# Software Engineering 2021/2022

## Assignment 2 – Class Design and Modelling

Teacher: Tom Pepels – tom.pepels@maastrichtuniversity.nl
Lab date: **11 April 2022** - Due date: **20 April 2022**

This assignment awards 30 points toward the total 100 assignment points.

***Important:*** *For this course, you work on all assignments in* ***teams*** *of* ***two****. Before submitting the assignment, you must join one of the* ***prepared groups*** *in Canvas (click on* ***People*** *in the course menu).*

*Do not make your own group in Canvas, nor submit the assignment without joining a group first!*

This assignment is the first stage of what leads to the gradual development of a small software project. In this first assignment, you will design the system in terms of classes and objects. Follow the following steps and hand in the deliverables before the deadline:

1. Read the assignment ***carefully***. Ask any questions you have before the start of the assignment.
2. Try to recognize ***classes*** that you need from the description of the software. Write down those classes and see if you can imagine implementing them such that they, combined together, form the software that implements all the features.
3. Now, draw a ***UML class diagram***. Make sure to include important constructors, public methods and properties. You do not need to include methods and properties that are not important to understanding how your software functions. However, make sure that you include them where they matter.
4. Draw ***separate*** sequence diagrams for ***all three*** ordering-entry requirement described below.
5. Write the **classes**, **interfaces**, **fields** and **important methods** that define your ***class model***. You do not have to implement the methods; just write empty methods without any implementation.
6. Place your classes and **interfaces** in appropriate **packages** (for Java, this means using an appropriate directory structure). Place classes that belong together in the same package (i.e. in the same directory). Make **dependencies** between these packages that are **loosely coupled**, i.e. you shouldn't create **circular dependencies** nor create **too many dependencies** between classes that should not have dependencies. Once you have done this, make sure that your class diagram matches your package structure.
7. For all relations (such as for instance the menu - dish / order - deliveries) create a few sample instances of whatever they aggregate. **For instance**, for the menu, create an *instance* of a starter, main dish, drink and desert in the constructor of the menu class. You may do this in a separate method in the case that it makes no sense to do so in the constructor. Create a few orders and deliveries. Every time you have some relation between two classes in your class diagram, you must show this relation using examples in your code.
8. **Deliverables:** Upload your diagrams and source code in separate files to canvas before the deadline to be graded. **Optional:** If you made any assumptions or you would like to give further explanation, you can write text in the UML diagram or upload a pdf with some text.

Here are a few free (to try) UML diagramming tools:

- https://www.visual-paradigm.com/solution/freeumltool/
- https://creately.com/
- https://umbrello.kde.org/ (open source)

**Grading:**

Separate parts of your assignment are graded separately. Your grade is composed of the sum of the three parts below. For each segment, the lowest score possible is 0.

- 15 points for the class diagram. It should be clear to an experienced developer how to start implementing the software.
    - Misuse of UML features: -1 point per case
    - The diagram is missing a feature -3 points per feature
    - Part of the diagram is not comprehensible by an experienced programmer or information is missing -3 points per case
    - Your code does not match the diagram -5 points
- 10 points for the sequence diagrams.
    - Missing lifelines or actors -1 point per case
    - Missing interactions -1 point per case
    - The diagram is missing information or is not comprehensible to an experienced programmer -2 points per case
- 5 points for the source code.
    - Missing instances -1 point per case
    - The code is not structured into packages -1 point per case
    - There are incorrect or too many dependencies between packages -1 point per case

**Submission Guidelines:**

Submit deliverables as a zip wile with filename: SE_ASS2_***GroupN_IDStudent1_IDStudent2***.zip substituting ***GroupN*** with your group number, and ***IDStudent1***, and ***IDStudent2*** with your student id's.

Make sure that you also write your:

- group name,
- student names,
- student Id's,

in **all submitted** files (write a comment at the top of all source code files)! Failure to do this may result in ***NG*** (no grade) for the assignment.

## Description

You work for a new start-up that wants to develop and sell software that allows restaurants to digitalize their menus and create an order process for their customers. The goal is to streamline the logistics for a food business. We need to track orders going through three stages: customer, kitchen, and waiter/delivery person.

The software should be designed such that it can be used for any type of restaurant, from pizzeria to bistro, from lunch café to snack bar. Both for **in-restaurant** dining as well as **take-away** and **delivery**. Your goal at the start of the development is to create a comprehensive design of all the classes involved.

Let us look at an overview of the requirements of the software.

**The software will consist of several modules:**

- The restaurant menu presenting all dishes with their ingredients, drinks and deserts.
  A restaurant has a menu with different menu items. These can consist of drinks, main dishes, side dishes, deserts etc. Menu items can, in some cases (such as main dishes) consist of multiple ingredients. In this case, the prices of these dishes should be calculated based on their ingredients.
- An order-entry module where, customers, servers or other systems can enter orders:
  - Through a web-interface for home delivery. The menu is presented to the user, where they can click through a web shop like interface to place and pay for their order to be delivered.
  - Through a web-interface for in-restaurant ordering by scanning a QR code, browsing through the menu and ordering. Customers pay the order later either through a server or using the QR code with a payment interface.
  - By a server/worker in the restaurant who takes and enters the customer's order
  - By offering their menu through an API to Uber-eats / Thuisbezorgd / Takeaway / Deliveroo etc.
- A delivery module that manages deliveries to customers
  - If customers order through the web-interface, a delivery must be made
  - A restaurant has several deliverers who deliver orders
  - When an order is placed, it takes some time to prepare it in the kitchen
  - After this, the order should be delivered as soon as possible by an available deliverer

**Pricing:**

Different restaurants use different ways to calculate the prices of their dishes. Therefore, you'll design the software such that it is possible to use on or more ***custom-made*** classes that calculate prices. For instance:

- Restaurant A wants their dishes to be 10% more expensive if eaten in.
- Restaurant B offers a discount of 5% for lunch orders placed between 10.00 and 11.00 a.m.
- Restaurant C has a happy hour between 16:00 and 18:00, two cocktails for the price of 1!
- Restaurant C also wants their dishes to be 5% more expensive if eaten in.

Our company will offer this customization as a bespoke service when the restaurant purchases the software. This means that, our company will implement the pricing strategy in Java for the customer. However, the design must allow for easy implementation, addition and modification of the pricing strategy. Ensuring that restaurants can easily change their pricing strategy.

**Some remarks:**

- In terms of class design. You do not have to **model** the **graphical user interface (GUI)**, only the back-end of the software. Meaning all classes that form the structure (the models, controllers and interfaces) of the software and how they relate.
- Make sure to include the **user** and **GUI** as actor and lifeline in the **sequence diagram**.
- You should however, include a class that acts as the **API** to the external ordering websites (Uber eats etc.). Think about how you could generalize this, internally you have one menu, but you must *“present”* this menu to different external services. Similarly, external services must be able to connect to your software to place orders. If you have no knowledge of how API’s work, you can either base the class on your own assumptions or first do some research online (search “REST API”) for examples and approaches.
- A good starting point may be to first define the classes needed to model the software. Next, see which classes aggregate others. Can you make the design as compact as possible?
- Once you think you have a correct diagram, write out the classes, make “sample” objects of your classes and see if your model still makes sense. Adjust if needed.
- Consider that the class diagram and your “empty” classes should be sufficient to start implementing the project without having to change the structure later on. Multiple teams will start working on implementing the software and they must all be able to trust the design.
- Check if your design can be structured into cohesive packages that have clear dependencies on other packages. At most **one or two classes** in a **package** should depend on another **package**.