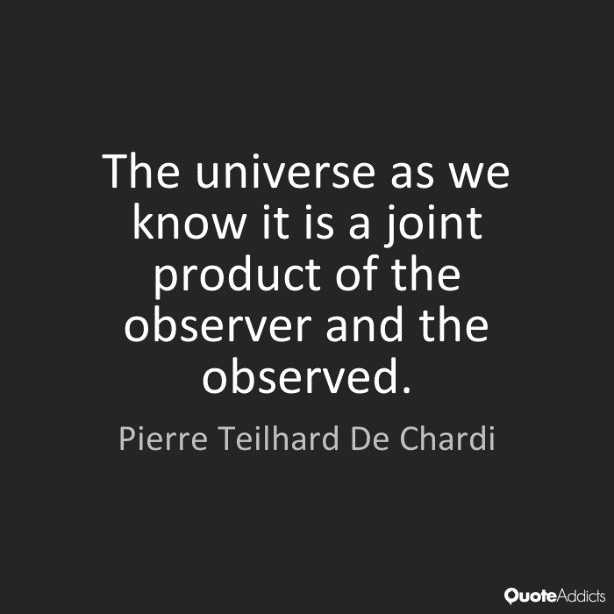
# Observer Pattern

|  |  |
| --- | --- |
| **Læringsmål** | Du kan:   * **1Pf1**: anvende centrale metoder til at specificere og konstruere algoritmer […] * **1Pf2**: anvende centrale faciliteter i programmeringssproget til realisering af algoritmer, **designmønstre**, abstrakte datatyper, datastrukturer, designmodeller og brugergrænseflader. |
| **Forventet læringsudbytte [SOLO]** | Programmeringssporet:   * [Unistrukturel] Du kan genkende enkelte kendetegn ved **designmønstre** –Observer Pattern * [Multistrukturel] Du kan beskrive flere kendetegn ved **C# klasse (objekt)** – specielt med fokus på abstrakt klasse, abstrakt metode og override |
| **Din forberedelse** | Programmeringssporet:   * [Observer Pattern Overview](https://www.linkedin.com/learning/c-sharp-design-patterns-part-1/observer-pattern-overview?u=57075649) (video: 2:23) * [Observer Pattern](https://www.oodesign.com/observer-pattern.html) * [Observer Design Pattern](https://www.dofactory.com/net/observer-design-pattern) * [C# Abstraction](https://www.w3schools.com/cs/cs_abstract.asp) (abstrakt klasse og abstrakt metode; bemærk at abstraktion kan opnås på flere måder end nævnt i artiklen) * [Override](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/override) * Ekstra:   + Læs ”GOF - Observer Pattern.pdf” (er på ItsLearning) |

I denne opgave (og de følgende) skal du se nærmere på forskellige implementeringer af Observer-mønstret. Formålet er både at introducere Observer-mønstrets funktion og formål samt at repetere anvendelsen af abstrakte klasser, som du har set i tidligere opgaver.

Observer-mønstret er et særdeles vigtigt designmønster, som bl.a. WPF benytter sig af i XAML-databinding til at sikre en løsere kobling (GRASP-princip: low coupling) i en-til-mange relationer mellem objekter.

# Dagens ord



# Øvelse 1: Terminologi

Del teamet op i mindre grupper og brug **Ordet rundt** til at reflektere over begreberne ”observer-mønstret”, ”abstrakt klasse”, ”abstrakt metode” og ”override”.  
Sørg for at alle får mulighed for at tale.

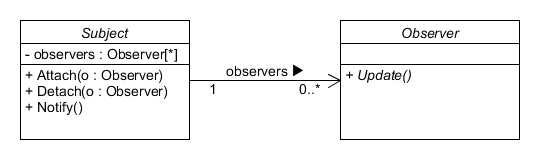
*Tidsramme: 20 minutter*

# Øvelse 2: Repræsentation af Observer-mønstret med abstrakte klasser

Observer-mønstret kan realiseres på forskellige måder i C#. Du ser i denne opgave først på abstrakte klasser. **Benyt parprogrammering.**

## Øvelse 2.1: Overvej sammenhænge og afhængigheder

De to roller i Observer-mønstret kan udtrykkes ved to abstrakte klasser (*klassenavn er i kursiv*):



*Bemærk, at Update-metoden i Observer-klassen er abstrakt (kursiv).*

Ud fra dagens forberedelse, overvej sammenhænge, afhængigheder og roller mellem de to klasser i deres metoder og felter for at afklare, hvordan klasserne udtrykker Observer-mønstret. Skriv dine observationer ned f.eks. i form af krav eller lignende.

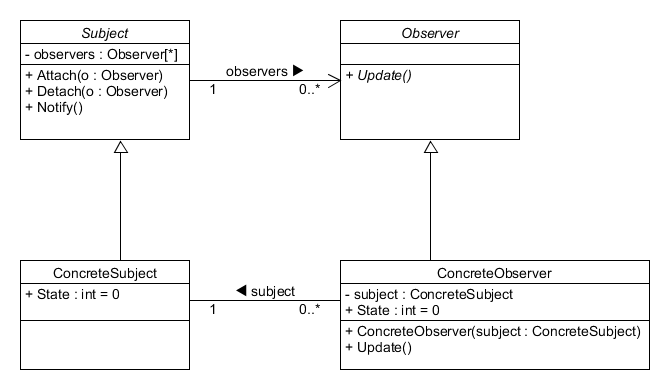
*Tidsramme: 10 minutter*

## Øvelse 2.2: Opret konsol-applikation og implementér abstrakte klasser

Opret en konsol-applikation (’Console Application’ i .NET 5), og implementér de to abstrakte klasser foroven med deres metoder ud fra de funktionelle sammenhænge, du har identificeret mellem klasserne. Bemærk at ’private’ attributter (startende med lille bogstav) i modellen skal realiseres som private felter i C#.

# Øvelse 3: Realisering af Observer-mønstret nu med konkrete klasser

Klassediagrammet udvides nu med to nye klasser ConcreteSubject og ConcreteObserver, der konkret realiserer Subject- og Observer-klassen via nedarvning:



Hver af de to konkrete klasser repræsenterer sin tilstand med et simpelt heltal i en State-property.

Udvid din applikation med de to nye klasser i henhold til klassediagrammet, således, at:

* Multiple instanser af ConcreteObserver bliver notificeret, når den samme instans af ConcreteSubject ændrer tilstand, dvs. dens State-property ændrer værdi
* De notificerede ConcreteObserver-objekters reaktion er at sætte deres State-property til samme værdi som den tilsvarende property i ConcreteSubject

Implementér ovenstående klassediagram i C# uden at ændre Subject- og Observer-klasserne.

Udfør dernæst følgende:

* Opret et testprojekt i løsningen (’MSTest Test Project’).
* Indsæt teksten fra ”Ex31-ObserverPattern.UnitTest.docx” i unit-testen (kan hentes fra Git materiale-folderen)
* **Læs unit-testen grundigt igennem, så du ved, hvad det er, den tester. Reflektér sammen med din partner.**
* Kør testen. Ret din kode, indtil testen kører igennem.

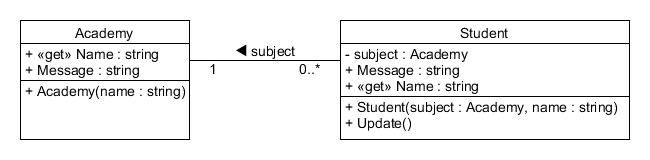
# Øvelse 4: Meddelelsessystem til akademiet

Et vist erhvervsakademi vil gerne give besked til alle sine studerende (dvs. dem der gerne vil modtage beskeder), når der er en fælles nyhed til dem. Du skal til dette formål udvikle et lille system, der netop benytter Observer-mønstret.

Benyt de nedenstående punkter (og erfaringen fra de forrige øvelser) til at realisere systemet:

* Erhvervsakademiet repræsenteres i systemet ved klassen ’Academy’
* Academy-klassen indeholder en ’Message’-property, som indeholder den besked eller nyhed, der ønskes meddeles til de abonnerende studerende. Derudover har klassen også en Name-property, der angiver navnet på akademiet. Dette navn kan kun sættes, når klassen instantieres
* En studerende repræsenteres ved klassen Student. Denne klasse har også en Name-property, der angiver navnet på den studerende, som også kun kan sættes, når klassen instantieres. En studerende kan abonnere på nyheder fra et akademi og får via en Update-besked at vide, når en ny nyhed er kommet
* En studerendes respons ved modtagelse af en nyhed fra et akademi er (via konsolvinduet) højlydt at proklamere modtagelsen i en sætning, hvor både den studerendes navn, nyheden og akademiets navn indgår, f.eks. ”Studerende <navn> modtog nyheden <nyhed> fra akademiet <navn>”

Dette kan delvist udtrykkes i klassediagrammet:



*Husk følgende navnekonvention anvendes:*

* *En operation (metode i C#) med samme navn som klassen indikerer en constructor*
* *En ”public”-attribut startende med stort bogstav er en property (med både get og set)*
* *En ”private”-attribut startende med lille bogstav er et felt*

Udfør følgende:

* Overvej, hvordan disse klasser kan realisere Observer-mønstret ved genbrug af de abstrakte klasser fra øvelse 2.1
* Implementér derefter meddelelsessystemet

Efter implementeringen:

* Indsæt følgende kode i Main-metoden:

static void Main(string[] args)

{

var p = new Academy("UCL");

var s1 = new Student(p, "Jens");

var s2 = new Student(p, "Niels");

var s3 = new Student(p, "Susan");

p.Attach(s1);

p.Attach(s2);

p.Attach(s3);

p.Message = "Så er der julefrokost!";

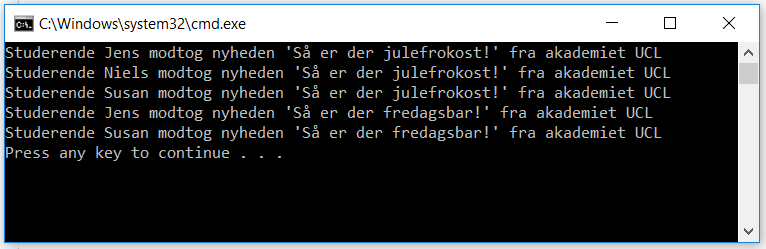
p.Detach(s2);

p.Message = "Så er der fredagsbar!";

Console.ReadLine();

}

* Kør konsolprogrammet, så du f.eks. får følgende output i konsolvinduet:



# Øvelse 5: Prisskiltesystem til tankstationer

Et helt nyt tankselskab RocketFuel har åbnet en hel del tankstationer rundt om i Danmark med ambitioner om at få afviklet biltrafikken hurtigere. På hver tankstation er der opsat et lysskilt, der viser priserne på deres tre tilgængelige brændstoftyper (baseret på flydende ilt), nemlig HydroOxygen, KeroOxygen, AlcoOxygen.

Kravene til systemet er:

* Brændstofpriserne styres centralt fra selskabets hovedkontor. Her fastlægges literprisen for KeroOxygen. HydroOxygen er altid 10% dyrere end KeroOxygen, og AlcoOxygen er tilsvarende 10% billigere end KeroOxygen
* En tankstation skal være placeret i en navngiven by med angivelse af hvilken region (landsdel), byen ligger i, enten Sjælland, Fyn eller Jylland
* Brændstofpriserne fra hovedkontoret tilpasses regionalt, så niveauet er 5% dyrere på Sjælland og 5% billigere på Fyn
* Når hovedkontoret ændrer sine brændstofpriser, skal denne ændring (tilpasset den enkelte region) automatisk afspejles på alle selskabets lysskilte i Danmark
* Når et lysskilt konkret ændrer sine priser, skrives de nye priser ud i konsolvinduet med angivelse tankstationens bynavn og priserne for de 3 brændstoftyper

Benyt evt. følgende oversættelser:

|  |  |
| --- | --- |
| Dansk | Engelsk |
| Tankselskab | Gas company |
| Tankstation | Gas station |
| Lysskilt til priser | Price board |

## Øvelse 5.1: Lav DCD

Overvej hvilke ’subjects’ og ’observers’, der er i systemet, dvs. hvem lytter til hvad. Fastlæg selv ud fra dette, hvorledes klasserne for systemet skal opbygges.

Tegn klassediagrammet (DCD) for systemet, når du er blevet enige med din kodepartner.

## Øvelse 5.2: Implementér prisskiltesystemet

Implementér derefter dette prisskiltesystem i en konsolapplikation med brug af Observer-mønstret foroven (brug abstrakte klasser).

Skriv testkode i Main()-metoden, hvor du opretter tankselskabet RocketFuel og tilknytter mindst tre tankstationer til tankselskabet placeret i hver sin region. Lad testkoden ændre selskabets priser flere gange, så du kan se resultatet i konsolvinduet.

## Øvelse 5.3: Udvidelse

Udvid systemet med følgende krav:

* Det er tilladt for en tankstation selv lokalt at tilbyde 10% rabat samtidigt på alle priser på tidspunkter, hvor der lokalt er få bilister på vejene

Implementér dette krav, og udvid testkoden, så du får vist resultatet, når tankstationen tilbyder rabat lokalt.