**Fire alarm system**

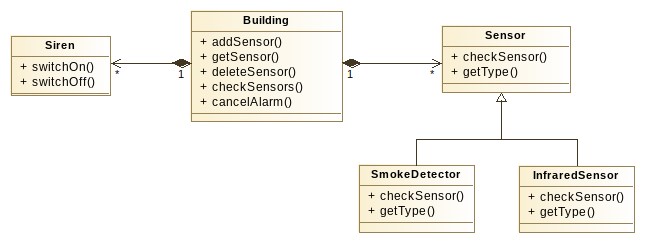
Your software company has been commissioned to develop a fire alarm system for the new MCI building. This fire alarm system consists of several sensors spread over several floors, several indoor sirens, one per floor, and an outdoor siren.

The sensors are to be queried at regular intervals. If a fire is detected, the indoor siren of the floor where it is burning and the outdoor siren should be switched on.

Sensors can also report errors. If an error is reported, then the caretaker should be informed (in our case, a message is simply issued on the console) so that he can take a look at the sensor.

# Basic Legend Software Architecture

The basic software architecture looks like this:



# Public Interface

The following public interface should be implemented in the same way as the one specified. You can also add more features to it if you deem it necessary.

## Class Building

The Building class represents the building itself and implements the central control of the building.

Fire alarm system. It manages the sirens and the sensors, i.e. it takes over the object for them.

Ownership. There is exactly one siren per floor and one outdoor siren, which you can add statically to the class (as an object variable). The sensor configuration, on the other hand, is subject to constant change, so its management is to be carried out via a dynamic data container. Choose a suitable data container yourself (small tip: it is best to use a separate data container for each floor).

• Object functions:

* Building(int floors): Constructor to which the number of floors is passed. ▪ int addSensor(int floor, Sensor\* sensor): Adds a new sensor to the specified floor (numbering starts at 0). The return value is an ID that uniquely identifies the respective sensor on the respective floor (ID only has to be unique within one floor, choose the appropriate ID yourself).
* Sensor\* getSensor(int floor, int id): Returns the sensor on the specified floor with the specified ID.
* void deleteSensor(int floor, int id): Destroys the sensor on the specified floor with the specified ID.
* void checkSensors(): Checks all sensors on all floors. This function is called at regular intervals (e.g. every seconds). The purpose of this feature is to detect and respond to fires.
* void cancelAlarm(): Turns off all sirens again.

## Class Siren

The Siren class represents a Siren. • Object functions: ▪ virtual void switchOn(): Turns on the siren (in our case output on the console).

▪ virtual void switchOff(): Turns off the siren.

## Class Sensor

|  |  |  |
| --- | --- | --- |
| The Class | Sensor | is the base class for all sensors and defines their public interface. Concrete |
| Sensors are represented by subclasses of this class. | | |

If a fire or an error has been detected, then the checkSensor() function should throw a corresponding exception. • Object functions: ▪ virtual void checkSensor(): Checks the sensor. If a fire has been detected, this should be communicated by means of a suitable exception.

▪ virtual int getType(): Returns the type of the specific sensor.

## Concrete sensors and actuators

The following specific sensors are to be implemented:

* SmokeDetector:
  + Detects Fire with a 3% chance
  + There is a 2% chance that an error may occur. ▪ getType() returns 1.
* InfraredSensor:
  + Detects fire with a 5% chance
  + There is a 4% chance that an error may occur.
  + getType() returns 2.

## Exceptions

The base class of all exceptions should be called FireAlarmException, which in turn inherits from std::exception. Override the const char\* what() function of the std::exception class so that a

(alternatively, you can inherit from std::runtime\_error and pass the error message to its constructor).

If a fire is detected, the corresponding sensor should throw the Exception FireDetectedException , and if an error is detected, the corresponding sensor should throw the Exception ErrorDetectedException .

If you deem it necessary, you can also add more exception classes.

# Internal implementation

How you implement the desired functionality internally is up to you. Choose suitable

|  |  |  |
| --- | --- | --- |
| visibility, and if possible, don't forget the | Const | Keyword at input parameters and if |
| Call-by-Reference. | | |

Select suitable data containers independently. There must be no memory or other resource leak at any time. Make sure that when the building instance is destroyed, all resources are released cleanly.

Also, consider where they use dynamic binding.

Please separate the main() function in a separate file, which you call main.cpp. In the main() function, periodically call the checkSensors() method in a loop . For practical reasons, let your program sleep for at least a second after each iteration (using [sleep())](https://man7.org/linux/man-pages/man3/sleep.3.html).

Make sure that there is enough debug output during program execution so that you can understand what the fire alarm system is doing.

You can name the rest of the files as you like.

With the submission, you should also submit a makefile that compiles your program.

Remember to switch off the sirens after a certain time (number of iterations) after a fire.

# Non-binding step-by-step instructions

1. Start by implementing the Siren class. Test them extensively.
2. Start by implementing the Sensor class and its subclasses. Go through the inheritance tree from top to bottom. Test them extensively.
3. Then implement the Building class and test it extensively.