Syntax : *Klassenschlüssel Klassenname* [*<:Basisliste>*] [*{ <Elementliste> }*]
Klassenschlüssel : **class**, **struct**, **union**
Basisliste : Liste der Elternklassen siehe Vererbung
Elementliste : Liste der Memberdaten und Funktionen

Sichtbarkeit : **public**, **private**, **protected**
public-Member sind inner- und außerhalb der Klasse bekannt, private-Member sind nur in den Memberfunktionen bekannt, protected tritt in Verbindung mit Vererbung auf.
union enthält nur public-Member, struct-Member sind als Standard public, class-Member sind als Standard private.

Memberdaten : Die Daten einer Klasse, sie repräsentieren die zeitvarianten/zeitinvarianten bzw. die Klassen-/Objektbezogenen Zustände von Objekten einer Klasse, sie können von jedem beliebigen, an der Stelle bekannten Typ, auch einem Klassentyp sein und sind in der Regel private oder protected vereinbart.

Memberfunktionen: Die Funktionen einer Klasse, sie sind in der Regel public und stellen das Interface der Klasse dar, ausschließlich über sie sollte auf die Memberdaten zugegriffen werden. Sie lassen sich gruppieren in Zugriffsfunktionen, Verwaltungsfunktionen, Implementationsfunktionen und Hilfsfunktionen. Sie verfügen über einen **this**-Pointer, der auf das aktuelle Objekt zeigt. Memberfunktionen können überladen werden und Defaultparameter besitzen.

Definition von Memberfunktionen: Memberfunktionen werden entweder vollständig in der Klassendeklaration definiert, dann werden sie automatisch als Inlinefunktionen übersetzt, oder sie werden außerhalb der Klassendeklaration definiert, dann wird dem Funktionsnamen der Klassenname gefolgt vom **scope**-operator '**::**' vorangestellt. Der Scopeoperator stellt die Verbindung zur zugehörigen Klasse her und öffnet für die Funktionsdefinition den Namensraum der zugehörigen Klasse.

Gültigkeitsbereiche:lokalerBlock, umgebende Blöcke, Klasse, Quelldatei
Ein Bezeichner eines "inneren" Gültigkeitsbereiches verdeckt einen gleichen Bezeichner eines äußeren Gültigkeitsbereiches.

Konstruktoren : Konstruktoren sind Memberfunktionen, die bei jeder Erzeugung eines Objektes automatisch aufgerufen werden. Sie haben keinen Returnwert, wohl aber Parameter, wobei Defaultparameter zulässig sind. Konstruktoren dürfen überladen werden. Existiert zu einer Klasse kein Konstruktor, so wird vom Kompiler ein Defaultkonstruktor erzeugt, der nichts macht. Ein spezieller Konstruktor ist der Kopierkonstruktor, der als Parameter eine Referenz auf ein Objekt der eigenen Klasse erhält. Er ist nötig, wenn Klassen Pointer als Memberdaten enthalten. Existiert kein Kopierkonstruktor, so werden bei der Initialisierung eines Objektes durch ein anderes Objekt der Klasse die Datenmember elementweise kopiert. Der Name eines Konstruktors besteht aus dem Klassennamen.

Destruktoren : Destruktoren werden automatisch bei der Vernichtung von Objekten aufgerufen, sie haben keinen Returnwert und keine Parameter, somit können sie nicht überladen werden. Ihr Name besteht aus dem Klassennamen und einer vorangestellten Tilde '~', ihre Aufgabe ist es, ein Objekt zu bereinigen.

Lebenszeit : Objekte können per Definition angelegt werden, dann hängt ihre Lebenszeit vom Gültigkeitsbereich und der Speicherklasse ihrer Definition ab. Wird der Gültigkeitsbereich des Bezeichners verlassen, so wird das Objekt automatisch vernichtet, wenn es nicht mit dem Speicherklassenattribut static definiert wurde. Objekte, die dynamisch mit **new** erzeugt worden sind, müssen mit **delete** vernichtet werden.

Operator **new** : '**new**' *<typname>*[*'***(' <***Initialisierungsparameterliste>'***)'**]
Er erzeugt ein Objekt, vom Typ typname, wobei zunächst der Speicher aus dem free store bereitgestellt, und danach der Konstruktor entsprechend der Initialisierungsparameterliste aufgerufen wird und gibt einen Pointer vom Typ typname* auf das erzeugte Objekt zurück.
Dieser Pointer darf nicht "verlorengehen", da sonst der durch das Objekt belegte Speicher nicht freigegeben werden kann.

Operator **delete** : Vernichtet ein Objekt, dazu wird zunächst der Destruktor ausgeführt und danach der Speicher zurückgegeben, darf nur in Verbindung mit Pointern, die mit new erzeugt wurden, verwendet werden.

Zugriff auf Member: Innerhalb von Memberfunktionen werden die Member der eigenen Klasse durch ihren Membernamen angesprochen. Von außen erfolgt der Zugriff über den Objektnamen, den Punkt- oder Pfeilselektor gefolgt von dem Membernamen.

const-objekts : Von Klassen können, so wie von jedem anderen Typ Konstanten definiert werden, diese müssen bei ihrer Definition initialisiert werden, ihr Wert ist später nicht mehr änderbar. Es können nur Memberfunktionen, die auf Datenmember nicht schreibend zugreifen, aufgerufen werden, sie müssen mit dem Schlüsselwort **const** hinter der Parameterliste gekennzeichnet sein. const-Memberfunktionen können durch nicht-const-Memberfunktionen mit gleichen Parameterlisten überladen werden.

Memberobjekte : Member einer Klasse können selbst von einem Klassentyp sein. Für ihre Initialisierung wird hinter dem Funktionskopf des Konstruktors ein Memberinitialisierer angegeben in der Form:
'**:**' <*membername*>'**(**'<*Initialisierungsparameterliste*>'**)**'
Mehrere Memberinitialisierer werden durch Komma getrennt nacheinander angegeben, die Abarbeitungsreihenfolge ist nicht definiert, die Memberinitialisierer werden vor dem Konstruktor ausgeführt. Auch ein Memberobjekt kann ein Konstobjekt sein, es kann nur durch den Memberinitialisierer mit einem Wert belegt werden.

Zuweisungoperator: Werden Objekte einander zugewiesen, so werden ihre Datenmember memberweise kopiert. Enthält ein Objekt Pointer, so ist dies nicht sinnvoll, der Zuweisungsoperator muss für die Klasse überladen werden, hierfür wird eine Funktion
<typ>& 'operator='('const' <typ> & other)
in die Klasse aufgenommen. Ihre Definition enthält dann die notwendigen Operationen zum Kopieren des Objektes. Eine Klasse mit überladenem Zuweisungsoperator benötigt in der Regel auch einen Kopierkonstruktor für die Initialisierung von Objekten durch Objekte ihrer Klasse sowie einen Destructor.

this-Pointer : Der **this**-Pointer ist ein Pointer auf das aktuelle Objekt. Er wird als impliziter Parameter bei jedem Memberfunktionsaufruf übergeben, er kann benutzt werden, um festzustellen, ob ein als Parameter übergebenes Objekt das aktuelle Objekt selbst ist.

static Memberdaten: Für statische Memberdaten existiert nur eine Instanz für alle Objekte der Klasse. Die statischen Memberdaten werden in der Klasse nur deklariert und außerhalb der Klassendeklaration mit vorangestelltem Klassennamen und **scope**-operator definiert, und müssen auch in ihrer Definition initialisiert werden. Sie werden nicht über die Konstruktoren initialisiert. Auf statische Memberdaten kann zusätzlich über den Klassennamen und den **scope**-operator zugegriffen werden.

friend-Functions : Friend-Funktionen sind keine Memberfunktionen, können aber trotzdem auf die privaten Daten der Klasse, die sie als Friend ausweist zugreifen, sie werden vor allem zur Konvertierung und zum Operatorüberladen benutzt. Um eine Friendfunktion zu vereinbaren, wird die Funktionsdeklaration in die Klassenvereinbarung aufgenommen und das Schlüsselwort **friend** vorangestellt.

friend-classes : Memberfunktionen von Friend-Klassen haben uneingeschränkten Zugriff zu den privaten Member der Klasse, die die Friend-Vereinbarung enthält. Eine Friend-Klassenvereinbarung besteht aus dem Schlüsselwort **friend class** gefolgt von dem Klassenamen der befreundeten Klasse. Klassen können auch wechselseitig befreundet sein. Auch einzelne Funktionen können als friend gekennzeichnet werden.