Exzerpt aus dem OOP-Skript von Dr. Golla (TU Dresden)

- OOP ist mehr eine Software-Entwurfsmethode als ein Programmierparadigma
- Der Trend zur OOP kann bestenfalls als Wechsel des Paradigmas in soweit angesehen werden, als man sich weg von den Begriffen Daten und Prozeduren bei der prozeduralen / strukturierten Programmierung hin zu den Begriffen Klassen, Objekte, Attribute und Botschaften bei der OOP bewegt.
- Die OOP kapselt die Daten und die darauf operierenden Methoden in Objekten. Diese Objekte verfügen über eine wesentliche Eigenschaft: Ist eine Klasse definiert, so kann man nicht nur von ihr Objekte dieses Typs deklarieren, man kann die Datenfelder, Eigenschaften und Methoden dieser Klasse auch an Nachfolgerklassen *vererben*. Damit müssen Algorithmen oder ganze Programmteile nicht immer wieder neu programmiert werden, denn durch geschicktes vererben kann der Code immer wieder verwendet werden.
- Wichtige Merkmale objektorientierter Programmierung sind:
  - **Datenkapselung:** Objekte beinhalten Daten und Methoden, nur bestimmte Methoden werden exportiert, die internen Details der Implementierung können verborgen bleiben → Geheimnisprinzip
  - **Vererbung:** Objekte (Nachkommen) können andere Objekte (Vorfahren) beerben und erhalten dadurch deren Datenfelder und Methoden, die Datenfelder können ergänzt werden
  - **Polymorphismus:** Verschiedene Objekte besitzen verschiedene Methoden mit gleichem Namen, aber unterschiedlicher Funktion, sie können dadurch auf die gleiche Botschaft unterschiedlich reagieren

Grundbegriffe der OOP

Klassen

Eine Klasse ist die Beschreibung für eine Menge von Objekten mit gleicher Struktur. Die Klasse definiert die Daten (Attribute), die diese Objekte besitzen, die Eigenschaften und die Methoden, die für diese Objekte zur Verfügung stehen.

Objekte (Instanzen)

Objekte sind konkrete Vertreter einer Klasse. Sie besitzen Daten*werte* (Attribut*werte*). Instanzen einer Klasse unterscheiden sich also nur durch die evtl. unterschiedliche Belegung der Datenfelder mit Werten. Von jeder Klasse lassen sich beliebig viele Instanzen erzeugen, die alle die gleiche Struktur (die der Klasse) besitzen. Jede Instanz belegt Platz im Arbeitsspeicher. Die Daten können hierbei nur über die entsprechenden Methoden, welche durch eine genau spezifizierte Schnittstelle angesprochen werden, modifiziert werden. Instanzen können durch Anwendung ihrer Methoden verschiedene Zustände annehmen (sie führen mit sich selbst Operationen aus: erzeugen, verändern, zerstören).

Methoden

Methoden nennt man die Prozeduren und Funktionen, die zu einer Klasse gehören.

Datenfelder (Attribute)

Es handelt sich um Variable, die innerhalb einer Klasse (mit Angabe des Typs) definiert werden. Bei der Erzeugung einer Instanz werden sie mit konkreten .Werten. belegt, den Attribut*werten*. Auf diese Daten kann nur durch objekteigene Methoden zugegriffen werden.

Eigenschaften (Properties)

Mittels der Vergabe von Eigenschaften (zum Lesen und Schreiben) können die Zugriffe auf die Daten (Attribute) geregelt werden.

Vorteile des OOP-Konzepts:

- Durch die ereignisgesteuerte Abarbeitung ist eine günstige Modularisierung möglich.
- Schnittstellen sind klar definiert und ermöglichen damit eine gute Koordination bzgl. Teilaufgaben. Es gibt keine globalen Variablen.
- Funktionen und Prozeduren können vererbt aber auch erweitert werden.

Definition von Klassen und Erzeugen von Objekten

Definition einer Klasse

Eine Klassendefinition darf nur im *Interface*-Teil einer Unit stehen. Innerhalb der Klassendefinition können für einzelne Teile bestimmte *Zugriffsrechte* (privat, public, protected) vergeben werden.
Klassendefinition allgemein:

```
type Tname_d_klasse =    class
                              Datenfelder
                              Methodenköpfe
                              Eigenschaften
                         end;
```

Beispiel:

```
type tbruch=class
     zaehler,nenner:integer; {Datenfeld}
     function bruchtostring:string;{Methodenköpfe}
     procedure initbruch(z_n,n_n:string);
     procedure kuerzebruch;
     procedure addbruch(b:tbruch);
     procedure subbruch(b:tbruch);
     procedure multibruch(b:tbruch);
     procedure divibruch(b:tbruch);
end;
```

Erzeugen von Objekten

Zum Erzeugen von Objekten benötigt man Variablen vom Typ der entsprechenden Klasse.
Beispiel:

```
var bruch1: tbruch;
```

Ein Objekt entsteht erst durch den Aufruf eines **Konstruktors** zum Erzeugen desselben. Damit ist klar, dass dieser Aufruf immer als erster erfolgen muss. Erst danach kann das jetzt existierende Objekt seine Methoden aufrufen.
In Object Pascal heißt dieser **Konstruktor** `create`. Es handelt sich um eine bereits vom System bereitgestellte Methode.
Das Erzeugen der Instanz von `bruch1` geschieht durch folgenden Aufruf:

```
bruch1:=tbruch.create;
```

Allgemein also

```
Klassenvariable:=Klasse.create;
```

Durch diesen Aufruf wird Speicherplatz im Hauptspeicher für alle Daten (Attribute) dieses Objektes reserviert.
Das Gegenstück zu einem Konstruktor ist der **Destruktor**. Er dient dazu, den von einem Objekt belegten Speicherplatz wieder freizugeben, falls das Objekt nicht mehr benötigt wird. Das System

stellt dafür zwei Möglichkeiten zur Verfügung, die Methoden `free` und `destroy`, also `bruch1.free;` oder `bruch1.destroy;`

Am Beispiel von `tbruch` kann das so aussehen:

```
type tbruch = class
     zaehler,nenner:integer;
     constructor create(z,n:integer);
     ...
end;

implementation

constructor tbruch.create(z,n:integer);
begin
     inherited create; //Aufruf des vorhandenen Konstruktors
     zaehler:=z;
     nenner:=n;
end;

var b1:tbruch;
...
```

Der Aufruf erfolgt in der Form `b1:=tbruch.create(2,5);`

Datenkapselung und Zugriffsrechte

Die Zusammenfassung von Datenfeldern (Attributen), Eigenschaften und Methoden zu einer Einheit nennt man **Datenkapselung**. Durch die Kapselung kann ein ungewollter Zugriff auf Daten oder Methoden eines Objektes vermieden werden. Außerdem existieren weitere Schutzmechanismen in Form von Zugriffsrechten.

Vererbung

Vererbung bedeutet:

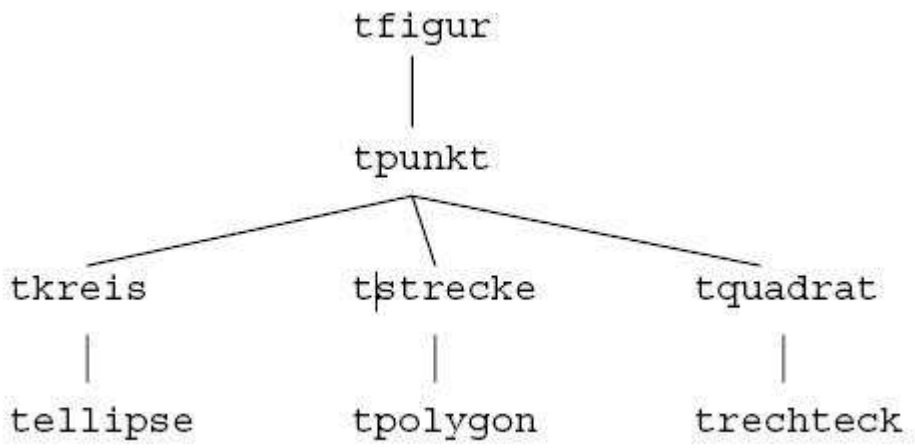
- Nachkommen können ihre Vorfahren beerben und erhalten damit automatisch deren Datenfelder, Eigenschaften und Methoden.
- Beim Nachfolger können Methoden verändert werden. Es ist auch möglich neue Methoden, Datenfelder und Eigenschaften zu definieren.

Die Klasse, von welcher geerbt wird, heißt Elternklasse, und die Klasse, welche erbt, nennt sich Kindklasse.

Definition eines Nachfolgers:
Ein Nachfolger (im Bsp. `TNameKlasse2`) einer Klasse wird definiert, indem man hinter das Schlüsselwort `class` in runden Klammern die Elternklasse (im Bsp. `TNameOberKlasse`) schreibt:

```
type TNameKlasse2 = class(TNameOberKlasse)
                         Datenfelder
                         Methodenköpfe
                         Eigenschaften
                    end;
```

Beispiel für eine Hierarchie:



Wir definieren einen Teil der Klassen dieser Hierarchie.

```
type
TFigur=class
     function ausgabe1:string;
     function ausgabe2:string;
end;

TPunkt=class(TFigur)
     x,y:integer;
     procedure init(x_neu,y_neu : integer);
     function getx:integer;
     function gety:integer;
     function zeige:string;
end;

TKreis=class(TPunkt)
     r:integer;
     procedure init(x_neu,y_neu,r_neu : integer);
     function getr:integer;
     function zeige:string;
end;

TQuadrat=class(TPunkt)
     a:integer;
end;

TRechteck=class(TQuadrat)
     b:integer;
end;
```

Vor dem erstmaligen Arbeiten mit Objekten selbstdefinierter Klassen muss die Methode `Create` aufgerufen werden, um die Objekte zu erzeugen. Schreibweise:

```
objekt:=objektklasse.create;
```

Erst danach können die Objekte ihre selbstdefinierten Methoden bzw. die ihres Vorgängers verwenden.

Beispiel:

```
var  p1:tpunkt;
```

```
     k1:tkreis;
     q1:tquadrat;
     r1:trechteck;

p1:=tpunkt.create;
k1:=tkreis.create;
q1:=tquadrat.create;
r1:=trechteck.create;
```

Erben von Datenfeldern

In der angegebenen Klassenhierarchie enthält `tkreis` nur das Datenfeld `r` (Radius). Objekte der Klasse `tkreis` (und alle anderen Nachkommen) können aber auch die Datenfelder `x` und `y` (z. B. als Koordinaten für den Kreismittelpunkt) ihres Vorgängers `tpunkt` nutzen. Das geschieht zum Beispiel innerhalb der Implementation der Methode `init` für den Kreis (s. unten).
Generell kann man eine Klassenhierarchie so anlegen, dass in der Oberklasse vorhandene Datenfelder genutzt werden und in der Nachfolgerklasse - wenn erforderlich. weitere definiert werden. Da in obiger Hierarchie z. B. `trechteck` eine Nachfolgerklasse von `tquadrat` ist, kommt nur ein Datenfeld `b` für die zweite Seite des Rechtecks hinzu, für die erste steht das Datenfeld `a` von `tquadrat` zur Verfügung.

Erben von Methoden von Vorfahr-Klassen

Beispiel:
`tfigur` ist Vorgänger von `tpunkt, tkreis, ...` und besitzt die beiden Methoden `ausgabe1` und `ausgabe2`. Es sind Funktionen vom Typ `string`.

```
     c:=k1.ausgabe1;
     d:=q1.ausgabe2;
     e:=p1.ausgabe1;
```

`k1` ist vom Typ `tkreis` und enthält die Methode `ausgabe1` nicht explizit. Es erbt sie von `tfigur`.
Das **Erben** kann auch durch Verwendung von `inherited` innerhalb der Definition von Methoden der Nachfolgerklassen erfolgen.
Beispiel:
Wir betrachten dazu die Methodenimplementationen zum Initialisieren von Punkt und Kreis:

```
procedure tpunkt.init(x_neu,y_neu : integer);
begin
     x:=x_neu;
     y:=y_neu
end;
procedure tkreis.init(x_neu,y_neu,r_neu : integer);
begin
     inherited init(x_neu,y_neu); //tpunkt.init(x_neu,y_neu)
     r:=r_neu
end;
```

Die Methode `init` von `tkreis` erbt also mittels `inherited` die Methode `init` von `tpunkt`.

Polymorphie

Der Begriff „polymorph“ stammt aus dem Griechischen und bedeutet soviel wie „mehrfache Form besitzen“ (Vielgestaltigkeit).
In der Informatik allgemein versteht man darunter, dass Programmteile - in Abhängigkeit von der jeweiligen Situation - ein unterschiedliches, aber für diese Situation eindeutiges Verhalten zeigen.
Ein polymorphes Objekt ist eine Objektvariable, von der beim Compilieren nicht klar ist, welcher Klasse diese Variable zur Laufzeit zuzuordnen ist.
Objekte sind kompatibel
a) wenn sie Instanzen derselben Klasse sind, also
`var p1, p2: tpunkt; . . .`
`p1:=p2;` oder `p2:=p1;`
b) wenn sie Vorfahr und Nachkomme in derselben Klasse sind. Hier gilt die Kompatibilität aber *nur in einer Richtung*: Die Instanz eines Nachkommen (rechte Seite) kann einem Vorfahrobjekt (linke Seite) zugewiesen werden.

```
var
     p1, p2: tpunkt;
     k1:tkreis;
     q1:tquadrat,
     r1:trechteck;
```

Korrekte Zuweisungen sind `p1:=r1; p1:=k1; p1:=q1; q1:=r1;` falsch dagegen
`k1:=p1; q1:=p1;`
Polymorphie liegt auch vor beim

- **Überschreiben** geerbter Methoden (override - nur für **virtuelle** und **dynamische** Methoden)
- **Überladen** von Methoden (overload - geht grundsätzlich mit allen Methoden)

Ein Beispiel für das **Überladen von** Methoden :

```
implementation
type Tueberl=class
                    a,b:integer;
                    c,d:real;
                    procedure init(a_neu,b_neu:integer);overload;
                    procedure init(c_neu,d_neu:real); overload;
                    function teile(x,y:real):real;overload;
                    function teile(x,y:integer):integer;overload;
               end;
```

Hier ist schon zu sehen, dass die Methoden `init` und `teile` in derselben Klasse jeweils zweimal aufgeführt werden. Dies geht nur, wenn sie mit `overload` gekennzeichnet sind und unterschiedliche Parameterlisten haben. Jedoch müssen Anzahl der Parameter und auch die Parametertypen (Werte- bzw. Referenzparameter) übereinstimmen.

```
//Implementation der Methoden
procedure Tueberl.init(a_neu,b_neu:integer);
begin
     a:=a_neu;
     b:=b_neu;
end;
procedure Tueberl.init(c_neu,d_neu:real);
begin
     c:=c_neu;
     d:=d_neu;
end;
function Tueberl.teile(x,y:real):real;
begin
```

```
     result:=x/y;
end;
function Tueberl.teile(x,y:integer):integer;
begin
     result:=x div y;
end;
```
Die Methoden sind nun also für beide Fälle, die sich jeweils in der Art der Typen unterscheiden definiert worden.

Zusammenfassung

Merkmale objektorientierter Software

- Das Software-System besteht aus Objekten.
- Das System aktivieren bedeutet Objekte zu aktivieren

Objekte besitzen folgende Eigenschaften

- Sie senden und empfangen Nachrichten.
- Sie haben eine bestimmte Lebensdauer (sie werden erzeugt und vernichtet).
- Sie sind aktiv und verändern sich während ihrer Lebensdauer.
- Sie sind an ihren konkreten Eigenschaften identifizierbar.

Aufbau und Aktivität von Objekten

Ein Objekt ist ein nach außen abgeschlossenes Gebilde (Kapsel), das aus zwei Teilen besteht:

- Es enthält Daten (Attribute), die seinen Zustand repräsentieren.
- Es besitzt Methoden (Operationen), mit denen es seine Daten verändern kann.

Ein Objekt ist aktiv, wenn es eine seiner Methoden ausführt. Die Aktivität wird ausgelöst, sobald das Objekt eine Nachricht (Botschaft) erhält.

Klassen (Objekttypen)

Die Klasse ist die allgemeine Beschreibung (Definition) für alle Objekte eines Typs.
Eine Klasse definiert für ihre Objekte

- die Datenfelder und Attribute und
- die Methoden, die die Objekte verwenden können.

Objekte sind konkrete Vertreter ihrer Klasse mit eigenen Daten- und Attributwerten. Zu einer Klasse können beliebig viele Objekte existieren. Sie unterscheiden sich in den Werten ihrer Daten bzw. Eigenschaften.

Vererbung

Vererbung setzt im allg. eine hierarchische Anordnung der Klassen voraus, also Oberklasse (Elternklasse) und Unterklasse (Kindklasse).
Die Unterklasse erbt von ihrer Oberklasse

- Datenfelder, Eigenschaften und
- Methoden.

Die Unterklasse kann neue Datenfelder und Methoden hinzufügen, sie kann geerbte Methoden modifizieren (neu definieren).

Nachrichten (Botschaften)

- Objekte reagieren auf Nachrichten, indem sie eine ihrer Methoden ausführen.
- Dazu müssen sie verstehen, ob die Nachricht für sie bestimmt ist und dann reagieren.

Polymorphismus

Polymorphismus bedeutet von verschiedener Gestalt sein. In Programmiersprachen gibt es mehrere Formen von „Vielgestaltigkeit“

- Variable können unterschiedliche Typen annehmen.
- Operatoren können auf unterschiedliche Operandentypen angewendet werden:
  z. B. der Operator +
  a+b wenn a, b vom Typ integer, dann ganzzahlige Addition
  a+b wenn a, b vom Typ string, dann Verkettung von Zeichenketten
  a+b wenn a, b vom Typ set, dann Bildung der Vereinigungsmenge

Im Rahmen der OOP:
Methoden mit gleichem Namen enthalten unterschiedliche Implementierung:

- Überladen (Overloading) von Methoden ist für alle Methoden möglich.
- Überschreiben (Overriding) von geerbten virtuellen oder dynamischen Methoden in der Unterklasse.