Abstrakt / Interface
Overskriv / Overload
2.HF

It & Data, Odense

SYDDANSK ERHVERVSSKOLE

Abstrakte klasser

- Kan udelukkende bruges til arv
- For at anvende klassen, oprettes først en egen klasser, der arver fra den abstrakte klasse.
- Bruges til helt generelle klasser (skabeloner).

```
2 references
abstract class Shape
{
    4 references
    public abstract int GetArea();
}

3 references
class Square : Shape
{
    int side;

    1 reference
    public Square(int n) => side = n;

    // GetArea method is required to avoid a compile-time error.
    4 references
    public override int GetArea() => side * side;
}

3 references
class Circle : Shape
{
    int radius;

    1 reference
    public Circle(int r) => radius = r;

    4 references
    public override int GetArea()
    {
        int area = (int)(Math.Pow(radius, 2) * Math.PI);
        return area;
    }
}
```

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

        Square sq = new Square(12);
        Console.WriteLine($"Area of the square = {sq.GetArea()}");

        Circle cl = new Circle(12);
        Console.WriteLine($"Area of the circle = {cl.GetArea()}");
```

Abstrakte klasser

Links:

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/abstract

https://csharp.net-tutorials.com/classes/abstract-classes/

https://syntaxdb.com/ref/csharp/abstract

Interface (grænseflade)

Et interface opremser, hvilke egenskaber, metoder og hændelser den klasse, der *implementere* interfacet skal bestå af.

En form for en kontrakt / grænseflade mellem de to klasser.

- En anden form for arv –> fælles grænseflade
- Syntaks:

```
[virkefelt] interface [navn på interface]
{
    [definition af medlemmer]
}
```

- Ved navngivning af interfaces er det kutyme at starte navnet med I (stort i).
- Eksempelvis **IComparable**, der sikre, at alle klasser har en metode **CompareTo**

Eksempler på eksisterende Interfaces

Disse eksempler på interfaces kan med fordel anvendes i jeres egen kode. Se eksemplet fra Zip-filen ”ConsoleAppPart”

De to første metoder har til formål at sammenlige et objekt med et andet og kan dermed bruges til at sortere objekter.

Den sidste metode undersøger, om objektet (fra en collection) er magen til et andet.

```
public interface IComparable
{
        int CompareTo(object o);
}

public interface IComparable<T>
{
        int CompareTo(T o);
}

public interface IEquatable<T>
{
        int Equals(T o);
}
```

Brug af interface

- Eksempel på sammenligning.

```
public class Dog : IComparable<Dog>
{
    2 references
    public string Name { get; private set; }
    6 references
    public int Weight { get; private set; }

    0 references
    public Dog(string name, int weight)
    {
        Name = name;
        Weight = weight;
    }

    0 references
    public override string ToString()
    {
        return "Jeg hedder " + Name + " og vejer " + Weight + " kg.";
    }

    0 references
    public int CompareTo(Dog obj)
    {
        int Result = 0;
        if (this.Weight < obj.Weight)
            Result = -1;
        else if(this.Weight > obj.Weight)
            Result = 1;

        return Result;
    }
```

```
static void Main(string[] args)
{
    // Interface demo
    List<Dog> hunde = new List<Dog>();
    hunde.Add(new Dog("Fido", 20));
    hunde.Add(new Dog("Sultan", 14));
    hunde.Add(new Dog("Bulder", 25));
    hunde.Add(new Dog("Bjæf", 19));

    foreach (Dog hund in hunde)
        Console.WriteLine(hund.ToString());

    hunde.Sort();
    Console.WriteLine();

    foreach (Dog hund in hunde)
        Console.WriteLine(hund.ToString());
    Console.WriteLine();
```

```
Jeg hedder Fido og vejer 20 kg.
Jeg hedder Sultan og vejer 14 kg.
Jeg hedder Bulder og vejer 25 kg.
Jeg hedder Bjæf og vejer 19 kg.

Jeg hedder Sultan og vejer 14 kg.
Jeg hedder Bjæf og vejer 19 kg.
Jeg hedder Fido og vejer 20 kg.
Jeg hedder Bulder og vejer 25 kg.
```

Forklaring til interface eksempel

Først oprettes og gemmes fire hunde i en collection, der udskrives.

Derefter kaldes **Sort**-metoden, der er en del af **List<T>** klassen. Listen udskrives igen og er nu sorteret korrekt.

Det kan kun lade sig gøre, fordi klassen **Dog** implementerer **Icomparable** – der sikre, at metoden **CompareTo** findes.

Hvordan den (CompareTo) er implementeret, ved **Sort** ikke noget om – den sortere blot på resultatet af metoden.

Brug af eget interface

- Opret dit eget interface som f.eks.

```
2 references
public interface IId
{
    2 references
    string Id();
}
```

- Derefter oprette en eller flere klasser, der anvender dit nye interface

```
public class Maskine : IId
{
    2 references
    public string MaskinId { get; set; }
    2 references
    public string Type { get; set; }

    0 references
    public Maskine(string maskinId, string type)
    {
        MaskinId = maskinId;
        Type = type;
    }

    2 references
    public string Id()
    {
        return Type + "/" + MaskinId;
    }
}
```

```
public class Person : IId
{
    2 references
    public string Cprnummer { get; set; }
    2 references
    public string Navn { get; set; }

    0 references
    public Person(string cpr, string navn)
    {
        Cprnummer = cpr;
        Navn = navn;
    }

    2 references
    public string Id()
    {
        return Navn + "/" + Cprnummer;
    }
}
```

Brug af eget interface

- Klasserne tages i brug i programmet.

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        List<IId> s = new List<IId>();
        s.Add(new Person("111111-1111","Svend"));
        s.Add(new Maskine("XYZ123", "Græsslåmaskine"));
        s.Add(new Person("121212-1212", "Pia"));
        s.Add(new Maskine("XYZ235", "Hækkeklipper"));
        foreach (IId item in s)
            Console.WriteLine(item.Id());
```

- Resultat

```
Svend/111111-1111
Græsslåmaskine/XYZ123
Pia/121212-1212
Hækkeklipper/XYZ235
```

- Konklusion: To vidt forskellige klasser, der begge har metoden Id til fælles (uden traditionel arv).

Interface

Forskellige nyttige link:

Microsoft Interface programmerings guide:

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface

Tutorial med eksempler:

https://www.tutorialspoint.com/csharp/csharp_interfaces.htm

Overskriv (Override)

- Metoder i hver sin klasse men med samme navn
- Metoden i den originale klasse ”overskrives” af metoden i klassen, der arver.
- Normalt kan to metoder ikke kaldes det samme navn.
- Ved brug af virtual / override tillades overskrivning.
- Eksempel

```
public class Person : IId
{
    5 references
    public string Fornavn { get; set; }
    3 references
    public string Efternavn { get; set; }
    3 references
    public string Cprnummer { get; set; }

    1 reference
    public virtual string VisNavn()
    {
        return Fornavn + " " + Efternavn;
    }
```

```
2 references
public class Teacher : Person
{
    1 reference
    public override string VisNavn()
    {
        return Efternavn + ", " + Fornavn;
    }
```

Override

- Et simpelt program, der anvender begge klasser

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Person p = new Person("121156-1357", "Ole", "Olesen");
        Teacher t = new Teacher("110291-8642", "Anne", "Andersen");
        Console.WriteLine();
        Console.Write(" ");
        Console.WriteLine(p.VisNavn());
        Console.Write(" ");
        Console.WriteLine(t.VisNavn());
```

- Resultatet af anstrengelserne.
- Som det ses, byttes der om på fornavn og efternavn selvom metoden, der kaldes, hedder det samme.

```
Ole Olesen
Andersen, Anne
```

Overload af en metode

Forskellige metoder ved samme navn kan ske ved at:

- Ændre af antallet af parametre på de to metoder ved samme navn
- Ændring af data typen på parametrene
- Rækkefølgen af parametrene

Note:

Flere metoder bør kun have samme navn, hvis funktionaliteten er ensartet.

```
public class Tester
{
    int sum;
    // adding two integer values.
    0 references
    public int Add(int a, int b)
    {
        sum = a + b;
        return sum;
    }

    // adding three integer values.
    0 references
    public int Add(int a, int b, int c)
    {
        sum = a + b + c;
        return sum;
    }

    // adding two strings / convert to integer
    0 references
    public int Add(string tal1, string tal2)
    {
        int a, b;
        Int32.TryParse(tal1, out a);
        Int32.TryParse(tal2, out b);
        sum = a + b;
        return sum;
    }
}
```

Overload versus overriding

- Links

https://www.c-sharpcorner.com/blogs/difference-between-method-overloading-and-method-overriding

https://www.c-sharpcorner.com/UploadFile/8a67c0/method-overloading-and-method-overriding-in-C-Sharp/

https://medium.com/@atandaoluchiaminat/overload-vs-override-in-object-oriented-programming-oop-a38ca0ccaf40

Slut på
abstrakt, interface, overskriv og overload