3. Das Basis- und statische Modell des Alarmsystems

Beim Erstellen eines objektorientierten Programmes sind die Grenzen zwischen Design und Analyse wesentlich unschärfer getrennt als bei der Entwicklung von strukturierten Programmen. Analyse und Design gehen sozusagen „Hand in Hand“.

3.1 Klassen und Objekte, das Basismodell

Um eine vorläufige Liste aller Objekte/ Klassen zu erhalten unterstreicht man im Text der Problemstellung zunächst die Substantive (Hauptwörter).

- ☞ **Analysieren Sie die Aufgabenstellung und erstellen Sie eine mögliche Liste der Klassen!**
- ☞ **Nutzen Sie auch die Erkenntnisse aus dem Anwendungsfallmodell!**

3.2 Ein erstes statisches Modell:

Im nächsten Schritt versucht man die Objekte zueinander in Beziehung zu setzen um zu einem statischen Modell zu gelangen, das die Wirklichkeit beschreibt.
Klassen (bzw. Objekte), welche im Programm miteinander kommunizieren stehen in der Regel in irgendeiner Beziehung zueinander.

- ☞ **Setzen Sie die Klassen zueinander in Beziehung. Nutzen Sie dazu die Problembeschreibung!**
- ☞ **Das Anwendungsfallmodell liefert ebenfalls Erkenntnisse für das Design der statischen Beziehungen!**

4. Die Realisierung des statischen Programmgerüsts

Im folgenden Schritt werden die grundlegenden Klassen und ihre Beziehung zueinander mit großteils leeren Methodendefinitionen erstellt.

4.1 Die Klasse Schließanlage

Entsprechend dem Anwendungsfallmodell ist die Schließanlage für das Öffnen und Verriegeln der Türen und des Tresorraums zuständig. Mit den Methoden TuerenAuf( ) bzw. TuerenZu( ) sowie TresorAuf( ) und TresorZu( ) erfüllt die Schließanlage ihren Zweck. Der aktuelle Zustand des Schließsystems wird in den Attributen tuereOffen bzw. tresorOffen vom Typ bool gespeichert. Die Türen sind standardmäßig geöffnet, der Tresorraum ebenso.

- ☞ **Ergänzen Sie das Klassendiagramm der Schliessanlage!**
- ☞ **Implementieren Sie die Schließanlage im Programm! Die Methoden sind mit leerem Methodenrumpf zu erstellen!**

4.2 Das Display

Das Display zeigt den Status der Schließanlage für den Benutzer an. Dazu verfügt es über die Methode Show(char* stat). Der übergebene String erscheint mit der Meldung "Status der Alarmanlage: ... " in der Anzeige.

- ☞ **Vervollständigen Sie das Klassendiagramm des Displays!**
- ☞ **Implementieren die Methode Show(...) mit ihrer Funktionalität!**

4.3 Die Klassenhierarchie der Melderklassen

Einbruchmelder und Feuersensor sind abgeleitete Klassen. Sie erben Eigenschaften und Methoden von der Basisklasse Melder. Da sämtliche Melder gemeinsame Methoden unterstützen, empfiehlt es sich, ein Interface für alle Melderklassen zu erstellen.

4.3.1 Das Interface IMelder:

Jeder Melder im System kann Alarm geben (void GibAlarm( )) sowie an eine Alarmanlage angeschlossen werden (void Anschliessen(Alarmanlage*al)).

In C++ existiert noch kein eigenes Schlüsselwort für Interfaces. Ein Interface wird daher als Klasse definiert, die ausschließlich abstrakte Methoden enthält.

Das Interface IMelder dient sozusagen als Vorlage für alle Melder.
Das Schlüsselwort **virtual** erklärt dem Compiler, dass die richtige Methodendefinition erst zur Laufzeit gesucht und ausgeführt wird (late binding).
Setzt man eine virtuelle Methode auf 0, so wird sie abstrakt (rein virtuell) und die gesamte Klasse wird abstrakt.

```
class IMelder
{
   public:
   virtual void GibAlarm( ) = 0;
   ...
};
```

Da die abstrakten Methoden in der Quelltextdatei jetzt nicht mehr definiert werden dürfen, kann vom Interface keine Instanz (Objekt) mehr gebildet werden. Das Interface dient als Vorlage zur Vererbung. Die eigentliche Definition findet in den abgeleiteten Klassen statt.
Interface Methoden dienen zur Vererbung und sind daher immer public zu definieren.

☞ **Ergänzen Sie das Klassendiagramm des Interfaces!**
☞ **Implementieren Sie das Interface IMelder im Programm!**

4.3.2 Die abstrakte Basisklasse Melder

Die abstrakte Basisklasse Melder liegt in der Vererbungshierarchie eine Ebene unter dem Interface. Sie bildet die Grundlage für die abgeleiteten Melder im Programm.
In C++ entspricht die Realisierung eines Interfaces einer Vererbung von der Interface Klasse. In der Klassendeklaration der abgeleiteten Klasse (Melder) werden nach einem Doppelpunkt das Schutzattribut (public) und der Name des Interfaces (IMelder) angegeben. Natürlich muss die Headerdatei des Interfaces per include eingebunden werden, damit die Melderklasse die Methodendeklarationen übernehmen kann.

Abstrakte Basisklassen entstehen, wenn eine Basisklasse ein oder mehrere abstrakte (rein virtuelle) Methoden besitzt.
Entweder deklariert man abstrakte Methoden wieder mit virtual und setzt diese auf 0 oder man überschreibt nicht alle Methoden des realisierten Interfaces.
Im Fall des Alarmsystems überlässt man die Definition von GibAlarm( ) den abgeleiteten Klassen. GibAlarm( ) darf daher nicht in der Melderklasse deklariert werden und wird auch nicht im Klassendiagramm aufgeführt.

```
#include "IMelder.h"
…
class Melder : public IMelder
{
   public:
   ...
   void Anschliessen(Alarmanlage* al);
   ...
```

Die Methode Anschliessen(…) ist bei allen Meldern gleich und kann bereits in der Basisklasse implementiert werden.

Da die rein virtuelle Methode GibAlarm( ) in der Quelltextdatei jetzt nicht mehr definiert werden darf, kann von der abstrakten Klasse Melder ebenso kein Objekt mehr gebildet werden. Sie dient als Vorlage zur Vererbung. Die eigentliche Definition findet in den abgeleiteten Klassen Feuersensor und Einbruchmelder statt.

Außerdem steht die Klasse Melder in einer 1 zu 1 Assoziation zur Klasse Alarmanlage. Assoziationen werden in C++ in der Regel als Zeiger dargestellt.
Melder hält als Attribut einen Zeiger auf die Alarmanlage.
Diese Zeiger können aber auch nur lokal in einer Methode bestehen (z.B. Parameterübergabe oder dynamisches Erzeugen eines Objekts).
Durch den Zeiger auf eine Alarmanlage kann jeder Melder später Nachrichten an die Alarmanlage verschicken.

```
…
   protected:
   Alarmanlage* seineAlarmanlage;
   …
};
```

Zum Erstellen der Assoziation muss jeder Melder mit der Methode Anschliessen(Alarmanlage* al) einen Zeiger auf ein Objekt vom Typ Alarmanlage übernehmen. Das Attribut seineAlarmanlage wird mit diesem Zeiger initialisiert.
Als Schutzattribut erhält seineAlarmanlage den Status protected (geschützt), damit die abgeleiteten Klassen darauf Zugriff erhalten. In der UML werden geschützte Eigenschaften und Methoden durch ein vorangehendes Nummernzeichen (#) dargestellt.

☞ **Ergänzen Sie das Klassendiagramm der Klasse Melder!**
☞ **Implementieren Sie die Klasse Melder im Programm! Die Methode Anschliessen(...) initialisiert den Zeiger seineAlarmanlage mit dem übergebenen Zeiger al.**

4.3.3 Die Klassen Einbruchmelder und Feuersensor

Einbruchmelder und Feuersensor sind abgeleitete Klassen. Sie erben Eigenschaften und Methoden von der Basisklasse Melder. Bei der öffentlichen Vererbung entsteht eine "ist ein" Beziehung zwischen abgeleiteter Klasse und Basisklasse.

In der Klassendeklaration der abgeleiteten Klasse (Feuersensor) werden nach einem Doppelpunkt das Schutzattribut (public) und der Name der Basisklasse (Melder) angegeben.

```
#include "Melder.h"

class Feuersensor : public Melder
{
   ...
};
```

Attribute und Methoden werden weitervererbt, auf private Elemente der Basisklasse kann jedoch nicht zugegriffen werden. C++ kennt für diesen Fall einen Zwischenstatus von public und private. Kennzeichnet man Elemente der Basisklasse als protected, so werden diese für abgeleitete Klassen sichtbar. Für Klassen außerhalb der Vererbungshierarchie erscheinen geschützte Elemente jedoch so, als wären sie privat.
In der UML werden geschützte Eigenschaften und Methoden durch ein vorangehendes Nummernzeichen (#) dargestellt. Dies wird beim Entwurf der Basisklasse Melder deutlich.
Um einen Alarm auslösen zu können, besitzen sowohl der Feuersensor als auch der Einbruchmelder die Methode GibAlarm( ).

- ☞ **Überarbeiten Sie die Klassendiagramme der beiden Melder!**
- ☞ **Programmieren Sie die beiden Melder im Alarmsystem! Die Methodenrümpfe bleiben dabei wieder leer.**

4.4 Die Klasse Alarmanlage

Da die Alarmanlage in Beziehung zur Schließanlage steht, verfügt sie als Attribut über den Zeiger ihreSchliessanlage vom Typ Schliessanlage.

Um das Attribut ihreSchliessanlage bei der Erzeugung der Alarmanlage zu initialisieren, übernimmt der Konstruktor als Parameter einen Zeiger auf die Schliessanlage. Im Methodenrumpf wird das Attribut initialisiert.

```
Alarmanlage::Alarmanlage(Schliessanlage* schl)
{
   ...
   ihreSchliessanlage = schl;
}
```

Als Komposition verfügt die Alarmanlage über ein Display. Kompositionen beinhalten ein echtes Objekt vom Typ der eingeschlossenen Klasse (Display). Dieses wird bei der Instantiierung des einschließenden Objekts automatisch erzeugt. Löscht man das einschließende Objekt (Alarmanlage), so werden auch alle eingeschlossenen Objekte (ihrDisplay) zerstört.

```
   ...
   Display ihrDisplay;
};
```

In einem String (char[10]) speichert die Alarmanlage ihren Status.

SetFeuer( ), SetEinbruch( ) und Reset( ) bewirken eine Änderung des Status der Alarmanlage. Mit ShowStatus( ) kann der aktuelle Status angezeigt werden.

- ☞ **Vervollständigen Sie das Klassendiagramm der Alarmanlage!**
- ☞ **Implementieren Sie die Methoden mit leerem Methodenrumpf!**
  Der Konstruktor initialisiert den Zeiger auf die Schliessanlage und setzt den Status auf "bereit" (Tipp: strcpy_s(…) kopiert einen String in ein char Array!).