Vererbung (inheritance)

- **Ziel** der **Vererbung** ist es, bereits definierte Klassen für die Definition neuer Klassen zu nutzen. (Erhöhung der **Wiederverwendbarkeit**, **Softwaresicherheit**, **Effizienz**, **Erweiterbarkeit**)

- Eine abgeleitete Klasse "**verfeinert**" bzw. "**spezialisiert**" die Basisklasse(n), indem weitere Methoden und Member hinzugefügt und "**geerbte**" Namen "**überschrieben**" werden können.

- Alternativ zur **Vererbung** kann die **Einbettung** verwendet werden, die eine **Aggregation** (Zeiger bzw. Referenzen von Objekten) bzw. **Komposition** (Objekte) realisiert.

- Öffentliche Vererbung bedeutet "**ist ein**", d.h. wenn **class student** öffentlich (**public**) von **class person** abgeleitet ist, dann "**ist ein student-Objekt ein person-Objekt**" (vgl. **person_s**).

- Genau dann wenn eine Klasse **B** von einer Klasse **A** öffentlich **abgeleitet** ist, **ist** jedes Objekt vom Typ **B auch ein** Objekt vom Typ **A**, jedoch **nicht umgekehrt**.

- Genau dann wenn eine Klasse **B** von einer Klasse **A** öffentlich **abgeleitet** ist, gilt alles, was für ein Objekt **A** gilt, auch für ein Objekt **B**, jedoch **nicht umgekehrt**.

- Genau dann wenn eine Klasse **B** von einer Klasse **A** öffentlich **abgeleitet** ist, repräsentiert **A ein allgemeineres Konzept als B** und **B ein spezielleres Konzept als A**.

Vererbung (inheritance)

- Genau dann wenn eine Klasse **B** von einer Klasse **A** öffentlich **abgeleitet** ist, kann überall dort, wo ein Objekt vom Typ **A** verwendet werden kann, auch ein Objekt vom Typ **B** verwendet werden.
- Letzte Aussage gilt u.a bei **Initialisierungen**, **Zuweisungen** und **Parameterersetzungen** von Funktionen und bei der **Rückgabe** von Funktionswerten.

- Wenn dort, wo ein Objekt einer **abgeleiteten Klasse** vorgesehen ist, ein Objekt einer hierzu gehörenden **Basisklasse** verwendet wird, wird ein **Compile-Error** generiert.

- Zusatz **public** bei der Vererbung besagt, dass die **Zugriffsrechte** der Member und Methoden der jeweiligen Basisklasse unverändert für die abgeleitete Klasse übernommen werden.

- Eine **abgeleitete Klasse** (Unterklasse, Subklasse) kann **mehrere Basisklassen** (Oberklassen, Superklassen) besitzen, hier spricht man von **mehrfacher Vererbung** (**multiple inheritance**).

- In abgeleiteten Klassen können **gleichnamige Member und Methoden überschrieben** werden, in der **abgeleiteten Klasse** existieren die gleichnamigen Member und Methoden **mehrfach**.

- **Überschriebene** Member und Methoden sind bzgl. ihres **Typs** und ihrer **Parameter unabhängig** (frei wählbar) von den gleichnamigen Membern und Methoden der Basisklassen.

Vererbung (inheritance)

- Ein Zugriff auf **gleichnamige Member oder Methoden der Basisklassen** gelingt seitens der Methoden der **abgeleiteten Klasse** mittels **Cast** bzgl. der **Basisklassennamen** oder mittels der Notation **basisklassenname :: methode** bzw. **basisklassenname :: member**, auch **using basisklassenname :: methode** bzw. **using basisklassenname :: member** ist formulierbar.

- Ein Zugriff auf **[gleichnamige] Member** oder **Methoden** der **Basisklassen** ist nur für **public** - und **protected** - Member bzw. Methoden der Basisklasse möglich, **nicht** jedoch für **private** - Member bzw. - Methoden der Basisklasse (vgl. **inh0**).

- Der Zugriff auf **protected** - Member bzw. Methoden der Basisklasse seitens der abgeleiteten Klasse ist uneingeschränkt möglich. Über Objekte der abgeleiteten Klasse sind **protected** - vererbte Member bzw. Methoden jedoch **nicht zugreifbar** (vgl. **inh0p**).

- **protected** wirkt wie **friend** für **private** - Member bzw. - Methoden, jedoch muß für jede abgeleitete Klasse eine **friend** - Deklaration in der Basisklasse angegeben werden (nachteilig) (vgl. **inh0f**).

- Eine abgeleitete Klasse hat **alle Eigenschaften** (Member und Methoden) ihrer Basisklassen, zuzüglich derer, die sie selbst definiert.

Vererbung (inheritance)

- **Ausnahme: Konstruktoren**, **Destruktoren** und **operator=()** werden **nicht** vererbt.

- Abgeleitete Klassen können wiederum als Basisklassen dienen --> **Vererbungshierarchie**

- Sei **class X : public Y { ... };**
  Die **Methodenschnittstelle einer Klasse X** besteht aus den **öffentlichen Methoden** der Basisklassen **Y** und den **öffentlichen Methoden** der abgeleiteten Klasse **X.**

- Bei der Zuweisung eines Objektes einer **abgeleiteten** Klasse an ein Objekt einer **Basisklasse** sind die **zusätzlichen** Member und Methoden der abgeleiteten Klasse über das betreffende **Objekt** der **Basisklasse nicht** mehr im Zugriff (**Slicing**). Auch ein **Cast** hilft hierbei **nicht** (vgl. **inh3n**).

- Die **Adresse** bzw. **Referenz** eines Objektes einer **abgeleiteten Klasse** kann an die **Zeigervariable** bzw. **Referenz** einer **Basisklasse** zugewiesen werden:

  - Die zusätzlichen Komponenten (Member und Methoden) der **abgeleiteten Klasse** sind über die **Zeigervariable** bzw. **Referenz** der **Basisklasse** dann **nicht sichtbar**.

  - Ein **Cast** des betreffenden **Basisklassenzeigers** oder **-referenz** auf die **abgeleiteten Klasse** macht die **zusätzlichen Member** und **Methoden** der **abgeleiteten Klasse** wieder **sichtbar**.

- **Achtung**:

  Die Zeigervariable oder Referenz einer **Basisklasse** kann mittels **Cast** in einen Zeiger bzw. in eine Referenz einer **abgeleiteten** Klasse verwandelt werden, ohne dass real auf ein abgeleitetes Objekt verwiesen wird. Der Zugriff mittels dieser gecasteten Zeigervariablen bzw. Referenzen auf Member und Methoden der abgeleiteten Klasse liefert **Fehler, falsche bzw. zufällige Werte** (vgl. **inh3n**).

  Mittels **dynamischer Casts** kann das während der Laufzeit der Typ des Objektes überprüft werden, auf den der Basisklassenzeiger bzw. die Basisklassenreferenz verweisen (vgl. **cast1**).

- **Castoperator (Typ)** kann durch **4 unterschiedliche Casts** mit abgestufter Wirkung ersetzt werden:

  static_cast <Typ> (ausdruck)
  Statischer Cast wird während der Compile-Time ausgewertet, d.h. ohne Prüfung des Typs des dynamisch während der Laufzeit zugewisenen Objektes, Zeigers oder Referenz. Kann bei **Downcasts** zu **falschen** Ergebnissen führen, ist unsicher, nur für nicht virtuelle Ableitungen zugelassen (vgl. **cast1**).

dynamic_cast <Typ> (ausdruck)

Dynamischer Cast für **Zeiger** oder **Referenzen** miteinander verwandter **polymorpher** Klassen (mindestens eine Methode muß **virtual** sein), nur solche werden ineinander konvertiert.
Auch eine Konvertierung virtueller Basisklassen in abgeleitete Klassen wird unterstützt.

Ausführung während der **Laufzeit**, **Rückmeldung 0**, falls Konvertierung **unzulässig** (bei Zeigern) bzw. **Exception**, falls Konvertierung **unzulässig** (bei Referenzen) (vgl. **cast1**).

const_cast <Typ> (ausdruck)

Entfernen der Konstantheit eines Typs

reinterpret_cast <Typ> (ausdruck)

beliebige Konvertierungen, gefährlich

Private Ableitungen:

class X : private Y { ... } bzw. **class X : Y { ... }** ist eine **private Ableitung**

Der Zugriff auf Basisklassenkomponenten seitens der abgeleiteten Klasse ist bei **private-Vererbung** äquivalent zur **public-Vererbung** (nicht auf private-, jedoch uneingeschränkt auf protected und public-Member der Basisklasse zugreifbar).

Von **außerhalb** eines Objektes einer **private**-abgeleiteten Klasse sind jedoch **keine** Komponenten der Basisklasse erreichbar (vgl. **private1**).

D.h. die **abgeleitete Klasse verdeckt** im Falle der **private**-Vererbung die Basisklasse.

Die **Methodenschnittstelle** der **private**-abgeleiteten Klasse besteht nur noch aus den **public**-Methoden der abgeleiteten Klasse.

In Klassen, die von einer **private**-abgeleiteten Klasse weiter abgeleitet werden, sind **keine** Komponenten **oberhalb** der **private**-abgeleiteten Klasse zugreifbar.

Vererbung (inheritance)

Soll trotz **privater** Ableitung eine **public** - Methode **f** oder Member **f** der Basisklasse von außerhalb der abgeleiteten Klasse zugreifbar sein, dann kann folgende Notation verwendet werden (vgl. **inh8**):

```
class X : Y {
                public:  int f (char  x) { return  Y::f(x); }
                                // ...
};
```

oder einfacher:

```
class X : Y {
                public:  Y::f(x);
                                // ...
};
```

Zeiger auf Objekte **private**-abgeleiteter Klassen können nicht Zeigern der Basisklasse zugewiesen werden, da hiermit die "Abchirmung" der Basisklassenkomponenten zu umgehen ist.

Referenzen von Basisklassen können **nicht** mit Instanzen **private-abgeleiteter** Klassen initialisiert werden (vgl. **inh6**, **inh8**)