

# C# Programming

med folk från TGJ



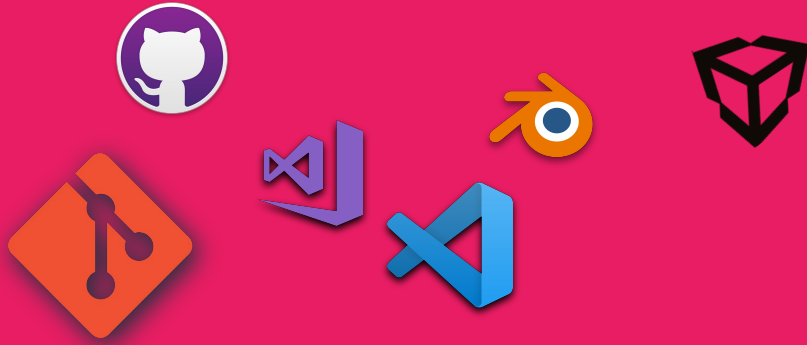


Karlstad Innovation Park  
(och Stora Enso)

# The Great Journey

- Spelutvecklings-Community
- Håller evenemang i Kronoparken varje månad

- Spelutvecklare sitter där
- Folk- och Yrkehögskola inom spel där också
- [TheGreatJourney.se](http://TheGreatJourney.se)



# Verktyg

- **Visual Studio**
  - eller **Visual Studio Code**, **Rider**, **MonoDevelop**, etc.
- **Github Desktop**
  - eller **Fork**, **SourceTree**, **Git** via **MinGW**, etc.
  - även Visual Studio har en enkel Git inbyggd
  - att hämta filer från **webbläsaren** fungerar också

# Filer:

för slides och kod

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

Kalkylator.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/>

— — —

# Dags att börja koda

— — —

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

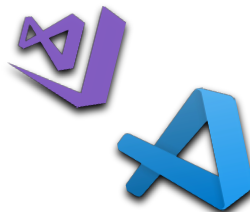
<https://dotnetfiddle.net/>

eller i Visual Studio,  
om ni vill

# 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



.NET

# Typer

hur vet vi vad vi sparar?

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

— — —



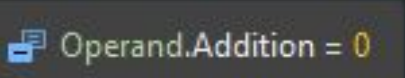
# Enums

— — —

Enums, enumerations, eller “uppräknningar” är ett sätt att namnge samlingar av nummer

bakom kulisserna är de bara nummer egentligen, och kan förvandlas till nummer om man vill

```
enum Operand
{
    Addition,
    Sul
    Mul
```



Operand.Addition = 0

```
enum Operand
{
    Addition,
    Subtaktion,
    Multiplikation,
    Division,
}
```

# Enums

— — —

Enums skapas utanför funktioner, men helst fortfarande inuti “klassen”

```
using System;

0 references
class Program
{
    0 references
    enum Operand
    {
        Addition,
        Subtaktion,
        Multiplikation,
        Division,
    }

    0 references
    static void Main(string[] args)
    {
        bool askForNumber = true;
        float lastNumber = 0;
    }
}
```

# Enums

— — —

istället för att spara och jämföra text, så kan vi istället använda enums

```
bool askForNumber = true;
float lastNumber = 0;
string lastOperand = "+";
while (true)
{
    if (askForNumber)
    {
        Console.WriteLine("Frågar efter nummer");
        var svar = Console.ReadLine();
        if (lastOperand == "+")
        {
            //lastNumber = lastNumber + float.Parse(svar);
            if (float.TryParse(svar, out float number))
            {
                lastNumber = lastNumber + number;
            }
        }
    }
}
```

```
bool askForNumber = true;
float lastNumber = 0;
Operand lastOperand = Operand.Addition;
while (true)
{
    if (askForNumber)
    {
        Console.WriteLine("Frågar efter nummer");
        var svar = Console.ReadLine();
        if (lastOperand == Operand.Addition)
        {
            //lastNumber = lastNumber + float.Parse(svar);
            if (float.TryParse(svar, out float number))
            {
                lastNumber = lastNumber + number;
            }
        }
    }
}
```

# Branching

få saker att hända,  
eller inte hända,  
flera gånger om

- `if`
  - `else`
  - `else if`
- `while`
  - `break`
- `for`
  - `foreach`
- `switch`
  - `case`
  - `default`

# If, else

— — —

“if” är ett speciellt ord i C#, som börjar en kontroll i en parentes

om kontrollen går bra så körs koden i “Scope”:et under den  
annars körs scopet under “else”, om det finns någon else skriven

Är det här sant?

Då körs det här

Annars körs det här

```
if (svar == "+" || svar == "-" || svar == "*" || svar == "/")  
{  
    Console.WriteLine("Det är en operand");  
}  
else  
{  
    Console.WriteLine("Inte en operand");  
}
```

# while

— — —

allt i scopet under en “while”-loop upprepas till den antingen bryts med en “break”, eller att värdet på while-raden blir falskt

det är som en “if” som körs många gånger

```
bool loopa = true;
while (true)
{
    if (!loopa)
    {
        break;
    }
}
```

Samma sak,  
2 sätt



```
bool loopa = true;
while (loopa)
{
    ...
}
```

# Switch

— — —

“Switch” är lite annorlunda

istället för att göra massa if/else för olika alternativ på samma variabel, kan man använda en switch för att välja alternativ

varje alternativ börjar med en **“case”**, och slutar med en **“break”** (eller ibland en **“return”**)

```
Operand lastOperand = Operand.Addition;  
  
switch (lastOperand)  
{  
    case Operand.Addition:  
        break;  
    case Operand.Subtaktion:  
        break;  
    case Operand.Multiplikation:  
        break;  
    case Operand.Division:  
        break;  
}
```

# Switch

— — —

vi kan skriva miniräknarens operand-kontroll med en switch istället för att minska upprepning, och göra den kortare

```
if (lastOperand == Operand.Subtaktion)
{
    lastNumber = lastNumber - float.Parse(svar);
}
if (lastOperand == Operand.Multiplikation)
{
    lastNumber = lastNumber * float.Parse(svar);
}
if (lastOperand == Operand.Division)
{
    lastNumber = lastNumber / float.Parse(svar);
}
```



```
switch (lastOperand)
{
    case Operand.Addition:
        lastNumber = lastNumber + float.Parse(svar);
        break;
    case Operand.Subtaktion:
        lastNumber = lastNumber - float.Parse(svar);
        break;
    case Operand.Multiplikation:
        lastNumber = lastNumber * float.Parse(svar);
        break;
    case Operand.Division:
        lastNumber = lastNumber / float.Parse(svar);
        break;
}
```



# Designmönster

hur programmerare delar  
kunskap mellan språk

[https://en.wikipedia.org/  
wiki/Software design patt  
ern](https://en.wikipedia.org/wiki/Software_design_pattern)

---

# “Try”-metoder


— — —

det finns ofta versioner av metoder med “try” i namnet

det antyder att metoden går att “fråga snällt” efter svar istället för att krascha programmet när något är fel

```
if (float.TryParse(svar, out float nummer))  
{  
    lastNumber = lastNumber + nummer;  
}
```

```
// lastNumber = lastNumber + float.Parse(svar);  
if (float.TryParse(svar, out float nummer))  
{  
    lastNumber =  
}
```

 **bool float.TryParse(string? s, out float result) (+ 3 overloads)**  
Converts the string representation of a number to its single-precision value. The return value indicates whether the conversion succeeded or failed.

# “Try”-metoder

— — —

Det går få mer än ett svar från en metod genom “out”-parametrar

**Try**-metoder svarar ofta både med ett sant/falskt svar för om allt gick bra, och även vad svaret är genom en **out**-parameter (**om det finns ett svar**)

Allt som svarar med sant/falskt kan läggas i en “if”-sats

```
if (float.TryParse(svar, out float nummer))  
{  
    ...  
    lastNumber = lastNumber + nummer;  
}
```

# “Try”-metoder

— — —

```
if (float.TryParse(svar, out float nummer))  
{  
    ...  
    lastNumber = lastNumber + nummer;  
}
```

Denna rad händer  
bara om det  
finns ett svar

Allt som svarar med sant/falskt kan läggas i en “if”-sats

# 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)  
}
```

# 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;
        }
    }
}
```





The Great Journey



Karlstad Innovation Park  
(och Stora Enso)

# TGT

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

- Spelutvecklare
- Även Folk- och Yrkeshögskola inom spel
- [TheGreatJourney.se](https://thegreatjourney.se)