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 small catering company’s booking system. The company offers set menus for lunch and dinner events. The system will keep track of the

[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.

events the company is catering for each meal time on a given day. All events (lunch and dinner) should include:

- The name of the client.
- The number of people who will be attending the event.

In addition to the general event information given above, each type of event has the following requirements and specifications:

- Lunch event:
    - The number of attendees must be between 15 and 90.
    - The number of sandwiches to provide, calculated as 1.05 times the number of attendees, rounded to the nearest whole number.
- Dinner event:
    - The number of attendees must be between 10 and 50.
    - The number of non-vegetarian entrees to provide, calculated as 80% of the number of attendees, rounded up to the nearest whole number.
    - The number of vegetarian entrees to provide, calculated as 20% of the number of attendees, rounded up to the nearest whole number.

The company can only cater one event per meal time per day i.e. one lunch event and one dinner event.

1. Design and implement classes to represent the meals as described above, being mindful of constraints on each meal type.
2. Implement a class called `Schedule` that tracks the events booked for each meal on a single day. You do not need to worry about representing dates in this assignment, but `Schedule` should have a way to track one lunch event, if any, and one dinner event, if any.
3. In `Schedule`, implement methods for booking lunch and dinner events. Your code should not allow a new event to be created for a particular meal time if there is already a booking for that event type. For example, if there is already a lunch event in the schedule, it should not be possible to overwrite the existing event.
4. Generate a UML diagram and write unit tests.

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.