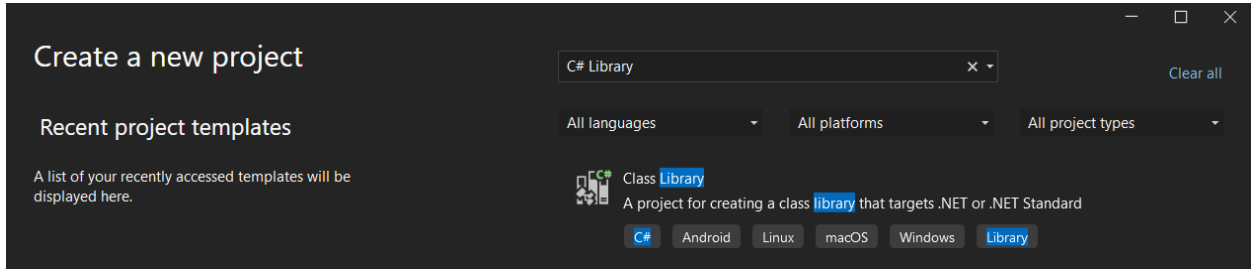
OOP

Class library projekt vs. Konsol app projekt

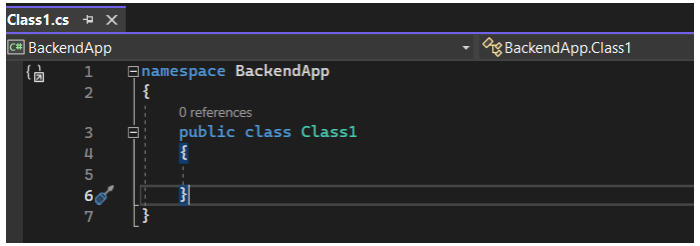
Genbrug din ‘types’(klasser)

Opgave 1, Opret C# class library projekt

- Opret en ”**C# library project**” (Dette er en konsol app uden konsol!), og give det et navn f.eks. ”**BackendApp**”.



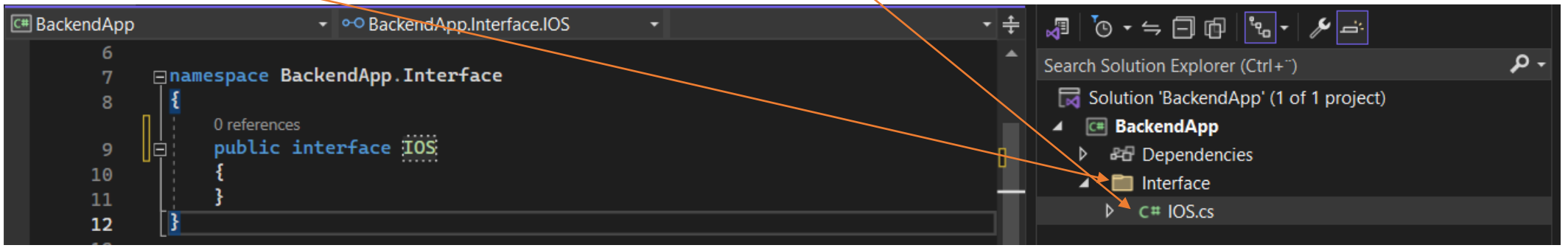
- Bemærk når projekt er oprettet, er det er fuldstændigt som en konsol app som i indtil nu har arbejdet med. MEN UDEN PROGRAM.CS FIL !!!!!
  - Som tidligere nævnt, alle .NET projekter oprettet med Visual Studio kommer med indbygget eksempel. I jeres konsol app, er der indbygget en konsol eksempel: Consol.WriteLine(”Hello World!”);. I denne class library app, er Class1 også bare en indbygget eksempel for at hentyde, at i her skal kun anvende klasser:



  - Lige som da vi sletter eksempel Consol.WriteLine(”Hello World!”); fra vores konsol app, så slet også klasse Class1 fordi det også kun er en eksempel!

Opgave 2, Opret interface

- Opret folder ”Interface”, og i folderen opret følgende interface type(klasse) i folderen:



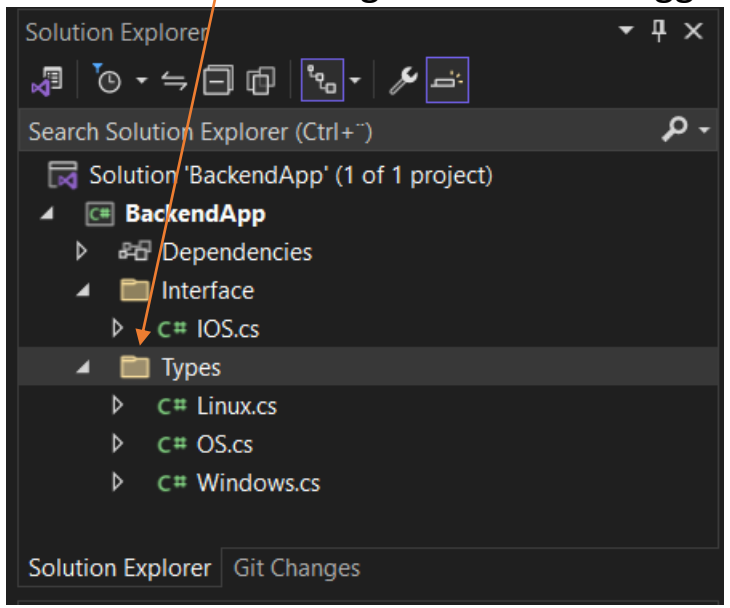
- Interface klassen **skal have en public modifyer** og skal indeholde følgende members : property UserFullName og abstrakt metode ShowOSType()

```
namespace BackendApp.Interface
{
    0 references
    public interface IOS
    {
        0 references
        public string UserFullName { get; set; }
        0 references
        public string ShowOSType();
    }
}
```

- Hvis alt er gjort korrekt, skal i kunne lave en **build** af projektet nu, uden at den fejler.

Opgave 3, Opret OOP arv

- Opret folder ”Types”, og i folderen opret følgende types(klasser) i folderen:
  - Base klasse: OS.cs som **skal have public modifyer.**
  - Klasse Windows.cs og Linux.cs skal begge **have public modifyer** og skal derive fra OS.cs




```
public class Linux : OS
{
}
```

```
public class Windows : OS
{
}
```

- Hvis alt er gjort korrekt, skal i kunne lave en **build** af projektet nu, uden at den fejler.

Opret OOP arv, forsæt fra forrige slide…

- Base klasse: OS.cs **skal have public modifyer** og skal indeholde members som alle dens derived klasser skal bruge (i dette tilfælde, kun en property: **UserFullName**):

```
public class OS
{
    1 reference
    public string UserFullName { get; set; }

    0 references
    public OS(string userFullName)
    {
        UserFullName = userFullName;
    }
}
```

- Opdaterer Windows.cs og Linux.cs types(klasserne), så de videregive det modtaget userFullName parameter vider til sin base klasse når/hvis den bliver istantieret:

```
public class Windows : OS
{
    0 references
    public Windows(string userFullName) : base(userFullName)
    {
    }
}
```

```
public class Linux : OS
{
    0 references
    public Linux(string userFullName) : base(userFullName)
    {
    }
}
```

- Hvis alt er gjort korrekt, skal i kunne lave en **build** af projektet nu, uden at den fejler.

Opgave 4, Implementer det oprettet interface

- Nu skal derived klasserne Windows.cs og Linux.cs implementerre interface IOS:

```
public class Windows : OS, IOS
{
    0 references
    public Windows(string userFullName) : base(userFullName)
    {
    }
}
```

```
public class Linux : OS, IOS
{
    0 references
    public Linux(string userFullName) : base(userFullName)
    {
    }
}
```

- Interface indikerer fejl, fordi der mangler at implementerer dens abstrakt metode `public string ShowOSType();`. implementerer metoden:

```
public class Windows : OS, IOS
{
    0 references
    public Windows(string userFullName) : base(userFullName)
    {
    }

    1 reference
    public string ShowOSType()
    {
        return $"Welcome {UserFullName} to Windows, running from backend!";
    }
}
```

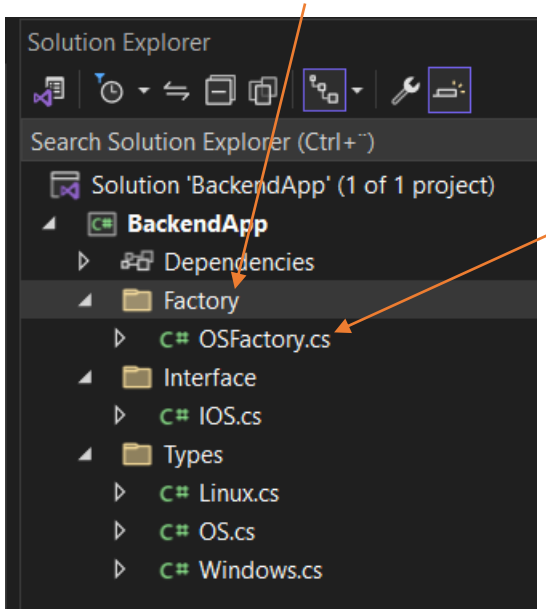
```
public class Linux : OS, IOS
{
    0 references
    public Linux(string userFullName) : base(userFullName)
    {
    }

    1 reference
    public string ShowOSType()
    {
        return $"Welcome {UserFullName} to Linux, running from backend!";
    }
}
```

- Hvis alt er gjort korrekt, skal i kunne lave en **build** af projektet nu, uden at den fejler.

Opgave 6, opret interface factory klasse

Opret folder **Factory** og i folderen opret en klasse : OSFactory.cs



OSFactory.cs **skal have public modifyer**. Implementerer herefter følgende metode i OSFactory.cs klasse således, når den kaldes, returner den enten Windows eller Linux objectet:

```
public class OSFactory
{
    0 references
    public IOS Indentify(string minUserFolder)
    {
        if (minUserFolder.Contains("C:"))
        {
            return new Windows("Niels Olesen");
        }
        else if (minUserFolder.Contains("/"))
        {
            return new Linux("Niels Olesen");
        }
        else
            return null;
    }
}
```

Hvis alt er gjort korrekt, skal i kunne lave en **build** af projektet nu, uden at den fejler.

Opgave 7

Opret en app som skal køre det implementeret class library projekt

Bemærk, der er ingen Console.WriteLine(), og der er ingen Program.cs i jeres class library projekt. Applikation er nemlig nu generisk og cross-device, og det er op til det applikation den skal køre i, at implementerer hvordan den vil vise/udskrive det tekst som skal udskrives. Vi skal nu prøve at køre den i en .NET web applikation:

- Source cotrol jeres class library projekt på GitHub.
- Læreren opretter nu en .NET Web applikation, som den applikation jeres class library skal køre i.
- For cross-platform test, skal i kommer op til læreren som henter jeres projekt fra jeres GitHub repo. Læreren vil så set en 'reference' til jeres class library projekt i lærerens .NET Webapplikation og kører Web applikationen. Applikation skal så udskrive følgende baseret på jeres class library projekt:
  - `Welcome Niels Olesen to Windows, running from backend!` Når læreren køre jeres projekt mod Windows test miljø.
  - `Welcome Niels Olesen to Linux, running from backend!` Når læreren køre jeres projekt mod Linux test miljø.