# Kapitel 19

# Vererbung, UML

# Vererbung

- Neben der Datenabstraktion und der Datenkapselung ist die Vererbung ein weiteres Merkmal der OOP.
- Durch Vererbung werden die Methoden und die Eigenschaften einer Klasse (Basisklasse) an eine andere Klasse (abgeleitete Klasse) vererbt.
- Die abgeleitete Klasse verfügt neben ihren eigenen Methoden und Eigenschaften auch über die der Basisklasse.

# Vererbung

- Wenn eine Klasse B von einer Klasse A abgeleitet werden soll, so muss dies wie folgt geschrieben werden:

```
class B : public A
{
};
```

- Die Klasse B erbt die hinter dem Doppelpunkt mit *public* angegebene Klasse A.

# Vererbung

- Im folgenden Beispiel wird die Klasse CGiroKonto von der Basisklasse CKonto abgeleitet.

```
class CGiroKonto : public CKonto
{
    ...
};
```

=> Bsp. CGirokonto

# Zugriffsrecht *protected*

- Bisher haben wir die Zugriffsrechte *private* und *public* kennen gelernt.
- Das Zugriffsrecht *protected* wird bei der Vererbung wichtig.
- Auf *protected* Member einer Klasse kann nur von abgeleiteten Klassen zugegriffen werden.

# Zugriffsrechte auf Basisklassen

- Eine abgeleitete Klasse hat nur Zugriff auf die *public* und die *protected* Member der Basisklasse.
- Sie hat keinen Zugriff auf die *private* Member.

# Konstruktoren und Destruktoren von abgeleiteten Klassen

- Für das Anlegen eines Objekts einer abgeleiteten Klasse (*CGiroKonto*) ist der Konstruktor der abgeleiteten Klasse zuständig.
- Da die abgeleitete Klasse alle Eigenschaften der Basisklasse enthält, sind auch diese mit Anfangswerten zu versorgen.
- Hierfür wird der Konstruktor der Basisklasse (*CKonto*) aufgerufen.
- Ist kein Aufruf explizit angegeben, wird implizit der Standardkonstruktor der Basisklasse (*CKonto*) aufgerufen.

# Konstruktoren und Destruktoren von abgeleiteten Klassen

- Vor dem Konstruktor der abgeleiteten Klasse wird der Konstruktor der Basisklasse aufgerufen.
- Die Aufrufreihenfolge bei den Destruktoren ist umgekehrt: Zunächst wird der Destruktor der abgeleiteten Klasse und dann der Destruktor der Basisklasse aufgerufen.

# Konstruktoren und Destruktoren von abgeleiteten Klassen

- Vor dem Aufruf des Konstruktors:

```
CGiroKonto(string name, int nummer, double stand,
                    double dispo, double sollzins);
```

- Wird der Standardkonstruktor der Basisklasse *CKonto*:

```
CKonto::CKonto();
```

- implizit aufgerufen.

# Basisinitialisierer

- Im Konstruktor werden Zugriffsmethoden (z.B. SetName) der Klasse *CKonto* zum Initialisieren der Basisklassen-Eigenschaften verwendet:

```
CGiroKonto::CGiroKonto(string name, int nummer, double stand,
                                    double dispo, double sollzins)
{
    SetName(name);
    SetNummer(nummer);
    SetStand(stand);
    m_dispo = dispo;   m_sollzins = sollzins;
}
```

# Basisinitialisierer

- Da die Basisklasse CKonto einen Konstruktor mit Parametern besitzt, kann dieser aufgerufen werden.
- Hierfür verwendet man dann den Basisinitialisierer:

```
CGiroKonto::CGiroKonto(string name, int nummer, double stand,
                       double dispo, double sollzins)   :
                       CKonto(name, nummer, stand)
{
    m_dispo = dispo;   m_sollzins = sollzins;
}
```

# Basisinitialisierer

- Unter Verwendung von Elementinitialisierern kann man zusätzlich zum Basisinitialisierer auch schreiben:

```
CGiroKonto::CGiroKonto(string name, int nummer, double stand,
                       double dispo, double sollzins)   :
                       CKonto(name, nummer, stand) ,
                       m_dispo(dispo), m_sollzins(sollzins)
{
}
```

# Zuweisung zwischen abgeleiteten Klassen

- Ein Objekt einer abgeleiteten Klasse (*CGiroKonto*) kann wie ein Objekt der Basisklasse (*CKonto*) verwendet werden.

```
CKonto konto;
CGiroKonto giroKonto("Maier, Wilhelm", 47114711, 0.00, 1000.00, 14.24);
giroKonto.Ausgabe();
cout << endl;
konto = giroKonto;
konto.Ausgabe();
cout << endl;
```

# Zuweisung zwischen abgeleiteten Klassen

- Alle Eigenschaften des Objekts *konto* erhalten die Werte der Eigenschaften des Objekts *giroKonto*.
- Eine Umkehrung der Zuweisung ist nicht möglich.

# Mehrfache Vererbung

- Es ist möglich, dass eine Klasse mehrere Basisklassen hat, also von mehreren Klassen abgeleitet wird.
- Im folgenden Beispiel wird die Klasse *CPlusKonto* von den *CSparKonto* und *CGiroKonto* abgeleitet.

```
class CPlusKonto : public CSparKonto, public CGiroKonto
{
...
}
```

# Mehrere gleiche Basisklassen

- Im obigen Beispiel haben die beiden Basisklassen *CSparKonto* und *CGiroKonto* jeweils die gleiche Basisklasse *CKonto*. Man spricht dann von einer mehrfach indirekten Basisklasse.
- Hierdurch entstehen Mehrdeutigkeiten. Sowohl *CSparKonto* als auch *CGiroKonto* besitzen eine Methode *SetName()*.
- Diese Mehrdeutigkeiten können unter Verwendung des Gültigkeitsbereichsoperators :: aufgelöst werden.

# Virtuelle Basisklassen

- Das mehrfache Vorkommen der indirekten Basisklasse *CKonto* kann mit Verwendung virtueller Basisklassen vermieden werden.

```
class CGiroKonto : public virtual CKonto
class CSparKonto : public virtual CKonto
```

- Durch das Schlüsselwort virtual wird erreicht, dass ein Objekt einer mehrfach abgeleiteten Klasse die Elemente einer virtuellen Basisklasse (CKonto) nur einmal enthält.

# Unified Modeling Language (UML)

- UML ist eine Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme.
- Sie wird häufig in der objektorientierten Softwareentwicklung eingesetzt.
- In UML ist eine Vielzahl unterschiedlicher Diagrammtypen definiert.
- Im Folgenden wird auf die Klassendiagramme eingegangen.

# UML Klassendiagramme

- Eine Klasse wird durch ein Rechteck dargestellt.
- Dieses Rechteck wird mit horizontalen Linien in drei Rubriken unterteilt: Klassenname, Eigenschaften und Methoden.
- Neben dem Namen der Eigenschaften, können noch Angaben zu ihrem Typ und einem Initialwert gemacht werden.
- Neben dem Namen der Methoden, können zusätzlich die Parameter mit ihrem Typ angegeben werden.

# UML Klassendiagramme

- Die Klassen-Zugriffsrechte *private*, *public* und *protected* werden durch Voranstellen von -, + und # dargestellt:

| CPerson |
| --- |
| -m_vorname:string<br>-m_nachname:string |
| +SetVorname(string):void<br>+GetVorname():string<br>+SetNachname(string):void<br>+GetNachname():string<br>+Ausgabe():void<br>+Eingabe():void |

# UML Klassendiagramme, Vererbung

- Eine Vererbungsbeziehung wird durch einen nicht ausgefüllten Pfeil dargestellt.
- Der Pfeil zeigt immer auf die Basisklasse.

# UML Klassendiagramme, Vererbung