

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)

mappen Kalkylator har
koden från förra gången

— — —

Några bra länkar:

samma som förut

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

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

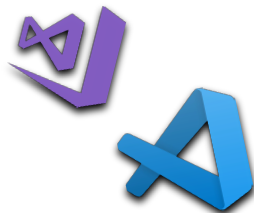
<https://dotnet.microsoft.com/en-us/platform/try-dotnet>

<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

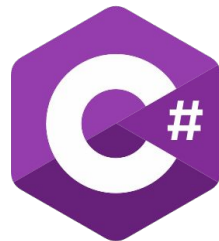
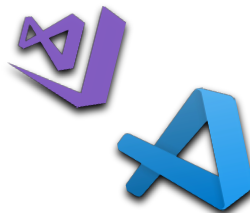


.NET

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
5. Klasser (på riktigt den här gången)
 - a. instansiering genom text
6. dela kod med Unity



.NET

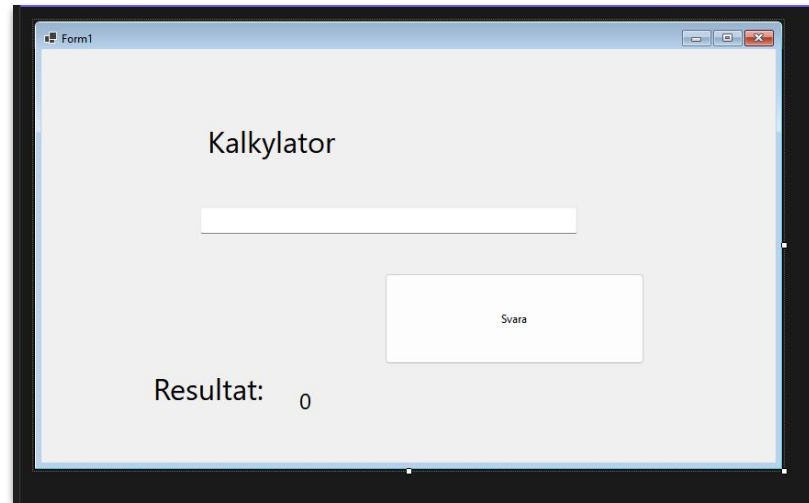
Dags att börja koda

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

Från och med nu krävs **Visual Studio**

Desktop-appar

inte bara text

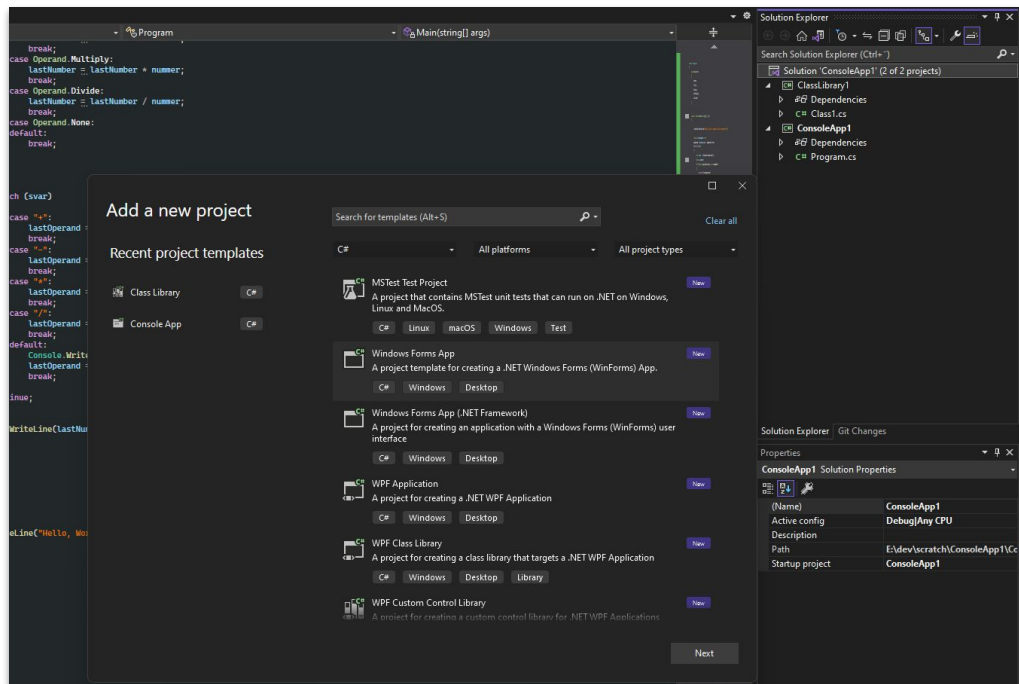


Desktop-appar

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

denna gång en
“Windows Forms App”

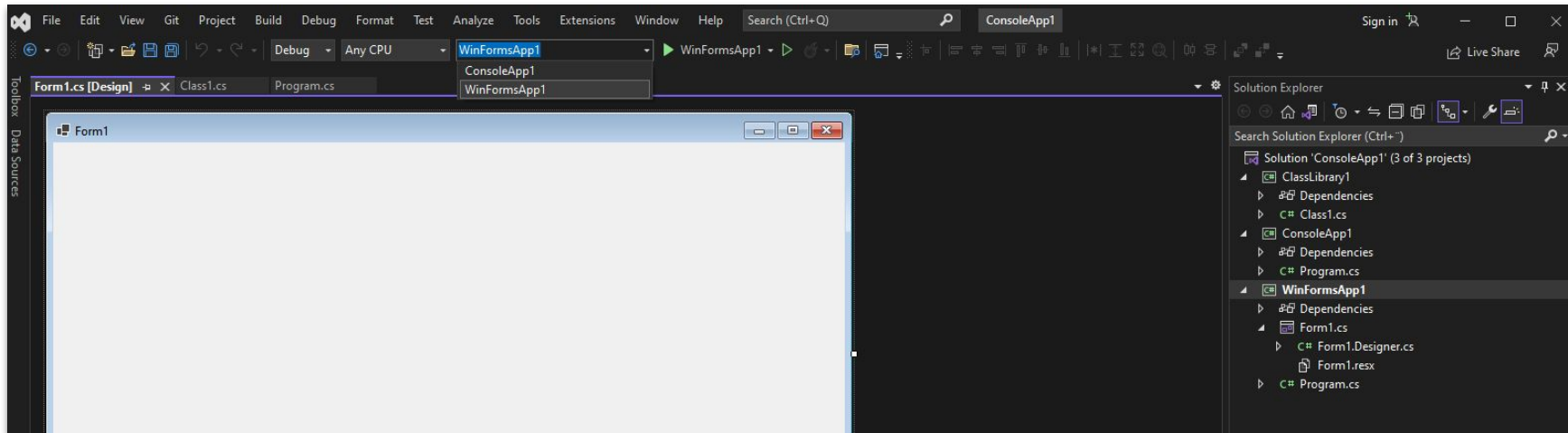
Det finns flera typer av desktop-apps ramverk (som WPF), men “WinForms” är lättanvänt



Desktop-appar

— — —

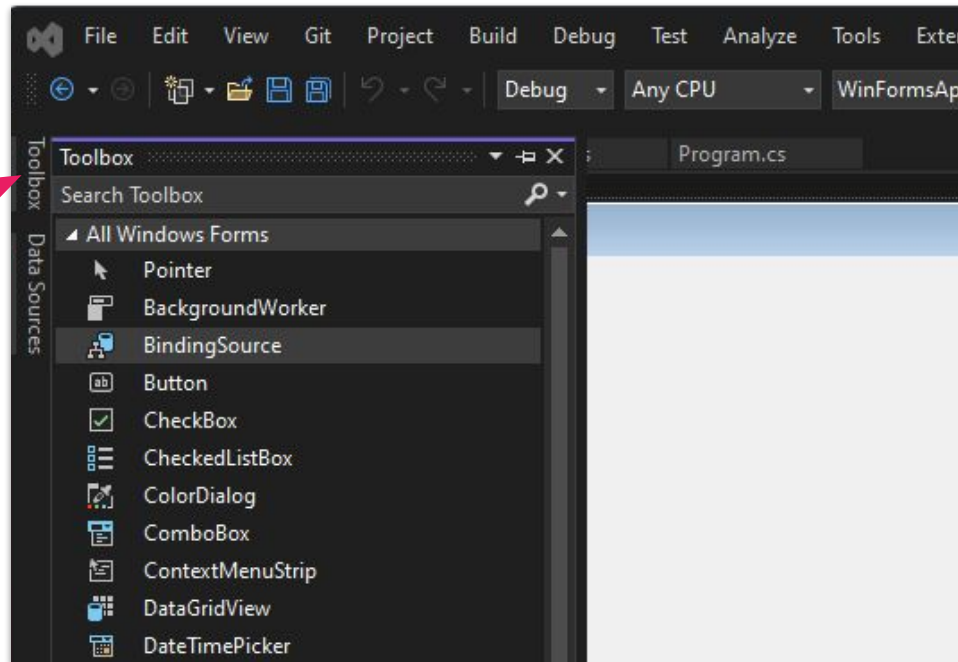
Har ni gjort rätt så borde ni snart se att fönster att arbeta med, när den har laddat färdigt



Desktop-appar

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

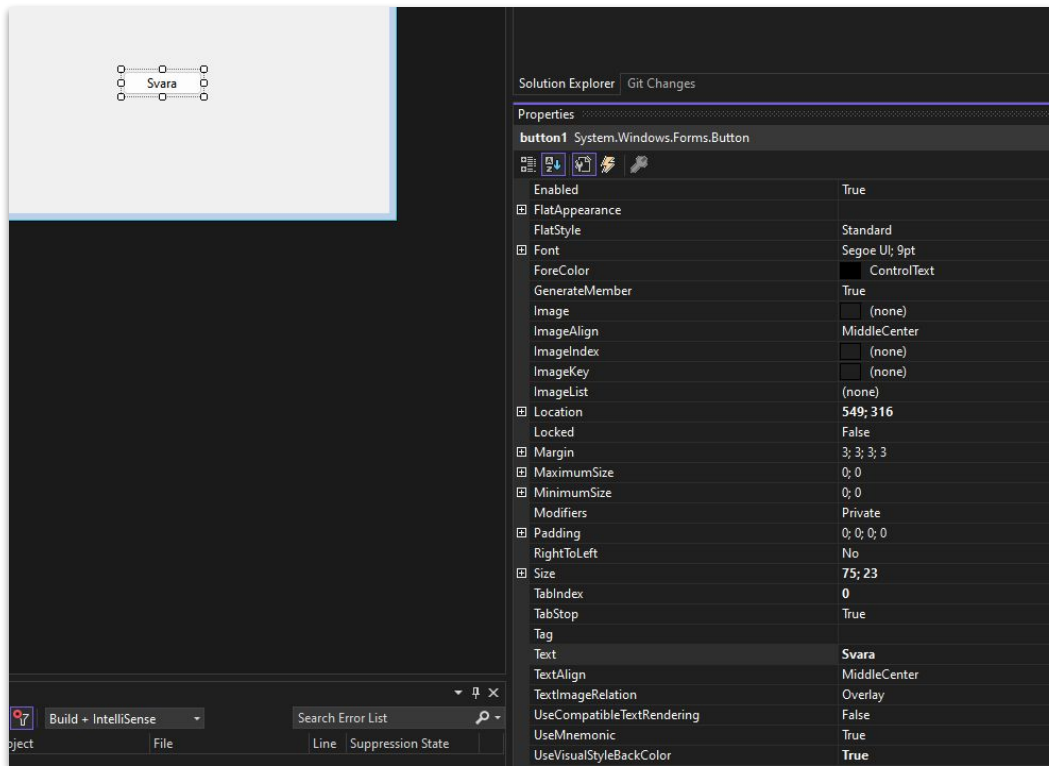
den finns ofta till vänster, annars
under “View”-menyn längst upp



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

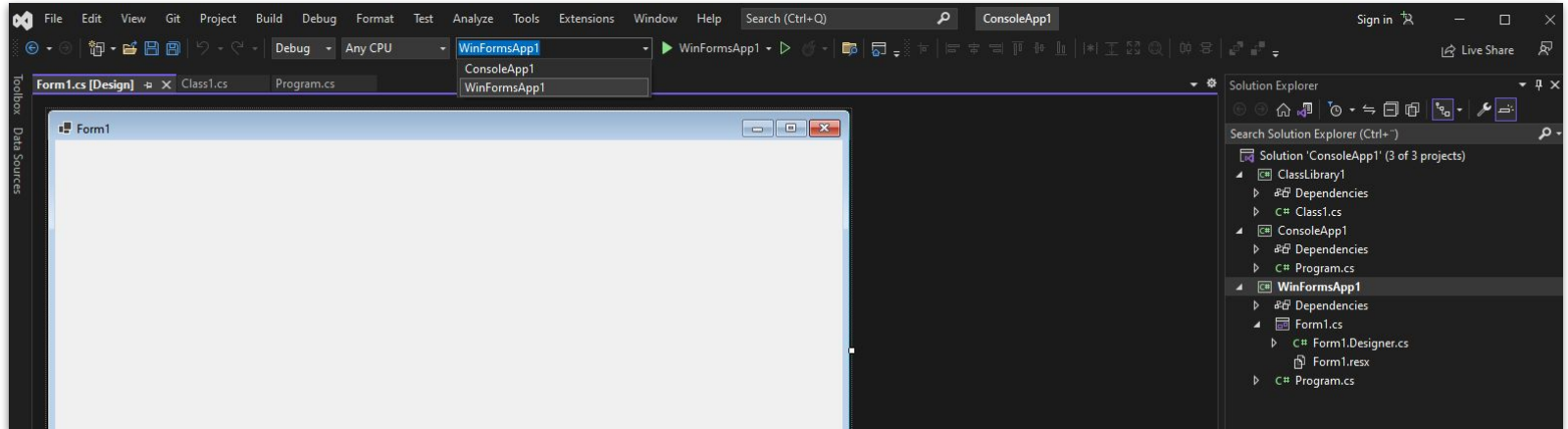
1 using ClassLibrary1;
2
3 namespace WinFormsApp1
4 {
5     3 references
6     public partial class Form1 : Form
7     {
8         1 reference
9         public Form1()
10         {
11             InitializeComponent();
12         }
13
14         1 reference
15         private void button1_Click(object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

Skriv rätt namn här!

Desktop-appar

— — —

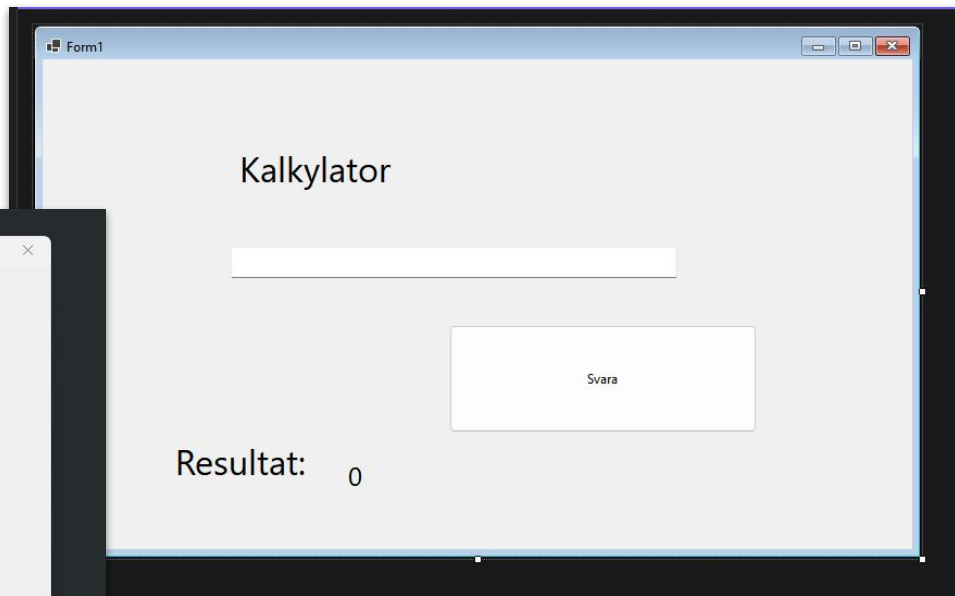
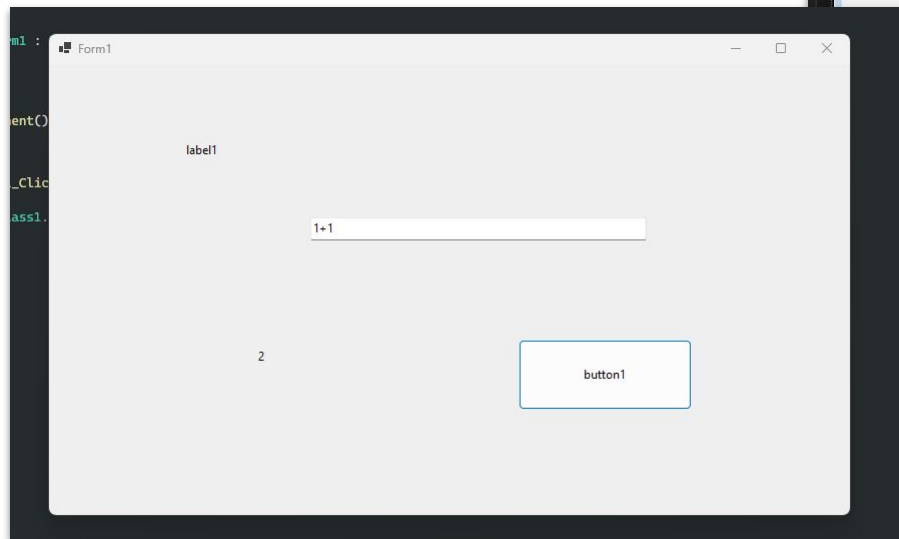
bredvid play-knappen finns en meny
för att välja vilken av projekten
som körs



Desktop-appar

— — —

Färdigt!



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;
        }
    }
}
```

Instansiering

skapa
“instanser”
av klasser

3 references

```
class Position
```

```
{
```

```
    float x;
```

```
    float y;
```

```
1 reference
```

```
public Position(float x, float y)
```

```
{
```

```
    this.x = x;
```

```
    this.y = y;
```

```
}
```

```
}
```

0 references

```
static void Main(string[] args)
```

```
{
```

```
    Position asdf = new Position(0.1f, 2.0f);
```

Instansiering

— — —

“vanliga” typer som text och nummer kallas för “built-in types”, eller ibland “primitiva typer”

de skapas genom att helt enkelt skriva text eller nummer efter “=”

Klasser däremot använder nyckelordet “new” följt av en speciell metod/funktion som kallas för en “konstruktor”

```
3 references
class Position
{
    float x;
    float y;

    1 reference
    public Position(float x, float y)
    {
        this.x = x;
        this.y = y;
    }
}

0 references
static void Main(string[] args)
{
    Position asdf = new Position(0.1f, 2.0f);
}
```

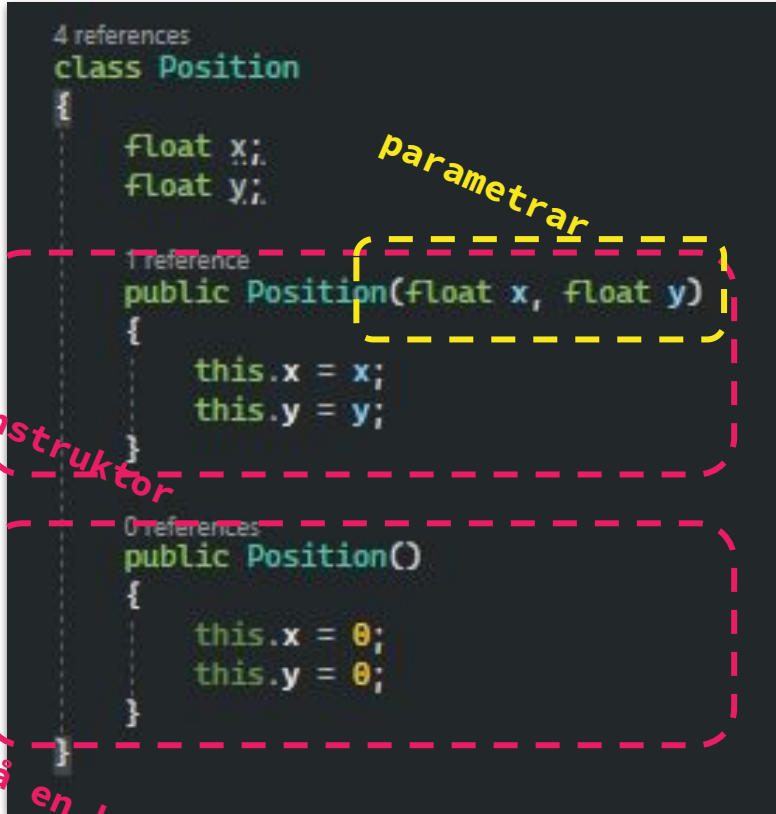
konstruktor

instansiering

Instansiering

Konstruktorn kan vi skriva själva i klassen, som en metod utan typ före och med samma namn som klassen

man kan skapa hur många konstruktorer man vill, så länge de inte har samma "parametrar"



```
4 references
class Position
{
    float x;
    float y;

    1 reference
    public Position(float x, float y)
    {
        this.x = x;
        this.y = y;
    }

    0 references
    public Position()
    {
        this.x = 0;
        this.y = 0;
    }
}
```

parametrar

konstruktör

också en konstruktör

Instansiering

— — —

Om ingen konstruktor skrivits så skapar C# en tom åt oss som är gömd

om variablerna i klassen är “public” så kan man se och ändra värdena utanför konstruktorn

```
2 references
class Position
{
    public float x;
    public float y;
}

0 references
static void Main(string[] args)
{
    Position asdf = new Position();
}
```

```
2 references
class Position
{
    public float x;
    public float y;
}

0 references
static void Main(string[] args)
{
    Position asdf = new Position() { x = 0.1f, y = 2.0f};
    asdf.x = 3.0f;
    asdf.y = asdf.x;
}
```


Instansiering

— — —

man kan sedan lägga till metoder i klassen som kan använda variablerna i klassen (eller metodens parametrar), även om variablerna inte är “public”

metoder används ofta för att ändra gömda värden i klassen, eller att ge tillbaka ett svar baserat på variablerna i klassen (använd då “return” för att leverera svaret)

```
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;
        }
    }
}
```

Interfaces och Inheritance

hur man delar kod eller
lovar att kod finns i
flera klasser

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

Inheritance

— — —

inheritance eller “**arv**” låter klasser “**ärva**” innehållet av en annan klass, så att man kan **återanvända** kod och även använda den nya klassen i platser där klassen den ärver ifrån kan användas

det räcker att skriva “**:**” efter **klassens namn**, och sedan namnet på klassen att ärva från

ibland behöver man justera konstruktor-metoderna också, men bara ibland

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

Klassen man ärver ifrån kallas
“basklass”/“base class”

Interface

ett interface kan användas istället för basklass, men de kan inte innehålla kod, bara mallar/deklarationer för kod som måste finnas i klasserna som ärver den

de är bara till för att kunna använda flera klasser till samma sak

det positiva med dom är att man kan ärva från hur många interfaces man vill samtidigt, medans man kan bara ärva från en basklass

```
interface ISomething
{
    1 reference
    void Print();
    1 reference
    float Magnitude();
}

1 reference
interface ISomethingElse
{
    1 reference
    void Asdf();
}

2 references
class Position : ISomething , ISomethingElse
{
    public float x;
    public float y;

    1 reference
    public float Magnitude()
    {
        return MathF.Sqrt(x + y);
    }

    1 reference
    public void Print()
    {
        Console.WriteLine(x + " " + y);
    }

    1 reference
    public void Asdf()
    {
    }
}
```

Abstrakta klasser

— — —

klasser som är markerade “abstract” blir något mellan en klass och ett interface

abstrakta klasser kan inte användas/instansieras, men de kan ärvas av klasser som kan instansieras

detta tillåter dom att innehålla mallar/deklarationer för kod, precis som interfaces

```
1 reference
abstract class FakePosition : ISomethingElse
{
    1 reference
    public void Asdf()
    {
        Console.WriteLine("asdf 1");
    }

    1 reference
    public abstract void Asdf2();
}

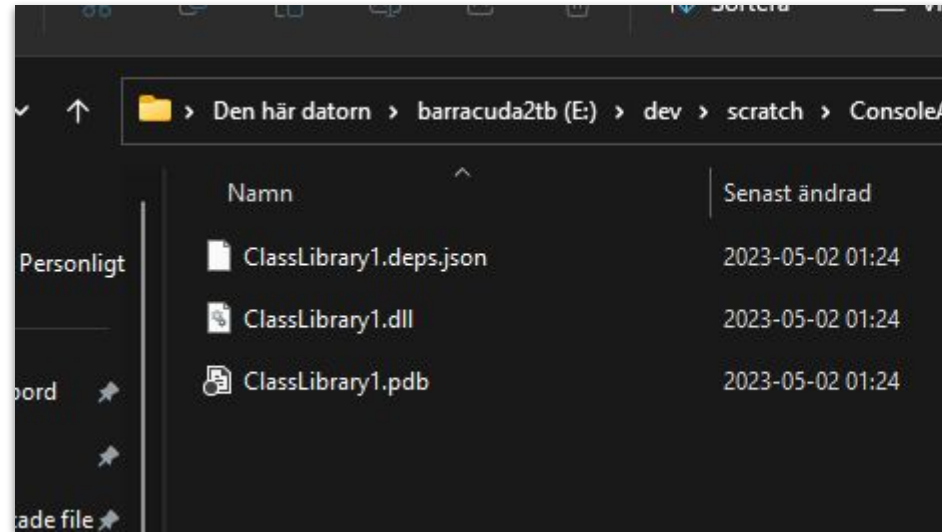
2 references
class Position : FakePosition, ISomething
{
    public float x;
    public float y;

    1 reference
    public override void Asdf2()
    {
        Console.WriteLine("asdf 2");
    }

    1 reference
    public float Magnitude()
    {
        return MathF.Sqrt(x + y);
    }
}
```

Exportera DLL:er

.dll filer som Unity och andra program kan använda





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