Marker som gennemført

Et af de mest benyttede design mønstre der findes er en singleton. En singleton går i alt sin enkelthed ud på at vi kun ønsker at instantiere **et** og kun **et objekt** af en klasse.

Men lad os kigge lidt ned i en singleton.....

Lad os forestille os at vi skal simulere et fjernsyn. Det er et meget unikt fjernsyn og der må derfor kun kunne laves een instans af klasse TV. Fjernsynet har 2 egenskaber. Det kan enten tændes/slukkes og så kan der justeres på lyden.

```
public class Tv
{
    int volume = 0;
    bool on = false;

    public int Volume { get => volume; set => volume = value; }
    public bool On { get => on; set => on = value; }

    public Tv()
    {
       //konstruktør
    }
 }
```

Som situationen er lige nu, kan vi bare instantiere en masse fjernsyn ved at bruge new og det kan vi fordi konstruktøren er sat public. Så det første vi skal gøre er at sætte denne private, så er det kun klassen selv der kan instatiere sig selv.

```
private Tv()
{
  //konstruktør
}
```

Det næste der skal gøres er at oprette en statisk instans af Tv og i C# vil man bruge en property til dette, bemærk at det altid er god skik at kalde sin instans "instance", så kan andre programmører se at der her er tale om et singleton pattern.

```
private static Tv instance;
```

I modsætning til almindelige properties implementeres der ikke en set metode, men kun en get metode. I get metoden tjekkes der på om instance er == null, hvis den er det så oprettes der en ny instans.

```
public static Tv Instance
{
   get
   {
      if (instance == null)
      {
         instance = new Tv();
      }
      return instance;
   }
}
```

For at få fat i fjernsynsobjektet skal du altså benytte Instance.

?

```
 //Tænd
 Tv.Instance.On = true;
 //Skru op
 Tv.Instance.Volume = 3;
```

Brug lige 2 minutter til at tænke over hvorfor følgende stump kode ikke giver nogen mening.....men at det virker.

```
Tv television = Tv.Instance;

//Tænd
television.On = true;

//Skru op
television.Volume = 3;
```

Hele koden

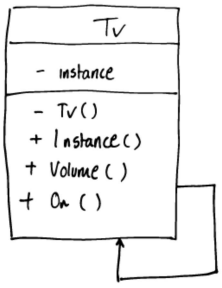
```
public class Tv
{
    int volume = 0;
    private static Tv instance;
    bool on =false;

    public int Volume { get => volume; set => volume = value; }
    public bool On { get => on; set => on = value; }

    public static Tv Instance
    {
        get
        {
            if (instance == null)
            {
              instance = new Tv();
            }
            return instance;
        }
    }

    private Tv()
    {
        //konstruktør
   }
}
```

UML diagrammet for en singleton er lidt sjov, for som du kan se er der en reference til klassen selv.



Nu tænker du måske, hvorfor bruger vi ikke bare en statisk klasse i stedet for..... Det kunne man jo i virkeligheden godt, men en statisk klasse kan f.eks. ikke implementere et interface og der kan ikke arves fra en statisk klasse. Det betyder reelt at vi vil gå på kompromis med OOP principperne, hvis vi benytter en statisk klasse.

Fordele ved singleton

1. En singleton kan implementere interfaces
2. Der kan arves fra en singleton.
3. Man kan lave det der kaldes "lazy load", hvilket betyder at vi først loader klassen i memory når den skal bruges
4. Den kan benyttes i et factory pattern
5. Indkapsler og skjuler afhængigheder
6. Nem at vedligeholde

Ulempe ved singleton

1. Når du har en applikation som benytter multitråde, kan du rende ind i en lang række problemer med singleton, med mindre at du arbejder med "locks"

Brugen af singleton

Du kan med fordele benytte et singleton pattern til logging, properties eller når du har en klasse som mange andre klasser skal have adgang til.

Singleton og tråde

Som tidligere nævnt kan du rende ind i en række udfordringer hvis din applikation arbejder med tråde. Du skal derfor være opmærksom på at din singleton implementering vil være en smule anerledes ved tråde. Du kan følge Microsofts guide lige her https://msdn.microsoft.com/en-us/library/ff650316.aspx

Senest ændret: onsdag den 10. januar 2018, 13:17

◄ Dependency injection eksempel

Spring til...

Opgave singleton : Chew and phew ►