Kapitel 17

Objektorientierte Programmierung

Prozedurale Programmierung

- Die Prozedurale Programmierung trennt die Daten von den Funktionen, die diese Daten bearbeiten.
- Das hat zur Folge, dass ein Programmierer selbst für die Initialisierung von Daten vor ihrer Verwendung sorgen muss und dass beim Aufruf einer Funktion korrekte Daten übergeben werden.
- Wird die Darstellung der Daten geändert, müssen auch die entsprechenden Funktionen angepasst werden.
- Ein solches Vorgehen erhöht den Wartungsaufwand und ist fehleranfällig.

Datenabstraktion

- Ziel der "Objektorientierten Programmierung", auch kurz OOP, ist es, komplexe Sachverhalte als Ganzes abzubilden, d.h. die Eigenschaften (Daten, Attribute) und die darauf wirkenden Methoden (Funktionen, Fähigkeiten) zusammenzufassen.
- Die Objekte der OOP bilden eine Einheit aus Eigenschaften und Methoden.
- Eigenschaften und Methoden einer Klasse nennt man auch Member einer Klasse.

Datenabstraktion

- Vor der Implementierung werden in einem Komponentenentwurf die Objekte herausgearbeitet. Mehrere gleichartige Objekte werden dann zu einer Klasse zusammengefasst.
- Dabei werden alle Gemeinsamkeiten der gleichartigen Objekte herausgearbeitet und in einer Klasse vereinigt.
- Die Bildung von Klassen zur Beschreibung von Objekten nennt man auch Datenabstraktion.

Klassen und Objekte

- Eine Klasse ist ein selbst definierter Datentyp.

- Erstellt/Benutzt man eine Variable dieses Datentyps, so nennt man dies nicht mehr Variable, sondern *Objekt* (oder auch Instanz) dieser Klasse.
- Die Funktionen, die zu dieser Klasse gehören, heißen: *Methoden*

Definition einer Klasse

- Die Klassendefinition beginnt mit einem *classdef*, gefolgt von einem *Klassennamen* und endet mit einem end.
- Dazwischen stehen die Eigenschaften (Variablen) und Methoden (Funktionen) der Klasse.

```
classdef myClass
    properties
    ...
    methods
    ...
end
```

Definition einer Klasse

- Jede Klasse besteht aus einer Menge von Eigenschaften und Methoden.
- Eigenschaften und Methoden verschiedener Klassen dürfen die gleichen Namen tragen.
- Jede Klasse wird beschrieben durch eine eigene .m-Datei, die genauso heißen muss, wie die Klasse.

Definition der Klasse CKonto

- Die weiteren Erklärungen zu Klassen erfolgt nun anhand eines Beispiels. Wir wählen hierzu eine Klasse mit dem Namen: *CKonto*.

- Eine Klasse *CKonto* enthält z.B. die Eigenschaften:
  - Name des Kontoinhabers (m_name)
  - Kontonummer (m_nummer)
  - Kontostand (m_stand)

Definition der Klasse CKonto

- Ferner enthält die Klasse *CKonto* die Methoden:
  - Name abfragen und ändern (GetName, SetName)
  - Kontonummer abfragen und ändern (GetNummer, SetNummer)
  - Kontostand abfragen und ändern (GetStand, SetStand)
  - Kontodaten ausgeben (Ausgabe)
  - Kontodaten eingeben (Eingabe)

- Diese Eigenschaften und Methoden gehören zu jedem beliebigen Konto.

Definition der Klasse CKonto

- Innerhalb der Klassendefinition für CKonto stehen zunächst die Eigenschaften:

```
classdef CKonto
    properties
        m_name;
        m_nummer;
        m_stand;
    end

    methods ...    % siehe übernächste Folie
end
```

Definition der Klasse CKonto

- Danach folgen die Methoden der Klasse.

- Für jede Eigenschaft gibt es eine Set- und eine Get-Methode.
- Mit diesen Methoden kann man auf die Eigenschaften zugreifen.

Definition der Klasse CKonto

```
classdef CKonto
    properties ...          % siehe 2 Folien vorher

    methods
        function obj = SetName(obj, name)
            obj.m_name = name;
        end

        function name = GetName(obj)
            name = obj.m_name;
        end
    end
end
```

Definition der Klasse CKonto

- Außerdem gibt es noch eine Ausgabe- und eine Eingabe-Methode.

```
function obj = Ausgabe(obj)
    fprintf("Name: %-8s \n", obj.m_name);
    fprintf("Nummer: %-8d \n", obj.m_nummer);
    fprintf("Stand: %-8.2f \n", obj.m_stand);
end
function obj = Eingabe(obj)
    name = input("Name eingeben : ", "s");
    nummer = input("Nummer eingeben : ");
    stand = input("Stand eingeben : ");
    obj.m_name = name;
    obj.m_nummer = nummer;
    obj.m_stand = stand;
end
```

Datenkapselung

- Die Eigenschaften einer Klasse sind bei guter Programmierung nur über die Methoden veränderbar.
- Die Menge aller Methoden einer Klasse nennt man auch Schnittstelle der Klasse.
- Die Schnittstelle einer Klasse muss sich nicht ändern, wenn sich die interne Darstellung der Daten ändert.
- Das hat den Vorteil, dass alle die Klasse verwendenden Programmteile unverändert bleiben können.

Zugriffsrechte bei Klassen

- Es ist möglich, den Membern (Eigenschaften, Methoden) einer Klasse verschiedene Zugriffsrechte zuzuordnen.
- Diese Zugriffsrechte definieren, ob auf ein Member von außerhalb der Klasse zugegriffen werden darf oder nicht.
- Die Schlüsselwörter hierfür sind: *private* und *public*.
- Verwendet werden können diese Schlüsselwörter bei
  - properties
  - methods

Zugriffsrechte bei Klassen

- Der Zugriff von außerhalb wird durch das Schlüsselwort *private* gesperrt.

Für properties:

```
properties (Access = private)
    m_name;
end
```

Für methods:

```
methods (Access = private)
    function name = ...
end
```

Zugriffsrechte bei Klassen

- Das Schlüsselwort *public* hingegen erlaubt den Zugriff von außerhalb.

```
properties (Access = public)
    m_name;
end
```

```
methods (Access = public)
    function name = …
end
```

- Wobei man das weglassen kann, denn es ist per default *public.*

```
properties
    m_name;
end
```

```
methods
    function name = …
end
```

Zugriffsrechte bei Klassen

- Ist kein Zugriffsrecht angegeben, ist es standardmäßig *public*.
- Methoden einer Klasse haben generell immer Zugriff auf alle anderen Member der eigenen Klasse.
- Die Schlüsselwörter (*public*, *private*) sind innerhalb einer Klasse gültig, bis sie durch ein anderes verändert werden.
- Die Reihenfolge und Anzahl von *private* bzw. *public* ist beliebig.
- Die Eigenschaften einer Klasse werden meist als *private* definiert. (Um aber trotzdem Zugriff zu haben, werden dann für diese Eigenschaften Set- und Get-Methoden erstellt.)

Zugriffsrechte bei Klassen

- Es gibt noch ein paar weitere Attribute, die man angeben kann.
- Hier sei nur noch erwähnt: *Constant*

- Durch *Constant* kann man die Veränderung einer Eigenschaft verbieten und quasi eine richtige Konstante erschaffen.

```
properties (Constant)
    PI = 3.1415;
end
```

Definition eines Objekts

- Ist die Klasse *CKonto* mit *properties* und *methods* beschrieben worden, können Objekte dieser Klasse definiert werden:

```
konto = CKonto;
```

Oder auch:

```
konto = CKonto();
```

- Man sagt: *konto* ist ein Objekt der Klasse *CKonto*.

Definition eines Objektfeldes

- Es ist auch möglich ein Feld von Objekten zu definieren:

```
konto(1:10) = CKonto;
```

Oder auch:

```
konto(1, 10) = CKonto;
```

Aufruf der Methoden einer Klasse

- *Funktion* wird in der objektorientierten Programmierung *Methode* genannt.
- Der Aufruf der Methode einer Klasse erfolgt immer für ein bestimmtes Objekt der Klasse.

- Das Objekt wird entweder als Parameter der aufrufenden Funktion mitgegeben. Oder der Name des Objekts muss durch einen Punkt getrennt vor den Namen der Methode geschrieben werden.

Aufruf der Methoden einer Klasse

- Das Objekt wird entweder als Parameter der aufrufenden Funktion mitgegeben:

```
SetName(konto, "Hans");
```

- Oder das Objekts muss, durch einen Punkt getrennt, **vor** den Namen der Methode geschrieben werden:

```
konto.SetName("Hans");
```

Aufruf der Methoden einer Klasse

- Damit das Objekt *konto* aber überhaupt geändert wird, muss man das, was die Methode *SetName()* zurückgibt, auch wieder im Objekt speichern.
- Daher sieht der komplette Aufruf dann so aus:

```
konto = SetName(konto, "Hans");
```

- Bzw.:

```
konto = konto.SetName("Hans");
```

Beispiel der Klasse CKonto

- Ein komplettes Beispiel der Klasse CKonto nebst Hauptprogramm finde Sie unter Beispiel 18 (auf der Webseite zu finden).