Assignment 2

Refer to Canvas for assignment due dates for your section.

Objectives[1]:

- Write classes using inheritance.
- Implement and test data validation.
- Implement custom exceptions to inform calling code of failed validation.
- Create UML diagrams.

General Requirements

Create a new Gradle project for this assignment in your course GitHub repo. Make sure to follow the instructions provided in “Using Gradle with IntelliJ” on Canvas.

Create a separate package for each problem in the assignment. Create all your files in the appropriate package.

To submit your work, push it to GitHub and create a release. Refer to the instructions on Canvas.

Your repository should contain:

- One .java file per Java class.
- One .java file per Java test class.
- One pdf or image file for each UML Class Diagram that you create. UML diagrams can be generated using IntelliJ or hand-drawn.
- All non-test classes and non-test methods must have valid Javadoc.

Your repository should **not** contain:

- Any .class files.
- Any .html files.
- Any IntelliJ specific files.

Problem 1

You are tasked with writing code that will be part of a hotel’s booking system. Your responsibility is to write classes to represent the various types of room the hotel has: single rooms, double rooms, and family rooms. All rooms have:

[1] Tip: Use the objectives to make sure you’re using course concepts. If a concept is listed in the objectives, you should be using it in your solution for at least one of the problems, often *all* of the problems, depending on the assigned tasks. The objectives list is not exhaustive—it focuses on concepts introduced in the most recent lecture. Concepts listed on past assignments will often carry over.

- A maximum occupancy - This is the maximum number of people that can stay in the room.
- A price - The cost of a single night's stay. Must be greater than 0.
- A number of guests - The number of guests currently booked into the room. This value should be 0 when the room is first created in the system.

The difference between each type of room (single, double, and family), is the maximum occupancy:

- Single rooms can sleep 1 person only.
- Double rooms can sleep a maximum of 2 people.
- Family rooms can sleep a maximum of 4 people.

All rooms should have the following functionality:

- A method, `isAvailable`, that checks if the room is available or not. The room is considered available if the number of guests is 0. If there are 1 or more guests, the room is not available.
- A method, `bookRoom`, that takes one parameter: the number of guests that would like to stay in the room. If the booking is valid (the room is available and the number of guests passed to the method is greater than 0 but does not exceed the maximum occupancy), set the room's number of guests.

1. Design and implement classes to represent the rooms as described above, as well as any additional classes necessary to implement all functionality.
2. Write unit tests to achieve 100% coverage of your code on this problem. *Note: test coverage is part of the standard design rubric for each homework. For this problem, additional points will be awarded for 100% coverage.*
3. Generate a UML diagram.

Problem 2

You are tasked with writing some code that will be part of a system managing package lockers (similar to Amazon Hub lockers). Your code will need to represent mail items, lockers, and recipients (the person the mail is addressed to).

A recipient has:

- A first name
- A last name
- An email address. You do not need to validate the format of an email address for this assignment.

A mail item has:

- A width in inches, an integer greater than or equal to 1.
- A height in inches, an integer greater than or equal to 1.

- A depth in inches, an integer greater than or equal to 1.
- A recipient.

A locker has:

- A maximum width in inches, an integer greater than or equal to 1.
- A maximum height in inches, an integer greater than or equal to 1.
- A maximum depth in inches, an integer greater than or equal to 1.
- A mail item—the item stores in the locker, if any. If there is no mail in the locker, this field should be set to null. You can assume that when a locker is first created, it is empty.

1. Write classes to represent a recipient, a mail item, and a locker as described above.
2. Write a `void` method, `addMail`, that consumes a mail item and stores it in a locker with two exceptions: if the locker is occupied (it already contains a mail item) or the mail item exceeds the dimensions of the locker (any single dimension of the mail item is bigger than the locker), the mail item should not be added to the locker.
3. Write a method, `pickupMail`, that takes one parameter, a recipient. This method will remove *and return* the mail item from the locker under the following conditions: there is a mail item in the locker AND the recipient passed to `pickupMail` matches the recipient of the mail item.
4. Generate a UML diagram and write unit tests.