# Observer Pattern

# Øvelse 1: Terminologi

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

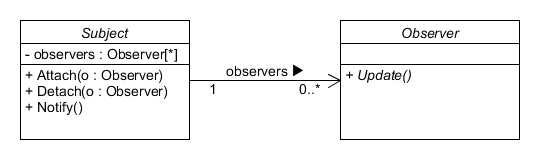
*Tidsramme: 20 minutter*

# 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 designklasser (*klassenavn er i kursiv*):



*Bemærk, at Update-operationen i Observer-klassen er abstrakt (vist med kursiv).*

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

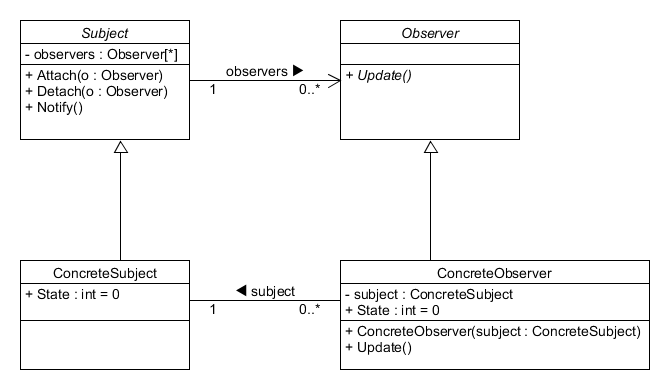
*Tidsramme: 10 minutter*

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

Opret en konsol-applikation (’Console App’), og implementér de to abstrakte klasser foroven i C# med deres operationer 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 designklasser 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-attribut. Bemærk at en public attribut (startende med stort bogstav) i modellen skal realiseres som en public property i C#.

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 ”Uge21\_ObserverPattern.UnitTest.docx” i unit-testen.
* **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.

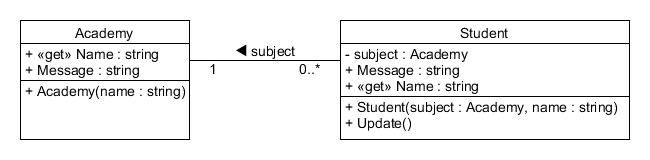
# Øvelse 4: Meddelelsessystem til akademiet

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 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!";

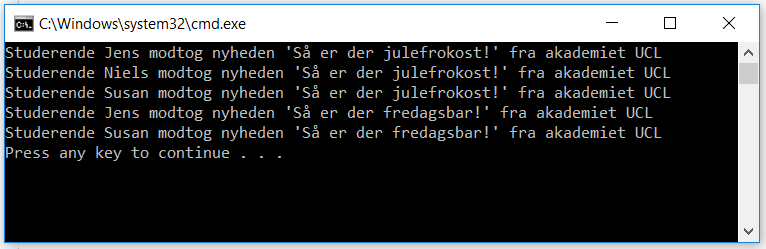
p.Detach(s2);

p.Message = "Så er der fredagsbar!";

Console.ReadLine();

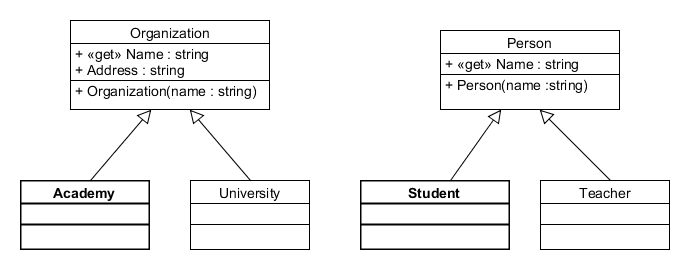
}

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



# Øvelse 5: Videreudvikling af meddelelsessystemet

Efter nogle iterative workshops med interessenterne til meddelelsessystemet fra forrige opgave (øvelse 4), er man kommet frem til en ny version af meddelelsessystemet. Du bliver nu nødt til at tage udgangspunkt i følgende klassediagram, før du begynder at realisere Observer-mønstret (bemærk at de to klasser Organization og Person er introduceret som et vilkår fra et andet projekt og kan ikke ændres):



Academy- og Student-klasserne indgår stadig i modellen som før, men de nedarver nu hver især fra henholdsvis Organization- og Person-klasserne.

Udfør følgende:

* Implementér ovenstående rettelse til diagrammet i din eksisterende løsning. Du behøver ikke implementere University- og Teacher-klasserne, men du skal have Organization- og Person-klasserne med og huske nedarvningen til henholdsvis Academy og Student
* Overvej nu, hvorledes man **med abstrakte klasser** kan implementere et Observer-mønster mellem Student og Academy i din konsol-applikation uden på nogen måde at påvirke de 4 andre klasser (Organization, University, Person, Teacher)

Er der et problem? Og hvad er det?

# Øvelse 6: Repræsentation af Observer-mønstret med interfaces

En alternativ måde at implementere Observer-mønstret på er ved hjælp af interfaces. Interfaces gør bl.a. implementeringen helt uafhængig af klassehierarkiet, hvilket er en fordel i den nye version af systemet i øvelse 5.

Vælges interfaces til at realisere Observer-mønstret, tilrettes klassediagrammet i øvelse 2 til følgende (de uberørte klasser og relationer er farvet i gråt):

Et billede, der indeholder tekst, diagram, linje/række, Plan

Automatisk genereret beskrivelse

Udfør følgende:

* Studér klassediagrammet nøje
* Implementér ovenstående meddelelsessystemet med brug af interfaces

Efter implementeringen indsæt følgende kode i Main-metoden:

static void Main(string[] args)

{

Academy p = new Academy("UCL", "Seebladsgade");

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

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

Student 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!";

}

Som i den forrige opgave får du f.eks. følgende output i konsolvinduet:

Et billede, der indeholder tekst, elektronik, skærmbillede, display/skærm/fremvisning

Automatisk genereret beskrivelse