

C# Programming

med folk från TGJ





Karlstad Innovation Park
(och Stora Enso)

The Great Journey

- Spelutvecklings-Community
- Håller evenemang i Kronoparken varje månad

- Spelutvecklare sitter där
- Folk- och Yrkehögskola inom spel där också
- TheGreatJourney.se

Filer:

för slides och kod

[https://
github.com/
ErikHogberg/
KGGCsharp](https://github.com/ErikHogberg/KGGCsharp)

Kalkylator2.cs har koden
från förra gången

— — —

Några bra länkar:

samma som förut

<https://learn.microsoft.com/en-us/dotnet/>

<https://try.dot.net/>

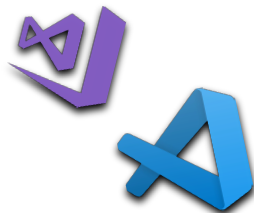
<https://dotnet.microsoft.com/en-us/platform/try-dotnet>

<https://dotnetfiddle.net/>

— — —

Dagens uppgifter

- ~~1. Gör ett "Hello, World!" program~~
- ~~2. Få programmet att säga hej till dig med ditt namn istället~~
 - ~~a. få den fråga efter ditt namn först~~
3. gör en miniräknare
 - a. få den att tolka åt oss vad man skrivit
 - b. få den att läsa hela rader matte
4. skapa en desktop-app som delar kod med console-appen



.NET

Dags att börja koda

~~<https://dotnetfiddle.net/>~~

Från och med nu krävs **Visual Studio**

Tolka vad man skrivit

— — —

Med hjälp av “TryParse” från förra gången så kan vi enkelt tolka varje svar istället för att fråga vad vi svarar varannan gång

Vi har redan fallet i “else” där vi ser att det inte är ett nummer och varnar användaren om det, så vi behöver bara flytta in hela andra frågans kod där i för att bli färdiga



```
if (askForNumber)
{
    var svar = Console.ReadLine();
    float number;
    if (float.TryParse(svar, out number))
    {
        switch (lastOperand)
        {
            case Operand.Plus:
                lastNumber = lastNumber + number;
                break;
            case Operand.Minus:
                lastNumber = lastNumber - number;
                break;
            case Operand.Multiply:
                lastNumber = lastNumber * number;
                break;
            case Operand.Divide:
                lastNumber = lastNumber / number;
                break;
            case Operand.None:
            default:
                break;
        }
    }
    else
    {
        Console.WriteLine("Inte ett nummer, försök igen");
        continue;
    }

    Console.WriteLine(lastNumber.ToString("0.00"));
}
else
{
    Console.WriteLine("frågar efter operand");
    var svar = Console.ReadLine();
    switch (svar)
    {
        case "+":
            lastOperand = Operand.Plus;
            break;
        case "-":
            lastOperand = Operand.Minus;
            break;
        case "*":
            lastOperand = Operand.Multiply;
            break;
        case "/":
            lastOperand = Operand.Divide;
            break;
        default:
            Console.WriteLine("Inte en operand");
            lastOperand = Operand.None;
            break;
    }
}
askForNumber = !askForNumber;
}
```

Tolka vad man skrivit

— — —

som bonus så behöver vi inte efteråt:

- “askForNumber”
- “if” fallen
- den andra **Console.ReadLine()**

```
static void Main(string[] args)
{
    Console.WriteLine("Börja skriva nummer eller operander");

    float lastNumber = 0;
    Operand lastOperand = Operand.Plus;
    while (true)
    {
        var svar = Console.ReadLine();
        float number;
        if (float.TryParse(svar, out number))
        {
            switch (lastOperand)
            {
                case Operand.Plus:
                    lastNumber = lastNumber + number;
                    break;
                case Operand.Minus:
                    lastNumber = lastNumber - number;
                    break;
                case Operand.Multiply:
                    lastNumber = lastNumber * number;
                    break;
                case Operand.Divide:
                    lastNumber = lastNumber / number;
                    break;
                case Operand.None:
                default:
                    break;
            }
        }
        else
        {
            switch (svar)
            {
                case "+":
                    lastOperand = Operand.Plus;
                    break;
                case "-":
                    lastOperand = Operand.Minus;
                    break;
                case "*":
                    lastOperand = Operand.Multiply;
                    break;
                case "/":
                    lastOperand = Operand.Divide;
                    break;
                default:
                    Console.WriteLine("Inte en operander");
                    lastOperand = Operand.None;
                    break;
            }
            continue;
        }
    }

    Console.WriteLine(lastNumber.ToString("0.00"));
}
```


Tolka hela rader

— — —

En “string” är en sträng av bokstäver, som namnet antyder

istället för att fråga ett nummer eller en operand i taget, så kan vi då istället låta den läsa hela raden genom att loopa igenom bokstäverna

```
foreach (char bokstav in svar)
{
    Console.WriteLine("bokstav: " + bokstav);

    float nummer;
    if (float.TryParse(bokstav.ToString(), out nummer))
    {
        switch (lastOperand)
```

Tolka hela rader

— — —

OBS:

vi introducerade nu en bugg!

Vi kan inte längre läsa nummer över 9!

Detta är för att vi läser en bokstav i taget, vilket betyder att vi läser även ett nummer i taget

```
foreach (char bokstav in svar)
{
    Console.WriteLine("bokstav: " + bokstav);

    float nummer;
    if (float.TryParse(bokstav.ToString(), out nummer))
    {
        switch (lastOperand)
```

Det finns flera sätt att lösa det, men vi kommer inte göra det idag :)



Class libraries

hur program delar kod



Typer av program

— — —

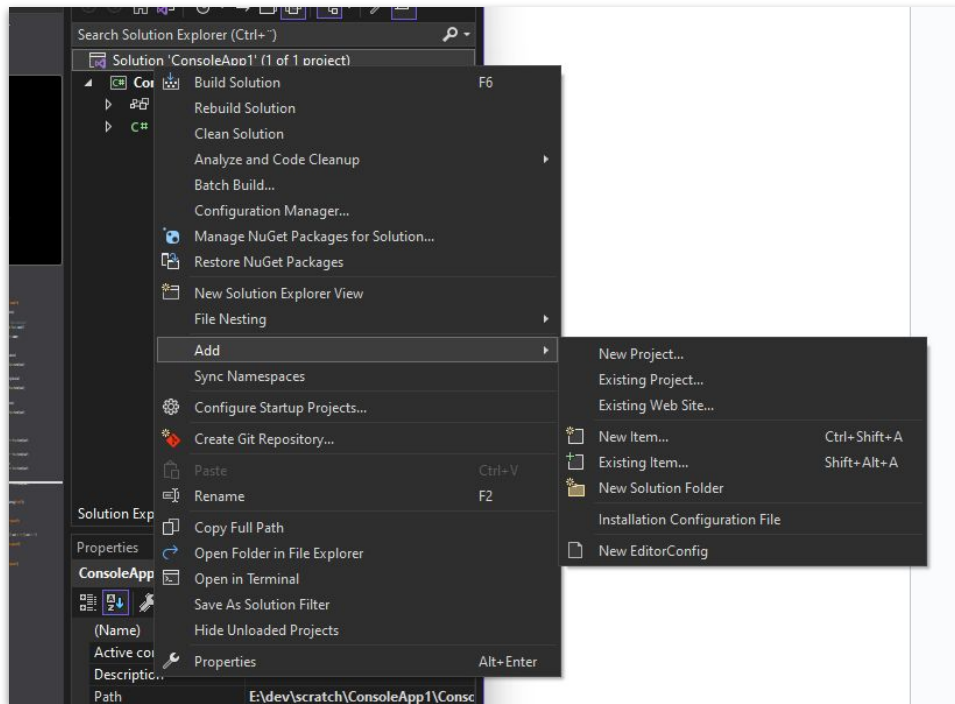
- Console app
 - program som bara visar text
 - bra för att snabbt göra små program
- **Class library** 
 - delar av program som inte går att köra
 - används för att dela kod mellan program
- **Desktop app** 
 - börjar från ett windows-fönster istället för en terminal/console
 - för att göra vanliga windows-program från grunden
- Scripts
 - kod som körs av andra program
 - t.ex. scripts i spelmotorer som Unity3D och Godot Engine

... och web apps,
Machine Learning,
Internet of Things,
Windows services,
Microsoft Azure,
och mycket mer

class libraries

— — —
nya projekt kan läggas till i
“Solution explorer”

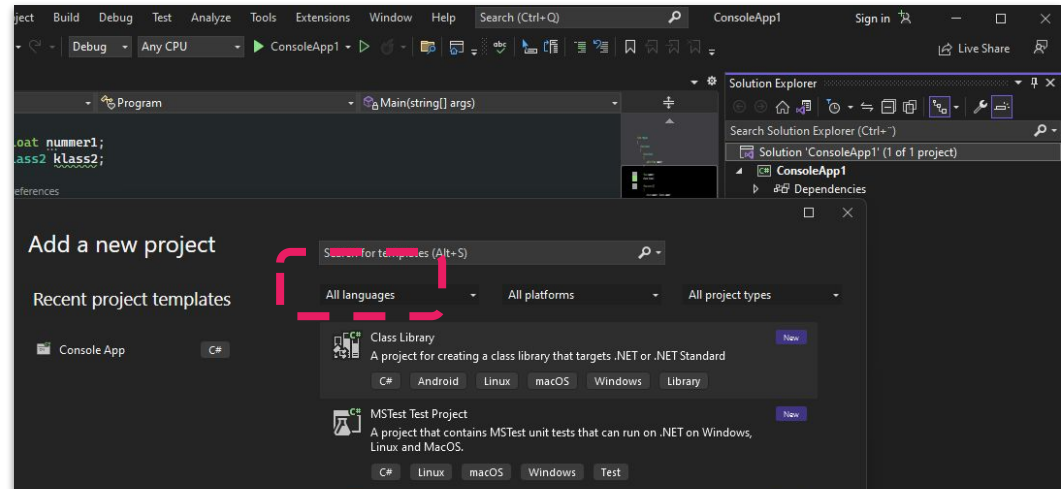
om den inte syns så hittas den
under “view”-menyn



— — —

Se till att det är rätt typ av projekt, det finns flera typer av Class Library

versioner som t.ex. med WPF eller andra språk än C# är fel



Typer

hur vet vi vad vi sparar?

- string
 - char?
- float
 - double?
- int
 - uint?
 - short?
 - long?
- bool
- enums
- “klasser”?

— — —

Klasser

- **Klass/Class**
 - **Basklass/Base class**
- **Metod/Method**
 - en typ av **Funktion/Function**
- (Funktions/Metods-)anrop
- **Scope**

Ett exempel från Unity:

```
public class hello : MonoBehaviour
{
    void Start()
    {
        Debug.Log("Hello, World!");
    }
}
```


Klasser

— — —

Klasser är sätt att samla ihop inte bara variabler, utan även funktioner/metoder

Dom kan sedan användas som typer istället för nummer och text

Ett exempel på klasser vi redan använt är “Console”

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

```
int counter = 0;  
0 references  
void Two()  
{  
    counter++;  
    Console.WriteLine("Hello, World! " + counter);  
}
```

Klasser

— — —

Vi kan skapa klasser på samma platser som man kan skapa enums

till och med inuti andra klasser!

för att lägga till variabler i klassen så skriver man dom utanför funktioner, helst längst upp i klassen

(om inte en annan klass ligger högst)

```
class Program
{
    0 references
    class Klass1
    {
        1 reference
        class Klass2
        {
            public float nummer2;
        }

        float nummer1;
        Klass2 klass2;

        0 references
        float metod1()
        {
            return nummer1 + klass2.nummer2;
        }
    }
}
```

“static” Klasser

— — —

klasser kan vara “statiska”

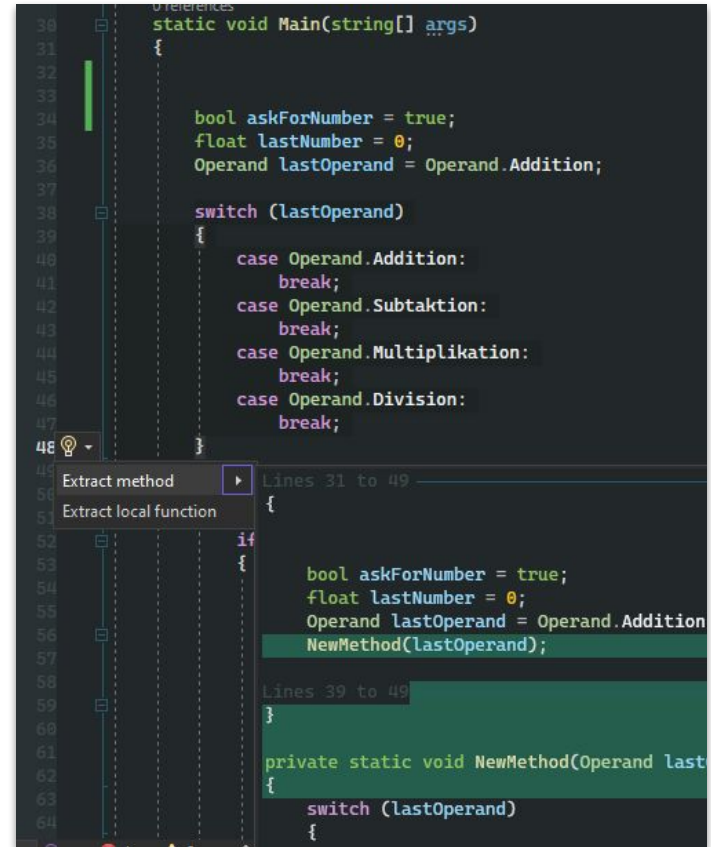
Det betyder att de inte kan användas som typer, endast samlingar av funktioner (ej metoder)

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

```
0 references  
class Klass1  
{  
    1 reference  
    class Klass2  
    {  
        public float nummer2;  
    }  
  
    float nummer1;  
    Klass2 klass2;  
}
```

Refactoring

gör om, gör rätt



```
30 0 references
31 static void Main(string[] args)
32 {
33
34     bool askForNumber = true;
35     float lastNumber = 0;
36     Operand lastOperand = Operand.Addition;
37
38     switch (lastOperand)
39     {
40     case Operand.Addition:
41         break;
42     case Operand.Subtraktion:
43         break;
44     case Operand.Multiplikation:
45         break;
46     case Operand.Division:
47         break;
48     }
49 }
```

Extract method
Extract local function

Lines 31 to 49

```
{
    bool askForNumber = true;
    float lastNumber = 0;
    Operand lastOperand = Operand.Addition;
    NewMethod(lastOperand);
}
```

Lines 39 to 49

```
}
private static void NewMethod(Operand last
{
    switch (lastOperand)
    {
```

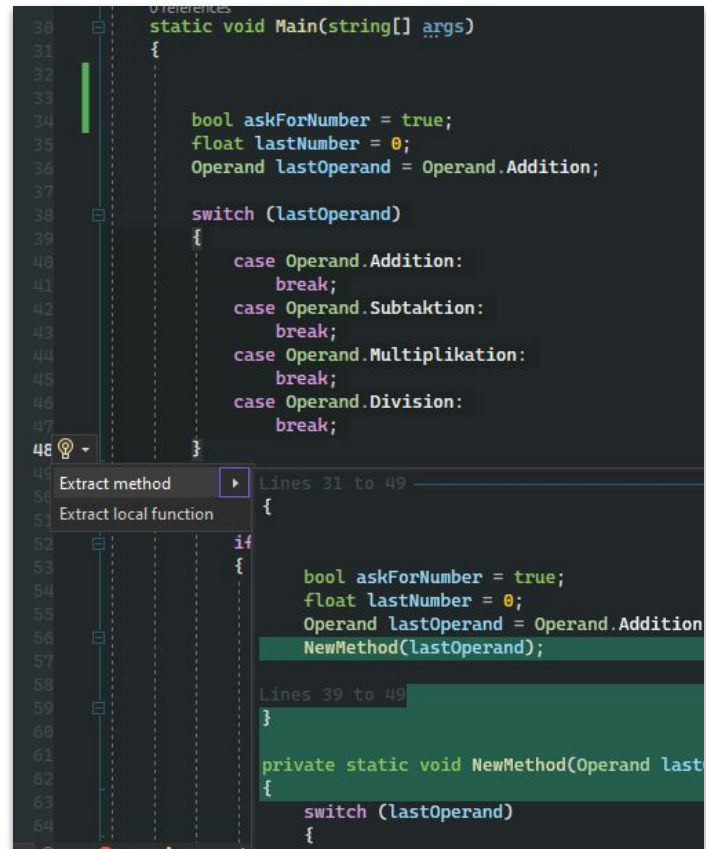
https://en.wikipedia.org/wiki/Code_refactoring

Refactoring

“Refaktoring” är en benämning på ett specifikt koncept inom all programmering, inte bara C#

Det syftar på att göra om kod (ofta bara att flytta på den) så att den är lättare att hantera, utan att ändra vad som händer när den körs

Visual Studio och många andra redigerare har inbyggda sätt att snabbt göra många vanliga “omstruktureringar”



```
30 static void Main(string[] args)
31 {
32
33
34     bool askForNumber = true;
35     float lastNumber = 0;
36     Operand lastOperand = Operand.Addition;
37
38     switch (lastOperand)
39     {
40     case Operand.Addition:
41         break;
42     case Operand.Subtaktion:
43         break;
44     case Operand.Multiplikation:
45         break;
46     case Operand.Division:
47         break;
48     }
49 }
```

Extract method
Extract local function

```
Lines 31 to 49
{
    bool askForNumber = true;
    float lastNumber = 0;
    Operand lastOperand = Operand.Addition;
    NewMethod(lastOperand);
}

Lines 39 to 49
}

private static void NewMethod(Operand last)
{
    switch (lastOperand)
    {
    {
```

Refactoring

Vi kan korta ned de 2 “switch” delarna av kalkylatorn mycket genom att byta till “switch expressions”

Om vi ser till att alla “case” i switchen skriver till samma variabel så kan Visual Studio refaktorisera åt er

lägg till

ta bort

```
foreach (char bokstav in svar)
{
    float number;
    if (float.TryParse(bokstav.ToString(), out number))
    {
        switch (lastOperand)
        {
            case Operand.Plus:
                lastNumber = lastNumber + number;
                break;
            case Operand.Minus:
                lastNumber = lastNumber - number;
                break;
            case Operand.Multiply:
                lastNumber = lastNumber * number; break;
            case Operand.Divide:
                lastNumber = lastNumber / number; break;
            case Operand.None:
            default:
                lastNumber = lastNumber;
                break;
        }
    }
    else
    {
        switch (bokstav.ToString())
        {
            case "+":
                lastOperand = Operand.Plus;
                break;
            case "-":
                lastOperand = Operand.Minus;
                break;
            case "*":
                lastOperand = Operand.Multiply;
                break;
            case "/":
                lastOperand = Operand.Divide;
                break;
            default:
                Console.WriteLine("Inte en operand");
                lastOperand = Operand.None;
                break;
        }
        continue;
    }
}
```

Refactoring

ctrl+. eller högerklicka på switchen och välj “Quick actions and refactorings”

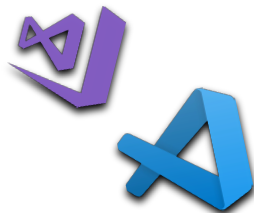
```
foreach (char bokstav in svar)
{
    float number;
    if (float.TryParse(bokstav.ToString(), out number))
    {
        switch (lastOperand)
        {
            case Operand.Plus:
                lastNumber = lastNumber + number;
                break;
            case Operand.Minus:
                lastNumber = lastNumber - number;
                break;
            case Operand.Multiply:
                lastNumber = lastNumber * number; break;
            case Operand.Divide:
                lastNumber = lastNumber / number; break;
            case Operand.None:
                lastNumber = lastNumber;
                break;
        }
    }
    else
    {
        switch (bokstav.ToString())
        {
            case "+":
                lastOperand = Operand.Plus;
                break;
            case "-":
                lastOperand = Operand.Minus;
                break;
            case "*":
                lastOperand = Operand.Multiply;
                break;
            case "/":
                lastOperand = Operand.Divide;
                break;
            default:
                lastOperand = Operand.None;
                break;
        }
        continue;
    }
}
```



```
foreach (char bokstav in svar)
{
    float number;
    if (float.TryParse(bokstav.ToString(), out number))
    {
        lastNumber = lastOperand switch
        {
            Operand.Plus => lastNumber + number,
            Operand.Minus => lastNumber - number,
            Operand.Multiply => lastNumber * number,
            Operand.Divide => lastNumber / number,
            _ => lastNumber,
        };
    }
    else
    {
        lastOperand = bokstav.ToString() switch
        {
            "+" => Operand.Plus,
            "-" => Operand.Minus,
            "*" => Operand.Multiply,
            "/" => Operand.Divide,
            _ => Operand.None,
        };
        continue;
    }
}
```

Dagens uppgifter

- ~~1. Gör ett "Hello, World!" program~~
- ~~2. Få programmet att säga hej till dig med ditt namn istället~~
 - ~~a. få den fråga efter ditt namn först~~
- ~~3. gör en miniräknare~~
 - ~~a. få den att tolka åt oss vad man skrivit~~
 - ~~b. få den att läsa hela rader matte~~
4. skapa en desktop-app som **delar kod** med console-appen



.NET

Refactoring

— — —

Nästa steg är att flytta delar av koden till **klassbiblioteket** vi skapade

Först måste vi skapa en funktion som vi kan flytta

Markera loopen som tolkar texten med matte, se till att få med alla variabler loopen behöver, flytta ned variabler om det behövs

tryck ctrl+. för att få upp menyn för att skapa en funktion av den (“Extract method”)

gör funktionen statisk genom att skriva “static” före typen (float) och namnet

```
private static float NewMethod(string svar)
{
    float lastNumber = 0;
    Operand lastOperand = Operand.Plus;
    foreach (char bokstav in svar)
    {
        float number;
        if (float.TryParse(bokstav.ToString(), out number))
        {
            lastNumber = lastOperand switch
            {
                Operand.Plus => lastNumber + number,
                Operand.Minus => lastNumber - number,
                Operand.Multiply => lastNumber * number,
                Operand.Divide => lastNumber / number,
                _ => lastNumber,
            };
        }
        else
        {
            lastOperand = bokstav.ToString() switch
            {
                "+" => Operand.Plus,
                "-" => Operand.Minus,
                "*" => Operand.Multiply,
                "/" => Operand.Divide,
                _ => Operand.None,
            };
            continue;
        }
    }

    return lastNumber;
}
```

Refactoring

```
11 references
class Program
{
    enum Operand
    {
        None,
        Plus,
        Minus,
        Multiply,
        Divide,
    }

    0 references
    static void Main(string[] args)
    {
        while (true)
        {
            Console.WriteLine("Skriv din mattefråga");
            var svar = Console.ReadLine();

            if (svar == null) { continue; }
            if (svar == "exit")
            {
                break;
            }

            float lastNumber = NewMethod(svar);

            Console.WriteLine(lastNumber.ToString("0.00"));
        }
    }
}
```

```
1 reference
private static float NewMethod(string svar)
{
    float lastNumber = 0;
    Operand lastOperand = Operand.Plus;
    foreach (char bokstav in svar)
    {
        float number;
        if (float.TryParse(bokstav.ToString(), out number))
        {
            lastNumber = lastOperand switch
            {
                Operand.Plus => lastNumber + number,
                Operand.Minus => lastNumber - number,
                Operand.Multiply => lastNumber * number,
                Operand.Divide => lastNumber / number,
                _ => lastNumber,
            };
        }
        else
        {
            lastOperand = bokstav.ToString() switch
            {
                "+" => Operand.Plus,
                "-" => Operand.Minus,
                "*" => Operand.Multiply,
                "/" => Operand.Divide,
                _ => Operand.None,
            };
            continue;
        }
    }

    return lastNumber;
}
```

Refactoring

Flytta funktionen in i det “class library”
som skapades tidigare

flytta även “Operand” enum:en

gör både funktionen och enumen “public”
genom att skriva public framför enum och
istället för “private”

```
namespace ClassLibrary1
{
    0 references
    public class Class1
    {
        11 references
        public enum Operand
        {
            None,
            Plus,
            Minus,
            Multiply,
            Divide,
        }

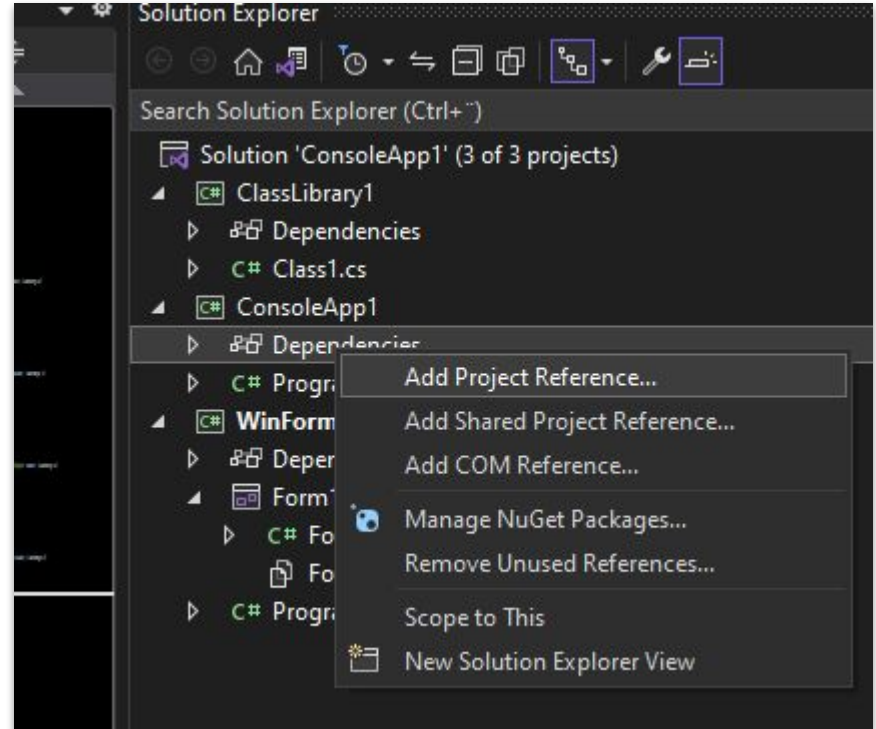
        0 references
        public static float NewMethod(string svar)
        {
            float lastNumber = 0;
            Operand lastOperand = Operand.Plus;
            foreach (char bokstav in svar)
            {
                float number;
                if (float.TryParse(bokstav.ToString(), out number))
                {
                    lastNumber = lastOperand switch
                    {
                        Operand.Plus => lastNumber + number,
                        Operand.Minus => lastNumber - number,
                        Operand.Multiply => lastNumber * number,
                        Operand.Divide => lastNumber / number,
                        _ => lastNumber,
                    };
                }
                else
                {
                    lastOperand = bokstav.ToString() switch
                    {
                        "+" => Operand.Plus,
                        "-" => Operand.Minus,
                        "*" => Operand.Multiply,
                        "/" => Operand.Divide,
                        _ => Operand.None,
                    };
                    continue;
                }
            }

            return lastNumber;
        }
    }
}
```

Refactoring

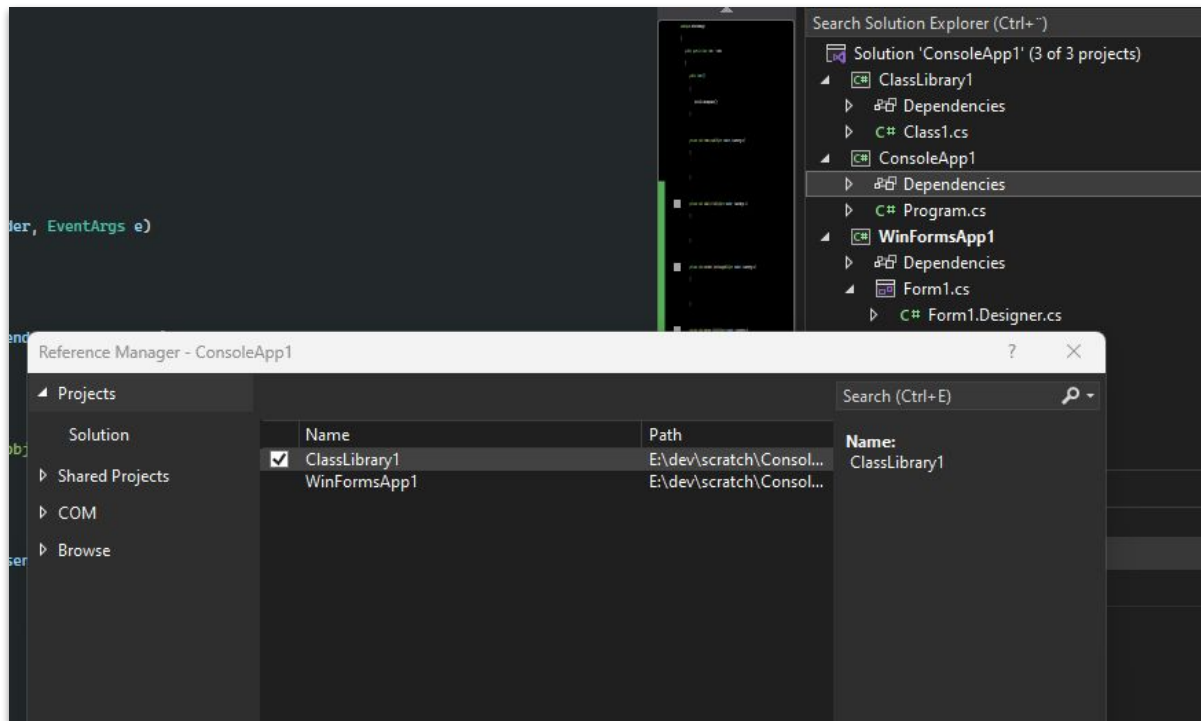
— — —

för att kunna se koden i klassbiblioteket så krävs först en referens till den i alla projekt som vill använda den



Refactoring

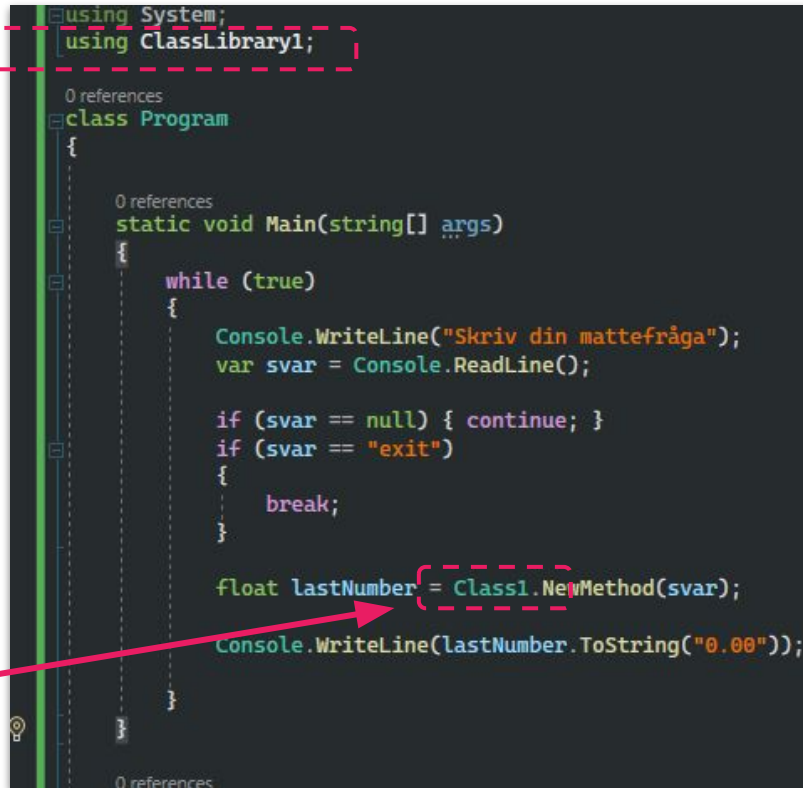
bocka
klassbibliotekets
namn och spara



Refactoring

— — —
Lägg till en “using statement” längst upp med klassbibliotekets namn för att hämta innehållet

efter det är det bara att förtydliga var funktionen ligger där i genom att skriva namnet på klassen vi la den i



The screenshot shows a C# code file in Visual Studio. At the top, there are two 'using' statements: 'using System;' and 'using ClassLibrary1;'. The 'using ClassLibrary1;' line is enclosed in a red dashed box, and a red arrow points from the text 'Lägg till en “using statement”...' to it. Below these is a class definition 'class Program' with a 'Main' method. Inside the 'Main' method, there is a 'while' loop. After the loop, there is a line 'float lastNumber = Class1.NewMethod(svar);'. This line is also enclosed in a red dashed box, and a red arrow points from the text 'att förtydliga var funktionen ligger där i' to it. The code continues with 'Console.WriteLine(lastNumber.ToString("0.00"));'. The background is dark, and the text is in a light color.

```
using System;
using ClassLibrary1;

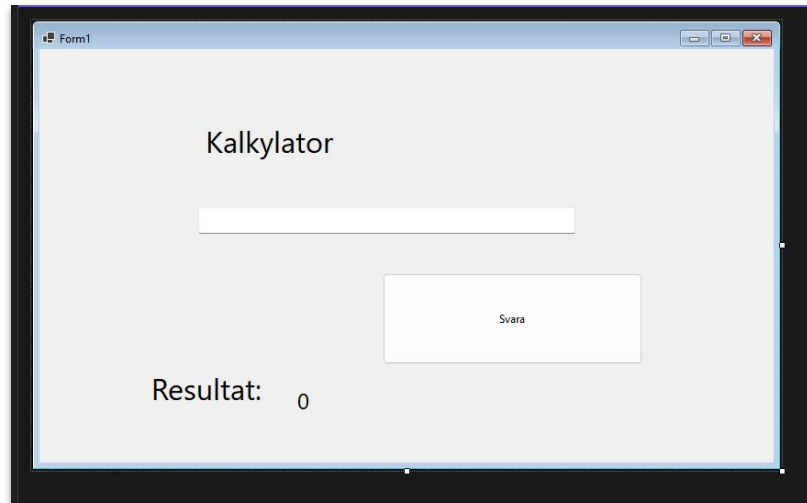
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        while (true)
        {
            Console.WriteLine("Skriv din mattefråga");
            var svar = Console.ReadLine();

            if (svar == null) { continue; }
            if (svar == "exit")
            {
                break;
            }

            float lastNumber = Class1.NewMethod(svar);
            Console.WriteLine(lastNumber.ToString("0.00"));
        }
    }
}
```

Desktop-appar

inte bara text

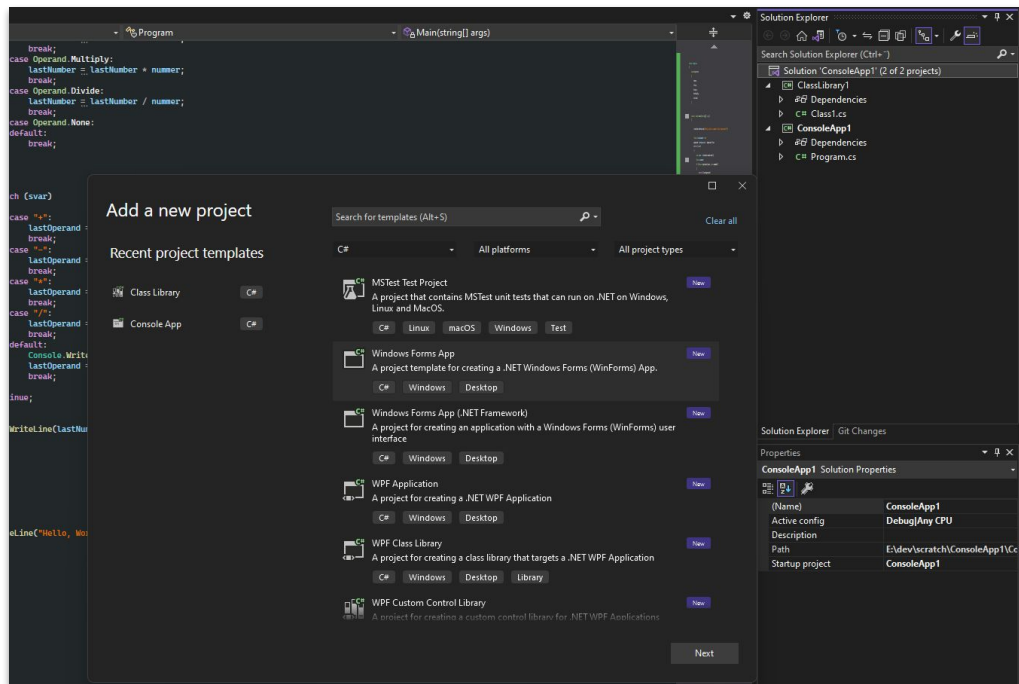


Desktop-appar

— — —
som innan, lägg till ett nytt projekt i er “Solution”

denna gång en
“Windows Forms App”

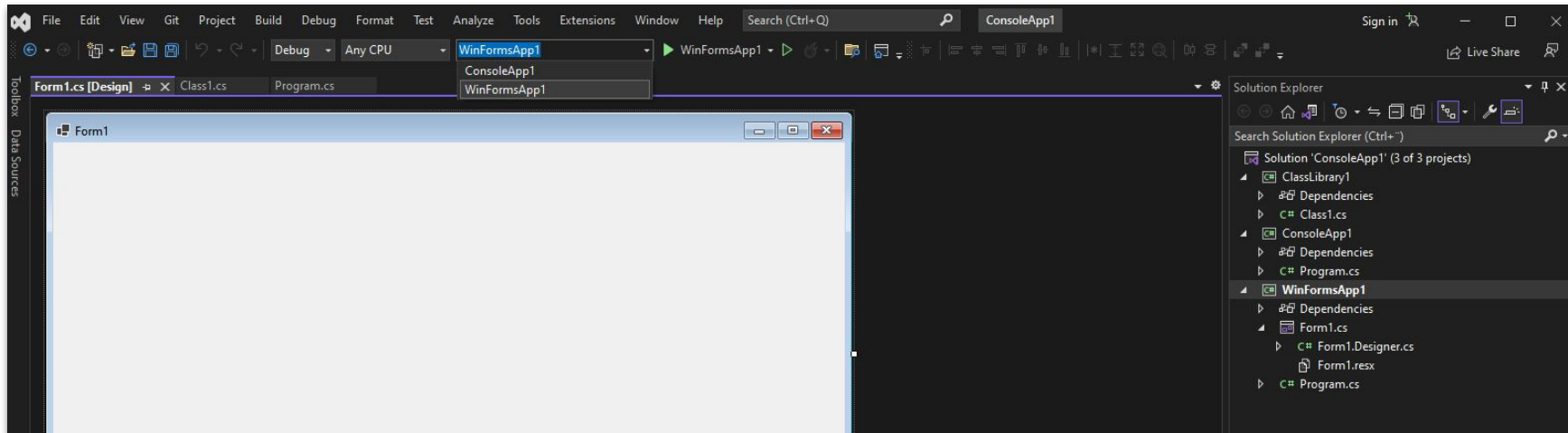
Det finns flera typer av desktop-apps ramverk (som WPF), men “WinForms” är lättanvänt



Desktop-appar

— — —

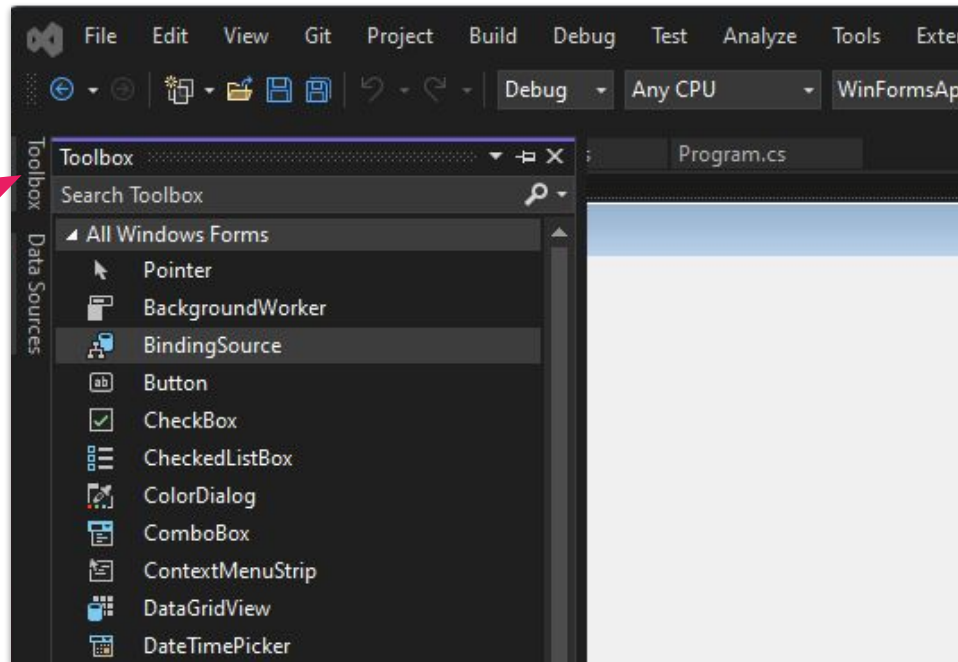
Har ni gjort rätt så borde ni snart se att fönster att arbeta med, när den har laddat färdigt



Desktop-appar

För att lägga till saker i
fönstret, öppna “toolbox”-fliken

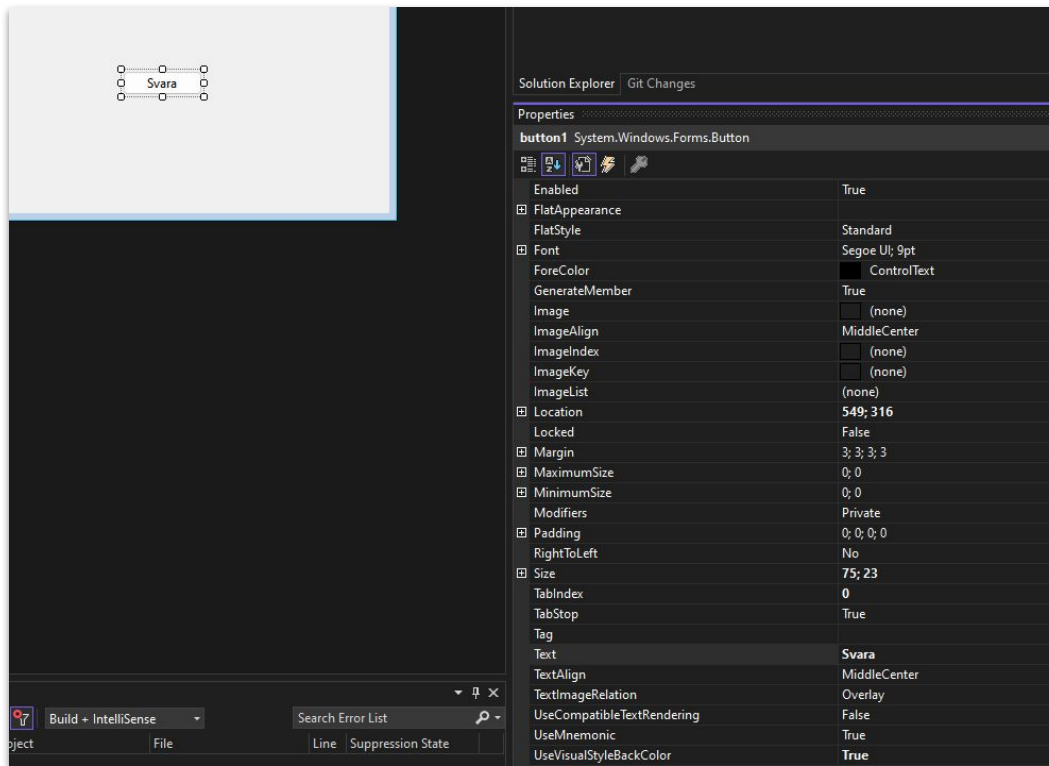
den finns ofta till vänster, annars
under “View”-menyn längst upp



Desktop-appar

— — —

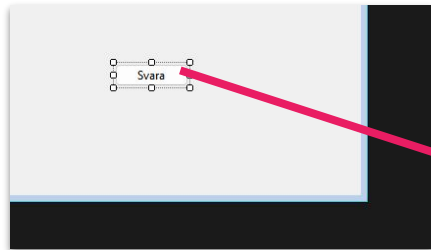
Under “properties” (ofta i hörnet)
finns inställningarna för det som
är markerat



Desktop-appar

— — —
dubbelklicka saker i fönstret
(knappar, textrutor, labels, etc.)
för att skapa funktioner
automatiskt i koden där man kan
lägga till vad som ska hända

gör det för knappen man kommer
ställa frågan med



```
1 namespace WinFormsApp1
2 {
3     3 references
4     public partial class Form1 : Form
5     {
6         1 reference
7         public Form1()
8         {
9             InitializeComponent();
10        }
11
12        1 reference
13        private void Form1_Load(object sender, EventArgs e)
14        {
15        }
16
17        1 reference
18        private void label1_Click(object sender, EventArgs e)
19        {
20        }
21
22        1 reference
23        private void textBox1_TextChanged(object sender, EventArgs e)
24        {
25        }
26
27        1 reference
28        private void button1_Click(object sender, EventArgs e)
29        {
30        }
31    }
32 }
```

Desktop-appar

— — —

Sakerna i fönstret kan nås som gömda variabler med deras namn som syns längst upp i “properties”

Lägg till en referens till klassbiblioteket och få kalkylatorn använda textrutorna när knappen trycks

```
WinFormsApp1
WinFormsApp1.Form1

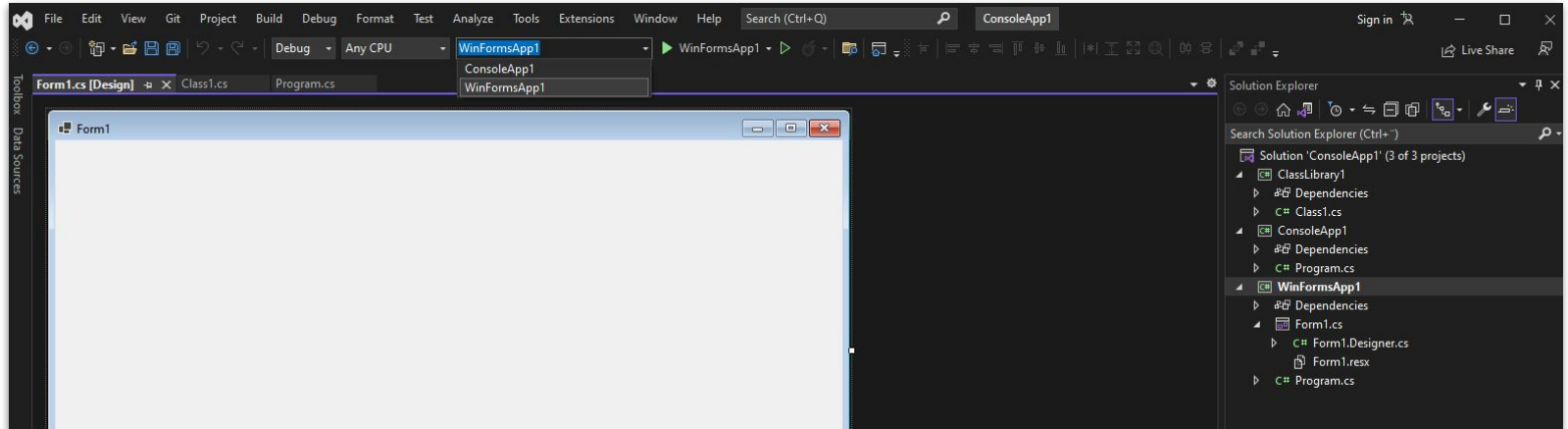
1  using ClassLibrary1;
2
3  namespace WinFormsApp1
4  {
5      3 references
6      public partial class Form1 : Form
7      {
8          1 reference
9          public Form1()
10         {
11             InitializeComponent();
12         }
13
14         1 reference
15         private void button1_Click(object sender, EventArgs e)
16         {
17             label2.Text = Class1.NewMethod(textBox1.Text).ToString();
18         }
19     }
20 }
```

Skriv rätt namn här!

Desktop-appar

— — —

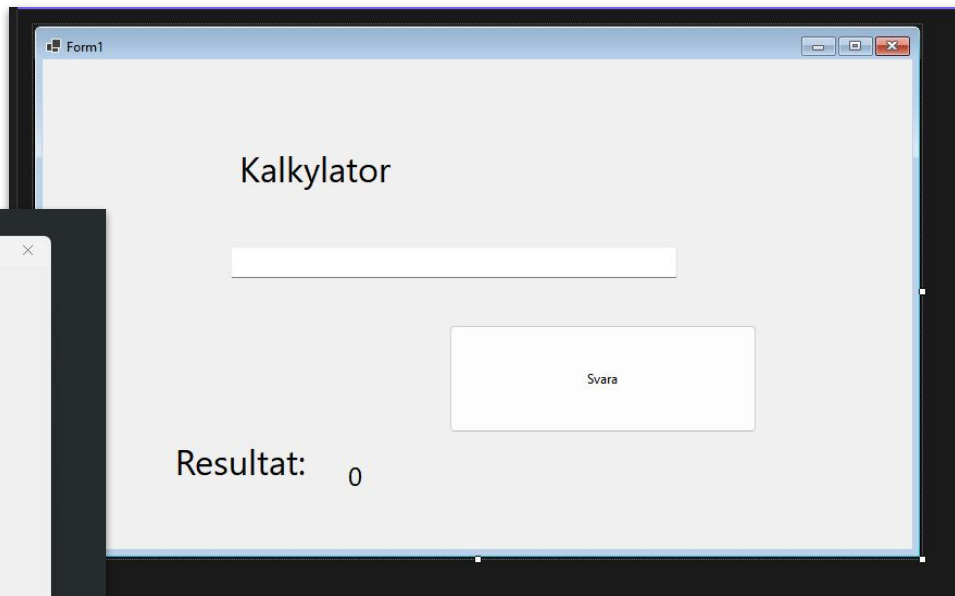
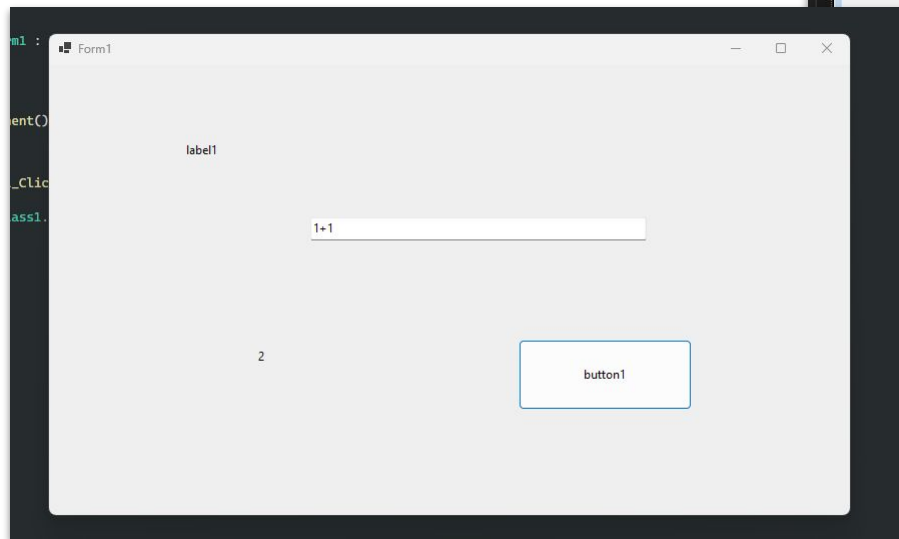
bredvid play-knappen finns en meny
för att välja vilken av projekten
som körs



Desktop-appar

— — —

Färdigt!





The Great Journey



Karlstad Innovation Park
(och Stora Enso)

TGJ

- Spelutvecklings-Community
- Håller evenemang i Kronoparken varje månad
- kalender i vår discord och web

- Spelutvecklare
- Även Folk- och Yrkeshögskola inom spel
- TheGreatJourney.se