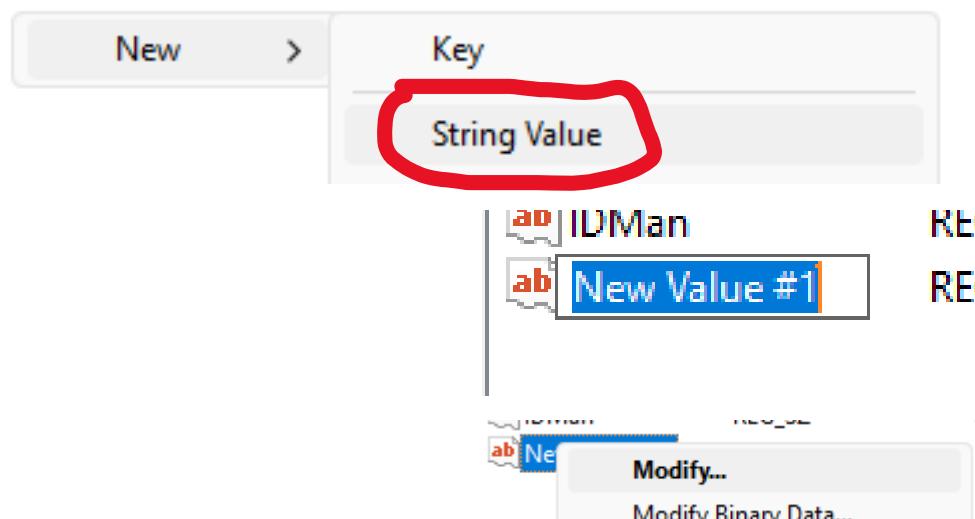
ال current user بيتحط فيه القيم الخاصه بالمستخدم الحالي فقط يعني كل مستخدم ليه

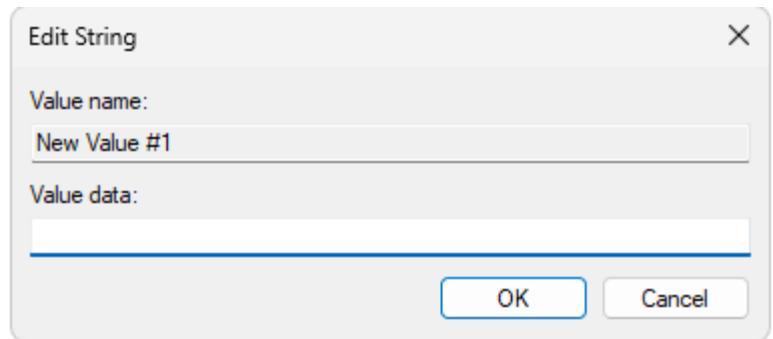
current user

لو روحت فتحت المسار ده

Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\Current Version\Run

تقدر تحط مسار لبرنامج معين وتشغله اول ما تشغلي الويندوز





وتحط المسار بناء البرنامج وتحفظه
نقدر تعمل أي حاجة في البرنامج بتاعك عن طريق ال registry بس عشان تضيف key في ال permission تحتاج local machine

key aspects of the Windows Registry:

1. Hierarchy: The Registry is organized into keys, subkeys, and values, forming a hierarchical structure. Keys are analogous to folders, subkeys are subfolders, and values are the actual data or configuration settings.
2. Centralized Configuration: The Registry consolidates system and application settings, making it a centralized location for configuration information. This centralized approach helps ensure consistency and allows for easy retrieval and modification of settings.
3. System and User Settings: The Registry stores both system-wide settings (applicable to all users) and user-specific settings. User-specific settings are stored in "HKEY_CURRENT_USER," while system-wide settings are stored in various other root keys like "HKEY_LOCAL_MACHINE."

What is Windows Registry?

- The Windows Registry is a hierarchical database that stores configuration settings and options on Microsoft Windows operating systems.
- It's used to store information about the system, applications, and user preferences.
- In C#, you can interact with the Windows Registry using the Microsoft.Win32 namespace.

key aspects of the Windows Registry:

4. Start-up Configuration: The Registry is used to store information about programs and services that start automatically when the system boots. This includes settings related to startup programs and services.

"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run"

5. Hardware Configuration: Information about installed hardware components, device drivers, and their configurations is stored in the Registry. This includes details about connected devices, hardware settings, and driver configurations.

Be Careful:

- **Modifying the Windows Registry can have significant consequences, so it's crucial to be careful and ensure that you have the necessary permissions. Always back up the Registry before making any changes.**

The Windows Registry is a hierarchical database that stores configuration settings and options on Microsoft Windows operating systems. It's used to store information about the system, applications, and user preferences. In C#, you can interact with the Windows Registry using the `Microsoft.Win32` namespace.

Here are some key aspects of the Windows Registry:

1. **Hierarchy:** The Registry is organized into keys, subkeys, and values, forming a hierarchical structure. Keys are analogous to folders, subkeys are subfolders, and values are the actual data or configuration settings.
2. **Centralized Configuration:** The Registry consolidates system and application settings, making it a centralized location for configuration information. This centralized approach helps ensure consistency and allows for easy retrieval and modification of settings.
3. **System and User Settings:** The Registry stores both system-wide settings (applicable to all users) and user-specific settings. User-specific settings are stored

in "HKEY_CURRENT_USER," while system-wide settings are stored in various other root keys like "HKEY_LOCAL_MACHINE."

4. Start-up Configuration: The Registry is used to store information about programs and services that start automatically when the system boots. This includes settings related to startup programs and services.

"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run"

1. Hardware Configuration: Information about installed hardware components, device drivers, and their configurations is stored in the Registry. This includes details about connected devices, hardware settings, and driver configurations.

Note: Modifying the Windows Registry can have significant consequences, so it's crucial to be careful and ensure that you have the necessary permissions. Always back up the Registry before making any changes.

Writing to Registry

عشان تكتب في ال registry بتسخدم المكتبه دي

```
using Microsoft.Win32;
```

وبعدين بتسدعي method اسمها set value وتدلها المسار والاسم وال value وال datatype والقيمه نفسها

```
using System;
using Microsoft.Win32;

class NullableExample
{
    static void Main()
    {
        string KeyPath = @"HKEY_LOCAL_MACHINE\SOFTWARE\YourSoftware";
        string valueName = "YourValueName";
        string valueData = "YourvalueData";

        try {
            Registry.SetValue(KeyPath, valueName, valueData, RegistryValueKind.String);
            Console.WriteLine($"Value {valueName} successfully written to the registry.");
        } catch(Exception e) {
            Console.WriteLine($"An error occurred:{e.Message}");
        }
        Console.ReadKey();
    }
}
```

Writing to the Registry:

To write values to the Registry, you can use the `Registry` class as well. Here's an example of how to write a string value to the Registry:

```
using Microsoft.Win32;
using System;

class Program
{
    static void Main()
    {
        // Specify the Registry key and path
        string keyPath = @"HKEY_LOCAL_MACHINE\SOFTWARE\YourSoftware";
        string valueName = "YourValueName";
        string valueData = "YourValueData";

        try
        {
            // Write the value to the Registry
            Registry.SetValue(keyPath, valueName, valueData, RegistryValueKind.String);

            Console.WriteLine($"Value {valueName} successfully written to the Registry.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
}
```

Remember to handle exceptions appropriately, as working with the Registry can result in exceptions if there are permission issues or if the specified key doesn't exist.

Note: Modifying the Windows Registry can have significant consequences, so it's crucial to be careful and ensure that you have the necessary permissions. Always back up the Registry before making any changes.

Reading from Registry

عشان تقرأ قيمه بتسخدم GET VALUE ولو القيمه مش موجوده بيحط القيمه NULL

```
using System;
using Microsoft.Win32;

class NullableExample
{
    static void Main()
    {
        string keyPath = @"HKEY_LOCAL_MACHINE\SOFTWARE\YourSoftware";
        string valueName = "YourValueName";

        try
        {
            string value = Registry.GetValue(keyPath, valueName, null) as string;

            if (value != null)
            {
                Console.WriteLine($"The value of {valueName} is: {value}");
            }
            else
            {
                Console.WriteLine($"Value {valueName} not found in the Registry.");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
        Console.ReadKey();
    }
}
```

Reading from the Registry:

To read values from the Registry, you can use the `Registry` class. Here's an example of how to read a string value from the Registry:

```
using Microsoft.Win32;
using System;

class Program
```

```
{  
    static void Main()  
{  
        // Specify the Registry key and path  
        string keyPath = @"HKEY_LOCAL_MACHINE\SOFTWARE\YourSoftware";  
        string valueName = "YourValueName";  
  
        try  
        {  
            // Read the value from the Registry  
            string value = Registry.GetValue(keyPath, valueName, null) as string;  
  
            if (value != null)  
            {  
                Console.WriteLine($"The value of {valueName} is: {value}");  
            }  
            else  
            {  
                Console.WriteLine($"Value {valueName} not found in the Registry.");  
            }  
        }  
        catch (Exception ex)  
        {  
            Console.WriteLine($"An error occurred: {ex.Message}");  
        }  
    }  
}
```

Remember to handle exceptions appropriately, as working with the Registry can result in exceptions if there are permission issues or if the specified key doesn't exist.

Note: Modifying the Windows Registry can have significant consequences, so it's crucial to be careful and ensure that you have the necessary permissions. Always back up the Registry before making any changes.

Permission to Write to Registry

عشان تاخد ال permission عشان تكتب على ال registry وخصوصا ال local machine عندك طریقین

اول واحدة و هیا انک تشغّل البرنامج run as administrator (في حالتنا هنا هنشغّل ال visual studio على ال run as administrator)

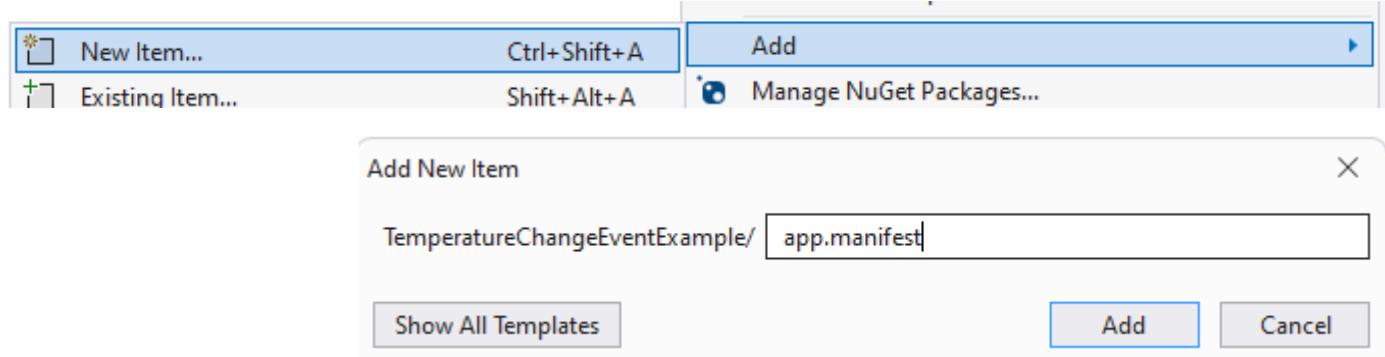
قبل مانشوف الطريقة الثانية خلينا نعرف حاجه و هیا اني لما كتبت ال registry استخدمت مكتبة اسمها win32 بس انا النظام عندي win 64

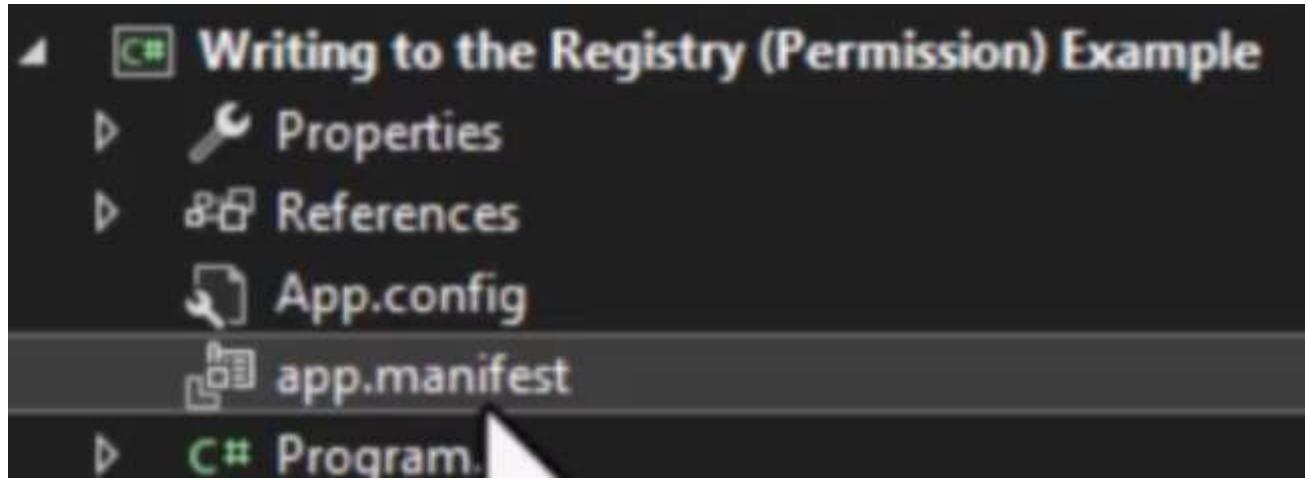
فهتلaci ال key بتاعك موجود في ملف wow6432Node وال WOW ده معناه windows on windows

ال win 64 بيقدر يشغل البرامج اللي مبنيه على ويندوز ٣٢ فيحيط ملفات ال registry بتاعتتها في الملف ده وده المسار بتاعه

HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node

الطريقة الثانية هيا انک تخلي البرنامج بتاعك يطلب من الويندوز الصلاحيات عن طريق انک تضييف ملف manifest وده ملف بيتحط فيه meta data خاصه بالبرنامج

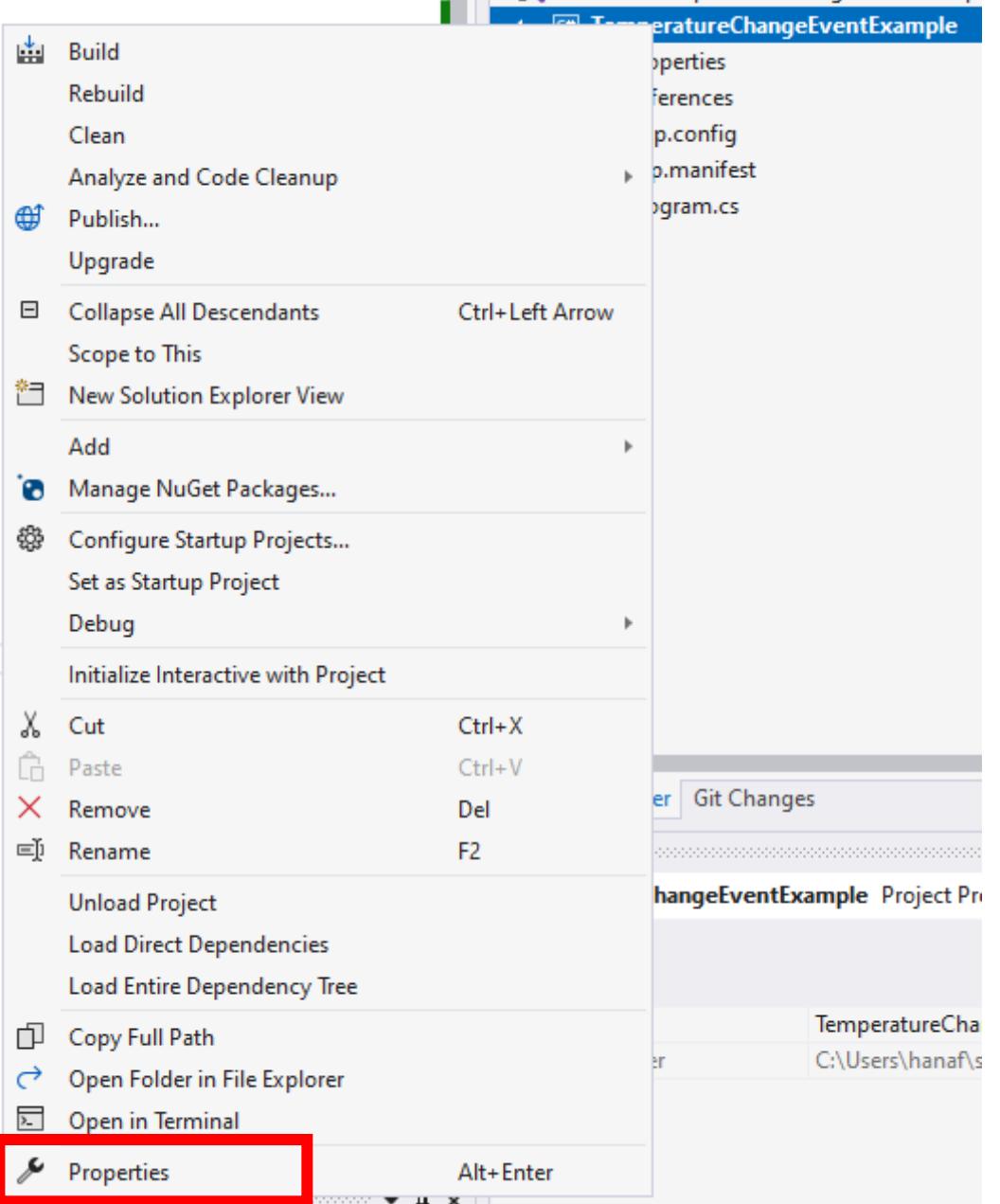


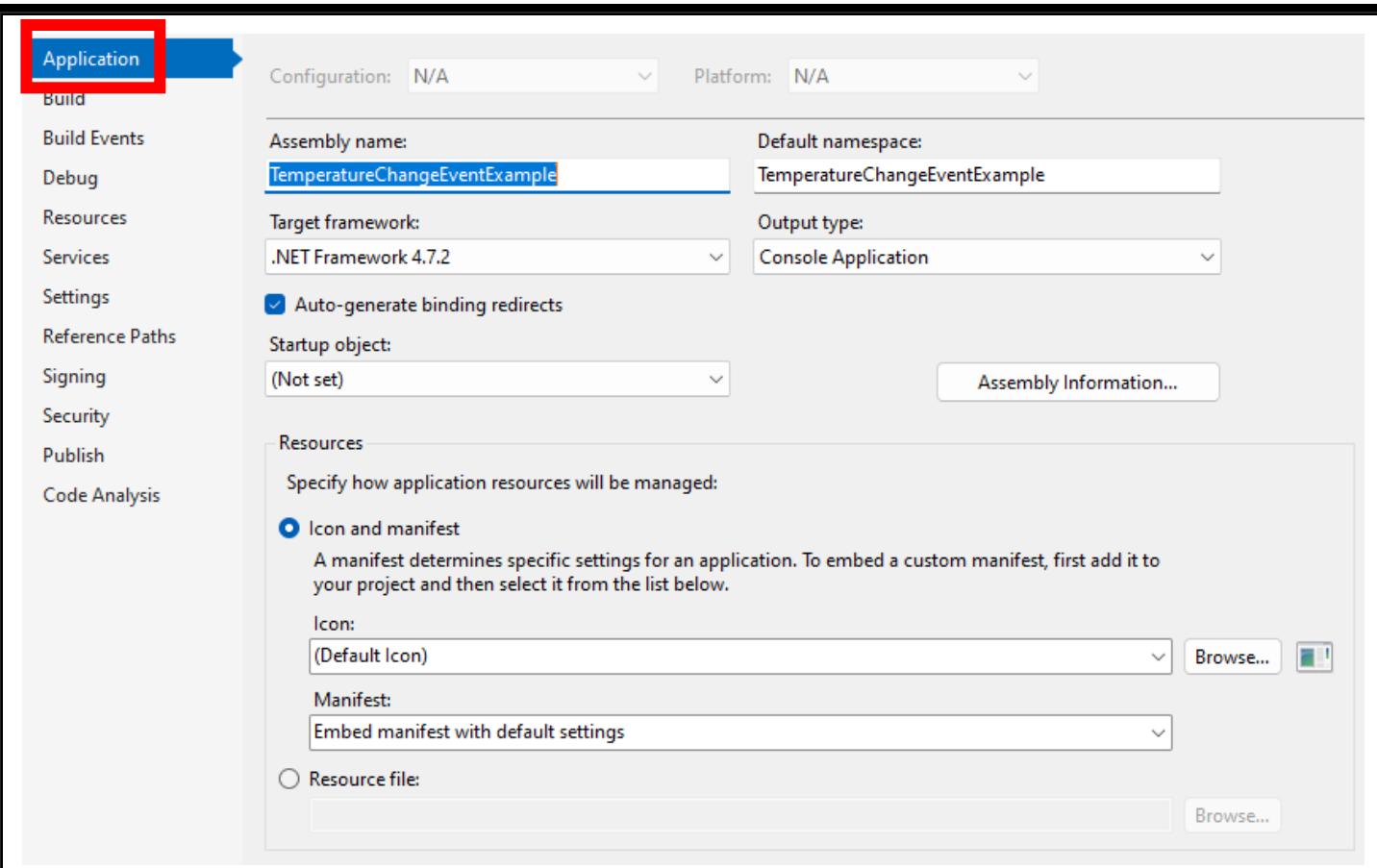


الملف ده عباره عن ملف xml بتكتب فيه الكود ده

```
<?xml version="1.0" encoding="utf-8"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1">
    <assemblyIdentity version="1.0.0.0" processorArchitecture="X86" name="YourAppName"
type="win32" />
    <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
        <security>
            <requestedPrivileges>
                <requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
            </requestedPrivileges>
        </security>
    </trustInfo>
</assembly>
```

بعدين بتروح عال properties





Manifest: app.manifest

بعدين هتعمل run هطلب منك restart وبعدين هتشتغل البرنامج عادي وهتقدر تستخدم أي ملف من registry ال

Why writing to registry under software on win64 writes to

HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node ?

When a 32-bit application runs on a 64-bit version of Windows, the operating system uses a feature called Windows-on-Windows 64-bit (WOW64) to provide compatibility. This allows 32-bit applications to run on a 64-bit system without modification.

One aspect of this compatibility is related to the Windows Registry. The registry on a 64-bit Windows system is divided into two parts:

1. HKEY_LOCAL_MACHINE\SOFTWARE: This is where 64-bit applications store their registry entries.

2. HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node: This is a special node created by the WOW64 subsystem to redirect 32-bit applications when they try to access the registry under the HKEY_LOCAL_MACHINE\SOFTWARE path. This redirection ensures that 32-bit applications access the correct part of the registry without causing conflicts with 64-bit applications.

So, when a 32-bit application writes to HKEY_LOCAL_MACHINE\SOFTWARE on a 64-bit system, it is redirected to HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node by the operating system. This redirection helps maintain separation between the registry entries for 32-bit and 64-bit applications and ensures compatibility in a mixed environment.

Why it is not using the same concept when writing on HKEY_CURRENT_USER?

The redirection concept for the registry is specific to the HKEY_LOCAL_MACHINE hive on 64-bit Windows systems. This redirection is implemented to avoid conflicts between 32-bit and 64-bit applications by directing 32-bit applications to a separate registry path (HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node).

However, when it comes to user-specific settings in the registry (e.g., settings stored under HKEY_CURRENT_USER), the redirection is not necessary. This is because each user has their own registry hive under HKEY_USERS, and 32-bit and 64-bit applications do not share the same registry space in the user-specific context.

For example:

- 32-bit applications on a 64-bit system would write to HKEY_CURRENT_USER\Software without redirection.
- 64-bit applications on the same system would also write to HKEY_CURRENT_USER\Software without redirection.

The user-specific registry settings are isolated to each user's profile, and there is no need for the redirection mechanism in this context. The redirection is mainly implemented at the machine level (HKEY_LOCAL_MACHINE) to avoid conflicts in a shared space between 32-bit and 64-bit applications.

Modifying the Windows Registry requires administrative privileges. Here's how you can run your C# application with elevated permissions to write to the local machine registry:

1. Run Visual Studio as Administrator:

- - Right-click on the Visual Studio icon.
 - Select "Run as administrator."
 - This will ensure that any application you run from Visual Studio inherits administrative privileges.

1. Require Administrative Privileges in Your Application:

- - In your C# application, you can include a manifest file that requests administrative privileges.
 - Create a file named `app.manifest` in your project (if it doesn't already exist).
- Add the following contents to the manifest file:

```
<?xml version="1.0" encoding="utf-8"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1">
    <assemblyIdentity version="1.0.0.0" processorArchitecture="X86" name="YourAppName"
type="win32" />
    <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
        <security>
            <requestedPrivileges>
                <requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
            </requestedPrivileges>
        </security>
    </trustInfo>
</assembly>
```

- - Replace "YourAppName" with the actual name of your application.

1. Embed Manifest in Your Application:

-

- Open your project properties in Visual Studio.
- Go to the "Application" tab.
- Set "Manifest" to the path of your `app.manifest` file.

Now, when you run your application, it will prompt for administrative privileges. This ensures that your application has the necessary permissions to write to the local machine registry.

Keep in mind that modifying the registry requires caution, as incorrect changes can impact the system's stability. Always make sure you have a backup of the registry before making any changes, especially if your application involves writing to critical areas of the registry.

Homework

Now , go to your Project in Course 19, and save the username and password to registry instead of file.

الحل بتاعي

```
using System;
using System.IO;
using System.Windows.Forms;
using DVLD_Buisness;
using Microsoft.Win32;

namespace DVLD.Classes
{
    internal static class clsGlobal
    {
        public static clsUser CurrentUser;

        public static bool RememberUsernameAndPassword(string Username, string Password)
        {
            try
            {
                string KeyName = @"HKEY_LOCAL_MACHINE\SOFTWARE\DVLD";
                string ValueName = "LastLogIn";
                string ValueData = Registry.GetValue(KeyName, ValueName, null) as string;

                if (Username == "" && !string.IsNullOrEmpty( ValueData ))
                {
                    ValueData = "";
                    Registry.SetValue(KeyName, ValueName, ValueData, RegistryValueKind.String);
                    return true;
                }
                ValueData = Username + "#//#" + Password;
                Registry.SetValue(KeyName, ValueName, ValueData, RegistryValueKind.String);

                return true;
            }
        }
    }
}
```

```

        }

        catch (Exception ex)
        {
            MessageBox.Show($"An error occurred: {ex.Message}");
            return false;
        }
    }

    public static bool GetStoredCredential(ref string Username, ref string Password)
    {
        //this will get the stored username and password and will return true if found and false if not found.
        try
        {
            string KeyName = @"HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\DVLD";
            string ValueName = "LastLogIn";
            string Value = Registry.GetValue(KeyName, ValueName, null) as string;

            if (!string.IsNullOrEmpty(Value))
            {
                string[] result = Value.Split(new string[] { "#//#" }, StringSplitOptions.None);

                Username = result[0];
                Password = result[1];
                return true;
            } else {
                return false;
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show($"An error occurred: {ex.Message}");
            return false;
        }
    }
}

```

Quiz 1

What is the Windows Registry?

A hierarchical database that stores configuration settings and options on Microsoft Windows operating systems

A file system used by Windows to store user data

A cloud-based storage system for Windows applications

A programming language used to interact with Windows

What is the purpose of the Windows Registry?

To store information about the system, applications, and user preferences

To store user data such as files and documents

To provide a centralized location for storing software licenses

To manage network connections and protocols

What is the hierarchy of the Windows Registry?

Keys, subkeys, and values

Files, folders, and data

Folders, subfolders, and files

Data, folders, and keys

Where are user-specific settings stored in the Registry?

"HKEY_USERS"

"HKEY_LOCAL_MACHINE"

"HKEY_CURRENT_USER"

"HKEY_CLASSES_ROOT"

Where is information about installed hardware components, device drivers, and their configurations stored in the Registry?

"HKEY_LOCAL_MACHINE\HARDWARE"

"HKEY_LOCAL_MACHINE\SOFTWARE"

"HKEY_CURRENT_USER\HARDWARE"

"HKEY_CURRENT_USER\SOFTWARE"

What precautions should you take before modifying the Registry?

Always back up the Registry before making any changes

Ensure that you have the necessary permissions to modify the Registry

Handle exceptions appropriately

All of the above

How can you write a value to the Registry in C#?

Using the Registry class and the SetValue method

Using the File class and the WriteAllText method

Using the StreamReader class and the ReadLine method

Using the Console class and the WriteLine method

How can you read a value from the Registry in C#?

Using the Registry class and the GetValue method

Using the File class and the ReadAllText method

Using the StreamWriter class and the WriteLine method

Using the Console class and the ReadLine method

How can you ensure that your C# application has elevated permissions to write to the local machine Registry?

Run Visual Studio as Administrator

Embed a manifest in your application that requests administrative privileges

Include a separate executable with administrative privileges

Use a third-party library for elevated Registry access

Quiz 2

What is the purpose of the WOW64 subsystem in a 64-bit version of Windows?

correct

To provide compatibility for 32-bit applications on a 64-bit system



To enhance the performance of 64-bit applications

To redirect 64-bit applications to a separate registry path

To ensure that user-specific registry settings are separate from machine-level settings

Which part of the Windows Registry is used by 64-bit applications to store their registry entries?

HKEY_LOCAL_MACHINE\SOFTWARE

HKEY_CURRENT_USER\SOFTWARE

HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node

HKEY_CURRENT_USER\SOFTWARE\WOW6432Node

Why is there a redirection when a 32-bit application writes to HKEY_LOCAL_MACHINE\SOFTWARE on a 64-bit system?

To avoid conflicts with 64-bit applications

To improve performance for 32-bit applications

To ensure that all applications access the same registry path

To separate user-specific registry settings from machine-level settings

Does the same redirection concept apply when writing to the HKEY_CURRENT_USER hive?

Yes, 32-bit applications are redirected to a separate registry path

No, redirection is not necessary for user-specific registry settings

Only 64-bit applications are redirected to a separate registry path

Redirection applies, but the path is different compared to HKEY_LOCAL_MACHINE

How can you run a C# application with elevated permissions to write to the local machine registry?

Run Visual Studio as Administrator

Add a manifest file to the project

Set the 'Manifest' property in Visual Studio

All of the above

Why is it important to be cautious when modifying the Windows Registry?

Changes can impact the stability of the system

Modifying the registry requires advanced programming skills

Incorrect changes can lead to data loss

All of the above

Delete value from Registry - Example

عشان نحذف registry هنحتاج اسم ال key والملف اللي قبليه علي طول زي كده

```
string keyPath = @"SOFTWARE\YourSoftware";
```

كمان هنحتاج اسم ال value

```
string valueName = "YourValueName";
```

الطريقه اللي هنسخدمها هتبقي اننا هنعمل اتنين objects واحد لـ base key والثانى لـ key بتاعنا
هنعمل دلوقتي ال base key تمام؟

هنأخذ object من كلاس اسمه registry key وهنستدعي اسمها method parameters دي بتاخذ اتنين

اول واحد المسار الأساسي سواء كان local machine او current user حسب انت حاطط ال بتاعك فين registry

تاني واحد هو انت شغال علي سيسنتم ٣٢ ولا ٦٤ ؟

و هنحط ال using

```
using (RegistryKey baseKey =  
RegistryKey.OpenBaseKey(RegistryHive.CurrentUser,  
RegistryView.Registry64))
```

بعدين هنعمل ال sub key

هناخد base key من object registry key بس المرادي هنسدعي method موجوده في ال key اسمها open sub key ودي بتاخذ المسار المختصر و عشان تبقى في ال write mode وتقدر تعدل عليه

```
using (RegistryKey key = baseKey.OpenSubKey(keyPath, true))
```

بعدين هتقوله لو ال key موجود احذفه

```
if (key != null)  
{  
    // Delete the specified value  
    key.DeleteValue(valueName);
```

بس كده

اهم حاجه رکز في ال parameters بتاعت ال base key عشان مانقعدش تلف حوالين نفسك عالفااضي

```
using Microsoft.Win32;  
using System;  
  
class Program  
{  
    static void Main()  
    {  
        // Specify the registry key path and value name  
        string keyPath = @"SOFTWARE\YourSoftware";  
        string valueName = "YourValueName";  
  
        try  
        {  
            // Open the registry key in read/write mode with explicit registry view  
            using (RegistryKey baseKey = RegistryKey.OpenBaseKey(RegistryHive.CurrentUser, RegistryView.Registry64))  
            {
```

```

using (RegistryKey key = baseKey.OpenSubKey(keyPath, true))
{
    if (key != null)
    {
        // Delete the specified value
        key.DeleteValue(valueName);

        Console.WriteLine($"Successfully deleted value '{valueName}' from registry key '{keyPath}'");
    }
    else
    {
        Console.WriteLine($"Registry key '{keyPath}' not found");
    }
}
catch (UnauthorizedAccessException)
{
    Console.WriteLine("UnauthorizedAccessException: Run the program with administrative privileges.");
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred: {ex.Message}");
}

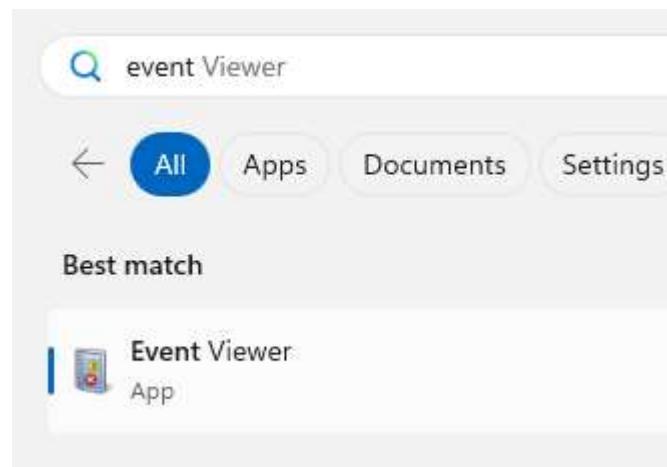
Console.ReadKey();
}
}

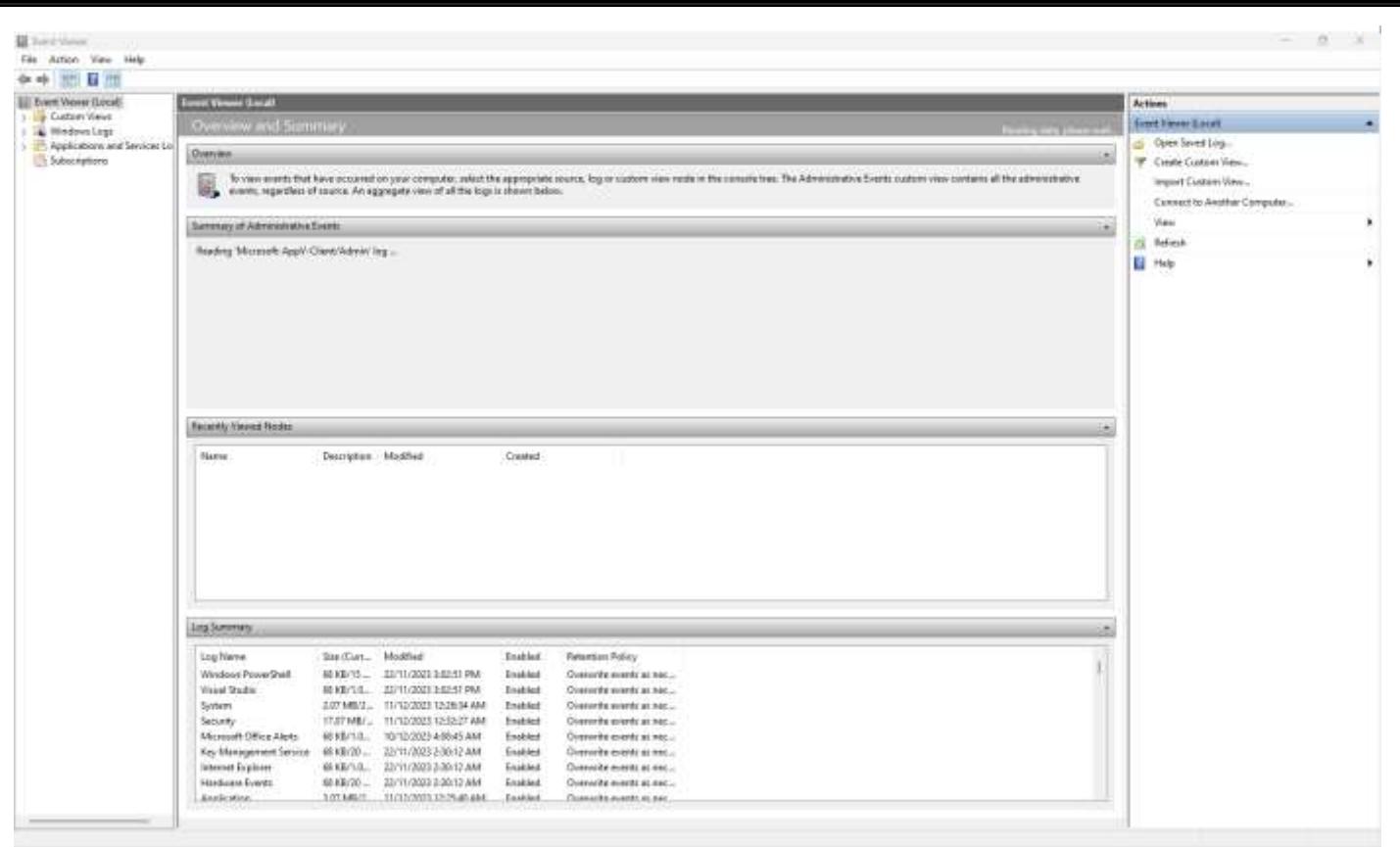
```

What is Event Log?

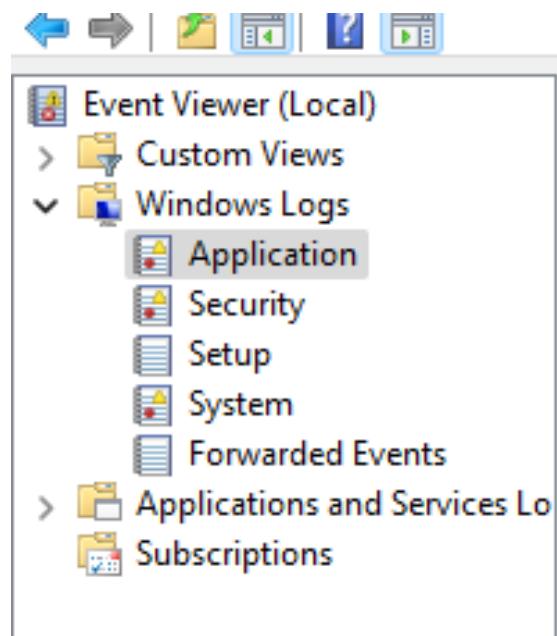
ال event log هي داتا بيز موجوده في الويندوز بيكون فيه كل رسائل ال log اللي بيكون فيها معلومات او أخطاء

بتفتحها عن طريق زرار الويندوز وتبحث عن event viewer





عندك مثل المثل logs بتاعت الويندوز بيكون فيها كل ال logs بتاعت التطبيقات



Level	Date and Time	Source	Event ID	Task Category
Information	11/12/2023 12:25:10 AM	VSS	8224	None
Information	11/12/2023 12:24:26 AM	SecurityCenter	15	None
Information	11/12/2023 12:23:17 AM	RestartManager	10001	None
Information	11/12/2023 12:23:12 AM	SecurityCenter	15	None
Information	11/12/2023 12:23:11 AM	ESENT	326	General
Information	11/12/2023 12:23:11 AM	ESENT	105	General

لما بتدوس على كل log بتطلع بيانته تحت

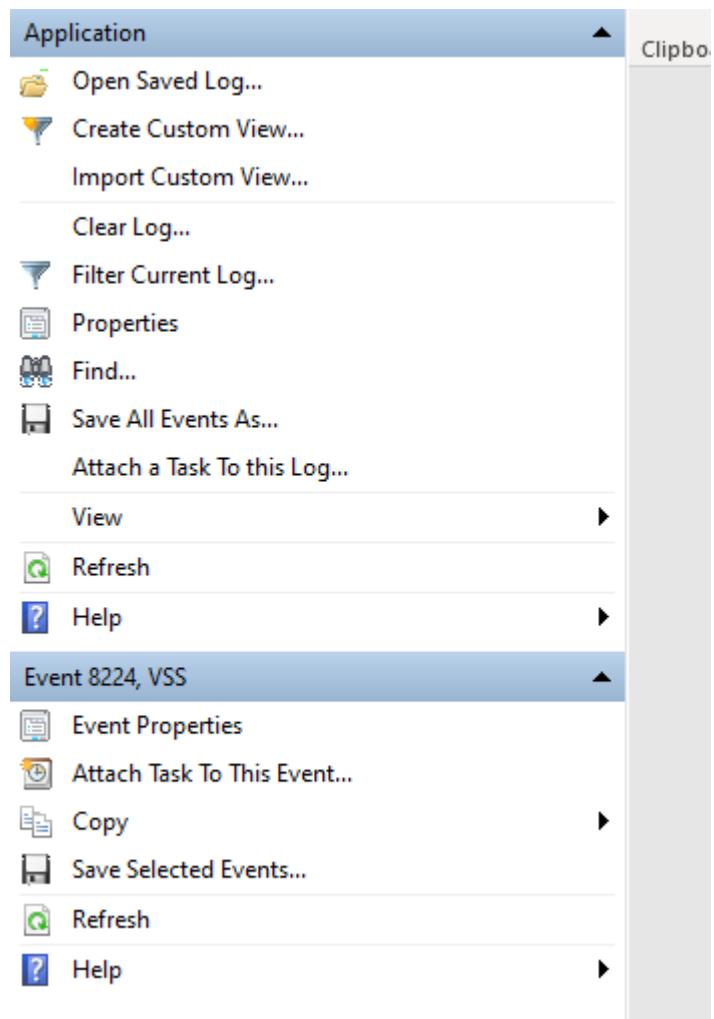
Event 8224, VSS

General Details

The VSS service is shutting down due to idle timeout.

Log Name:	Application
Source:	VSS
Event ID:	8224
Level:	Information
User:	N/A
OpCode:	Info
More Information:	Event Log Online Help

تقدر تعمل tracing بكل خطوة بيقوم فيها البرنامج بحيث لما العميل يقولك فيه مشكله في البرنامج تقوله يدخل عال logs ويفلتر علي التطبيق بتاعك



What is Event Log?

- In computing, an event log is a file or database used to store events that occur in a system.
- These events can include information about errors, warnings, system events, user activities, and more.
- Event logs are commonly used for troubleshooting, monitoring, and auditing purposes.



In computing, an event log is a file or database used to store events that occur in a system. These events can include information about errors, warnings, system events, user activities, and more. Event logs are commonly used for troubleshooting, monitoring, and auditing purposes.

To run the Windows Event Viewer, you can follow these steps:

1. Open the Start Menu:

- - In Windows 10, you can click on the Windows icon in the bottom left corner of your screen.

1. Search for "Event Viewer":

- - Type "Event Viewer" into the search bar.

1. Open Event Viewer:

- - Click on the "Event Viewer" app that appears in the search results.

1. Navigate to Event Logs:

- - In the Event Viewer window, you'll see a left-hand navigation pane. Click on "Windows Logs" to expand it.

1. Choose a Log:

- - Under "Windows Logs," you'll find several logs like "Application," "Security," "Setup," "System," and more. Choose the log you're interested in viewing. For the example provided earlier, you would likely find the log entry in the "Application" log.

1. View Log Entries:

- - Click on the log you selected, and you'll see a list of log entries in the center pane. Look for the entry created by your C# program.

The logs in the Event Viewer contain information about various events on your system, including system errors, warnings, information events, and more. You can filter, sort, and search for specific events within the Event Viewer interface to help you locate the information you need.

After running this program, you can check the Windows Event Viewer to see the log entry. Open the Event Viewer, navigate to "Windows Logs," and select "Application" to view the log entries.

Logging Example

بيقولك قبل ماتعمل **log** لازم تحدد انت عايز تعمله فين
فالحنا هنعمله في ال **application**



طيب عشان نعمل **log** جديد لازم نستدعي المكتبه دي

```
using System.Diagnostics;
```

تاني حاجه بنحدد ال **source** اللي هو اسم البرنامج

```
string Source = "KokoApp";
```

بعدين بنقول لو ال **source** ده مش موجود

```
if (!EventLog.SourceExists(Source)) {
```

اعمل واحد جديد واديله ال source وال category اللي هيتعمل فيه

```
if (!EventLog.SourceExists(Source)) {
```

```
    EventLog.CreateEventSource(Source, "Application");  
    Console.WriteLine("Event Source created");
```

```
}
```

احنا كده لأننا عملنا ملف هنحط جواه كل المعلومات بتاعتتنا وكل اللي هنعمله بعد كده عشان نضيف معلومه هو انتا تستدعي اسمها source وبتاخذ اسم ال write entry method والمعلومه نوع ال log

كأنك بتعمل message box

```
EventLog.WriteEntry(Source, "this is information", EventLogEntryType.Information);  
EventLog.WriteEntry(Source, "this is warning", EventLogEntryType.Warning);  
EventLog.WriteEntry(Source, "this is Error", EventLogEntryType.Error);
```

بس هحتاج صلاحيات ودي شرحناها قبل كده

	Error	11/12/2023 12:59:39 AM	KokoApp
	Warning	11/12/2023 12:59:39 AM	KokoApp
	Information	11/12/2023 12:59:39 AM	KokoApp
	Information	11/12/2023 12:59:39 AM	KokoApp
	

In computing, an event log is a file or database used to store events that occur in a system. These events can include information about errors, warnings, system events, user activities, and more. Event logs are commonly used for troubleshooting, monitoring, and auditing purposes.

In C#, you can work with the event log using the `EventLog` class, which is part of the `System.Diagnostics` namespace. Here's a simple example of how to write an entry to the Windows event log in C#:

```
using System;  
using System.Diagnostics;  
  
  
class Program  
{  
    static void Main()  
    {  
        // Specify the source name for the event log  
        string sourceName = "MyApp";
```

```

// Create the event source if it does not exist
if (!EventLog.SourceExists(sourceName))
{
    EventLog.CreateEventSource(sourceName, "Application");
    Console.WriteLine("Event source created.");
}

// Log an information event
EventLog.WriteEntry(sourceName, "This is an information event.",
EventLogEntryType.Information);

Console.WriteLine("Event written to the log.");
}
}

```

In this example:

1. We define a `sourceName` variable, which represents the source of the event log entries. This is typically the name of your application.
2. We check if the event source exists, and if not, we create it using `EventLog.CreateEventSource`.
3. We use `EventLog.WriteEntry` to write an entry to the event log. In this case, it's an information event, but you can choose from different entry types such as `Information`, `Warning`, or `Error`.

Make sure to run this program with sufficient permissions, as creating an event source may require administrative privileges.

After running this program, you can check the Windows Event Viewer to see the log entry. Open the Event Viewer, navigate to "Windows Logs," and select "Application" to view the log entries. You should see the entry created by your C# program.

Quiz

What is the purpose of an event log in computing?

To store events that occur in a system

To monitor user activities

To troubleshoot system errors

To create backups of important data

How can you open the Windows Event Viewer?

Click on the Windows icon in the bottom left corner of the screen

Type "Event Viewer" into the search bar

Click on the "Event Viewer" app in the search results

Click on the Event Viewer shortcut on the desktop

Where can you find the log entries in the Event Viewer?

Under "Windows Logs"

Under "Applications"

Under "Security"

Under "System"

What types of events can be found in the Event Viewer?

System errors

User activities

Security breaches

Backup tasks

Which namespace is used to work with the event log in C#?

System.Diagnostics

System.IO

System.Windows.Forms

System.Threading

Homework

Go to the DVLD project in course 19 and log all exceptions in the event log :-)

Wherever there is try catch, use event log to log the exception.

```
using System.Diagnostics;

namespace DVLD_DataAccess
{
    public class clsDataAccessLogs
    {
        public static void CreateExceptionLog(string Message)
        {
            string SourceName = "DVLD";

            if (!EventLog.SourceExists(SourceName))
            {
                EventLog.WriteEntry(SourceName, Message, EventLogEntryType.Error);
            }
        }
    }
}
```

What is App.Config?

في الكورس اللي فات كنا بنخزن ال connection string جوه البرنامج نفسه
بس لو جيت تحط الكود عند العميل هحتاج تعده وخصوصا لو عندك اكتر من عميل او مثلا هنتحا
تغير السيرفر

فالكل انه فيه ملف اسمه app.config ولما بتعمل build بيشيل كلمة app ويحط مكانها اسم المشروع
بتاعاك الملف ده ده بتخزن الحاجات اللي زي كده وده بيكون ملف xml وده بيكون عباره عن أجزاء كل
جزء بيمثل settings معينه جوه البرنامج وفيه مكتبه بتقدر تخليك توصله وتعديل فيه

What is App.config?

- In C#, App.config refers to the configuration file associated with a .NET application.
- The configuration file is an XML file that contains settings and configuration information for an application.
- The file is named App.config during development, and when the application is compiled, the settings are often moved to the application's executable file with a .config extension (e.g., YourAppName.exe.config).

Some key points about App.config

- Settings: It is commonly used to store settings such as connection strings, app settings, and other configuration parameters that can be changed without modifying the code.
- XML Format: The App.config file is an XML file. It consists of elements and attributes that define various settings for the application.
- Configuration Sections: It allows you to define custom configuration sections for your application. These sections can be used to organize and group related settings.
- ConfigurationManager Class: In C#, you can access the settings defined in App.config using the ConfigurationManager class, which is part of the System.Configuration namespace. This class provides methods and properties to read configuration settings.

In C#, App.config refers to the configuration file associated with a .NET application. The configuration file is an XML file that contains settings and configuration information for an

application. The file is named `App.config` during development, and when the application is compiled, the settings are often moved to the application's executable file with a `.config` extension (e.g., `YourAppName.exe.config`).

Here are some key points about `App.config`:

1. Settings: It is commonly used to store settings such as connection strings, app settings, and other configuration parameters that can be changed without modifying the code.
2. XML Format: The `App.config` file is an XML file. It consists of elements and attributes that define various settings for the application.
3. Configuration Sections: It allows you to define custom configuration sections for your application. These sections can be used to organize and group related settings.
4. ConfigurationManager Class: In C#, you can access the settings defined in `App.config` using the `ConfigurationManager` class, which is part of the `System.Configuration` namespace. This class provides methods and properties to read configuration settings.

If you are using the .NET Framework and still facing issues with `ConfigurationManager`, there are a few more steps you can take to troubleshoot the problem:

- Check System.Configuration Reference:
 - Right-click on your project in Visual Studio.
 - Choose "Add" -> "Reference..."
 - In the Reference Manager, make sure that `System.Configuration` is checked.
- Verify Namespace and Using Directive:
- Ensure that your code file has the following using directive:

```
using System.Configuration;
```

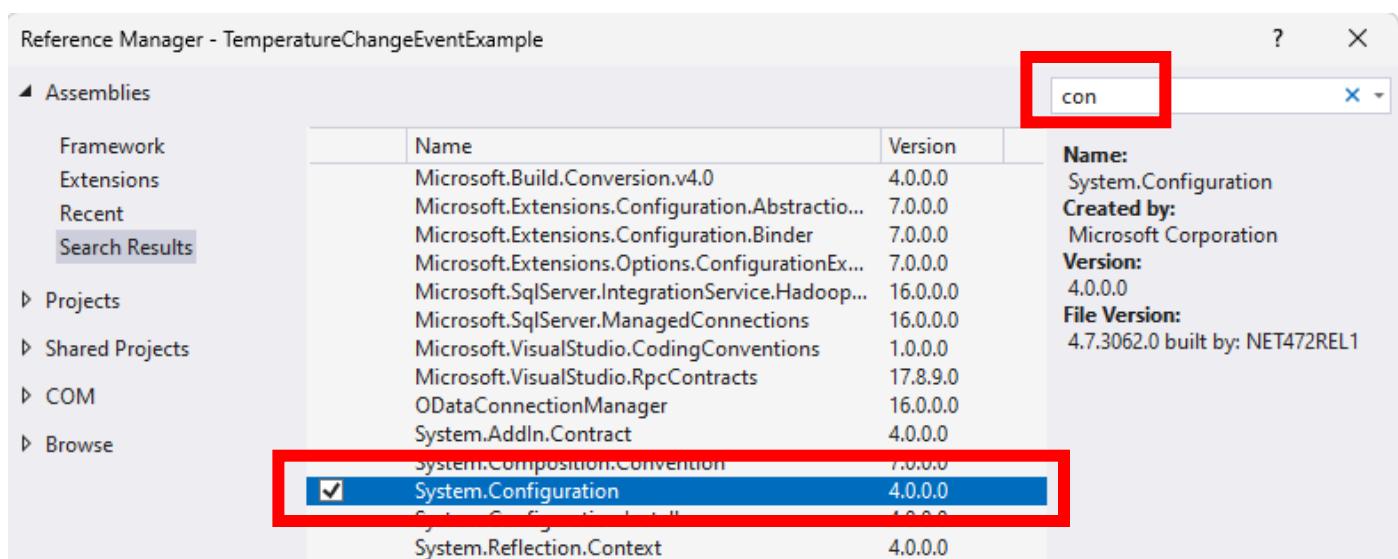
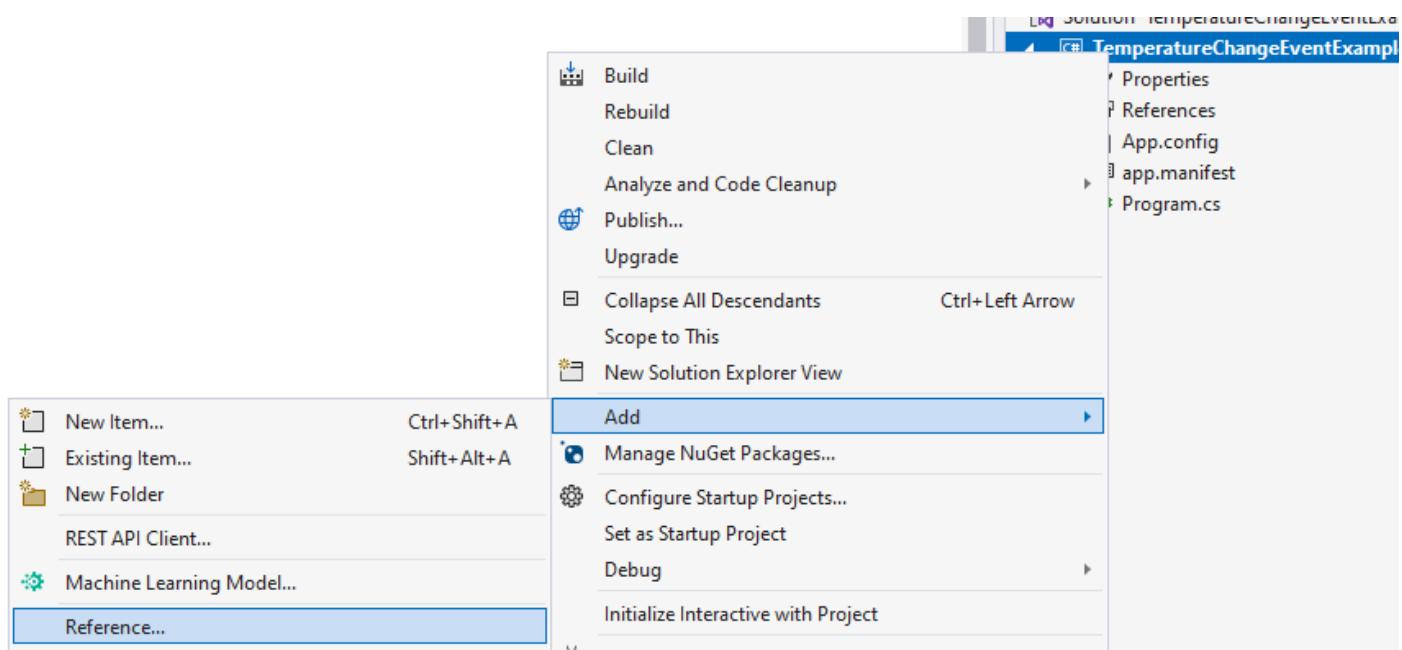
- Framework Version:
 - Confirm that your project is targeting a version of the .NET Framework that supports `ConfigurationManager`. Open your project properties and check the "Target framework" setting. It should be set to a version of the .NET Framework (e.g., .NET Framework 4.x).

- Clean and Rebuild:
 - Try cleaning and rebuilding your solution. Right-click on your solution in Solution Explorer and choose "Clean," then right-click again and choose "Rebuild."
- Restart Visual Studio:
 - Sometimes, issues can be resolved by restarting Visual Studio.

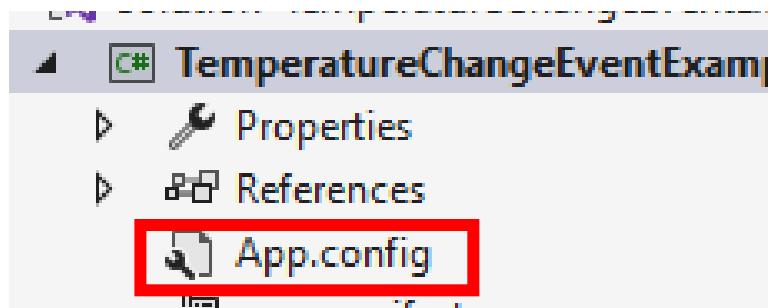
Remember that the `App.config` is primarily used during development. In production, the settings are often moved to the application's actual configuration file.

How to use App.config Example

عسان تستخدم المكتبه بتاعت ال reference system.configurations لازم تعملها الأول



بالنسبة لقيت الملف بناءً على config موجود بـ مالقيتهوش هنضيفه من خلال add>>Item ونسميه



ده الكود اللي لقيته جوه

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0"
sku=".NETFramework,Version=v4.7.2" />
    </startup>
</configuration>
```

انت بتكتب الكود بناءً على كله جوه ال configuration يعني مابين ال ده tag

```
<configuration>
</configuration>
```

اللي فوق ده اسمه tag عباره عن كلمة بين علامات اكبر من واصغر من وال compiler بيفهم ان الكود اللي جوه ال tag ده انتهي لما بتقفل ال tag عن طريق انك تكتب اسمه بين علامتين اكبر واصغر بـ بتحط قبل الاسم شرطه مايله

وتحط تعمل اكتر من tag جوه ال الواحد

ال tag عامتاً يمثل object او داتا معينه ولو هتحط جواه داتا يبقى بتحطها بالمنظار ده

```
<Tag>
--Your code here--
</Tag>
```

طيب لو ال tag ده ليه خصائص معينه زي انه مثلاً بيمثل button ال button ده ليه خصائص زي ال name وال text وهذا

تحط الخصائص دي قبل علامه اكبر من بالمنظار ده

```
<Tag
property1= PropertyValue
```

```
Property2=propertyvalue
```

```
>
```

```
--Your code here--
```

```
</Tag>
```

فيه حاجه كمان وهيا انك لو مش هتكتب حاجه جوه ال tag و هتكتب الخصائص بتاعته بس او حتى
مفيش خصائص تقدر تعمله بالمنظار ده

```
<Tag
```

```
property1= PropertyValue
```

```
Property2=propertyvalue
```

```
/>
```

```
<Tag />
```

طيب تعالى نعمل الكود بتاع ال connection string

هنا قالك فيه طريقتين

اول طريقة انك تعمل tag واحد تحط فيه اعدادات البرنامج كلها ويكون من ضمنها ال
connection string

وده عن طريق انك بتعمل tag تكتب فيه add key وتكتب اسم لـ key والقيمه بتاعته زي كده
الـ key ده الاسم اللي هتسعدني القيمه من خلاله يعني كأنك بتعرف متغير

```
<add key="ConnectionString" value="Data Source=ServerName;Initial Catalog=DatabaseName;Integrated Security=True" />
<add key="LogLevel" value="Debug" />
<add key="koko" value="Koko Value" />
```

في الكود اللي فات احنا كأننا عرفنا متغير واديئاه القيمه بتاعته

تاني طريقة وهيا انك تعمله في object لوحده كأنك بتخزن section

```
<connectionStrings>
    <add name="MyDbConnection" connectionString="Data Source=ServerName;Initial
Catalog=DatabaseName;Integrated Security=True" providerName="System.Data.SqlClient" />
</connectionStrings>
```

وده الكود كله

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
        <add key="ConnectionString" value="Data Source=ServerName;Initial Catalog=DatabaseName;Integrated
Security=True" />
        <add key="LogLevel" value="Debug" />
        <add key="koko" value="Koko Value" />
```

```

</appSettings>

<connectionStrings>
    <add name="MyDbConnection" connectionString="Data Source=ServerName;Initial
Catalog=DatabaseName;Integrated Security=True" providerName="System.Data.SqlClient" />
</connectionStrings>
</configuration>

```

بالنسبة لي أنا هاخد الكود وأضيفه عالملي اللي عندي

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
    </startup>

    <appSettings>
        <add key="ConnectionString" value="Data Source=ServerName;Initial Catalog=DatabaseName;Integrated
Security=True" />
        <add key="LogLevel" value="Debug" />
        <add key="koko" value="Koko Value" />
    </appSettings>

    <connectionStrings>
        <add name="MyDbConnection" connectionString="Data Source=ServerName;Initial
Catalog=DatabaseName;Integrated Security=True" providerName="System.Data.SqlClient" />
    </connectionStrings>
</configuration>

```

طيب نيجي للكود

أول حاجه هنضيف المكتبه

```
using System.Configuration;
```

طيب عشان تستدعي القيم اللي خزنتها هتعمل ايه؟

أول حاجه هتستدعي ال configuration manager وبعدين نقطه وبعدين اسم ال tag اللي مخزن
تحبيه القيمه اللي انت عاييزها وبعدين بتكتب ال name او ال key

لو كنت مخزن key فالانت بتكتب اسم ال key وخلصنا

انما لو كنت مخزن object فهتكتب ال name وبعدها نقطه وبعدها تشووف انت عاييز تستدعي ايه من
جواه

```
using System;
using System.Configuration;
```

```
class NullableExample
{
    static void Main()
    {
        string ConnectionString = ConfigurationManager.AppSettings["ConnectionString"];
        string LogLevel = ConfigurationManager.AppSettings["LogLevel"];
        string koko = ConfigurationManager.AppSettings["koko"];
        string MyDbConnection = ConfigurationManager.ConnectionStrings["MyDbConnection"].ConnectionString;
        Console.WriteLine($"\\nConnectionString: {ConnectionString}");
        Console.WriteLine($"\\nLogLevel: {LogLevel}");
        Console.WriteLine($"\\nkoko: {koko}");
        Console.WriteLine($"\\nMyDbConnection: {MyDbConnection}");
        Console.ReadKey();
    }
}
```

خلي بالك لو هتشغل البرنامج عند العميل مش هتعدل عال app config لا الملف هيكون واحد اسم .config البرنامج

Quiz

What is App.config in C# used for?

To store connection strings, app settings, and other configuration parameters

To access settings defined in the configuration file

All of the above

What format is the App.config file in?

JSON

XML

CSV

YAML

What class is used to access the settings defined in App.config?

System.Configuration.AppSettings

System.Configuration.ConfigurationManager

System.Configuration.ConnectionStrings

System.Configuration.XmlConfiguration

Which reference needs to be added for using ConfigurationManager?

System.ConfigurationManager

System.ConfigurationUtility

System.Configuration

System.Config

What is the typical file extension for the configuration file in production?

.config

.xml

.cfg

.conf

What is String Builder and Why?

دلوقي كلنا عرفنا انه ال string هوا عباره عن immutable data type يعني قيمته لا يمكن ان تتغير فكل مره انت بتعدل عاليمه اللي فيه بيروح يعمل نسخه منه بالقيمه الجديد عشان كده عملوا حاجه تانيه اسمها string builder وده mutable datatype

فکرته انه بيعمل buffer بحيث انك تقدر تعديل عال string ولو ال string ده اتملي بيعمل ويضيف عليه

فعدد ال string هنا اقل بكثير من ال copies

What is String Builder?

- **StringBuilder in C# is a class provided by the .NET Framework that belongs to the System.Text namespace.**
- **It is designed to efficiently manipulate strings, especially when there are multiple concatenations or modifications involved.**
- **The primary advantage of StringBuilder over simple string concatenation (+ operator) is its ability to minimize memory overhead and improve performance in scenarios where you're making repeated modifications to a string.**

key points about StringBuilder:

- **Mutable:**
 - Unlike regular strings in C#, which are immutable (meaning once created, their content cannot be changed), StringBuilder provides a mutable representation of a sequence of characters. This mutability allows you to modify the content of the string without creating new instances.
- **Efficient Concatenation:**
 - When you concatenate strings using the + operator or String.Concat, a new string is created, and the old ones are discarded. This process can be inefficient, especially when dealing with a large number of concatenations. StringBuilder addresses this issue by providing a more efficient mechanism for building and modifying strings.
- **Performance:**
 - StringBuilder is designed for better performance in scenarios where you need to concatenate or modify strings frequently. It uses a resizable buffer to store the characters, and as you append or modify content, it adjusts the buffer size accordingly. This helps avoid unnecessary memory allocations and reduces the overhead associated with string manipulation.
- **Methods for Modification:**
 - StringBuilder provides methods like Append, Insert, Remove, and Replace that allow you to modify the content of the string efficiently.

```
using System;
using System.Diagnostics;
using System.Text;

class NullableExample
{
    static void ConcatenateStrings(int iterations) {
        string result = "";
        for (int i=0;i<iterations;i++) {
            result += "a";
        }
    }
}
```

```

}

static void ConcatenateStringBuilder(int iterations) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < iterations; i++)
    {
        sb.Append("a");
    }
    string result=sb.ToString();
}

static void Main()
{
    int Iterations = 200000;
    Stopwatch stopwatch1 = Stopwatch.StartNew();
    ConcatenateStrings(Iterations);
    stopwatch1.Stop();
    Console.WriteLine($"ConcatenateStrings took : {stopwatch1.ElapsedMilliseconds}");

    Stopwatch stopwatch2 = Stopwatch.StartNew();
    ConcatenateStringBuilder(Iterations);
    stopwatch2.Stop();
    Console.WriteLine($"ConcatenateStrings took : {stopwatch2.ElapsedMilliseconds}");

    Console.ReadKey();
}
}

```

`StringBuilder` in C# is a class provided by the .NET Framework that belongs to the `System.Text` namespace. It is designed to efficiently manipulate strings, especially when there are multiple concatenations or modifications involved. The primary advantage of `StringBuilder` over simple string concatenation (`+` operator) is its ability to minimize memory overhead and improve performance in scenarios where you're making repeated modifications to a string.

Here are some key points about `StringBuilder`:

- Mutable:
 - Unlike regular strings in C#, which are immutable (meaning once created, their content cannot be changed), `StringBuilder` provides a mutable representation of a sequence of characters. This mutability allows you to modify the content of the string without creating new instances.
- Efficient Concatenation:
 - When you concatenate strings using the `+` operator or `String.Concat`, a new string is created, and the old ones are discarded. This process can be inefficient, especially when dealing with a large number of concatenations. `StringBuilder` addresses this issue by providing a more efficient mechanism for building and modifying strings.

- Performance:
 - `StringBuilder` is designed for better performance in scenarios where you need to concatenate or modify strings frequently. It uses a resizable buffer to store the characters, and as you append or modify content, it adjusts the buffer size accordingly. This helps avoid unnecessary memory allocations and reduces the overhead associated with string manipulation.
- Methods for Modification:
 - `StringBuilder` provides methods like `Append`, `Insert`, `Remove`, and `Replace` that allow you to modify the content of the string efficiently.
- Capacity Management:
 - You can explicitly set the initial capacity of the `StringBuilder` to reduce the number of reallocations if you have an estimate of the final string size.

What is Cryptography?

الـ **cryptography** معناها التشفير وفایدته انك حتى لو اتھکرت الهاکر هیوصل للمعلومه مشفرة ومش
هیعرف يستفيد منها

وبيتم تشفیر البيانات عن طريق **algorithms** معقدة

دي المكتبه اللي هنستخدمها في التشفير

System.Security.Cryptography

وبيقولك فيه ٣ أنواع من التشفير هما

(hashing-symmetric encryption – asymmetric encryption)

What is Cryptography?

- Cryptography is a method of protecting information and communications through the use of codes, so that only those for whom the information is intended can read and process it.
- In computer science, cryptography refers to secure information and communication techniques derived from mathematical concepts and a set of rule-based calculations called algorithms, to transform messages in ways that are hard to decipher.
- These deterministic algorithms are used for cryptographic key generation, digital signing, verification to protect data privacy, web browsing on the internet and confidential communications such as credit card transactions and email.

- In C#, you can use the `System.Security.Cryptography` namespace to work with cryptographic functions.
- Here's a brief overview of some commonly used cryptographic operations in C# such as:
 - Hashing
 - Symmetric Encryption
 - Asymmetric Encryption (Public Key Cryptography)

Cryptography is a method of protecting information and communications through the use of codes, so that only those for whom the information is intended can read and process it.

In computer science, **cryptography** refers to secure information and communication techniques derived from mathematical concepts and a set of rule-based calculations called algorithms, to transform messages in ways that are hard to decipher. These deterministic algorithms are used for cryptographic key generation, digital signing, verification to protect data privacy, web browsing on the internet and confidential communications such as credit card transactions and email.

Cryptography is the practice and study of techniques for secure communication in the presence of third parties. In C#, you can use the `System.Security.Cryptography` namespace to work with cryptographic functions. Here's a brief overview of some commonly used cryptographic operations in C# such as:

- Hashing
- Symmetric Encryption
- Asymmetric Encryption (Public Key Cryptography)

Cryptography Quiz

What is cryptography?

A method of protecting information and communication through the use of codes

A programming language

A type of encryption

A method of securing physical objects

What does cryptography in computer science refer to?

Secure information and communication techniques derived from mathematical concepts

Creating secure passwords

Data visualization techniques

Data compression techniques

What can deterministic algorithms in cryptography be used for?

Cryptographic key generation, digital signing, and verification

Database management

Network security

Mobile app development

What namespace can you use in C# for cryptographic functions?

System.Security.Cryptography

System.Data

System.IO

System.Net

What are some commonly used cryptographic operations in C#?

Hashing, symmetric encryption, and asymmetric encryption

Sorting and searching

Array manipulation

Exception handling

What is Hashing?

ال hashing هي طريقة لتشغير المعلومات وده بيكون one way يعني لما بيشفر لك الداتا بيطعلوك ماتفهمش منه حاجه وماتقدرش ترجعه لاصله text

اما هنعرف القيمه الاصلية منين؟

قالك لو عندك مثلا باسورد بتعمله hashing هيطعلوك text تخزنه في الداتابيز وبعدين لما يجي اليوزر يكتب الباسورد بتاعه هتاخذ الباسورد اللي اتنكتب ده وبعدين تعمله تشفير هو كمان وبعدها تقارن المترشف بالمتشرف

يعني الحل الوحيد لو نسيت الباسورد هو انك تقعد تجرب تدخله او تعمله reset عشان كده لازم تستعمل نفس ال algorithm في التشفير عشان تطلع نفس النتيجه عشان تستعمل ال hashing بتستعمل حاجه اسمها SHA-265 وده اختصار ل Secure hash algorithm خاصه بمجموعتين من 265 bit معناها الكود او ال text الناتج عن عملية التشفير دي بيكون 64 حرف ممثلين بال hexa decimal ضربتهم في 4 يطعلوك ال 265 bit

What is Hashing?

- Hash functions are commonly used to create a fixed-size string of characters, which is typically a hash value, from variable-size input data.
- Hash functions, by design, are one-way functions.
- This means that you cannot directly reverse a hash to retrieve the original data. Hashing is primarily used for integrity verification and not for data retrieval.
- If you need to check whether a given input matches a previously computed hash, you can hash the new input and compare the resulting hash with the stored hash. If the hashes match, it suggests that the input data is the same.

What is SHA-256?

- SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that belongs to the SHA-2 family of hash functions.
- It's one of the widely used hash functions for various security applications and protocols.
- The "256" in SHA-256 refers to the bit length of the hash output, which is 256 bits.
- To use SHA-256 in programming, you typically leverage libraries or classes provided by the programming language. In C#, for example, you can use the System.Security.Cryptography namespace

Common use cases for SHA-256 include:

- Password Hashing: Storing hashed passwords rather than plain text in databases for security.
- Digital Signatures: Creating digital signatures for documents and messages to ensure authenticity and integrity.
- Blockchain Technology: SHA-256 is used in many blockchain protocols (e.g., Bitcoin) to secure transactions and blocks.
- Data Integrity: Verifying the integrity of transmitted or stored data by comparing hash values.

Hashing:

Hash functions are commonly used to create a fixed-size string of characters, which is typically a hash value, from variable-size input data.

Hash functions, by design, are one-way functions. This means that you cannot directly reverse a hash to retrieve the original data. Hashing is primarily used for integrity verification and not for data retrieval.

If you need to check whether a given input matches a previously computed hash, you can hash the new input and compare the resulting hash with the stored hash. If the hashes match, it suggests that the input data is the same.

SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that belongs to the SHA-2 family of hash functions. It's one of the widely used hash functions for various security applications and protocols. The "256" in SHA-256 refers to the bit length of the hash output, which is 256 bits.

Common use cases for SHA-256 include:

- Password Hashing: Storing hashed passwords rather than plain text in databases for security.
- Digital Signatures: Creating digital signatures for documents and messages to ensure authenticity and integrity.
- Blockchain Technology: SHA-256 is used in many blockchain protocols (e.g., Bitcoin) to secure transactions and blocks.
- Data Integrity: Verifying the integrity of transmitted or stored data by comparing hash values.

To use SHA-256 in programming, you typically leverage libraries or classes provided by the programming language. In C#, for example, you can use the `System.Security.Cryptography` namespace, as demonstrated in the code examples provided in previous responses.

Hashing Example

عسان تستخدم الـ hashing بتسديعي المكتبه دي

```
using System.Security.Cryptography;
```

عشان تعمل hash بتستخدم ال object عن طريق SHA256 وتعمل using عن طريق CREATE اسمها

```
using (SHA256 sHA256=SHA256.Create()) {
```

وبعدين بتعمل array من النوع byte عشان تضيف فيه الكود اللي هيطلع

وهتستدعي اسمها compute hash method وتدلها النص اللي عايز تشفره بس عن طريق ال encoding

```
Byte[] hashBytes =
```

```
SHA256.ComputeHash(Encoding.UTF8.GetBytes(input));
```

ولما يخلص هتحول ال bytes ل string

```
return BitConverter.ToString( hashBytes ).Replace("-", "").ToLower();
```

بس كده وده الكود كله

```
using System;
using System.Security.Cryptography;
using System.Text;

class NullableExample
{
    static void Main()
    {
        string data = "Mohammed";
        string hashData = ComputeHash(data);

        Console.WriteLine($"Original:{data}");
        Console.WriteLine($"hashed:{hashData}");

        Console.WriteLine($"Hashed Data LENGTH: {hashData.Length}");

        Console.ReadKey();
    }

    static string ComputeHash(string input) {

        using (SHA256 sHA256=SHA256.Create()) {
            Byte[] hashBytes = sHA256.ComputeHash(Encoding.UTF8.GetBytes(input));
            return BitConverter.ToString(hashBytes).Replace("-", "").ToLower();
        }
    }
}
```

To use SHA-256 in programming, you typically leverage libraries or classes provided by the programming language. In C#, for example, you can use

the `System.Security.Cryptography` namespace, as demonstrated in the code examples provided in previous responses.

The following C# program is a simple example of computing the SHA-256 hash of a string using the `System.Security.Cryptography` namespace. Here's a breakdown of your code:

```
using System;
using System.Security.Cryptography;
using System.Text;

class Program
{
    static void Main()
    {
        // Input data
        string data = "Mohammed Abu-Hadhoud";

        // Compute the SHA-256 hash of the input data
        string hashedData = ComputeHash(data);

        // Display the original data and its hash
        Console.WriteLine($"Original Data: {data}");
        Console.WriteLine($"Hashed Data: {hashedData}");

        // Pause to keep the console window open for viewing the results
        Console.ReadKey();
    }

    static string ComputeHash(string input)
    {
        //SHA is Secutred Hash Algorithm.
        // Create an instance of the SHA-256 algorithm
        using (SHA256 sha256 = SHA256.Create())
        {
```

```

    // Compute the hash value from the UTF-8 encoded input string
    byte[] hashBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(input));

    // Convert the byte array to a lowercase hexadecimal string
    return BitConverter.ToString(hashBytes).Replace("-", "").ToLower();
}

}

}

```

Explanation:

1. Input Data: You have a string `data` containing the text "Mohammed Abu-Hadhoud".
2. Hashing: The `ComputeHash` method takes a string as input, converts it to a byte array using UTF-8 encoding, and then computes the SHA-256 hash using the `SHA256` class. The resulting hash is a byte array.
3. Formatting the Hash: The byte array is converted to a lowercase hexadecimal string using `BitConverter.ToString()` and then removing the hyphens with `Replace("-", "")`. This creates a string representation of the hash.
4. Displaying Results: The original data and its computed hash are then displayed in the console.
5. `Console.ReadKey()`: This line is used to pause the console window so that you can view the output. Pressing a key will close the console window.

The code demonstrates the basics of computing a SHA-256 hash in C#. It's a common practice to use hash functions for various purposes, such as data integrity verification and storing passwords securely. Keep in mind that for password storage, you should use additional techniques like salting to enhance security.

Hashing Homework

Go to the project in course 19 , and use hashing to hash the password and save it to database , then update your project accordingly (update the login, change password forms).

تم تكبير خانة الباسورد في الداتايز بحيث تسع ٦٤ حرفاً وتم استخدام نفس ال method في المثال السابق تم وضعها في كلاس ال util وتم استخدامها اثناء حفظ الباسورد في

=====
RememberUsernameAndPassword
=====

frmChangePassword>>>>btnSave_Click _User.Password
=clsUtil.ComputeHash(txtNewPassword.Text);
=====

frmAddUpdateUser>>>>btnSave_Click _User.Password =
clsUtil.ComputeHash(txtPassword.Text.Trim());
=====

frmLogin>>>>>>btnLogin_Click clsUser user=
clsUser.FindByUsernameAndPassword(txtUserName.Text.Trim(),clsUtil.
ComputeHash(txtPassword.Text.Trim()));
=====

Hashing Quiz

What is the main purpose of a hash function?

To create a fixed-size string from variable-size input data

To reverse a hash and retrieve the original data

Which of the following is NOT a common use case for SHA-256?

Storing hashed passwords in databases

Creating digital signatures

Data retrieval

What does the '256' in SHA-256 refer to?

The bit length of the hash output

The number of characters in the hash

The number of bytes used

Can you reverse a hash to retrieve the original data?

Yes, I can.

No, you cannot. Hash functions, by design, are one-way functions. This means that you cannot directly reverse a hash to retrieve the original data. Hashing is primarily used for integrity verification and not for data retrieval.

Hashing result will always produce:

256 characters.

64 characters because hexadecimal character takes 4 bits, so $64 \times 4 = 256$ bit

What is Symmetric Encryption?

ال symmetric encryption ودي بيتم تشفير الداتا وبتقدر تفك التشفير تاني
الحوار ده بيعتمد على key انت بتديهوله عشان يعمل تشفير وبتحتاج نفس ال key عشان تفك التشفير
زي ماكنا بنعمل في ال C++ لما كنا بنزود حرفين وننقص حرفين فبيطلع كلام مش مفهوم
بيقولك انه فيه اكتر من DES algorithm للتشفيـر منها وـهيـا اختصار لـ

Data encryption standard

و 3DS وـهيـا اختصار لـ triple des وـAES وـهيـا اختصار لـ

RC4 وـكمـان ال Advanced encryption standard

احـنا هـنسـتـخدم ال AES

هـنا بـيـقولـك انه ال key ال اللي ال AES بـيـكون حـجمـه 128 بت ليـهـ؟

عشان مثلا لو انت عامل الباسورد بتاعك من رقم واحد او حرف واحد ده بيكون من السهل تخمينه لأن اللي عايز يهكرك يدوب هيجرب يدخل الرقم ١٠ مرات ولو حرف فهيكون بعدد الحروف هيجرب لكن لو ال key بتاعك طويل زي في حالة ال aes هنا ١٢٨ bit هيكون صعب تخمينه طيب ال 128 bit دول كام حرف ؟
هما ١٦ حرف لأن الحرف الواحد بيأخذ مساحه byte واحد اللي هيا 8 bits فلو ضربت $128 = 8 * 16$
بيقولك كمان انه ال AES بتسخدم ٣ احجام من ال key (128bit-192-bit-256bit)
طيب ال 128bit عشان يقدر حد يخمنه هيحتاج يعمل محاولات عددها ٢ اس ١٢٨ وده الشائع استخدامه

What is Symmetric Encryption?

- Symmetric encryption is a type of encryption where the same key is used for both the encryption and decryption of the data.
- In C#, the .NET Framework and .NET Core provide classes in the `System.Security.Cryptography` namespace to perform symmetric encryption.
- The most commonly used symmetric encryption algorithms are DES (Data Encryption Standard), 3DES (Triple DES), AES (Advanced Encryption Standard), and RC4.

128-bit Key, What does that mean?

- They key size is commonly used for AES encryption is 128-bit, what does that mean?
- The "key size" in the context of AES (Advanced Encryption Standard) refers to the length of the cryptographic key used for encryption and decryption. In AES, the key size determines the number of bits in the key, and it directly affects the strength of the encryption.

128-bit How many characters?

- 16 characters, why?
- Each character represents one byte (8 bits), so
16 characters * 8 bits/character = 128 bits

AES supports three key sizes.

- 128-bit, 192-bit, and 256-bit.
- The number in each key size (128, 192, or 256) represents the length of the key in bits. Here's a breakdown of what each of these key sizes means:
 - 128-bit Key:
 - The key is 128 bits long, which means it has 2^{128} possible combinations.
 - This is considered strong encryption and is widely used for many secure applications.
 - 192-bit Key:
 - The key is 192 bits long, providing a higher level of security than 128-bit keys.
 - While it offers increased security, 192-bit keys are less commonly used than 128-bit and 256-bit keys.
 - 256-bit Key:
 - The key is 256 bits long, providing the highest level of security among the three key sizes.
 - AES with a 256-bit key is often used in situations where maximum security is required.

The highest key size the harder to attack.

- The key size directly impacts the difficulty of a brute-force attack, where an attacker tries all possible combinations of the key to decrypt the data. As the key size increases, the number of possible combinations grows exponentially, making it significantly more difficult and time-consuming for an attacker to break the encryption through brute force.

Should we always use the highest key size?

- No.
- In practice, a 128-bit key is considered secure for most applications, and it is widely used in various cryptographic protocols and systems.
- However, for scenarios where an extra layer of security is desired or required, one might opt for a 192-bit or 256-bit key.
- It's important to note that the increased security comes with a trade-off in terms of computational overhead, as encryption and decryption with longer keys can be more computationally intensive.

Symmetric encryption is a type of encryption where the same key is used for both the encryption and decryption of the data. In C#, the .NET Framework and .NET Core provide classes in the `System.Security.Cryptography` namespace to perform symmetric encryption. The most commonly used symmetric encryption algorithms are DES (Data Encryption Standard), 3DES (Triple DES), AES (Advanced Encryption Standard), and RC4.

They key size is commonly used for AES encryption is 128-bit, what does that mean?

The "key size" in the context of **AES (Advanced Encryption Standard)** refers to the length of the cryptographic key used for encryption and decryption. In AES, the key size determines the number of bits in the key, and it directly affects the strength of the encryption.

AES supports three key sizes: 128-bit, 192-bit, and 256-bit. The number in each key size (128, 192, or 256) represents the length of the key in bits. Here's a breakdown of what each of these key sizes means:

- 128-bit Key:
 - The key is 128 bits long, which means it has 2^{128} possible combinations.
 - This is considered strong encryption and is widely used for many secure applications.
- 192-bit Key:
 - The key is 192 bits long, providing a higher level of security than 128-bit keys.
 - While it offers increased security, 192-bit keys are less commonly used than 128-bit and 256-bit keys.
- 256-bit Key:
 - The key is 256 bits long, providing the highest level of security among the three key sizes.
 - AES with a 256-bit key is often used in situations where maximum security is required.

The key size directly impacts the difficulty of a brute-force attack, where an attacker tries all possible combinations of the key to decrypt the data. As the key size increases, the number of

possible combinations grows exponentially, making it significantly more difficult and time-consuming for an attacker to break the encryption through brute force.

In practice, a 128-bit key is considered secure for most applications, and it is widely used in various cryptographic protocols and systems. However, for scenarios where an extra layer of security is desired or required, one might opt for a 192-bit or 256-bit key. It's important to note that the increased security comes with a trade-off in terms of computational overhead, as encryption and decryption with longer keys can be more computationally intensive.

You should not lose the key!

Remember that in real-world scenarios, you would typically use a more secure method for key management, and for sensitive information, it's important to handle encryption keys securely. Additionally, consider using authenticated encryption modes for added security.

Symmetric Encryption Example

قبل مانبدأ تشفير لازم نحدد ال key وال key ده ليه طرق في تخزينه ولازم مايسيعش لكن حاليا هنحطه في string ولازم يكون 16 حرف

```
string Key = "1234567890123456";
```

عشان نشفر الداتا هناخد object من aes ونعمله جوه using

```
using (Aes aes=Aes.Create()) {
```

ال object ده فيه parameters صح؟

صح

من ال parameters دي حاجه اسمها key وفيه بخزن ال key اللي انا عملته بس في شكل bytes

```
aes.Key=Encoding.UTF8.GetBytes(Key);
```

بعدين لازم اعمل vector عشان يخزن فيه الداتا هنقوله ناخذ حجم ال key بال bits ونقسمه على 8 عشان يديينا عدد الخلايا اللي في ال vector

```
aes.IV = new byte[aes.BlockSize / 8];
```

بعدين هنعمل encryptor ونديله ال key وال vector اللي موجودين في ال aes

```
ICryptoTransform encryptor=aes.CreateEncryptor(aes.Key,aes.IV);
```

بعدين نفتح اتصال مع ال memory

```
using (var msEncrypt=new System.IO.MemoryStream()) {
```

بعدين هنأخذ object من حاجه اسمها crypto stream

```
using (var csEncrypt=new CryptoStream(msEncrypt,encryptor,CryptoStreamMode.Write)) {
```

بعدين هنعمل stream writer ونقوله يكتب اللي طلعه ونرجع الداتا بعد التشفير

```
using (var swEncrypt=new System.IO.StreamWriter(csEncrypt)) {  
    swEncrypt.WriteLine(plainText);  
}  
return Convert.ToString(msEncrypt.ToArray());
```

طيب في فاك التشفير ؟

نفس الخطوات بس هتشيل كلمه encrypt وتحط decrypt وتشيل write وتحط read وهكذا

وده الكود كله

```
using System;  
using System.Security.Cryptography;  
using System.Text;  
  
class NullableExample  
{  
    static void Main()  
    {  
        string OriginalData = "Sensitive information";  
  
        string Key = "1234567890123456";  
  
        string encryptedData = Encrypt(OriginalData,Key);  
        string decryptedData = Decrypt(encryptedData, Key);  
  
        Console.WriteLine($"OriginalData: {OriginalData}");  
        Console.WriteLine($"encryptedData: {encryptedData}");  
        Console.WriteLine($"decryptedData: {decryptedData}");  
  
        Console.ReadKey();  
    }  
  
    static string Encrypt(string plainText,string Key) {  
  
        using (Aes aes=Aes.Create()) {  
            aes.Key=Encoding.UTF8.GetBytes(Key);  
            aes.IV = new byte[aes.BlockSize / 8];  
  
            ICryptoTransform encryptor=aes.CreateEncryptor(aes.Key,aes.IV);  
  
            using (var msEncrypt=new System.IO.MemoryStream()) {  
  
                using (var csEncrypt=new CryptoStream(msEncrypt,encryptor,CryptoStreamMode.Write)) {
```

```

        using (var swEncrypt=new System.IO.StreamWriter(csEncrypt)) {
            swEncrypt.Write(plainText);
        }
        return Convert.ToBase64String(msEncrypt.ToArray());
    }
}

static string Decrypt(string cipherText,string Key) {
    using (Aes aes = Aes.Create())
    {
        aes.Key = Encoding.UTF8.GetBytes(Key);
        aes.IV = new byte[aes.BlockSize / 8];

        ICryptoTransform decryptor = aes.CreateDecryptor(aes.Key, aes.IV);

        using (var msdecrypt = new System.IO.MemoryStream(Convert.FromBase64String(cipherText)))
        {
            using (var csdecrypt = new CryptoStream(msdecrypt, decryptor, CryptoStreamMode.Read))
            {
                using (var srdecrypt = new System.IO.StreamReader(csdecrypt))
                {

                    return srdecrypt.ReadToEnd();
                }
            }
        }
    }
}

```

The Following C# program demonstrates basic symmetric encryption and decryption using the AES (Advanced Encryption Standard) algorithm. Here's a breakdown of your code:

```

using System;
using System.Security.Cryptography;
using System.Text;

class Program
{
    static void Main()
    {
        // Original data
        string originalData = "Sensitive information";

        // Key for AES encryption (128-bit key)
        string key = "1234567890123456";
    }
}

```

```
// Encrypt the original data
string encryptedData = Encrypt(originalData, key);

// Decrypt the encrypted data
string decryptedData = Decrypt(encryptedData, key);

// Display results
Console.WriteLine($"Original Data: {originalData}");
Console.WriteLine($"Encrypted Data: {encryptedData}");
Console.WriteLine($"Decrypted Data: {decryptedData}");

}

static string Encrypt(string plainText, string key)
{
    using (Aes aesAlg = Aes.Create())
    {
        // Set the key and IV for AES encryption
        aesAlg.Key = Encoding.UTF8.GetBytes(key);
        aesAlg.IV = new byte[aesAlg.BlockSize / 8];

        // Create an encryptor
        ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

        // Encrypt the data
        using (var msEncrypt = new System.IO.MemoryStream())
        {
            using (var csEncrypt = new CryptoStream(msEncrypt, encryptor,
CryptoStreamMode.Write))
                using (var swEncrypt = new System.IO.StreamWriter(csEncrypt))
                {
                    swEncrypt.Write(plainText);
                }
        }
    }
}
```

```

        // Return the encrypted data as a Base64-encoded string
        return Convert.ToBase64String(msEncrypt.ToArray());
    }
}

}

static string Decrypt(string cipherText, string key)
{
    using (Aes aesAlg = Aes.Create())
    {
        // Set the key and IV for AES decryption
        aesAlg.Key = Encoding.UTF8.GetBytes(key);
        aesAlg.IV = new byte[aesAlg.BlockSize / 8];

        // Create a decryptor
        ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

        // Decrypt the data
        using (var msDecrypt = new
System.IO.MemoryStream(Convert.FromBase64String(cipherText)))
            using (var csDecrypt = new CryptoStream(msDecrypt, decryptor,
CryptoStreamMode.Read))
                using (var srDecrypt = new System.IO.StreamReader(csDecrypt))
                {
                    // Read the decrypted data from the StreamReader
                    return srDecrypt.ReadToEnd();
                }
    }
}

```

Explanation:

1. Original Data: You have a string `originalData` containing sensitive information.

2. Encryption: The `Encrypt` method takes the original data and a key as input, encrypts the data using AES encryption, and returns the result as a Base64-encoded string.
3. Decryption: The `Decrypt` method takes the encrypted data and the same key, decrypts the data using AES decryption, and returns the original data.
4. Displaying Results: The original data, encrypted data, and decrypted data are then displayed in the console.

Symmetric Encryption Quiz

What is symmetric encryption?

A type of encryption where the same key is used for both encryption and decryption

A type of encryption where different keys are used for encryption and decryption

A type of encryption that uses public and private keys

A type of encryption that uses a random key for each encryption and decryption

What does a 128-bit key mean in AES encryption?

The key is 128 characters long

The key size of AES encryption is 128

The key has 128 possible combinations

The key has 128 bits and determines the strength of the encryption

How many characters are in a 128-bit key?

8 characters

16 characters

32 characters

64 characters

What are the three key sizes supported by AES?

64-bit, 128-bit, and 256-bit

128-bit, 192-bit, and 256-bit

256-bit, 512-bit, and 1024-bit

512-bit, 1024-bit, and 2048-bit

Does the key size directly affect the difficulty of a brute-force attack?

No, the key size does not affect the difficulty of a brute-force attack

Yes, the larger the key size, the easier the brute-force attack

Yes, the larger the key size, the harder the brute-force attack

No, the key size determines the number of possible combinations

Should we always use the highest key size?

Yes, the highest key size provides the best security

No; a 128-bit key is considered secure for most applications

Yes, longer keys are always more secure

No, longer keys are less secure

What is Asymmetric Encryption?

ال asymmetric encryption زيها زي ال keys بس دي بتسخدم اتنين واحد decryption عشان ال public والثاني encryption عشان ال private

هنا بيقولك انك تقدر تستخدم ال public key عشان تتأكد انك هتبعد الداتا للشخص الصح زي لما تيجي تبعدت لحد عن طريق ال share it او لو نسيت الباسورد بيعتلك رمز ودلوقتي لو فتحت ال gmail على جهاز جديد بيعت لجهازك القديم رقم

فيه مكتبات بنسخدمها عشان التشفير ده منها ال RSA و ECC و DSA

What is Asymmetric Encryption?

- Asymmetric encryption, also known as public-key cryptography, is a cryptographic system that uses pairs of keys:
 - Public keys, which are widely shared or distributed.
 - Private keys, which are kept secret.
- This system enables two parties to secure their communication over an insecure channel without having to share a secret key beforehand.

A brief overview of how asymmetric encryption works:

- Key Pairs:
 - Public Key: This key is made available to anyone and can be freely distributed. It's commonly used for encryption.
 - Private Key: This key is kept secret and is known only to the owner. It's used for decryption.
- Encryption and Decryption:
 - If Party A wants to send an encrypted message to Party B, Party A uses Party B's public key to encrypt the message.
 - Only Party B, who possesses the corresponding private key, can decrypt and read the message.

A brief overview of how asymmetric encryption works:

- Digital Signatures:
 - The roles can be reversed for digital signatures. If Party A wants to sign a message and prove its authenticity, Party A uses its private key to create a digital signature.
 - Anyone with access to Party A's public key can verify that the signature is valid, confirming that the message has not been tampered with and is indeed from Party A.

A brief overview of how asymmetric encryption works:

- Key Distribution:
 - Asymmetric encryption helps solve the key distribution problem inherent in symmetric key cryptography. In a symmetric key system, both parties need to have the same secret key, which can be challenging to distribute securely.
 - With asymmetric encryption, each participant has their own pair of public and private keys.

Algorithms used in asymmetric encryption:

- Popular algorithms used in asymmetric encryption include:
 - RSA (Rivest-Shamir-Adleman)
 - ECC (Elliptic Curve Cryptography)
 - and DSA (Digital Signature Algorithm).
- Asymmetric encryption is often used in combination with symmetric encryption for efficiency and security in various cryptographic protocols and applications, such as secure communication over the internet, digital signatures, and key exchange in secure protocols like SSL/TLS.

Asymmetric encryption, also known as public-key cryptography, is a cryptographic system that uses pairs of keys: public keys, which are widely shared or distributed, and private keys, which are kept secret. This system enables two parties to secure their communication over an insecure channel without having to share a secret key beforehand.

Here's a brief overview of how asymmetric encryption works:

- Key Pairs:
 - Public Key: This key is made available to anyone and can be freely distributed. It's commonly used for encryption.
 - Private Key: This key is kept secret and is known only to the owner. It's used for decryption.
- Encryption and Decryption:
 - If Party A wants to send an encrypted message to Party B, Party A uses Party B's public key to encrypt the message.
 - Only Party B, who possesses the corresponding private key, can decrypt and read the message.
- Digital Signatures:
 - The roles can be reversed for digital signatures. If Party A wants to sign a message and prove its authenticity, Party A uses its private key to create a digital signature.

- Anyone with access to Party A's public key can verify that the signature is valid, confirming that the message has not been tampered with and is indeed from Party A.
- Key Distribution:
 - Asymmetric encryption helps solve the key distribution problem inherent in symmetric key cryptography. In a symmetric key system, both parties need to have the same secret key, which can be challenging to distribute securely.
 - With asymmetric encryption, each participant has their own pair of public and private keys.

Popular algorithms used in asymmetric encryption include RSA (Rivest-Shamir-Adleman), ECC (Elliptic Curve Cryptography), and DSA (Digital Signature Algorithm). Asymmetric encryption is often used in combination with symmetric encryption for efficiency and security in various cryptographic protocols and applications, such as secure communication over the internet, digital signatures, and key exchange in secure protocols like SSL/TLS.

Why two Keys?

The use of two keys (public and private keys) in asymmetric encryption serves several important purposes:

1. Encryption and Decryption:

-
- Public Key: Used for encryption. Anyone can use the public key to encrypt a message.
- Private Key: Used for decryption. Only the owner of the private key can decrypt messages encrypted with the corresponding public key.
- This separation of roles allows for secure communication between parties without the need for them to share a secret key for encryption and decryption.

1. Digital Signatures:

-
- Private Key: Used to create a digital signature, providing a way for the owner of the private key to authenticate messages.

- Public Key: Used to verify the digital signature. Anyone with access to the public key can verify that the message was signed by the corresponding private key.
1. Digital signatures help ensure the integrity and authenticity of messages, allowing the recipient to verify that a message has not been tampered with and that it indeed comes from the claimed sender.
 2. Key Distribution:
 -
 - Public Keys: Can be freely distributed and shared. For example, in secure communication, users can publish their public keys on a public key server or share them directly with others.
 - Private Keys: Kept secret by the key owner. They do not need to be shared with others.
 1. This separation simplifies the key distribution problem that exists in symmetric key cryptography, where both parties need to have the same secret key. With asymmetric encryption, each user has their own pair of keys, and they only need to share their public keys.

The combination of public and private keys in asymmetric encryption provides a flexible and secure framework for various cryptographic applications, including secure communication, digital signatures, and key exchange.

Asymmetric Encryption Example

هنسخدم ال RSA في التشفير

الفكره هنا انك بتعمل ملف xml بتحط فيه داتا

ايه هيا الداتا ملناش دعوه المهم انه ملف فيه داتا وخلاص

الملف ده بيكون هو ال private key

وبعدين بتنسخ جزء من الداتا دي في ملف تاني وبيكون ده هو ال public key

طيب قبل التشفير بنجهز الداتا الأول

فبنأخذ rsa service من كلاس object

```
using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
```

وبعدين بنأخذ ال parameters اللي من النوع public بس وبنحولها لملف xml ونحولها تاني ل string

```
string publicKey = rsa.ToXmlString(false);
```

ونأخذ كل الملف نحوله ل private key

```
string privateKey = rsa.ToXmlString(true);
```

وبعدين بنقوله encrypt و decrypt

طيب ال encrypt بتشتغل ازاي؟

هناخد object من rsa برضه ونحط فيه ال public key بعد مانرجعه تاني لملف xml وبعدين بنقوله encrypt

```
rsa.FromXmlString(publicKey);
```

```
byte[] encryptedData = rsa.Encrypt(Encoding.UTF8.GetBytes(plainText), false);
return Convert.ToString(encryptedData);
```

طيب بنفك التشغیر از اي؟

بنأخذ ال private key نحوله ل xml ونخزنه في ال rsa وبعدين بنقوله decrypt

```
using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
{
    rsa.FromXmlString(privateKey);
```

```
byte[] encryptedData = Convert.FromBase64String(cipherText);
byte[] decryptedData = rsa.Decrypt(encryptedData, false);
```

بس كده

وده الكود كله

```
using System;
using System.Security.Cryptography;
using System.Text;

class NullableExample
{
    static void Main()
    {
        try
        {
            using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
            {
                string publicKey = rsa.ToXmlString(false);
```

```

        string privateKey = rsa.ToXmlString(true);

        string originalMessage = "Hello, this is a secret message!";

        string encryptedMessage = Encrypt(originalMessage, publicKey);

        string decryptedMessage = Decrypt(encryptedMessage, privateKey);

        Console.WriteLine($"\\n\\nPublic Key:\\n {publicKey}");
        Console.WriteLine($"\\n\\nPrivate Key:\\n {privateKey}");
        Console.WriteLine($"\\nOriginal Message:\\n {originalMessage}");
        Console.WriteLine($"\\nEncrypted Message:\\n {encryptedMessage}");
        Console.WriteLine($"\\nDecrypted Message:\\n {decryptedMessage}");

        Console.WriteLine("\\nPress any key to exit...");
        Console.ReadKey();
    }
}

catch (CryptographicException ex)
{
    Console.WriteLine($"Encryption/Decryption error: {ex.Message}");
    Console.ReadKey();
}
catch (Exception ex)
{
    Console.WriteLine($"An unexpected error occurred: {ex.Message}");
    Console.ReadKey();
}
}

static string Encrypt(string plainText, string publicKey)
{
try
{
    using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
    {
        rsa.FromXmlString(publicKey);

        byte[] encryptedData = rsa.Encrypt(Encoding.UTF8.GetBytes(plainText), false);
        return Convert.ToString(encryptedData);
    }
}
catch (CryptographicException ex)
{
    Console.WriteLine($"Encryption error: {ex.Message}");
    throw;
}
}

static string Decrypt(string cipherText, string privateKey)
{
try
{
    using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
    {
        rsa.FromXmlString(privateKey);

```

```

        byte[] encryptedData = Convert.FromBase64String(cipherText);
        byte[] decryptedData = rsaDecrypt(encryptedData, false);

        return Encoding.UTF8.GetString(decryptedData);
    }
}
catch (CryptographicException ex)
{
    Console.WriteLine($"Decryption error: {ex.Message}");
    throw;
}
}
}

```

Asymmetric Encryption (Public Key Cryptography):

Asymmetric encryption involves a pair of public and private keys. Data encrypted with the public key can only be decrypted with the corresponding private key.

```

using System;
using System.Security.Cryptography;
using System.Text;

class AsymmetricEncryptionExample
{
    static void Main()
    {
        try
        {
            // Generate public and private key pair
            using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
            {
                // Get the public key
                /*
                 * When exporting the public key, ToXmlString(false) is used with the argument
                 * set to false to indicate that only the public parameters should be included in the
                 * XML string.
                */
                string publicKey = rsa.ToXmlString(false);
            }
        }
    }
}

```

```
// Get the private key
string privateKey = rsa.ToXmlString(true);

// Original message
string originalMessage = "Hello, this is a secret message!";

// Encrypt using the public key
string encryptedMessage = Encrypt(originalMessage, publicKey);

// Decrypt using the private key
string decryptedMessage = Decrypt(encryptedMessage, privateKey);

// Display the results
Console.WriteLine($"\\n\\nPublic Key:\\n {publicKey}");
Console.WriteLine($"\\n\\nPrivate Key:\\n {privateKey}");
Console.WriteLine($"\\nOriginal Message:\\n {originalMessage}");
Console.WriteLine($"\\nEncrypted Message:\\n {encryptedMessage}");
Console.WriteLine($"\\nDecrypted Message:\\n {decryptedMessage}");

// Wait for user input before closing the console window
Console.WriteLine("\\nPress any key to exit...");
Console.ReadKey();

}

}

catch (CryptographicException ex)
{
    Console.WriteLine($"Encryption/Decryption error: {ex.Message}");
    Console.ReadKey();
}

catch (Exception ex)
{
    Console.WriteLine($"An unexpected error occurred: {ex.Message}");
    Console.ReadKey();
}

}
```

```
static string Encrypt(string plainText, string publicKey)
{
    try
    {
        using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
        {
            rsa.FromXmlString(publicKey);

            byte[] encryptedData = rsa.Encrypt(Encoding.UTF8.GetBytes(plainText), false);
            return Convert.ToString(encryptedData);
        }
    }
    catch (CryptographicException ex)
    {
        Console.WriteLine($"Encryption error: {ex.Message}");
        throw; // Rethrow the exception to be caught in the Main method
    }
}

static string Decrypt(string cipherText, string privateKey)
{
    try
    {
        using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
        {
            rsa.FromXmlString(privateKey);

            byte[] encryptedData = Convert.FromBase64String(cipherText);
            byte[] decryptedData = rsa.Decrypt(encryptedData, false);

            return Encoding.UTF8.GetString(decryptedData);
        }
    }
    catch (CryptographicException ex)
```

```

    {
        Console.WriteLine($"Decryption error: {ex.Message}");
        throw; // Rethrow the exception to be caught in the Main method
    }
}

```

This C# code demonstrates a simple example of asymmetric encryption using the RSA algorithm with the `RSACryptoServiceProvider` class in the .NET Framework. Here's a breakdown of the code:

- - **Key Generation:** A new instance of `RSACryptoServiceProvider` is created within a `using` statement. The `using` statement ensures that the `RSACryptoServiceProvider` object is properly disposed of when it goes out of scope.
 - Public and private keys are generated within this block.

```

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
{
    // Key generation and usage go here
}

```

- - **Public Key Extraction:** The public key is extracted using `ToXmlString(false)`, which means only the public parameters are included in the XML string.
 - The public key is then stored in the `publicKey` variable.

```
string publicKey = rsa.ToXmlString(false);
```

-

- **Private Key Extraction:** The private key is extracted using `ToXmlString(true)`, indicating that both public and private parameters are included in the XML string.
- The private key is stored in the `privateKey` variable.

```
string privateKey = rsa.ToXmlString(true);
```

-
- **Encryption:** The `Encrypt` method takes a plaintext message (`originalMessage`) and the public key (`publicKey`) as input.
- Inside the `Encrypt` method, a new `RSACryptoServiceProvider` instance is created and configured with the provided public key.
- The plaintext message is encrypted using the `Encrypt` method, and the result is converted to a Base64-encoded string.

```
byte[] encryptedData = rsa.Encrypt(Encoding.UTF8.GetBytes(plainText), false);
return Convert.ToBase64String(encryptedData);
```

-
- **Decryption:** The `Decrypt` method takes a ciphertext message (`cipherText`) and the private key (`privateKey`) as input.
- Inside the `Decrypt` method, a new `RSACryptoServiceProvider` instance is created and configured with the provided private key.
- The ciphertext message is decrypted using the `Decrypt` method, and the result is converted back to a UTF-8 encoded string.

```
byte[] decryptedData = rsa.Decrypt(encryptedData, false);
return Encoding.UTF8.GetString(decryptedData);
```

- - **Main Method:** The `Main` method orchestrates the key generation, encryption, and decryption processes.
 - It displays the public and private keys, the original message, the encrypted message, and the decrypted message on the console.

```
Console.WriteLine($"\\n\\nPublic Key:\\n {publicKey}");  
Console.WriteLine($"\\n\\nPrivate Key:\\n {privateKey}");  
Console.WriteLine($"\\nOriginal Message:\\n {originalMessage}");  
Console.WriteLine($"\\nEncrypted Message:\\n {encryptedMessage}");  
Console.WriteLine($"\\nDecrypted Message:\\n {decryptedMessage}");
```

- - **Error Handling:** The code includes exception handling (`try-catch` blocks) to capture and display errors related to encryption and decryption.

```
catch (CryptographicException ex)  
{  
    Console.WriteLine($"Encryption/Decryption error: {ex.Message}");  
    Console.ReadKey();  
}  
catch (Exception ex)  
{  
    Console.WriteLine($"An unexpected error occurred: {ex.Message}");  
    Console.ReadKey();  
}
```

This example demonstrates the basics of RSA encryption and decryption using the .NET Framework's `RSACryptoServiceProvider` class. It is important to note that this approach might not be suitable for all scenarios, and in modern applications, you might want to consider more advanced libraries or frameworks for cryptographic operations.

Remember to handle keys securely, as they play a crucial role in the security of cryptographic operations. The examples above are for educational purposes, and in a real-world scenario, you should use key management best practices.

Asymmetric Encryption Quiz

What is asymmetric encryption also known as?

Public-key cryptography

Symmetric encryption

Digital signatures

RSA

What are the two types of keys in asymmetric encryption?

Public and secret keys

Shared and private keys

Public and private keys

Symmetric and asymmetric keys

How is a message encrypted in asymmetric encryption?

Using the private key

Using the public key

Using both the public and private keys

Using a secret key

How is a message decrypted in asymmetric encryption?

Using the private key

Using the public key

Using both the public and private keys

Using a secret key

What is the purpose of digital signatures in asymmetric encryption?

To encrypt messages

To authenticate messages

To distribute public keys

To create a secret key

Encrypt and Decrypt Image Example

ده مثال عالتشفير

فبيقولك قبل ما تبدأ اعمل ملف في ال C وخد الصورة وحطها فيه

c:\Image

وبعدين حط الكود ده

هوا بيستخدم الكود بتاع ال aes

وفي التشفير بدل مكان بيكتب عال ram لا بيستخدم الملف اللي عمله

في عمليه التشفير هوا بيحول الصوره ل bytes وبعدين بيشفر ال bytes وبيرجعهم تاني

بس دي الفكره

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;
```

```

class NullableExample
{
    static void Main()
    {
        string inputFile = "c:\\\\Image\\\\MyImage.jpg";
        string encryptedFile = "c:\\\\Image\\\\encrypted.jpg";
        string decryptedFile = "c:\\\\Image\\\\decrypted.jpg";

        // Generate a random IV for each encryption operation
        byte[] iv;
        using (Aes aesAlg = Aes.Create())
        {
            iv = aesAlg.IV;
        }

        string key = "1234567890123456";

        EncryptFile(inputFile, encryptedFile, key, iv);
        DecryptFile(encryptedFile, decryptedFile, key, iv);

        Console.WriteLine("Encryption and decryption completed successfully.");
        Console.WriteLine("go to c:\\\\Image folder to see the results");
        Console.ReadKey();
    }

    static void EncryptFile(string inputFile, string outputFile, string key, byte[] iv)
    {
        using (Aes aesAlg = Aes.Create())
        {
            aesAlg.Key = System.Text.Encoding.UTF8.GetBytes(key);
            aesAlg.IV = iv;

            using (FileStream fsInput = new FileStream(inputFile, FileMode.Open))
            using (FileStream fsOutput = new FileStream(outputFile, FileMode.Create))
            using (ICryptoTransform encryptor = aesAlg.CreateEncryptor())
            using (CryptoStream cryptoStream = new CryptoStream(fsOutput, encryptor, CryptoStreamMode.Write))
            {
                // Write the IV to the beginning of the file
                fsOutput.Write(iv, 0, iv.Length);
                fsInput.CopyTo(cryptoStream);
            }
        }
    }

    static void DecryptFile(string inputFile, string outputFile, string key, byte[] iv)
    {
        using (Aes aesAlg = Aes.Create())
        {
            aesAlg.Key = System.Text.Encoding.UTF8.GetBytes(key);
            aesAlg.IV = iv;
        }
    }
}

```

```

        using (FileStream fsInput = new FileStream(inputFile, FileMode.Open))
        using (FileStream fsOutput = new FileStream(outputFile, FileMode.Create))
        using (ICryptoTransform decryptor = aesAlg.CreateDecryptor())
        using (CryptoStream cryptoStream = new CryptoStream(fsOutput, decryptor, CryptoStreamMode.Write))
        {
            // Skip the IV at the beginning of the file
            fsInput.Seek(iv.Length, SeekOrigin.Begin);
            fsInput.CopyTo(cryptoStream);
        }
    }
}

static void Main()
{
    string inputFile = "c:\\\\Image\\\\MyImage.jpg";
    string encryptedFile = "c:\\\\Image\\\\encrypted.jpg";
    string decryptedFile = "c:\\\\Image\\\\decrypted.jpg";

    // Generate a random IV for each encryption operation
    byte[] iv;
    using (Aes aesAlg = Aes.Create())
    {
        iv = aesAlg.IV;
    }

    string key = "1234567890123456";

    EncryptFile(inputFile, encryptedFile, key, iv);
    DecryptFile(encryptedFile, decryptedFile, key, iv);

    Console.WriteLine("Encryption and decryption completed successfully.");
    Console.WriteLine("go to c:\\\\Image folder to see the results");
    Console.ReadKey();
}

```

This is the entry point of the program. Here's what each part does:

1. File Paths and Key: Defines file paths for the input image (`MyImage.jpg`), the encrypted image (`encrypted.jpg`), and the decrypted image (`decrypted.jpg`). It also

defines a key ("1234567890123456"). Note that in real-world scenarios, using a hardcoded key is not secure, and key management should be handled more securely.

2. Initialization Vector (IV) Generation: Creates an instance of Aes to generate a random IV for each encryption operation. The IV is stored in the `iv` variable.
3. Encryption and Decryption Calls: Calls the `EncryptFile` method to encrypt the input image and the `DecryptFile` method to decrypt the encrypted image. Both methods receive the input file, output file, key, and IV as parameters.
4. Console Output and User Interaction: Prints messages indicating the success of encryption and decryption. It prompts the user to press any key before exiting, allowing them to view the results.

EncryptFile Method

```
static void EncryptFile(string inputFile, string outputFile, string key, byte[] iv)
{
    using (Aes aesAlg = Aes.Create())
    {
        aesAlg.Key = System.Text.Encoding.UTF8.GetBytes(key);
        aesAlg.IV = iv;

        using (FileStream fsInput = new FileStream(inputFile, FileMode.Open))
        using (FileStream fsOutput = new FileStream(outputFile, FileMode.Create))
        using (ICryptoTransform encryptor = aesAlg.CreateEncryptor())
        using (CryptoStream cryptoStream = new CryptoStream(fsOutput, encryptor,
CryptoStreamMode.Write))
        {
            // Write the IV to the beginning of the file
            fsOutput.Write(iv, 0, iv.Length);
            fsInput.CopyTo(cryptoStream);
        }
    }
}
```

This method performs file encryption using the AES algorithm:

1. AES Initialization: Creates an instance of `Aes` and sets its key and IV based on the provided parameters.
2. File Streams and CryptoStream: Creates file streams for input and output files. Also, creates a `CryptoStream` to perform the encryption. The encrypted data is written to `fsOutput` through the `CryptoStream`.
3. Write IV to Output File: Writes the IV to the beginning of the output file before writing the actual encrypted content.

DecryptFile Method

```
static void DecryptFile(string inputFile, string outputFile, string key, byte[] iv)
{
    using (Aes aesAlg = Aes.Create())
    {
        aesAlg.Key = System.Text.Encoding.UTF8.GetBytes(key);
        aesAlg.IV = iv;

        using (FileStream fsInput = new FileStream(inputFile, FileMode.Open))
        using (FileStream fsOutput = new FileStream(outputFile, FileMode.Create))
        using (ICryptoTransform decryptor = aesAlg.CreateDecryptor())
        using (CryptoStream cryptoStream = new CryptoStream(fsOutput, decryptor,
CryptoStreamMode.Write))
        {
            // Skip the IV at the beginning of the file
            fsInput.Seek(iv.Length, SeekOrigin.Begin);
            fsInput.CopyTo(cryptoStream);
        }
    }
}
```

This method performs file decryption using the AES algorithm:

1. AES Initialization: Creates an instance of `Aes` and sets its key and IV based on the provided parameters.

- File Streams and CryptoStream: Creates file streams for input and output files. Also, creates a `CryptoStream` to perform the decryption. The decrypted data is written to `fsOutput` through the `CryptoStream`.
- Skip IV during Decryption: Skips the IV bytes at the beginning of the input file before starting the decryption process.

Overall, this code demonstrates a simple file encryption and decryption process using the AES algorithm in C#. It includes the generation of a random IV for each encryption operation and writes the IV to the output file for later use during decryption.

What is Operator Overloading

ال operator overloading زی علامه اجمع یمکن استخدامها اکثر من استخدا ها فلو حطیتها بین رقمین هتجمعهم ولو حطیتها بین نصین هترصم جنب بعض

What is Operator Overloading?

- Operator overloading in C# allows you to define how operators behave for instances of your own classes.
- This means you can customize the behavior of operators such as `+`, `-`, `*`, `/`, `==`, `!=`, and others for your own types.

Example:

```
int x=10,y=20;
string s1 = "Mohammed", s2 = "Abu-Hadhoud";

//The '+' operator is used for normal addition
int z = x + y;

//The '+' operator is used for concatenation
string s3 = s1 + ' ' + s2;
```

Can I overload all operators?

- No, while you can overload many operators in C#, there are some that you cannot overload, and others have specific restrictions or requirements.

Operator overloading in C# allows you to define how operators behave for instances of your own classes. This means you can customize the behavior of operators such as +, -, *, /, ==, !=, and others for your own types.

Normal Addition vs. Concatenation:

The behavior of the + operator depends on the context and the types involved. For numeric types, it performs addition, while for string types, it performs concatenation. However, with operator overloading, you can define custom behaviors for the + operator based on the types you are working with.

In this example, we are demonstrating the use of the + operator for both normal addition and string concatenation.

```
using System;

public class Program
{
    static void Main(string[] args)
    {

        int x=10,y=20;
        string s1 = "Mohammed", s2 = "Abu-Hadhoud";

        //The '+' operator is used for normal addition
```

```

    int z = x + y;

    //The '+' operator is used for concatenation
    string s3 = s1 + ' ' + s2;

    Console.WriteLine($"The '+' operator is used for normal addition resulting z = {z}");
    Console.WriteLine($"The '+' operator is used for concatenation resulting s3 = {s3}");

    Console.ReadKey();
}

}

```

Explanation:

1. You have two integers `x` and `y` representing numeric values, and you use the `+` operator for normal addition, calculating the sum and storing it in the variable `z`.
2. You have two strings `s1` and `s2` representing names. You use the `+` operator for string concatenation, combining the two strings along with a space in the middle, and storing the result in the variable `s3`.
3. You use `Console.WriteLine` to print the results to the console, indicating whether the `+` operator is used for normal addition or concatenation.
4. Finally, `Console.ReadKey()` is used to wait for a key press before the console window is closed.

When you run this program, it will output:

```

The '+' operator is used for normal addition resulting z = 30
The '+' operator is used for concatenation resulting s3 = Mohammed Abu -Hadhoud

```

This demonstrates the dual nature of the `+` operator in C#, where its behavior depends on the types of the operands. For numeric types, it performs addition, and for string types, it performs concatenation.

Operator Overloading Example

عشان تعمل operator overloading بتيجي بعد اسم ال method وبتكتب كلمة overload وبعدها العلامة اللي عايز تعملها

يعني بص المثال ده

هنعمل كلاس بيخزن نقطة مكونه من محوريين x و y

```
class Point
{
    public int X { get; set; }
    public int Y { get; set; }

    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }
}
```

دلوقي انا عايز لما اجي اجمع اتنين objects من نفس الكلاس ده يأخذ اول متغير من اول object ويجمعه مع اول متغير من ثاني object
ويأخذ ثاني متغير من اول object ويجمعه مع ثاني متغير من ثاني object
ك ده بمجرد ماحط object1+object2

```
public static Point operator +(Point p1, Point p2)
{
    return new Point(p1.X + p2.X, p1.Y + p2.Y);
}
```

بص بقى هستخدمها ازاي

```
Point point3 = point1 + point2;
```

ونفس الكلام بالنسبة لعلامة الطرح وال = != بتعمل method بالكود اللي عايز تنفذه لو تم استدعاء علامه من العلامات دي

و فيه ملحوظه انك لو جيت تعمل ال == لازم معاها تعمل ال !=

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;

class NullableExample
{
    class Point
    {
        public int X { get; set; }
```

```
public int Y { get; set; }

public Point(int x, int y)
{
    X = x;
    Y = y;
}
public static Point operator +(Point p1, Point p2)
{
    return new Point(p1.X + p2.X, p1.Y + p2.Y);
}

public static Point operator -(Point p1, Point p2)
{
    return new Point(p1.X - p2.X, p1.Y - p2.Y);
}

// Overloading the == operator for fraction equality
public static bool operator ==(Point p1, Point p2)
{
    return (p1.X == p2.X) && (p1.Y == p2.Y);
}

// Overloading the != operator for fraction equality
public static bool operator !=(Point p1, Point p2)
{
    return (p1.X != p2.X) || (p1.Y != p2.Y);
}

// Overriding ToString for better readability
public override string ToString()
{
    return $"({X}, {Y})";
}

static void Main()
{
    Point point1 = new Point(1, 2);
    Point point2 = new Point(3, 4);

    // Using the overloaded + operator for point addition
    Point point3 = point1 + point2;

    // Using the overloaded + operator for point addition
    Point point4 = point1 - point2;

    Console.WriteLine($"Point1 : {point1.ToString()}");
    Console.WriteLine($"Point2 : {point2.ToString()}");
    Console.WriteLine($"Point3 is the result of point1 + point2: {point3.ToString()}");
    Console.WriteLine($"Point4 is the result of point1 - point2: {point4.ToString()}");

    // Using the overloaded == operator for point equality
    if (point1 == point2)
        Console.WriteLine("Using == : Yes, Point1 = Point2");
    else
        Console.WriteLine("Using == : No, Point1 does not equal Point2");

    // Using the overloaded != operator for point inequality
}
```

```

if (point1 != point2)
    Console.WriteLine("Using != : Yes, Point1 does not equal Point2");
else
    Console.WriteLine("Using != : No, Point1 = Point2");

Console.ReadKey();
}
}

```

This code defines a `Point` class to represent 2D points and demonstrates operator overloading in C# by overloading the `+`, `-`, `==`, and `!=` operators.

```

using System;

class Point
{
    public int X { get; set; }
    public int Y { get; set; }

    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }

    // Overloading the + operator for point addition
    public static Point operator +(Point p1, Point p2)
    {
        return new Point(p1.X + p2.X, p1.Y + p2.Y);
    }

    public static Point operator -(Point p1, Point p2)
    {
        return new Point(p1.X - p2.X, p1.Y - p2.Y);
    }

    // Overloading the == operator for fraction equality
    public static bool operator == (Point p1, Point p2)
    {
        return (p1.X == p2.X) && (p1.Y == p2.Y);
    }
}

```

```
// Overloading the != operator for fraction equality
public static bool operator !=(Point p1, Point p2)
{
    return (p1.X != p2.X) || (p1.Y != p2.Y);
}

// Overriding ToString for better readability
public override string ToString()
{
    return $"({X}, {Y})";
}

}

class Program
{
    static void Main()
    {
        Point point1 = new Point(1, 2);
        Point point2 = new Point(3, 4);

        // Using the overloaded + operator for point addition
        Point point3 = point1 + point2;

        // Using the overloaded + operator for point addition
        Point point4 = point1 - point2;

        Console.WriteLine($"Point1 : {point1.ToString()}");
        Console.WriteLine($"Point2 : {point2.ToString()}");
        Console.WriteLine($"Point3 is the result of point1 + point2: {point3.ToString()}");
        Console.WriteLine($"Point4 is the result of point1 - point2: {point4.ToString()}");

        // Using the overloaded == operator for point equality
        if (point1 == point2)
            Console.WriteLine("Using == : Yes, Point1 = Point2");
        else
            Console.WriteLine("Using == : No, Point1 does not equal Point2");

        // Using the overloaded != operator for point inequality
        if (point1 != point2)
```

```

        Console.WriteLine("Using != : Yes, Point1 does not equal Point2");
    else
        Console.WriteLine("Using != : No, Point1 = Point2");

    Console.ReadKey();
}

}

```

Here's an explanation of the code:

Point Class:

- The `Point` class has two properties, `X` and `Y`, representing the coordinates of a 2D point.
- The `+` operator is overloaded for point addition.
- The `-` operator is overloaded for point subtraction.
- The `==` operator is overloaded for point equality.
- The `!=` operator is overloaded for point inequality.
- The `ToString` method is overridden to provide a custom string representation of the point.

Program Class:

- In the `Main` method, two `Point` objects (`point1` and `point2`) are created with different coordinates.
- The overloaded `+` operator is used to add `point1` and `point2`, and the result is stored in `point3`.
- The overloaded `-` operator is used to subtract `point2` from `point1`, and the result is stored in `point4`.
- The program then prints the original points and the results of the addition and subtraction.
- Finally, the overloaded `==` and `!=` operators are used to check for equality and inequality between `point1` and `point2`, and the results are printed accordingly.

When you run this program, it will output the coordinates of points, the results of point addition and subtraction, and whether the points are equal or not.

Note: you can apply the same concept for most of other operators.

Also note that not all operators can be overloaded in C#, we will discuss this in the next lesson.

Quiz 1

What is the purpose of operator overloading in C#?

To define the behavior of operators for custom classes

To modify the behavior of built-in operators

To disable certain operators

To create new operators

How does the + operator behave for numeric types?

It performs addition

It performs concatenation

It performs subtraction

It performs multiplication

How does the + operator behave for string types?

It performs addition

It performs concatenation

It performs subtraction

It performs multiplication

Can I overload all operators in C#?

هنا بيقولك انك ماتقدرش تعمل overloading لكل ال operators وبيعرفك ايه اللي ينفع وايه اللي ماينفعش وایه اللي عليه قيود

دول ماينفعش تعملهم overloading

- Operators You Cannot Overload:

- o `&&` (conditional AND)
- o `||` (conditional OR)
- o `? :` (conditional)
- o `sizeof` (size of)
- o `typeof` (type of)
- o `->` (member access)
- o `.` (member access)
- o `checked` and `unchecked`

دول ينفع بس بشرط

١ - علامة يساوي تنفع تعملها overloading بس بشرط ان تكون نتيجة ال method انك تعمل object من الكلاس اللي انت فيه بحيث انك مثلا بدل ماتقول
`clsMyclass myObject=new clsMyClass(42);`

لاتقدر تعمل كده

`clsMyclass myObject=42;`

٢- العلامه بتاعت ال short hand if overloading بشرط ان ال data type تقدر تعملها النتائج تكون واحده

٣- علامة الجمع او الطرح المفردة اللي هما (+ و -) ماتقدرش تستخدمهم مع ال bool وال byte وال long وال ulong وال int وال uint وال short وال ushort وال sbyte

٤- العلامات اللي هيا += و -= والكلام ده بيتعملهم overloading تلقائيا لما بيتعملهم overloading للنظائر بتاعتھم يعني لما تيجي تعمل مثلا لعلامة الجمع هتلقي overloading انه ال += اتعملها هيا كمان method وھتسخدم نفس ال overloading بتاعت علامة الجمع

٥- العلامات اللي بترجلك قيمه true او false لو عملت overloading لعلامه لازم تعمل overload للعلامه المقابل ليها يعني لو جيت تعمل overload لعلامه اكتر من لازم تعمل لعلامة اصغر من ولو عملت لعلامة == لازم تعمل لعلامه لايساوي !=

٦- علامة ال array لازم نتيجتها تكون عنصر من عناصر ال

- Operators with Restrictions:

- = (assignment) can be overloaded only by defining a user-defined conversion to the type of the left-hand operand.
- ? : (conditional) can be overloaded only if the types of the second and third operands are the same.

- Unary Operators Restrictions:

- + (unary plus) and - (unary minus) cannot be overloaded for bool, byte, sbyte, ushort, short, uint, int, ulong, or long.

- Binary Operators Restrictions:

- The assignment operators (+=, -=, *=, /=, and so on) can be overloaded, but their behavior is determined by the behavior of the corresponding binary operator.

- Equality and Comparison Operators Restrictions:

- The == and != operators must be overloaded together. The same applies to <, >, <=, and >= when overloading.

- Indexing Operator Restrictions:

- The `[]` operator can only be overloaded if the containing type is an array or an indexer property.
- Conversion Operators Restrictions:
 - `implicit` and `explicit` conversion operators can be overloaded with certain restrictions. They must involve user-defined types and follow specific rules.

In summary, while you can overload many operators in C#, there are some that you cannot overload, and others have specific restrictions or requirements. Always refer to the C# language specification for the most accurate and up-to-date information regarding operator overloading.

Quiz 2

Can you overload all operators in C#?

Yes, you can overload all operators

No, you cannot overload all operators

Which operators cannot be overloaded in C#?

`&&` (conditional AND) and `||` (conditional OR)

`:?` (conditional) and `sizeof` (size of)

`->` (member access) and `.` (member access)

All of the above

Which operators have restrictions on their overloading in C#?

= (assignment)

? : (conditional)

+ (unary plus) and - (unary minus)

All of the above

Which operators must be overloaded together in C#?

== and !=

<, >, <=, and >=

= and ==

? : and sizeof

Can the [] operator be overloaded in C#?

Yes, it can be overloaded for any type

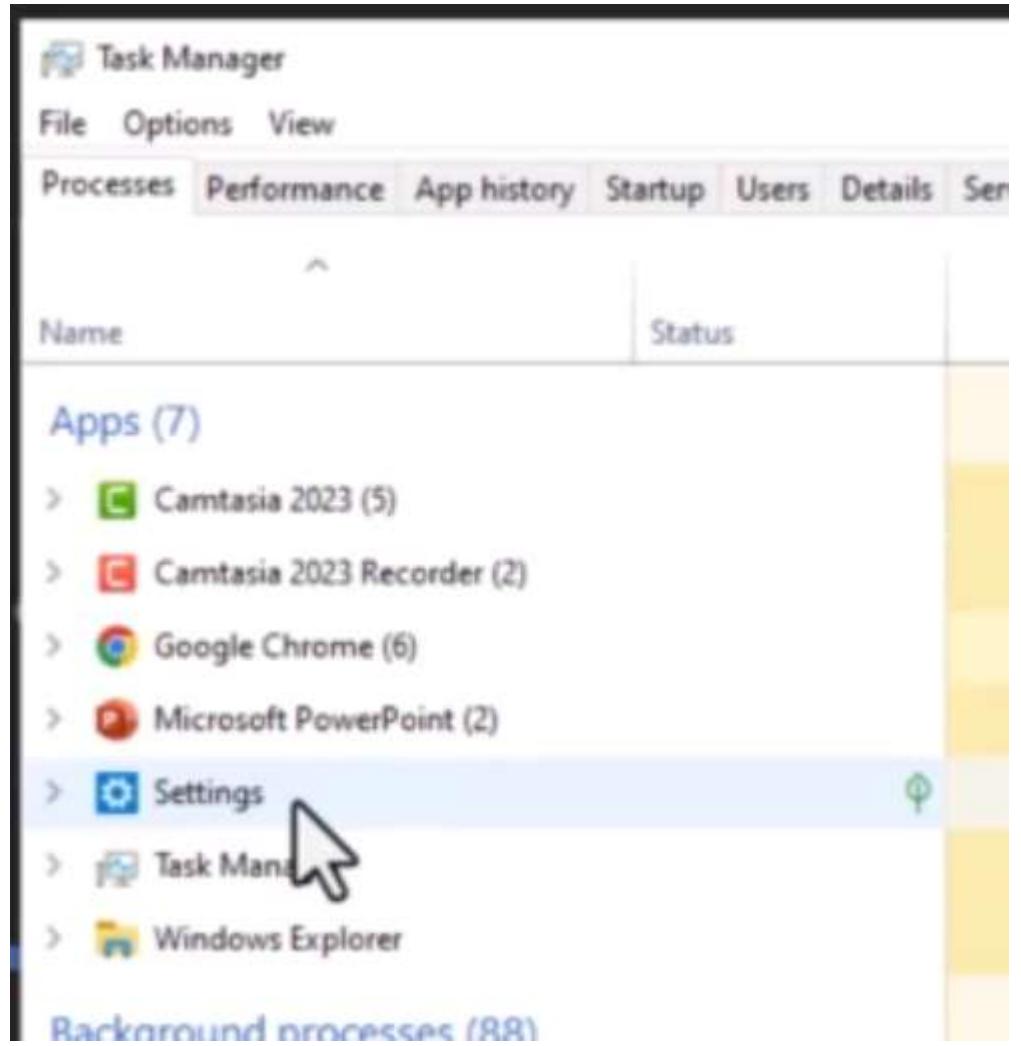
No, it cannot be overloaded

Only if the containing type is a dictionary

Only if the containing type is an array or an indexer property

Process vs Thread

ال process هيا مكان في ال memory بيتم استخدامه عشان تشغله فيه البرنامج بتاعك زي ال task manager اللي في ال process



ال process بتكون مستقله بذاتها ومسؤوله عن البرنامج بتاعاك لو واحدة باذت التانيه اللي بتشغله
برنامجه تاني بيتفضل شغاله

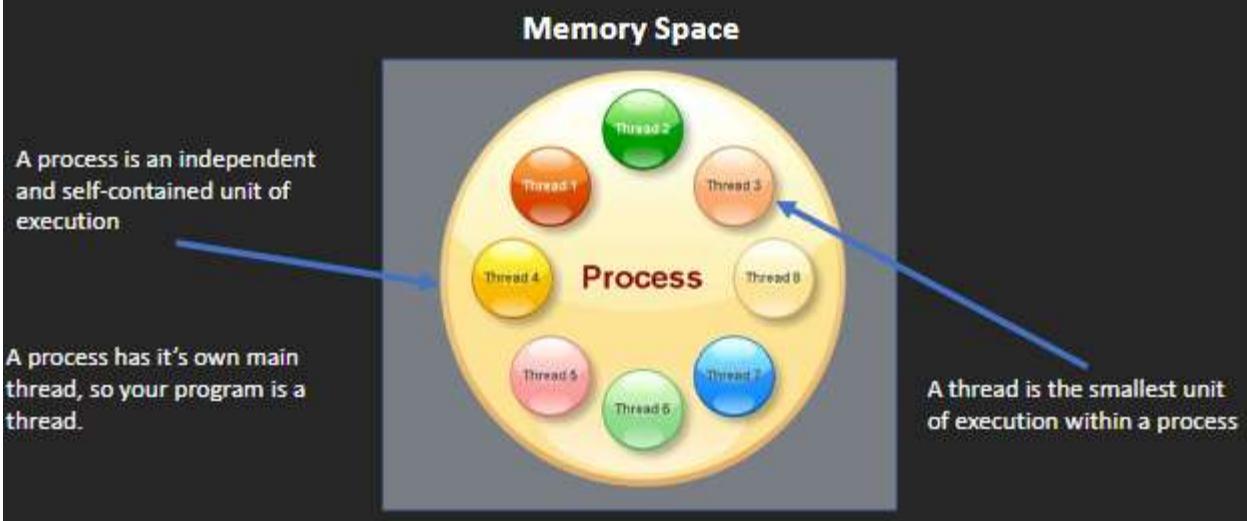
ال process دي بت تكون من threads وب يكون ليها main thread واحد اللي هوا ال main thread كل البرامج اللي عملناها بتسخدم واحد اللي هوا ال main thread

بتقدر تعمل اكتر من thread وبتشغلهم مع بعض وده ال parallel multi threading او ال parallel programming

ال threads بتاعت البرنامج الواحد بنتشارك مع نفس المساحه المخصصه لل process

ال threads بتاعت البرنامج الواحد بتقدر تتواصل مع بعضها اعتبارها زي ال functions

Process vs Thread?



Process vs Thread – Definition

- **Process:** A process is an independent and self-contained unit of execution in a computer system. It is an instance of a running program that includes its own memory space, resources, and system state.
- **Thread:** A thread is the smallest unit of execution within a process. It shares the same resources and memory space as other threads within the same process.

Process vs Thread – Memory Space

- **Process:** Each process has its own separate memory space. Processes do not share memory with other processes by default.
- **Thread:** Threads within the same process share the same memory space. They can directly access the memory of other threads within the same process.

Process vs Thread – Resource Allocation

- **Process:** Processes are allocated system resources, including CPU time, memory, file handles, and more. Each process operates independently of other processes.
- **Thread:** Threads within a process share the same resources. They can efficiently communicate with each other through shared memory.

Process vs Thread – Communication

- **Process:** Inter-process communication (IPC) mechanisms, such as message passing or shared memory, are required for processes to communicate with each other.
- **Thread:** Threads within the same process can communicate directly through shared variables and data structures. No special mechanisms are required for communication.

Process vs Thread – Isolation

- **Process:** Processes are isolated from each other, meaning that the failure of one process does not affect the execution of other processes.
- **Thread:** Threads within the same process are not fully isolated. The failure of one thread can potentially affect the entire process.

Process vs Thread – Overhead

- **Process:** Processes have a higher overhead compared to threads due to the isolation and resource allocation.
- **Thread:** Threads have lower overhead compared to processes because they share resources and memory.

Process vs Thread – Creation and Termination

- **Process:** Creating and terminating processes is generally more time-consuming than creating and terminating threads.
- **Thread:** Creating and terminating threads is generally faster than creating and terminating processes.

Process vs Thread

Process:

- Definition:
 - A process is an independent and self-contained unit of execution in a computer system. It is an instance of a running program that includes its own memory space, resources, and system state.
- Memory Space:
 - Each process has its own separate memory space. Processes do not share memory with other processes by default.
- Resource Allocation:
 - Processes are allocated system resources, including CPU time, memory, file handles, and more. Each process operates independently of other processes.

- Communication:
 - Inter-process communication (IPC) mechanisms, such as message passing or shared memory, are required for processes to communicate with each other.
- Isolation:
 - Processes are isolated from each other, meaning that the failure of one process does not affect the execution of other processes.
- Overhead:
 - Processes have a higher overhead compared to threads due to the isolation and resource allocation.
- Creation and Termination:
 - Creating and terminating processes is generally more time-consuming than creating and terminating threads.
- Process typically has a main thread. The main thread is the initial thread that is created when a program starts. It is the thread responsible for executing the main entry point of the program, such as the main function in C or C++.

Thread:

- Definition:
 - A thread is the smallest unit of execution within a process. It shares the same resources and memory space as other threads within the same process.
- Memory Space:
 - Threads within the same process share the same memory space. They can directly access the memory of other threads within the same process.
- Resource Allocation:
 - Threads within a process share the same resources. They can efficiently communicate with each other through shared memory.
- Communication:
 - Threads within the same process can communicate directly through shared variables and data structures. No special mechanisms are required for communication.

- Isolation:
 - Threads within the same process are not fully isolated. The failure of one thread can potentially affect the entire process.
- Overhead:
 - Threads have lower overhead compared to processes because they share resources and memory.
- Creation and Termination:
 - Creating and terminating threads is generally faster than creating and terminating processes.

Summary:

- A process is an independent program with its own memory space, resources, and system state.
- A thread is the smallest unit of execution within a process, sharing the same resources and memory space with other threads in the same process.
- Processes are isolated, and communication between them requires special mechanisms.
- Threads within the same process can communicate directly through shared memory.
- Processes have higher overhead due to their isolation and separate resource allocation.
- Threads have lower overhead as they share resources within the same process.
- Processes are more robust in the face of failures since one process's failure does not affect others.
- Threads within a process are less isolated, and the failure of one thread can potentially affect the entire process.

Quiz 1

What is a process?

A unit of execution within a computer system

A unit of memory in a computer system

A unit of communication between programs

A process typically has a main thread. The main thread is the initial thread that is created when a program starts. It is the thread responsible for executing the main entry point of the program, such as the main function in C or C++.

True

False

A thread is the smallest unit of execution within a process. It shares the same resources and memory space as other threads within the same process.

True

False

What is the memory space of a process?

Processes do not have memory spaces

Processes share the same memory space

Each process has its own separate memory space

Processes have limited memory spaces

Threads within the same process share the same memory space. They can directly access the memory of other threads within the same process.

True

False

Which has higher overhead: processes or threads?

Processes

Threads

Processes and threads have the same overhead

Overhead depends on the specific system

Threads have lower overhead compared to processes because they share resources and memory.

True

False

The failure of one process does not affect the execution of other processes.

True

False

Threads within the same process are not fully isolated. The failure of one thread can potentially affect the entire process.

True

False

Inter-process communication (IPC) mechanisms, such as message passing or shared memory, are required for processes to communicate with each other.

True

False

Threads within the same process can communicate directly through shared variables and data structures. No special mechanisms are required for communication.

True

False

Processes are allocated system resources, including CPU time, memory, file handles, and more. Each process operates independently of other processes.

True

False

Creating and terminating threads is generally slower than creating and terminating processes.

True

False, they are faster

Each process has its own separate memory space. Processes do not share memory with other processes by default.

True

False

Threads within the same process share the same memory space. They can directly access the memory of other threads within the same process.

True

False

Creating and terminating processes is generally more time-consuming than creating and terminating threads.

True

False

Creating and terminating threads is generally faster than creating and terminating processes.

True

False

Threads within the same process share the same memory space. They can directly access the memory of other threads within the same process.

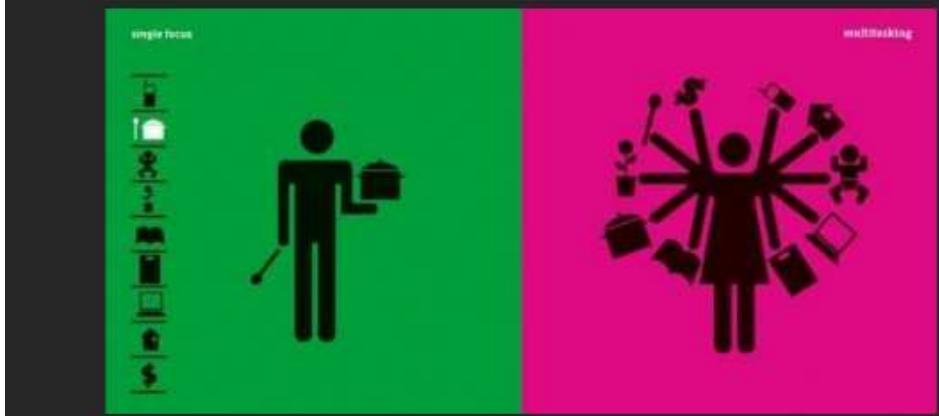
True

False

What is Multithreading?

الـ multithreading هو انك تشغّل اكتر من thread مع بعض وده بيكون اسرع بس لازم تكون فاهم انت بتعمل ايه

What is Multithreading?



What is Multithreading?

- Multithreading in C# refers to the concurrent execution of multiple threads within the same application.
- A thread is the smallest unit of execution in a process.
- The program itself is a thread (Main Thread).
- Multithreading allows you to perform multiple tasks simultaneously, improving performance and responsiveness in certain scenarios.
- Threading in C# allows you to create and manage threads to execute multiple operations concurrently.

What is Multithreading?

- Multithreading is a powerful technique that can significantly improve the performance of applications that can benefit from concurrent execution of tasks.
- However, it also introduces challenges such as synchronization and coordination between threads.
- Therefore, careful design and understanding of multithreading concepts are essential for creating robust multithreaded applications in C#.

Example of multithreaded Applications.

- Windows
- Chatting Applications
- Games
- Webservices
- And Many others

Multithreading in C# refers to the concurrent execution of multiple threads within the same application. A thread is the smallest unit of execution in a process, and multithreading allows you to perform multiple tasks simultaneously, improving performance and responsiveness in certain scenarios.

Threading in C# allows you to create and manage threads to execute multiple operations concurrently.

Multithreading is a powerful technique that can significantly improve the performance of applications that can benefit from concurrent execution of tasks. However, it also introduces challenges such as synchronization and coordination between threads. Therefore, careful design and understanding of multithreading concepts are essential for creating robust multithreaded applications in C#.

Quiz 2

What is a thread in C#?

The smallest unit of execution in a process

A collection of tasks in C#

A programming language in C#

What is multithreading in C#?

Executing multiple threads within an application

Executing multiple processes within an application

Executing a single thread within an application

What are some challenges introduced by multithreading in C#?

Synchronization and coordination between threads

Improved performance and responsiveness

Easier development process

Multithreading gives us an Improved performance and responsiveness.

True

False

What is Thread Class?

لو عمل عمليات على ال mainthread يكفي انك تكتب اسم الكلاس و تستدعي ال method اللي انت عايزها
زي كده

ال sleep thread بتخلي ال ينام لفتره وبعدين يقوم بعمل شغل

```
static void Main()
{
    for (int i = 0; i <= 5; i++)
    {
        Console.WriteLine("Main Thread:" + i);
        Thread.Sleep(500);
    }
    Console.ReadKey();
}
```

طيب لو انا هعمل thread خاصه بيا ؟

عمل object من الكلاس thread وال constructor بتاعه بيطلب delegate فبديله ال اللي عايز اشغلها اثناء مال main thread شغال وبقوله start عشان يشتغل

ملحوظه هنا ال threads كلها بتشتغل مع بعضها وانت ماتضمنش مين يشتغل قبل مين لانه ممكن ال thread racing اللي انت عاملها تسبق ال main thread وده اسمه

فيه حاجه كمان وهيا ال method اللي اسمها join ودي بتخلي ال main thread يوقف لحد ماال يخلص شغله thread

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;
using System.Threading;

class NullableExample
{
    static void Method1()
    {
        for (int i=0;i<=5;i++) {
            Console.WriteLine("Thread Method1:" + i);
            Thread.Sleep(500);
        }
    }

    static void Main()
    {
        Thread t= new Thread(Method1);
        t.Start();

        for (int i = 0; i <= 5; i++)
        {
            Console.WriteLine("Main Thread:" + i);
            Thread.Sleep(500);
        }

        Console.ReadKey();
    }
}
```

Thread Class:

The `Thread` class in the `System.Threading` namespace is a fundamental building block for creating and managing threads in C#.

```
class Program
{
    static void Main()
    {
```

```

// Note that your program is the main thread.

// Create a new thread and start it
Thread t = new Thread(ThreadMethod1);
t.Start();

// Main thread continues its execution
for (int i = 1; i <= 10; i++)
{
    Console.WriteLine("Main Thread: " + i);
    Thread.Sleep(1000); // Sleep for 1 second
}
Console.ReadKey();
}

static void ThreadMethod1()
{
    for (int i = 1; i <= 5; i++)
    {
        Console.WriteLine("Thread Method1: " + i);
        Thread.Sleep(1000);
    }
}

```

Explanation:

- **Main Method:**
 - The `Main` method is the entry point of the program and represents the main thread.
 - A new thread (`t`) is created by instantiating the `Thread` class, and the method `ThreadMethod1` is specified as the method to be executed by this thread.

- The new thread (`t`) is started with the `Start` method.
- Main Thread Execution:
 - The main thread then enters a loop that runs for `10` iterations.
 - In each iteration, it prints a message to the console indicating the current iteration number along with the text "Main Thread."
 - After printing the message, the main thread sleeps for `1000` milliseconds (1 second) using `Thread.Sleep(1000)`.
- Thread `t` Execution (`ThreadMethod1`):
 - Meanwhile, the new thread (`t`) executes the `ThreadMethod1`.
 - `ThreadMethod1` runs in its own thread and enters a loop that runs for `5` iterations.
 - In each iteration, it prints a message to the console indicating the current iteration number along with the text "Thread Method1."
 - After printing the message, the thread (`t`) sleeps for `1000` milliseconds (1 second) using `Thread.Sleep(1000)`.
- Console Output:
 - The output in the console will be a mix of messages from the main thread and the `ThreadMethod1` thread, indicating concurrent execution.
 - The main thread prints messages while the `ThreadMethod1` thread is also printing its messages.
- `Console.ReadKey()`:
 - The `Console.ReadKey()` at the end is used to prevent the console window from closing immediately, allowing you to observe the output.

In summary, this code demonstrates the concurrent execution of the main thread and an additional thread (`ThreadMethod1`). Both threads print messages to the console in a loop, and the `Thread.Sleep` calls introduce delays to observe the interleaved execution of the two threads. The program will wait for a key press before exiting the console window.

Quiz 3

Which namespace contains the Thread class in C#?

System.Threading

System.Thread

System.Concurrent

System.Parallel

What is the entry point of a C# program?

Main

EntryPoint

ThreadMethod

Start

What is the method executed by the new thread (t) in the program?

Method1

Main

Thread.Sleep

Console.WriteLine

.join() method will wait for thread to finish its execution then the main thread will continue execution.

True

False

Parameterized Thread

احنا دلوقتي عايزة نبت thread لـ parameters هنا بيقولك انه فيه طريقة تخلی ال start تأخذ parameters بس الاسهل انك تستعمل ال lambda في انك تبعت parameters

الطريقه هيا انك تعمل thread method تشغل ال thread الخاصه بال method

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;
using System.Threading;

class NullableExample
{
    static void Method1(string Text) {
        for (int i=0;i<=5;i++) {
            Console.WriteLine("Thread Method1:" +Text + i);
            Thread.Sleep(500);
        }
    }

    static void Main()
    {
        Thread t= new Thread(()=>Method1("Thread1"));
        t.Start();

        for (int i = 0; i <= 5; i++)
        {
            Console.WriteLine("Main Thread:" + i);
            Thread.Sleep(500);
        }

        Console.ReadKey();
    }
}
```

Download 3 Web Pages using Multi Threading

نهعمل 3 tasks كل واحد فيهم بينزل صحفه من الانترنت وكلهم بيشتغلوا مع بعض

الكود مش محتاج شرح

```

using System;
using System.IO;
using System.Net;
using System.Security.Cryptography;
using System.Text;
using System.Threading;

class NullableExample
{
    static void DownloadAndPrint(string url) {
        string Content;

        using (WebClient client=new WebClient()) {
            Thread.Sleep(100);
            Content = client.DownloadString(url);
        }
        Console.WriteLine($"{url}: {Content.Length} Characters Downloaded");
    }

    static void Main()
    {
        Console.WriteLine("Starting threads...");

        Thread t1= new Thread(()=>DownloadAndPrint("https://www.cnn.com"));
        t1.Start();
        Console.WriteLine("Thread 1 started...");

        Thread t2 = new Thread(() => DownloadAndPrint("https://www.amazon.com"));
        t2.Start();
        Console.WriteLine("Thread 2 started...");

        Thread t3 = new Thread(() => DownloadAndPrint("https://www.ProgrammingAdvises.com"));
        t3.Start();
        Console.WriteLine("Thread 3 started...");

        t1.Join();
        t2.Join();
        t3.Join();
        Console.WriteLine("Done");
        Console.ReadKey();
    }
}

```

What is Race Condition?

زي ماقولنا ان انت مابتقدرش تحدد مين اللي بيشتغل الاول في ال racing threads وده اسمه ال عشان كده بتحاول تتجنب ال shared resources وانك تستعمل immutable objects عشان تتجنب الأخطاء

هنا بيقولوك انه فيه حل للمشكله دي وهو ال thread synchronization

What is Race Condition?



- A race condition is a situation in concurrent programming where the behavior of a program depends on the relative timing of events, such as the order in which threads are scheduled to run.
- In other words, a race condition occurs when the correctness of a program's execution depends on the unpredictable interleaving of operations from multiple threads.
- Race conditions can lead to unexpected and undesirable outcomes, including:
 - Data corruption,
 - Application crashes
 - or other forms of incorrect behavior.
- They are particularly common in multithreaded or parallel programming, where multiple threads execute concurrently and may access shared resources or variables.

What is the solution for race condition?

Thread Synchronization

What is Race Condition?

A race condition is a situation in concurrent programming where the behavior of a program depends on the relative timing of events, such as the order in which threads are scheduled to run.

In other words, a race condition occurs when the correctness of a program's execution depends on the unpredictable interleaving of operations from multiple threads.

Race conditions can lead to unexpected and undesirable outcomes, including:

- Data corruption,
- Application crashes
- or other forms of incorrect behavior.

They are particularly common in multithreaded or parallel programming, where multiple threads execute concurrently and may access shared resources or variables.

Quiz 4

What is a race condition?

A situation in concurrent programming where the behavior of a program depends on the relative timing of events

A situation in which a program executes correctly regardless of the order in which threads are scheduled to run

A condition where one thread wins a race against another thread

A condition where threads never run concurrently

What can race conditions lead to?

Data corruption

Application crashes

Incorrect behavior

All of the above

Where are race conditions most common?

In single-threaded programming

In concurrent programming

In sequential programming

In parallel programming

Introduction to Thread Synchronization

ال حل لل race condition هو عباره عن انك تعمل تنسيق لل threads اشهر واحدة فيهم lock statement ودي بتتأكد انه مفيش اكتر من thread شغالين على نفس ال object في نفس الوقت بيقف ال threads طابور زي طابور العيش اول thread تستعمل ال object بيخليها تستعمله وبيعمل lock عليها ولو جه thread تاني يشتغل عليه بيقفه في الطابور لحد مايخلص

فيه اكتر من synchronization mechanisms تانية غير ال lock statement فيه اللي بتخلي ال processes متاح ل thread mechanism بتنسق العمل بين انتين object و فيه اللي بتخلي ال threads زي الداتابيز كده واحد فقط وفيه اللي بتخليك انت تحدد عدد ال threads زي الداتابيز

What is Thread Synchronization?

- Thread synchronization is crucial when multiple threads access shared resources to prevent race conditions and ensure data consistency.
- Thread synchronization in C# refers to the coordination and control of the execution of multiple threads to ensure that they access shared resources or perform critical sections of code in a mutually exclusive and orderly manner.
- Without proper synchronization, multiple threads operating concurrently can lead to data corruption, race conditions, and unpredictable behavior.
- In the context of multithreading and concurrent programming, "mutually exclusive" refers to a situation where only one thread at a time is allowed to execute a particular section of code or access a shared resource. The idea is to ensure that certain critical operations are performed atomically, without interference from other threads.
- One common synchronization mechanism in C# is the lock statement.

Thread Synchronization:

Thread synchronization in C# refers to the coordination and control of the execution of multiple threads to ensure that they access shared resources or perform critical sections of code in a mutually exclusive and orderly manner. Without proper synchronization, multiple threads operating concurrently can lead to data corruption, race conditions, and unpredictable behavior.

In the context of multithreading and concurrent programming, "mutually exclusive" refers to a situation where only one thread at a time is allowed to execute a particular section of code or access a shared resource. The idea is to ensure that certain critical operations are performed atomically, without interference from other threads.

Thread synchronization is crucial when multiple threads access shared resources to prevent race conditions and ensure data consistency. One common synchronization mechanism in C# is the `lock` statement.

Multithreading is a powerful technique that can significantly improve the performance of applications that can benefit from concurrent execution of tasks. However, it also introduces challenges such as synchronization and coordination between threads. Therefore, careful design and understanding of multithreading concepts are essential for creating robust multithreaded applications in C#.

Synchronization Example

هنشرح مثل على ال lock mechanism واللي بتحقق من ال mutually exclusive معناها انه ال object اللي يوصله thread واحد كل مره

هنا هنعمل متغير من النوع int وه يكون static object و هنعمل static هيكون برضه

```
static int sharedCounter = 0;  
static object lockObject = new object();
```

الكود بتاعنا في ال main ماشي عادي جدا

```
static void Main()  
{  
    Thread t1 = new Thread(IncrementCounter);  
    Thread t2 = new Thread(IncrementCounter);  
  
    t1.Start();  
    t2.Start();
```

```

t1.Join();
t2.Join();

Console.WriteLine("Final Counter Value: " + sharedCounter);
Console.ReadKey();
}

```

مال هنعمل ال lock ازاي؟

فالك بتكتب lock وتفتح قوسين تحط فيهم ال object وقوسين تانين تكتب فيهم الكود اللي انت عايزه او العمليه اللي مش عايز اكتر من thread يدخل عليها

هنا ال lock ده يعتبر البواب اللي بيرس الكود من ان اكتر من حد يشتغل عليه

```

for (int i = 0; i < 100000; i++)
{
    lock (lockObject)
    {
        sharedCounter++;
    }
}

```

بس كده

وده الكود كله

```

using System;
using System.IO;
using System.Net;
using System.Security.Cryptography;
using System.Text;
using System.Threading;

class NullableExample
{
    static int sharedCounter = 0;
    static object lockObject = new object();

    static void Main()
    {
        Thread t1 = new Thread(IncrementCounter);
        Thread t2 = new Thread(IncrementCounter);

        t1.Start();
        t2.Start();

        t1.Join();
    }

    static void IncrementCounter()
    {
        lock (lockObject)
        {
            sharedCounter++;
        }
    }
}

```

```

t2.Join();

Console.WriteLine("Final Counter Value: " + sharedCounter);
Console.ReadKey();
}

static void IncrementCounter()
{
    for (int i = 0; i < 100000; i++)
    {
        lock (lockObject)
        {
            sharedCounter++;
        }
    }
}

```

Here's an example demonstrating thread synchronization using `lock`:

```

using System;
using System.Threading;

class Program
{
    static int sharedCounter = 0;
    static object lockObject = new object();

    static void Main()
    {
        // Create two threads that increment a shared counter
        Thread t1 = new Thread(IncrementCounter);
        Thread t2 = new Thread(IncrementCounter);

        t1.Start();
        t2.Start();

        // Wait for both threads to complete
        t1.Join();
        t2.Join();
    }

    static void IncrementCounter()
    {
        for (int i = 0; i < 100000; i++)
        {
            lock (lockObject)
            {
                sharedCounter++;
            }
        }
    }
}

```

```

        Console.WriteLine("Final Counter Value: " + sharedCounter);
        Console.ReadKey();
    }

static void IncrementCounter()
{
    for (int i = 0; i < 100000; i++)
    {
        // Use lock to synchronize access to the shared counter
        lock (lockObject)
        {
            sharedCounter++;
        }
    }
}

```

Explanation:

1. sharedCounter and lockObject:

- - `sharedCounter` is a shared variable that both threads will increment.
 - `lockObject` is an object used as a lock to synchronize access to the shared resource.

1. IncrementCounter Method:

- - The `IncrementCounter` method is the code that each thread will execute.
 - It contains a loop where the shared counter is incremented multiple times.
 - The `lock` statement is used to ensure that only one thread can execute the critical section (the code inside the `lock`) at a time.

1. Main Method:

- - Two threads (`t1` and `t2`) are created, both calling the `IncrementCounter` method.

- The threads are started using the `Start` method.
- The `Join` method is used to wait for both threads to complete before proceeding.

1. Console Output:

- - The final value of the shared counter is printed to the console.

In this example, without the `lock` statement, there could be a race condition where both threads try to increment `sharedCounter` simultaneously, leading to incorrect results. The `lock` statement ensures that only one thread can access the critical section at a time, preventing such race conditions and ensuring data consistency.

In the example provided earlier:

```
static int sharedCounter = 0;  
static object lockObject = new object();  
  
static void IncrementCounter()  
{  
    for (int i = 0; i < 100000; i++)  
    {  
        // Use lock to synchronize access to the shared counter  
        lock (lockObject)  
        {  
            sharedCounter++;  
        }  
    }  
}
```

Here, `lockObject` is the lock object. Let's break down its use:

- Creating the Lock Object:
 - `lockObject` is created as a simple object instance using `new object()`.

- It is a dedicated object whose sole purpose is to be used as a synchronization object.
- Synchronization with `lock`:
 - The `lock` statement ensures that only one thread at a time can enter the critical section of code (the block of code inside the `lock` statement).
 - When a thread encounters the `lock` statement, it attempts to acquire the lock on the specified lock object (`lockObject` in this case).
 - If the lock is already held by another thread, the current thread will be blocked until the lock becomes available.
- Preventing Race Conditions:
 - A race condition is a situation in concurrent programming where the behavior of a program depends on the relative timing of events, such as the order in which threads are scheduled to run. In other words, a race condition occurs when the correctness of a program's execution depends on the unpredictable interleaving of operations from multiple threads.
 - Race conditions can lead to unexpected and undesirable outcomes, including data corruption, application crashes, or other forms of incorrect behavior. They are particularly common in multithreaded or parallel programming, where multiple threads execute concurrently and may access shared resources or variables.
 - In this example, the critical section is the increment operation on the `sharedCounter`.
 - Without the `lock` statement and proper synchronization, multiple threads could attempt to increment `sharedCounter` simultaneously, leading to race conditions and incorrect results.
- Ensuring Data Consistency:
 - By using `lock` and the designated lock object, data consistency is ensured. Only one thread can be inside the critical section at any given time, preventing conflicting updates to the shared resource.
- Avoiding Deadlocks:
 - Using a dedicated lock object (such as `lockObject`) is a good practice because it helps avoid deadlocks. Deadlocks can occur when multiple threads contend for multiple locks simultaneously, leading to a situation where no thread can make progress.

In summary, the `lockObject` is a synchronization object used with the `lock` statement to ensure mutually exclusive access to a critical section of code. It plays a crucial role in preventing race conditions, ensuring data consistency, and avoiding deadlocks in multithreaded applications.

Quiz 5

What is thread synchronization?

Controlling the execution of multiple threads to ensure they access shared resources in a mutually exclusive and orderly manner

Stopping the execution of multiple threads to prevent data corruption

Running multiple threads concurrently without any coordination

Ensuring multiple threads always execute in parallel without any synchronization

Why is thread synchronization important?

To prevent data corruption, race conditions, and unpredictable behavior

To improve the performance of applications

What does "mutually exclusive" mean in the context of thread synchronization?

Only one thread at a time can execute a particular section of code or access a shared resource

Multiple threads can execute a particular section of code or access a shared resource simultaneously

Threads can execute in any order without any coordination

Only two threads can execute a particular section of code or access a shared resource

What can happen if thread synchronization is not implemented?

Data corruption, race conditions, and unpredictable behavior

Improved performance of applications

Complete isolation of threads

Thread execution failure

Which statement is a common synchronization mechanism in C#?

Lock statement

Thread statement

Sync statement

Unlock statement

What is Async Programming? how it's Different from multi threading?

هنشرح الفرق بين ال multi threading وال synchronous programming

Asynchronous programming

هنا تخيل لو عندنا 3 tasks

اول واحد بتأخذ ٨ خطوات و مدة تشغيلها ٨ ثواني

الثانية بتأخذ ٤ خطوات و مدة تشغيلها ٤ ثواني

الثالثة بتأخذ خطوتين و مدة تشغيلها ثانيتين

هنا بيقولك لو المبرمج مش فاهم هيستخدم ال synchronous

ومعناها باختصار هوا انك هتشغل البرنامج على thread واحد فيبدأ بال task الاولى يخلصها ينقل لل task الثانية يخلصها ينقل للثالثة

وبكده مدة تشغيل البرنامج ١٤ ثانية

طيب لو حد فاهم شوية ؟

هنا هيستخدم ال multi threading وها انه يشغل البرنامج على 3threads وبكده هيشغل ال 3tasks مع بعضهم

وبكدة مدة تشغيل البرنامج بتكون هيا مدة أطول task فيهم يعني ٨ ثواني طيب أي هيا ال asynchronous programming ؟

بيقولك انها تكون عامل واحد يعني thread واحد بس لما يكون فيه حاجه بتعطله ما بيناش او بيستنى لما تخلص لا ده بيروح يشوف task تانىه يخلصها

يعنى مثلا افرض انه ال task الاولاته كان من ضمن الخطوات اللي فيها كان فيه ٣ ثواني انتظار الرد من الداتا بيز

هل هينام جنبها لحد ما الرد يجيء ؟

قالك لا هيروح يشتغل في task تانىه مدة ال ٣ ثواني دي لحد مالداتابيز ترد عليه وأول ماترد عليه بيسيب اللي في ايده ويروح يكمل اول task ولما يخلص ال task بيرجع على task 2 يكمل الشغل اللي فيها وبعدين يروح على task 3

وبالتالي الوقت المطلوب هو عدد الثواني اللي اشتغلها فعلا من غير ال idle time يعني ١١ ثانية

ال sequential programming هيا synchronous

ال parallel programming هيا multi threading

ال concurrent programming هيا asynchronous

ال multi threading asynchronous بيبقى فيه حالات هوا بيكون فيها اسرع من ال

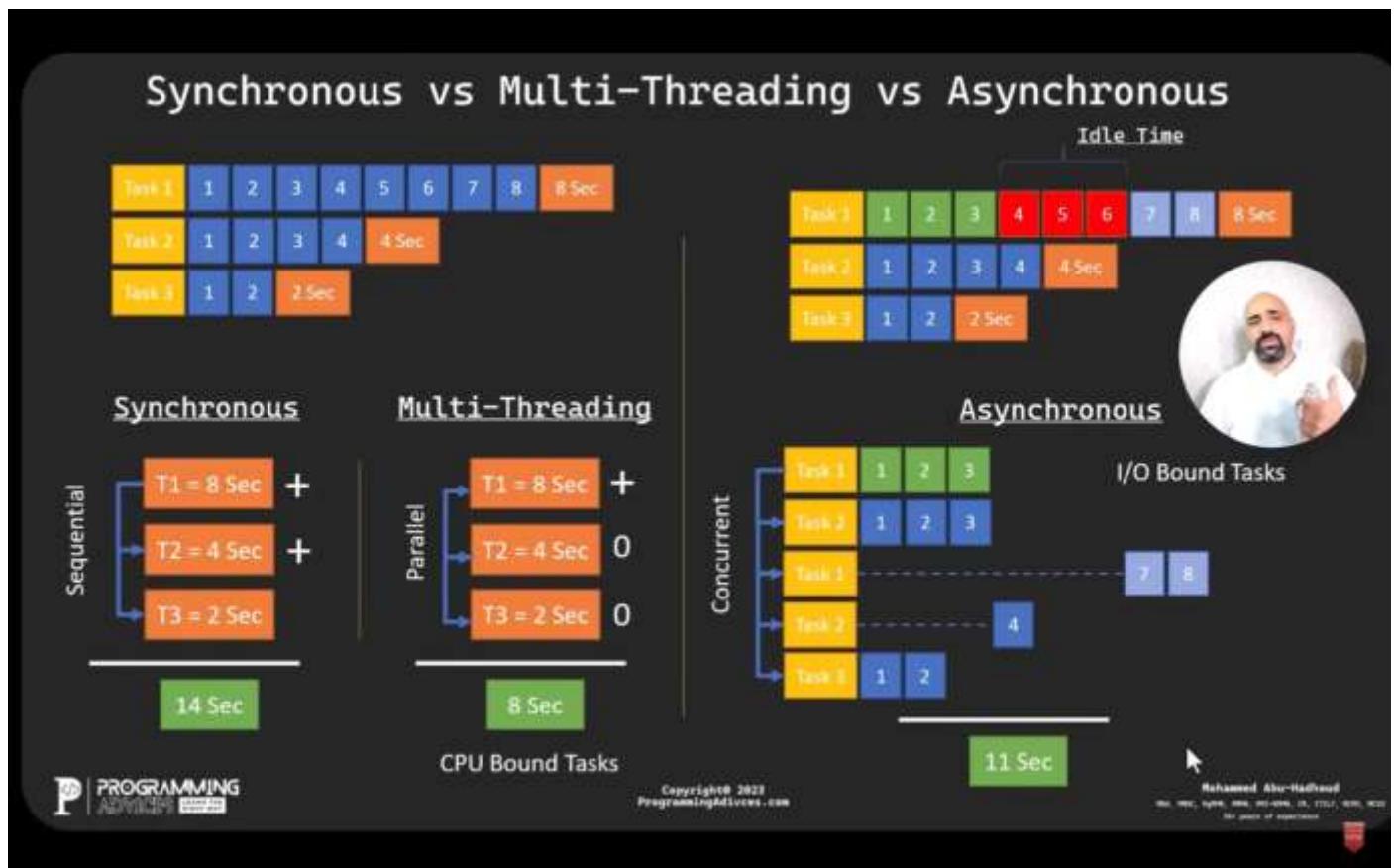
ال asynchronous cost بتعتبر تشتغل ال multithreading اعلى من ال

طيب امتى استخدم ال asynchronous وامتي استخدم ال multi threading ؟

قالك بتستخدم ال asynchronous بتسخدمها في عمليات ال I/O Bound يعني عمليات ال input وال output زي التعامل مع الملفات لانه بيكون فيها اسرع من ال multi threading لأنك لو استخدمت ال multi threading مع العمليات دي هستهلك موارد اكتر

ال multithreading اسرع في حالة ال cpu bound tasks زي العمليات الحسابية

ال asynchronous بتعرف ان الداتا بيز خلصت ؟



Synchronous Vs. multithreading Vs. Synchronous programming in various aspects:

- **Concurrency Model:**
 - Synchronous Programming: Concurrency is achieved through sequential execution. Each task is completed before moving on to the next, and the program waits for each operation to finish.
 - Multithreading: Concurrency is achieved by creating multiple threads of execution within a process. Threads can run independently, and the operating system scheduler decides when to switch between them.
 - Asynchronous Programming: Concurrency is achieved through non-blocking operations. The program can continue executing other tasks while waiting for certain operations (e.g., I/O or network requests) to complete.
- **Parallelism:**
 - Synchronous Programming: Does not inherently support parallelism. Tasks are executed sequentially.
 - Multithreading: Supports parallelism as multiple threads can execute tasks simultaneously on multiple CPU cores.

- Asynchronous Programming: Does not necessarily imply parallelism but enables efficient use of resources by allowing tasks to run concurrently without blocking the main thread.

- **Programming Model:**

- Synchronous Programming: Uses a straightforward, blocking model. Tasks are executed one after another in a linear fashion.
- Multithreading: Requires explicit creation and management of threads, often involving synchronization mechanisms like locks.
- Asynchronous Programming: Utilizes non-blocking constructs like callbacks, promises, or `async/await`. Facilitates the creation of non-blocking code, making it easier to handle multiple tasks concurrently.

- **Complexity and Safety:**

- Synchronous Programming: Typically less complex and easier to understand, but can lead to blocking and potential inefficiencies.
- Multithreading: Introduces complexities related to thread synchronization, shared data safety, and potential race conditions. Requires careful management to avoid issues like deadlocks.
- Asynchronous Programming: Can be more readable for certain tasks, especially I/O-bound operations. However, managing callbacks and ensuring proper error handling can introduce complexity.

- **Resource Overhead:**

- Synchronous Programming: Generally has lower resource overhead compared to multithreading.
- Multithreading: Can have higher resource overhead due to the creation and management of multiple threads. Synchronization mechanisms add to complexity.
- Asynchronous Programming: Tends to have lower resource overhead as it doesn't require creating and managing multiple threads.

- **Use Cases:**

- Synchronous Programming: Well-suited for simpler applications or scenarios where blocking operations do not significantly impact performance.

- Multithreading: Suitable for CPU-intensive tasks (CPU-Bound tasks) that can be parallelized, such as complex calculations or image processing.
- Asynchronous Programming: Well-suited for scenarios where I/O operations are prevalent, such as networking, file I/O, or database queries.

In summary, the choice between synchronous programming, multithreading, and asynchronous programming depends on the nature of the tasks, the specific requirements of the application, and the desired balance between simplicity, concurrency, and parallelism. In many cases, a combination of these techniques may be used to achieve optimal performance.

Quiz

Which concurrency model achieves concurrency through sequential execution?

Synchronous Programming

Multithreading

Asynchronous Programming

Which concurrency model achieves concurrency by creating multiple threads of execution within a process?

Synchronous Programming

Multithreading

Asynchronous Programming

Which concurrency model achieves concurrency through non-blocking operations?

Synchronous Programming

Multithreading

Asynchronous Programming

Which programming model uses a straightforward, blocking model?

Synchronous Programming

Multithreading

Asynchronous Programming

Which programming model requires explicit creation and management of threads?

Synchronous Programming

Multithreading

Asynchronous Programming

Which programming model utilizes non-blocking constructs like callbacks, promises, or async/await?

Synchronous Programming

Multithreading

Asynchronous Programming

Which concurrency model does not inherently support parallelism?

Synchronous Programming

Multithreading

Asynchronous Programming

Which concurrency model can execute tasks simultaneously on multiple CPU cores?

Synchronous Programming

Multithreading

Asynchronous Programming

Which concurrency model enables efficient use of resources by allowing tasks to run concurrently without blocking the main thread?

Synchronous Programming

Multithreading

Asynchronous Programming

Which concurrency model has lower resource overhead compared to multithreading?

Synchronous Programming

Multithreading

Asynchronous Programming

Which concurrency model can have higher resource overhead due to the creation and management of multiple threads?

Synchronous Programming

Multithreading

Asynchronous Programming

Which concurrency model is well-suited for scenarios where I/O operations are prevalent, such as networking, file I/O, or database queries?

Synchronous Programming

Multithreading

Asynchronous Programming

Threading introduces complexities related to:

Non-blocking code

Thread synchronization and potential race conditions

Error handling in asynchronous code

Synchronous programming is well-suited for:

I/O operations

CPU-intensive tasks

Simpler applications

Threading is more closely associated with:

Blocking operations

Parallelism

Sequential execution

Synchronous programming relies on:

Non-blocking constructs

Callbacks, promises, or async/await

Blocking operations

Asynchronous code can be more readable for:

CPU-bound operations

I/O-bound operations

Sequential tasks

Asynchronous programming is well-suited for scenarios where:

CPU-intensive tasks are prevalent

I/O operations are prevalent

Multithreading is essential

Threading involves explicit creation and management of:

Callbacks

Threads

Promises

Synchronous code tends to be less complex and easier to:

Parallelize

Understand

Multithread

Asynchronous programming generally has:

Higher resource overhead

Lower resource overhead

Similar resource overhead to multithreading

Asynchronous programming enables efficient use of resources without:

Creating multiple threads

Blocking the main thread

Parallelism

Task Class

عشان تستخدم ال async programming بتستخدم كلاس اسمه tasks وده بيكون في مكتبه اسمها TPL وهيا اختصار ل task parallel library و هنستخدم اتنين key words اسمهم async و await

كلمة async بتحطها جنب ال method عشان تضيفها للعمليات اللي عايز تضيفها لل await
كلمة await بتنفع ال thread من انه ينتقل للسطر الجاي من غير مايخلص method معينه

Task Class



- System.Threading.Tasks is a namespace in the .NET Framework (and its successor, .NET Core) that provides a set of classes and interfaces for working with asynchronous programming and parallel processing.
- The namespace is part of the Task Parallel Library (TPL) and is designed to simplify the development of parallel and asynchronous code.
- Async and Await Keywords: The `async` and `await` keywords, introduced in C# 5.0, are used to simplify asynchronous programming. They allow developers to write asynchronous code in a more natural and readable way.

طيب اول خطوة عشان تستخدم ال `async` بتنستدي المكتبه دي

```
using System.Threading.Tasks;
```

بعدين بنروح لل `main` ونكتب جنبها كلمة `async task` عشان تضمها للعمليات اللي عايز تعملها `async`

```
static async Task Main()
```

بعدين هنعمل `method` بسيطه بتتيم ال `thread` لمدة ٤ ثواني وبعدها بترجع رقم

عشان نعمل ده محتاجين حاجتين

اول حاجه نكتب كلمة `async` عشان نضيف ال `method` للعمليات بتاعت ال `async`

وفي ال `method` هنقوه يرجعنا `task` ال `task` بتاعه `data type`

ال `task` هنا هوa `operation` بترجع قيمه ولو كانت ال `method` بترجع `void` بتكتب `task` بس

```
static async Task<int> PerformAsyncOperation() {  
    await Task.Delay(4000);  
    return 42;  
}
```

بعدين نيجي لل `main`

هناخد `object` من `task` ونخلي ال `task` يكون `int` او من نفس النوع اللي راجع من ال `method` ونديله ال `method` اللي هيشغلها

```
Task<int> resultTask = PerformAsyncOperation();
```

وبعدين هنعرف متغير نخزن فيه القيمه اللي راجعه من ال `task` وبنقوله يستني لحد مايخلصها عشان يرجع بالرقم

```
int result = await resultTask;  
Console.WriteLine($"Result: {result}");
```

ده الكود كله

```
using System;  
using System.Threading.Tasks;  
  
class NullableExample  
{  
    static async Task<int> PerformAsyncOperation() {  
        await Task.Delay(4000);  
    }  
}
```

```

    return 42;
}

static async Task Main()
{
    Task<int> resultTask = PerformAsyncOperation();

    Console.WriteLine("Do Some Other Work... ");

    int result = await resultTask;
    Console.WriteLine($"Result: {result}");

    Console.ReadKey();
}
}

```

طب ايه اللي بيحصل فيه؟

اول حاجه بيروح لأول سطر وهو ان يعمل object من task ويبدأ يشغل ال method وبعدين
بلاقي امر بيعطله اللي هو ال delay

كده المفروض انه

ينقل للسطر اللي بعده ويعمل return

بس عشان فيه delay فيروح للسطر بتاع do some work عمال ما الرابع ثواني تخلص
بعدين بينقل للسطر اللي بعده وهو انه يعرف متغير من النوع int

المفروض هنا انه يأخذ القيمه بتاعت ال result task بس بلاقي كلمة await يعني استني لحد مال
result Task تخلص

فيينقل لل perform operations عشان يكمل شغل

هنا المفروض انه هينفذ ال return بس بلاقي كلمة await

فيبيستني لحد مال delay يخلص

بعدين بينفذ السطر اللي بعده ويرجع القيمه

بعدين بيروح يكمل تعريف المتغير وبدليله القيمه اللي راجعه

بعدين بيكمي شغله في ال main عادي

`System.Threading.Tasks` is a namespace in the .NET Framework (and its successor, .NET Core) that provides a set of classes and interfaces for working with asynchronous programming and parallel processing. The namespace is part of the Task Parallel Library (TPL) and is designed to simplify the development of parallel and asynchronous code.

Here are some key components and concepts within the `System.Threading.Tasks` namespace:

- Task Class:

- The `Task` class is a fundamental building block in the Task Parallel Library. It represents an asynchronous operation and provides methods to monitor and interact with the operation's status.
- Async and Await Keywords:
 - The `async` and `await` keywords, introduced in C# 5.0, are used to simplify asynchronous programming. They allow developers to write asynchronous code in a more natural and readable way.

Here's a simple example of using `Task` and `async/await`:

```
using System;
using System.Threading.Tasks;

class Program
{
    static async Task Main()
    {
        // Create and run an asynchronous task
        Task<int> resultTask = PerformAsyncOperation();

        // Do some other work while waiting for the task to complete
        Console.WriteLine("Doing some other work...");

        // Wait for the task to complete and retrieve the result
        int result = await resultTask;

        // Process the result
        Console.WriteLine($"Result: {result}");
    }

    static async Task<int> PerformAsyncOperation()
    {
        // Simulate an asynchronous operation
    }
}
```

```

        await Task.Delay(2000);

        // Return a result
        return 42;
    }
}

```

In this example, the `PerformAsyncOperation` method simulates an asynchronous operation using `Task.Delay`, and the `async/await` keywords are used to compose asynchronous operations in a readable manner. The `Main` method can perform other work while waiting for the asynchronous operation to complete. The `Task<int>` represents the asynchronous operation's result.

Overall, `System.Threading.Tasks` plays a crucial role in modern .NET development, making it easier for developers to write efficient, scalable, and readable asynchronous and parallel code.

Task Class Example 2

بعض کده کود ده

```

using System;
using System.Net;
using System.Net.Mime;
using System.Threading.Tasks;

class NullableExample
{
    static async Task DownloadAndPrintAsync(string url) {
        string Content;

        using (WebClient Client=new WebClient()) {
            await Task.Delay(100);
            Content = await Client.DownloadStringTaskAsync(url);
        }
        Console.WriteLine($"{url}: {Content.Length} Charachters dowloaded");
    }

    static async Task Main()
    {
        Console.WriteLine("Starts Tasks...");

        Task task1 = DownloadAndPrintAsync("https://www.cnn.com");
        Console.WriteLine("Task 1 started...");

        Task task2 = DownloadAndPrintAsync("https://www.amazon.com");
        Console.WriteLine("Task 2 started...");

        Task task3 = DownloadAndPrintAsync("https://www.ProgrammingAdvices.com");
        Console.WriteLine("Task 3 started...");
    }
}

```

```
        await Task.WhenAll(task1, task2, task3);

        Console.WriteLine($"AllDone");

        Console.ReadKey();
    }
}
```

البرنامج هنا هيمشي ازاي؟

اول حاجه هينفذ السطر ده

```
Console.WriteLine("Starts Tasks...");
```

بعدين هيدخل عالسطر ده يعمل ال object

```
Task task1 = DownloadAndPrintAsync("https://www.cnn.com");
```

وبعدين هيدخل عال method وينفذ لحد هنا

```
static async Task DownloadAndPrintAsync(string url) {
    string Content;

    using (WebClient Client=new WebClient()) {

        await Task.Delay(100);
    }
}
```

هيلامي نفسه قاعد فاضي فيرجع لل main وينفذ السطر ده

```
Console.WriteLine("Task 1 started...");
```

وبعدين ينفذ السطر ده ويدخل عال method

```
static async Task DownloadAndPrintAsync(string url) {
    string Content;

    using (WebClient Client=new WebClient()) {
        await Task.Delay(100);
    }
}
```

كده انت بقى عندك مسارين عطلان فيهـم

فيدخل عال main لثالث مره وينفذ السطر ده

```
Console.WriteLine("Task 2 started...");
```

وبعدين يكمل في السطر ده

```
Task task3 = DownloadAndPrintAsync("https://www.ProgrammingAdvises.com");
```

ويدخل في ال method وينفذ في السطر ده

```
static async Task DownloadAndPrintAsync(string url) {
    string Content;

    using (WebClient Client=new WebClient()) {
        await Task.Delay(100);
    }
}
```

هيلالي نفسه عطلان لثالث مره فيدخل عال main وينفذ السطر ده

```
Console.WriteLine("Task 3 started...");
```

وبعدها ينقل عالسطر اللي بعده

```
await Task.WhenAll(task1, task2, task3);
```

هنا بيقوله استني لحد مال tasks كلها تخلص

فيرجع لأول task ويستني المده تخلص وبيجي ينفذ السطر ده

```
Content = await Client.DownloadStringTaskAsync(url);
```

يقوله استني لحد مالتحميل يخلص

يقدر فاضي؟

لا يدخل عال task 2 ويستني المده تخلص وبيجي ينفذ السطر ده

```
Content = await Client.DownloadStringTaskAsync(url);
```

يقوله استني لحد مالتحميل يخلص

يقدر فاضي؟

لا يدخل عال task3 ويستني المده تخلص وينفذ السطر ده

```
Content = await Client.DownloadStringTaskAsync(url);
```

يقوله استني لما التحميل يخلص

يقدر فاضي؟

هنا هيقدر فاضي لانه لو دخل عال main هيلاليه بيقوله استني برضه

فيقدر مستني لحد ما صفحه من الصفحات تحمل وينزل ينفذ جمله الطباعه دي

```
Console.WriteLine($"{url}: {Content.Length} Charachters dowloaded");
```

ويستني لما task تانيه تحمل ويطبع ويستني حد ما task تالته تحمل وتطبع
وبعدين يدخل عال main بيقوله خلاوبيص؟ بيقوله خلاص يقوم مكمل في الكود عادي

Task Class With Call back Event Example

هنا احنا عاززين نشغل task معينه ولما تخلص تشغلي event تبلغك انه خلص
اول حاجه هيا انا نجهز ال event ونأخذ object منه

```
public class CustomEventArgs : EventArgs {
```

```

public int Parameter1 { get; }
public string Parameter2 { get; }

public CustomEventArgs(int Parameter1, string Parameter2) {
    this.Parameter1 = Parameter1;
    this.Parameter2 = Parameter2;
}

public delegate void CallbackEventHandler(object sender, CustomEventArgs e);

public static event CallbackEventHandler CallbackEvent;

```

وبعدين هنعمل event ونضيفها لـ method

```

static void OnCallBackRecieved(object sender, CustomEventArgs e) {
    Console.WriteLine($"Event Recieved: Parameter 1 = {e.Parameter1} Parameter 2= {e.Parameter2}");
}

static async Task Main()
{
    CallbackEvent += OnCallBackRecieved;
}

```

وبعدين نعمل الـ event بتاعت الـ task ونخليها تأخذ parameter من نوع الـ object عن طريق الـ invoke بعد ما نخلص كل الكود بتاعت الـ method بحيث انه لما يخلص شغل يسدعني الـ on call back received method بتاعت

```

static async Task PerformAsyncOperation(CallbackEventHandler callBack) {
    await Task.Delay(2000);
    CustomEventArgs eventArgs = new CustomEventArgs(42, "Hello from event");
    callBack?.Invoke(null, eventArgs);
}

```

بس كده وبقيت الكود مافيهوش مشكله

هوا الجديد بس انه بيقولك لو عايز تعمل event مع الـ async task خلي الـ method بتاعت الـ event من الـ invoke بحيث انها تعمل لما تخلص شغل task تأخذ parameter

```

using System;
using System.Net;
using System.Threading.Tasks;

class NullableExample
{

    public class CustomEventArgs : EventArgs {

        public int Parameter1 { get; }
        public string Parameter2 { get; }

        public CustomEventArgs(int Parameter1, string Parameter2) {
            this.Parameter1 = Parameter1;
        }
    }
}

```

```

        this.Parameter2 = Parameter2;
    }

public delegate void CallbackEventHandler(object sender, CustomEventArgs e);

public static event CallbackEventHandler CallbackEvent;

static async Task PerformAsyncOperation(CallbackEventHandler callBack) {
    await Task.Delay(2000);
    CustomEventArgs eventArgs = new CustomEventArgs(42, "Hello from event");
    callBack?.Invoke(null, eventArgs);
}

static void OnCallBackRecieved(object sender, CustomEventArgs e) {
    Console.WriteLine($"Event Recieved: Parameter 1 = {e.Parameter1} Parameter 2= {e.Parameter2}");
}

static async Task Main()
{
    CallbackEvent += OnCallBackRecieved;
    Task PerformTask = PerformAsyncOperation(CallbackEvent);

    Console.WriteLine("Doing some Other work");

    await PerformTask;
    Console.WriteLine("Done");

    Console.ReadKey();
}
}

```

Task.Run

هنا بيقولك انه ال `task.run` بستخدمها عشان تنفذ كود بتاع `thread` معين في الخلفيه بحيث ان الفورم مایتعطلاش ويسمح لليوزر بتحريكه

هنا انت كأنك بتعمل `thread` وبنشغلها

وهوا بيستخدم مش `threads`

وكمان بيقولك انك ممكن تدمج مابين ال `async` وال `threading` بحيث تخلی البرنامج اسرع

Task.Run

- Task.Run is a powerful method used for multithreading in C#. It allows you to execute code concurrently by offloading it to a background thread from the thread pool. This keeps the UI responsive while the background task is running.
- Benefits:
 - Improved responsiveness: UI remains responsive while long-running tasks execute in the background.
 - Increased efficiency: Utilizing multiple cores of the CPU simultaneously speeds up overall execution.
- When to use it:
 - For long-running tasks that block the UI.
 - When you need to perform multiple tasks concurrently.
 - When you want to improve responsiveness and efficiency.
- Remember:
 - Use Task.Run judiciously, as excessive thread creation can impact performance.
 - Always manage resources and ensure proper thread synchronization to avoid race conditions.



وذه مثال

```
using System;
using System.Threading;
using System.Threading.Tasks;

class NullableExample
{
    static void DownloadFile(string TaskName){
        Console.WriteLine($"{TaskName} : Started!");
        Thread.Sleep(5000);
        Console.WriteLine($"{TaskName} : Completed!");
    }

    static async Task Main()
    {
        Task task1 = Task.Run(() => DownloadFile("Download File 1"));
        Task task2 = Task.Run(() => DownloadFile("Download File 2"));

        await Task.WhenAll(task1, task2);

        Console.WriteLine($"Task 1 and 2 completed");

        Console.ReadKey();
    }
}
```

What is Task.Run?

Task.Run is a powerful method in C# that facilitates multithreading. It is part of the Task Parallel Library (TPL) and serves as a convenient way to execute code concurrently by offloading it to a background thread from the thread pool. This is particularly useful for scenarios where you want to

keep the user interface (UI) responsive while computationally intensive or time-consuming tasks run in the background.

Benefits:

1. Improved Responsiveness:

By utilizing `Task.Run`, you ensure that long-running tasks are executed in the background, preventing them from blocking the UI. This enhances the overall responsiveness of your application, providing a smoother user experience.

2. Increased Efficiency:

`Task.Run` enables the simultaneous utilization of multiple CPU cores, optimizing the execution of tasks. This can lead to improved efficiency and performance in scenarios where parallel processing is beneficial.

When to use it:

1. For Long-Running Tasks that Block the UI:

`Task.Run` is particularly well-suited for tasks that have the potential to block the UI due to their duration. Offloading such tasks to a background thread ensures that the UI remains responsive to user interactions.

2. When You Need to Perform Multiple Tasks Concurrently:

If your application involves multiple independent tasks that can run concurrently, `Task.Run` provides a straightforward way to parallelize the execution of these tasks.

3. When You Want to Improve Responsiveness and Efficiency:

`Task.Run` is a valuable tool when the goals include both improving UI responsiveness and optimizing the overall efficiency of task execution through parallelism.

Remember:

1. Use Task.Run Judiciously:

While `Task.Run` is a powerful tool, excessive use can lead to the creation of too many threads, impacting performance negatively. Exercise caution and use it judiciously based on the specific needs of your application.

2. Manage Resources and Ensure Thread Synchronization:

To prevent potential issues such as race conditions, it is crucial to manage resources carefully and ensure proper thread synchronization. This involves using synchronization mechanisms when accessing shared resources to avoid conflicts.

By understanding `Task.Run`, you can leverage multithreading effectively in your C# applications, leading to more responsive and efficient code.

Quiz 2

What is Task. Run?

A method in C# that facilitates multithreading

A method in C# that is used to handle exceptions

A method in C# that is used for input/output operations

What is one benefit of using Task. Run?

Improved Responsiveness

Reduced Memory Usage

Faster Compilation

When should you use Task.Run?

For tasks that block the UI

For single-threaded applications

For simple calculations

What should you be cautious of when using Task.Run?

Creating too many threads

Using too much memory

Causing deadlock

Slowing down the UI

How does Task.Run improve efficiency?

Task.Run uses multiple CPU cores, optimizing task execution for better efficiency.

Task.Run only uses a single CPU core.

When is Task.Run useful?

Task.Run is useful for tasks that might block the UI due to their duration, ensuring responsiveness.

Task.Run is not suitable for long tasks.

What kind of application benefits from Task.Run?

Task.Run is best for sequential tasks.

Applications with multiple independent tasks can benefit from Task.Run for parallel execution.

What should developers be cautious about when using Task.Run?

Excessive use of Task.Run can create too many threads, impacting performance negatively.

Use it judiciously based on your application's needs.

Task.Run can be used without any performance considerations.

How can developers prevent potential issues with Task.Run?

Task.Run automatically manages resources, eliminating the need for manual intervention.

Manage resources carefully and ensure proper thread synchronization to avoid problems like race conditions.

Introduction to Task Factory Class

ال task factory class هي مكتبه بتسهل عليك التعامل مع ال tasks و بتتيح لك انك تعمل معينه قبل ما تشغلى ال task او انك تضم اكتر من task تحت task configurations و تتعامل معاهم كهم مرة واحده منها مثلا انك لما تيجي تشغلى task معينه تقدر في اي وقت تلغىها او انك تدي task معينه اولويه عن الثانية

الفكرة هنا انك بتعمل task factory من object و بتحدد فيه ازاي ال tasks اللي هتضيفها ليه هتشتغل

وبعدين لما تحب تضيف له task جديدة بتسدعي start new method اسمها task و تخزنها في object من النوع

يعني اعتبر نفسك بتعمل tasks عاديه جدا بس بتحددلها الاطار العام اللي هتشتغل فيه من خلال ال task factory

```
using System;
using System.Threading;
using System.Threading.Tasks;
```

```
class NullableExample
{
    static async Task Main()
    {
        CancellationTokenSource cts=new CancellationTokenSource();
        CancellationToken token=cts.Token;

        TaskFactory taskFactory = new TaskFactory(token,
            TaskCreationOptions.AttachedToParent,
            TaskContinuationOptions.ExecuteSynchronously,
            TaskScheduler.Default);

        Task task1 = taskFactory.StartNew(() =>
        {
            Console.WriteLine("Task 1 is running");
            Thread.Sleep(2000);
            Console.WriteLine("Task 1 Completed");
        });

        // cts.Cancel();
        Task task2 = taskFactory.StartNew(() =>
        {
            Console.WriteLine("Task 2 is running");
            Thread.Sleep(2000);
            Console.WriteLine("Task 2 Completed");
        });

        try {
            Task.WaitAll(task1, task2);
            Console.WriteLine("All tasks Completed");
        } catch (AggregateException ae) {
            foreach (var e in ae.InnerExceptions)
            {

                Console.WriteLine(e.Message);
            }
        }

        cts.Dispose();

        Console.ReadKey();
    }
}
```

TaskFactory Class

- The TaskFactory class in C# is part of the Task Parallel Library (TPL) and is used to create and control tasks more efficiently.
- TaskFactory is a class in the System.Threading.Tasks namespace that provides methods for creating and scheduling tasks.
- It simplifies the process of working with tasks by offering a set of convenient factory methods.
- It provides methods for creating and scheduling Task and Task<TResult> instances, thus offering more control over how these tasks are executed.
- The TaskFactory class can be especially useful when you need to create multiple tasks with common configuration settings, such as cancellation tokens, task creation options, or task continuations.

Key Uses of TaskFactory:

- Creating Tasks: Simplify the creation of tasks, especially when you have multiple tasks that share similar configurations.
- Task Scheduling: Manage how tasks are scheduled for execution. It can be used to queue tasks on a specific TaskScheduler.
- Continuation Tasks: Easily create continuation tasks that execute after the completion of a previous task.

TaskFactory Class:

The TaskFactory class in C# is part of the Task Parallel Library (TPL) and is used to create and control tasks more efficiently.

TaskFactory is a class in the System.Threading.Tasks namespace that provides methods for creating and scheduling tasks.

It simplifies the process of working with tasks by offering a set of convenient factory methods.

It provides methods for creating and scheduling `Task` and `Task<TResult>` instances, thus offering more control over how these tasks are executed. The `TaskFactory` class can be especially useful when you need to create multiple tasks with common configuration settings, such as cancellation tokens, task creation options, or task continuations.

Key Uses of TaskFactory:

1. **Creating Tasks:** Simplify the creation of tasks, especially when you have multiple tasks that share similar configurations.
2. **Task Scheduling:** Manage how tasks are scheduled for execution. It can be used to queue tasks on a specific `TaskScheduler`.
3. **Continuation Tasks:** Easily create continuation tasks that execute after the completion of a previous task.

Example Usage of TaskFactory:

Here's a simple example demonstrating the use of `TaskFactory`:

```
using System;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static void Main(string[] args)
    {
        // Define a cancellation token so we can stop the task if needed
        CancellationTokenSource cts = new CancellationTokenSource();
        CancellationToken token = cts.Token;

        // Create a TaskFactory with some common configuration
        TaskFactory taskFactory = new TaskFactory(
            token,
```

```
TaskCreationOptions.AttachedToParent,
TaskContinuationOptions.ExecuteSynchronously,
TaskScheduler.Default);

// Use the TaskFactory to create and start a new task
Task task1 = taskFactory.StartNew(() =>
{
    Console.WriteLine("Task 1 is running");
    // Simulate some work
    Thread.Sleep(2000);
    Console.WriteLine("Task 1 completed");
});

// Create another task using the same TaskFactory
Task task2 = taskFactory.StartNew(() =>
{
    Console.WriteLine("Task 2 is running");
    // Simulate some work
    Thread.Sleep(1000);
    Console.WriteLine("Task 2 completed");
});

try
{
    // Wait for both tasks to complete
    Task.WaitAll(task1, task2);
    Console.WriteLine("All tasks completed.");
}
catch (AggregateException ae)
{
    // Handle exceptions if any task throws
    foreach (var e in ae.InnerExceptions)
        Console.WriteLine($"Exception: {e.Message}");
}

// Dispose of the CancellationTokenSource
```

```
        cts.Dispose();  
    }  
}
```

In this example, a `TaskFactory` is created with specific configuration options, and it's used to start two tasks. Both tasks are managed by the same `TaskFactory`, thus inheriting its configuration settings. The `CancellationTokenSource` and `CancellationToken` are used to demonstrate how you could cancel the tasks if necessary.

When to Use TaskFactory:

- When you need multiple tasks with similar configurations.
- When you need more control over task scheduling.
- When working with task continuations or parent-child task relationships.

Remember, while `TaskFactory` is a powerful tool, it's not always necessary for simple task parallelism scenarios. The `Task.Run` method is often sufficient for running tasks without requiring specific configurations that `TaskFactory` provides.

`TaskCreationOptions` is an enumeration in the .NET Framework that provides several options to control the behavior of tasks created using the Task Parallel Library (TPL). These options give you more fine-grained control over how tasks are scheduled, executed, and how they behave in relation to other tasks. Here are some of the key options available in the `TaskCreationOptions` enumeration:

1. **None:** Specifies that the default configuration should be used. This is the most common setting when none of the other options are needed.
2. **PreferFairness:** Hints to the Task Scheduler that fairness should be prioritized. This means that tasks will be scheduled in the order they were queued, attempting to avoid situations where a later-created task runs before an earlier-created one.

3. LongRunning: Indicates that a task will be a long-running, coarse-grained operation involving fewer, larger components. This hint allows the Task Scheduler to optimize the execution of these tasks, possibly avoiding the overhead of context switching.
4. AttachedToParent: Specifies that a task is attached to a parent in the task hierarchy. When a parent task waits on its children, it will implicitly wait for all tasks attached to it as well.
5. DenyChildAttach: Prevents any child tasks from attaching to the current task. This option is useful when you want to create a task within another task but prevent it from being attached to the parent task.
6. HideScheduler: Prevents the ambient task scheduler from being seen as the current scheduler in the created task. This means that tasks created within this task will not see the current task's scheduler as the default scheduler.
7. RunContinuationsAsynchronously: Ensures that continuations added to the current task run asynchronously. This option can be used to avoid potential deadlocks and improve responsiveness.

`TaskContinuationOptions` is an enumeration in C# that provides a set of options that control the behavior of task continuations. Task continuations are created using methods like `Task.ContinueWith`, which allow you to specify actions that should be executed after a task completes. These options give you fine-grained control over when and how continuations are executed in relation to the task they are continuing from. Here are some of the key options available in the `TaskContinuationOptions` enumeration:

1. None: Specifies that the default behavior should be used, with no special conditions applied to the continuation task.
2. PreferFairness: Indicates that the continuation task should be scheduled in a fair manner in relation to other tasks. This can be useful in scenarios where you want to maintain a first-in, first-out (FIFO) order of task execution.
3. LongRunning: Suggests that the continuation will be a long-running operation. This option hints to the Task Scheduler that an additional thread might be advantageous for running the continuation, thus reducing the chance of it blocking other tasks.

4. AttachedToParent: Indicates that the continuation task is attached to the parent task in a nested task structure. The parent task will wait for the attached child continuation tasks to complete before it completes.
5. ExecuteSynchronously: Advises the Task Scheduler to try to execute the continuation synchronously on the same thread that ran the antecedent task, rather than queuing it to the ThreadPool. This can be more efficient for short continuations, but care should be taken as it can potentially lead to deadlocks or stack overflows.
6. LazyCancellation: Specifies that the continuation task should not be canceled immediately if its antecedent task is canceled. Instead, the continuation will start and then check the cancellation token, if any, to determine whether to continue executing.
7. NotOnRanToCompletion: Specifies that the continuation should not be executed if the antecedent task completes successfully.
8. NotOnFaulted: Specifies that the continuation should not be executed if the antecedent task throws an unhandled exception (faults).
9. NotOnCanceled: Specifies that the continuation should not be executed if the antecedent task is canceled.
10. OnlyOnRanToCompletion: Specifies that the continuation should only be executed if the antecedent task completes successfully.
11. OnlyOnFaulted: Specifies that the continuation should only be executed if the antecedent task throws an unhandled exception.
12. OnlyOnCanceled: Specifies that the continuation should only be executed if the antecedent task is canceled.

`TaskScheduler` is a fundamental class in the Task Parallel Library (TPL) in C#. It serves as the engine behind task scheduling, execution, and management. The `TaskScheduler` abstract class provides a means to queue tasks to the ThreadPool or to a custom scheduler.

Key Concepts of TaskScheduler:

1. Default Task Scheduler: By default, the TPL uses the `ThreadPoolTaskScheduler`. This default scheduler is efficient for most scenarios, balancing responsiveness and throughput. It queues tasks to the system's thread pool.

2. Custom Task Schedulers: You can implement custom task schedulers by deriving from the `TaskScheduler` class. This is useful for specialized scenarios, like scheduling tasks to run on a UI thread, handling tasks with specific priorities, or managing tasks in an environment with unique threading requirements.
3. Queuing Tasks: The `TaskScheduler` determines how tasks are queued and when they are executed. It abstracts the details of how tasks are managed, whether that's on a thread pool, a single thread, or some other mechanism.
4. Task Execution: The scheduler determines when and how a task is executed. This includes considerations like concurrency limits, prioritization, and dealing with unhandled exceptions in tasks.
5. Synchronization Context Integration: For applications with a synchronization context (like Windows Forms or WPF applications), the default scheduler can ensure that task continuations that update the UI are executed on the correct thread.

Quiz 3

Which namespace does the `TaskFactory` class belong to?

System.Threading

System.Tasks

System.Threading.Tasks

System.Factory

What is the primary purpose of the `TaskFactory` class in C#?

To provide a set of methods for synchronous task execution.

To provide methods for creating and scheduling tasks efficiently.

To manage threading and synchronization primitives.

To handle exception logging in asynchronous tasks.

Which of the following is NOT a key use of the TaskFactory class?

Creating tasks with shared configurations.

Scheduling tasks for execution.

Automatically handling all exceptions thrown by tasks.

Creating continuation tasks.

The TaskFactory class simplifies the process of working with which of the following?

Event handlers

Task and Task instances

Database connections

File I/O operations

What does the TaskFactory class allow you to do in terms of task scheduling?

Prevent tasks from being queued on any TaskScheduler.

Force all tasks to run on the main application thread.

Queue tasks on a specific TaskScheduler.

Schedule tasks to run only at system startup.

Which method would you typically use to create a task with TaskFactory?

TaskFactory.StartNew()

TaskFactory.Run()

TaskFactory.CreateNew()

TaskFactory.Initiate()

What advantage does TaskFactory offer when creating multiple tasks?

Automatic cancellation of all tasks on failure.

Creating multiple tasks with common configuration settings.

Ensuring all tasks run on the same thread.

Increasing the priority of all tasks in the system.

The TaskFactory class is particularly useful for which type of tasks?

Tasks that require UI thread synchronization.

Tasks that need common configuration such as cancellation tokens.

Tasks that interact with external databases.

Short-running tasks that require immediate execution.

Which of the following is a true statement about TaskFactory?

It replaces the need for the Task class.

It is primarily used for synchronous task execution.

It provides a more convenient way to work with tasks than using Task directly.

It is only available in .NET Core and not in .NET Framework.

When using TaskFactory, what does the option `TaskCreationOptions.AttachedToParent` imply?

The task should be executed independently of its parent task.

The task is a long-running operation and should use a dedicated thread.

The task is attached to a parent task and will be executed alongside it.

The task will be executed only after its parent task has completed.

Which of the following best describes the relationship between TaskFactory and TaskScheduler?

TaskFactory replaces the need for TaskScheduler.

TaskFactory uses TaskScheduler internally to queue and execute tasks.

TaskFactory and TaskScheduler are unrelated classes in the TPL.

TaskScheduler is used to create tasks, while TaskFactory is used to schedule them.

Which statement is true regarding task cancellation with TaskFactory?

TaskFactory does not support task cancellation.

TaskFactory automatically cancels all tasks if one task fails.

Tasks created with TaskFactory can be cancelled using CancellationToken.

TaskFactory only allows cancellation of tasks before they start executing.

Parallel Class in C#

ال parallelism بيوفر لك parallel class تقدر تستخدمها عشان تحقق ال loops و methods في البرمجه

Parallel Class in C#



- The Parallel class offers methods for parallelizing loops, executing parallel tasks, and performing parallel aggregation.
- It simplifies parallel programming by abstracting many of the low-level details.
- In C#, the Parallel class is a part of the Task Parallel Library (TPL) that simplifies parallel execution of code.
- This class provides methods for running tasks in parallel, making it easier to perform multi-threading and parallel processing.
- It's especially useful for data parallelism: performing the same operation concurrently across a collection of data.

Key Concepts:

- **Parallelism:** The concept of executing multiple operations simultaneously.
- **Task Parallel Library (TPL):** A set of APIs in .NET Framework for parallel programming.
- **Data Parallelism:** The type of parallelism where the same operation is performed concurrently on elements in a collection.

Methods of the Parallel Class

- `Parallel.For`: Executes a for loop in which iterations may run in parallel.
- `Parallel.ForEach`: Executes a foreach loop over any `IEnumerable` or `IEnumerable<T>` where iterations may run in parallel.
- `Parallel.Invoke`: Executes each provided Action in parallel.

Parallel Class in C#

Introduction

The `Parallel` class offers methods for parallelizing loops, executing parallel tasks, and performing parallel aggregation. It simplifies parallel programming by abstracting many of the low-level details.

In C#, the `Parallel` class is a part of the Task Parallel Library (TPL) that simplifies parallel execution of code. This class provides methods for running tasks in parallel, making it easier to

perform multi-threading and parallel processing. It's especially useful for data parallelism: performing the same operation concurrently across a collection of data.

Key Concepts

1. Parallelism: The concept of executing multiple operations simultaneously.
2. Task Parallel Library (TPL): A set of APIs in .NET Framework for parallel programming.
3. Data Parallelism: The type of parallelism where the same operation is performed concurrently on elements in a collection.

Using the Parallel Class

Methods of the Parallel Class

- `Parallel.For`: Executes a `for` loop in which iterations may run in parallel.
- `Parallel.ForEach`: Executes a `foreach` loop over any `IEnumerable` or `IEnumerable<T>` where iterations may run in parallel.
- `Parallel.Invoke`: Executes each provided `Action` in parallel.

Best Practices and Considerations

- Thread Safety: Ensure your code is thread-safe when using parallel constructs.
- Overhead: Parallel operations introduce some overhead. Use them when the benefits of parallelism outweigh this overhead.
- I/O Bound vs CPU Bound: The `Parallel` class is best suited for CPU-bound operations. For I/O-bound operations, consider using asynchronous programming with `async` and `await`.

Conclusion

The `Parallel` class in C# provides a simple and efficient way to implement parallelism in your applications. It's particularly useful for scenarios where you need to perform operations concurrently across collections or execute multiple independent actions simultaneously. However, always consider thread safety and the nature of the tasks when using this class.

Parallel.For

ال parallel.for هيا **for loop** عاديه بس مخصوصه لل threads يعني بدل ماتعمل اكتر من **method** او بدل ماتعمل **for loop** وجوهاها تقوله اعمل task جديد وخلبها تشغله ال thread الفلانيه لا هوا وفر عليك القصه دي

```
using System;
using System.Threading.Tasks;

class NullableExample
{
    static async Task Main()
    {
        int NumberOflterations = 10;

        Parallel.For(0, NumberOflterations, i => {
            Console.WriteLine(i + " " + Task.CurrentId);
            Task.Delay(1000).Wait();
        });

        Console.WriteLine("Finished");

        Console.ReadKey();
    }
}
```

وفي الكود ده هوا بدل ال method ب lambda expression

```
using System;
using System.Threading.Tasks;

class NullableExample
{
    static async Task Main()
    {
        int NumberOflterations = 10;

        Parallel.For(0, NumberOflterations, ProcessIteration);

        Console.WriteLine("Finished");

        Console.ReadKey();
    }

    static void ProcessIteration(int i) {
        Console.WriteLine(i + " " + Task.CurrentId);
        Task.Delay(1000).Wait();
    }
}
```

Parallel.For

`Parallel.For` is a method in C# that allows you to execute a for loop in parallel, making it easier to perform parallel operations on a collection or for a specific number of iterations.

Example:

Here's an example demonstrating how to use `Parallel.For`:

```
using System;
using System.Threading.Tasks;

class Program
{
    static void Main()
    {
        // Define the number of iterations
        int numberOfIterations = 10;

        // Use Parallel.For to execute the loop in parallel
        Parallel.For(0, numberOfIterations, i =>
        {
            Console.WriteLine($"Executing iteration {i} on thread {Task.CurrentId}");
            // Simulate some work
            Task.Delay(1000).Wait();
        });

        Console.WriteLine("All iterations completed.");
    }
}
```

In this example:

1. `Parallel.For` is used to execute iterations in parallel from 0 to `numberOfIterations - 1`.
2. Inside the `Parallel.For`, a lambda expression is defined to perform the work for each iteration. In this case, it's printing the iteration number and the current task ID to the console, followed by a simulated work delay.

3. `Parallel.For` will handle the distribution of these iterations across multiple threads.
4. After `Parallel.For` completes, a message is printed to indicate that all iterations are done.

Note: The actual order of execution of the iterations may vary each time you run the program, as `Parallel.For` will execute them concurrently on different threads.

Same Example using separate function:

To use `Parallel.For` with separate functions instead of a lambda expression, you can define a method that encapsulates the work you want to perform in each iteration. Here's how you can modify the example:

```
using System;
using System.Threading.Tasks;

class Program
{
    static void Main()
    {
        // Define the number of iterations
        int numberOfIterations = 10;

        // Use Parallel.For to execute the loop in parallel
        Parallel.For(0, numberOfIterations, ProcessIteration);

        Console.WriteLine("All iterations completed.");
    }

    static void ProcessIteration(int i)
    {
        Console.WriteLine($"Executing iteration {i} on thread {Task.CurrentId}");
        // Simulate some work
        Task.Delay(1000).Wait();
    }
}
```

In this revised example:

1. The `ProcessIteration` method is defined to handle the work for each iteration. It takes an integer `i` as a parameter, which represents the current iteration number.
2. `Parallel.ForEach` is used to execute iterations in parallel, calling the `ProcessIteration` method for each iteration.
3. Within `ProcessIteration`, the iteration number and the current task ID are printed, followed by a simulated delay to represent some work.

By using this approach, the loop's work is encapsulated in a method, making the code more modular and potentially easier to understand and maintain.

Parallel.ForEach

ده ال for loop نفس فکرہ ال for each

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Threading;
using System.Threading.Tasks;

class NullableExample
{
    static List<string> urls = new List<string> {
        "https://www.cnn.com",
        "https://www.amazon.com",
        "https://www.ProgrammingAdvises.com"
    };

    static void Main()
    {
        Parallel.ForEach(urls, url =>
        {
            DownloadContent(url);
        });

        Console.WriteLine("Done");
        Console.ReadKey();
    }

    static void DownloadContent(string url) {
        string content;
        using (WebClient client=new WebClient()) {
            Thread.Sleep(100);
            content = client.DownloadString(url);
        }
        Console.WriteLine(url+ " "+content.Length);
    }
}
```

Parallel.ForEach:

`Parallel.ForEach` is used in C# to execute a parallel loop over a collection or an enumerable. It can efficiently process items in parallel, improving performance for suitable tasks.

Here's an example demonstrating `Parallel.ForEach`:

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static List<string> urls = new List<string>
    {
        "https://www.cnn.com",
        "https://www.amazon.com",
        "https://www.programmingadvices.com"
    };

    static void Main()
    {

        // Use Parallel.ForEach to download the web pages concurrently
        Parallel.ForEach(urls, url =>
        {
            DownloadContent(url);

        });

        Console.WriteLine("Done!");
        Console.ReadKey();
    }
}
```

```
static void DownloadContent(string url)
{
    string content;

    using (WebClient client = new WebClient())
    {
        // Simulate some work by adding a delay
        Thread.Sleep(100);

        // Download the content of the web page
        content = client.DownloadString(url);
    }

    Console.WriteLine($"'{url}': {content.Length} characters downloaded");
}

}
```

downloading content from a list of URLs using `Parallel.ForEach`. This approach will download the contents of each URL in parallel, and after downloading, it prints the length of the content directly within the `DownloadContent` method. This is a thread-safe way to handle the console output since `Console.WriteLine` is inherently thread-safe.

Here's a brief overview of your code:

1. List of URLs: You have a list of URLs (`urls`) from which the content will be downloaded.
2. Parallel Download: `Parallel.ForEach` is used to iterate over the URLs. For each URL, `DownloadContent` is called in parallel.
3. Download and Print: Inside `DownloadContent`, each URL's content is downloaded, and the length of the content is printed to the console.

- Completion Message: After all parallel operations are complete, a "Done!" message is displayed.
- Thread Delay: There's a `Thread.Sleep(100)` call in `DownloadContent`, simulating some work or delay. This may represent the network latency in a real-world scenario.

This code will efficiently utilize multiple threads to download data from different URLs simultaneously, which can significantly speed up the process compared to downloading each URL sequentially.

Just a few points to consider:

- Error Handling: Consider adding error handling within `DownloadContent` to manage situations where a download might fail (e.g., due to a bad URL or network issues).
- Resource Utilization: Be aware that making too many simultaneous web requests can strain the network and the server you are querying. It's generally good practice to limit the number of concurrent network requests.
- Async Approach: For network-bound operations like web requests, an asynchronous approach using `async` and `await` with `HttpClient` might be more efficient and can help avoid blocking threads. This is especially relevant for I/O-bound operations.

Parallel.Invoke

الطريقتين اللي فاتوا كنت بتسعملهم لو عندك اكتر من `task` بيستخدموا نفس الـ `method` هنا بقي انت عندك اكتر من `method` بتشغل كل واحده في `task` لوحدها

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Threading;
using System.Threading.Tasks;

class NullableExample
{
    static void Main()
    {
        Console.WriteLine("Starting parallel functions");
        Parallel.Invoke(Function1, Function2, Function3);
        Console.WriteLine("all functions are completed");
        Console.ReadKey();
    }

    static void Function1()
    {
        Console.WriteLine("Function 1 is starting");
        Task.Delay(1000).Wait();
        Console.WriteLine("Function1 is completed");
    }
}
```

```
        }

    static void Function2()
    {
        Console.WriteLine("Funcion 2 is starting");
        Task.Delay(1000).Wait();
        Console.WriteLine("Function2 is completed");
    }

    static void Function3()
    {
        Console.WriteLine("Funcion 3 is starting");
        Task.Delay(1000).Wait();
        Console.WriteLine("Function3 is completed");
    }
}

class Program
{
    static void Main()
    {
        Parallel.Invoke(
            () => Console.WriteLine($"Action 1 on thread {Task.CurrentId}"),
            () => Console.WriteLine($"Action 2 on thread {Task.CurrentId}"),
            () => Console.WriteLine($"Action 3 on thread {Task.CurrentId}")
        );
        Console.ReadKey();
    }
}
```

Parallel.Invoke

`Parallel.Invoke` allows you to execute multiple actions in parallel.

Parallel.Invoke - Example

```
using System;
using System.Threading.Tasks;

class Program
{
    static void Main()
    {
        Parallel.Invoke(
            () => Console.WriteLine($"Action 1 on thread {Task.CurrentId}"),
            () => Console.WriteLine($"Action 2 on thread {Task.CurrentId}"),
            () => Console.WriteLine($"Action 3 on thread {Task.CurrentId}")
        );
    }
}
```

```
    }  
}
```

Explanation:

This code executes three actions in parallel. Each action is a lambda expression that prints a message including the ID of the thread it's running on.

Simpler way:

Parallel.Invoke - Example2

Within `Parallel.Invoke`, you can define separate methods for each function and then call these methods within `Parallel.Invoke`. Here's the modified example:

```
using System;  
using System.Threading.Tasks;  
  
class Program  
{  
    static void Main()  
    {  
        // Run the functions in parallel  
        Console.WriteLine("Starting parallel functions.");  
        Parallel.Invoke(Function1, Function2, Function3);  
        Console.WriteLine("All parallel functions are completed.");  
    }  
  
    static void Function1()  
    {  
        Console.WriteLine("Function 1 is starting.");  
        Task.Delay(1000).Wait(); // Simulating work  
        Console.WriteLine("Function 1 is completed.");  
    }  
}
```

```
}

static void Function2()
{
    Console.WriteLine("Function 2 is starting.");
    Task.Delay(1000).Wait(); // Simulating work
    Console.WriteLine("Function 2 is completed.");
}

static void Function3()
{
    Console.WriteLine("Function 3 is starting.");
    Task.Delay(1000).Wait(); // Simulating work
    Console.WriteLine("Function 3 is completed.");
}

}
```

In this example:

1. Function1, Function2, and Function3 are defined as separate static methods in the Program class.
2. Each function simulates some work by calling Task.Delay.
3. Parallel.Invoke is used to execute these methods in parallel. The methods are passed as arguments to Parallel.Invoke without using lambda expressions.
4. The program prints messages to the console indicating the start and completion of each function, as well as the overall completion of all functions.

Quiz4

What is the primary purpose of the Parallel class in C#?

To handle network operations asynchronously.

To execute multiple actions or iterations in parallel.

To manage database connections.

To serialize operations for thread safety.

In which library is the Parallel class located in C#?

System.Linq

System.Collections

System.Threading.Tasks

System.IO

What should be considered when using parallel constructs?

Thread safety

Overhead

I/O Bound vs CPU Bound

All of the above

Which method of the Parallel class is used to execute a for loop in parallel?

Parallel.For

Parallel.ForEach

Parallel.Invoke

Parallel.While

Which method would you use to parallelize a foreach loop?

Parallel.For

Parallel.ForEach

Parallel.Invoke

Parallel.Run

Parallel.Invoke is used for:

Invoking a method asynchronously.

Running a set of actions in parallel.

Initializing a new parallel data structure.

Creating a new thread.

Which of the following methods is NOT part of the Parallel class?

Parallel.For

Parallel.ForEach

Parallel.Invoke

Parallel.Execute

When using Parallel.For or Parallel.ForEach, what should you ensure about your code?

It should be written in a functional style.

It must connect to a database.

It should be thread-safe.

It must be executed on the main application thread.

What is a potential downside of using parallel operations with the Parallel class?

Decreased code readability.

Additional overhead in certain scenarios.

Inability to handle exceptions.

Limited to only two parallel operations at a time.

The Parallel class is best suited for which type of operations?

I/O-bound operations.

CPU-bound operations.

Network operations.

Single-threaded operations.

What should be considered when using the Parallel class for parallel programming?

The color scheme of the user interface.

Thread safety and nature of the tasks.

The brand of the server hardware.

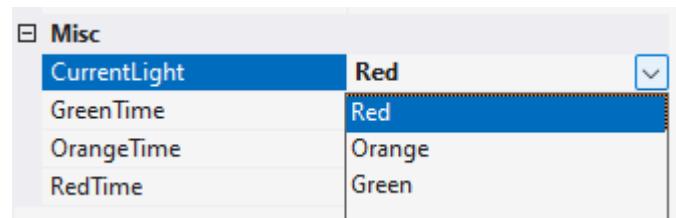
The programming language used.

Traffic Light Project Idea (Traffic Light User Control)

بتعمل ال control ده

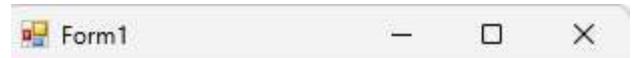


ولما تروح عال properties بتكون عامل ال properties دي
من هنا تقدر تغير اللون ال default وبالتالي يتغير النور بتاع ال control

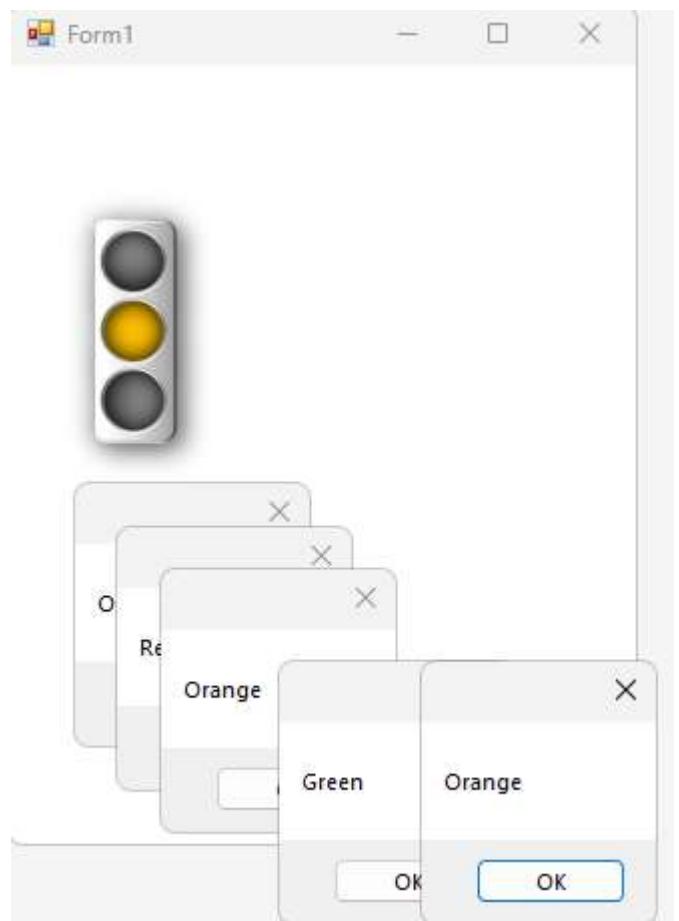


ومن هنا تقدر تتحكم في الوقت اللي كل لون بيظهر فيه

GreenTime	6
OrangeTime	5
RedTime	10



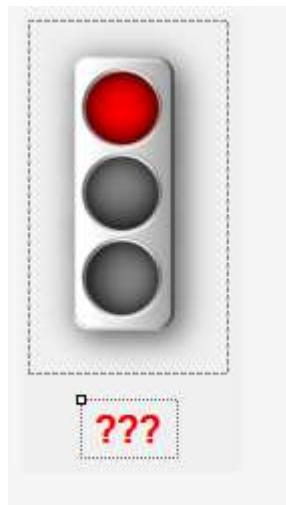
وبعد ما يخلص كل لون مده بطلعلك رسالة



تعالي نعملها واحده واحده

اول حاجه هنعملها الديزain

هنعمل control جديد ونحط فيه timer و label و picture box
وهنخلي ال بتاعت ال timer تكون قيمتها ب 1000 اللي هيا تعادل ثانية واحدة



تعالي بقى نخش للكود

اول حاجه محتاجين نعمل enum عشان نقدر نحدد نوع اللون بتاع الاشاره بسهوله
ومحتاجين نعمل متغير نخزن فيه اللون الحالى اللي ظاهر على الشاشه وكمان متغيرات بنخزن فيها القيم
الافتراضيه للوقت بتاع كل لون ومتغير تاني هنسخدمه كعداد

```
public enum enTrafficLight {Red=0,Orange=1,Green=2}
private enTrafficLight _CurrentLight = enTrafficLight.Red;

private int _RedLightDuration=3;
private int _OrangeLightDuration = 5;
private int _GreenLightDuration = 10;
private int _CurrentCountDownValue;
```

هنعمل متغيرات تانيه تتحكم في المتغيرات دي بحيث اننا نخلي اليوزر يقدر يعملها initialization من
بره

```
public int RedLightDuration {
    get { return this._RedLightDuration; }
    set { this._RedLightDuration = value; }
}
public int OrangeLightDuration
{
    get { return this._OrangeLightDuration; }
    set { this._OrangeLightDuration = value; }
}
public int GreenLightDuration
{
    get { return this._GreenLightDuration; }
    set { this._GreenLightDuration = value; }
}
```

دي method من خلالها اقدر اجيب ال current time اللي هو الوقت بتاع اللون الحالي

```
public int GetcurrentTime() {  
    switch (this._CurrentLight) {  
        case enTrafficLight.Red:  
            return this.RedLightDuration;  
            break;  
        case enTrafficLight.Orange:  
            return this.OrangeLightDuration;  
            break;  
        case enTrafficLight.Green:  
            return this.GreenLightDuration;  
            break;  
        default: return this.RedLightDuration;  
    }  
}
```

ودول 2 من خلالهم اقدر اشغل ال timer واوشه

```
public void Start() {  
    this._CurrentCountDownValue = GetcurrentTime();  
    timer1.Start();  
}  
  
public void Stop() {  
    timer1.Stop();  
}
```

وهنا method بغير منها اللون

بتغير الصوره المعروضه وتتغير الوقت الحالي والرقم اللي في ال label ويتشغل event

```
private void ChangeLight() {  
  
    switch (this._CurrentLight) {  
        case enTrafficLight.Red:  
  
            this.CurrentLight = enTrafficLight.Orange;  
            this._CurrentCountDownValue = this.OrangeLightDuration;  
            lblTimer.Text = this._CurrentCountDownValue.ToString();  
            OrangeLightRaisedOn();  
            break;  
        case enTrafficLight.Orange:  
            this.CurrentLight = enTrafficLight.Green;  
            this._CurrentCountDownValue = this.GreenLightDuration;  
            lblTimer.Text = this._CurrentCountDownValue.ToString();  
            GreenLightRaisedOn();  
            break;  
        case enTrafficLight.Green:  
            this.CurrentLight = enTrafficLight.Red;  
            this._CurrentCountDownValue = this.RedLightDuration;  
            lblTimer.Text = this._CurrentCountDownValue.ToString();  
            RedLightRaisedOn();  
            break;  
    }  
}
```

```
}
```

هنا بشغل event على ال timer بقوله يحدت قيمة ال label ويشوف لو الوقت الحالي لايساوي صفر
يقل العداد بوحد ولو كده يغير اللون

فلا هبيجي يغير اللون هيرجع يعمل اختبار للرقم وكده كده مش هيف لاني ماقولتهوش يقف

```
private void timer1_Tick(object sender, EventArgs e)
{
    lblTimer.Text = this._CurrentCountDownValue.ToString();

    if (this._CurrentCountDownValue == 0) {
        ChangeLight();
    }
    else {
        -- this._CurrentCountDownValue;
    }
}
```

وبعدين نبيجي لـ events ودي امرها سهل
هنعمل كلاس بالحاجات اللي عايزين نبعتها

```
public class TrafficLightEventArgs : EventArgs {
    public enTrafficLight CurrentLight;
    public int LightDuration;

    public TrafficLightEventArgs(enTrafficLight CurrentLight, int LightDuration) {
        this.CurrentLight = CurrentLight;
        this.LightDuration = LightDuration;
    }
}
```

وبعدين هنعمل eventhandler من object

```
public event EventHandler<TrafficLightEventArgs> RedLightOn;
```

وبعدين ال methods اللي هنشغله

```
private void RaiseRedLightOn(TrafficLightEventArgs e) {
    RedLightOn?.Invoke(this, e);
}

private void RedLightRaisedOn() {

    RaiseRedLightOn(new TrafficLightEventArgs(enTrafficLight.Red, _RedLightDuration));
}
```

ونكرر نفس الكلام مع كل الألوان
وفي الفورم نفسه هنطلع رساله تظهر بعد ما كل لون يشتعل
وده كود ال control

```
using MyTrafficLights.Properties;
using System;
using System.Drawing;
using System.Windows.Forms;

namespace MyTrafficLights
{
    public partial class ctrlTrafficLights : UserControl
    {
        public ctrlTrafficLights()
        {
            InitializeComponent();
        }

        public enum enTrafficLight {Red=0,Orange=1,Green=2}
        private enTrafficLight _CurrentLight = enTrafficLight.Red;

        private int _RedLightDuration=3;
        private int _OrangeLightDuration = 5;
        private int _GreenLightDuration = 10;
        private int _CurrentCountDownValue;

        public int RedLightDuration {
            get { return this._RedLightDuration; }
            set { this._RedLightDuration = value; }
        }

        public int OrangeLightDuration
        {
            get { return this._OrangeLightDuration; }
            set { this._OrangeLightDuration = value; }
        }

        public int GreenLightDuration
        {
            get { return this._GreenLightDuration; }
            set { this._GreenLightDuration = value; }
        }

        public int GetcurrentTime() {
            switch (this._CurrentLight) {
                case enTrafficLight.Red:
                    return this.RedLightDuration;
                    break;
                case enTrafficLight.Orange:
                    return this.OrangeLightDuration;
                    break;
                case enTrafficLight.Green:
                    return this.GreenLightDuration;
                    break;
                default: return this.RedLightDuration;
            }
        }

        public void Start() {
            this._CurrentCountDownValue = GetcurrentTime();
            timer1.Start();
        }

        public void Stop() {
            timer1.Stop();
        }
    }
}
```

```

private void ChangeLight() {

    switch (this._CurrentLight) {
        case enTrafficLight.Red:

            this.CurrentLight = enTrafficLight.Orange;
            this._CurrentCountDownValue = this.OrangeLightDuration;
            lblTimer.Text = this._CurrentCountDownValue.ToString();
            OrangeLightRaisedOn();
            break;
        case enTrafficLight.Orange:
            this.CurrentLight = enTrafficLight.Green;
            this._CurrentCountDownValue = this.GreenLightDuration;
            lblTimer.Text = this._CurrentCountDownValue.ToString();
            GreenLightRaisedOn();
            break;
        case enTrafficLight.Green:
            this.CurrentLight = enTrafficLight.Red;
            this._CurrentCountDownValue = this.RedLightDuration;
            lblTimer.Text = this._CurrentCountDownValue.ToString();
            RedLightRaisedOn();
            break;
    }
}

private void timer1_Tick(object sender, EventArgs e)
{
    lblTimer.Text = this._CurrentCountDownValue.ToString();

    if (this._CurrentCountDownValue == 0) {
        ChangeLight();
    }
    else {
        -- this._CurrentCountDownValue;
    }
}

public class TrafficLightEventArgs : EventArgs {
    public enTrafficLight CurrentLight;
    public int LightDuration;

    public TrafficLightEventArgs(enTrafficLight CurrentLight, int LightDuration) {
        this.CurrentLight = CurrentLight;
        this.LightDuration = LightDuration;
    }
}

public event EventHandler<TrafficLightEventArgs> RedLightOn;

private void RaiseRedLightOn(TrafficLightEventArgs e) {
    RedLightOn?.Invoke(this, e);
}

private void RedLightRaisedOn() {
    RaiseRedLightOn(new TrafficLightEventArgs(enTrafficLight.Red, _RedLightDuration));
}

```

```

public event EventHandler<TrafficLightEventArgs> OrangeLightOn;
private void RaiseOrangeLightOn(TrafficLightEventArgs e) {
    OrangeLightOn?.Invoke(this, e);
}

private void OrangeLightRaisedOn() {
    RaiseOrangeLightOn(new TrafficLightEventArgs(enTrafficLight.Orange, _OrangeLightDuration));
}

public event EventHandler<TrafficLightEventArgs> GreenLightOn;
private void RaiseGreenLightOn(TrafficLightEventArgs e) {
    GreenLightOn?.Invoke(this, e);
}

private void GreenLightRaisedOn() {
    RaiseGreenLightOn(new TrafficLightEventArgs(enTrafficLight.Green, _GreenLightDuration));
}

public enTrafficLight CurrentLight
{
    get { return this._CurrentLight; }

    set
    {
        this._CurrentLight = value;
        switch (this._CurrentLight)
        {
            case enTrafficLight.Red:
                pbTrafficLight.Image = Resources.Red;
                lblTimer.ForeColor = Color.Red;
                break;
            case enTrafficLight.Orange:
                pbTrafficLight.Image = Resources.Orange;
                lblTimer.ForeColor = Color.Orange;
                break;
            case enTrafficLight.Green:
                pbTrafficLight.Image = Resources.Green;
                lblTimer.ForeColor = Color.Green;
                break;
            default:
                lblTimer.ForeColor = Color.Red;
                break;
        }
    }
}

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MyTrafficLights
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            ctrlTrafficLights1.Start();
        }

        private void ctrlTrafficLights1_GreenLightOn(object sender, ctrlTrafficLights.TrafficLightEventArgs e)
        {
            MessageBox.Show(e.CurrentLight.ToString());
        }

        private void ctrlTrafficLights1_OrangeLightOn(object sender, ctrlTrafficLights.TrafficLightEventArgs e)
        {
            MessageBox.Show(e.CurrentLight.ToString());
        }

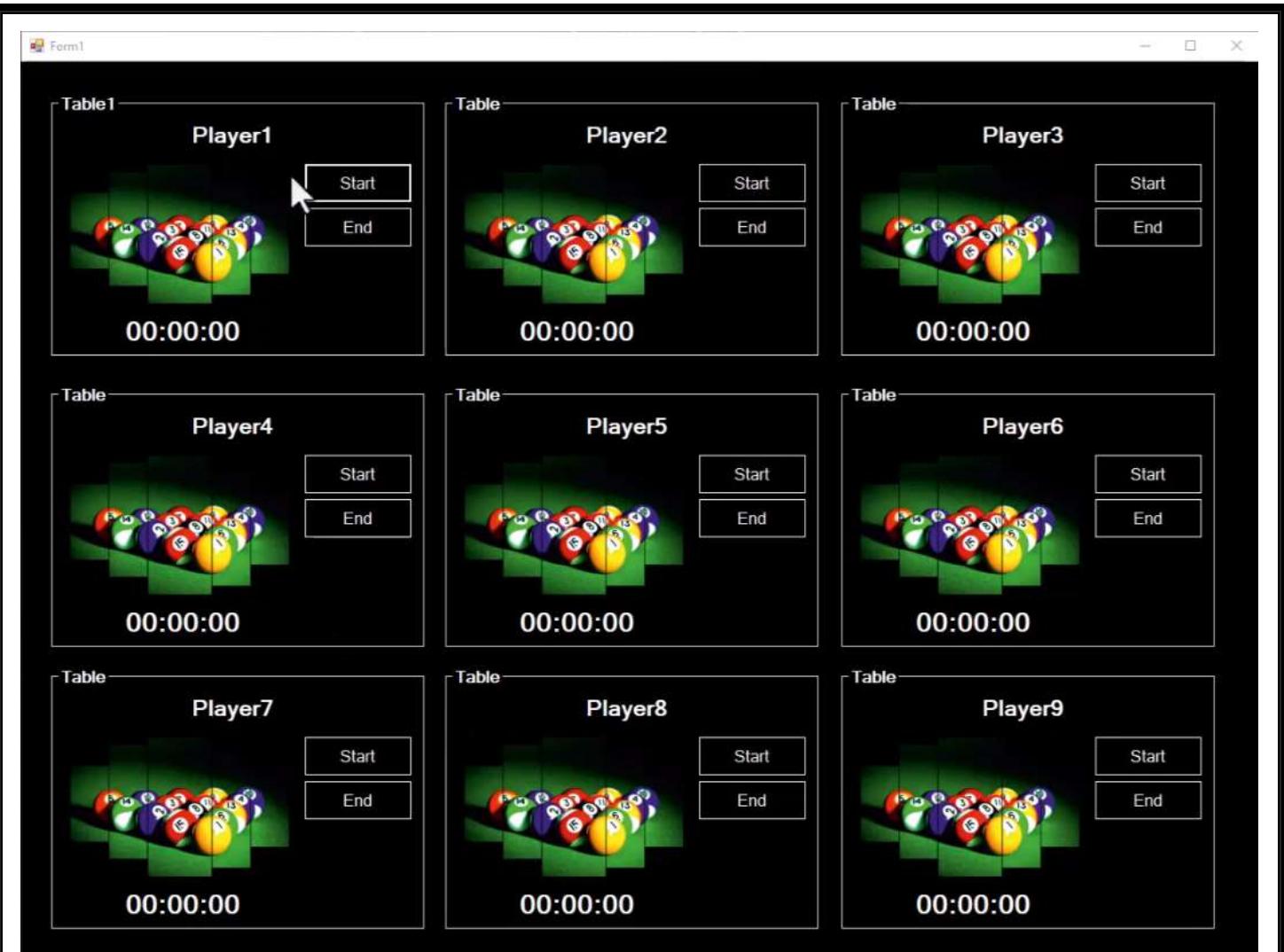
        private void ctrlTrafficLights1_RedLightOn(object sender, ctrlTrafficLights.TrafficLightEventArgs e)
        {
            MessageBox.Show(e.CurrentLight.ToString());
        }
    }
}

```

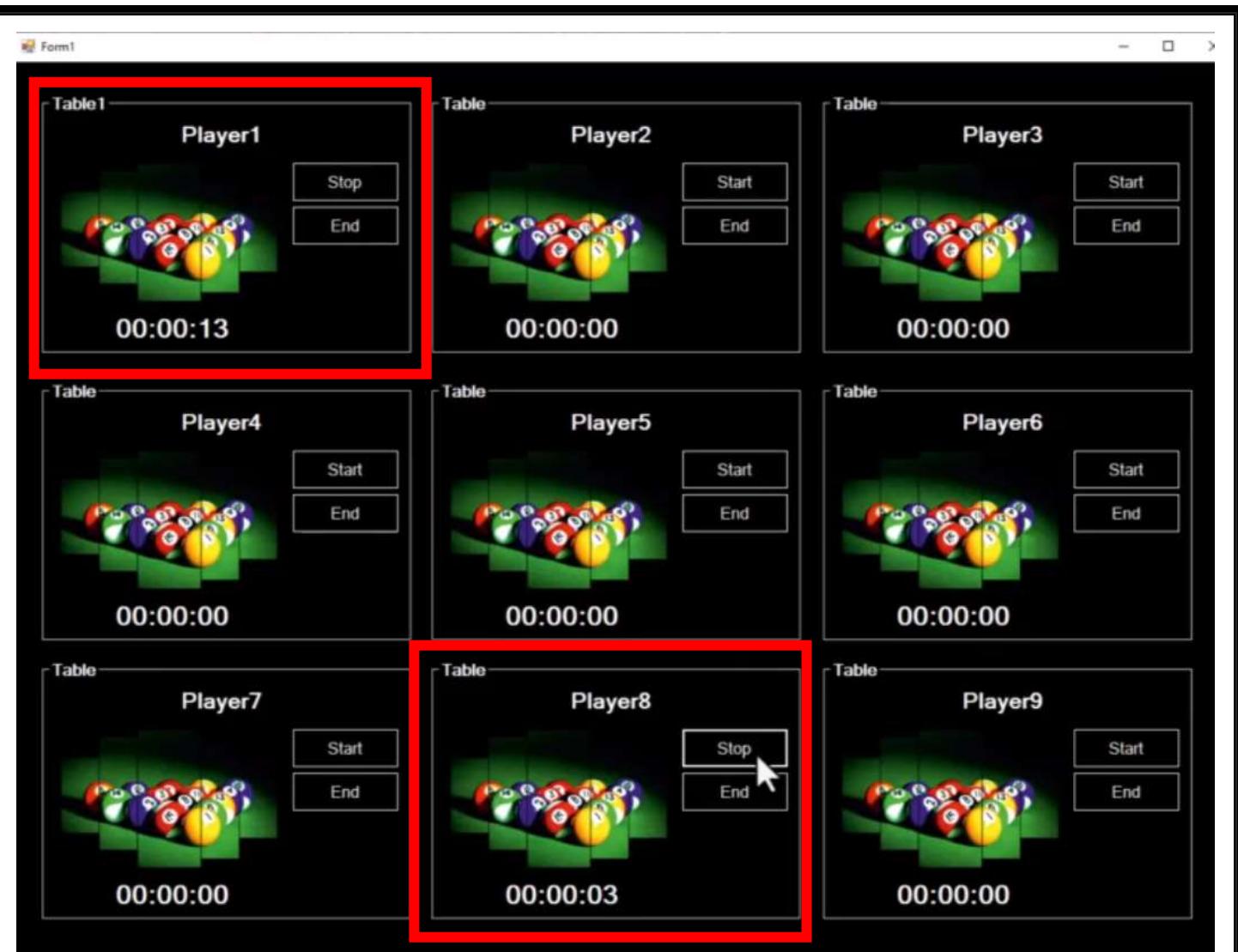
Pool Club Project Idea (Pool User Control)

هندلر مشروع ترايبيزات بلياردو

هندلر control واحد وبيتكرر معانا ٩ مرات في الفورم

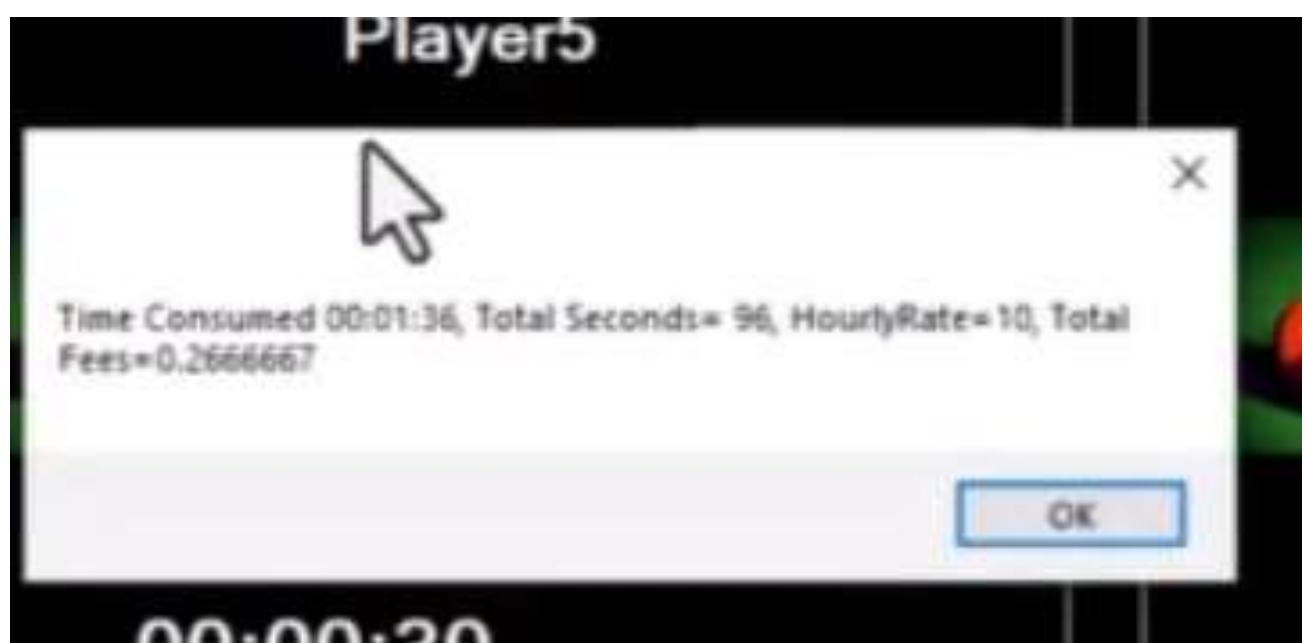


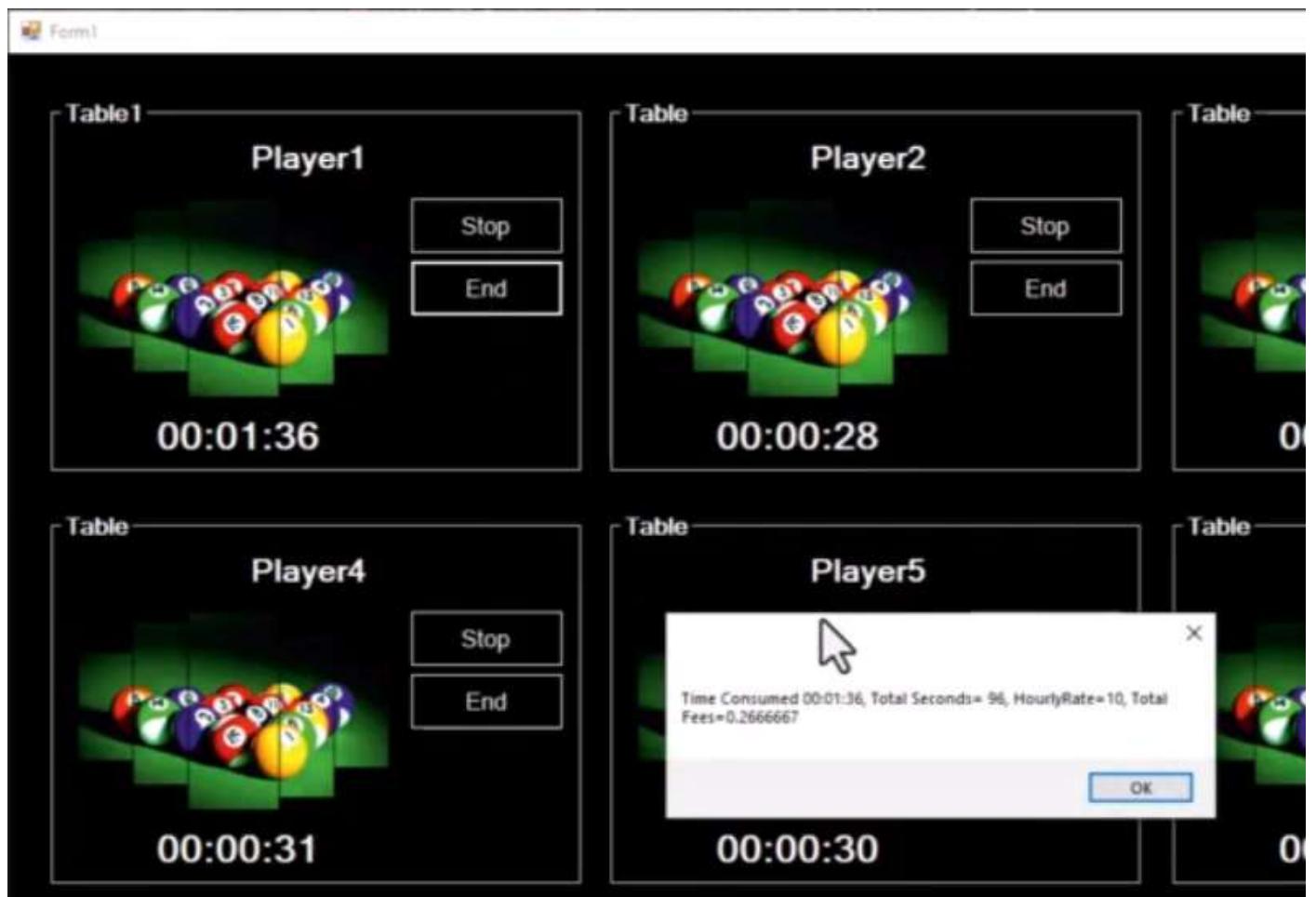
لما تدوس على start يشغل العداد



ولما تدوس على stop العداد يوقف بحيث يعمل pause

ولما تدوس على end يظهر لك الوقت وسرع الساعه والاجمالي وتدوس ok يصفر العداد وتقدر بعد كده
تعمل api ينور النور بتاع الترايبيزه ويقفلها وتخلی كل ترايبيزه بسرع معين وهكذا





فيه هنا عندك ال properties مش بتطلع في ال misc category هنا في لوحدها
و دي طرائقها بتحط اسم ال description category فوق كل property انت عاييزها

```
private string _TablePlayer = "Player";
[
Category("Pool Config"),
Description("The Player Name..")
]
```



وده الكود
نفس الفكرة بتاعت المثال اللي فات

```
using System;
using System.ComponentModel;
using System.Windows.Forms;

namespace _8Pool
{
    public partial class PoolTable : UserControl
    {

        public PoolTable()
        {
            InitializeComponent();
        }

        public class TableCompletedEventArgs : EventArgs
        {
            public string TimeText { get; }
            public int TimelnSeconds { get; }
            public float RatePerHour { get; }
            public float TotalFees { get; }

            public TableCompletedEventArgs(string TimeText, int TimelnSeconds, float RatePerHour, float TotalFees)
            {
                this.TimeText = TimeText;
                this.TimelnSeconds = TimelnSeconds;
                this.RatePerHour = RatePerHour;
                this.TotalFees = TotalFees;
            }
        }

        public event EventHandler<TableCompletedEventArgs> OnTableComplete;

        public void RaiseOnTableComplete(string TimeText, int TimelnSeconds, float RatePerHour, float TotalFees)
        {
            RaiseOnTableComplete(new TableCompletedEventArgs(TimeText, TimelnSeconds, RatePerHour, TotalFees));
        }

        protected virtual void RaiseOnTableComplete(TableCompletedEventArgs e)
        {
        }
    }
}
```

```
        OnTableComplete?.Invoke(this, e);
    }

    int _Seconds;

    // Private data member that backs the EndColor property.
    private string _TableTitle = "Table";

    // The Category attribute tells the designer to display
    // it in the Flash grouping.
    // The Description attribute provides a description of
    // the property.
    [
        Category("Pool Config"),
        Description("The table Name.")
    ]
    // The public property EndColor accesses endColor.
    public string TableTitle
    {
        get
        {
            return _TableTitle;
        }
        set
        {
            _TableTitle = value;

            grpTable.Text = value;

            // The Invalidate method calls the OnPaint method, which redraws
            // the control.
            Invalidate();
        }
    }

private string _TablePlayer = "Player";
[
    Category("Pool Config"),
    Description("The Player Name.")
]

public string TablePlayer
{
    get
    {
        return _TablePlayer;
    }
    set
    {
        _TablePlayer = value;

        lblName.Text = value;

        // The Invalidate method calls the OnPaint method, which redraws
        // the control.
        Invalidate();
    }
}
```

```
}

private float _HourlyRate= 10.00F;

[
Category("Pool Config"),
Description("Rate Per Hour.")
]
public float HourlyRate
{
    get
    {
        return _HourlyRate;
    }
    set
    {
        _HourlyRate = value;
    }
}

private void btnStartStop_Click(object sender, EventArgs e)
{
    if(btnStartStop.Text == "Start")
    {
        btnStartStop.Text = "Stop";
        timer1.Start();
    }
    else
    {
        btnStartStop.Text = "Start";
        timer1.Stop();
    }
}

private void timer1_Tick(object sender, EventArgs e)
{
    _Seconds++;

    TimeSpan time = TimeSpan.FromSeconds(_Seconds);
    string str = time.ToString(@"hh\:mm\:ss");
    lblTime.Text = str;
    lblTime.Refresh();
}

private void PoolTable_Load(object sender, EventArgs e)
{
    grpTable.Text = _TableTitle;
    lblName.Text=_TablePlayer;

}

private void lblTime_Click(object sender, EventArgs e)
{

}

private void btnEnd_Click(object sender, EventArgs e)
{
```

```

        timer1.Stop();
        float TotalFees = ((float)_Seconds / 60 / 60) * _HourlyRate;
        RaiseOnTableComplete(lblTime.Text, _Seconds, _HourlyRate, TotalFees);
        grpTable.Text = "Table";
        lblName.Text = "Player";
        lblTime.Text = "00:00:00";
        btnStartStop.Text = "Start";
        _Seconds = 0;

    }

private void toolStripTextBox1_TextChanged(object sender, EventArgs e)
{
    lblName.Text = toolStripTextBox1.Text;
}
}
}

```

```

using System;
using System.Windows.Forms;

namespace _8Pool
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {

        }

        private void poolTable4_Load(object sender, EventArgs e)
        {

        }

        private void poolTable_OnTableComplete(object sender, PoolTable.TableCompletedEventArgs e)
        {
            string TableResults = "";

            TableResults = "Time Consumed " + e.TimeText;
            TableResults = TableResults + ", Total Seconds= " + e.TimeInSeconds;
            TableResults = TableResults + ", HourlyRate=" + e.RatePerHour.ToString();
            TableResults = TableResults + ", Total Fees=" + e.TotalFees.ToString();

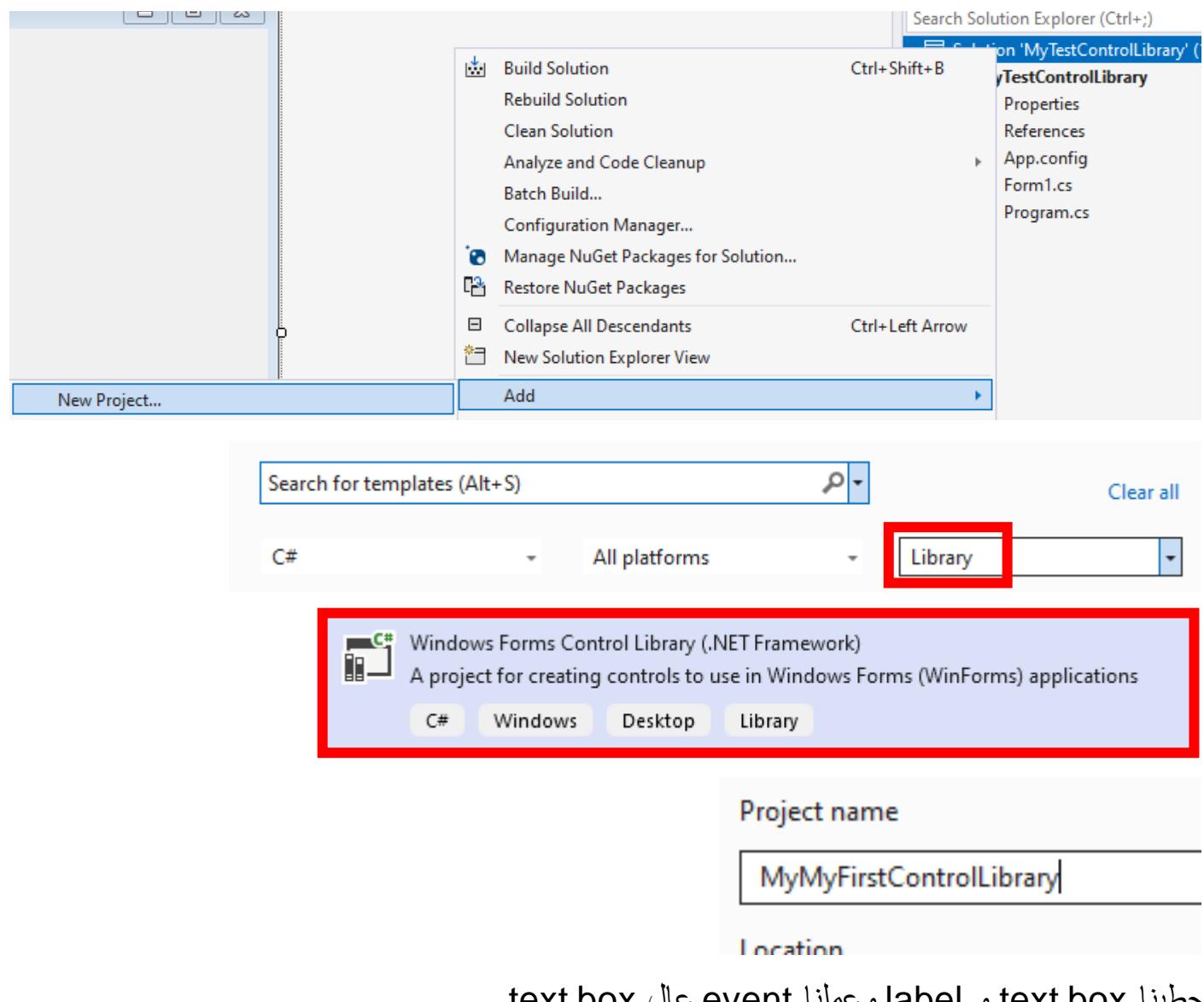
            MessageBox.Show(TableResults);
        }
    }
}

```

Introduction

الفكرة هنا اننا هنعمل control ونحطه في مكتبه بحيث نقدر نشغل بيه في أي مشروع والمكتبه هتكون بصيغة dll هنعمل مشروع عادي

وبعدين كليك يمين علي ال add>>new project solution ونختار



```
using System;
using System.Windows.Forms;

namespace MyMyFirstControlLibrary
{
    public partial class UserControl1 : UserControl
    {
        public UserControl1()
        {
            InitializeComponent();
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
        }
    }
}
```

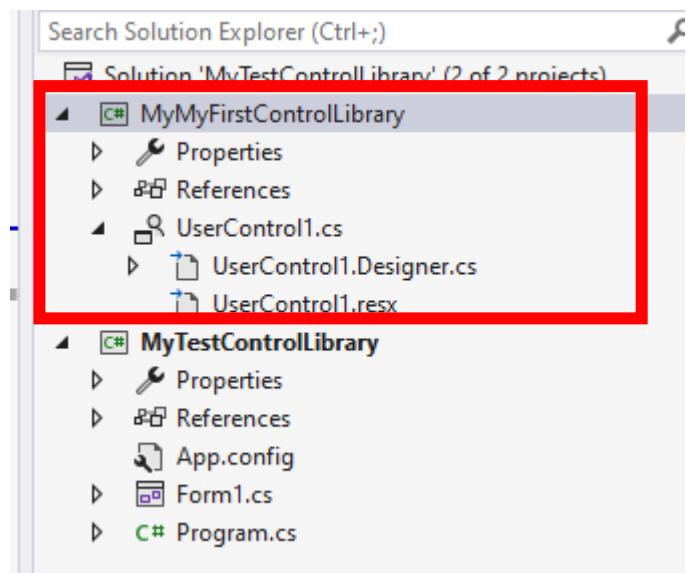
```

        label1.Text = textBox1.Text;
    }
}
}

```

دلوقي اصبح عندك control library

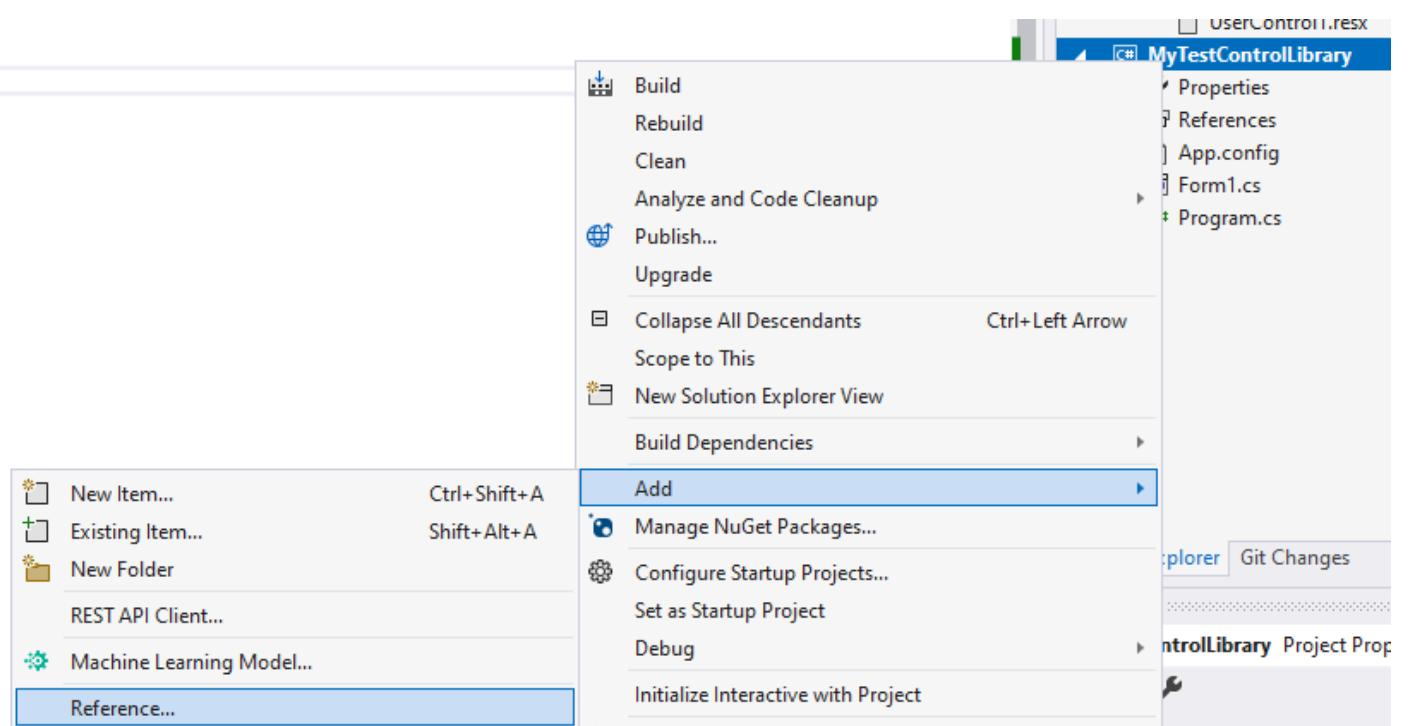
ازاي هتسخدمه في ال windows forms

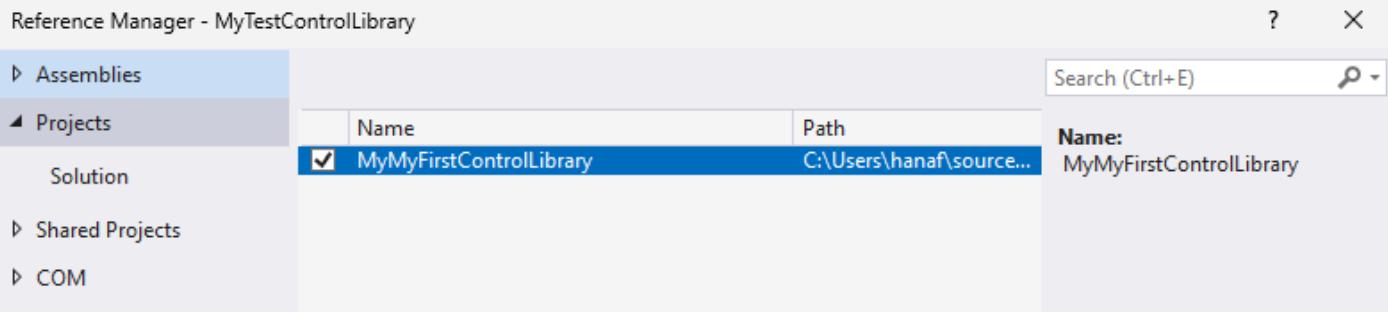


وبتضيفه لك reference

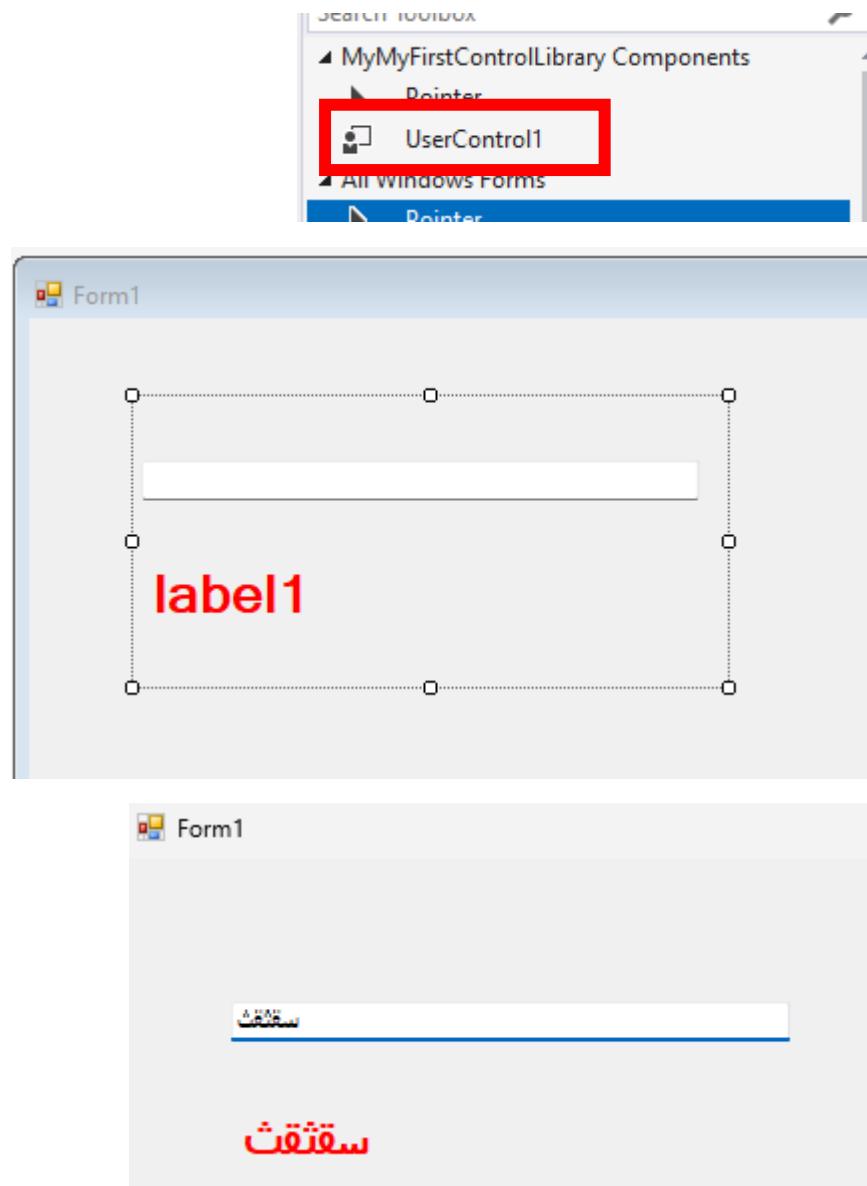
لو هو موجود ضمن ال project هتلقيه عندك

ولو مش من ضمن المشروع هتضيف ال dll بتاعه





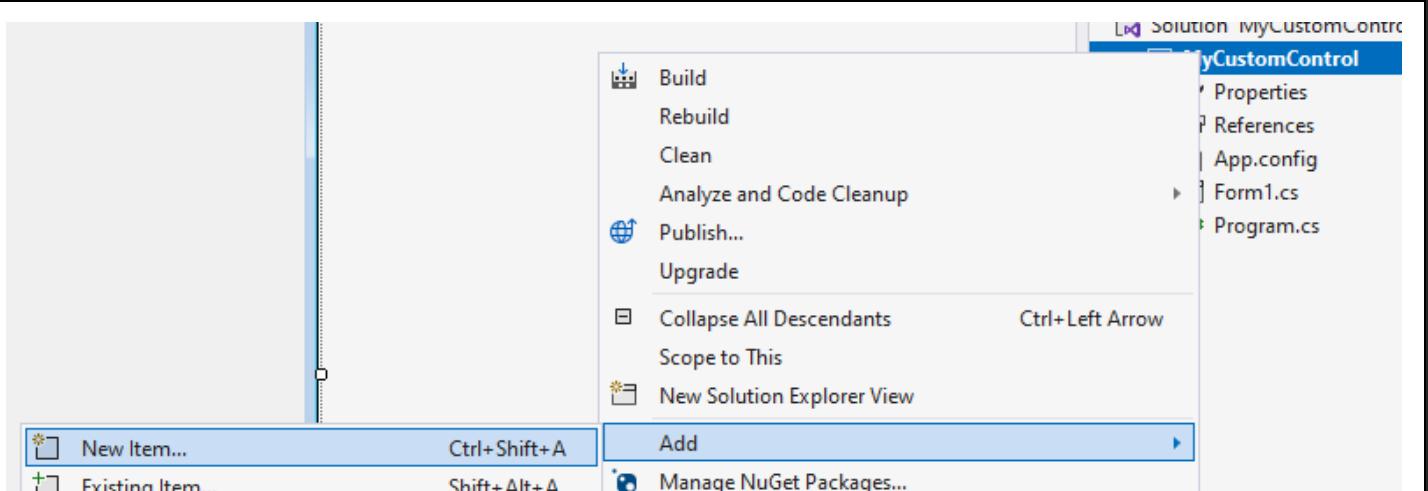
هتروح للفورم هتلacie في ال tool box اعمل بيه اللي انت عاوزه



Introduction to Custom Controls

هنا الفكرة انك عايز تأخذ ال controls الموجوده واللي معموله جاهزة زي ال textbox وتوترث منها بحيث تقدر تعدل عليها

هنعمل مشروع عادي وبعدين add>>new item



هطلعلوك الشاشه دي لأنك مانقدرش ترسم عليه

انت بتبنيه من الصفر او تورث من واحد موجود



احنا هنورث من ال textbox عشان نعمل عليه validation

هتدوس علي دي

ur class, [switch to code view.](#)

هيطلعلك الكلاس بيورث من control

احنا هنخليه يورث من textbox

```
using System.Windows.Forms;

namespace MyCustomControl
{
    public partial class MyCustomTextBox : TextBox
    {
        public MyCustomTextBox()
        {
```

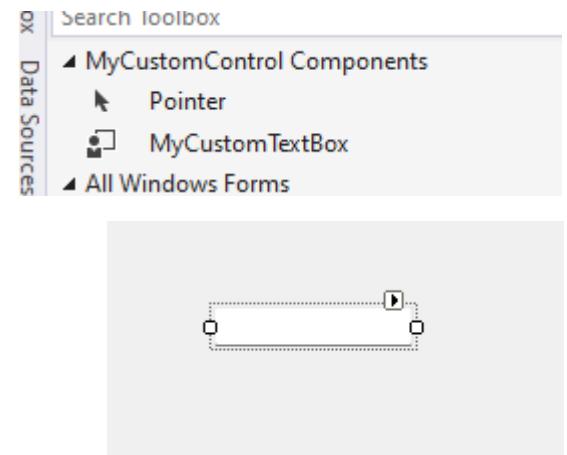
```

        InitializeComponent();
    }

    protected override void OnPaint(PaintEventArgs pe)
    {
        base.OnPaint(pe);
    }
}

```

اعمل build هتلacieh هنا



بعدين هنضيف 2 properties

```

public bool IsRequired { get; set; }

public enum InputTypeEnum { TextInput, NumberInput }

public InputTypeEnum InputType { get; set; } = InputTypeEnum.TextInput;

```

وده كود بسيط عشان يعمل validation

```

public bool IsRequired
{ get; set; }

public enum InputTypeEnum { TextInput, NumberInput }

public InputTypeEnum InputType
{ get; set; } = InputTypeEnum.TextInput;

private bool IsNumeric()
{
    string s = this.Text.Trim();
    foreach (char c in s)
    {
        if (!char.IsDigit(c) && c != '.')
        {
            return false;
        }
    }
    return true;
}

```

```
}

public Boolean IsValid()
{
    if (IsRequired)
    {
        if (this.Text.Trim().Length == 0)
            return false;
    }

    if (InputType == InputTypeEnum.NumberInput)
    {

        return IsNumeric();
    }

    return true;
}
```

وهنا بيستخدمه في الفورم

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show(myCustomTextBox1.IsValid().ToString());
}
```

Calling a SP AddNewPerson from C# Using ADO.NET

هنا عشان تقدر تتعامل مع ال ado.net مع ال stored procedures نفس اللي كنا بنعمله مع أي query بس باختلاف بسيط وهو سطر ال command

```
using SqlCommand command = new SqlCommand("SP_AddNewPerson", connection);
command.CommandType = CommandType.StoredProcedure;
```

بس وباقى المثال عادي

Calling a SP_AddNewPerson from C# Using ADO.NET

Introduction

In this lesson, we'll learn how to interact with a SQL Server database from a C# application using ADO.NET. We'll focus on calling a stored procedure to insert data into a database and retrieve a generated ID.

SQL Stored Procedure

First, we have a stored procedure `SP_AddNewPerson` in SQL Server:

```
CREATE PROCEDURE SP_AddNewPerson
    @FirstName NVARCHAR(100),
    @LastName NVARCHAR(100),
    @Email NVARCHAR(255),
    @NewPersonID INT OUTPUT
AS
BEGIN
    INSERT INTO People (FirstName, LastName, Email)
    VALUES (@FirstName, @LastName, @Email);

    SET @NewPersonID = SCOPE_IDENTITY();
END
```

This procedure inserts a new person into the `People` table and returns the new person's ID.

Calling the Stored Procedure from C#

To call this stored procedure from a C# application using ADO.NET, follow these steps:

Setting Up the Connection: First, establish a connection to your SQL Server database.

```
using System.Data.SqlClient;

string connectionString = "YourConnectionStringHere";
using SqlConnection connection = new SqlConnection(connectionString);
```

Creating the Command: Create a command object to represent the stored procedure.

```
using SqlCommand command = new SqlCommand("SP_AddNewPerson", connection);
command.CommandType = CommandType.StoredProcedure;
```

Adding Parameters: Add parameters to the command.

```
command.Parameters.AddWithValue("@FirstName", "John");
command.Parameters.AddWithValue("@LastName", "Doe");
command.Parameters.AddWithValue("@Email", "john.doe@example.com");
SqlParameter outputIdParam = new SqlParameter("@NewPersonID", SqlDbType.Int)
{
    Direction = ParameterDirection.Output
};
command.Parameters.Add(outputIdParam);
```

Executing the Command: Open the connection and execute the command.

```
connection.Open();
command.ExecuteNonQuery();
```

Retrieving the Output Parameter: After execution, retrieve the value of the output parameter.

```
int newPersonID = (int)command.Parameters["@NewPersonID"].Value;
Console.WriteLine($"New Person ID: {newPersonID}");
```

Closing the Connection: Finally, close the connection.

```
connection.Close();
```

Example

Here's a simple C# console application example that demonstrates the entire process:

```
using System;
using System.Data;
using System.Data.SqlClient;

class Program
{
    static void Main()
    {
        string connectionString = "YourConnectionStringHere";
        using SqlConnection connection = new SqlConnection(connectionString);
        using SqlCommand command = new SqlCommand("SP_AddNewPerson", connection);
        command.CommandType = CommandType.StoredProcedure;

        // Add parameters
        command.Parameters.AddWithValue("@FirstName", "John");
        command.Parameters.AddWithValue("@LastName", "Doe");
        command.Parameters.AddWithValue("@Email", "john.doe@example.com");
        SqlParameter outputIdParam = new SqlParameter("@NewPersonID", SqlDbType.Int)
        {
            Direction = ParameterDirection.Output
        };
        command.Parameters.Add(outputIdParam);

        // Execute
        connection.Open();
        command.ExecuteNonQuery();
```

```

        // Retrieve the ID of the new person
        int newPersonID = (int)command.Parameters["@NewPersonID"].Value;
        Console.WriteLine($"New Person ID: {newPersonID}");

        connection.Close();
    }
}

```

In this example, replace "YourConnectionStringHere" with your actual database connection string. This application will add a new person to the `People` table and print the new person's ID to the console.

Calling SP_CheckPersonExists from C# Using ADO.NET

وهذا مثال تاني

Calling SP_CheckPersonExists from C# Using ADO.NET

Introduction

In this lesson, you'll learn how to call a SQL Server stored procedure from a C# application using ADO.NET. The focus will be on executing a procedure that checks whether a record exists in a database based on a given ID.

SQL Stored Procedure

Firstly, we have a stored procedure named `SP_CheckPersonExists` in SQL Server:

```

CREATE PROCEDURE SP_CheckPersonExists
    @PersonID INT
AS
BEGIN
    IF EXISTS(SELECT * FROM People WHERE PersonID = @PersonID)
        RETURN 1; -- Person exists
    ELSE
        RETURN 0; -- Person does not exist
END

```

This procedure checks if a person exists in the `People` table using `PersonID`.

Calling the Stored Procedure from C#

To call this stored procedure from a C# application, follow these steps:

1. Setting Up the Connection:
2. Establish a connection to the SQL Server database.

```
using System.Data.SqlClient;

string connectionString = "YourConnectionStringHere";
using SqlConnection connection = new SqlConnection(connectionString);
```

1. Creating the Command:
2. Create a command object for the stored procedure.

```
using SqlCommand command = new SqlCommand("SP_CheckPersonExists", connection);
command.CommandType = CommandType.StoredProcedure;
```

1. Adding Parameters:
2. Add the necessary parameter to the command.

```
command.Parameters.AddWithValue("@PersonID", 123); // Example ID
```

1. Executing the Command:
2. Open the connection and execute the command. In this case, we'll use `ExecuteScalar()` since we're expecting a single value in return.

```
connection.Open();
int result = (int)command.ExecuteScalar();
```

1. Interpreting the Result:
2. The returned value indicates the existence of the person.

```
bool personExists = result == 1;  
Console.WriteLine($"Person exists: {personExists}");
```

1. Closing the Connection:
2. Close the connection to the database.

```
connection.Close();
```

Example

Here's a complete C# console application example demonstrating the entire process:

```
using System;  
using System.Data;  
using System.Data.SqlClient;  
  
  
class Program  
{  
    static void Main()  
    {  
        string connectionString = "YourConnectionStringHere";  
        using SqlConnection connection = new SqlConnection(connectionString);  
        using SqlCommand command = new SqlCommand("SP_CheckPersonExists", connection);  
        command.CommandType = CommandType.StoredProcedure;  
  
        // Add parameter  
        command.Parameters.AddWithValue("@PersonID", 123); // Example ID
```

```
// Execute  
connection.Open();  
int result = (int)command.ExecuteScalar();  
  
// Check if person exists  
bool personExists = result == 1;  
Console.WriteLine($"Person exists: {personExists}");  
  
connection.Close();  
}  
}
```

In this example, replace "YourConnectionStringHere" with your actual database connection string, and adjust the PersonID value accordingly. This application will check if a person with the specified ID exists in the People table and print the result to the console.

Message

الקורס اللي جاي بتابع ال data structure هيعلمهولنا بال c# عشان نستخدم ال tools اللي في ال C# ويقع العائق عليك في التطبيق

The End