

---

# Adapter Design Pattern

---

RAPPORT

SWD, GRUPPE 1

*Udarbejdet af:*

201371052 - Alexander Kledal

201206129 - Rasmus Nielsen

201370939 - Dennis Frisgaard

201371000 - Anders Methmann



# Indholdsfortegnelse

<b>Kapitel 1</b>	<b>Adapter Pattern</b>	<b>2</b>
1.1	Intro . . . . .	2
1.2	Adapter Pattern . . . . .	2
1.3	Case . . . . .	3
1.4	Adapter Pattern løsning . . . . .	4
1.5	Sammenligning . . . . .	7
	1.5.1 Abstract Server Pattern . . . . .	7
1.6	Konklusion . . . . .	8

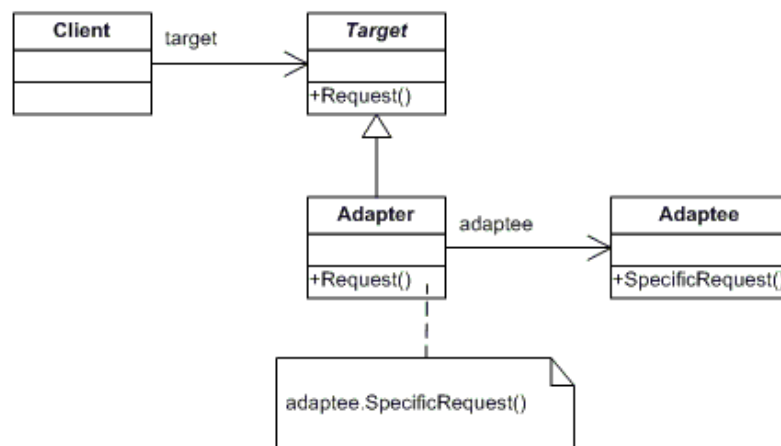
# 1 Adapter Pattern

## 1.1 Intro

Denne rapport omhandler Adapter pattern og vil beskrive teorien bag dette pattern, anvende det på en case, undersøge alternative patterns, og slutteligt konkludere på brugen af adapter pattern.

## 1.2 Adapter Pattern

Et adapter pattern anvendes til at gøre ikke kompatible systemer, kompatible. Dette kan fx være imellem et gammelt og nyt system, hvor det ikke er ønsket at skulle ændre det gamle system. Man kan således lave et adapter pattern til at gøre det nye system kompatibel, uden at ændre på det nuværende system. Adapter Pattern skal tilpasse adfærden på det nye system således det bliver kompatibel med det gamle system, uden at ændre i det gamle system (overholder open/close)



*Figur 1.1.* Eksempel på teoretisk adapter pattern

Det kan også ses på figur 1.1 ovenfor, hvordan en adapter anvender et interface og derved kan implementere "SpecificPrequest" uden at det påvirker det øvrige system. Denne metodik vil også blive demonstreret i sektion 1.4

Der er to typer Adapter pattern : Class og Object.

**Class adapter**

Class adapter anvender arv og kan kun wrappe klasser. Den kan altså ikke wrappe interfaces eftersom den skal nedarve fra en basisklasse

**Object adapter**

Object adapter derimod anvender komposition og kan wrappe både klasser og interfaces. Dette kan den eftersom, at den har en instans af det objekt den wrapper.

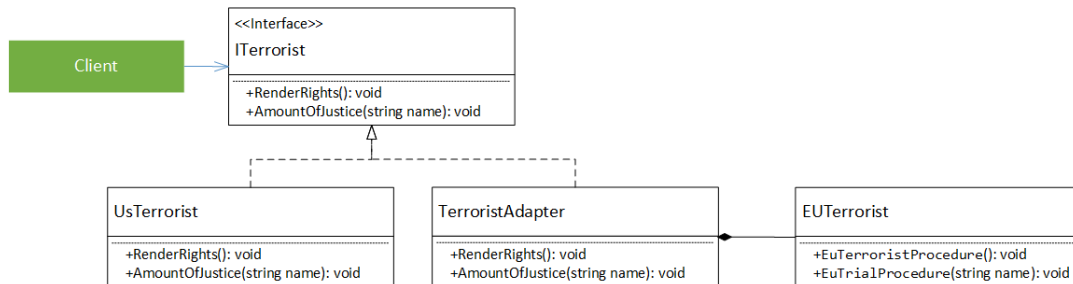
I TerroristCasen i afsnit 1.4 anvender vi objekt adapter pattern til at løse casen.

## 1.3 Case

Den følgende case er anvendt til at beskrive et problem, som kan løses med adapter pattern. Der vil blive refereret til denne case flere steder i rapporten under navnet "TerroristCase". I vores case har vi en database, der indeholder data på kendte terrorister, som bruges af den amerikanske myndighed CIA. Denne database vil den europæiske myndighed Interpol også gerne benytte sig af. Problemet ligger dog i at der er forskel på, hvordan man håndterer terrorister i USA og EU. De benytter altså ikke samme interface til deres terrorist håndtering. Der skal således laves en løsning med et adapter pattern, så de amerikanske myndigheder kan fortsætte uforstyrret med deres system samtidig med, at Interpol kan koble sig op på systemet, men bruge deres egen strafudmåling og procedure.

## 1.4 Adapter Pattern løsning

I denne løsning er der lavet to forskellige typer af terrorister, en amerikansk og en europæisk terrorist. Disse to forskellige terrorist typer implementere hvert deres interface. Den essentielle forskel i de to myndigheders interfaces omhandler, hvordan man strafudmåler en terrorist samt håndter rettigheder for terrorister. Funktionaliteten for den europæiske terrorist klasse passer ikke med terrorist interface, derfor er der lavet en adapter, så funktionaliteten for EUTerroristen bliver pakket ind, så det stemmer overens med Terrorist interface. På den måde kan EUTerrorists funktionalitet nu benyttes som om, EUTerrorist overholdte interface.



**Figur 1.2.** UML for implementeret adapter pattern i løsningen

Figur 1.2 viser et UML diagram for løsningen af casen. UsTerrorist klassen passer til interface, men det er nødvendigt med en TerroristAdapter for tilpasse EUTerrorist til interface.

```

12 2 references
13 public class UsTerrorist : ITerrorist
14 {
15     4 references
16     public void RenderRights()
17     {
18         Console.WriteLine("[Rights] Terrorist has no rights, they are not human and should not be treated like humans!");
19     }
20     4 references
21     public void AmountOfJustice(string terroristName)
22     {
23         Console.WriteLine("[Amount of justice] " + terroristName + " is going to Guantanamo to rot in hell!");
24     }
25 }
  
```

**Figur 1.3.** Implementering for UsTerrorist klassen

Figur 1.3 viser, hvordan metoderne fra ITerrorist interface er blevet implementeret.

```

5 2 references
6 public class EUTerrorist
7 {
8     2 references
9     public void EuTerroristProcedure()
10    {
11        Console.WriteLine("[Procedure for dealing with terrorists] Shoot terrorist - dead terrorist = good terrorist!");
12    }
13    2 references
14    public void EuTrialProcedure(string terroristName)
15    {
16        Console.WriteLine("[Trialprocedure for terrorist] " + terroristName + " has been shot on the street. Justice has been served");
17    }
18 }
  
```

**Figur 1.4.** Implementering for EUTerrorist klassen

Figur 1.4 viser den anderledes funktionalitet i EUTerrorist, som ikke overholder ITerrorist interface.

```
3 references
5 public class TerroristAdapter : ITerrorist
6 {
7     EuTerrorist _euTerrorist;
8     1 reference
9     public TerroristAdapter(EuTerrorist terrorist)
10    {
11        _euTerrorist = terrorist;
12    }
13    4 references
14    public void RenderRights()
15    {
16        _euTerrorist.EuTerroristProcedure();
17    }
18    4 references
19    public void AmountOfJustice(string name)
20    {
21        _euTerrorist.EuTrialProcedure(name);
22    }
23 }
```

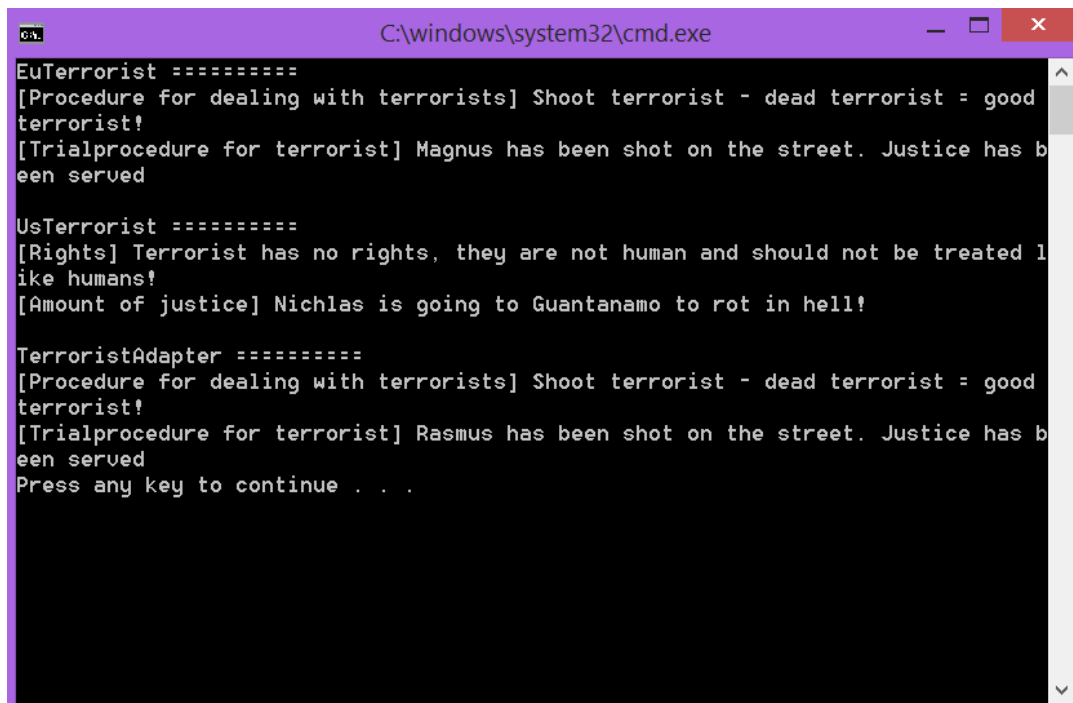
*Figur 1.5.* Implementering for TerroristAdapter

Figur 1.5 viser, hvordan adapterens implementering overholder ITerrorist interfacet og anvender metoderne fra EuTerrorist og på den måde tilpasse EuTerrorist klassen til ITerrorist interfacet.

Figur 1.6 er et testprogram, som først tester funktionaliteten i EuTerrorist, derefter UsTerrorist og til sidst TerroristAdapter. Resultatet på konsol vinduet kan ses på figur 1.7

```
11 static void Main(string[] args)
12 {
13     Console.WriteLine("EuTerrorist =====");
14     EuTerrorist euTerrorist = new EuTerrorist();
15     euTerrorist.EuTerroristProcedure();
16     euTerrorist.EuTrialProcedure("Magnus");
17     Console.WriteLine(" ");
18
19     Console.WriteLine("UsTerrorist =====");
20     UsTerrorist usTerrorist = new UsTerrorist();
21     usTerrorist.RenderRights();
22     usTerrorist.AmountOfJustice("Nichlas");
23     Console.WriteLine(" ");
24
25     Console.WriteLine("TerroristAdapter =====");
26     TerroristAdapter adapter = new TerroristAdapter(euTerrorist);
27     adapter.RenderRights();
28     adapter.AmountOfJustice("Rasmus");
29
30
31 }
32 }
```

Figur 1.6. Implementering for TerroristAdapter



```
C:\windows\system32\cmd.exe
EuTerrorist =====
[Procedure for dealing with terrorists] Shoot terrorist - dead terrorist = good
terrorist!
[Trialprocedure for terrorist] Magnus has been shot on the street. Justice has b
een served

UsTerrorist =====
[Rights] Terrorist has no rights, they are not human and should not be treated l
ike humans!
[Amount of justice] Nichlas is going to Guantanamo to rot in hell!

TerroristAdapter =====
[Procedure for dealing with terrorists] Shoot terrorist - dead terrorist = good
terrorist!
[Trialprocedure for terrorist] Rasmus has been shot on the street. Justice has b
een served
Press any key to continue . . .
```

Figur 1.7. Konsolvinduet, som viser udskriften, som afslører, at TerroristAdapter udskriver, som EUTeroorist metoderne

## 1.5 Sammenligning

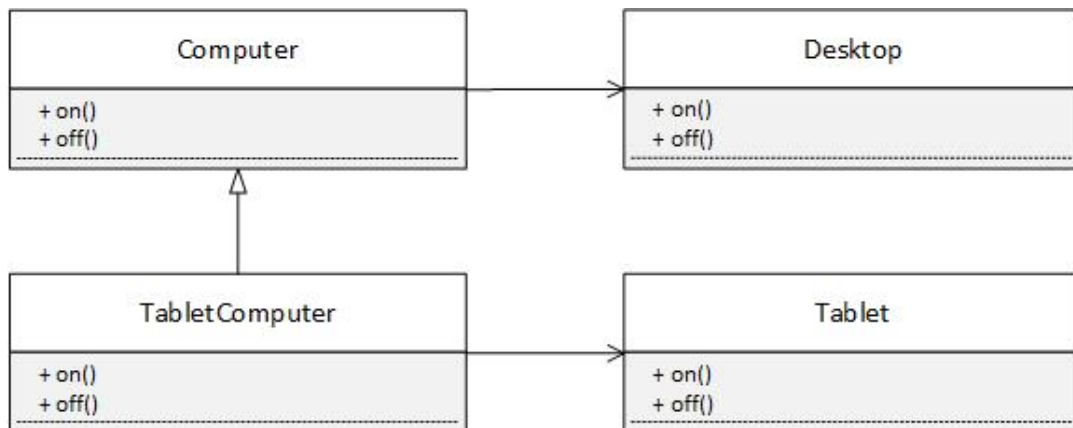
### 1.5.1 Abstract Server Pattern

For at forklare dette pattern, tager vi udgangspunkt i et eksempel for en computer. Til at starte med har vi kun en enkelt form for computer, en desktop computer, som kan tændes og slukkes, som det ses på figur 1.8.



*Figur 1.8.* Eksempel på enkel computer

Vi kommer dog til at bryde DIP og OCP, hvis vi skal udvide dette system til at en tablet computer også skal nedarve fra computer. På figur 1.9 er der tilføjet en `tabletComputer`, som nedarver direkte fra `Computer`. Denne udvidelse af `Computer` bryder DIP.

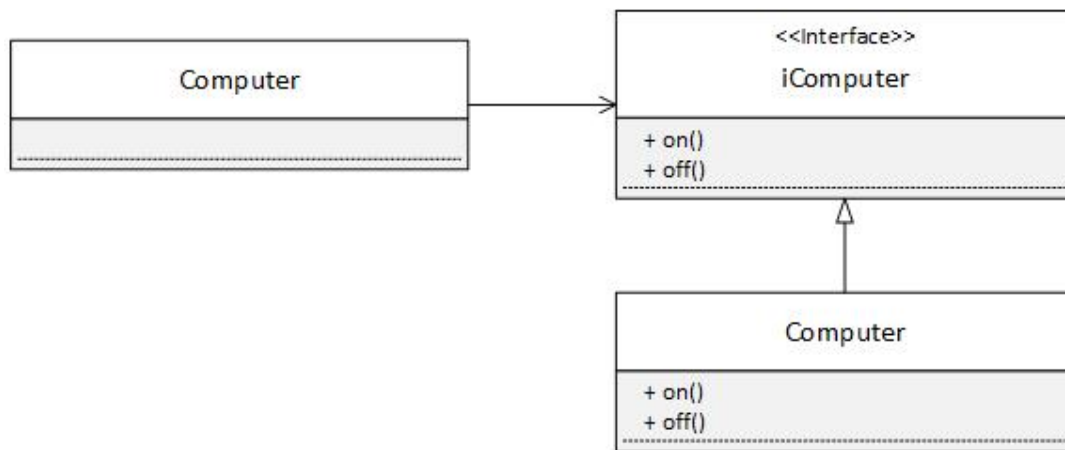


*Figur 1.9.* Dårlig løsning på udvidelse

Vi vil derfor gerne anvende et pattern, hvor vi kan tilføje en ny computer uden at bryde DIP. Her kommer Abstract Server Pattern til undsætning.

Ved et abstract server pattern laves der et interface således, at højniveau delene ikke har en direkte forbindelse til lavniveau delene, men har en association til et interface. Computeren eller evt. en tablet nedarver fra dette interface, og derved laver de to forskellige implementeringer af `on` og `off`. På figur 1.10 er et UML diagram for abstract server pattern.





**Figur 1.10.** Abstract server pattern

Ved brug af abstract server pattern undgår vi at bryde OCP samt DIP. Vi undgår at bryde OCP, da vi til hver tid kan implementere en ny computer uden, at dette vil gøre, at vi skal ændre i computer klassen. Desuden er lavniveau delen med Desktop og Tablet afhængig af interfacet iComputer, hvilket gør at high level delen Computer ikke har en direkte forbindelse til low level delen, hvilket gør, at DIP bliver overholdt.

Et adapter pattern er en udvikling af abstract server pattern. Et adapter pattern vil blive anvendt så snart, at det et abstract server pattern har en anden implementering end en anden klasse har. Dette kun fx være en laptop computer, som kun kan sleep og wakeUp, og altså ikke har on og off.

## 1.6 Konklusion

Det kan konkluderes ud fra vores erfaringer med vores case, at adapter patteren er meget anvendeligt, når det skal implementeres i et allerede eksisterende system og udgifterne eventuelt er for store til at foretage en refaktorering af sit eksisterende system til at overholde de nye krav.

Det er ressourcekrævende at anvende adapter pattern og der er derfor vigtigt, at man bygger en stærk designmodel således, at man undgår at anvende adapter pattern.