Built Environment, Engineering and Computing

# Assessment Brief

## Component 1

| | | | | | |
|---|---|---|---|---|---|
| **Module name and CRN** | Software Systems Development CRN:14242 | | | | |
| **Module Leader** | Mark Dixon | | | | |
| **Term** | 2 | **Level** | 5 | **Approx No of Students** | 110 |

**COMPONENT TITLE:** Programming Assignment

**COMPONENT WEIGHTING:** 50% of Module Marks

**HAND-OUT DATE:** Week 1

**SUGGESTED STUDENT EFFORT:** 30 hours

**SUBMISSION DATE:** Sunday 28th Feb 2021 (23:59)

**SUBMISSION INSTRUCTIONS:**

*Submission of program code should be via the VLE. Your solutions should be zipped up into a single file in the form `<studentID>.zip`. You may be contacted following submission to either demonstrate or discuss your solution with a tutor.*

**FEEDBACK MECHANISM:**

*You will be provided with feedback regarding your work following submission and marking.*

**LEARNING OUTCOMES ADDRESSED BY THIS COMPONENT:**

| | |
|---|---|
| Learning outcome 1 | Demonstrate the ability to implement program code which follows good practice in terms of use of design concepts, exception handling mechanisms and reusable components. |
| Learning outcome 2 | Have the ability to demonstrate the application of fundamental collection based programming data structures and manipulation techniques. |

**NOTES:**

**This is an individual assessment. Submission of an assessment indicates that you, as a student, have completed the assessment yourself and the work of others has been fully acknowledged and referenced.**

**By submitting for this assessed work, you are declaring that you are fit to submit, and you will therefore not normally be eligible to submit a request for mitigation for this work.**

**If you fail to perform any requested demonstration or discussion at the scheduled date and time without agreed mitigation, you will be given one further opportunity to demonstrate or discuss your work (incurring a 5% late penalty) at a time scheduled by the module team. If you miss this second opportunity, your result will be recorded as Non-Submission.**

**If your result for this component is recorded as Non-Submission or your mark for this component and for the whole module is below 40%, you will have opportunity to take reassessment with a submission date of summer 2021 (5th July) and your mark capped at 40% (see Reassessment information below). If you are granted deferral through the mitigation process, you may complete the reassessment with a full range of marks available.**

**For further information, please refer to your Course Handbook or University Assessment Regulations.**

## DETAILS OF ASSESSMENT

**Component 1 – 50% Weighting**

For this assignment you are going to complete the development of an existing system that is currently only partially complete. All code to be edited is located in the ‘`manager`’ package.

Download the **SSDAssignment.zip** project file from the VLE. Import this into Eclipse by selecting File | Import | “General” | “Existing Projects into Workspace”. Then tick the “Select archive file:” option and point at the downloaded zip file.

To aid in development and testing the initial system also includes several JUnit test classes, which you will use during development to confirm your solution matches the specified requirements. You will therefore be applying a Test Driven Development (TDD) approach.

Whether a test passes or fails must be determined by the program code within the system itself. i.e. YOU MUST NOT CHANGE the provided JUnit test code in any way (i.e. do not update any code in the ‘test’ package).

Once imported into Eclipse, examine the code and get familiar with the existing classes, attributes and methods. Also run the JUnit tests, at this point all tests should fail. As you complete more of the assignment, more and more tests should start to pass.

The specification has been split into three distinct parts. Each of these requirements has an associated JUnit test class, allowing you to explicitly test each independently. A JUnit Test Suite is also provided, which tests all requirements at once. To run a specific JUnit test from within the Eclipse IDE, right click on the filename, e.g. `Part1Test.java` then select “Run As” | “JUnit Test”.

In order to aid your understanding of the system, a UML Class Diagram is shown in Appendix A. This can be used to help understanding of the overall system, and to identify required attributes and the associations between classes.

Ensure examine the `TODO` comments within the code to help identify which methods need completing in order to pass the tests for each part (these often include a part number). Additionally, ensure you read the Javadoc comments provided within the code to help establish the purpose of the various methods. Finally, you may sometimes find it useful to examine the test code itself, so the expected behaviour of the called methods can be determined.

**Part 1 – Basic Creation and Manipulation (10 Tests)**

The first requirement is to complete some missing basics from the existing system.

In order to pass the tests defined within the `Part1Test` class, complete the following tasks -

- Add any missing attributes
- Complete implementation of constructors correctly (i.e. set attribute values)
- Complete implementation of getter and setter methods.

Look for the `TODO:Part1` comments along with the UML Class Diagram in order to help you identify missing attributes and method content.

**Part 2 – Implementing an Interface (10 Tests)**

The second requirement is to complete implementation of the methods declared within the `SecuredAccess` Interface. The two classes which implement this interface are `Room` and `Office`, however they implement the methods in slightly different ways (see below).

In order to pass the tests defined within the `Part2Test` class, implement the missing code in the `Room` and `Office` classes, the methods to be completed are -

- `checkCode()`
- `setCode()`
- `resetToDefault()`
- `isLockedOut()`
- `getIncorrectAttempts`

For the `Room` class the following behaviour should be implemented.

- When a `Room` object is first created, or `resetToDefault()` is called, the stored code should be set to "`9999`".
- Calls to `isLockedOut()` should always return `false`.
- Calls to `getIncorrectAttempts()` should always return `0`.

For the `Office` class the following behaviour should be implemented.

- When an `Office` object is first created, or `resetToDefault()` is called, the stored code should be set to "`1234`".
- When `setCode()` or `resetToDefault()` is called the `incorrectAttempts` count should be reset to `0`.
- The `isLockedOut()` method should return true if the `incorrectAttempts` count is greater than `5`.
- Calls to `getIncorrectAttempts()` should return the `incorrectAttempts` count value.
- When `checkCode()` is called, it should only return `true` if the given code matches the stored code, and `isLockedOut()` returns `false`. It should also reset the `incorrectAttempts` count to `0` when `true` is returned, or increment this count otherwise.

Look for the `TODO:Part2` comments within the code to help you identify the methods to be implemented.

**Part 3 – Implementing Associations (5 Tests)**

The final part is to implement code which ensures the tests pass related to manipulation of associations.

In order to pass the tests defined within the `Part3Test` class, implement the missing code within the following methods -

Within the `Property` class, complete implementation of -

- `setTenant()`
- `removeTenant()`
- `hasTenant()`

Within the `PropertyOwner` class, complete implementation of -

- `addProperty()`
- `removeProperty()`
- `clearOwnedProperties()`
- `getPropertyCount()`

Within the `Room` class, complete implementation of -

- `setOccupant()`
- `removeOccupant()`
- `hasOccupant()`

Within the `Hotel` class, complete implementation of -

- `occupyRoom()`
- `freeRoom()`
- `getOccupiedRoomCount()`

Look for the `TODO:Part3` comments within the code to help you identify the methods to be implemented.

## MARKING SCHEME / CRITERIA

The solution you complete will be mainly assessed via the number of JUnit tests which are passed. **Each passed test will contribute 2 marks to the final grade** (up to 50 marks).

A maximum mark for the submission will only be awarded if it fully complies with the specification and you are able to discuss your solution with the tutor in a knowledgeable way if requested to do so. The general marking criteria is shown below.

Maximum of 50 marks are available.

| **Mark** | Criteria |
|---|---|
| 0-10 | No program is submitted; the program does not compile or the student is incapable of discussing the submitted work.<br><br>*Fewer than 5 of the JUnit test cases pass.* |
| 10-20 | A program is submitted which can be generally explained by the student, but the program provides only a minimal solution to the specification.<br><br>*Upto 10 of the JUnit test cases pass.* |
| 20-30 | A program is submitted which can be explained by the student. The specification is mostly satisfied by the solution, but may include some minor issues, bugs or omissions. Commenting or indentation may lack accuracy.<br><br>*Upto 15 of the JUnit test cases pass.* |
| 30-40 | A program is submitted which can be explained by the student. The specification is almost fully satisfied by the solution Commenting and indentation is generally accurate.<br><br>*Upto 20 of the JUnit test cases pass.* |
| 40-50 | A fully commented and correctly indented program is submitted which can be fully explained by the student. The specification is fully satisfied by the solution.<br><br>*Upto 25 of the JUnit test cases pass.* |

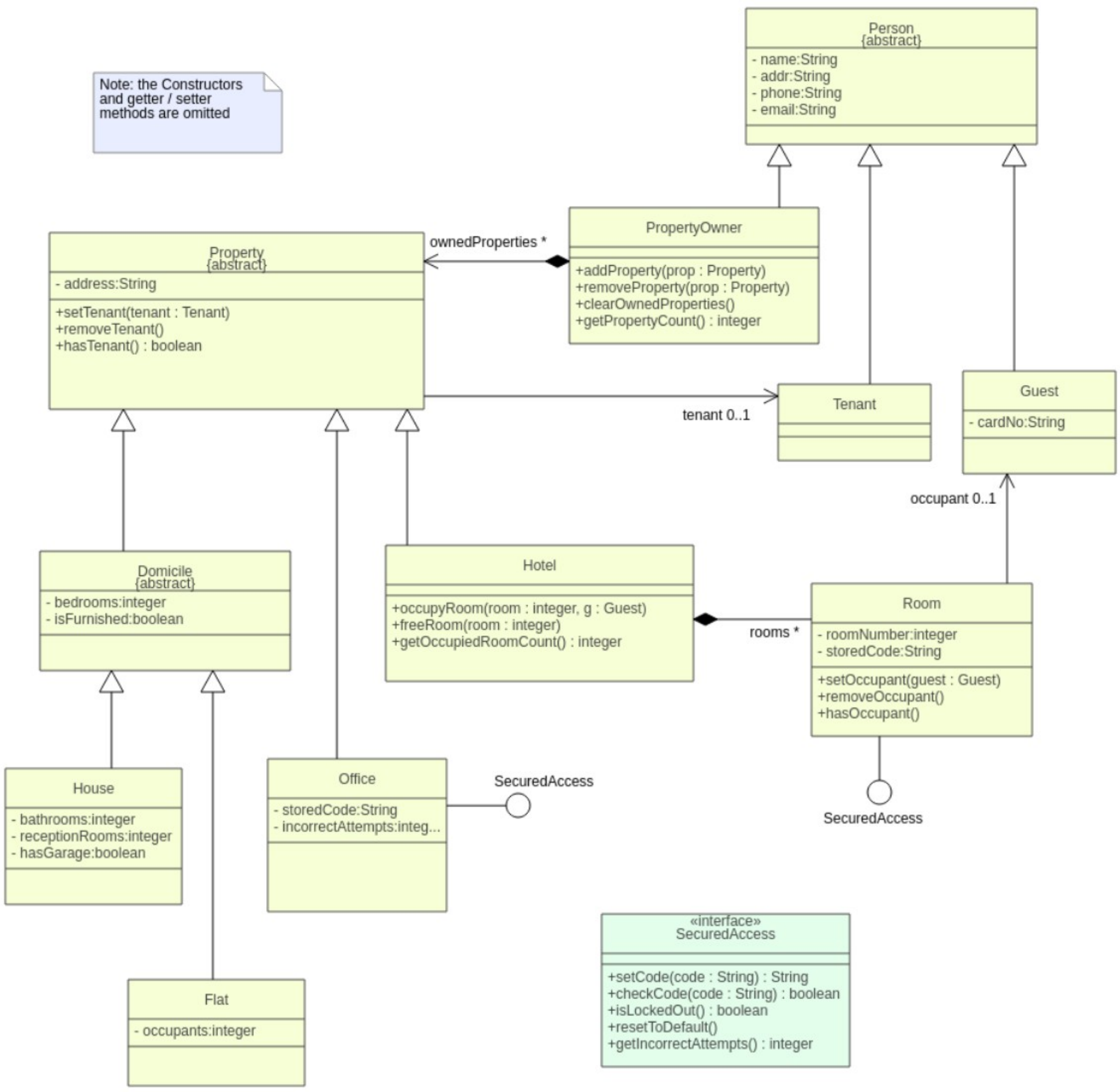
*Note: The submission of a working program that satisfies the specification does not in itself guarantee marks. If requested to do so, a student must have the ability to demonstrate, discuss and explain their submitted work.*

## REASSESSMENT and DEFERRAL OPPORTUNITIES

Reassessment will take the form of re-submission, based on the original assignment specification with a submission date of summer 2021 (deadline the 5th July).

Upon submission of reassessment you must contact the module leader in order to arrange a demonstration. Details will be made available on the VLE.

## Appendix A – UML Class Diagram


Note: the Constructors and getter / setter methods are omitted
Person {abstract}
- name:String
- addr:String
- phone:String
- email:String
PropertyOwner
+addProperty(prop : Property)
+removeProperty(prop : Property)
+clearOwnedProperties()
+getPropertyCount() : integer
ownedProperties *
Property {abstract}
- address:String
+setTenant(tenant : Tenant)
+removeTenant()
+hasTenant() : boolean
tenant 0..1
Tenant
Guest
- cardNo:String
occupant 0..1
Domicile {abstract}
- bedrooms:integer
- isFurnished:boolean
Hotel
+occupyRoom(room : integer, g : Guest)
+freeRoom(room : integer)
+getOccupiedRoomCount() : integer
rooms *
Room
- roomNumber:integer
- storedCode:String
+setOccupant(guest : Guest)
+removeOccupant()
+hasOccupant()
SecuredAccess
House
- bathrooms:integer
- receptionRooms:integer
- hasGarage:boolean
Office
- storedCode:String
- incorrectAttempts:integ...
SecuredAccess
Flat
- occupants:integer
«interface» SecuredAccess
+setCode(code : String) : String
+checkCode(code : String) : boolean
+isLockedOut() : boolean
+resetToDefault()
+getIncorrectAttempts() : integer