# Sprint 1 Presentation

* It is preferred that all team members be present for the presentation.
* Your completed Sprint 1 Report should be in your GitHub docs folder by 7am on July 11 so that I can look at it before the presentation.
* Before your presentation begins, have these things open: your code in an IDE, your Sprint 1 Report, your class diagram.
* The presentation will be interactive, meaning I’ll stop you and ask questions.
* You’ll begin by doing these things:

1. Explain your design. Show your class diagram(s) and explain. Do: (a) explain what the major purpose is of each class (“this class represents…”, (b) how classes are related (has-a, or has-many, or is-a), (c) any major decision choices you made, if applicable. Don’t: (a) mention every method and instance variable – you can mention a method or two, if that helps explain things.
2. Show me 3 interesting unit tests and briefly explain.
3. Demo your system for about 10 minutes. State the user story and then demo. Repeat. You don’t have to do them all, but be prepared if asked.

Note: Your objective in a demo is to clearly show that your system works as intended. Thus, do NOT move too quickly, make sure the audience (me) is following, remind if things need to be remembered to understand a current step, etc. You’ve likely run your software a zillion times and move very, very quickly through the software when you are running it to test. Don’t do that in a demo!

**July 11th**

**Document 03 – Sprint 1 Report**

CS 4321 – Summer 2022

Contents

[1 Sprint 1 Backlog 1](#_30j0zll)

[2 Class Diagram 2](#_1fob9te)

[3 System Tests 2](#_3znysh7)

[4 Sprint 1 Presentation 2](#_2et92p0)

# Sprint 1 Backlog

| **Deliverable**  Develop a set of user stories that you can accomplish in the form of a working system by the end of the sprint (July 11). Note that “Status” is one of these values:   | **Status** | **Description** | | --- | --- | | Complete | System tests verify that the story has been implemented correctly. | | Buggy | Sometimes works according to system tests | | Incomplete | Started but not completed | | Not Implemented | No code written. | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

| US Number | 1 – Manager Complete |
| --- | --- |
| User Story | As a manager, I want to manage employee access so that I can regulate hotel access |
| Requirement | 3 |
| Notes | Class is complete |

| US Number | 2 – Employee Complete |
| --- | --- |
| User Story | As an employee, I want to access parts of the hotel, so that I can do my job |
| Requirement | 3 |
| Notes | We still need to develop the tests |

| US Number | 3 – AccessManager Incomplete |
| --- | --- |
| User Story | As a security guard, I want to monitor access to the building so that I can prevent break-ins |
| Requirement | 3 |
| Notes | Code is not fully tested |

# Class Diagram

| **Deliverable**  At the conclusion of this sprint, you should make a class diagram using StarUML. You can make multiple diagrams at different levels of granularity, or just break it up. Your objective is to convey your design to me, thus some brief remarks would be useful. The diagram(s) must be readable. Also include the diagrams saved as image files in your docs folder on GitHub. |
| --- |

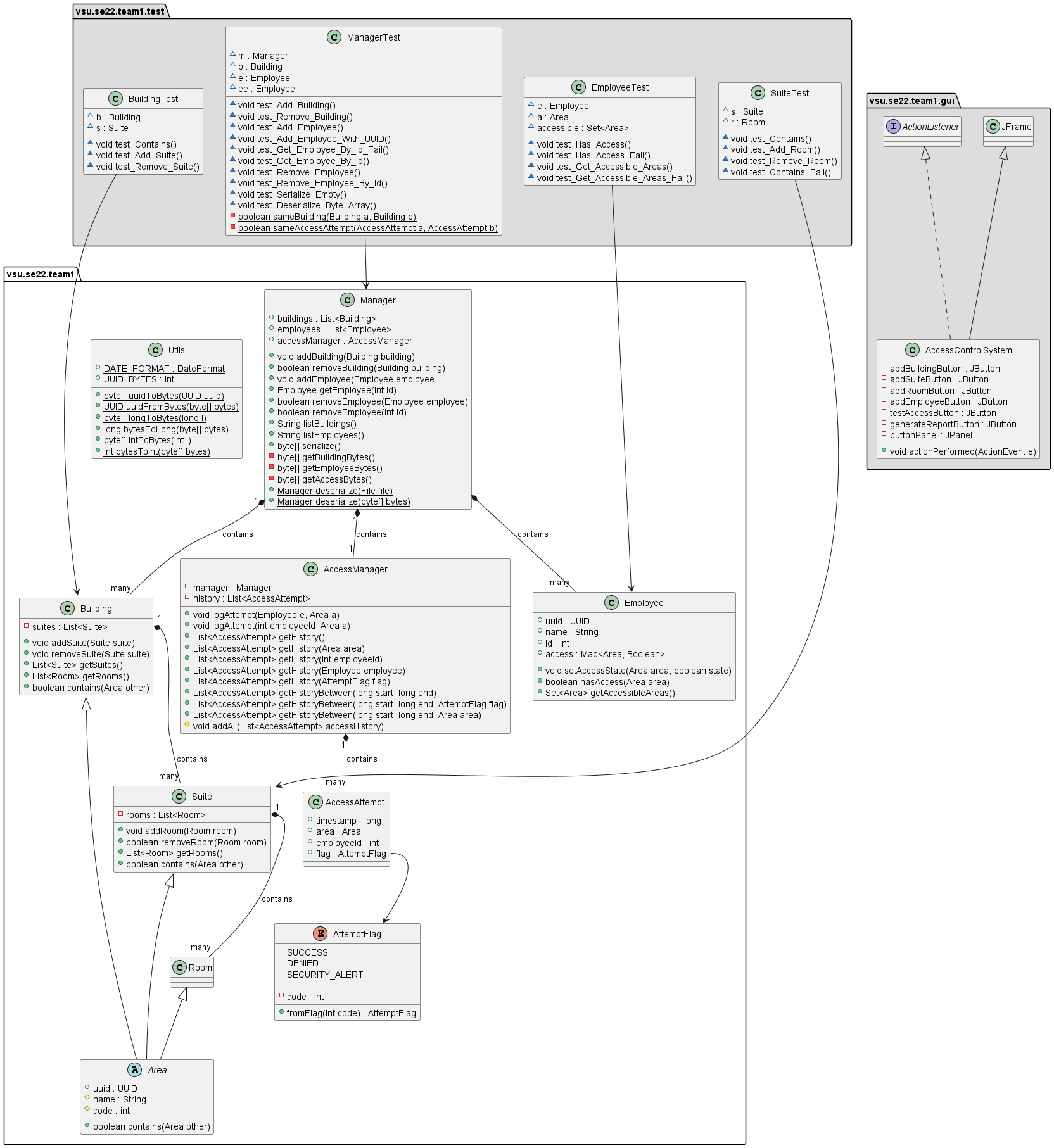
The Manager class represents the main class of the program. It contains methods for creating and removing buildings and employees, printing statistics, and reading/writing the data to the disk.

The Building, Suite, and Room classes all extend the abstract Area class. Buildings have methods for adding or removing Suites, while Suites have methods for adding or removing Rooms.

The Employee class contains a list of access attempts the employee has made, as well as a map representing which Areas the employee has access to. The class also has methods for changing the employee's access to a given Area, listing the Areas the employee has access to, and printing statistics about the employee's access history.

The AccessAttempt class stores metadata about each access attempt that is made.

The AttemptFlag enum represents the result of an AccessAttempt.



# System Tests

| **Deliverable**  You probably will have multiple System Tests for each user story. |
| --- |

| Test Number | 1 |
| --- | --- |
| US Number | 1 |
| Description | Tests the getEmployee method that uses the employees’ ID. |
| Status | Success |

| Test Number | 2 |
| --- | --- |
| US Number | 1 |
| Description | Intentionally fails the getEmployee method that uses an employee’s ID. The output looked the same, but was not. |
| Status | Success |

| Test Number | 3 |
| --- | --- |
| US Number | 2 |
| Description | Tests hasAccess to ensure that employees can access areas of the building |
| Status | Success |

| Test Number | 4 |
| --- | --- |
| US Number | 2 |
| Description | Intentionally fail getAccessibleAreas to ensure that an employee cannot access an unauthorized area |
| Status | Success |