Brandmeldeanlage

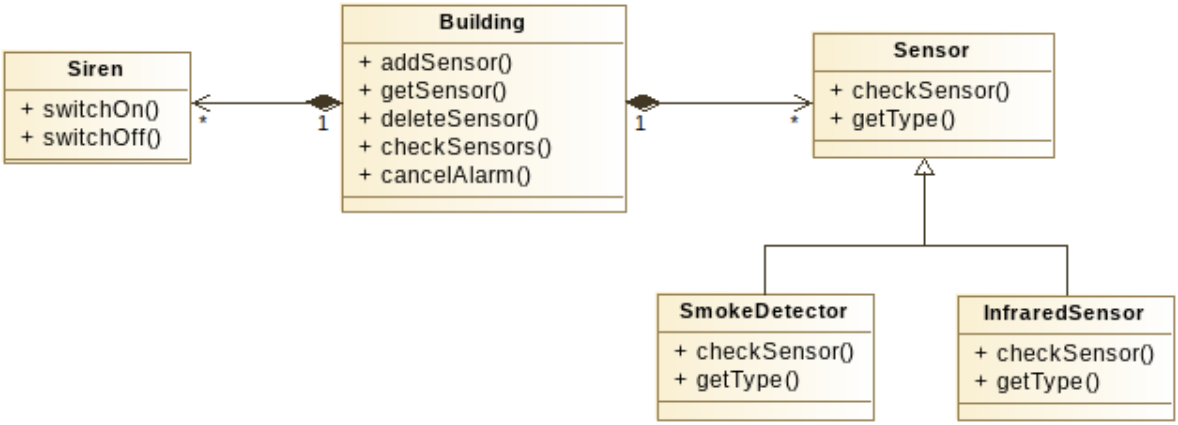
Ihre Softwarefirma hat den Auftrag bekommen, für den MCI-Neubau eine Brandmeldeanlage zu entwicklen. Diese Brandmeldeanlage besteht aus mehreren Sensoren, die über mehrere Stockwerke verteilt sind, mehreren Innensirenen, eine pro Stockwerk, und einer Aussensirene.

In regelmäßigen Abständen sollen die Sensoren abgefragt werden. Wenn ein Brand erkannt wird, dann soll die Innensirene des Stockwerks, wo es brennt, und die Aussensirene angeschalten werden.

Sensoren können auch Fehler melden. Wenn ein Fehler gemeldet wird, dann soll der Hausmeister informiert werden (in unserem Fall wird einfach eine Meldung auf der Konsole ausgegeben), damit dieser einen Blick auf den Sensor werfen kann.

Grundlegende Softwarearchitektur

Die grundlegende Software-Architektur schaut folgendermaßen aus:



Public Interface

Das im folgenden gegebene öffentliche Interface soll genauso wie angegebene implementiert werden. Sie können dieses auch um weitere Funktionen erweitern, wenn Sie dies für nötig erachten.

Klasse Building

Die Klasse `Building` repräsentiert das Geäude selber und implementiert die zentrale Steuerung der Brandmeldeanlage. Sie verwaltet die Sirenen und die Sensoren, d.h. sie übernimmt für diese die Object Ownership. Es gibt genau eine Sirene pro Stockwerk und eine Aussensirene, diesen können Sie statisch zur Klasse hinzufügen (als Objektvariable). Die Sensorkonfiguration hingegen ist ständigen Änderungen unterworfen, daher soll deren Verwaltung über einen dynamische Datencontainer erfolgen. Wählen Sie selbstständig einen passenden Datencontainer aus (kleiner Tipp: verwenden Sie am besten für jedes Stockwerk einen eigenen Datencontainer).

- Objektfunktionen:
  - `Building(int floors)`: Konstruktor, dem die Anzahl der Stockwerke übergeben wird.
  - `int addSensor(int floor, Sensor* sensor)`: Fügt einen neuen Sensor zum angegebenen Stockwerk hinzu (Nummerierung startet bei 0). Der Rückgabewert ist eine ID, die den jeweiligen Sensor eindeutig im jeweiligen Stockwerk identifiziert (ID muss nur innerhalb eines Stockwerks eindeutig sein, geeignete ID selbstständig wählen).
  - `Sensor* getSensor(int floor, int id)`: Gibt den Sensor im angegebenen Stockwerk mit der angegebenen ID zurück.
  - `void deleteSensor(int floor, int id)`: Zerstört den Sensor im angegebenen Stockwerk mit der angegebenen ID.
  - `void checkSensors()`: Überprüft alle Sensoren in allen Stockwerken. Diese Funktion wird in regelmäßigen Zeitabständen aufgerufen (z.B. alle Sekunden). Der Zweck dieser Funktion ist es, Brände zu erkennen und darauf zu reagieren.
  - `void cancelAlarm()`: Schaltet alle Sirenen wieder aus.

Klasse Siren

Die Klasse `Siren` repräsentiert eine Sirene.

- Objektfunktionen:
  - `virtual void switchOn()`: Schaltet die Sirene ein (in unserem Fall Ausgabe auf der Konsole).
  - `virtual void switchOff()`: Schaltet die Sirene.

Klasse Sensor

Die Klasse `Sensor` ist die Basisklasse für alle Sensoren und definiert deren öffentliches Interface. Konkrete Sensoren werden von Subklassen dieser Klasse repräsentiert.

Wenn ein Feuer oder ein Fehler entdeckt wurde, dann soll die Funktion `checkSensor()` eine entsprechende Exception werfen.

- Objektfunktionen:
  - `virtual void checkSensor()`: Überprüft den Sensoren. Wenn ein Brand entdeckt wurde, dann soll dies mittels einer geeigneten Exception mitgeteilt werden.
  - `virtual int getType()`: Gibt den Typ des konkreten Sensors zurück.

Konkrete Sensoren und Aktuatoren

Folgende konkreten Sensoren sollen implementiert werden:

- SmokeDetector:
  - Entdeckt mit einer Wahrscheinlichkeit von 3% Feuer
  - Mit einer Wahrscheinlichkeit von 2% kann ein Fehler auftreten.
  - `getType()` gibt 1 zurück.

- InfraredSensor:
  - Entdeckt mit einer Wahrscheinlichkeit von 5% Feuer
  - Mit einer Wahrscheinlichkeit von 4% kann ein Fehler auftreten.
  - `getType()` gibt 2 zurück.

Exceptions

Die Basisklasse aller Exceptions soll `FireAlarmException` heißen, die wiederum von `std::exception` erbt. Überschreiben Sie die Funktion `const char* what()` der Klasse `std::exception`, sodass eine aussagekräftige Fehlermeldung zurückgegeben wird (alternativ können Sie auch von `std::runtime_error` erben und die Fehlermeldung dessen Konstruktor übergeben).

Falls ein Feuer detektiert wird, soll der entsprechende Sensor die Exception `FireDetectedException` werfen, und falls ein Fehler detektiert wird, soll der entsprechende Sensor die Exception `ErrorDetectedException` werfen.

Wenn Sie es für notwendig erachten, können Sie auch weitere Exception-Klassen hinzufügen.

Interne Implementierung

Wie Sie die gewünschte Funktionalität intern implementieren, bleibt Ihnen überlassen. Wählen Sie passende Sichtbarkeiten und vergessen Sie nicht, wenn möglich das `const` Keyword bei Eingabeparametern und wenn passend Call-by-Reference zu verwenden.

Wählen Sie selbstständig passende Datencontainer aus. Es darf zu keinem Zeitpunkt zu einem Speicher- oder sonstigen Ressourcenleck kommen. Sorgen Sie dafür, dass, wenn die Gebäude-Instanz zerstört wird, alle Ressourcen wieder sauber freigegeben werden.

Überlegen Sie sich außerdem, wo sie dynamische Bindung verwenden.

Separieren Sie bitte die `main()` Funktion in einer eigenen Datei, die Sie `main.cpp` nennen. Rufen Sie in der main()-Funktion in einer Schleife regelmäßig die Methode `checkSensors()` auf. Lassen Sie aus praktischen Gründen nach jeder Iteration Ihr Programm mindestens eine Sekunde schlafen (mit sleep()).

Sorgen Sie für ausreichend Debug-Ausgaben bei der Programmausführung, damit man nachvollziehen kann, was die Brandmeldeanlange gerade tut.

Die restlichen Dateien können Sie benennen, wie Sie wollen.

Geben Sie mit der Abgabe auch ein Makefile ab, welches Ihr Programm kompiliert.

Denken Sie daran, nach einem Feuer die Sirenen nach einer gewissen Zeit (Anzahl Iterationen) wieder auszuschalten.

Unverbindliche Schritt-für-Schritt Anleitung

1. Beginnen Sie mit der Implementierung der Klasse `Siren`. Testen Sie diese ausführlich.
2. Beginnen Sie mit der Implementierung der Klasse `Sensor` und deren Subklassen. Gehen Sie dabei den Vererbungsbaum von oben nach unten durch. Testen Sie diese ausführlich.
3. Implementieren Sie anschließend die Klasse `Building` und testen diese ausführlich.