Übersichtsblatt:
Abstrakte Klasse und Schnittstellen

In der Softwareentwicklung müssen in größeren Projekten oft viele Klassen, von verschiedenen Entwicklern zusammenarbeiten. Um den reibungslosen Ablauf dieser Zusammenarbeit zu gewährleisten bietet C++ die Möglichkeit an so genannte Abstrakte Klassen und Schnittstellen (auch Interfaces genannt) zu implementieren.

Abstrakte Klassen:

Eine abstrakte Klasse ist zunächst eine ganz normale Klasse mit Exemplarvariablen und Exemplarmethoden. Der unterschied zu anderen Klassen ist lediglich, dass sie über einige Exemplarmethoden verfügt, die nicht ausimplementiert sind. Sie werden in der Klassendefinition im Header zwar deklariert, hier allerdings „=0" gesetzt. Die Implementierung dieser Klasse wird dann in einer anderen Klasse ausgeführt, die von der abstrakten Klasse erbt (→ siehe Übersichtsblatt: Vererbung).

Der Vorteil ist, dass die normal Methoden dieser Klasse ganz wie gewohnt genutzt und überschrieben werden können. Die abstrakten Methoden allerdings müssen bei der Vererbung auf jeden fall implementiert werden. Es besteht nicht, wie bei einer normal Vererbung die Freiheit, genau nur die Klassen zu implementieren die man will.

Schematisches Beispiel:

```
class AbstrakteKlasse{
      privat:
            int exemplarVariable;

      public:
            void setExemplarVariable(int neuerWert);
            virtual void abstrakteMethode()=0;
};
```

Schnittstellen (Interfaces):

Eine Schnittstelle ist eine Klasse, die keine eigenen Exemplarvariablen hat und nur aus abstrakten Methoden besteht.

Der Sinn hierbei ist, dass eine Schnittstelle im Grund nur eine Vorgabe ist. Jede Klasse die von der Schnittstelle erbt muss die in der Schnittstelle deklarierten Methoden anbieten. Die Implementierung dieser Methoden sprich deren Inhalt, bleibt dem Entwickler der erbenden Klasse überlassen.

Anwendung findet dieses System in großen Softwareprojekten. Hier greift das Hauptprogramm nur auf Objekte vom Typ der Schnittstelle zu, durch Polymorphie wird dann die entsprechende Unterklasse aufgerufen. So kann sichergestellt werden, dass alle Klassen, auch von verschiedenen Entwicklern, kompatibel zueinander sind.

Schematisches Beispiel:

```
class Schnittstelle{
      public:
            virtual int abstrakteMehtode1()=0;
            virtual void abstrakteMethode2()=0;
};
```

© Marco Rancinger

eTeachers.de
Lernen leicht gemacht