CSci 3081W: Program Design and Development
Homework 2 – UML Basics and Designs
Fall 2023

This document is subject to minor changes regarding phrasing, grammar, spelling, and clarity until Tuesday, October 10th at 7:00 pm.

Due Dates:

- **Full Paper -** Sunday, October 22nd 2023, before 11:59 pm.

Overview: This homework consists of two parts. The first question will be writing about UML class diagrams and how they work (like a guide). For questions 2-4, you will be given some different scenarios and you will have to draw UML class diagrams in order to solve these designs. Make sure to start early, **you won't be able to complete this assignment if you start it the day that it's due**. This is an individual assignment. Looking at someone else's UML class diagram(s) is considered academic dishonesty.

Problem 1: (25 points)

UML (Unified Modeling Language) encompasses many aspects of documenting software system artifacts. In this course, we do not use all of these aspects, nor will we learn them. Here are all the UML diagrams: Class diagram, Object diagram, Use case diagram, Sequence diagram, Collaboration diagram, Activity diagram, Statechart diagram, Deployment diagram, and Component diagram. **We will only be doing Class Diagrams.**

UML's building blocks are broken down into "*things*". In this assignment, the "Things" that we will focus on are the following:

Structural things:

- Class, Interface, Collaboration, Use Case, Component, Node.

Annotational things:

- Annotation (also known as notes)

Relationships:

- Dependency
- Association
- Realization/Implementation
- Generalization/Inheritance
- Aggregation
- Composition

(These are also the arrows from workshop 2.)

We will only focus on Classes, Annotations, Relationships.
For part 1, write at least one page describing and explaining how to use each of the following aspects of UML: Classes, Annotations (notes), and all Relationships. Make sure to describe all aspects of each completely! Provide examples with diagrams/pictures as necessary (including an example of each relationship - ie. children Ducks inherited from a base Duck). **Cite your sources.** (You do not need to cite the lectures as a source.)

Problem 2: (25 points)
For the situation below, draw a UML Class Diagram using Lucid Chart to design the system.

Plants
In this problem, we will first consider Plants' relationships and properties. All **Plants** should have the following attributes:

- Height
- Shape
- Color
- Date

When a plant is planted (i.e. instantiated with a constructor), all 4 properties should be set. The **Date** planted **cannot** be modified afterward. However, **Height**, **Shape**, and **Color can** be changed. Ensure you have functions that can return the value of these properties and functions that only modify Height, Shape, and Color. Add in a *status* function as well: a pure virtual function.

The types of plants in this problem can be separated into two categories: **Apple Trees** and **House Plants**.
Apple Trees contain the following attributes upon creation:

- Weight (for apple)
- Sweetness
- Price

House Plants contain the following attributes upon creation:

- Lifespan
- Benefit
- Leaf shape

The following 6 **plants** are considered in this problem:
Honeycrisp, **Gala, Pink Lady, Orchid, Monstera, Snake Plant**
Remember to override the *status* function in each of these subclasses.

Next, let's create a **Garden** class with the attributes:

- Location
- Area

The types of gardens in this problem can be separated into two categories: **Apple Orchards** and **Greenhouses**.
Apple Orchards contain the following attribute upon creation:

- Apple Trees (vector of **Plants** [**Honeycrisp, Gala, and Pink Lady**] only)
- Apple Orchards **cannot** exist without apple trees

Greenhouses contain the following attributes upon creation:

- House Plants (vector of **Plants** [**Orchid, Monstera, Snake Plant**] only)
- Greenhouses **can** exist without house plants

And then, let's create a **Location** class with the attributes:

- Longitude
- Latitude

Finally, let's create a **Date** class with the attributes:

- Year
- Month
- Day

All **Gardens** classes must have functions to access each attribute, but not all attributes should be able to be modified. The **Date** class should have a constructor to set the initial attributes as well as a print function to get the date accordingly.

Create the UML class diagram that relates **Plants**, **Apple Trees**, **House Plants**, the **six plants**, **Gardens**, the **garden types**, **Location**, and **Date** considered in this problem. Make sure to encapsulate all of the properties and functions specified above.

Remember that you're just making a UML class diagram. **You are not coding this!** Make use of annotations in your UML class diagram to cover some of the non-obvious aspects of your system.

Problem 3: (25 points)
For the situation listed below, draw a UML Class Diagram using Lucid Chart to design the system.

Canvas
We want to represent a simplified version of Canvas, so to do that we'll need the following components.

Users - have to be a specific type and cannot be instantiated as just users. The available types are as follows.

- **Students** - users who take classes and have the following traits.
  - A current GPA which should not be accessible to anyone but the user
  - A list of **Course**s they're enrolled in which should not be accessible to anyone but the user
- **Instructors** - users who teach classes and have the following traits.
  - A list of **Course**s they're teaching which should not be accessible to anyone but the user
- **TAs** - users who can both take classes and teach classes.

All **User**s have an x500, a first name, a last name, and an email which are publicly available.

All **User**s will have two core functions.

- *UpdateProfile*, which updates the users parameters to match their myU profile and returns a bool on whether the operation was successful.
- *ViewProfile*, which prints out the users parameters.

The **Student** has three unique core functions in addition to the two general **User** functions.

- *GetGPA*, which returns a float that represents the GPA of a student.
- *SubmitAssignment*, which takes a reference to an **Assignment** and the content of the assignment (text) and returns a boolean on whether the assignment has been successfully submitted.
- *AddCourse,* which takes a reference to a **Course** and returns a boolean of whether the course was successfully added.

The **Instructor** has two unique core functions in addition to the two general **User** functions.

- *EnrollStudent* which takes a reference to a **Student** and a reference to a **Course** then returns a boolean on whether the student was successfully enrolled in the course.
- *GradeAssignment* which takes a reference to an **Assignment** and a float (which is their grade) then returns void.

The **TA** has no unique functions in addition to the two general User functions.

Course - has a unique course ID, a course title, a number of credits it is worth, and a list of **Assignment**s. The course ID should be stored as a string and it, along with the list of assignments, should not be accessed directly. The number of credits and the course title should be able to be accessed, but not changed after initialization.

Course has three core functions.

- *ViewCourseDetails* which prints out all the associated course details.
- *GetAllAssignments* which returns all the **Assignment**s currently contained in the course.
- *AddAssignment* which adds an **Assignment** to the course's list.

Assignment - This should include a title, a description, a due date, a point total and text submission. None of these parameters should be able to be accessed. The title, description, due date and submission should be stored as strings and point total should be stored as float.

Assignment has five functionalities.

- *GetGradePercentage* which returns a float value that represents the grade percentage obtained for that assignment.
- *AddSubmission* which takes in a text submission as input and returns a bool to signify that the functionality was a success.
- *PrintAssignmentDetails* prints out the details pertaining to the assignment without returning anything.
- *AddGrade* which takes in a float and returns void on execution.
- *GetSubmission* which returns the text submission as string.

Create the UML class diagram that relates **Users**, **Students**, **Instructors**, **TAs**, **Courses**, and **Assignments** considered in this problem. Make sure to encapsulate all of the properties and functions specified above.

Remember that you're just making a UML class diagram. **You are not coding this!** Make use of annotations in your UML class diagram to cover some of the non-obvious aspects of your system.

Problem 4: (25 points)
For the situation listed below, draw a UML Class Diagram using Lucid Chart to design the system.

Package Delivery Simulation System
In this problem, you will be drawing the UML diagram of the Package Delivery Simulation System project. We will not be giving the problem description but the actual code itself. From there, you will have to create the UML class diagram which incorporates the project system design itself.

Important: **Do not** create the class diagrams for the classes under

- `dependencies/`
- `libs/transit/include/util`
- `libs/transit/include/math`
- `libs/transit/src/math`
- `libs/routing`
- `apps/graph_viewer`

The project base code can be found in the lab04 repository:
https://github.umn.edu/umn-csci-3081-f23/public-lab04