School of ITEE
CSSE2002/7023 — Semester 2, 2020
Assignment 1
**Due: 11 September 2020 17:00**
Revision: 1.0.0


### Abstract

The goal of this assignment is to implement and test a set of classes and interfaces[1] to be used in the second assignment. You must implement the public and protected members exactly as described in the supplied documentation (no extra public/protected members or classes). Private members may be added at your own discretion.


**Language requirements**: Java version 14, JUnit 4

# Preamble

All work on this assignment is to be your own individual work. As detailed in Lecture 1, code supplied by course staff (from this semester) is acceptable, but there are no other exceptions. You are expected to be familiar with "What not to do" from Lecture 1 and `https://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`. If you have questions about what is acceptable, please ask course staff.

All times are given in *Australian Eastern Standard Time*. It is your responsibility to ensure that you adhere to this timezone for all assignment related matters. Please bear this in mind, especially if you are enrolled in the External offering and may be located in a different time zone.

# Introduction

In this assignment, and continuing into the second assignment, you will build a simple simulation of a building management system (BMS). The first assignment will focus on implementing the classes that provide the core model for the system. A building management system monitors the structures of buildings, including their internal floors and rooms. A BMS uses different types of sensors to gather contextual information about the building.

Four sensors used by the BMS are occupancy, temperature, noise, and carbon dioxide sensors. Occupancy sensors give an indication of how many people are within a given room. Temperature sensors determine the temperature (degrees Celcius) within a given room. Noise sensors determine the level of noise (dB) within a given room. Carbon dioxide sensors determine the level of carbon dixode (parts per million) within a given room. These sensors have some on-board processing capability to provide some determination about the hazard level at their location. They can also report the raw data that they collect.

A BMS needs to maintain a collection of buildings. Each building contains floors. Each floor contains rooms. The BMS needs to know what sensors are in each room. The BMS gathers data from the sensors to determine the hazard levels, and activates a fire alarm if necessary.

When implementing the assignment you need to remember that it is implementing a simulation of the BMS and not the real BMS. Interfaces are provided for the sensors to allow easy replacement of sensor implementations in the program. You will not be collecting data from real sensors but

[1] From now on, classes and interfaces will be shortened to simply "classes"

will be implementing classes that demonstrate the behaviour of sensors. They store a set of data values that are used to simulate the sensors returning different values over time.

To manage simulation of time, there is a `TimedItem` interface and a `TimedItemManager` class. Sensors implement the `TimedItem` interface, as they are items which need to react to timed events. `TimedItemManager` stores all the `TimedItem` objects in the application. It allows the application (to be implemented in assignment two) to simulate time passing, via its `elapseOneMinute` method. This method sends the `elapseOneMinute` message to all `TimedItem`s.

`TimedItemManager` is an implementation of a design pattern[2] called singleton[3]. The key idea is that a singleton prevents you from creating more than one instance of the class. To ensure this happens, the `TimedItemManager`'s constructor is private and the `getTimedItemManager` method returns a reference to the only `TimedItemManager` instance. (The first time the `getTimedItemManager` method is invoked it creates the single `TimedItemManager` instance.) The `registerTimedItem` method allows you to add `TimedItem` objects to the manager.

## Supplied Material

- This task sheet.
- Code specification document (Javadoc).[4]
- A subversion repositiory for submitting your assignment called `ass1`.[5]
- Two files (`JdkTest` and `SimpleDisplay`) are provided in your subversion repository. `JdkTest` executes a test to ensure that you are using Java 14 to build and run your assignment. `SimpleDisplay` is a simple user interface for the assignment. If you wish to use this interface you will need to uncomment some code and add the creation of your building. Any code you write in `SimpleDisplay` will not be marked. As the first step in the assignment you should checkout the `ass1` repository from Subversion. Once you have created a new project from the repository you have checked out, you should run the JdkTest to ensure you have correctly set up the project to use Java 14.

## Javadoc

Code specifications are an important tool for developing code in collaboration with other people. Although assignments in this course are individual, they still aim to prepare you for writing code to a strict specification by providing a specification document (in Java, this is called Javadoc). You will need to implement the specification precisely as it is described in the specification document.

The Javadoc can be viewed in either of the two following ways:

1. Open `https://csse2002.uqcloud.net/assignment/1/` in your web browser. Note that this will only be the most recent version of the Javadoc.
2. Navigate to the relevant assignment folder under `Assessment` on Blackboard and you will be able to download the Javadoc `.zip` file containing html documentation. Unzip the bundle somewhere, and open `doc/index.html` with your web browser.

## Tasks

1. Fully implement each of the classes described in the Javadoc.

[2] `https://refactoring.guru/design-patterns` provides a simple overview of the concept of design patterns. Follow on courses will explore design patterns in detail.

[3] `https://refactoring.guru/design-patterns/singleton` provides a reasonably detailed description of the singleton pattern.

[4] Detailed in the `Javadoc` section

[5] Detailed in the `Submission` section

2. Write JUnit 4 tests for all the methods in the following classes:
    - `Floor` (in a class called `FloorTest`)
    - `CarbonDioxideSensor` (in a class called `CarbonDioxideSensorTest`)

# Marking

The 100 marks available for the assignment will be divided as follows:

| *Symbol* | *Marks* | *Marked* | *Description* |
|---|---|---|---|
| $F$ | 50 | Electronically | Functionality according to the specification |
| $R$ | 35 | Course staff | Code review (Style and Design) |
| $J$ | 15 | Electronically | Whether JUnit tests identify and distinguish between correct and incorrect implementations |

The overall assignment mark will be $A_1 = F + R + J$ with the following adjustments:

1. If $F < 5$, then $R = 0$ and code style will not be marked.
2. If $R > F$, then $R = F$.

For example: $F = 22, R = 25, J = 17 \Rightarrow A_1 = 22 + 22 + 17$.
The reasoning here is to place emphasis on good quality functional code. Well styled code that does not implement the required functionality is of no value in a project, consequently marks will not be give to well styled code that is not functional.

## Functionality Marking

The number of functionality marks given will be

$$F = \frac{\text{Tests passed}}{\text{Total number of tests}} \cdot 50$$

Each of your classes will be tested independently of the rest of your submission. Other required classes for the tests will be copied from a working version of the assignment.

## Code Review

Your assignment will be style marked with respect to the course style guide, located under `Learning Resources > Guides`. The marks are broadly divided as follows:

| | |
|---|---|
| Naming | 7 |
| Commenting | 8 |
| Structure and Layout | 13 |
| Good Object-Oriented Practices | 7 |

*Note that style marking does involve some aesthetic judgement (and the marker's aesthetic judgement is final).*

## JUnit Test Marking

The JUnit tests that you provide in `FloorTest` and `CarbonDioxideSensorTest` will be used to test both correct *and* incorrect implementations of the `Floor` and `CarbonDioxideSensor` classes. Marks will be awarded for test sets which distinguish between correct and incorrect implementations[6]. A test class which passes every implementation (or fails every implementation) will likely get a low mark. Marks will be rewarded for tests which pass or fail correctly.

There will be some limitations on your tests:

[6] And get them the right way around

1. If your tests take more than 20 seconds to run, or
2. If your tests consume more memory than is reasonable or are otherwise malicious,

then your tests will be stopped and a mark of zero given. These limits are very generous (e.g. your tests should not take anywhere near 20 seconds to run).

### Electronic Marking

The electronic aspects of the marking will be carried out in a linux environment. The environment will not be running Windows, and neither IntelliJ nor Eclipse (or any other IDE) will be involved. OpenJDK 14 will be used to compile and execute your code and tests.

*It is critical that your code compiles.*
If one of your classes does not compile, **you will recieve zero** for any electronically derived marks for that class.

## Submission

Submission is via your subversion repository. You must ensure that you have committed your code to your subversion repository *before* the submission deadline. Code that is submitted after the deadline will **not** be marked. Details of how to commit code to your repository are described in the `SVN Guides` on Blackboard. Your repository url is:
`https://source.eait.uq.edu.au/svn/csse2002-s???????/trunk/ass1` — **CSSE2002** students
or
`https://source.eait.uq.edu.au/svn/csse7023-s???????/trunk/ass1` — **CSSE7023** students

Your submission should have the following internal structure:

| | |
|---|---|
| `src/` | folders (packages) and `.java` files for classes described in the Javadoc |
| `test/` | folders (packages) and `.java` files for the JUnit test classes |

A complete submission would look like:

```
src/bms/building/Building.java
src/bms/display/SimpleDisplay.java
src/bms/exceptions/DuplicateFloorException.java
src/bms/exceptions/DuplicateRoomException.java
src/bms/exceptions/DuplicateSensorException.java
src/bms/exceptions/FireDrillException.java
src/bms/exceptions/FloorTooSmallException.java
src/bms/exceptions/InsufficientSpaceException.java
src/bms/exceptions/NoFloorBelowException.java
src/bms/floor/Floor.java
src/bms/room/Room.java
src/bms/room/RoomType.java
src/bms/sensors/Sensor.java
src/bms/sensors/HazardSensor.java
src/bms/sensors/CarbonDioxideSensor.java
src/bms/sensors/NoiseSensor.java
src/bms/sensors/OccupancySensor.java
src/bms/sensors/TemperatureSensor.java
src/bms/sensors/TimedSensor.java
src/bms/util/FireDrill.java
src/bms/util/TimedItem.java
src/bms/util/TimedItemManager.java
test/bms/floor/FloorTest.java
test/bms/sensors/CarbonDioxideSensorTest.java
```

Ensure that your assignments correctly declare the package they are within. For example, `CarbonDioxideSensor.java` should declare `package bms.sensors`.

**Do not** submit any other files (e.g. no `.class` files). Note that `FloorTest` and `CarbonDioxideSensorTest` will be compiled without the rest of your files.

## Prechecks

Prechecks will be performed on your assignment repository multiple times before the assignment is due. They will assess whether your folders and files are in the correct structure and whether your public interface aligns with the expected public interface. **Successfully passing a precheck does not guarantee any marks. No functionality or style is assessed.**

```
Precheck #1: Approximately 5pm on 25 August 2020.
Precheck #2: Approximately 5pm on 31 August 2020.
Precheck #3: Approximately 5pm on 4 September 2020.
Precheck #4: Approximately 5pm on 8 September 2020.
```

Please endeavour to have code written and in your repository before at least one of these prechecks in order to make the most of them. No additional prechecks will be run for people who did not start the assignment in time, neglected to commit their code to their repository, or received extensions past the due date. Prechecks are valid only for currently released version of the Javadoc, if an update is made it may invalidate the precheck results.

## Late Submission

Assignments submitted after the submission deadline of 17:00 on September 11 2020, will recieve a mark of zero unless an extension is granted as outlined in the ECP — see the ECP for details.

Do not wait until the last minute to commit the final version of your assignment. A commit that starts before 17:00 but finishes after 17:00 will not be marked.

## Remark Requests

To submit a remark of this assignment please follow the information presented here: `https://my.uq.edu.au/information-and-services/manage-my-program/exams-and-assessment/querying-result`.

## Revisions

If it becomes necessary to correct or clarify the task sheet or Javadoc, a new version will be issued and an announcement will be made on the Blackboard course site.