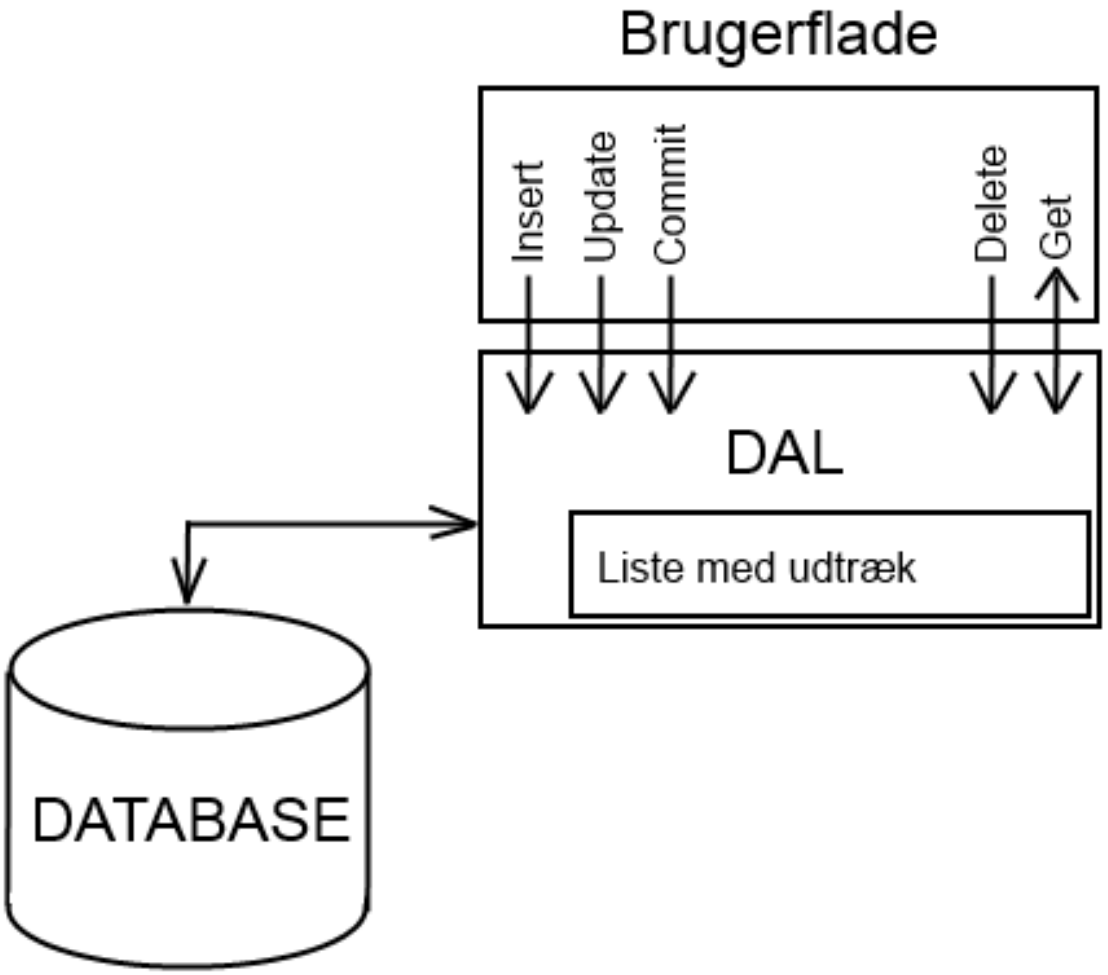
I den foregående opgave lavede vi en Person Class, der kunne indeholde få simple informationer omkring en person. Ikke specielt realistisk, men som eksempel ganske godt.

Vi skal i denne opgave prøve at lave et separat datalag, som kan benyttes ikke kun i den tilhørende WPF applikation, men også i en Console App en ASP.NET eller WCF service, hvis det skulle blive nødvendigt.

Herunder er vist et muligt scenario, men der er kun en af mange andre løsninger på denne opgave.

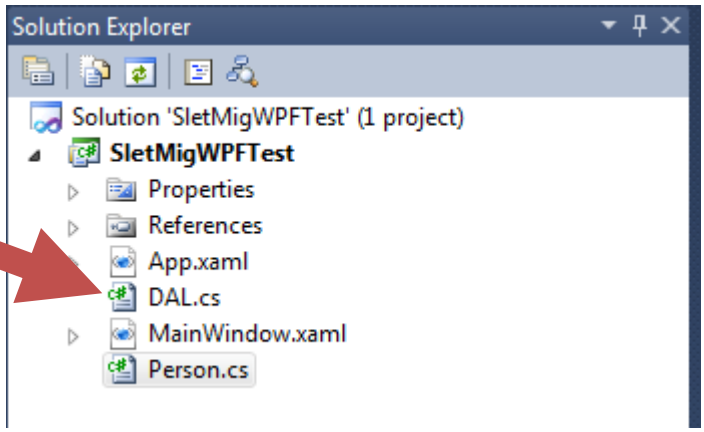


Husk på at dette blot er en simulering af et meget simpelt Datalaget. Og intensionen er kun at vise hvorledes en forbindelse til eksempelvis en SQL database, XML fil eller en web service kunne laves.

Start med at lave et nyt WPF projekt og tilføj din gamle person class fra det foregående projekt. Du kan enten copy paste koden direkte over i en nytilføjet fil, eller vælge at tilføje filen med Person classen, fra den gamle projekt mappe.

Her efter skal vi i gang med at lave selve datalaget.

Så du tilføjer også en ny Class fil, via Solution explorer.



Så skal vi i gang med den indledende kode. Men inden du går i gang er der et par småting du skal være klar over. For det første har vi ikke adgang til en database i dette eksempel. Derfor simulerer vi denne forbindelse med en privat liste kaldet "DataBase". Denne liste er der kun adgang til, internt i DAL classen. MEN det kunne lige så godt have været en SQL database, et XML dokument eller en Web service som vi hentede den pågældende data i.

Herunder ser du den indledende kode for DAL classen:

```
public class DAL
{
    private ObservableCollection<Person> DataBase; // Da vi ikke har adgang til en database,
                                                   // simulerer vi med denne private liste....

    private ObservableCollection<Person> _publicListe; // Dette er objektet med elementer vi
                                                       // "deler ud" til brugeren af vores class.

    //Constructoren genererer data til vores falske database
    public DAL()
    {
        DataBase = new ObservableCollection<Person>();
        DataBase.Add(new Person(0, "Svend", "Bendt", 1234));
        DataBase.Add(new Person(1, "Bein", "Stagge", -987654321));
        DataBase.Add(new Person(2, "Turt", "Khorsen", 0));
        DataBase.Add(new Person(3, "Gill", "Bates", int.MaxValue));

        _publicListe = new ObservableCollection<Person>();
    }
}
```

Det eneste der er nyt i kode herover, er at vi ikke længere benytter os af en List<> men en ObservableCollection (Som er en anden slags Liste). Grunden til dette, er at en almindelig liste ikke kan melde tilbage omkring ændringer til selve listen. Men det kan en ObservableCollection. Og så skal vi ikke selv opdatere vores brugerflade, hver gang vi laver ændringer til den lokale liste med de kommende insert, delete og update funktioner...

Derudover kan du se at vi laver en lokal liste med navnet "Personer", dette gør vi for at simulere overførsel af data, fra eksempelvis en database og over i en lokal datastore. (dette kaldes nogle gange for disconnected data, da det ikke længere har nogen direkte forbindelse til den oprindelige database.) Denne lokale liste kan vi så ændre i, som vi har lyst til. Og når vi er tilfredse med ændringerne, kan vi opdatere tilbage til databasen...

Men først skal vi have fyldt denne lokale liste med data fra vores fiktive database. Og for at tingene skal kunne virke, er vi nød til at kopiere hvert enkelt element i database listen, til Personer listen. Dette gøres på følgende måde:

```
// Get henter data fra databasen og kopierer det over i den lokale kopi
public ObservableCollection<Person> Get()
{
    _publicListe.Clear();     //Først tømmes den lokale kopi

   //Så løber vi alle elementerne igennem i databasen og overfører til lokal kopi
    foreach (Person p in DataBase)
    {
        _publicListe.Add(p);
    }

    return _publicListe;
}
```

Og for at opdatere tilbage til den simulerede database (Committe vores ændringer), kan vi i dette eksempel bare kopiere hele vores Personer Liste direkte over i database filen.

Dette er selvfølgelig ikke den korrekte måde at gøre det på hvis vi havde adgang til en reel database, da det hurtigt kunne bliver til nogle temmelig lange transaktioner. Men hvordan det skal gøres rigtigt i database sammenhæng, vil vi først kigge nærmere på, når vi kommer til at arbejde med ADO.NET.

Herunder er vist hvordan en Commit funktion kunne se ud:

```
// Commit indsætter vores lokale kopi af data, i databasen
public void Commit()
{
    DataBase = new ObservableCollection<Person>(_publicListe);
}
```

Du har nu en DAL.cs fil der ser ud nogenlunde som på næste side.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
using System.Collections.ObjectModel;

namespace SimpelDAL
{
    public class DAL
    {
        // Da vi ikke har adgang til en database, simulerer vi med denne private liste....
        private ObservableCollection<Person> DataBase;

        // Dette er objektet med elementer vi "deler ud" til brugeren af vores class.
        private ObservableCollection<Person> _publicListe;

        //Constructoren genererer data til vores falske database
        public DAL()
        {
            DataBase = new ObservableCollection<Person>();
            DataBase.Add(new Person(0, "Svend", "Bendt", 1234));
            DataBase.Add(new Person(1, "Bein", "Stagge", -987654321));
            DataBase.Add(new Person(2, "Turt", "Khorsen", 0));
            DataBase.Add(new Person(3, "Gill", "Bates", int.MaxValue));

            _publicListe = new ObservableCollection<Person>(DataBase);
        }

        // Get henter data fra databasen og kopierer det over i den lokale kopi
        public ObservableCollection<Person> Get()
        {
            _publicListe.Clear();     //Først tømmes den lokale kopi

            //Så løber vi alle elementerne igennem i databasen og overfører til lokal kopi
            foreach (Person p in DataBase)
            {
                _publicListe.Add(p);
            }

            return _publicListe;
        }

        // Commit indsætter vores lokale kopi af data, i databasen
        public void Commit()
        {
            DataBase = new ObservableCollection<Person>(_publicListe);
        }
    }
}
```

Så mangler der bare Insert, Delete ikke mindst Update funktioner. Men dem skal i selv lave! Prøv om i, i grupper af 2 kan finde en brugbar løsning. Et hint kunne eksempelvis være at kigge på ObservableCollection i Help (F1) for at se om den har nogle funktioner i kan benytte i de to Insert og Delete Funktioner i skal tilføje til jeres DAL.

I Update og Delete funktionen kunne ”for” eller ”foreach” løkkerne og Person.ID helt sikkert være anvendeligt...