M226 – Objektorientiert implementieren

1 Einstieg OOP

1.1 Grundlegende Konzepte der objektorientierten Programmierung

1.1.1 schlecht am alten „strukturierten“ Ansatz

- Keine Schutzmöglichkeit der Daten
- Daten und Funktionen gehören nicht zusammen

1.2 Idee des objektorientierten Ansatz

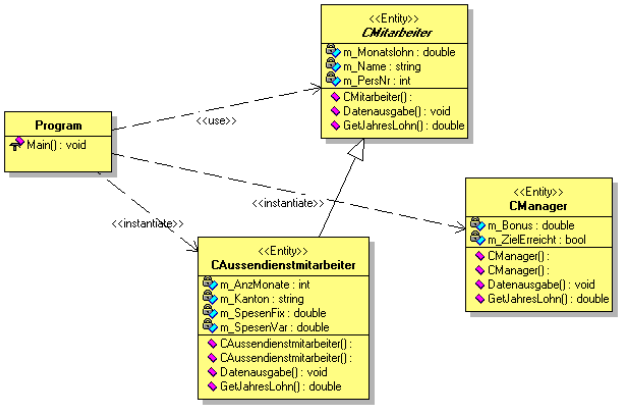
In der OOP gehören Funktionen und Variablen zu einem bestimmten Objekt. Man spricht dann von Eigenschaften oder Membervariablen und Methoden oder Memberfunktionen.

Bei der OOP bilden die Objekte eine Einheit aus:

- **Eigenschaften(Daten)**
- **Methoden (Funktionen)**

1.3 Klassen und Objekte

Klassen sind eine Art Bauplan für Objekte.



Das Erstellen eines Objektes nennt man Instanziierung.

1.4 Datenkapselung

Objektorientierte Programmiersprachen bieten Sprachelemente, mit deren Hilfe man den Zugriff auf Eigenschaften und Methoden einschränken bzw. sperren kann. Hierzu gehört die Möglichkeit, die Elemente einer Klasse als *private* oder *public* zu deklarieren:

- Auf private-Elemente einer Klasse können nur die Methoden dieser Klasse zugreifen. Diese sind damit vor dem direkten, unkontrollierten Zugriff von aussen geschützt.
- Die public-Elemente bilden die öffentliche Schnittstelle. Auf sie kann uneingeschränkt zugegriffen werden.

In einer objektorientierten Anwendung sollten alle Eigenschaften vor Zugriffen von aussen geschützt sein (=private) und nur über die öffentlichen Methoden (=public / Schnittstelle) verändert werden können.

Die Summe aller öffentlichen Elemente einer Klasse nennt man Schnittstelle.

1.6 Konstruktoren / Destruktoren

ein Konstruktor ist eine ganz spezielle Methode, die automatisch bei der Erzeugung eines Objektes aufgerufen wird.

```
//Standard-Konstruktor
public CMitarbeiter()
        {
            setPersNr(1);
            setName("Neue Person");
            setMonatslohn(1000.00);
        }

//erw. Konstruktor
public CMitarbeiter(int persnr, string
name, double lohn)
        {
            setPersNr(persnr);
            setName(name);
            setMonatslohn(lohn);
        }
```

1.7 Properties (get / set-Methoden)

get und set-Methoden verwendet man für den Zugriff auf die privaten Membervariablen.

```
public void setPersNr(int nummer)
        {
            m_PersNr = nummer;
        }

public int getPersnr()
        {
            return m_PersNr;
        }
```

1.8 Statische Eigenschaften, statische Methoden

Statische Methoden dürfen nur auf statische Eigenschaften zugreifen.

2 Vererbung

2.1 Die Vererbung

Vererbung bei objektorientierten Programmiersprachen bedeutet, dass alle Elemente einer Klasse an eine andere Klasse vererbt werden. Die Klasse, welche vererbt wird nennt man **Basis- oder auch Superklasse**. Die Klasse die erbt kann man noch um weitere Elemente (Eigenschaften und Methoden) erweitern oder bestehende Methoden überschreiben.

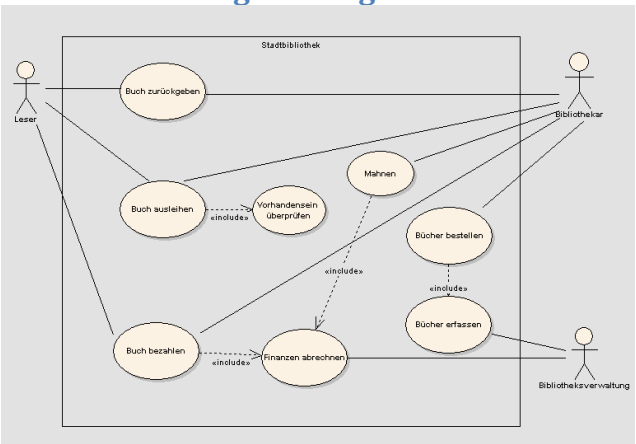
```
using System;
using System.Collections.Generic;
using System.Text;

namespace Teko.Programmieren2.Jumlitest
{
    class CManager:CMitarbeiter    ← Vererbung
    {
        //Eigenschaften
                    private double m_Bonus;
                    private bool m_ZielErreicht = false;    //gilt
sofern in einem Konstruktor


          //nicht anders definiert wurde
                    //Konstruktoren
                    public CManager(int persnr, string name,
double lohn, double bonus,bool
zielerreicht):base(persnr,name,lohn)
                    {
                              setPersNr(persnr);
                              setName(name);
                              setMonatslohn(lohn);
                              setBonus(bonus);
            setZielErreicht(zielerreicht);
                    }
                    //Propreties
                    public double getBonus() {
                               return m_Bonus;
                    }
        public void setBonus(double value)
        {
            m_Bonus = value;
        }
                    public bool getZielErreicht() {
                              return m_ZielErreicht;
                    }
                    public void setZielErreicht(bool value) {
            m_ZielErreicht = value;

                    }

        public override double getJahresLohn()
        {
            if (getZielErreicht())
                return (getMonatslohn()*13 + getBonus());
            else
                return (getMonatslohn()*13);

        }

        public override void Datenausgabe()
        {
            System.Console.ForegroundColor = ConsoleColor.Yellow;

System.Console.Out.WriteLine("*****************************");
            System.Console.Out.WriteLine("Typ:\tManager");
            System.Console.Out.WriteLine("PersNr:\t" +
getPersnr());
            System.Console.Out.WriteLine("Name:\t" + getName());
            System.Console.Out.WriteLine("Bonus
erhalten:\t"+getZielErreicht());
            System.Console.Out.WriteLine("Bonus:\t" +
getBonus());
            System.Console.Out.WriteLine("JahresLohn:\t" +
getJahresLohn());

System.Console.Out.WriteLine("*****************************");
        }
    }
}
```

Wenn von einer (Basis-)Klasse keine Objekte erstellt werden sollen, kann man die Klasse als abstrakt definieren.
abstract vor den Klassennamen schreiben

```
public abstract class CMitarbeiter
```

3 UML-Anwendungsfalldiagramm

3.1 Anwendungsfalldiagramme



Anwendungsfalldiagramme (engl. Use Case Diagrams) modellieren die Funktionalität des Systems auf einem hohen Abstraktionsniveau aus der sogenannten Black-Box-Sicht des Anwenders. Es werden die Anwendungsfälle definiert, die ein externer Anwender wahrnehmen kann und deren Ausführung ihm einen erkennbaren Nutzen erbringt.

Die Modellierung beschreibt, was für Anwendungsfälle das System anbietet und nicht, wie sie im System realisiert werden!

Mit ihrer Hilfe werden Anwenderwünsche erfasst und dokumentiert.

Von den Anwendungsfalldiagrammen ausgehend, die eine grobe Sicht auf die Funktionalität des Systems darstellen, kann das dynamische Verhalten in einem Top-Down-Verfahren durch den Einsatz weiterer Verhaltensdiagramme verfeinert und präzisiert werden.

3.2 Notationselemente

3.2.1 Systemgrenze

Die Systemgrenze (engl. System Boundary) umfasst ein System das die benötigten Anwendungsfälle bereitstellt und mit dem die Anwender interagieren.

3.2.2 Akteur

Ein Akteur (engl. Actor) modelliert einen Typ oder eine Rolle, die ein externer Benutzer oder ein externes System während der Interaktion mit einem System einnimmt.

3.2.3 Anwendungsfall

Ein Anwendungsfall (engl. Use Case) spezifiziert eine abgeschlossenen Menge von Aktionen, die von einem System bereitgestellt werden und einen erkennbaren Nutzen für einen oder mehrere Akteure erbringen.

3.2.4 Assoziation

Eine Assoziation (engl. Association) modelliert in Anwendungsfalldiagrammen eine Beziehung zwischen Akteuren und Anwendungsfällen.

3.2.5 Generalisierung / Spezialisierung

eine Spezialisierung (engl. Generalization) kann in Anwendungsfalldiagrammen zwischen Akteuren oder Anwendungsfällen modelliert werden und definiert eine Beziehung zwischen einem spezifischen und einem allgemeinen Element.

3.2.6 Include Beziehung

Eine Include-Beziehung (engl. Include Relationship) modelliert eine unbedingte Einbindung der Funktionalität eines Anwendungsfalls in einen anderen Anwendungsfall.

Jedes Mal, wenn der einbindende Anwendungsfall ausgeführt wird, muss auch der eingebundene Anwendungsfall aufgerufen werden. Der einbindende Anwendungsfall ist abhängig vom Ausführungsergebnis des eingebundenen Anwendungsfalls und ist damit ohne ihn nicht vollständig. Insgesamt kann die Include-Beziehung auch mit dem Aufruf einer Unterfunktion verglichen werden.

3.2.7 Extend-Beziehung

Eine Extend-Beziehung (engl. Extend Relationship) modelliert die bedingte Einbindung der Funktionalität eines Anwendungsfalls in einen weiteren Anwendungsfall.

Die Funktionalität des erweiterten Anwendungsfalls kann in die Funktionalität des erweiterten Anwendungsfalls am Erweiterungspunkt eingebunden werden. Im Unterschied zu einer Include-Beziehung ist der erweiterte Anwendungsfall vom erweiterten unabhängig und kann auch ohne ihn ausgeführt werden.

3.4 Anwendungsfälle präzise Beschreiben

Af1.1 Alkoholisches Getränk erfassen

Akteure: Benutzer, Access-Datenbank

Auslöser: Benutzer will ein neues alkoholisches Getränk erfassen.

Ablauf:

1. Der Benutzer klickt auf Datei → Neu → Alkoholisches Getränk. Im nachfolgenden Dialog kann er die relevanten Daten eingeben. Hierzu gehören Name, Hersteller, Volumenprozent, Art des Getränks, Bar... Des Weiteren kann man noch Infos oder Zusammensetzung des Getränks speichern.
2. durch Drücken von "Speichern" werden die eingegebenen Daten überprüft und in die Datenbank geschrieben
3. Ggf wird Schritt 1 mit anderen Daten wiederholt
4. Durch Auslösen der Aktion "Abbrechen" wird der Dialog wieder in den Ausgangszustand versetzt.

Varianten 1.2 Benutzer will Softdrink erfassen
1.3 Bar ist nicht in der Liste vorhanden. Durch Klick auf "Neue Bar" wird der Barerfassungsdialog aufgerufen.

4 Kommunikation mit einer Datenbank

4.1 Möglichkeiten auf eine Datenbank zuzugreifen

4.1.1 Vorteile und Nachteile (C-API)

Vorteile

- Die schnellste Art, auf eine DB zuzugreifen, ist die C-API
- Wir können alle Funktionen der DB verwenden, welche die C-API zur Verfügung stellt.

Nachteile

- Unsere Applikation wird nur gerade mit diesem Datenbanktyp (und wahrscheinlich) auch nur mit dieser Version des Datenbanktyps) funktionieren, da wir mit herstellerabhängigen Funktionen entwickeln. D.h., wenn wir dieselbe SW mit MS SQL-Server oder Oracle laufen lassen möchten, müssen wir eine andere Bibliothek einbinden und den Aufruf sämtlicher Funktionen ersetzen, da die bestimm andere Bezeichnungen und Parameter haben.

- Man muss die Funktionen der MySQL-Server-Bibliothek erst kennen lernen.
- Flaschenhals, da gute Performance nur bedingt nutzbar.

4.1.2 Zugriff über Abstraktionsebenen

ODBC
DAO
ADO
ADO.NET

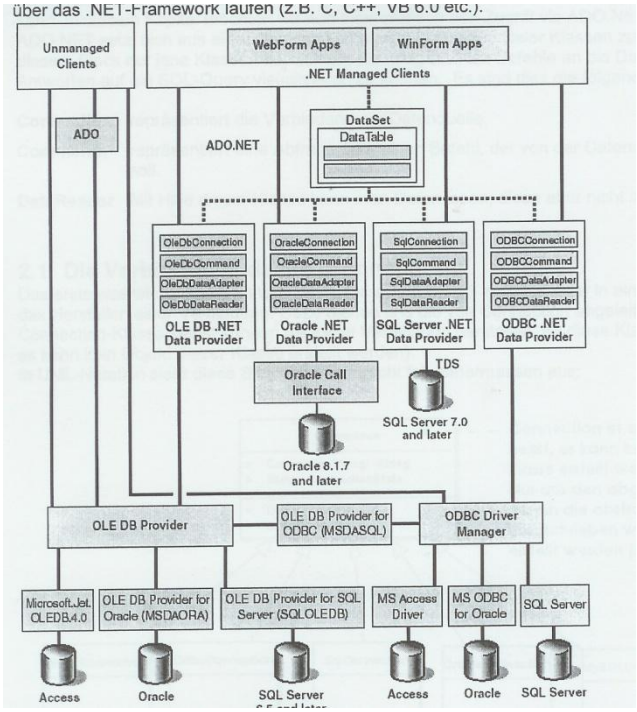
4.1.2.1 Vor und Nachteile hat die Verwendung von ODBC?

Nachteile

- Beschränkter Funktionsumfang
- Langsamer, da eine weitere Abstraktionsebene eingefügt wurde (Funktionen, die Funktionen aufrufen) und nicht die optimalen, serverspezifischen Vorteile und Funktionen verwendet werden können.

Vorteile

- Man kann Software schreiben, die mit allen Datenbanken funktioniert, bei denen ein ODBC-Treiber installiert ist.



4.2 Auf eine Datenbank zugreifen

4.2.1 Der Connection-String

Der Connection-String ist das zentrale Element des Verbindungsaufbaus. Er sieht je nach Zugriffsmethode (ODBC, OLEDB oder direkt) und Datenbanktyp (SQL-Server, Access) unterschiedlich aus. So gibt es z.B. das Element „Jet OLEDB:Database Passwort“ nur bei OLEDB-Zugriff auf Access-Datenbanken.

4.2.2 Commands

Objekt der entsprechenden Klasse erstellen
OleDbCommand cmd = new OleDbCommand();

mit CommandText kann man den auszuführenden Befehl mitteilen
cmd.CommandText = „DELETE FROM Personen WHERE Vorname = ‚Hans‘“;

dem Objekt müssen wir noch mitteilen, auf welche Datenbank sich der Befehl bezieht (=Verbindung)
cmd.Connection = cnn_Access;

absenden (wenn kein Rückgabewert (=SELECT))
cmd.ExecuteNonQuery();

4.2.3 Command & DataReader

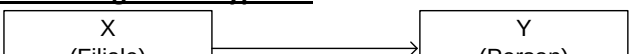
OleDbCommand cmd = new OleDbCommand();
cmd.CommandText = “SELECT * FROM Personen”;
cmd.Connection = cnn_Access;

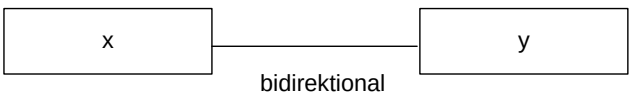
DataReader-Objekt
OleDbDataReader myReader = cmd.ExecuteReader();

Anhang → Codebeispiel

5 Beziehungen zwischen Klassen

Beziehungen des Typs 1:1

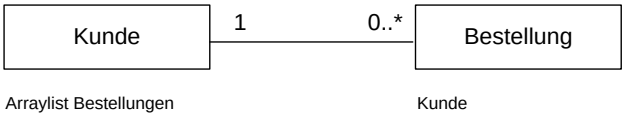




→ Code hinten im Anhang…

Gerichtete 1:n Beziehungen

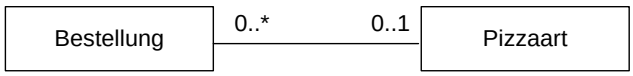
Bidirektionale n:1 Beziehungen



Beziehungen des Typs n:m

n:m-Beziehungen sind immer bidirektional.

➔ Hashtable: siehe Block 5 Seite 7



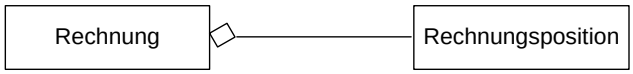
Assoziation, Aggregation, Komposition

Assoziation

Parent- und Child-Objekte können allein existieren.

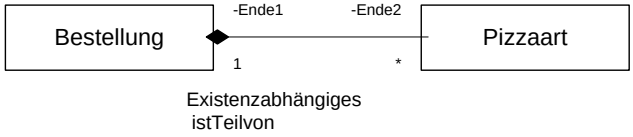
Aggregation

istTeilvon-Beziehung



Komposition

existenzabhängige istTeilvon-Beziehung



6 Theorie-Fragen

In der objektorientierten Programmierung schreibt der Softwareentwickler während der Programmcodierung ***Klassen***. Es existiert kein einziges ***Objekt***.
Beim Programmstart beginnt die Programmausführung mit der **statischen Main-Methode**, die sich irgendwo innerhalb irgendeiner **Klasse** befinden muss.
Eine Klasse besteht aus **Eigenschaften, Methoden und Konstruktoren**. Erstere können **Variablen** oder **Referenzen** auf **Objekte** sein, die im **Konstruktor** erzeugt werden.
Der Konstruktor wird bei der **Instanziierung** einer Klasse aufgerufen. Man unterscheidet zwischen **Standard** und **erweiterten Konstruktoren**.
Die in einem C-Programm geschriebene Funktion mit der Bezeichnung SINUS() bekommt als Parameter den Winkel in Radiant und liefert den Sinuswert als Rückgabewert. Wenn Sie diese Funktion in einer objektorientierten Sprache wie C# schreiben möchten, würden Sie statt einer Funktion eine **Methode** schreiben. Diese müsste sinnvollerweise **statisch** sein, damit sie auch ohne **Instanziierung** verwendet werden kann.

Vorteile, wenn man Eigenschaften vor direktem Zugriff von aussen schützt

- Programmcodeänderungen müssen nur an einer Stelle erfolgen
- Nur Lese- oder nur Schreiberlaubnis möglich

Unterschiede zwischen lokalen Variablen und Membervariablen

- Membervariablen sind Variablen der Klasse und existieren nach der Instanziierung nur solange wie das Objekt
- Membervariablen haben eine Sichtbarkeit (private / public), lokale Variablen nicht

Unterschiede zwischen Instanz- und Klassenelementen (=statisch)

- Instanzelemente können mehrmals existieren, statische nur einmal
- Statische Methoden dürfen nur auf statische Eigenschaften zugreifen
- Statische Elemente kann man über den Klassennamen ansprechen
- Statische Elemente werden ganz am Anfang vor der Programmausführung erstellt und bleiben während der ganzen Ausführung im Speicher

Konstruktoren

- Pro Klasse kann gleichzeitig mehr als ein Objekt existieren
- Wenn kein Konstruktor existiert, erzeugt uns das System einen Standardkonstruktor der alle nicht initialisierten Eigenschaften auf 0 resp. NULL setzt
- Eine Klasse kann mehrere Konstruktoren besitzen. Einzige Bedingung dafür ist, dass sich die Signaturen unterscheiden
- Nicht statische Methoden dürfen auf statische Eigenschaften zugreifen.
- Statische Methoden dürfen nur auf statische Methoden zugreifen
- Ein Objekt basiert immer auf einer Klasse
- Als Softwareentwickler können wir genau bestimmen, welcher Konstruktor aufgerufen werden soll.

7 Glossar

Block „Namenskonventionen“
Block „Dynamische Arrays“

Singleton

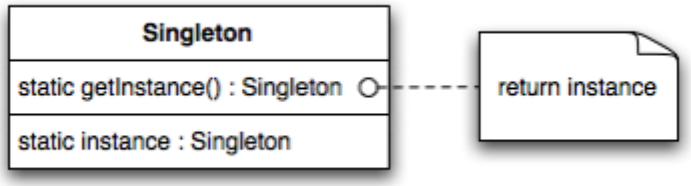
Erzeuge genau eine Instanz einer Klasse und stelle einen zentralen Zugriffspunkt auf diese Instanz bereit.

Zweck

Bisweilen muss verhindert werden, dass von einer Klasse mehrere Instanzen erzeugt werden können - genau dies leistet das Singleton-Muster.

Ein Beispiel für solch einen Fall ist eine Klasse, die die Verbindung zu einer Datenbank verwaltet und für den Rest des Programms Funktionen zur Verfügung stellt, um auf die Datenbank zuzugreifen. Angenommen die Datenbank bietet selbst keine Mechanismen, um beispielsweise atomare Operationen zu gewährleisten; dies bleibt dann der Datenbankklasse vorbehalten. Gäbe es nun mehrere Instanzen dieser Datenbankklasse, könnten wiederum verschiedene Teile des Programms gleichzeitig Änderungen an der Datenbank vornehmen, indem sie sich unterschiedlicher Instanzen bedienen; kann sichergestellt werden, dass es nur genau ein Exemplar der Datenbankklasse gibt, tritt dieses Problem nicht auf.

UML



Entscheidungshilfen

Ein Singleton sollte dann eingesetzt werden, wenn sichergestellt sein muss, dass nicht mehr als ein Objekt einer Klasse erzeugt werden kann.

Funktionsweise

Anstatt selbst eine neue Instanz durch Aufruf des Konstruktors zu erzeugen, müssen sich Benutzer der Singleton-Klasse eine Referenz auf eine Instanz über die statische getInstance()-Methode besorgen - die Singleton-Klasse ist also selbst für die Verwaltung ihrer einzigen Instanz zuständig. Die getInstance()-Methode kann nun sicherstellen, dass bei jedem Aufruf eine Referenz auf dieselbe und einzige Instanz - gehalten in einer (versteckten) Klassenvariable - zurückgegeben wird. Üblicherweise wird diese eine Instanz von getInstance() beim ersten Aufruf erzeugt.
In nebenläufigen Programmen muss der Programmierer beim Schreiben der getInstance()-Methode besondere Vorsicht walten lassen (siehe Code-Beispiele).

Code

```
using System;
using Org.Wikibooks.De.Csharp.Pattern;

namespace Org.Wikibooks.De.Csharp.Pattern.Creational
{
 class Singleton
 {
  // Eine (versteckte) Klassenvariable vom Typ der eigene Klasse
  private static Singleton m_Instance;

  // Konstuktor
  // Dieser Konstruktor kann von außen nicht erreicht werden.
  private Singleton() {}

  // Instanziierung
  public static Singleton getInstance()
  {
   // lazy creation
   if (m_Instance == null)
   {
    m_Instance = new Singleton();
   }
   return m_Instance;
  }
 }
}
```

8 Anhang → Codebeispiele

Mitarbeiter (mit vererbten Klassen)

(Klassenauszug)

Mitarbeiter.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Teko.Programmieren2.Jumlitest
{
        public abstract class CMitarbeiter
        {
         //Membervariablen
         private int m_PersNr;
         private string m_Name;
         private double m_Monatslohn;

         //Konstruktoren
         public CMitarbeiter()
         {
             setPersNr(1);
             setName("Neue Person");
             setMonatslohn(1000.00);
         }

         public CMitarbeiter(int persnr, string name, double lohn)
         {
             setPersNr(persnr);
             setName(name);
             setMonatslohn(lohn);
         }


         //Properties
         public void setPersNr(int nummer)
         {
             m_PersNr = nummer;
         }

         public int getPersnr()
         {
             return m_PersNr;
         }

         public void setName(string name)
         {
             m_Name = name;
         }

         public string getName()
         {
             return m_Name;
         }

         public void setMonatslohn(double lohn)
         {
             m_Monatslohn = lohn;
         }

         public double getMonatslohn()
         {
             return m_Monatslohn;
         }

         //Methoden
         public virtual double getJahresLohn()
         {
             return m_Monatslohn * 12;
         }

         //public virtual void Datenausgabe()
         //{
         //    System.Console.Out.WriteLine("------------------------------");
         //    System.Console.Out.WriteLine("PersNr=" + getPersnr());
         //    System.Console.Out.WriteLine("Name=" + getName());
         //    System.Console.Out.WriteLine("JahresLohn=" + getJahresLohn());
         //    System.Console.Out.WriteLine("------------------------------");
         //}
```

```
        public abstract void Datenausgabe();
       }
}
```

CManager.cs

```
using System.Collections.Generic;
using System.Text;

namespace Teko.Programmieren2.Jumlitest
{
    class CManager:CMitarbeiter
    {
        //Eigenschaften
              private double m_Bonus;
              private bool m_ZielErreicht = false; //gilt sofern in einem Konstruktor
                                                                                    //nicht anders
definiert wurde
              //Konstruktoren
              public CManager(int persnr, string name, double lohn, double bonus,bool
zielerreicht):base(persnr,name,lohn)
               {
            //setPersNr(persnr);
            //setName(name);
            //setMonatslohn(lohn);
                        setBonus(bonus);
            setZielErreicht(zielerreicht);
               }
               //Propreties
               public double getBonus() {
                        return m_Bonus;
               }
        public void setBonus(double value)
        {
            m_Bonus = value;
        }
               public bool getZielErreicht() {
                       return m_ZielErreicht;
               }
               public void setZielErreicht(bool value) {
            m_ZielErreicht = value;

               }

        public override double getJahresLohn()
        {
            if (getZielErreicht())
                return (getMonatslohn()*13 + getBonus());
            else
                return (getMonatslohn()*13);

        }

        public override void Datenausgabe()
        {
            System.Console.ForegroundColor = ConsoleColor.Yellow;
            System.Console.Out.WriteLine("******************************");
            System.Console.Out.WriteLine("Typ:\tManager");
            System.Console.Out.WriteLine("PersNr:\t" + getPersnr());
            System.Console.Out.WriteLine("Name:\t" + getName());
            System.Console.Out.WriteLine("Bonus erhalten:\t"+getZielErreicht());
            System.Console.Out.WriteLine("Bonus:\t" + getBonus());
            System.Console.Out.WriteLine("JahresLohn:\t" + getJahresLohn());
            System.Console.Out.WriteLine("******************************");
        }
    }
}
```

CSekretaerin.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Teko.Programmieren2.Jumlitest;


public class CSekretaerin:CMitarbeiter
{
```

```
    //Membervariablen
       private string m_Masse;
       private string m_Sprachen;
    private string m_Bewertung;


       //Konstruktoren
       public CSekretaerin()
       {
        //
       }

    public CSekretaerin(int persnr, string name, double lohn, string masse, string sprachen,string
bewertung): base(persnr, name, lohn)
    {
            //setPersNr(persnr);
            //setName(name);
            //setMonatslohn(lohn);
            setM_Masse(masse);
            setM_Sprachen(sprachen);
            setBewertung(bewertung);
    }

       public override void Datenausgabe()
       {
            System.Console.ForegroundColor = ConsoleColor.Magenta;
            System.Console.Out.WriteLine("*****************************");
            System.Console.Out.WriteLine("Typ:\tSekretärin");
            System.Console.Out.WriteLine("PersNr:\t" + getPersnr());
            System.Console.Out.WriteLine("Name:\t" + getName());
            System.Console.Out.WriteLine("Sprachen:\t"+getM_Sprachen());
            System.Console.Out.WriteLine("JahresLohn:\t" + getJahresLohn());
            System.Console.Out.WriteLine("Masse:\t"+getM_Masse());
            System.Console.Out.WriteLine("Bemerkung:\t" + getBewertung());
            System.Console.Out.WriteLine("*****************************");
       }

    //public override void GetJahresLohn()
    //{
    //}

       public string getM_Masse()
       {
        return m_Masse;
       }

       public string getM_Sprachen()
       {
        return m_Sprachen;
       }

    public string getBewertung()
    {
        return m_Bewertung;
    }

       public void setM_Masse( string value )
       {
        m_Masse = value;
       }

       public void setM_Sprachen( string value )
       {
        m_Sprachen = value;
       }

    public void setBewertung(string Bewertung)
    {
        m_Bewertung = Bewertung;
    }
} (L)
```

Verbindung mit Access herstellen und trennen

```
private void btnConnectAccess_Click(object sender, EventArgs e)
        {
            if (cnn.State == ConnectionState.Closed)
            {
                cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0"; //Datenbanktyp
                cnn.ConnectionString += ";Data Source="+txtAccessPfad.Text;       //Datei
```

```
                cnn.ConnectionString += ";User Id=" + txtAccessUser.Text;                    //Username
                cnn.ConnectionString += ";Password=" + txtAccessPwd.Text;                         //User-
Passwort
                cnn.ConnectionString += ";Jet OLEDB:Database Password=";    //DB-Passwort
                try
                {
                    cnn.Open();
                }
                catch (OleDbException er)
                {
                    MessageBox.Show(er.Message,"Error",MessageBoxButtons.OK,MessageBoxIcon.Error);
                    return;
                }
                lblAccessStatus.ForeColor = Color.Green;
                lblAccessStatus.Text = "Verbindung OK";
            }
        }

        private void btnDisconnectAccess_Click(object sender, EventArgs e)
        {
            cnn.Close();
            this.lblAccessStatus.ForeColor = Color.Magenta;
            this.lblAccessStatus.Text = "Keine Verbindung";
        }
```

Datenbankverbindung

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

//……

OleDbConnection m_cnnPersVerw = null;


            private bool TryToConnectBD()
            {
                    try
                    {
                            //Verbindung zur Datenbank herstellen
                            if (m_cnnPersVerw == null)
                                    m_cnnPersVerw = new OleDbConnection();
                            if (m_cnnPersVerw.State != ConnectionState.Open)
                            {
                                    m_cnnPersVerw.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0";
                                    m_cnnPersVerw.ConnectionString += ";User ID=Admin";
                                    m_cnnPersVerw.ConnectionString += ";PASSWORD=";
                                    openFileDialog1.InitialDirectory = Application.StartupPath;
                                    if (DialogResult.OK != openFileDialog1.ShowDialog())
                                            return false;
                                    m_cnnPersVerw.ConnectionString += ";Data Source= " +
openFileDialog1.FileName;
                                    m_cnnPersVerw.Open();
                            }
                            return true;
                    }
                    catch//Allg.  Fehlerbehandlung
                    {
                            return false;
                    }

            }

            private void btnAusfuehren_Click(object sender, System.EventArgs e)
            {
                    if(false == TryToConnectBD())
                    {
                            MessageBox.Show("Keine Datenbank --> Der Vorgang wird abgebrochen");
                            return;
                    }
                    try
                    {
                            // Select Kommando absenden und das Resultat im ListView darstellen
                            //-------------------------------------------------------------------------
-----------
```

```
                        // HIER CODE ERGAENZEN

                //Variablen Abfrage
                string PNrVon = "1";
                string PNrBis = "1000";
                string a_Verknüpfung = "OR";
                string a_Name = "A%";

                PNrVon = txtPNrVon.Text;
                PNrBis = txtPNrBis.Text;
                a_Verknüpfung = comboBox1.Text;
                a_Name = txtNamenFilter.Text;


                //VariablenAusgabe
                int PNr = 1;
                string Anrede = "Herr";
                string Namen = "Name";
                string Vorname = "Vorname";

                //*****************************************************

                OleDbCommand cmd = new OleDbCommand();
                cmd.CommandText = "SELECT * FROM Personen inner join Anreden on Personen.fk_Anrede =
Anreden.AnredeNr ";
                cmd.CommandText += "WHERE PNr >" + PNrVon + " and PNr <" + PNrBis;
                cmd.CommandText += " " + a_Verknüpfung + " Namen LIKE '" + a_Name + "%'";
                cmd.CommandText += " Order by Namen, Vornamen;";
                cmd.Connection = m_cnnPersVerw;

                OleDbDataReader myReader = cmd.ExecuteReader();

                listView1.Items.Clear();
                while (myReader.Read() == true)
                {
                    ListViewItem item;
                    item = new ListViewItem(Convert.ToString(myReader.GetValue(0))); //PNr
                    item.SubItems.Add(Convert.ToString(myReader.GetString(5))); //Anrede
                    item.SubItems.Add(myReader.GetString(1)); //Name
                    item.SubItems.Add(myReader.GetString(2)); //Vorname
                    item.Checked = false;
                    listView1.Items.Add(item);

                }

                //Reader schliessen
                myReader.Close();


                        //-------------------------------------------------------------------------
-----------
                }
                catch(InvalidOperationException e2) //Fehlerbehandlung falls Verbindung nicht steht
                {
                        MessageBox.Show(e2.Message);
                }
                catch(OleDbException e2)      //Allg OleDb Fehlerbehandlung
                {
                        string errorMessages = "";
                        for (int i=0; i < e2.Errors.Count; i++)
                                errorMessages += "Message: " + e2.Errors[i].Message + "\n";
                        MessageBox.Show(errorMessages);
                }
                catch(Exception e2)//Allg.  Fehlerbehandlung
                {
                        MessageBox.Show(e2.Message);
                }
        }

        private void OnLoeschen(object sender, System.EventArgs e)
        {
                if(false == TryToConnectBD())
                {
                        MessageBox.Show("Keine Datenbank --> Der Vorgang wird abgebrochen");
                        return;
                }
                try
                {
                        // Delete Kommando für jeden selektierten Datensatz absenden und Abfrage neu
darstellen
```

```
                        //-------------------------------------------------------------------------
-----------
                        // HIER CODE ERGAENZEN
                for (int i = 0; i < listView1.CheckedItems.Count; i++)
                {
                    OleDbCommand cmd = new OleDbCommand();
                    cmd.Connection = m_cnnPersVerw;
                    cmd.CommandText = "DELETE FROM Personen Where PNr ="
+listView1.CheckedItems[i].SubItems[0].Text;
                    cmd.ExecuteNonQuery();
                }
                        //Datensätze erneut anzeigen
                btnAusfuehren_Click(null, null);
                        //-------------------------------------------------------------------------
-----------
                    }
                    catch(OleDbException e2)      //Allg OleDb Fehlerbehandlung
                    {
                            string errorMessages = "";
                            for (int i=0; i < e2.Errors.Count; i++)
                                    errorMessages += "Message: " + e2.Errors[i].Message + "\n";
                            MessageBox.Show(errorMessages);
                    }
                    catch(Exception e2)//Allg.  Fehlerbehandlung
                    {
                            MessageBox.Show(e2.Message);
                    }
            }
```