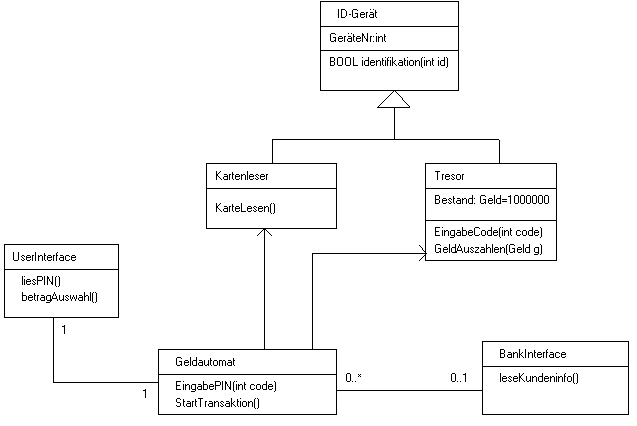
**Modul 320**

***Block 02***

***- Vererbung und Polymorphie -***



**Inhaltsverzeichnis:**

[1 Die Vererbungsbeziehung (Generalisierung) 2](#_Toc121759729)

[2 Übungen 6](#_Toc121759730)

# Die Vererbungsbeziehung (Generalisierung)

Vererbung bedeutet, dass alle Elemente wie Membervariablen und Methoden 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 eigene Elemente (Membervariablen & Methoden) erweitern oder bestehende Methoden überschreiben.

Konkret heisst das, dass die abgeleitete Klasse, mindestens alles das kann, was die Basis-Klasse kann. Mit der Vererbungshierarchie findet also eine Spezialisierung statt.

Vererbung können wir immer dann verwenden, wenn wir “ist-ein“ sagen können. Bei dem untenstehenden Beispiel können wir z.B. sagen: Jeder Manager ist ein Mitarbeiter, aber nicht jeder Mitarbeiter ist ein Manager. Das ist eine Spezialisierung und somit typisch für die Vererbung. Das heisst in unserem Fall, dass Manager von Mitarbeiter erbt.

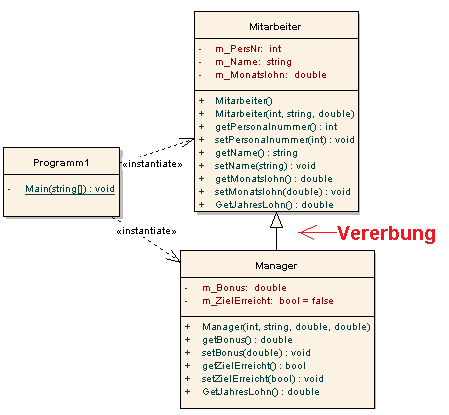
Man spricht auch von Basisklasse oder bei Java von Super-Klasse.

Mitarbeiter ist die Basisklasse (oder Superklasse) von Manager

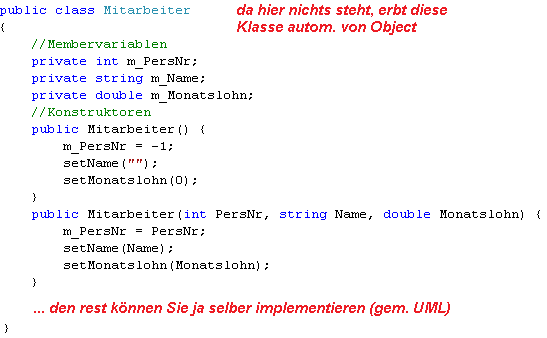
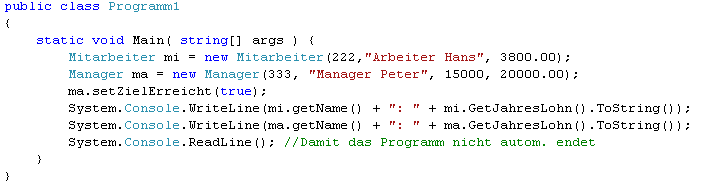
Manager ist die von Mitarbeiter abgeleitete Klasse.

Eine Klasse kann nur genau von einer anderen Klasse erben. Wenn wir keine Aussagen machen bezüglich der Vererbung, dann erbt eine Klasse automatisch von der obersten aller Klassen und die heisst Object.

Betrachten Sie das Klassendiagramm und den Sourcecode um das wesentliche der Vererbung zu verstehen.



Die Ausgabe:

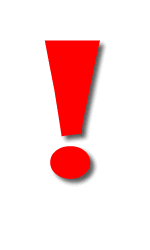


**Aufgabe A-01**:

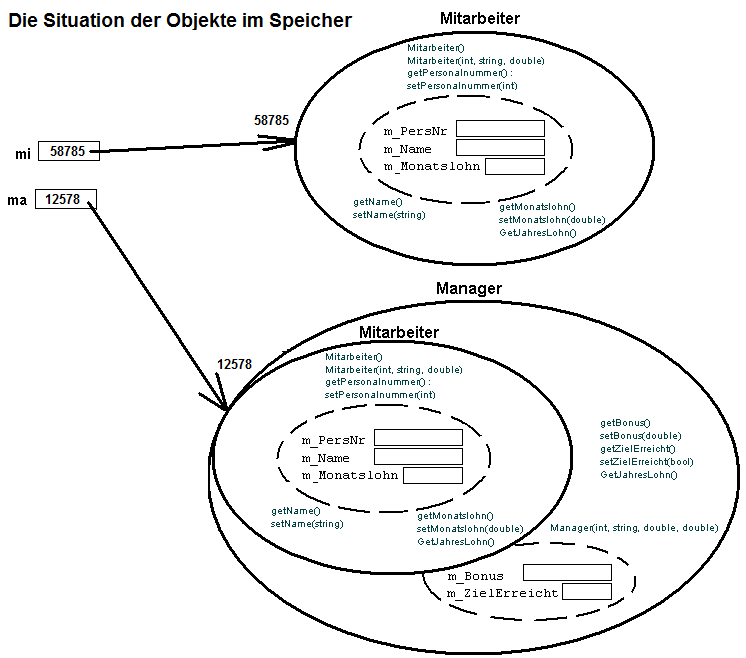
Form: Einzelarbeit ohne Kommunikation.

Zeit: 30 Minuten

Aufgabe: Setzen Sie das hier beschriebene Beispiel in Programmcode um und testen Sie es intensiv mit Haltepunkten. Notieren Sie sich Programmcode-Passagen die sie nicht verstehen.



Überlegen Sie danach, was wohl die Inhalte der Membervariablen sind. Notieren Sie Ihre Annahmen in der nachfolgenden Skizze und kontrollieren Sie diese durch Haltpunkte und Variablenüberwachung. Versuchen Sie auch mit Hilfe der Haltpunkte herauszufinden, ob die Konstruktoren beider Klassen aufgerufen werden und in welcher Sequenz dies geschieht (welcher wird zuerst ausgeführt, welcher zuletzt).



**Aufgabe A-02**:

Form: Einzelarbeit

Zeit: 10 Minuten

Aufgabe: Setzen Sie die Membervariable m\_Name

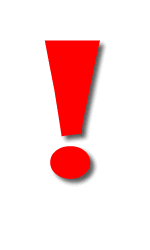
auf die Sichtbarkeit protected. Was sind die Auswirkungen bezüglich des Zugriffs?

**Aufgabe A-03**:

Form: Partnerarbeit

Zeit: 10 Minuten

Aufgabe: Wir machen einen kleinen Exkurs zum Thema Polymorphie: Überlegen Sie sich welche der sechs Instanziierungen der Kompiler akzeptiert. Testen Sie Ihre Annahmen mit der Entwicklungsumgebung.



01: Manager ref1 = new Manager();

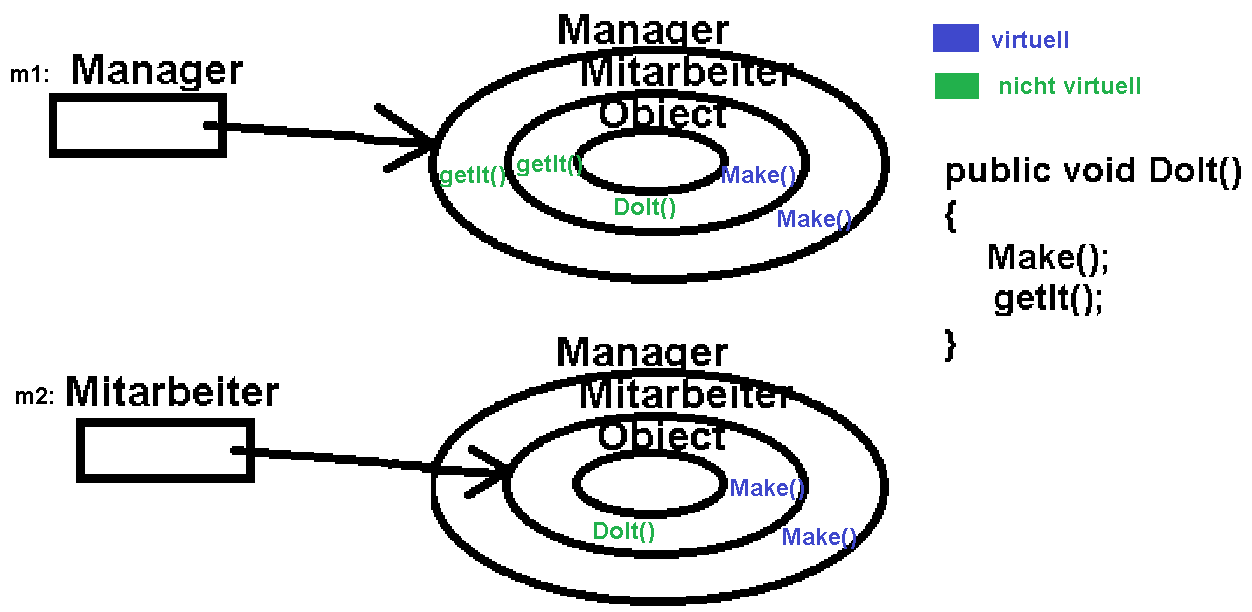
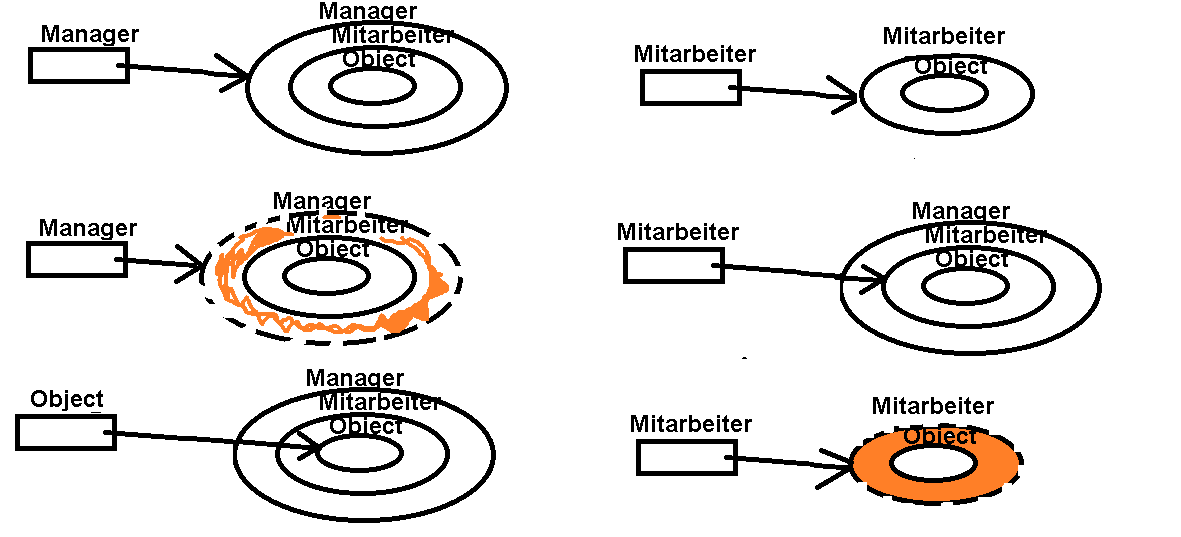
02: Mitarbeiter ref2 = new Mitarbeiter();

03: Manager ref3 = new Mitarbeiter();

04: Mitarbeiter ref4 = new Manager();

05: Object ref5 = new Manager();

06: Mitarbeiter ref6 = new Object();



# Übungen

**Aufgabe 01**

**Lernziel**: Programmcode erstellen

**Zeit**: 50’ (+25’ Besprechung)

**Aufgabe**: Gegeben ist die untenstehende Main-Methode und die Sollausgabe. Versuchen Sie mit Hilfe der Vererbung und deren Konzepte (abstrakte Klassen, virtuelle und abstrakte Methoden etc.) den Programmcode möglichst sinnvoll und redundanzfrei zu erstellen.

class Program

{

static void Main(string[] args)

{

// Person p = new Person(); geht nicht da abstrakt

Person p01 = new Jugendlicher(04, "Karlotta", true);

Person p02 = new Jugendlicher(05, "Ludwig-Bernhard", false);

Person l01 = new Lehrer(02, "Lämpel", 259.95);

List<Person> personen = new List<Person>();

//bei Java: ArrayList<Person> personen = new ArrayList<>();

personen.Add(p01);

personen.Add(p02);

personen.Add(l01);

//bei Java: for (Person p in person){

foreach (Person p in personen) {

p.Ausgabe();

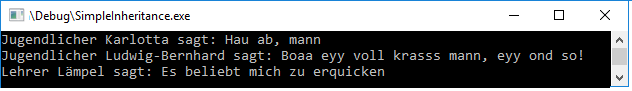
}

Console.ReadLine();

}

}

}



**Aufgabe 02 (**

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

**Zeit**: 60’

**Aufgabe**: Betrachten sie die gegebenen Elemente (Programmcode, UML-Klassendiagramm und die Ausgabe) des beiliegenden Vorlagenprojektes. Lösen sie danach die nachfolgend beschriebenen Aufgaben.





Wenn Sie die Applikation starten erfolgt fogende Ausgabe:



Der Programmcode der Mainfunktion:

class Program {

static void Main(string[] args) {

//Mitarbeiter

Mitarbeiter m1 = new Mitarbeiter("Kunigunde", "Hugentobler", 4800.00, "Roche");

Mitarbeiter m2 = new Mitarbeiter("Adelheit", "Joggenmoser", 6200.00, "Siemens");

Mitarbeiter m3 = new Mitarbeiter("Melchior", "Käser");

m3.setLohn(5900.00);

m3.setFirma("Roche");

//Parteimitglieder

Parteimitglied pm1 = new Parteimitglied("Blocher", "Christoph", "SVP");

Parteimitglied pm2 = new Parteimitglied("Brunner", "Heidi", "FDP");

//Politiker

Politiker po1 = new Politiker("Maurer", "Ueli", 33000, "SVP", "Bundesrat");

Politiker po2 = new Politiker("Cassis", "Ignazio", 33000, "FDP", "Bundesrat");

Politiker po3 = new Politiker("Schneider-Ammann", "Johann", 33000, "FDP", "Bundesrat");

Politiker po4 = new Politiker("Sommaruga", "Simmonetta", 33000, "SP", "Bundesrat");

Politiker po5 = new Politiker("Parmelat", "Guy", 33000, "SVP", "Bundesrat");

Politiker po6 = new Politiker("Berset", "Alain", 33000, "SP", "Bundesrat");

Politiker po7 = new Politiker("Leuthard", "Doris", 33000, "CVP", "Bundesrat");

Politiker po9 = new Politiker("Levrat", "Christian", 100000, "SP", "Nationalrat");

Politiker po8 = new Politiker("Schelbert", "Louis", 100000, "Grüne", "Nationalrat");

Politiker po10 = new Politiker("Darbellay", "Christoph", 100000, "CVP", "Nationalrat");

Politiker po11 = new Politiker("Brunner", "Toni", 100000, "SVP", "Nationalrat");

Politiker po12 = new Politiker("Höltschi", "Pius", 6000, "SVP", "Kantonsrat");

//

Funktionen.Lohnerhoehung(m1, 1.20);

Funktionen.Lohnerhoehung(po12, 2.50);

//

m1.Datenausgabe();

m2.Datenausgabe();

m3.Datenausgabe();

pm1.Datenausgabe();

pm2.Datenausgabe();

po1.Datenausgabe();

po2.Datenausgabe();

po3.Datenausgabe();

po4.Datenausgabe();

po5.Datenausgabe();

po6.Datenausgabe();

po7.Datenausgabe();

po8.Datenausgabe();

po9.Datenausgabe();

po10.Datenausgabe();

po11.Datenausgabe();

po12.Datenausgabe();

}

}

**a)** Debuggen Sie durch den Programmcode um zu verstehen, wie er aufgebaut ist.

**b)** Überlegen Sie sich, wie Sie mit Hilfe der Vererbung möglichst viel redundanten Programmcode eliminieren können. Zeichnen Sie ein UML-Klassendiagramm auf und besprechen Sie Ihre Idee mit dem Nachbarn bevor Sie fortfahren. Setzen Sie danach Ihre Ideen in Programmcode um.

**c)** Ändern Sie den Programmcode der main Methode so ab, dass er wie nachfolgend dargestellt aufgebaut ist. Passen sie die Methode Datenausgabe() so an, dass wieder die korrekte Ausgabe erscheint. Verwenden sie dazu die Schlüsselwörter abstract, virtual, override, etc.

class Program {

static void Main(string[] args) {

**List<Person> allePersonen = new List<Person>();**

//Mitarbeiter

Mitarbeiter m1 = new Mitarbeiter("Kunigunde", "Hugentobler", 4800.00, "Roche");

**allePersonen.Add(m1);**

**allePersonen.Add(** newMitarbeiter("Adelheit", "Joggenmoser", 6200.00, "Siemens") **)**;

Mitarbeiter m3 = new Mitarbeiter("Melchior", "Käser");

m3.setLohn(5900.00);

m3.setFirma("Roche");

**allePersonen.Add(**  m3 **)**;

//Parteimitglieder

**allePersonen.Add(**  new Parteimitglied("Blocher", "Christoph", "SVP") **)**;

**allePersonen.Add(**  new Parteimitglied("Brunner", "Heidi", "FDP") **)**;

//Politiker

**allePersonen.Add(**  new Politiker("Maurer", "Ueli", 33000, "SVP", "Bundesrat") **)**;

**allePersonen.Add(**  new Politiker("Cassis", "Ignazio", 33000, "FDP", "Bundesrat") **)**;

**allePersonen.Add(**  new Politiker("Schneider-Ammann", "Johann", 33000, "FDP", "Bundesrat") **)**;

**allePersonen.add(**  new Politiker("Sommaruga", "Simmonetta", 33000, "SP", "Bundesrat") **)**;

**allePersonen.add(**  new Politiker("Parmelat", "Guy", 33000, "SVP", "Bundesrat") **)**;

**allePersonen.add(**  new Politiker("Berset", "Alain", 33000, "SP", "Bundesrat") **)**;

**allePersonen.add(**  new Politiker("Leuthard", "Doris", 33000, "CVP", "Bundesrat") **)**;

**allePersonen.add(**  new Politiker("Levrat", "Christian", 100000, "SP", "Nationalrat") **)**;

**allePersonen.add(**  new Politiker("Schelbert", "Louis", 100000, "Grüne", "Nationalrat") **)**;

**allePersonen.add(**  new Politiker("Darbellay", "Christoph", 100000, "CVP", "Nationalrat") **)**;

**allePersonen.Add(**  new Politiker("Brunner", "Toni", 100000, "SVP", "Nationalrat") **)**;

Politiker po12 = new Politiker("Höltschi", "Pius", 6000, "SVP", "Kantonsrat");

**allePersonen.Add( po12 );**

Funktionen.Lohnerhoehung(m1, 1.20);

Funktionen.Lohnerhoehung(po12, 2.50);

//

**foreach (Person p in allePersonen) {**

**p.Datenausgabe();**

**}**

}

}