Klasse - individuell gestaltbarer Datentyp zur **Kapselung** von semantisch zusammengehörenden **Daten** und der **Methoden** (Funktionen, Tools), die auf diese Daten unmittelbar zugreifen.

- Ziel ist u.a. die **höhere Effizienz** und **Fehlerfreiheit** bei der Programmierung

- entsteht aufgrund der **OO Analyse** und des **OO Designs** einer Problemstellung

- besitzt kein, ein oder mehrere **Datenmember** und keine, eine oder mehrere **Methoden**

- stellt Verallgemeinerung von ***struct*** - Typen aus **C** dar, in C++ ist ***struct*** wie ***class*** nutzbar

- besitzt **Zugriffsspezifizierer** ***private*** (default), ***public*** und ***protected***, um einen kontrollierten Zugriff auf Datenmember und Methoden von außerhalb der Klasse zu ermöglichen

- **Methoden** und **Daten** einer Klasse werden allgemein ***Member*** dieser Klasse genannt

- Klassen werden bei der Variablendeklaration instantiiert, d.h. es entstehen **Objekte** bzw. **Instanzen** dieser Klasse (des Datentyps)

- Von den **Datenmembern** gibt es einen Satz pro **Objekt**

- Unbeschränkten Zugriff auf die Member eines Objektes haben nur die Methoden der Klasse (des Typs), von der das Objekt abstammt

- **Syntax** einer Klasse:

class [name [:base_list]] { [private:]
Datenmember
Methoden
public:
Konstruktoren
Destruktoren
Methoden
protected:
Datenmember
Methoden
} [declarators];

- ***base_list*** beschreibt **Basisklassen**, von der die aktuelle Klasse ***public***, ***private*** oder ***protected*** abgeleitet ist

- ***private***, ***public***, ***protected*** können beliebig oft und in beliebiger Reihenfolge vorkommen

- **Konstruktoren** sind Methoden, die bei der Deklaration einer Variablen (Instanz, Objekt) dieser Klasse **automatisch** aufgerufen werden und die Datenmember **initialisieren**

- **Destruktoren** sind Methoden, die unmittelbar vor dem Existenzende eines Objektes automatisch gerufen werden. Destruktoren dienen u.a. der Freigabe des dynamisch vergebenen Speicherplatzes innerhalb von Objekten

- Jede Klasse sollte einen **Defaulkonstruktor** besitzen, der die **parameterlose Initialisierung** bei der Instantiierung ermöglicht

- Der Zugriff auf die Member einer Klasse bzw. Objektes wird über **Zugriffsspezifizierer** **private**, **public**, **protected** streng reglementiert (**Kapselung**)

- **private** Member: alle Methoden innerhalb der Klasse haben uneingeschränkten Zugriff, die Verwendung außerhalb der Klasse wird verhindert und verursacht Übersetzungsfehler

- **public** Member: alle Methoden, sowohl innerhalb als auch außerhalb der Klasse, haben uneingeschränkten Zugriff . Können von außerhalb der Klasse benutzt werden

- **protected** Member: **wie private**, zusätzlich haben Methoden abgeleiteter Klasse Zugriff

auf **protected** Member

- **Methodenschnittstelle einer Klasse**: alle von außerhalb der Klasse aufrufbaren Methoden, d.h. alle **public**-Methoden der Klasse

- Innerhalb einer Klasse, die instantiiert wurde, existiert immer der Zeiger ***this***, der auf das **aktuelle Objekt** zeigt. Member können von innerhalb der Klasse mit ***this->member*** bzw. ***(*this).member*** verwendet werden

- Klassen können wieder Klassendefinitionen, Objekte oder Zeiger auf Klassen enthalten

- Klassen können von anderen Klassen abgeleitet sein (**Vererbung**), einfach oder multipel

- Klassen können ***static*** - Member besitzen, diese sind **ohne Objekte** der Klasse nutzbar

- Funktionen, Methoden oder Klassen können **friends** von Klassen sein, **friends** können auf private- und protected-Member einer Klasse **uneingeschränkt** zugreifen

- **Kopierkonstruktoren** sind sinnvoll, um bei Klassen mit Zeigermembern zu verhindern, daß diese Member nach dem Kopieren des Objektes auf den gleichen Speicherplatz zeigen

- Konstruktoren und Methoden können innerhalb einer Klasse **überladen** sein, d.h. der gleiche Methodenname kann mit jeweils unterschiedlichen Parameterlisten (Signaturen) **mehrfach** vorkommen und unterschiedlich implementiert sein (**Polymorphie**)

- Objekte können **automatisch** oder **dynamisch mit new** angelegt werden

- Mit **new** angelegte Objekte sollten mit **delete** wieder freigegeben werden

- Eine Klasse soll das entsprechende Objekt der realen Welt möglichst gut abbilden

- Methoden und deren Implementation sollen programmtechnisch möglichst effektiv und leicht handhabbar sein

- Entwurf von Klassen ist ein **iterativer** (hoffentlich konvergierender) Prozeß, der im Wechselspiel von Planung und Erprobung stattfindet

- Klassen können, ohne Implementierung der Methoden, in Form **abstrakter Klassen** als Vorlagen (Methodenschnittstellen) für davon abgeleitete Klassen dienen

- In eine Folge von abgeleiteten Klassen können **virtuelle** Methoden in Abhängigkeit von der konkreten Klasse unterschiedlich (**polymorph**) implementiert werden und erst während der Laufzeit des Programmes, in Abhängigkeit vom Wert eines Objektzeigers/einer Objektreferenz **spät gebunden** werden (**Späte Bindung, late binding, Polymorphie**)