

COMP201: Software Engineering I

Assignment 1.2 (2023/24)

(100% mark for Assignment 1.2 is 5% of COMP201 grade)

Deadline for Assignment 1.2: 12nd of December 2023, 17:00

OBJECTIVE

This assignment is mainly about “design/implementation”.
You should be implementing a simulation of part of the security system. This simulation will follow the requirements that have been defined in coursework 1 and follow the use cases that you defined. This simulation will not be a full implementation of the original requirement but only implement the card/code entry part of the system. For this case, there is no UI code required. You are strongly encouraged to start this work as soon as possible.

The purpose of this exercise is to understand the challenges of implementing a design from requirements and the state modelling required. This code will be implemented using a simulated version of the hardware, it would then be possible later to implement a security system by replacing the simulation code with software drivers connected to real hardware.

Assignment number	1 of 2 (part 2)
Weighting	5%
Assignment Circulated date provided to class	2/11/2023
Deadline Day & Date & Time	12th of December 2023 at 17:00
Submission Mode	Electronic submission/Canvas
Learning outcome assessed	<ol style="list-style-type: none">1. Realise the problems in designing and building significant computer systems2. Understand the need to design systems that fully meet the requirements of the intended users3. Be able to apply these principles in practice4. Be able to demonstrate how to effectively implement an OO design in an O-O language such as Java or Python.
Submission necessary in order to satisfy Module requirements	No

Purpose of assessment	To assess the students ability to analyse a problem and implement it in code
-----------------------	--

Marking criteria	See end of document
Late Submission Penalty	Standard UoL Policy

Description of problem

Produce code to support a security system simulation in Java using the existing code as a base. (This has been made available to you as a zip file).

NOTE You only need to fulfil the requirements of this coursework not the whole of the requirement detailed in coursework 1.1.

Card handling

All the code you will implement is done as part of files Authenticator.java and Card.java, you will need to modify both these files. It is very important you do **NOT modify the public interface of these** Java classes.

Look at the comments in the source files for information on what has to be done. Everywhere there is a TO DO comment, please complete the code as requested.

checkFireCode

The behaviour should be as follows, when calling checkFireCode().

If the card status is CARD_NEW, then the method should return INVALID_CARD.

If the card is locked, it will return CARD_LOCKED. If the code is correct, the method returns the OK status. If the code is invalid the method returns INVALID_FIRE_CODE.

If the code is incorrect, the method returns BAD_FIRE_CODE. Getting the code wrong 3 times will lock the card and return CARD_LOCKED. Using an invalid code, does **NOT** affect the lock count.

checkBurglaryCode

The behaviour should be as follows, when calling checkBurglaryCode().

If the card status is CARD_NEW, then the method should return INVALID_CARD.

If the card is locked, it will return CARD_LOCKED. If the code is correct, the method returns the OK status. If the code is invalid the method returns INVALID_BURGLARY_CODE.

If the code is incorrect, the method returns `BAD_BURGLARY_CODE`. Getting the code wrong 3 times will lock the card and return `CARD_LOCKED`. Using an invalid code, does **NOT** affect the lock count.

Notice for either method, any incorrect code (fire or burglar alarm code) will increase the lock count, invalid codes do not change the lock count. A correct code will set the lock count back to zero.

For this functionality you will have to add behaviour to the card simulator which allows it to count the wrong code count and save it persistently.

Marking criteria

This code will be marked using automatic testing. The structure and format of the code will **NOT** be marked for this assignment, **however** you are strongly encourage to format and comment your code correctly. The code will be marked be using a number of test cases, for each test case passed, a point will be awarded. All the behaviour described in this document as well as the comments in the code in `Card.java` and `Authenticator.java` must be covered to gain full marks. **Code which does not compile may be awarded zero marks, please do not submit code which contains compilation errors.**

Notes on code

This code design shows two approaches to class design which are commonly used in OO systems, data driven design and responsibility driven design. `Card.java` is an example of a class which results from data driven design, the focus of objects of this class is to model, store and validate the data contained within a card. `Authenticator.java` is an example of a class which results from responsibility driven design, instances of this class are responsible for the action or responsibility of authentication.