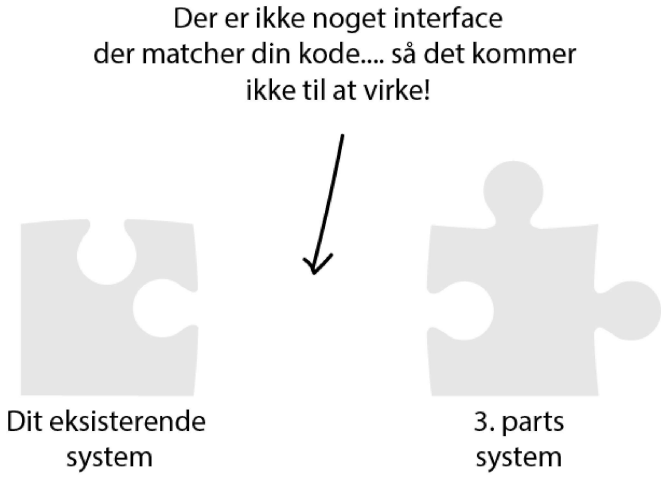
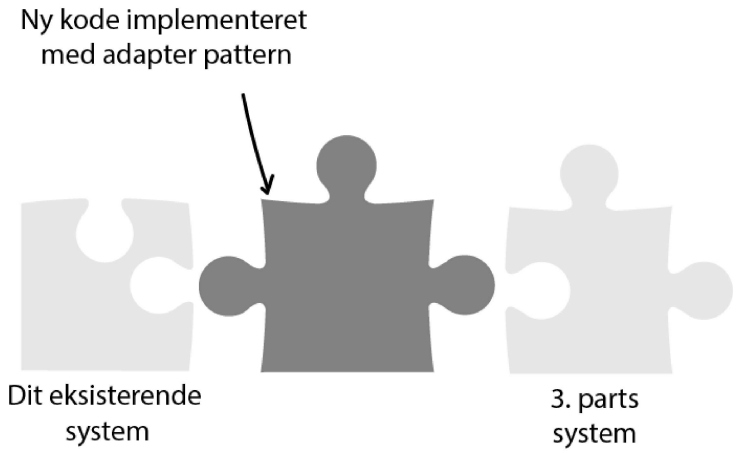
Marker som gennemført

Det sker ofte at vi som programmører render ind i nogle alvorlige problemer. Enten har vi ikke fået overholdt SOLID principperne (Det sker faktisk) andre gange vil vi gerne have 3 parts. klasser og objekter indarbejdet i vores eksisterende system, men grænsefladerne er forskellige i forhold til de grænseflader vi har defineret i vores eget system.



Det giver en del udfordringer, men det behøver det ikke. Et adapter pattern er designet til at indkapsle andre objekter, så de matcher det vi har behov for.



Du har sikkert tit været rendt i denne problematik.

Lad os forestille os at du har lavet en lille app, som kan afspille MP3 filer. Du har selvfølgelig lavet et fint interface og indkapslet din kode.

?

```
public interface IMediaPlayer
{
  void Play();
  void Next();
  void Stop();
  void Pause();
  void Prev();
}

public class MyMP3MediaPlayer : IMediaPlayer
{
  public void Next()
  {
     //go to next song
  }

  public void Pause()
  {
     //pause the current song
  }

  public void Play()
  {
     //Start playing
  }

  public void Prev()
  {
     //go to previous song
  }

  public void Stop()
  {
          //stop playing
  }
}
```

og din main kunne se således ud....

```
static void Main(string[] args)
{
  IMediaPlayer player = new MyMP3MediaPlayer();
  player.Play();
}
```

Nu er du så, så heldig at der er en gut der har implementeret et modul der gør det muligt at afspille MP4 som musik og dette modul vil du vildt gerne bruge. Problemet er bare at modulets offentlige metoder bare slet ikke passer til dine metoder. Og du har ikke lyst til at ændre på din kode (Og hvorfor nu det?)

MP4 modulet ser således ud, bemærk at modulet må være lavet af en dansker som ikke husker at kode på engelsk!

```
public class SuperDuperMP4Player
{

   public void AfspilSange()
   {
     //Super duper implementation her........
   }

   public void Pause()
   {
      //Super duper implementation her........
   }

   public void Stop()
   {
      //Super duper implementation her........
   }

   public void Næste()
   {
      //Super duper implementation her........
   }

   public void Forrige()
   {
      //Super duper implementation her........
   }
}
```

Problemet er at du ikke vil ændre på din eksisterende struktur, du har jo gjort det helt rigtige. Du har både brugt polymorfi og interfaces. Du kan heller ikke ændre i hans eksisterende kode. Du er istedet nødt til at indkapsle MP4 koden, så den passer til dit system - og det gøres ved hjælp af at implementere en adapter, som overholder dit interface og oprette en konstruktør der tager det objekt du ønsker at indkapsle med som parameter.

```
public class MP4PlayerAdapter : IMediaPlayer
{
  SuperDuperMP4Player player;

  public MP4PlayerAdapter(SuperDuperMP4Player player)
  {
    this.player = player;
  }

  public void Next()
  {
    throw new NotImplementedException();
  }

  public void Pause()
  {
    throw new NotImplementedException();
  }

  public void Play()
  {
    throw new NotImplementedException();
  }

  public void Prev()
  {
    throw new NotImplementedException();
  }

  public void Stop()
  {
    throw new NotImplementedException();
  }
}
```

Når det er gjort er det nemt at implementere de enkelte metoder så de passer.

```
public void Play()
{
   player.AfspilSange();
}
```

Sådan instantieres adapteren

```
IMediaPlayer playermp4 = new MP4PlayerAdapter(new SuperDuperMP4Player());
playermp4.Play();
```

Eksemplet her kan måske virke simpelt, men hvis du har en masse metodekalde som tager en IMediaPlayer med som parameter, hvis du kunne se fordelen langt bedre.

Senest ændret: tirsdag den 9. januar 2018, 13:19

◄ Opgave factory : Vito's pizza

Spring til...

Opgave adapter : Fight Arena ►