F20-21SF Software Engineering Foundations

Lab 3 – OOP Design and Inheritance

Author: Pierre Le Bras

Introduction

In Lab 3 you will work through the material presented in Lectures 11 to 14. You will be presented with set of requirements and tasked with the design of an Object-Oriented system that would satisfy these requirements. You will then implement a basic version of this system.

Learning Outcomes

From this lab, you should know how to do the following:

- Abstracting requirements into a system design.
- Building Class diagram for communicating your system design.
- Implementing object relationships and inheritance in Java.

Part 1: Requirements

A local charity has contacted you with to develop an information that will help their day-to-day administration. You are given the following requirements:

- The charity wants to get a summary of its staff (worked time and wages due) and invoices.
- There are two types of staff working for the charity: employees, who receive a salary, and volunteers, who work on their spare time with no remuneration.
- There are full-time employees who work full-time (35 hours weekly) and receive a fixed monthly salary (e.g., £1800).
- There are part-time employees who work between 6 and 18 hours a week, and receive a hourly salary set to £10 per hour.
- Volunteers work a target numbers of hours per month.
- Every staff member has a Name and Email address, employees have a bank account number.
- Invoices have a date, a recipient and a value, the system should also record if they have been paid.
- The charity would like the system to report the number of hours worked by staff in total.
- The charity would like the system to report the total salary amount due to its employees.
- The charity would like the system to report the total invoice value due.
- The charity would like the system to report on the total amount of money to pay for the current month.
- Invoices and employees should be able to get paid.

For your first task, you should abstract these requirements into a set of classes and/or interfaces, listing their knowing and doing responsibilities. You should include these requirements in *requirements.md* on the GitLab repository.

You then need to write the relationships between the classes/interfaces of your design, using the following keywords: *inherits* (inheritance), *implements* (interface implementation), *knows* (association), *has* (aggregation), *owns* (composition).

Notes:

- For simplicity, you can assume that a month always has 4 weeks.
- You can store dates and names as String objects.
- There should be a CharitySystem class (which isn't the main class) that centralises list management operations.
- You don't need to write the requirements for the Main class.
- You should demonstrate at least one abstract class and one interface.

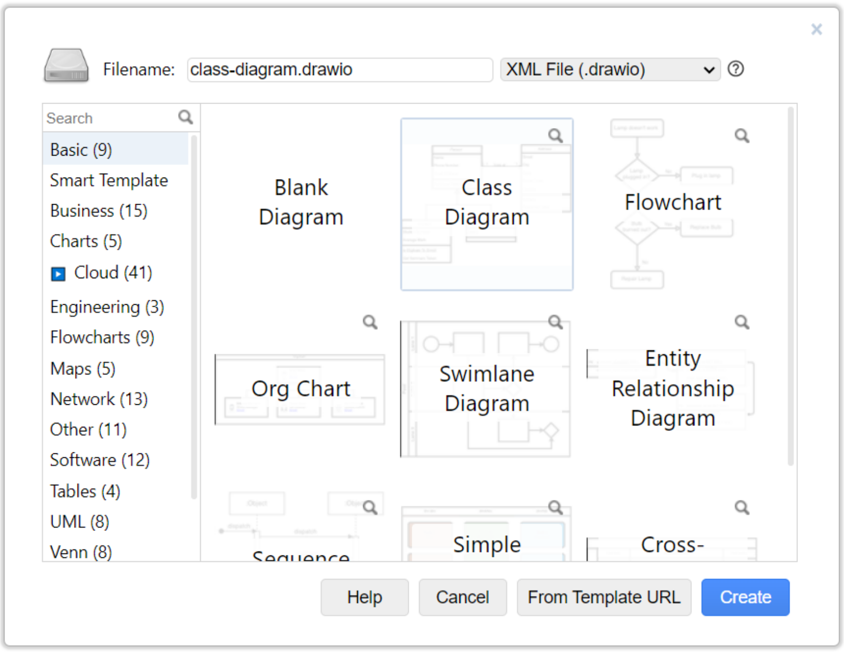
Part 2: Class Diagram

Your second task is to draw the class diagram associated with your system design from Part 1. You must use the UML notation introduced in the lectures.

You don't need to include the Main class in the diagram.

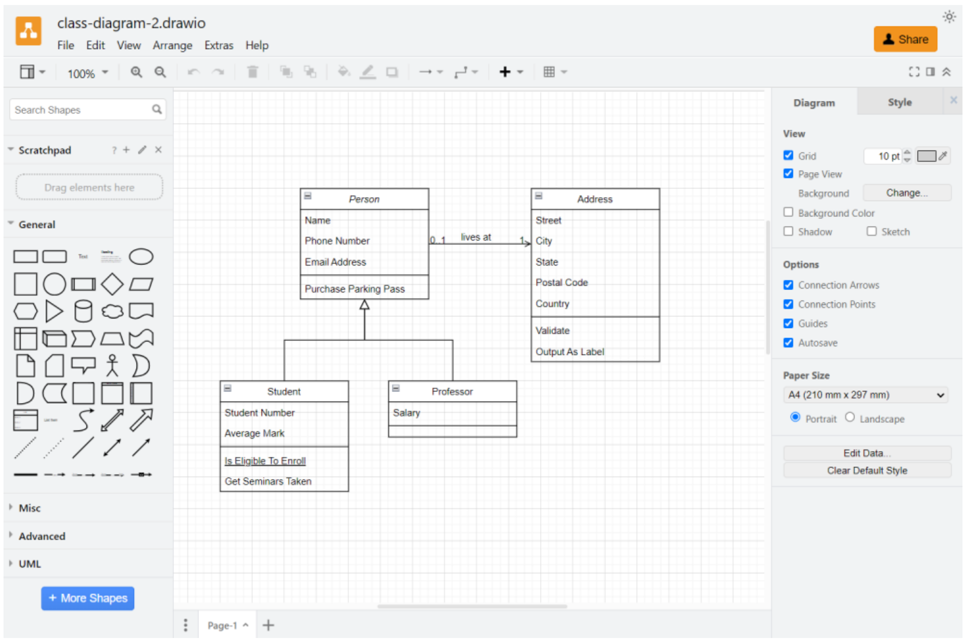
There are many free tools available for this. One recommended tool is drawio.com, it's free, runs in a browser and doesn't require installation. You can use other tools if you are more familiar with them.

Upon launching the app, you will be prompted to select a storage location (local or cloud based, e.g., your HWU account's OneDrive). You will then have to select a template to get started, you can choose the *Class Diagram* template.



You should then see a default class diagram that you can start editing.

Once you are finished, you can export the diagram as an image file, and upload it on your GitLab repository.



View
- [x] Grid 10 pt
- [x] Page View
- Background: Change...
- [ ] Background Color
- [ ] Shadow
- [ ] Sketch

Options
- [x] Connection Arrows
- [x] Connection Points
- [x] Guides
- [x] Autosave

Paper Size: A4 (210 mm x 297 mm)
[✓ Portrait] Landscape

Edit Data...
Clear Default Style

Notes:

- If you do not see the UML library of shapes, click on the “+ More Shapes” button to add it.
- Classes, attributes and methods can be edited by double clicking on them.
- Attributes and methods can be copy-pasted and dragged between classes.
- Connectors can be edited by selecting them and changing settings in the right-hand side panel, e.g., the line-start/line-end types.
- Double clicking on a connector will create a text box attached to that connector, you can drag that text in place later.
- You can drag either ends of a connector to attach it to a class/interface box.
- Your diagram can be exported to an image file by selecting File > Export.
- Remember to annotate methods and classes that are abstract, as well as interfaces.

Part 3: Implementation

Finally, on the GitLab repository, inside the *src* folder, you should implement a *basic* version of the system you have designed, and showcase key functionalities:

- List of staff details.
- List of invoices due.
- List of all payments due.
- Show the total number of hours worked by staff members.
- Show the total salary amount due to employees.

Your main method should call methods from the *CharitySystem* class to retrieve information. You should also have a method for generating dummy data (i.e., a few staff members and invoices).

At this point, you might notice that your original design needs to be updated: reflects those changes in your *requirements.md* file and the class diagram.

There is no code sample provided, instead you should create a new project in the IDE of your choice.

Notes:

- This Lab is focused on the ***design*** of an Object-Oriented system, e.g., the inheritance between classes and relationships between objects.
- Beyond getters, simple calculations and methods that print lists, you shouldn't spend too much time on writing complicated code.
- For some methods, you are allowed to simply have a stubbed implementation, e.g., printing *"This action has been performed"* in the console.