Beziehungen zwischen Klassen

Zwischen Klassen respektive Objekten von Klassen existieren sehr häufig Beziehungen. In der UML kann man folgende Beziehungstypen unterscheiden:

1. Vererbung,
2. Assoziation,
3. Aggregation und
4. Komposition

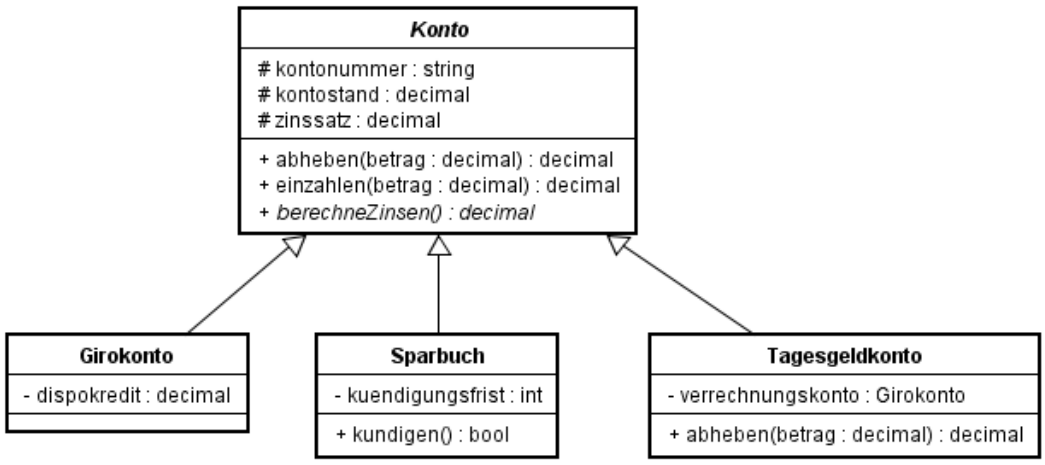
Vererbung[1]

Hierzu ein Zitat aus der Wikipedia:

> *„Die Vererbung (englisch inheritance) ist eines der grundlegenden Konzepte der Objektorientierung und hat große Bedeutung in der Softwareentwicklung. Die Vererbung dient dazu, aufbauend auf existierenden Klassen neue zu schaffen, wobei die Beziehung zwischen ursprünglicher und neuer Klasse dauerhaft ist. Eine neue Klasse kann dabei eine Erweiterung oder eine Einschränkung der ursprünglichen Klasse sein. Neben diesem konstruktiven Aspekt dient Vererbung auch der Dokumentation von Ähnlichkeiten zwischen Klassen, was insbesondere in den frühen Phasen des Softwareentwurfs von Bedeutung ist. Auf der Vererbung basierende Klassenhierarchien spiegeln strukturelle und verhaltensbezogene Ähnlichkeiten der Klassen wider."*[2]

Durch Vererbung wird eine eher allgemein gehaltene Basisklasse (oder auch: Superklasse, Oberklasse, Elternklasse) spezialisiert und in eine abgeleitete Klasse (oder auch: Subklasse, Unterklasse, Kindklasse) überführt. Diese „Spezialisierung" kann eine Erweiterung oder auch eine Einschränkung der Eigenschaften und Methoden der Basisklasse in der abgeleiteten Klasse bedeuten. Die umgekehrte Richtung nennt man „Generalisierung".

Eine Vererbung stellt prinzipiell eine Beziehung der Art „***ist ein***" bzw. „i***s a***" dar. Sehen Sie sich das am folgenden Beispiel einer Basisklasse „**`Konto`**" an:



Ein Girokonto, ein Sparbuch und ein Tagesgeldkonto lassen sich allgemein als Konto bezeichnen; ein Girokonto *ist ein* Konto, ein Sparbuch *ist ein* Konto und ein Tagesgeldkonto ebenso.

1 Eine ausführlichere Beschreibung zur Vererbung findet sich in den Microsoft Docs unter **`https://docs.microsoft.com/de-de/dotnet/csharp/tutorials/inheritance`** abgerufen am 10.12.2017

2 Siehe **`https://de.wikipedia.org/wiki/Vererbung_(Programmierung)`** abgerufen am 10.12.2017

Die Basisklasse **`Konto`** beinhaltet diejenigen Attribute, die jedes Konto beschreiben. Jede Art von Konto hat eine Kontonummer, einen Kontostand und einen Verzinsungssatz für den Kontostand. Des Weiteren ist für jede Art von Konto eine Verzinsung zu berechnen, die sich aber bei verschiedenen Kontenarten unterscheiden kann.

Ein Girokonto hat üblicherweise noch einen Dispositionsrahmen innerhalb dessen das Girokonto überzogen werden darf, Sparbücher können innerhalb einer Kündigungsfrist gekündigt (aufgelöst) werden und von einem Tagesgeldkonto kann nicht direkt abgehoben werden, sondern der abzuhebende Betrag wird auf ein Girokonto umgebucht und das Tagesgeldkonto ist typischerweise ein Guthabenkonto, so dass keine negativen Kontostände entstehen können.

Im Rahmen der Spezialisierung erhalten dann die abgeleiteten Klassen zusätzliche Attribute und zusätzliche Methoden zu denen der Basisklasse. Es können aber auch die Methoden der Basisklasse neu definiert („überschrieben“) werden, so dass diese ein neues Verhalten aufweisen! So kann beim Tagesgeldkonto bei jeder Abhebung überprüft werden, ob der Kontostand negativ würde und diese Abhebung damit nicht möglich ist.

Die Basisklasse kann als eine sogenannte „abstrakte Klasse“ definiert werden. In diesem Fall wird der Name der Klasse im Klassendiagramm in kursiver Schrift geschrieben. Abstrakte Klassen können nicht instanziiert werden; d.h. es können keine Objekte mit diesem Datentyp erstellt werden. Im oben dargestellten Beispiel ist Konto eine abstrakte Klasse.

Nachfolgend ist die (unvollständige) Implementation der vier Klassen dargestellt:

```
public abstract class Konto
{
      protected string kontonummer;
      protected decimal kontostand;
      protected decimal zinssatz;

      public virtual decimal abheben(decimal betrag)
      {
            kontostand -= betrag;
            return kontostand;
      }

      public decimal einzahlen(decimal betrag)
      {
            kontostand += betrag;
            return kontostand;
      }

      public abstract decimal berechneZinsen();
}
```

```
public class Sparbuch : Konto
{
      private int kuendigungsfrist;

      public bool kundigen()
      {
            // Sparbuch auflösen
      }
}
```

```
public class Girokonto : Konto
{
      private decimal dispokredit;
}
```

```
public class Tagesgeldkonto : Konto
{
      private Girokonto verrechnungskonto;

      public override decimal abheben(decimal betrag)
      {
            if (kontostand - betrag >= 0)
            {
                  kontostand -= betrag;
                  verrechnungskonto.einzahlen(betrag);
            }
            return kontostand;
      }
}
```

Um eine Klasse als abstrakte Klasse zu implementieren, wird das Schlüsselwort „**`abstract`**“ verwendet.

Nun können keine Objekte dieser Klasse instanziiert werden! Damit die Attribute der Basisklasse auch bei den Kindklassen verfügbar sind, müssen diese mit dem Zugriffsbezeichner „**`protected`**“ versehen werden.

Attribute mit dem Zugriffsbezeichner „**`private`**“ sind nur in Objekten der Basisklasse verfügbar, sofern diese nicht abstrakt ist!

Um die Vererbung bei den abgeleiteten Klassen vorzunehmen, wird hinter dem Klassennamen, getrennt durch einen Doppelpunkt, der Name der Basisklasse angegeben.

Soll die Methode einer Basisklasse durch eine Methode einer abgeleiteten Klasse überschrieben werden können, enthält die Methode der Basisklasse das Schlüsselwort „**`virtual`**“. Falls sie dann tatsächlich überschrieben wird, enthält die Methode der abgeleiteten Klasse das Schlüsselwort „**`override`**“, um dies deutlich zu machen.

Basisklassen können sogenannte „abstrakte Methoden“ enthalten. Abstrakte Methoden in einer Basisklasse werden ebenfalls mit dem Schlüsselwort „**`abstract`**“ gekennzeichnet und enthalten keine Methodendefinition (siehe die Methode **`berechneZinsen()`** in **`Konto`**).

Durch dieses Vorgehen wird erzwungen, dass eine abgeleitete Klasse diese Methode **implementieren muss** (dies ist in den Beispielen oben nicht gemacht worden). Andernfalls meldet der Compiler entsprechende Fehler! Abstrakte Methoden werden ebenfalls in kursiver Schrift dargestellt.

Ein sehr nützliche Verhaltensweise bei der Vererbung ist, dass Objekte der Kindklasse quasi auch immer Objekte vom Typ der Elternklasse sind. Somit lassen sich in einem generischen Container vom Typ der Elternklasse auch Objekte der Kindklassen deponieren.

> Dieses Verhalten zählt zu der sogenannten „**Polymorphie**“ (Vielgestaltigkeit) und ist ein grundlegendes Konzept in der Objektorientierung.

In der Programmiersprache C# existiert keine Mehrfachvererbung wie z.B. in C++. Bei der Mehrfachvererbung kann **eine** Kindklasse von **mehreren** Elternklassen erben. Allerdings existiert in C# das Konzept der sogenannten „*Interfaces*“[3] (Schnittstellen).

Weiterhin findet das Schlüsselwort **`sealed`** („versiegelt“) im Rahmen der Vererbung seine Anwendung.

```
public abstract class Konto
{}

public sealed class Tagesgeldkonto : Konto
{}

// Das Folgende ist NICHT möglich,
// weil Tagesgeldkonto „versiegelt“ ist
public class BesonderesTagesgeldkonto :  Tagesgeldkonto
{}
```

Wird eine Kindklasse als „versiegelte Klasse“ implementiert, kann von solch einer Kindklasse ausgehend keine weitere Vererbung vorgenommen werden; sie ist somit das Ende einer Vererbungslinie. Im Beispiel wird die Klasse **`Tagesgeldkonto`** von der Klasse **`Konto`** abgeleitet und versiegelt. Eine Kindklasse **`BesonderesTagesgeldkonto`**, die von der Klasse **`Tagesgeldkonto`** erbt, kann somit nicht definiert werden.

Der Modifizierer **`sealed`** kann auch auf Eigenschaften und Methoden einer Klasse angewendet werden.

Sehen Sie sich hierzu die Beispiele aus der C#-Referenz von Microsoft an:
`https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/sealed`

Ebenfalls wichtig im Rahmen der Vererbung sind die beiden Begriffe „early Binding (frühe Bindung)“ und „late Binding (späte Bindung)“. Informieren Sie sich über die Bedeutung dieser beiden Begriffe. Einen Ausgangspunkt zur Information kann der folgende Link darstellen:
`http://walter-digital.de/java.sem/seminar/syntax/polymorphie.html`

3 Siehe hierzu z.B. **`http://openbook.rheinwerk-verlag.de/visual_csharp_2010/visual_csharp_2010_04_007.htm`** abgerufen am 10.12.2017

Aufgaben

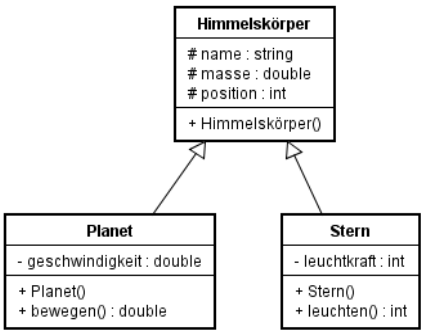
1. Kopieren Sie sich aus dem Pool den Ordner „Beziehungen zwischen Klassen“ mit der gleichnamigen darin enthaltenen Projektmappe in Ihren Ordner mit den Visual Studio-Projektmappen.

   Sehen Sie sich das Beispiel-Projekt „Vererbung“ in dieser Projektmappe an und bringen Sie es zur Ausführung.

   Erstellen Sie drei weitere Objekte; ein Girokonto-Objekt **`g2`**, ein Tagesgeldkonto-Objekt **`t2`** und ein Sparbuch-Objekt **`s2`**.

   Fügen Sie alle drei Objekte der Liste aller Konten hinzu und führen Sie Konto-Operationen wie „abheben“ und „einzahlen“ durch. Kontrollieren Sie mit Hilfe von Testausgaben die Funktionalität.

2. Implementieren Sie in einem neuen Projekt namens „Himmel“ die dargestellte Vererbung. Ergänzen Sie die Klassen um Properties für die jeweiligen Attribute.
   Erzeugen Sie in Main() mehrere Objekte vom Typ **`Planet`** bzw. **`Stern`**, speichern Sie diese in einer Liste und geben Sie alle Attributwerte aus.
   Überprüfen Sie hierzu, um welchen Objekttyp es sich jeweils handelt, um dessen Attributwerte für **`geschwindigkeit`** bzw. **`leuchtkraft`** auszugeben.



3. Das Schreiben und Lesen der Daten aus dem Adressbuch soll auf verschiedene Arten erfolgen können: z.B. in der bisherigen Form in einer Textdatei, zusätzlich in Form einer XML-Datei oder in einer Datenbank, die von einem Datenbankserver verwaltet wird. Diese verschiedenen Ausprägung des Speichern und Lesen der Daten soll durch das Konzept der Vererbung realisiert werden.

   Als abstrakte Basisklasse soll eine Klasse namens **`ReaderWriter`** dienen. Diese Klasse soll ein einziges Attribut bzw. Property namens **`Persons`** mit dem Typ **`List<Person>`** haben, welches auf die Personenliste des Adressbuches verweist.

   Die Klasse soll eine Standard-Konstruktormethode ohne Parameter haben. Weiter sollen lediglich die beiden abstrakten Methoden **`readAllPersons():void`** und **`storeAllPersons():void`** vorhanden sein.

   Mittels Vererbung sollen von dieser Basisklasse die Kindklassen **`TextReaderWriter`**, **`XMLReaderWriter`** und **`DatabaseReaderWriter`** abgeleitet werden. In diesen drei Kindklassen sollen nun die beiden genannten abstrakten Methoden der Basisklasse definiert werden. Fernerhin sind notwendige Attribute bei den Kindklassen hinzuzufügen.

   Überlegen Sie, welche Attribute dies sein sollten und entwerfen Sie ein entsprechendes Klassendiagramm.