**Modul 320**

***Block 03***

***Schnittstellen / Interfaces***

# Eine Klasse implementiert eine Schnittstelle

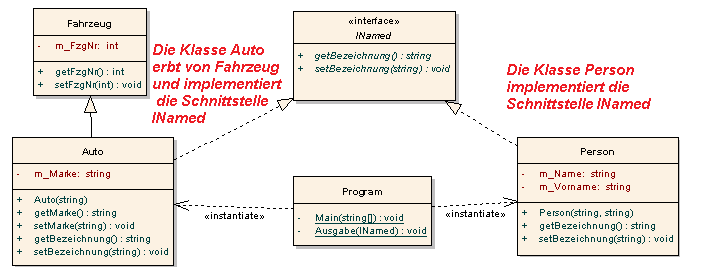
Wir behandeln das Thema Interface (Schnittstellen) an dieser Stelle nur sehr grundsätzlich. Sie müssen lediglich Wissen, was eine Schnittstelle ist und wie man eine Schnittstelle, welche auf einem UML-Klassendiagramm vorgegeben ist, implementiert. Der grosse Sinn und Zweck dieser Schnittstellen werden wir später betrachten.

Schnittstellen (engl. Interfaces) definieren, ähnlich wie Basisklassen, eine Grundfunktionalität. In der Regel handelt es sich dabei um eine Gruppe von sachlich zusammengehörenden Methoden.

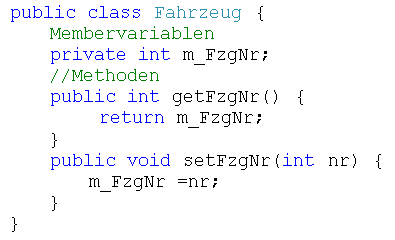
Schnittstellen dürfen im Gegensatz zu Basisklassen der Vererbung, keine Membervariablen enthalten. Schnittstellen dürfen auch keine ausprogrammierten Methoden enthalten, sondern nur eine Art Prototypen. Man nennt diese Prototypen abstrakte Methoden, da sie noch nicht fertiggestellt wurden.

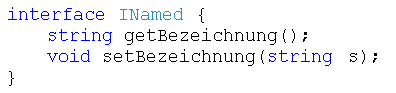
Eine abstrakte Methode sagt lediglich, wie eine Methode heissen soll, was sie zurückgibt (Funktionsrückgabewert) und welche Parameter sie benötigt. Fertig!

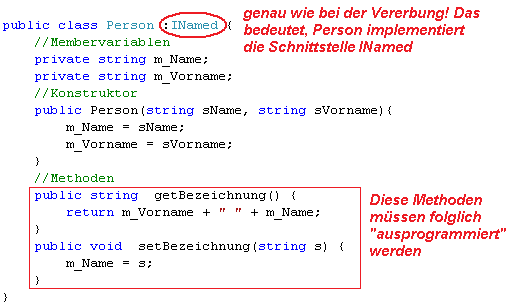
Schauen wir uns ein Beispiel an:

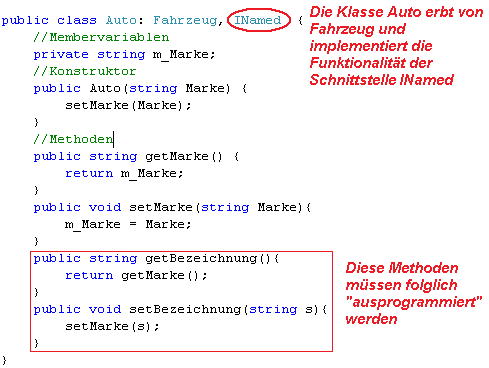


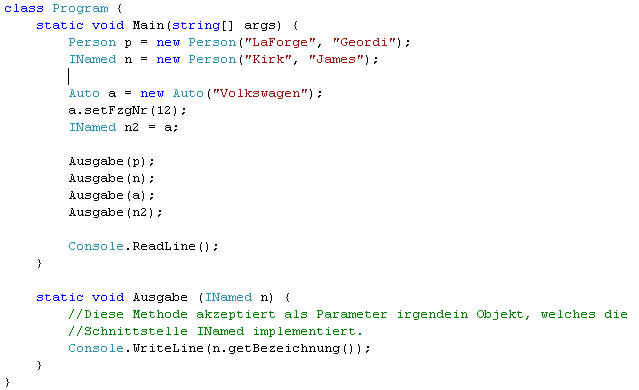
Per Programmcode sieht das folgendermassen aus:

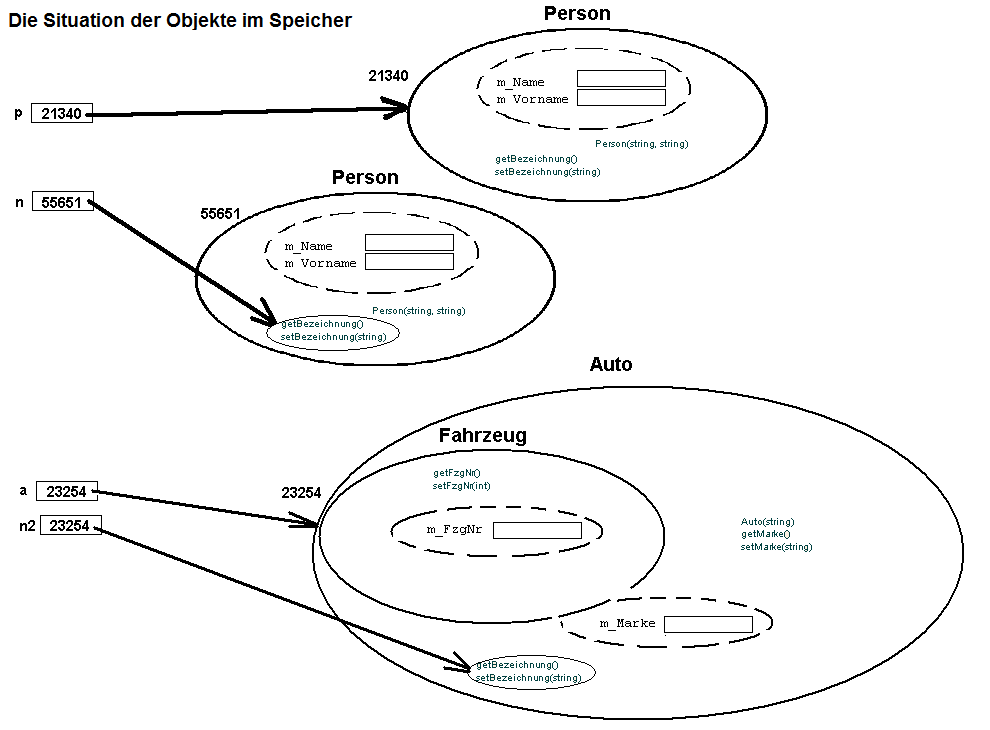












Erinnern sie sich noch an das im Modul 226B Block 01 erstellte Projekt mit den Mitarbeitern, Politiker und Parteimitgliedern? Wir versuchten mit Hilfe der Vererbung Redundanz zu beseitigen. Allerdings hatten wir eine Unschönheit im Code: Die Funktion Lohnerhöhung gab es zweimal im Code. Erst nach der Verwendung einer Schnittstelle (ILohn) konnten wir das Problem lösen.



# Aufgabe 01

**Lernziel**: Aus Programmcode ein Klassendiagramm erstellen

**Zeit**: 20’

**Aufgabe:** a) Betrachten Sie den untenstehenden Programmcode und überlegen Sie sich, was das Programm ausgibt.

b) Zeichnen Sie für den gegebenen Programmcode das UML-Klassendiagramm inkl. den Beziehungen auf.

class **Program** {

static void Main(string[] args) {

Zylinder z = new Zylinder();

z.setHoehe(12);

z.setDurchmesser(5);

Console.WriteLine("Volumen = " + z.getVolumen());

IKoerper ik = new Zylinder(5,12);

Console.WriteLine("Volumen = " + ik.getVolumen());

Console.ReadLine();

}

}

public class **Mathematik** {

class **Zylinder** : IKoerper {

private Kreis m\_Grundflaeche;

private int m\_Hoehe;

public Zylinder() {

m\_Grundflaeche = new Kreis(0);

setHoehe(0);

}

public Zylinder(int d, int h) {

m\_Grundflaeche = new Kreis(d);

setHoehe(h);

}

public int getHoehe() {

return m\_Hoehe;

}

public void setHoehe(int value) {

m\_Hoehe = value;

}

public int getDurchmesser() {

return m\_Grundflaeche.getDurchmesser();

}

public void setDurchmesser(int value) {

m\_Grundflaeche = new Kreis(value);

}

public double getVolumen() {

return m\_Grundflaeche.GetFlaeche() \* getHoehe();

}

public double getStandflaeche() {

return m\_Grundflaeche.GetFlaeche();

}

}

public const double PI = 3.1415927;

public static int Quadrat (int Basis, int Exponent){

int res = 1;

for (int i = 1; i <= Exponent; i++)

res = res \* Basis;

return res;

}

public static double Addition(double z1, double z2) {

return z1 + z2;

}

}

public interface **IKoerper** {

double getVolumen();

double getStandflaeche();

}

public class **Kreis** {

private int m\_Durchmesser;

public Kreis() {

setDurchmesser(0);

}

public Kreis(int d) {

setDurchmesser(d);

}

private void setDurchmesser(int value) {

m\_Durchmesser = value;

}

public int getDurchmesser() {

return m\_Durchmesser;

}

public double GetFlaeche() {

return Mathematik.Quadrat(m\_Durchmesser, 2) \* Mathematik.PI / 4;

}

}

**Aufgabe 02a (**

**Lernziel**: Aus Programmcode ein Klassendiagramm erstellen.

**Zeit**: 60’

**Aufgabe**: a) Betrachten Sie den untenstehenden Programmcode und überlegen Sie sich, welche Klassen es gibt. Vergleichen Sie Ihre Antwort mit der Übersicht der Klassen auf der Folgeseite.

b) Überlegen Sie, welche Beziehungen zwischen den Klassen existieren.Zeichnen Sie die Beziehungen auf der Folgeseite in das Klassendiagramm ein.

c) Öffnen Sie das Vorlageprojekt und versuchen Sie das Klassendiagramm in Programmcode umzusetzen, so dass es die untenstehende Ausgabe liefert.

namespace Aufgabe01 {

public enum eLayer : byte {SingleLayer=1, DualLayer=2 };

public enum eInhalt : byte { Musik = 1, Film = 2, Daten =3 };

**class Program**  **{**

**static void Main(string[] args) {**

SpeicherMedium r1 = new CompactDisc(eInhalt.Daten, "01.11.2018");

CompactDisc r2 = new CompactDisc(eInhalt.Musik, "The Best of Mittelwelle 531");

SpeicherMedium r3 = new DigitalVersatileDisc(eInhalt.Film, "Ein Quantum Trost");

DigitalVersatileDisc r4 = new DigitalVersatileDisc();

r4.setInhalt(eInhalt.Musik);

r4.setTitel("mp3-Sammlung 05");

SpeicherMedium r5 = new BluRay(eInhalt.Film, "KillBill", eLayer.SingleLayer);

BluRay r6 = new BluRay();

r6.setInhalt(eInhalt.Film);

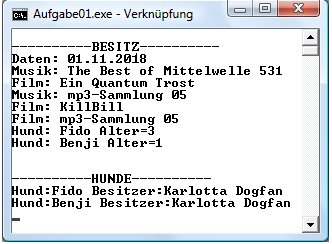
r6.setLayer(eLayer.DualLayer);

r6.setTitel("mp3-Sammlung 05");

Hund r7 = new Hund("Fido", 3);

Hund r8 = new Hund("Benji");

r8.setAlter(1);

Person person = new Person("Dogfan", "Karlotta");

person.AddBesitz(r1);

person.AddBesitz(r2);

person.AddBesitz(r3);

person.AddBesitz(r4);

person.AddBesitz(r5);

person.AddBesitz(r6);

person.AddBesitz(r7);

person.AddBesitz(r8);

Console.WriteLine("\n----------BESITZ----------");

foreach (IBezeichnung ib in person.getBesitz())

Console.WriteLine(ib.getBezeichnung());

Console.WriteLine("\n\n----------HUNDE----------");

Console.WriteLine("Hund:" + r7.getName() + " Besitzer:" + r7.getBesitzer().getFullName());

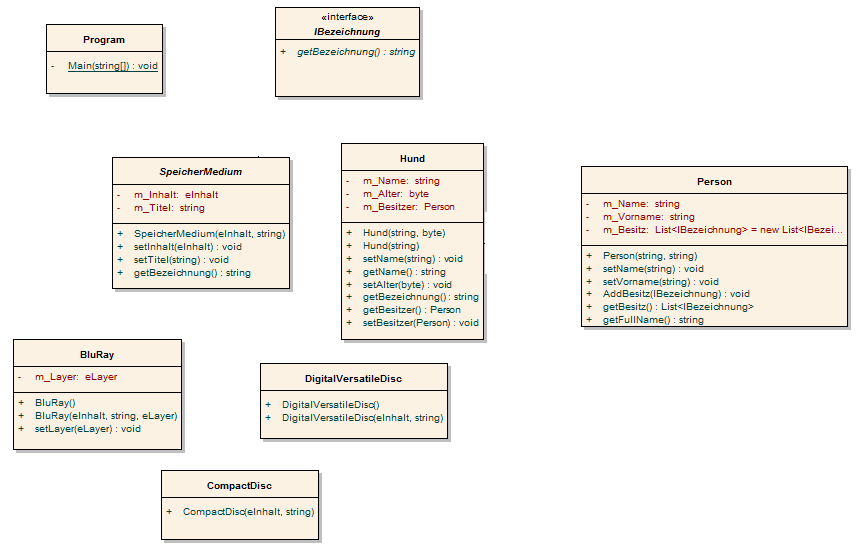
Console.WriteLine("Hund:" + r8.getName() + " Besitzer:" + r8.getBesitzer().getFullName());

Console.ReadLine();

**}**

**}**

}



**Aufgabe 02b (**

**Lernziel**: Das Open-Closed Prinzip verstehen und anwenden können.

**Zeit**: 50’

**Aufgabe**: Bei der Aufgabe 02a haben sie eine Lösung erarbeitet die funktioniert. Aber ist die Lösung auch optimal?

Nein. Wir haben ein Problem. Wenn wir den Sachverhalt um neue Klassen ergänzen, kann es sein, dass wir bestehenden Programmcode anpassen müssen.

a) Finden sie heraus, wo der Code nicht optimal und erklären sie die nötigen Anpassungen am bestehendem Code anhand eines konkreten Beispiels.

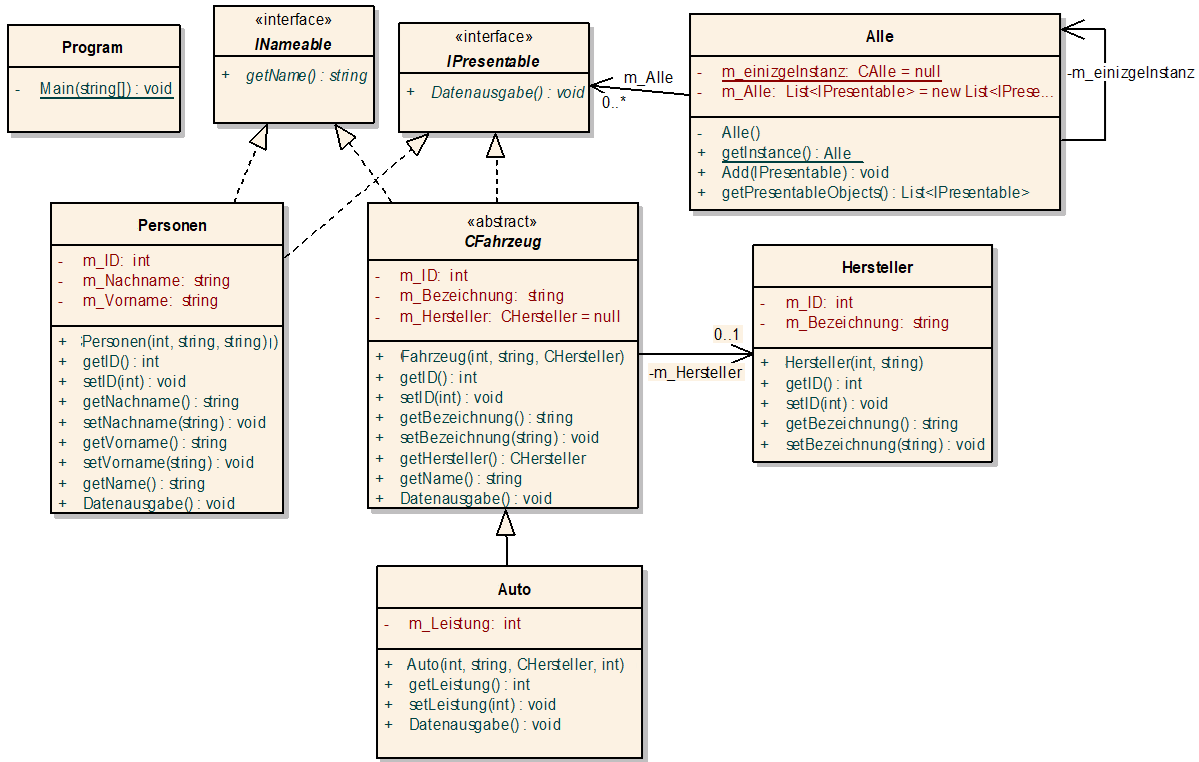
b) Versuchen Sie die Situation zu verbessern

**Aufgabe 03 (**

**Lernziel**: Aus Programmcode ein Klassendiagramm erstellen.

**Zeit**: 15’

**Aufgabe**: Betrachten Sie das UML-Klassendiagramm und beantworten Sie die zugehörigen Fragen



a) Welche Aktionen sind erlaubt? (þ = erlaubt, o = nicht erlaubt)

o IPresentable ref1 = new Fahrzeug();

o IPresentable ref2 = new Hersteller(5, “Volkswagen”);

o IPresentable ref3 = new Personen(11, “Huber”, “Hans”);

o Personen ref4 = new IPresentable();

o Personen ref5 = new Personen(1, “Obrist”, “Lienhard”);

o Fahrzeug ref6 = new Fahrzeug(2, “Zastava”);

o Fahrzeug ref7 = new Auto(2, “Corolla”, new Hersteller(7, “Toyota”));

o Hersteller ref8 = new Hersteller (44, “Mazda”);

o Alle ref9 = new Alle();

o Alle.getInstance().Add(new Personen(22, “Hugentobler”, “Karl”));

o Alle.getInstance().Add(new Hersteller (33, “Fiat”));

b) Welche Aussagen sind korrekt? (þ = erlaubt, o = nicht erlaubt)

o Innerhalb der Methode getLeistung() kann man ohne Instanziierung auf das Herstellerobjekt zugreifen;

o Innerhalb der Methode getLeistung() kann man direkt auf die Membervariable m\_ID zugreifen;

o Ein Hersteller-Objekt speichert als Membervariable eine Referenz auf ein Objekt des Typs CFahrzeug

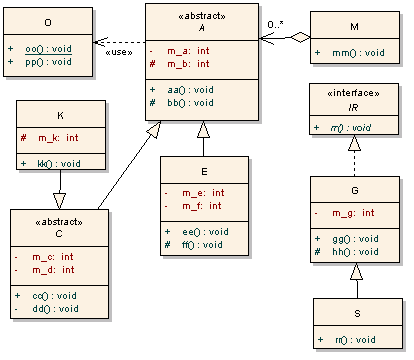
o Es kann nur ein einziges Objekt der Klasse CAlle geben

**Aufgabe 04 (**

**Lernziel**: Aus Programmcode ein Klassendiagramm erstellen.

**Zeit**: 15’

**Aufgabe**: Betrachten Sie das UML-Klassendiagramm und beantworten Sie die zugehörigen Fragen



**Fragen zu den Beziehungen:**

¨ Die Klasse C und K erben von der Klasse A

¨ Die Klasse E ist die Basisklasse der Klasse K

¨ Die Klassen A und M stehen in einer Kompositionsbeziehung

¨ Die Klassen A und E stehen in einer Generalisierungsbeziehung

¨ Die Klasse G erbt von der Klasse Object (bei C# und java)

¨ Die Klasse G erbt von der Klasse IR

¨ Die Beziehung zwischen den Klassen A und O ist schwächer als die Beziehung zwischen den Klassen A und M.

¨ Die Klasse M erbt von der Klasse A

¨ Die Klasse A hat eine Membervariable, die auf ein Objekt der Klasse M verweisen kann

¨ Die Klasse M hat eine Membervariable, die auf ein Objekt der Klasse A verweisen kann

¨ Die Klasse A hat eine Membervariable, die auf ein Objekt der Klasse O verweisen kann

¨ Die Klassen S erbt von der Klasse IR

**Fragen zur Aufrufbarkeit:**

¨ Innerhalb der Methode ff() kann man ohne Instanziierung die Methode bb() aufrufen

¨ Innerhalb der Methode cc() kann man direkt auf die Membervariable m\_a zugreifen

¨ Innerhalb der Methode kk() kann man ohne Instanziierung die Methode dd() aufrufen

¨ Innerhalb der Methode bb() kann man ohne Instanziierung die Methode mm() aufrufen

¨ Innerhalb der Methode ee() kann man ohne Instanziierung die Methode oo() aufrufen

¨ Innerhalb der Methode ff() kann man direkt auf die Membervariable m\_b zugreifen

¨ Innerhalb der Methode gg() kann man durch Instanziierung die Methode ff() aufrufen

¨ Innerhalb der Methode rr() kann man direkt auf die Membervariable m\_g zugreifen

**Fragen zur Polymorphie:**

¨ C myObj = new A(): ¨ K myObj = new A();

¨ E myObj = new C(); ¨ C myObj = new K();

¨ A myObj = new K(); ¨ Object myObj = new C();

¨ C myObj = new E(); ¨ Object myObj = new G();

¨ IR myObj = new G(); ¨ A myObj = new M();

¨ IR myObj = new S(); ¨ G myObj = new IR();