

## Klassendeclaration // Beispiel h

#pragma once

```
class Beispiel  
{  
public:  
    Beispiel();  
    ~Beispiel();  
    void DieMethode();  
  
private:  
    int seinAttribut;  
};
```

3 Schutz gegen Mehrfacheinbindung



Klassendeclaration mit

- Konstruktor
- Destruktor
- Methoden

- Attribute

## Konstruktor / Destruktor

Sondermethoden ohne Rückgabewert, Aufruf erfolgt beim Erstellen (→ initialisieren) bzw. Zerstören ("Aufräumarbeiten") des Objekts.

public: öffentlich: Jedes Objekt hat Zugriff auf öffentliche Attribute und Methoden  
+ private: privat: Nur das Objekt selbst hat Zugriff auf private Attribute und Methoden  
- protected: geschützt: Nur das Objekt selbst und davon abgeleitete Objekte haben Zugriff auf protected Attribute und Methoden (→ Vererbung!)

Beispiel
- seinAttribut: int = 0
+ Beispiel()
+ ~Beispiel()
+ DieMethode()

## Klasse und Headerdefinitionen: // Beispiel.cpp

```
#include "Beispiel.h"
```

```
Beispiel::Beispiel()
```

```
{  
    seinAttribut = 0;  
}
```

```
Beispiel::~Beispiel()
```

```
{  
}
```

```
void Beispiel::DieMethode()
```

```
{  
}
```

} Header erobert

} Methodendefinition mit

- Konstruktor

- Destruktor

- Methoden

## Komposition:



// Header Class1.h

#include "Class2.h"

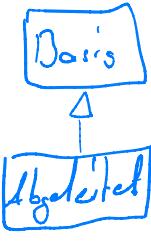
class Class1

```
{  
    private: Class2 obj2;  
};
```

Ein Objekt vom Typ Class2 wird als private Attribut in Class1 erstellt.

Assoziationen werden in C++ in der Regel als Zeiger dargestellt

Vererbung:



// Abgeleitet.h  
#include "Basis.h"  
class Abgeleitet : public Basis  
{  
};

Durch Vererbung entsteht eine "ist ein"-Beziehung. Die abgeleitete Klasse erhält Eigenschaften und Methoden der Basisklasse.

Abstrakte Klasse

- dienen als Vorlage zur Vererbung (-> Polymorphismus)
- können nicht instanziiert werden (kein Objekt)
- besitzen in C++ mindestens eine abstrakte Methode (ein virtuelle Methode)

Interfaces

besitzen  nur  abstrakte Methoden und  keine  Attribute

Iso: virtual int abstrakteMethode() = 0;

Abstrakte Methoden werden virtual deklariert und =0 gesetzt

Es gibt  keine  Methodenkodierungen.

## Nachrichten an Objekte im Quelltext

```
void Class1::MethodXY()  
{  
    Objekt2.Meldung();  
}
```

In einer Methode von Klasse1 wird die Methode Meldung() von Objekt2 aufgerufen.  
=> "Objekt1 schickt eine Botschaft an Objekt2"

Zeigeroperator =>  
... dereferenziert einen Zeiger auf ein Objekt und ruft den Punktoperator auf:

pObj2->Meldung(); // entspricht (\*pObj2).Meldung();

Objekte erstellen im Stack und Heap:

Stack:

Class2 objekt2;  
Class2 objekt2(Parameter);

Heap:

Class2 \* pObj2 = new pObj2;  
Class2 \* pObj2 = new pObj2(Parameter);

Virtuell: (C++ Schlüsselwort)

- Kennzeichnet Methoden die Polymorph eingesetzt wird (Überschreiben)  
=> die passende Methodendefinition wird zur Laufzeit ermittelt / ausgeführt  
=> late binding  
**Gegenstück**  
early binding (Binden zur Compilierzeit)

Polymorphie in C++:

=> Herstellen einer "ist ein" Beziehung durch Vererbung oder Realisierung  
=> Basisklasse kann als Zeiger für alle abgeleiteten Objekte verwendet werden  
 Melder \* derFSensor = new Feuersensor;  
=> derFSensor ist vom Typ Zeiger auf einen Feuersensor, kann aber immer verwendet werden, wenn ein Zeiger auf Melder erwartet wird  
=> Polymorphie -> Methoden werden zur Laufzeit aufgelöst -> late binding