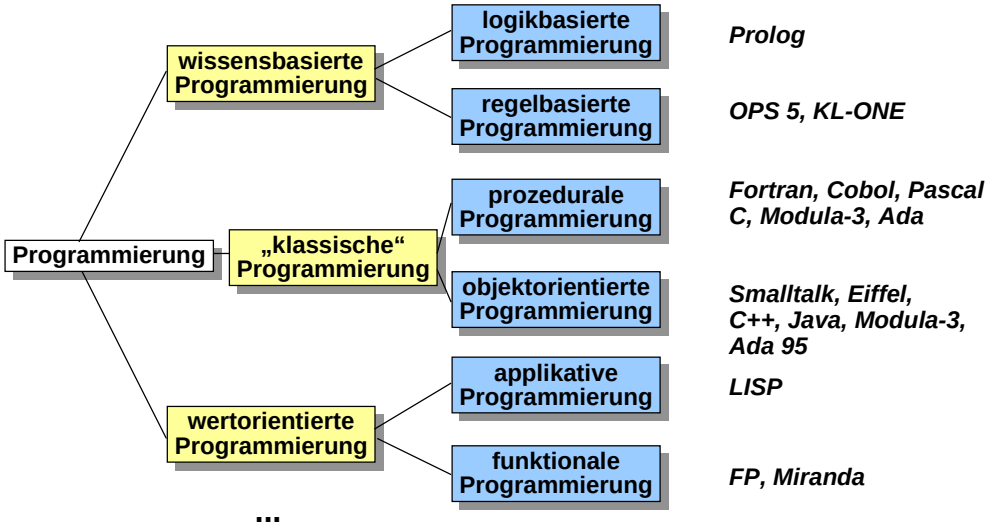
Objektorientierte Programmierung I

- Verständnis der objektorientierten Programmierung
- Objekte
- Klassen
- Vererbung
- Polymorphismus und dynamisches Binden
- Diskussion

Verständnis der OO Programmierung

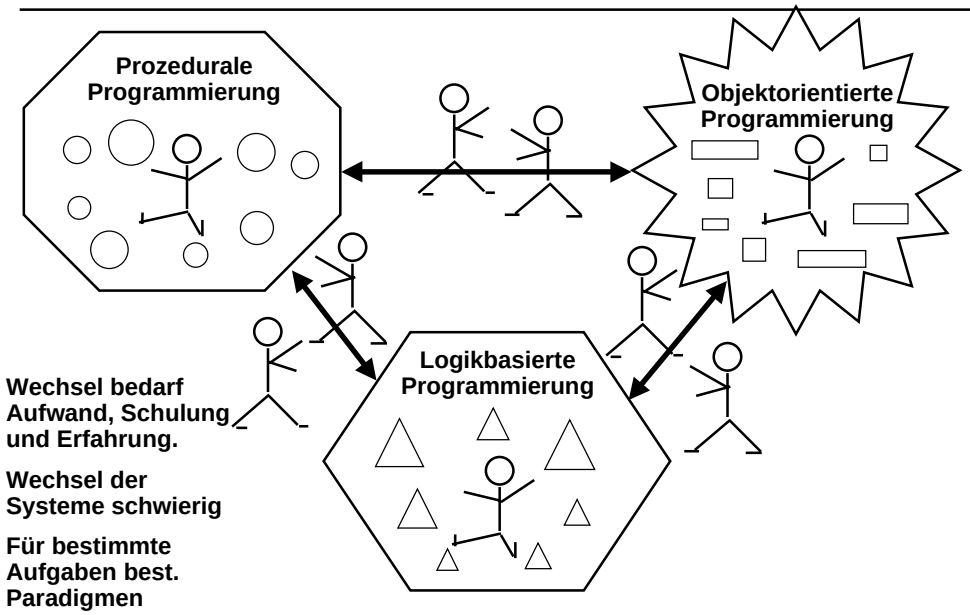
Programmier-Paradigmen

- Es gibt unterschiedliche Arten der Programmierung:



Verständnis der OO Programmierung

Wechsel des Programmier-Paradigmas



Wechsel bedarf Aufwand, Schulung und Erfahrung.

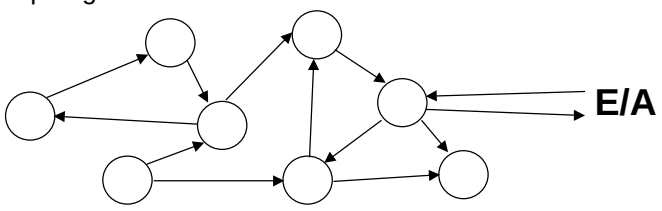
Wechsel der Systeme schwierig

Für bestimmte Aufgaben best. Paradigmen

Verständnis der OO Programmierung

Was ist ein rein objektorientiertes System

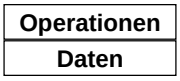
- **Ein Software-System besteht aus Objekten**
- **Jede Systemaktivität ist die Aktivität eines Objektes**
- **Objekte**
  - haben eine Lebensdauer (werden erzeugt und vernichtet)
  - sind identifizierbar
  - sind aktiv
  - verändern sich während ihrer Lebensdauer
  - senden und empfangen Nachrichten



Objekte

Objekte - die Dynamik des Systems

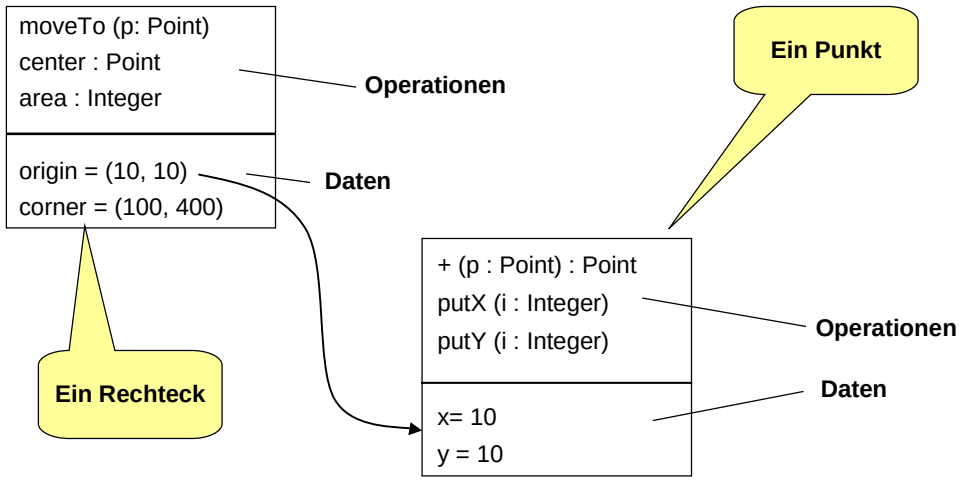
- **Ein Objekt ist eine Datenkapsel, die aus zwei Teilen besteht**
  - Der Wert der Daten repräsentiert den **Zustand** des Objekts
  - Daten können nur mithilfe von **Operationen** verändert werden (Kapselung)



- **Die Aktivität der Objekte ist die Ausführung ihrer Operationen**
  - Eine Operation wird ausgeführt, wenn ein Objekt eine entsprechende **Nachricht** erhält.
  - Der Objektzustand kann sich **verändern**.
  - Mögliche Operationen sind durch den **Objekttyp** bestimmt.
- **Ein Objekt ist ein Exemplar genau einer Klasse.**
- **Objekte existieren nur zur Laufzeit**
  - können aber auch persistent gespeichert werden

Objekte

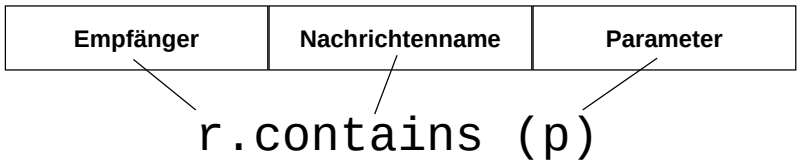
Beispiele für Objekte



Objekte

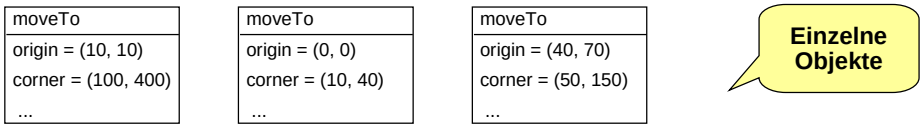
Nachrichten (Botschaften)

- **Objekte kommunizieren miteinander,**
  - dadurch daß sie **Nachrichten** versenden und Nachrichten empfangen.
- **Objekte reagieren auf Nachrichten,**
  - indem sie eine **Operation** (**Methode, Zugriffsunterprogramm**) ausführen.
- **Der Empfänger einer Nachricht (immer ein Objekt) ist verantwortlich für**
  - **Entschlüsselung** der Nachricht (verstehe ich die Nachricht?)
  - **Wirkung** (was tue ich?)



Klassen

Klassen - die Statik des Systems



- **Gleichartige Objekte werden zusammengefaßt**
  - und an einer Stelle beschrieben (Objekttyp oder Klasse)
- **Eine Klasse definiert für ihre Objekte**
  - die **Speicherstruktur**
    - Daten, Attribute, Exemplarvariablen
  - die **Operationen** (Methoden, Routinen),
    - von denen ein Teil exportiert wird (exported, public <-> private)
    - andere werden nur intern benötigt
- **Von einer Klasse können beliebig viele Objekte erzeugt werden**

Klassen

Beispiel : Klasse Point

```
class Point

feature
  x, y : Integer;

feature
  +: (p : Point) is
      result : Point;
    do
      result.x(x + p.x);
      result.y(x + p.y);
    end;
  x: (i : Integer) is
    do
      x := i;
    end;
  y: (i : Integer) is
    do
      y := i;
    end;
  ...
end -- class Rectangle
```

Exemplarvariablen (int. Verbundkomponente)

exportierte Operationen

Klassen

Beispiel : Klasse Rechteck

```
class Rectangle

feature
  origin, corner : Point;

feature
  moveTo: (p : Point) is
    do
      origin := origin + p;
      corner := corner + p;
    end;
  contains (p : Point) : Boolean is
    do ...
    end;

feature {NONE}
  extent : Point
    do
      -- liefert einen Punkt, der die Höhe und
      -- Weite des Rechtecks repräsentiert
    end;
end -- class Rectangle
```

Exemplarvariablen

exportierte Operationen

private Operationen

Klassen

Klasse - Objekt

- **Eine Klasse entspricht einem ADT**
- **Ein Objekt "entspricht " einem abstrakten Datenobjekt**
  - d.h. die Datenimplementierung ist verborgen.
- **Von einer Klasse sind außerhalb sichtbar:**
  - der Klassenname
  - die exportierten Operationen
- **Jedes Objekt ist ein Exemplar**
  - einer Klasse des Programms
- **Objekte einer Klasse kennen dieselben Operationen**
- **Objekte einer Klasse unterscheiden sich in den Werten ihrer Daten**

Klassen

Objekte und Klassen

- Nach außen sichtbar ist

```
class Rectangle
interface
  Create;
  origin : Point;
  corner : Point;
  moveTo (p : Point);
  contains (p : Point)
   Boolean:
end -- class Rectangle
```

Zusammenhang Objekt <-> Klasse

```
r : Rectangle;
p : Point

p.Create;
r.Create;

r.contains (p);
```

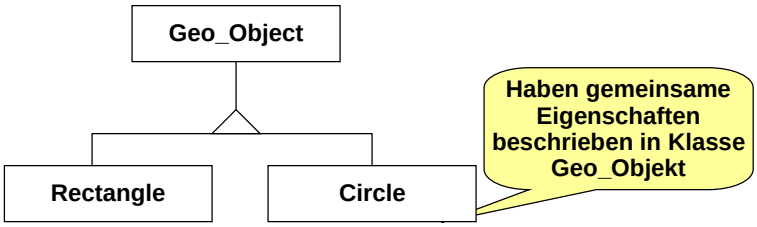
```
class Rectangle

feature
  origin, corner : Point;

feature
  moveTo: (p : Point) is
    do
      origin := origin + p;
      corner := corner + p;
    end;
  contains (p : Point) : Boolean is
    do
    end;
feature {NONE}

  extent : Point
    do
      -- liefert ...
    end;
end -- class Rectangle
```

Vererbung

Gemeinsame Eigenschaften, Spezialisierung

- **Gemeinsame Eigenschaften verschiedener Klassen K_1, K_2**
  - werden in einer **eigenen Klasse K** zusammengefaßt und definiert,
  - und anschließend an K_1, K_2 vererbt.

- **Beispiel**



- **Regel:**
  - Eine Klasse K_1 erbt von einer Klasse **K** genau dann, wenn K_1 eine **Spezialisierung** (Unterbegriff) von **K** ist. Umgekehrt wird **K** die **Generalisierung** genannt.

Vererbung

Beispiel 1: Einfachvererbung

```
class Geo_Object

feature
  moveTo: (p : Point) is
    deferred
    end;

  contains (p : Point) : Boolean
   is
    deferred
    end;
end -- class Geo_Object
```

```
class Rectangle

inherit
  Geo_Object
    define moveTo, contains

feature
  origin, corner : Point;
  ...
end -- class Rectangle
```

```
class Circle

inherit
  Geo_Object
    define moveTo, contains

feature
  center : Point;
  radius : Integer:
  ...
end -- class Circle
```

Abstrakte Klasse Spezifikationsklasse

Einfachvererbung

Vererbung

Beispiel 1: Mehrfachvererbung

```
class Geo_Object

feature
  moveTo: (p : Point) is
    deferred
    end;
contains (p : Point) : Boolean
   is
    deferred
    end;
end -- class Geo_Object
```

```
class Rectangle

inherit
  Geo_Object
    define moveTo, contains

feature
  origin, corner : Point;
  ...
end -- class Rectangle
```

```
class DisplayableObject

feature
  psDescription : String is
    deferred
    end;
end -- class DisplayableObject
```

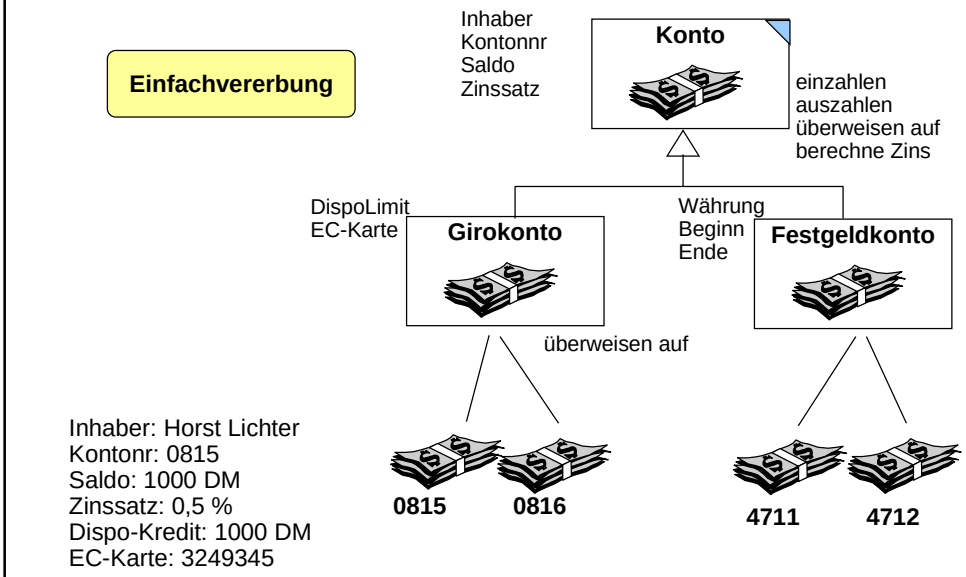
```
class DisplayableRectangle

inherit
  Rectangle

inherit
  DispayableObject
    define psDescription

feature
  ...
end -- class DisplayableRectangle
```

Mehrfachvererbung

Vererbung

Beispiel 2: Einfachvererbung

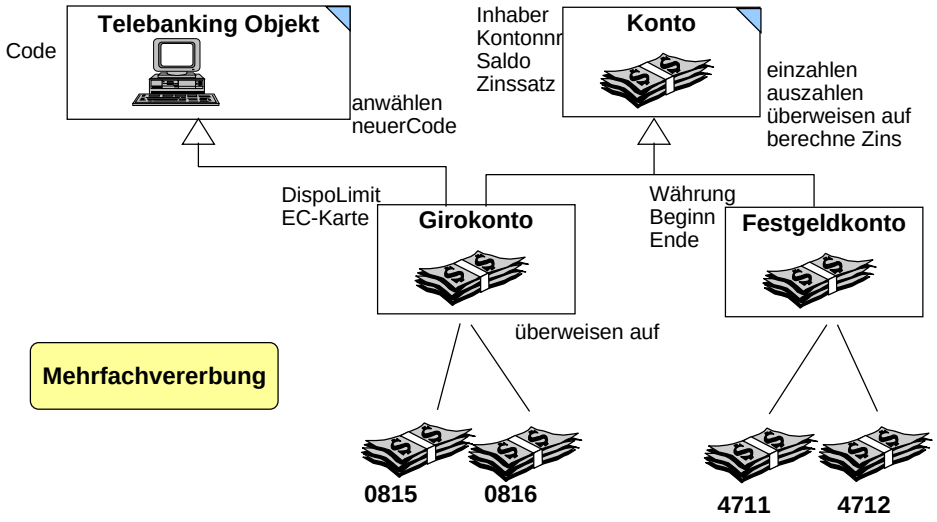
Einfachvererbung



Inhaber: Horst Lichter
Kontonr: 0815
Saldo: 1000 DM
Zinssatz: 0,5 %
Dispo-Kredit: 1000 DM
EC-Karte: 3249345

Vererbung

Beispiel 2: Mehrfachvererbung



Vererbung

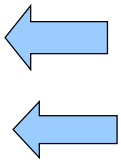
Abstrakte Klassen

- **Eine Klasse ist ein**
  - möglicherweise **nicht vollständig implementierter** ADT.
- **In konkreten Klassen (implementiert, effektiv)**
  - sind alle Operationen **ausführbar.**
- **In abstrakten Klassen (deferred, virtual)**
  - sind einige Operationen ***noch nicht implementiert*** , sondern nur spezifiziert (deklariert). Diese müssen in den konkreten Klassen angegeben werden.
- **Von konkreten Klassen**
  - können **beliebig viele** Objekte (Exemplare) erzeugt werden.
  - new- bzw. create- Nachrichten (Instanziierung)
- **Von abstrakten Klassen**
  - können **keine** Objekte erzeugt werden!

Vererbung

Beschreibungsschema von Klassen

- **Eine Klasse ist definiert durch**
- **ihren Namen**
- **ihre direkten Oberklassen**
  - die erbt_von-Beziehung muß zyklenfrei sein
- **eine Speicherstrukturbeschreibung**
  - erweitert die geerbten Beschreibungen
- **eine Menge von Operationsbeschreibungen**
  - erweitert die geerbten Beschreibungen



Vererbung

Oberklasse <-> Unterklasse

Geerbte Eigenschaften können auf drei Arten in einer Unterklasse modifiziert werden

Erweitern	etwas Neues hinzufügen
Redefinieren	sich ähnlich verhalten, Diff. formulieren
Definieren	etwas Versprochenes realisieren

Vererbung

Erweitern

```
class Geo_Object

feature
  moveTo: (p : Point) is
    deferred
    end;

  contains (p : Point) : Boolean
   is
    deferred
    end;
end -- class Geo_Object
```

```
class Rectangle

inherit
  Geo_Object
    define moveTo,
    contains

feature
  origin, corner :
   Point;

feature

  height : Integer is
    do ...
    end;
  width : Integer is
    do ...
    end;
  ...
end -- class Rectangle
```

Spezifische Eigenschaften werden hinzugefügt

Vererbung

Definieren

```
class Geo_Object

feature
  moveTo: (p : Point) is
    deferred
    end;

  contains (p : Point) : Boolean
   is
    deferred
    end;
end -- class Geo_Object
```

```
class Rectangle

inherit
  Geo_Object
    define moveTo, contains

feature
  origin, corner : Point;

feature
  moveTo: (p : Point) is
    do
      origin := origin + p;
      corner := corner + p;
    end;
  contains (p : Point) : Boolean
   is
    do
      ...
    end;
  ...
end -- class Rectangle
```

Definieren versprochener, aber noch nicht implementierter Eigenschaften

Vererbung

Redefinieren

```
class DisplayableObject
  feature
  psDescription: String is
    deferred
    end;
end -- class DisplayableObject
```

```
class Rectangle
...
feature
  initialize is do
      origin.Create; corner.Create;
      origin.initialize;
      corner.initialize;
    end
  ...
end -- class Rectangle
```

```
class DisplayableRectangle
inherit
  Rectangle
    rename initialize as initRect
    redefine initialize
...
feature
  color : Color;

feature
  initialize is do
    current.initRect;
    color.create; color.black;
  end;
...
end -- class DisplayableRectangle
```

Die ererbte Implementierung paßt im Kontext der Unterklasse nicht mehr.

Sie wird redefiniert!

Polymorphismus und dyn. Binden

Polymorphismus

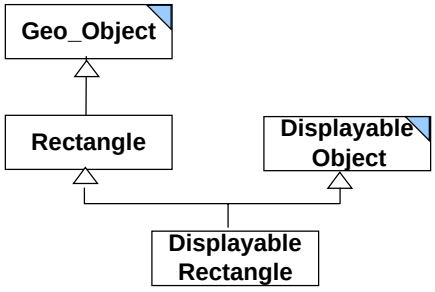
- **Allgemein:**
  - "Polymorphismus ist die Fähigkeit von Etwas, von verschiedener Gestalt zu sein"
- **In Programmiersprachen:**
  - Etwas: Variablen, Funktionen, Prozeduren
  - Gestalt: Typ
- **Eine polymorphe "Entität" kann in verschiedenen Kontexten verwendet werden,**
  - die unterschiedliche Typen verlangen.
- **Beispiele**

```
write (100, 3)
write ('a')
```

Polymorphismus und dyn. Binden

Beispiel: Polymorphismus - 1

- **Die Vererbung ist *ein* Mechanismus, um Polymorphismus in Programmiersprachen zu realisieren.**



ein DisplayableRectangle **verhält sich wie** ein Rechteck und wie ein DisplayableObject

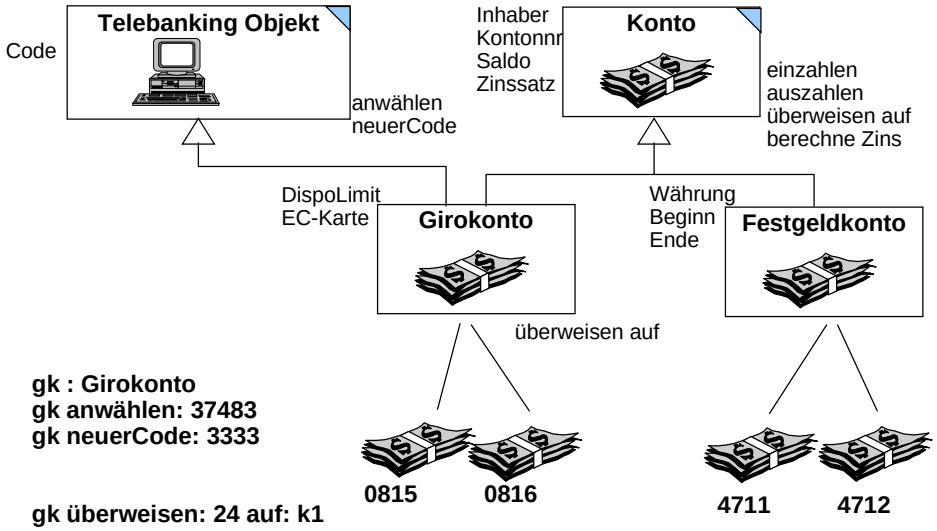
```
dr : DisplayableRectangle;
dr.Create;

dr.contains (p);
x := dr.center;
dr.moveTo (p);

s := dr.psDescription
```

Rectangle

DisplayableObject

Polymorphismus und dyn. Binden

Beispiel: Polymorphismus - 2



gk : Girokonto
gk anwählen: 37483
gk neuerCode: 3333

gk überweisen: 24 auf: k1

Polymorphismus und dyn. Binden

Dynamisches Binden: Beispiel 1

```
g : Geo_Object;
r : Rectangle;
c : Circle;

r.Create; c.Create

g := r;
g.contains (p);

g := c;
g.contains (p);
```

```
class Geo_Object

feature
  contains (p : Point) : Boolean is
    deferred
    end;
end -- class Geo_Object
```

```
class Rectangle

feature
   contains (p : Point) :Boolean is
     ...
end -- class Rectangle
```

```
class Circle

feature
  contains (p : Point) :Boolean is  ...
end -- class Circle
```

Dynamisches Binden heißt:

die **richtige** Implementierung zur **Laufzeit** finden

Polymorphismus und dyn. Binden

Dynamisches Binden: Beispiel 2

Kontoarten
- Festgeld
- Giro
- Spar
- Kredit

k : Konto;
k := system.waehleKonto;
...
k überweisen: 2000 auf: k1;

```
überweisen: betrag auf: empfKonto
   saldo := saldo - betrag.
   empfängerKonto einzahlen: betrag.
```
Konto

```
überweisen: betrag auf: empfKonto
   if (saldo - betrag) >= DispoLimit then
     ...
     super überweisen: betrag auf:empfKonto.
   else
     ...
```
Girokonto

```
überweisen: betrag auf: empfKonto
   if (Datum heute <= Ende) then
     ...
     super überweisen: betrag auf:empfKonto.
   else
     ...
```
Festgeldkonto

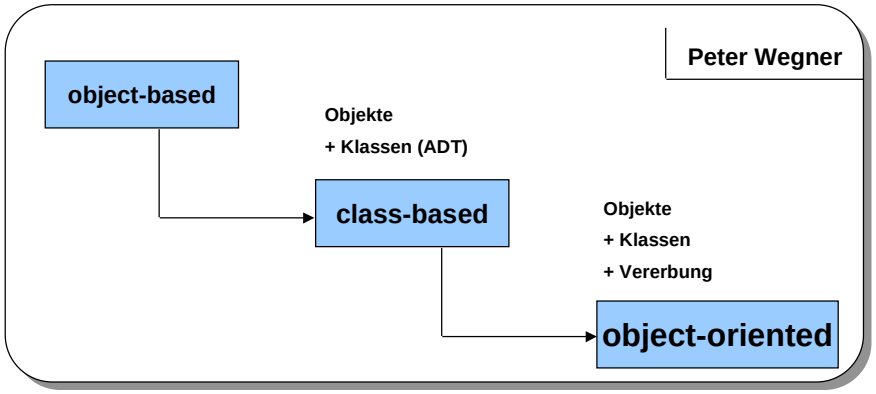
Diskussion

Varianten der Vererbung

Modifikations-möglichkeiten \ Anzahl der Oberklassen	eine oder keine	beliebig viele
nur Erweitern und Definieren	***strikte Einfachvererbung***	***strikte Mehrfachvererbung***
Erweitern Redefinieren Definieren	***nicht-strikte Einfachvererbung***	***nicht-strikte Mehrfachvererbung***

Verständnis der OO Programmierung

Merkmale objektorientierter Systeme/Sprachen



Ada
Modula-2

CLU
(Ada 83)
(Modula-2)

Simula
Smalltalk-80
Eiffel
C++
Ada 95
Java
Modula-3

Verständnis der OO Programmierung

Klassifikation

- **Objektbasiert**
  - Objektbasierte Programmiersprachen bieten die Möglichkeit, **Objekte** im Sinne einer **Datenkapsel** resp. eines **Objektmoduls** zu realsieren.
  - Jedes Objekt wir **einzeln** beschrieben und benutzt
- **Klassenbasiert**
  - Klassenbasierte Programmiersprachen bieten die Möglichkeit, Objekte in Forn von **Objekttypen (Klassen)** zu beschreiben.
  - Von Objekttypen können beliebig viele **Exemplare** (Objekte des Typs) erzeugt werden.
- **Objektorientiert**
  - Objektorientierte Programmiersprachen erlauben, Objekttypen (Klassen) mithilfe der **Vererbungs-Beziehung** zu strukturieren.
  - Dadurch lassen sich **Objekttyp-Hierarchien** im Sinne der Spezialisierung resp. Generalisierung modellieren.

Was haben wir gelernt!

- **Klassifikation**
  - objektbasiert, klassenbasiert
  - objektorientiert
- **Klassen sind (partiell) implementierte ADTs**
  - abstrakte Klassen
  - konkrete Klassen
- **Vererbung**
  - dient dazu, Spezialisierungsbeziehung zwischen Begriffen programmiertechnisch zu realisieren.
  - Unterschied zwischen Einfach- und Mehrfachvererbung
- **Polymorphismus**
  - kann mit Hilfe der Vererbung realisiert werden
  - führt zum dynamischen Binden von Implementierungen

Glossar

- **wissenbasierte, wertorientierte, prozedurale, objektorientierte Programmierung**
- **Objektmodul, ADT, Klasse**
- **Struktur eines objektorientierten Programms**
- **Nachrichten, Methoden, Laufzeitzustand eines objektorientierten Systems**
- **abstrakte Klasse, konkrete Klasse**
- **Definieren, Redefinieren, Erweitern von Methoden bei einem Spezialisierungsschritt**
- **Vererbung (Spezialisierung), Verallgemeinerung (Generalisierung)**
- **Einfach-, Mehrfachvererbung; strikte Vererbung, nichtstrikte Vererbung**
- **Polymorphismus, Polymorphismus durch Vererbung**
- **dynamisches Binden (Dispatching)**