„Was du ererbt von
Deinen Vätern hast,
erwirb es,
um es zu besitzen.“

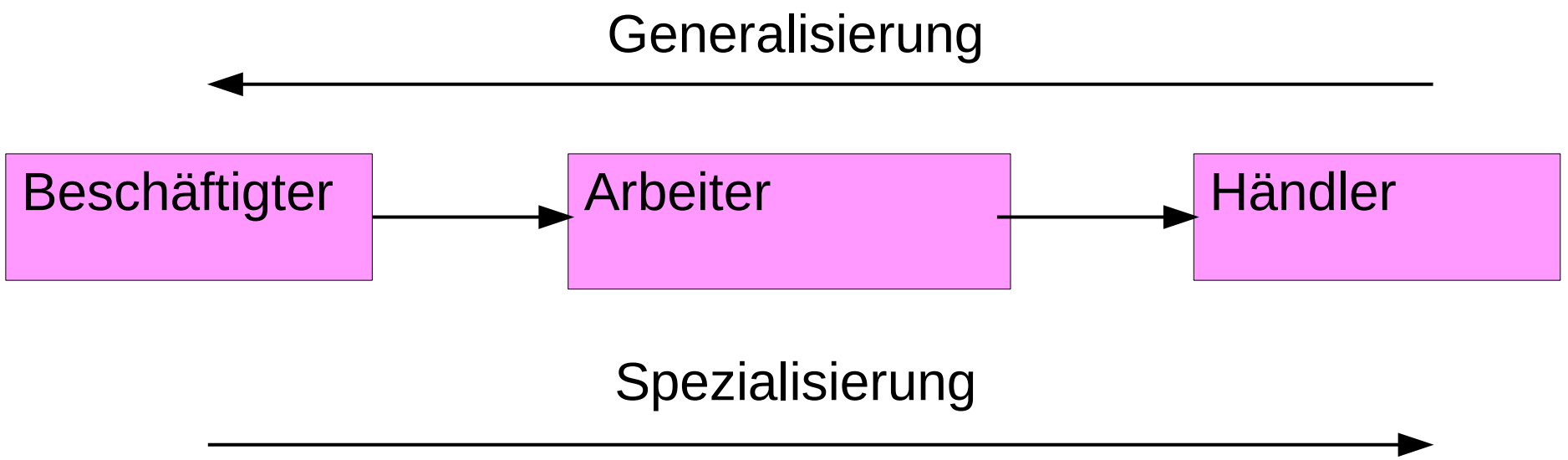
Quelle: http://www.neue-deutsche-monarchie.de/texte/orient/Herder_Goethe_und_der_Islam_I.php

Vererbung

- Worum geht es?
- Es wird etwas, was bereits existiert, weitergegeben. In der Programmierung sind es die Member einer Klasse.
- Die Klasse, die etwas vererbt, nennt man Basisklasse, die erbende Klasse abgeleitete Klasse.
- Auch die Begriffe Eltern- und Kindklasse sind üblich.
- Vererbung kann mehrfach hintereinander geschehen, bei Vererbung in einem Schritt spricht man auch von direkter Basis- bzw. direkt abgeleiteter Klasse.

Vererbung

- Vererbung ist mit einer Spezialisierung verbunden.
- Durch Hinzufügen oder Modifizieren von Membern wird eine Klasse spezialisiert.



Vererbung

unsere Basisklasse

```
#ifndef _BESCH_H_
#define _BESCH_H_
#include "cstrg.h"
class Besch
{
 private:
  CString Name;
 public:
  Besch( CString BeschName):Name(BeschName){};
  CString getName() const{return Name;}
  void setName(CString Name){this->Name=Name;};
  void display(ostream& out)const  {out<<this->Name;}

  friend ostream &operator<<(ostream& os, const Besch& b);
};
#endif
```

Vererbung

erste Kind- (abgeleitete) klasse

```
#ifndef _ARBEITER_H_
#define _ARBEITER_H_
#include "besch.h"
class Arbeiter : public Besch
{
 public:
  Arbeiter(CString BName, double stdLohn);
  void setLohn(float stdLohn){lohn=stdLohn;};
  void setStd (float stunden){this->stunden=stunden;}
  float getLohn(){return lohn;}
  float getStrunden(){return stunden;}
  void display(ostream &out)const;
  friend ostream &operator<< (ostream& os, const Arbeiter&b);
 private:
  float lohn, stunden;
};
#endif
```

Vererbung

erste Kind- (abgeleitete) klasse

- Die Klasse Arbeiter erbt von der Klasse Besch
- Sie übernimmt alle Member der Basisklasse (hier Besch) unabhängig davon, ob sie sichtbar sind oder nicht.
- Private Member der Basisklassse sind in der abgeleiteten Klasse vorhanden, aber nicht sichtbar.
- In der Klasse Arbeiter werden neue Member hinzugefügt.
- Aus dem allgmeingehaltenen Beschäftigten wird nun ein Arbeiter, der arbeitsstundenabhängig Lohn erhält.
- Wird ein Objekt einer abgeleiteten Klasse erzeugt, so wird zunächst der Defaultconstructor der Basisklasse ausgeführt.

Vererbung

erste Kind- (abgeleitete) klasse

- Im Constructor der abgeleiteten Klasse können die Member der Basisklasse über set-Methoden eingestellt werden.
- Eine bessere Methode besteht in der Anwendung des Basisinitialisierers, einer Konstruktion, die dem Memberinitialisierer ähnelt.
- Er wird durch einen Doppelpunkt abgetrennt hinter der Parameterliste der Constructorimplementation durch Angabe des Namens der Basisklasse, gefolgt von der Parameterliste, angegeben (vergl. Memberinitialisierer).

```
Arbeiter(CString BName, double StdLohn)
          :Besch(BName)
{Stunden=0;Lohn=StdLohn;}
```

Vererbung

erste Kind- (abgeleitete) klasse

- In der abgeleiteten Klasse können Funktionen der Basisklasse redefiniert werden. Sie haben den selben Namen und die selben Parameter, wie die originale Funktion in der Basisklasse. Man nennt sie **überschriebene Funktionen**.
- Im Beispiel betrifft dies die Funktion display().
- Für den Funktionsaufruf gelten folgende Regeln:
  - **Bei Aufruf der Funktion über das eigene Objekt und den Punktoperator oder über einen Pointer der eigenen Klasse wird die Funktion der eigenen Klasse ausgeführt.**
  - **Bei Aufruf über einen Pointer der Basisklasse, wird die Funktion der Basisklasse ausgeführt.**

Vererbung

erste Kind- (abgeleitete) klasse

```
...
#include "besch.h"
#include "arbeiter.h"
int main()
{
  Arbeiter a1("Hans Huckebein",9.25);
  Arbeiter*p1=&a1;
  Besch   *b1=&a1;

  cout<<"a1.display:  ";
  a1.display(cout);

  cout<<"p1->display: ";
  p1->display(cout);

  cout<<"b1->display: ";
  b1->display(cout);
  cout<<endl;
}
```

```
beck@U330:~/LVSS2017/CPP/vorb/1$ ./a.out
a1.display:  Hans HuckebeinStdLohn: 9.25 Stunden: 0
p1->display: Hans HuckebeinStdLohn: 9.25 Stunden: 0
b1->display: Hans Huckebein
beck@U330:~/LVSS2017/CPP/vorb/1$
```

Aufruf über Pointer der Basisklasse Besch

Vererbung

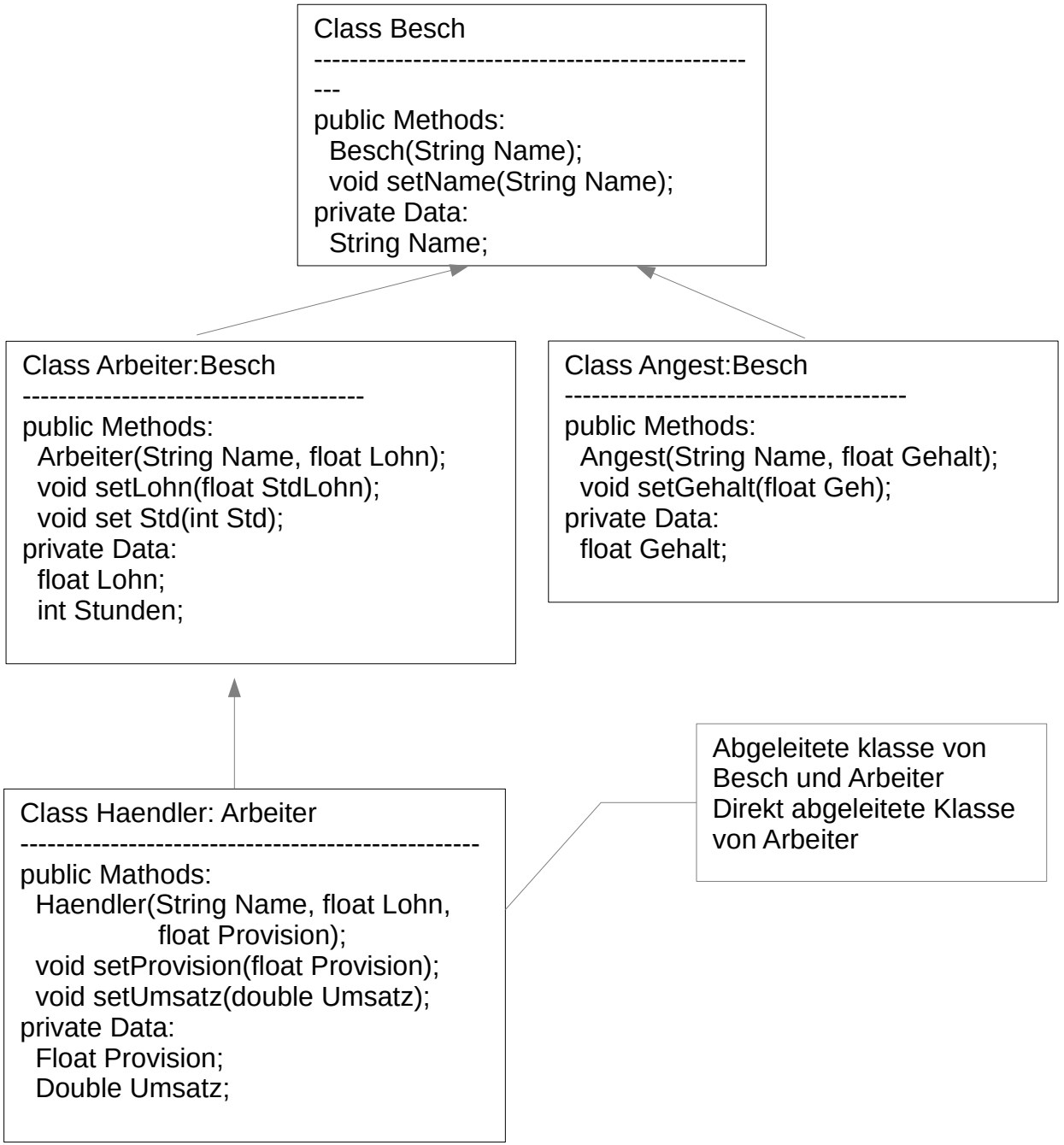
- Eine weitere abgeleitete Klasse könnte einen Händler modellieren, der zusätzlich zu seinem Stundenlohn noch eine Provision bezüglich des Umsatzes erhält.

```
class Haendler : public Arbeiter
{
 public:
  Haendler(CString Name,double lohn,double prov);
  void setProv (double prov){provision=prov;}
  double getProvision()const {return provision;}
  void setUmsatz (double ums){umsatz=ums;}
  double getUmsatz()const {return umsatz;}
  void display(ostream& out)const;
  friend ostream& operator<<(ostream & os,const Haendler &b);
  ~Haendler(){}
 private:
  double provision, umsatz;
};
```

Vererbung

- Eine weitere abgeleitete Klasse könnte einen Angestellten modellieren, der ein Festgehalt bezieht.

```
class Angest : public Besch
{
 public:
  Angest(CString BName, double gehalt);
  void setGehalt(double gehalt){this->gehalt=gehalt;};
  double getGehalt(){return gehalt;}
  void display(ostream & out)const;
  friend ostream& operator<<(ostream & os, const Angest &b);
 private:
  double gehalt;
};
```


Class Besch
public Methods:
Besch(String Name);
void setName(String Name);
private Data:
String Name;
Class Arbeiter:Besch
public Methods:
Arbeiter(String Name, float Lohn);
void setLohn(float StdLohn);
void set Std(int Std);
private Data:
float Lohn;
int Stunden;
Class Angest:Besch
public Methods:
Angest(String Name, float Gehalt);
void setGehalt(float Geh);
private Data:
float Gehalt;
Class Haendler: Arbeiter
public Mathods:
Haendler(String Name, float Lohn,
float Provision);
void setProvision(float Provision);
void setUmsatz(double Umsatz);
private Data:
Float Provision;
Double Umsatz;
Abgeleitete klasse von Besch und Arbeiter
Direkt abgeleitete Klasse von Arbeiter

Vererbung

- Im nächsten Schritt werden alle Klassen durch eine Funktion calc erweitert.
- Die Funktion calc berechne für einen jeden Beschäftigten (Arbeiter, Angestellter, Händler) den auszuzahlenden Lohn.
- Die Funktionen errechnen aus den jeweiligen Memberdaten den Betrag und geben ihn als Returnwert zurück.
- Die Funktionen modifizieren keine Membervariablen und können somit als const -funktion realisiert werden.
- Es handelt sich um überschriebene Funktionen.

Vererbung

```
double Besch::calc()const{return 0.0;}
```

```
double Angest::calc()const{return Gehalt;}
```

```
double Arbeiter::calc()const{return Lohn*Stdunden;}
```

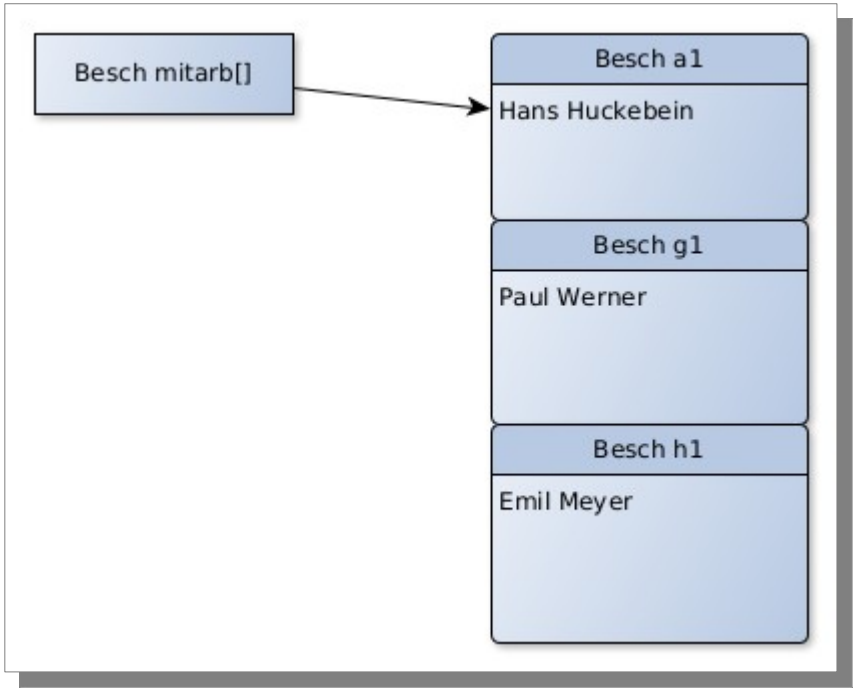
```
double Haendler::calc()const
{Provision*Umsatz+Arbeiter::calc();}
```

- Es kann nun für jeden Typ Beschäftigter das auszuzahlende Geld berechnet werden.

Vererbung

- Fassen wir jedoch die gesamte Belegschaft in einem Array zum Basistyp Besch zusammen, erhalten wir beim Iterieren über dieses Array nur 0.0 für alle Beschäftigte, weil nur die calc-funktion von Besch ausgeführt wird.
- Außerdem werden die Objekte zu Objekten des Typs Besch reduziert.
- Somit ergibt calc immer 0.0.

```
Besch mitarb[]={a1,h1,g1};
```



Vererbung

- Fassen wir jedoch die gesamte Belegschaft in einem Array von Pointern des Basistyp Besch zusammen, bleiben die Objekte vollumfänglich erhalten. Beim Iterieren über dieses Array erhalten wir aber wieder nur 0.0 für alle Beschäftigte, weil nur die calc-funktion von Besch ausgeführt wird.
- Abhilfe schaffen hier virtuelle Funktionen.
- Die Funktionen calc und display können in der Klasse, in der sie erstmalig definiert werden, mit dem Schlüsselwort **virtual** gekennzeichnet werden.
- Bei virtuellen Funktionen bestimmt nicht der Typ des Pointers auf ein Objekt die auszuführende Funktion, sondern das Objekt selbst.
- Somit ergibt sich nachfolgende Basisklassse Besch:

Vererbung

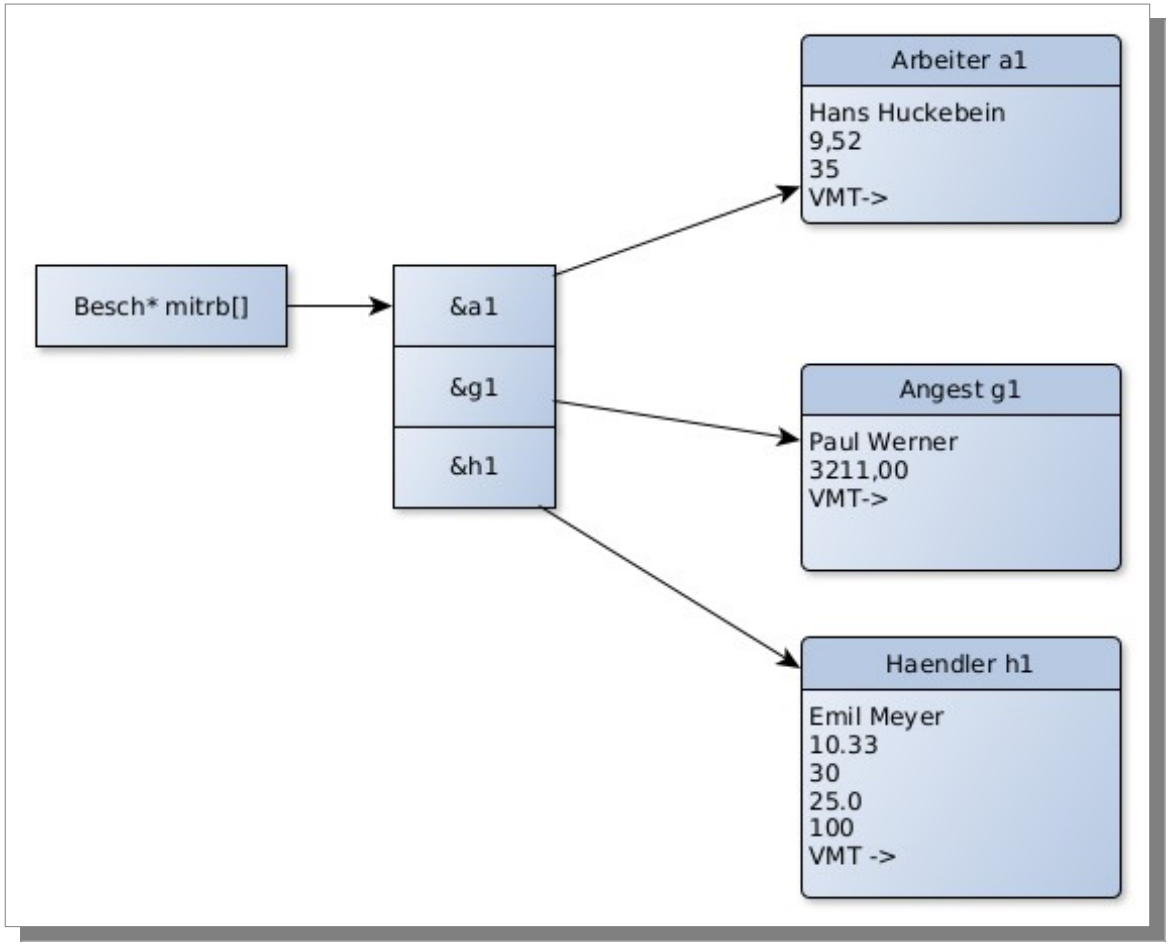
```
#ifndef _BESCH_H_
#define _BESCH_H_
#include "cstrg.h"
class Besch
{
  public:
    Besch( CString BeschName):Name(BeschName){};
    CString getName() const{return Name;}
    void setName(CString Name){this->Name=Name;};
    virtual void display()const;
    virtual double calc() const{return 0.0};
  private:
    CString Name;
};
#endif
```

Vererbung

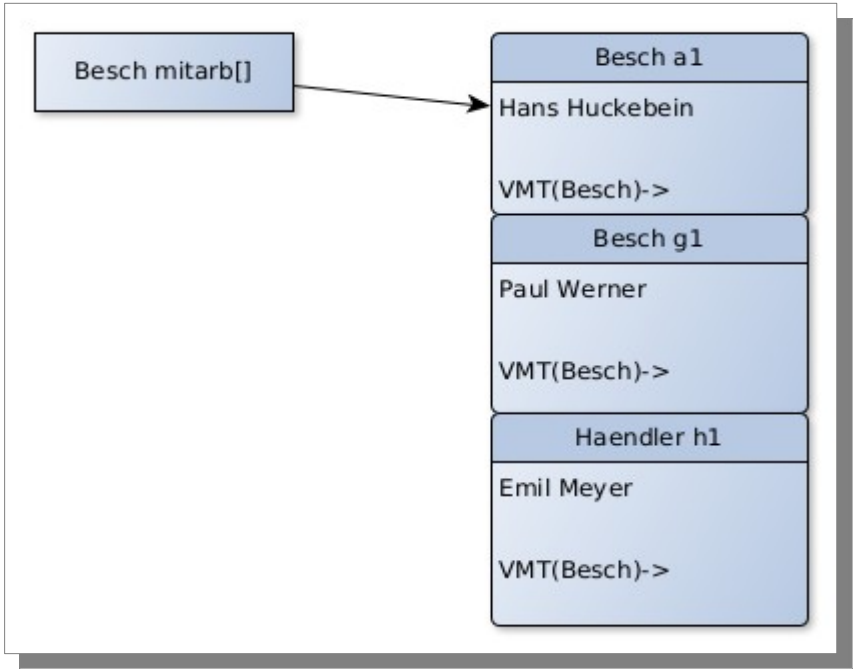
```
Arbeiter a1("Hans Huckebein",9.25);
Haendler h1("Emil Meyer", 10.33,25.0);
Angest   g1("Paul Werner", 3211.00);

Besch* mitarb[]={&a1,&h1,&g1};


for (int i=0; i<3; i++)
{
   mitarb[i]->display(cout);
   cout<<": "<<mitarb[i]->calc();
   cout<<endl;
}
```

Vererbung

Array von Pointern auf verschiedene Objekte der Basisklasse Besch.
VMT: VirtualMethodTable (pointer auf eine Tabell mit Funktionspointern auf die virtuellen Funktionen)



Array von Objekten der Basisklasse Besch.
Hier werden die Objekte auf die Attribute (Member) von Besch reduziert!



Vererbung und Destructor

Hat eine Klasse virtuelle Funktionen, so muss die Klasse auch einen virtuellen Destructor, der leer sein kann, enthalten.

Der virtuelle Destructor sorgt für den Aufruf des Destructors der konkreten Klasse, der auch leer sein kann.

Dieser wiederum sorgt für den Destructoraufruf von Memberobjekten, wie hier des Stringobjektes

Virtueller Ausgabeoperator?

- Der Ausgabeoperator kann nur als externe Funktion (friend) realisiert werden.
- Er kann deshalb kein Attribut virtual tragen, da die Funktion ja kein Member der Klasse ist.
- Die Display-Funktion kann aber sehr wohl virtuell sein.
- Ein Ausgabeoperator in Besch, der die (virtuelle) Funktion display aufruft, schafft Abhilfe.

Virtueller Ausgabeoperator?

```
class Besch
{
  public:
    Besch( CString BeschName):Name(BeschName){};
    CString getName() const{return Name;}
    void setName(CString Name){this->Name=Name;};
    virtual void display(ostream& out)const
                    {out<<this->Name;}
   virtual double calc()const =0;// {return 0.0;}
   friend ostream & operator<<(ostream& os, Besch&b);
  private:
    CString Name;
};
```

Rein virtuelle Funktionen

- Gibt es für eine virtuelle Methode in einer Basisklasse keine sinnvolle Implementierung, kann man an Stelle des Bodys auch =0; oder const =0; angeben.
- Man spricht von einer rein virtuellen Funktion.
- Hat eine Klasse rein vituelle Funktionen, so wird sie zu einer abstrakten Klasse, man kann von ihr keine Instanzen erzeugen, sie kann nur Basisklasse sein.
- In einer abgeleiteten Klasse werden dann die abstrakten Funktionen implementiert.

Rein virtuelle Funktionen

```
class Besch
{
  public:
   Besch( CString BeschName):Name(BeschName){};
   CString getName() const{return Name;}
   void setName(CString Name){this->Name=Name;};
   virtual void display(ostream& out)const =0;
   virtual double calc()const =0;
   friend ostream & operator<<(ostream& os, Besch&b);
  private:
    CString Name;
};
```

Durchsichtigkeit

- Bislang wurde die Basisklasse grundsätzlich mit **`:public baseclass`** angegeben.
- Dies bedeutet, dass die public Member der Basisklasse auch public bezüglich der abgeleiteten Klasse sind.
- Man spricht von Durchsichtigkeit.
- Im Gegensatz dazu gibt es die private - vererbung **`:private baseclass.`**
- Hier werden die public Member der Basisklasse zu private Member der abgeleiteten Klasse. Sie sind in dieser Klasse zu sehen, in einer weiter abgeleiteten Klasse jedoch nicht. Man spricht von Undurchsich-tigkeit der Klasse.

Sichtbarkeitsattribut protected

- Neben public und private gibt es noch das Sichtbarkeitsattribut protected.
- Protected Member sind nach außen nicht sichbar, wohl aber in abgeleiteten Klassen.
- Protected sollte sparsam verwendet werden, da zusätzliche Abhängigkeiten zwischen Klassen entsehen.

Merfachvererbung

- In C++ kann eine Klasse von mehreren Basisklassen erben.
- Dies führt u.Ust. zu Problemen, wenn die Basisklassen eine gemeinsame Basisklasse haben.
- Member sind dann ggf. doppelt in den resultierenden Objekten.
- Abhilfe schafft hier virtuelle Vererbung.


Class Besch
public Methods:
Besch(String Name);
void setName(String Name);
private Data:
String Name;
Class Lohnempf:Besch
public Methods:
Arbeiter(String Name, float Lohn);
void setLohn(float StdLohn);
void set Std(int Std);
private Data:
float Lohn;
int Stunden;
Class Angest:Besch
public Methods:
Angest(String Name, float Gehalt);
void setGehalt(float Geh);
private Data:
float Gehalt;
Class Haendler: Lohnempf
public Methods:
Haendler(String Name, float Lohn,
float Provision);
void setProvision(float Provision);
void setUmsatz(double Umsatz);
private Data:
Float Provision;
Double Umsatz;
Mehrfachvererbung!
Class Manager: Haendler, Angest
public Methods:
Manager(. . . . . .);
. . .
private Data: