5. Das dynamische Design der Alarmanlage

Nachdem das Basismodell sowie das statische Modell der Alarmanlage entworfen sind wird im nächsten Schritt das dynamische Modell für das Verhalten des Systems zur Laufzeit implementiert.

5.1 Anforderungen zum Brandfall

Die Alarmanlage befindet sich nach dem Einschalten im Status "bereit". Trifft eine Feuermeldung ein, so werden die Türen geöffnet und die Feuerwehr alarmiert. Der Status des Alarmsystems wechselt auf "Feuer" und wird angezeigt.
Der Feuersensor meldet dazu über die Methode GibAlarm( ) einen Brand. Daraufhin ruft er die Methode SetFeuer( ) der Alarmanlage auf.
Die Alarmanlage setzt ihren Status auf "Feuer" und zeigt den neuen Status über das Display an.
Die Schließanlage wird von der Alarmanlage angewiesen, die Türen sowie den Tresorraum zu öffnen.

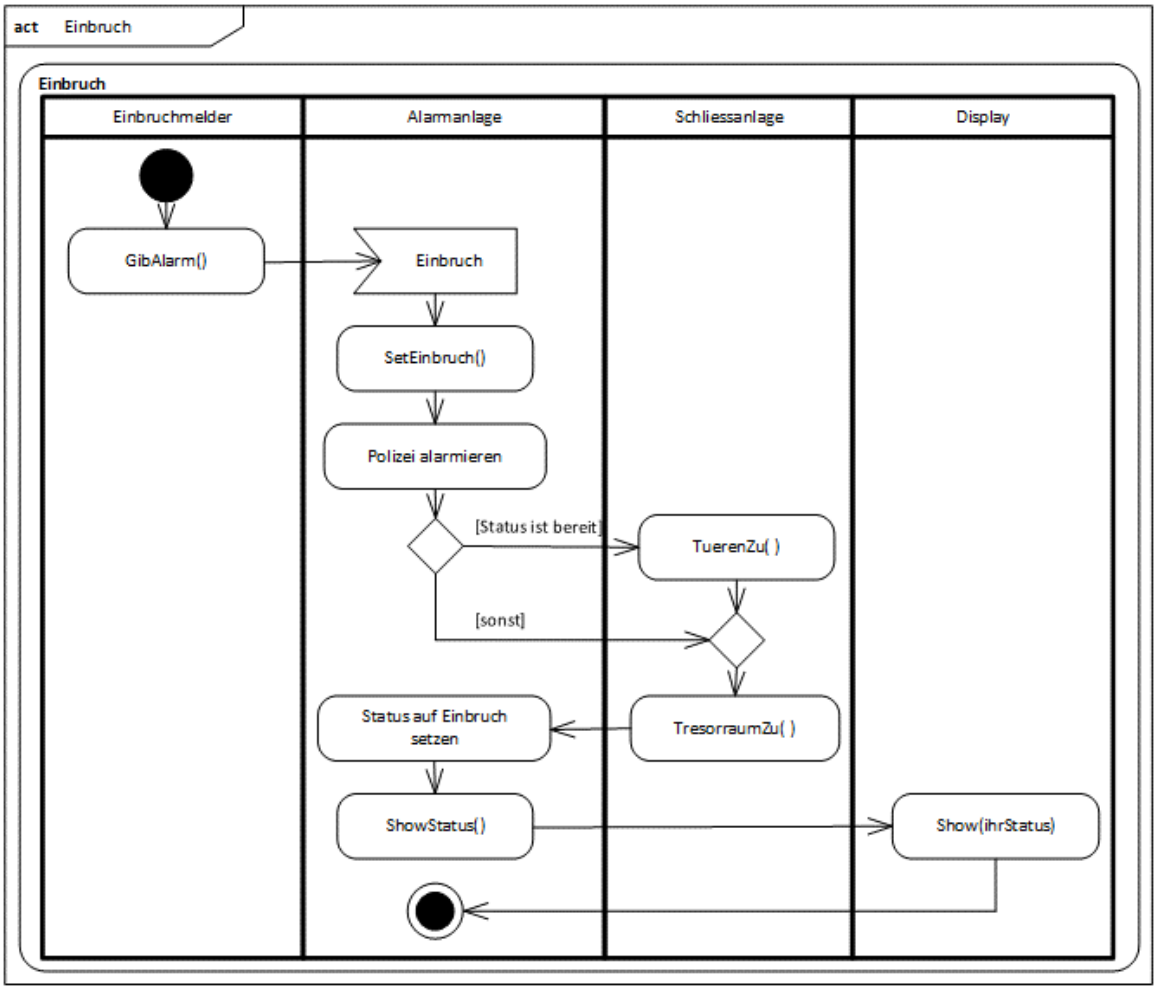
☞ **Erstellen Sie eine Aufrufkette der Methoden im Brandfall (vgl. Stacktrace)!**

5.2 Das Aktivitätsdiagramm zum Einbruchsfall

In der Operation SetEinbruch( ) wird der Status der Alarmanlage auf "Einbruch" gesetzt. Die Polizei ist zu alarmieren. Nur wenn der Status der Alarmanlage auf „bereit“ und nicht auf „Feuer“ steht, soll die Schließanlage die Türen verriegeln, ansonsten wird einzig der Tresorraum versperrt. Dies soll verhindern, dass Personen z.B. auf der Flucht vor dem Brand durch den Einbruchalarm im Gebäude eingesperrt werden.
Für den Einbruchsfall liegt bereits ein dynamisches UML-Activity Diagramm vor.

☞ **Informieren Sie sich mit Hilfe einer Internetrecherche über die Notationselemente eines Aktivitätsdiagrammes!**

☞ **Analysieren Sie das nachfolgende Aktivitätsdiagramm!**



5.3 Der Ablauf zum Test des Alarmsystems

Beim Test der Alarmanlage steuert eine Applikation in der Methode Run( ) den Programmablauf.

- Zuerst erzeugt man die Objekte dieSchliessanlage, dieAlarmanlage, derFSensor und derEMelder im Heap. Die beiden Melder sollen polymorph über das Interface deklariert werden.
- Beide Melder sind anzuschließen.
- Im Anschluss daran testet man Feuer- und Einbruchmelder mit der Alarmfunktion.
- Nach einem Reset der Alarmanlage wird erneut die Einbruchsmeldung ausgelöst.

☞ **Weshalb wird der Einbruchfall zwei Mal getestet?**

☞ **Ergänzen Sie das Klassendiagramm um die Klasse Applikation!**

6. Die Realisierung der dynamischen Anforderungen

Unter Punkt 4 wurde bereits das statische Modell des Alarmsystems mit allen Beziehungen realisiert. Nachdem das Alarmsystem nach Punkt 5. auch dynamisch durchgeplant ist, kann die Implementierung der Aktionen des Systems beginnen.

6.1 Feuersensor und Einbruchmelder

Laut Beschreibung sendet der Feuersensor im Alarmfall die Nachricht SetFeuer( ) an die Alarmanlage. Dies wird in der Methode GibAlarm( ) des Feuersensors implementiert.

```
void Feuersensor::GibAlarm( )
{
    // Nachricht an Alarmanlage schicken
    seineAlarmanlage -> SetFeuer( );
}
```

Da jeder Feuersensor einen Zeiger auf seineAlarmanlage von der abstrakten Basisklasse Melder erbt, kann er Nachrichten über diese Assoziation verschicken. Der Zugriff auf die Methode SetFeuer( ) erfolgt nicht über den Punktoperator sondern über den Dereferenzzeiger, da seineAlarmanlage als Zeiger und nicht als Objekt realisiert ist.

☞ **Implementieren Sie die Methode GibAlarm( ) des Einbruchmelders analog zum Feuersensor!**

6.2 Die Klasse Display

Das Display gibt in der Methode void Show(char* stat) den übergebenen Status der Alarmanlage auf der Konsole aus. Die Methode void Show(...) wurde bereits unter Punkt 4.2 definiert.

6.3 Die Klasse Schliessanlage

Für die Schließanlage benötigt man im Test vier Stub – Methoden. Diese werden später durch Aufrufe der Methoden des Hardwaretreibers der Schließanlage ersetzt.

void TuerenAuf( )	**void TuerenZu( )**
Bildschirmausgabe:“Tueren geoeffnet!“, die Eigenschaft tuereOffen wird auf true gesetzt.	Bildschirmausgabe:“Tueren geschlossen!“, die Eigenschaft tuereOffen wird auf false gesetzt.

Die Methoden TresorAuf( )/ TresorZu( ) arbeiten analog.

☞ **Programmieren Sie die vier Methoden der Klasse Schliessanlage!**

6.4 Die Funktionen der Alarmanlage

Reset( ):	Der Zustand der Alarmanlage wird auf bereit zurückgesetzt und mit ShowStatus( ) ausgegeben.
ShowStatus( ):	Dem Display wird der aktuelle Status zur Anzeige geschickt.
SetFeuer( )	Bildschirmausgabe: "Die Feuerwehr wird alarmiert!“. Entsprechend der Beschreibung sind noch der Status auf Feuer zu ändern und auszugeben. Die Schließanlage muss alle Türen öffnen.
SetEinbruch( )	SetEinbruch( ) gestaltet sich etwas schwieriger, dürfte Ihnen aber mit Hilfe des Aktivitätsdiagramms (Punkt 5.2) und einer if - Auswahl keine Probleme bereiten. Ansonsten verläuft SetEinbruch( ) analog zu SetFeuer( )!

☞ **Definieren Sie die Methoden der Alarmanlage!**

☞ **Nutzen Sie dazu die Beschreibung und UML Diagramme (Aktivitätsdiagramm)!**

6.5 Die Methode Run( ) der Applikation

Die Steuerung des Testlaufs übernimmt die Applikation in ihrer Methode Run( ).
Dazu sind zunächst die Klasse Applikation (siehe Klassendiagramm) und ihre Methode Run( ) zu implementieren.
Die Einzelschritte des Testlaufs der Alarmanlage sind in der folgenden Beschreibung sowie unter Punkt 5.3 aufgeschlüsselt.

Testlauf des Alarmsystems:

- Erzeugen Sie die folgenden Objekte im Heap:
  dieSchliessanlage,
  dieAlarmanlage,
  derFSensor und
  derEMelder im Heap.
  Die beiden Melder sollen polymorph über das Interface deklariert werden.
- Beide Melder sind an der Alarmanlage anzuschließen.
- Testen Sie die Feuer- und Einbruchmelder mit der Alarmfunktion.
- Nach einem Reset der Alarmanlage wird erneut die Einbruchsmeldung ausgelöst.

☞ **Implementieren Sie die Methode Run( ) der Applikation!**

6.6 Die Hauptfunktion main( )

Um das Programm testen zu können erzeugt man in der Hauptfunktion main( ) eine Instanz der Applikation und Startet die Methode Run( ).

```
...
Applikation dieApp;
dieApp.Run( );
...
```

☞ **Fügen Sie dem Projekt die Quelltextdatei main.cpp hinzu!**
☞ **Erstellen Sie die Hauptfunktion wie beschrieben!**

7. Zusatzprogramm: Das Observer - Pattern

Das bisherige Alarmsystem folgt in vereinfachter Form einem gebräuchlichen Muster. Um bei Softwareprojekten das Rad nicht ständig neu erfinden zu müssen, sind eine ganze Reihe solcher Software-Pattern (Muster) beschrieben. Diese ermöglichen programmiersprachenunabhängige Lösungen zu wiederkehrenden Problemstellungen.

Teamarbeit:

☞ **Informieren Sie sich mit Hilfe einer Internetrecherche über das Observer Pattern!**
☞ **Erstellen Sie eine Kurzpräsentation zum Observer-Pattern!**

- Kurzbeschreibung in eigenen Worten
- UML Diagramm
- Einsatzbeispiel mit Push und Pull
- Die Quellen (z.B. Webseite etc.) sind anzugeben!

☞ **Implementieren Sie das Observer-Pattern im Alarmsystem und überarbeiten Sie auch das bestehende Klassendiagramm!**

- Die Aufteilung nach Push und Pull erfolgt arbeitsteilig.

☞ **Präsentieren Sie Ihre Ergebnisse!**