Objektorientiertes Programmieren mit C++ für Fortgeschrittene

Kapitel 3

3. Mehrfachvererbung

3.1. Eigenschaften und Problematik

3.2. Virtuelle Basisklassen

Mehrfachvererbung in C++ (1)

- **Eigenschaften**
  - ◇ **Mehrfachvererbung** liegt vor, wenn eine abgeleitete Klasse **mehr als eine direkte Basisklasse** hat, d.h. in ihrer Abstammungsliste zwei oder mehr Basisklassen aufgeführt sind.
  - ◇ **Beispiel :**

```
class Auto
{ public:
    void fahreWeiter(int d) { km+=d; }  // Fortbewegung um d km
    // ...
  protected:
    int km;                             // insgesamt gefahrene Kilometer
    // ...
};


class Boot
{ public:
    void fahreWeiter(int d) { sm+=d; }  // Fortbewegung um d sm
    // ...
  protected:
    int sm;                             // insgesamt gefahrene Seemeilen
    // ...
};


enum antrieb { reifen, schraube };


class AmphFahrz : public Auto, public Boot
{ public:
    void fahre(antrieb, int);
    // ...
  protected:
    antrieb aktAntr;                    // akt. Antrieb
    // ...
};


void AmphFahrz::fahre(antrieb a, int d)
{
  aktAntr=a;
  if (a==reifen)
     Auto::fahreWeiter(d);         // fahreWeiter(d) allein nicht zulässig
  else                             // da mehrdeutig
     Boot::fahreWeiter(d);
}
```

  - ◇ Bei Mehrfachvererbung **erbt** die **abgeleitete Klasse** die **Komponenten aller ihrer Basisklassen**.
    Die "***is a ..*** "-Beziehung gilt zwischen der abgeleiteten Klasse und **jeder ihrer Basisklassen**.
    → Ein **Objekt der abgeleiteten Klasse** kann zugleich auch als ein **Objekt von jeder der Basisklassen** betrachtet werden ("**sowohl** ... **als auch**...").
    Zu obigem Beispiel : Ein `AmphFahrz` ist sowohl ein `Auto` als auch ein `Boot`.

- **Namenskonflikte**
  - ◇ Die - jederzeit mögliche - Verwendung eines **gleichen Komponentennamens** in **verschiedenen Basisklassen** führt zu **Namenskonflikten**: in der **abgeleiteten Klasse** ist der **Komponentenname** allein **nicht mehr eindeutig**.
  - ◇ **Lösung:** Zur eindeutigen Auswahl einer bestimmten Komponente dieses Namens muß ihr **vollqualifizierter Name** angegeben werden.

Mehrfachvererbung in C++ (2)

• Teilobjekte der Basisklassen

◇ Ein **Objekt einer mehrfach abgeleiteten Klasse** besitzt - neben den neu definierten spezifischen Komponenten – **Teilobjekte von jeder ihrer Basisklassen**.

◇ Die Anordnungs-Reihenfolge der Teil-Objekte im Arbeitsspeicher ist implementierungsabhängig.
Typischerweise entspricht sie der Reihenfolge, in der die Basisklassen in der Abstammungsliste angegeben sind.
(Anmerkung : nur die Datenkomponenten belegen Speicherplatz im Objekt)

◇ **Beispiel** :

```
class Auto
{ // ...
};


class Boot
{ // ...
};


class AmphFahrz : public Auto, public Boot
{ // spezifische
  // Komponenten
};
```

Objekt der Klasse `AmphFahrz`

• (Teil-)Objektadressen

◇ Ein **Pointer** oder eine **Referenz** auf ein **Objekt der abgeleiteten Klasse** kann - **implizit oder explizit** - in einen **Pointer** oder eine **Referenz** auf ein **Objekt jeder ihrer Basisklassen umgewandelt** werden.
Der **umgewandelte Zeiger** (bzw die umgewandelte Referenz) **referiert** das **entsprechende Teil-Objekt** innerhalb des Gesamt-Objekts.
Die **Anfangsadressen der einzelnen Teilobjekte** sind aber **unterschiedlich**.
Das bedeutet, daß bei **Mehrfachableitung** sich durch die o.a. Typkonvertierung die **Objekt-Adresse ändern** kann.
Bei einfacher Ableitung bleibt die Adresse dagegen gleich.

◇ **Beispiel :**

```
#include <iostream>
using namespace std;

int main(void)
{ AmphFahrz a;
  Auto* ap=&a;         // implizite Typkonvertierung AmphFahrz* --> Auto*
  Boot* bp=&a;         // implizite Typkonvertierung AmphFahrz* --> Boot*

  cout << "\nAdresse als AmphFahrz : " << &a;
  cout << "\nAdresse als Auto      : " << ap;
  cout << "\nAdresse als Boot      : " << bp << '\n';
  return 0;
}
```

Erzeugte Ausgabe :

```
Adresse als AmphFahrz : 0012FF74
Adresse als Auto      : 0012FF74
Adresse als Boot      : 0012FF78
```

Mehrfachvererbung in C++ (3)

- **Mehrfachvererbung derselben Basisklasse**

  ◇ Es ist **nicht zulässig**, ein und dieselbe Basisklasse **mehrfach direkt zu erben**.

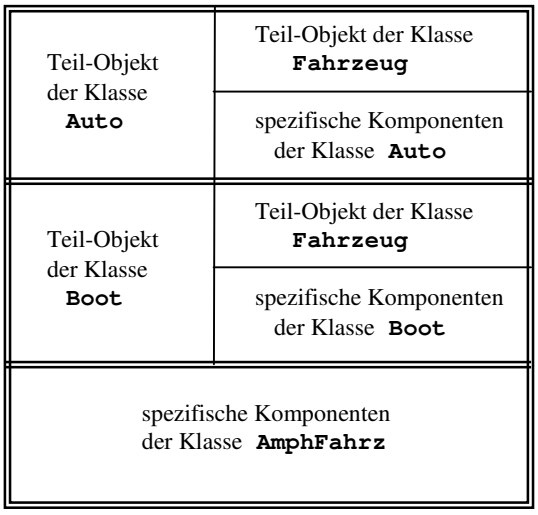
  Beispiel :
  ```
  class A { /* ... */ };
  class B : public A, public A     // unzulässig !
  { /* ... */ };
  ```

  ◇ Es ist jedoch möglich, ein und dieselbe Basisklasse über **unterschiedliche Ableitungswege** (mehrstufige Ableitungen) **mehrfach indirekt** zu erben.

  Beispiel :
  ```
  class Fahrzeug { protected : int baujahr; /* ... */ };
  class Auto : public Fahrzeug { /* ... */ };
  class Boot : public Fahrzeug { /* ... */ };
  class AmphFahrz : public Auto, public Boot { /* ... */ };
  // AmphFahrz erbt die Klasse Fahrzeug zweimal,
  // einmal über Auto und einmal über Boot.
  ```

  In einem derartigen Fall ist in einem Objekt der zuletzt abgeleiteten Klasse ein **Teil-Objekt** der **mehrfach geerbten indirekten Basisklasse** - mit allen seinen Daten-Komponenten - **mehrfach enthalten** (für jeden Ableitungsweg einmal).

  Zu obigen Beispiel : **Objekt der Klasse `AmphFahrz`**



  ◇ Für die **Komponenten** der **mehrfach geerbten Basisklasse** ergeben sich dadurch **Mehrdeutigkeitsprobleme**:
  Eine derartige Komponente kann weder über ihren Komponentennamen allein noch über ihren - den Basisklassennamen verwendenden - voll-qualifizierten Namen eindeutig angesprochen werden.
  Eindeutigkeit ist nur gegeben, wenn ein **voll-qualifizierter Name**, der sich auf eine der **Zwischenklassen** bezieht, benutzt wird.

  Zu obigen Beispiel :
  ```
  // In einer Member-Funktion der Klasse AmphFahrz :
  baujahr=1900;               // unzulässig
  Fahrzeug::baujahr=1900;     // unzulässig
  Auto::baujahr=1900;         // eindeutig --> zulässig
  ```

Virtuelle Basisklassen in C++ (1)

• Virtuelle Ableitung

Eine Basisklasse kann **virtuell abgeleitet** werden :
Ergänzung des Basisklassen-Eintrags in der Abstammungsliste um das Schlüsselwort **`virtual`** (vor oder nach dem Zugriffs-Specifier).
→ **virtuelle Basisklasse**

Beispiel :

```
class Fahrzeug { /* ... */ };
class Auto : virtual public Fahrzeug { /* ... */ };
```

• Aufbau eines Objekts mit einer virtuellen Basisklasse

◇ Ein Objekt einer Klasse mit einer virtuellen Basisklasse besitzt einen **anderen Aufbau** als ein entsprechendes Objekt mit nicht-virtueller Basisklasse :
Es enthält - im Unterschied zu einer nicht-virtuellen Ableitung - einen **Pointer auf das Teil-Objekt der Basisklasse**. Dieses wird üblicherweise **nach** den **spezifischen Komponenten der abgeleiteten Klasse** angelegt.

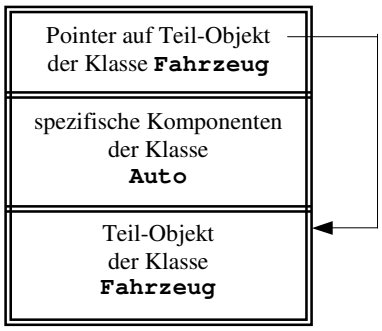
Beispiel :

nicht-virtuelle Ableitung

```
class Fahrzeug { /* ... */ };
class Auto : public Fahrzeug
{ /* ... */ };
```

virtuelle Ableitung

```
class Fahrzeug { /* ... */ };
class Auto : virtual public Fahrzeug
{ /* ... */ };
```



◇ Von einer virtuellen Basisklasse ist bei **indirekter Mehrfachvererbung** in abgeleiteten Klassen immer **nur ein Teil-Objekt** vorhanden
(Voraussetzung : **alle Ableitungen** der betreffenden Klasse müssen **virtuell** sein)

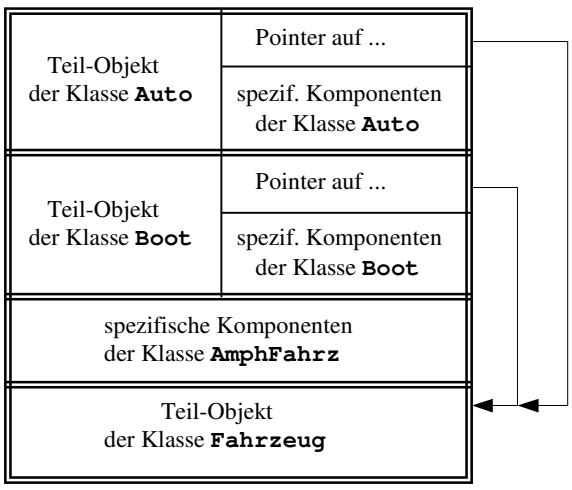
Beispiel :

```
class Fahrzeug
{ /* ... */ };

class Auto :
virtual public Fahrzeug
{ /* ... */ };

class Boot :
virtual public Fahrzeug
{ /* ... */ };

class AmphFahrz :
public Auto, public Boot
{ /* ... */ };
```



Virtuelle Basisklassen in C++ (2)

- Für die **Komponenten** einer **mehrfach geerbten virtuellen Basisklasse** existieren **keine Mehrdeutigkeitsprobleme**.
  → Sie können **sowohl** über ihren **Komponentennamen allein** als auch über ihren - den Basisklassennamen verwendenden - **voll-qualifizierten Namen** eindeutig angesprochen werden.

  Zu obigen Beispiel :

```
// In einer Member-Funktion der Klasse AmphFahrz
baujahr=1900;             // eindeutig --> zulässig
Fahrzeug::baujahr=1900;   // eindeutig --> zulässig
Auto::baujahr=1900;       // eindeutig --> zulässig
```

- **Initialisierung von Objekten einer Klasse mit mehrfach indirekt abgeleiteten virtuellen Basisklassen :**

  ◇ Da jedes (Teil-) Objekt nur einmal initialisiert werden darf (Konstruktoraufruf), ergeben sich Änderungen in der **Initialisierungs-Reihenfolge** gegenüber der indirekten Mehrfachableitung nicht-virtueller Basisklassen :
  - ▻ Die **Konstruktoren virtueller Basisklassen** werden **immer zuerst** ausgeführt (pro Klasse ein Konstruktoraufruf !).
  - ▻ Anschließend werden die Konstruktoren der übrigen direkten bzw indirekten Basisklassen in der Reihenfolge der Abstammungsliste bzw der Ableitung aufgerufen.
  - ▻ Zuletzt wird der Konstruktor der zuletzt abgeleiteten Klasse (der Klasse des anzulegenden Objekts) ausgeführt.

  Beispiel :

```
class Besitz { /* ... */ };
class Fahrzeug { /* ... */ };
class Auto : virtual public Fahrzeug { /* ... */ };
class Boot : virtual public Fahrzeug { /* ... */ };
class AmphFahrz : public Besitz, public Auto, public Boot
{ /* ... */ };
```

  Reihenfolge der Konstruktoraufrufe für ein Objekt von `Amphfahrz` :

```
Konstruktor von Fahrzeug
Konstruktor von Besitz
Konstruktor von Auto
Konstruktor von Boot
Konstruktor von AmphFahrz
```

  Aufruf-Reihenfolge bei nicht-virtueller Basisklasse Fahrzeug :

```
Konstruktor von Besitz
Konstruktor von Fahrzeug
Konstruktor von Auto
Konstruktor von Fahrzeug
Konstruktor von Boot
Konstruktor von AmphFahrz
```

  ◇ Vom **Konstruktor** einer **virtuellen Basisklasse benötigte Initialisierungswerte** müssen diesem **direkt vom Konstruktor** der **zuletzt abgeleiteten Klasse übergeben** werden.
  Eine eventuelle Übergabe durch Zwischenklassen wird ignoriert. Eine **stufenweise Übergabe über Zwischenklassen** (wie bei nicht-virtuellen Basisklassen erforderlich) ist daher **nicht möglich**.
  Dies kann bei der Ableitung von Klassen, bei denen das Vorhandensein virtueller Basisklassen nicht bekannt ist (z.B. nicht selbst geschriebene Klassen), zu **Problemen** führen.