CP1A - Office Space Allocation

Requirements

To complete this checkpoint you must have completed Modules 1, 2, 3 and 4.

Problem Description

In this exercise you will be required to model a room allocation system for one of Andela's facilities called Amity.

Constraints

Amity has rooms which can be offices or living spaces. An office can occupy a maximum of 6 people. A living space can inhabit a maximum of 4 people.

A person to be allocated could be a fellow or staff. Staff cannot be allocated living spaces. Fellows have a choice to choose a living space or not.

This system will be used to automatically allocate spaces to people at random.

Task 0 - Model the system

For this task you are required to model the entire system using class definitions. Create the following classes and implement the required functionality within them - `Person`, `Fellow`, `Staff`, `Amity`, `Room`, `Office`, `LivingSpace`.

Ensure that you are properly structuring your class files and using easily navigable folder structures.

You should have UML Class Diagrams for your system. Keep the diagrams in a `/designs` folder.

Task 1 - Write Tests for the models

Create tests for the models you just created in the previous exercise that ensure the functionality created in the classes are working.

You are then advised to schedule a Test-Cases review with your Simulations Facilitator before you can proceed with Task 2.

Task 2 - Create a command line interface

After reading through this task, jump to task 2.5 and then come back here.

For this task you are required to create a command line interface using docopt that does the following using your models.

1. `create_room <room_name>...` - Creates rooms in Amity. Using this command I should be able to create as many rooms as possible by specifying multiple room names after the `create_room` command.
2. `add_person <person_name> <FELLOW|STAFF> [wants_accommodation]` - Adds a person to the system and allocates the person to a random room. `wants_accommodation` here is an optional argument which can be either `Y` or `N`. The default value if it is not provided is `N`.
3. `reallocate_person <person_identifier> <new_room_name>` - Reallocate the person with `person_identifier` to `new_room_name`.
4. `load_people` - Adds people to rooms from a txt file. See Appendix 1A for text input format.
5. `print_allocations [-o=filename]` - Prints a list of allocations onto the screen. Specifying the optional `-o` option here outputs the registered allocations to a txt file. See Appendix 2A for format.
6. `print_unallocated [-o=filename]` - Prints a list of unallocated people to the screen. Specifying the `-o` option here outputs the information to the txt file provided.
7. `print_room <room_name>` - Prints the names of all the people in `room_name` on the screen.

8. `save_state [--db=sqlite_database]` - Persists all the data stored in the app to a SQLite database. Specifying the `--db` parameter explicitly stores the data in the `sqlite_database` specified.
9. `load_state <sqlite_database>` - Loads data from a database into the application.

Task 2.5 - Write Tests for all functionality created

For this project you are expected to use TDD. What this means is that you will be writing tests before you create functionality. Just as a rule of thumb before creating any function in the previous task, ensure you have written at the very least, an empty test for it.

Ensure you have test coverage of over 70% at the end of the project.

You may now proceed with Task 2.

Appendix 1A - Sample Input Format

```
OLUWAFEMI SULE FELLOW Y
DOMINIC WALTERS STAFF
SIMON PATTERSON FELLOW Y
MARI LAWRENCE FELLOW Y
LEIGH RILEY STAFF
TANA LOPEZ FELLOW Y
KELLY McGUIRE STAFF
```

Appendix 2A - Sample Output Format

```
ROOM NAME
-----------------------------------
MEMBER 1, MEMBER 2, MEMBER 3

ROOM NAME
-----------------------------------
MEMBER 1, MEMBER 2
```

Task 3 - Have any ideas?

You are free to add on extra functionality beyond what's specified in the specifications. You are encouraged to do this to exceed expectations.

Assessment Rubric

Criterion	Does not Meet Expectation	Meets Expectations	Exceed Expectations
Code Functionality	The code does not work in accordance with the ideas in the problem definition.	The code meets all the requirements listed in the problem definition.	The code handles more cases than specified in the problem definition.
Comments	Solution is not commented.	Solution contains adequate comments.	Solution uses doc style comments and is self documenting.
Code Readability	Code is not easily readable or is not commented. The names for variables, classes, and procedures are inconsistent and/or not meaningful. Negligence of style guides.	Code is easily readable and necessarily commented. The names for variables, classes, and procedures are consistent and/or meaningful. Style Guides are adhered to.	
OOP Usage	Solution did not use OOP or does not use OOP properly by not modelling required objects as required.	Solution made use of OOP according to the requirement of the assignment and does so in the appropriate fashion.	
Test Coverage	Solution did not attempt to use TDD	70% test coverage	100% test coverage or 0% test coverage like a Bawse.
Defense	Cannot clearly articulate why they did what they did or why certain portions of their code behaves in a certain way. Does not understand underlying concepts/ Copied and pasted code.	Understands exactly what they did and is able to clearly communicate that.	In addition to knowing and explaining exactly what pieces of their code works, they can also articulate how the source code of libraries used influences code behavior.