Marker som gennemført

Open / Closed princippet omhandler at både klasser og metoder skal være åbne for udvidelser men lukket for ændringer.

Det betyder i praksis at alle klasser du skriver / designer skal kunne bruges som de er, men at de kan udvides f.eks. via arv eller override, hvis nødvendigt. Man bør aldrig ændre i den eksisterende kode, f.eks. ændre på signaturen for en metode. I stedet overloader man en metode.

Lad os sige at vi har en rectangle klasse med tilhørende properties.

```
public class Rectangle
{
    public double Width { get; set; }
    public double Height { get; set; }
}
```

Nu kommer vores kunde, som stiller et nyt krav til vores applikation. Han ønsker at kunne applikation kan tage en liste (collection) af rektangler og kunne beregne arealet på disse. Det er da heldigvis ikke noget problem, for vi kender jo formlen for en areal beregning højde * bredde.

```
public class AreaCalculator
{
    public double Area(Rectangle[] shapes)
    {
        double area = 0;
        foreach (var shape in shapes)
        {
            area += shape.Width*shape.Height;
        }

        return area;
    }
}
```

Vores kunde bliver meget begejstret "Det var jo lige det jeg gerne ville ha'", udbryder han... "men jeg kunne faktisk godt tænke mig at applikationen også kan udregne arealet på cirkler også".

Det gør jo tingene en lille smule mere besværligt, men vi udvider metoden og kunden er tilfreds.

```
public double Area(object[] shapes)
{
    double area = 0;
    foreach (var shape in shapes)
    {
        if (shape is Rectangle)
        {
            Rectangle rectangle = (Rectangle) shape;
            area += rectangle.Width*rectangle.Height;
        }
        else
        {
            Circle circle = (Circle)shape;
            area += circle.Radius * circle.Radius * Math.PI;
        }
    }

    return area;
}
```

?

Men det er sådan med kunder og kvinder at de aldrig rigt bliver helt tilfredse der er altid noget der skal laves om..... og kunden vil nu også gerne have at applikationen kan udregne arealet for trekanter.....

I dette eksempel bryder vi altså med open / closed princippet fordi vi bliver ved med at ændre i klassen i stedet for at udvide.

Havde vi nu tænkt os godt om inden vi gik i gang med opgave, havde vi arbejdet på et højrere abstraktionsniveau og lavet vores kode rigtigt første gang.

Allerførst ville vi istedet for at oprette en klasse rectangle have lavet en abstract klasse som vi havde kaldt shape.

```
public abstract class Shape
{
    public abstract double Area();
}
```

Efterfølgende ville vi have lave to konkrete klasser en til at repræsentere et rectangle og en til circle.

```
public class Rectangle : Shape
{
    public double Width { get; set; }
    public double Height { get; set; }
    public override double Area()
    {
        return Width*Height;
    }
}

public class Circle : Shape
{
    public double Radius { get; set; }
    public override double Area()
    {
        return Radius*Radius*Math.PI;
    }
}
```

Og til sidst ....

```
public double Area(Shape[] shapes)
{
    double area = 0;
    foreach (var shape in shapes)
    {
        area += shape.Area();
    }

    return area;
}
```

Nu er alle klasser åbne for udvidelser .....

Senest ændret: søndag den 22. april 2018, 06:40

◄ Vejledende løsning - "Hjælp Jakob"

Spring til...

Se videoen : The O In SOLID ►