Desk booking app

Use case (criteria 1)
This app is designed for offices to use to help keep track of employee numbers in the office on any given day.
Hot desking is the principle of desks not belonging to any one employee. A desk on any particular day can be used
by any particular employee. Hot desking is becoming more and more prevelant in covid times due to rapid expansion
of industries that allow working from home. To allow hot desking to work properly, however, a system of knowing
who is coming into the office on any particular day is needed, so as to prevent an over populated office.

The app will use object oriented principles to make an office that can hold a certain number of office desks in an
open floor plan. An office will have a minimum of 2 sub-rooms, and a minimum of 1 desk in the open floor plan. The
2 sub rooms belong to the CEO and the Human Resources manager (high level employees get their own office). The desks
can be booked by employees and project managers. The types of employees include CEO, HR manager, Manager, and employee;
each of which can only book into particular spaces.

Classes (criteria 2, 3 & 4)
Classes for this project include office type objects (Office, room, desk, etc) and Employee type objects (CEO, HR,
etc.). The office type objects include
- Office
- Room (aggregated by Office)
- Desk (abstract, aggregated by office)
- Manager_Desk (inherits from Desk)
- Employee_Desk (inherits from Desk)

The employee type objects include
- Worker (abstract)
- CEO (inherits from Worker)
- Manager (inherits from Worker)
- Emplopyee (inherits from Worker)
- Project (aggregated by Workers)

Data types for office type objects include things like
- Business_name
- *desks
- Occupied (bool)

Data types for Worker include things like
- name
- Allocated_Project(s)
- Project_Cost
- Salary

Timeline (criteria 5)
Week 9 -> 10: define all classes and set-up relationships (plus minor integration testing as progress is made)
Week 10 -> 11: Makefile construction, testing, and automated inputs
Week 10 -> 11: Extra alloted time in case earlier goals are not met

User interface (criteria 6)
The app is designed to be a command-line application, that when complete, can be run using automated inputs (Makefile)
or manual command line interaction with the code.

Unit testing (criteria 7)
De-bugging will be relatively intuitive, where each component as it is written will be integration tested before
continuing onto the next class member. As the classes largely just hold data, the unit tests will largely just be
integration tests that will be written out in a form where a future user can see the test coverage.