MAS Notes

(Allegedly, how the subject works and how to pass it)

Good luck

Quick note

Sigh... Use ChatGPT to come up with ideas if you have to. However, do not use any generated code in the projects that you submit. A zero from even one project means you fail the subject. The teachers will nitpick, judge, and suspect you. If you copy code from somewhere, be honest about it and make sure that you understand what you copy.

Basic background

This subject is considered one of the hardest you will ever take. It can be made somewhat approachable with some structure, preparation, and time management. Getting a 5 is hard but in no way impossible. In either case, read this and start preparing **before** the semester starts.

The subject consists of 3 parts:

1. Mini projects
2. Final project documentation
3. Final project implementation

To proceed to the next part, one needs to pass (obtain at least 50% of the maximum possible grade) the previous part. To pass the subject one needs to pass all individual parts.
There are **no exemptions** from the exam.

Mini projects

You only have a few weeks to create and present a mini project. As soon as you finish one, start working on the next. Otherwise, you will not have time to finish and present all of them.

Use this section as a guide to prepare the skeletons of the projects before the semester starts. Once you get your actual tasks, you will save time by modifying existing skeletons to suit the task. What the teacher specifically asks for heavily depends on the teacher. Try not to spend too much time preparing the skeletons, as you may have to scrap one or two during the semester.

There are 5 mini projects (for a total of **100** points):

Classes, Attributes - 20 points

Design a system (UML + implementation) that includes all of the following:

- **Attribute types:** `basic`, `complex`, `multivalue`, `optional`, `class`, `derived`.
- **Methods types:** `base`, `static`, `overload`, `override`.

For easy points, a custom `toString()` usually counts as `override` (ask you teacher first).

This task is similar to one in BYT, with some extra requirements. Use your past notes as much as possible. If you have access to your BYT notes right now, download them in case the group will be disabled.

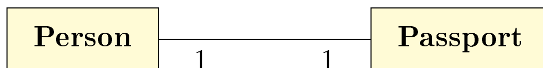
Pay attention to whether your teacher wants every class to have a class extent. If they don't mention it, ask directly after class.

Associations - 20 points

Design a system (UML + implementation) that includes all of the following:

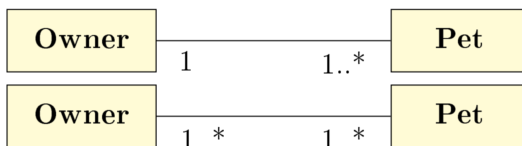
- **Association types:** `basic`, `qualified`, `composition`, `with attribute`. (`aggregation` and `reflex` might not be required, check in with your teacher.)
- **Reverse connections** are mandatory for every association. Make sure you have at least some tests for this, infinite loops are the most common issue.

Do not use one-to-one associations like these:



Even though these associations are perfectly correct, the teacher will not be happy to see them. They are considered basic and uninteresting.

Also, unless specifically asked to, avoid using either one of these:



They are hard to implement and would require additional constructors / factory methods for creating pairs of objects.

`qualified` associations should not be implemented using IDs (in general, avoid IDs where possible, with the exception for database PKs). Here are some good options to use instead:

- company names,
- phone numbers,
- role name,

- location names,
- addresses,
- product codes,
- `datetime(now)`,
- any other unique string attribute, or
- a unique combination of two or more attributes.

Inheritance - 20 points

Design a system (UML + implementation) that includes all of the following:

- **Inheritance types:** `abstract`, `disjoint`, `overlapping`, `dynamic`, `multi-inheritance`, `multi-aspect`.
  Do **not** combine inheritance types for this project (e.g. `{overlapping, dynamic}`). It will complicate the implementation and the defense. It's easier to implement an extra class than a combined inheritance type. If needed, design several smaller systems (ask your teacher if they think it would be better).
- **Analytical diagram:** Showcasing the **initial design** using UML.
- **Design diagram:** Showcasing the **actual implementation** as it is in code.

BYT covers everything related to this mini project, pull up your old repos and copy from there.

Constraints - 16 points

Design a system (UML + implementation) that includes all of the following:

- **Attribute Constraints:**
  - `static` - any type of constraint that is applied regardless if the attribute is being assigned for the first time or changed.
  - `dynamic` - any type of constraint that change depending on the value of the attribute. They only work on the update. For example:
    - Cannot increase / decrease value
    - Cannot change value more than X times per Y time
    - Value must change at least X times per Ys time
  - `unique` - self-explanatory, technically a `static` constrain. Use this constraint on something other than the ID. This will get you extra points (or prevent you from losing some).
- **Universal Constraints:**
  - `subset` - only applies if there are 2 or more associations between 2 classes. One association is the "main" association (`superset`), the other one(s) are the "dependent" associations (*subset*). The `subset` association can only exist if the `superset` association that links the same objects exists.
    Here is a StackOverflow question describing the `subset` association.

`ordered` - basic constraint that is applied to a class / association / list attribute. States that the contents must be always sorted.

`bag` / `history` - allows duplicate associations between the two classes using an extra association table (association class / association by attribute). I.e. stores a list of associations based on some identifier (usually date when the association was created / expired). If the association is not a `bag` / `history`, the previous association needs to be deleted if the new one gets added.

`XOR` - states that for the given 2 associations `A1` and `A2`, either `A1` **OR** `A2` must exist, but **not both** and **not neither**. Most common with `{Dynamic}` associations.

`business logic` - not a real thing but some teachers ask to implement it. Business logic is whatever makes sense as constraint in your system. Write some custom validator and package it in a constraint.
For example: "This string can be `null`, but if it is not `null`, it must be strictly longer than 9 characters" (like an optional phone number).

- **Validators:**
  All constrains must be enforced with validators. If your teacher does not specify how to implement them, the choice is up to you. Common validators are annotations and if statements, do not overcomplicate the system if you don't have to.

Don't forget to put the constraints both in the UML diagram and the implementation.
This task is the "easy points" that you can use to catch up to the needed 50% if you did not get enough from the first 3. Note that it is worth a little less than the previous ones.

Relational Model - 24 points

Connect the system to a database, in whatever method you see fit. The details are defined by the teacher.
For a skeleton project - make a simple ORM example with a database. This will save you time once you need to implement it for a proper system.

Notes

Use the first 3 projects to get over the 50% threshold for passing this part. They are covered in the PRI and BYT classes, which you have hopefully taken (God help you if not). The other 2 are meant to catch up to a passing grade if you couldn't do it with the first 3, or for people that are aiming to get a 5 from the subject.
The best way to pass mini projects is to take inspiration (and maybe code) from your BYT project (talk with your teammates first). Try to design the simplest possible system that fits the criteria. The teacher will thank you for a quick defense (they have ~20 defenses in 3 hours for your group), and you will avoid doing extra work.

Final project documentation

This and the next sections comprise your final project for this class. A good rule of thumb for the size of the project is **at least 80%** of the combined features of all mini-projects, or about as big as your final BYT project. Some people note that this has now changed to somewhere around 50% of the combined mini-projects, ask your teacher if you are not sure. Grades are given based on the complexity of the system (not the real world complexity, but rather the amount of stuff from mini projects that you can cram into it). Try to implement as many different features without making the system complex to understand.
Approximate grading system for this part (for a total of **100** points):

- Complexity of the business domain - **10** points
- Documenting the use cases (use case scenario, use case diagram) - **10** points
- Correctness and complexity of the design class diagram - **35** points
- Correctness and complexity of the activity diagram - **10** points
- Correctness and complexity of the state diagram - **10** points
- GUI design - **10** points
- Discussion of the design decisions (dynamic analysis) - **10** points
- Readability and the organization of the document - **5** points

I cannot emphasize enough how important it is to have most of the stuff correct on the first defense. If the teacher notices a lot of errors they might (allegedly) skip checking the rest, give you feedback on the stuff that they checked, and send you for the next week's defense. The problem arises when you realize that there are more errors that the teacher hasn't checked and hasn't given feedback on. This means that you might be presenting a faulty system for your last chance at a defense. Make sure that your system does not have enough missing part to make the teacher "give up" on you during the first defense.

For the UI design, choose use cases that either:

1. creates an object based on already existing objects, e.g.:
   add a new appointment for the patient and the doctor.
2. uses object referenced between already existing objects, e.g.:
   assign a product to a cart.

Avoid use cases that just retrieve information, they are considered too primitive. Use cases should also demonstrate reverse connections as much as possible.

Final project implementation

This part is the biggest time consumer. The trick is to make this part **first**, and then use parts of the complete systems as mini-projects.
In this part, using a database is the best decision you can make. The subject is not language-specific, you are free to choose any combination.
Remember to have some kind of a main program flow that will showcase the functioning of the system. This is what you will be presenting at your last defense.

The system you are creating will not be too different from what you have made on BYT, but this time you are on your own.

Other things to keep in mind

No More Unit Tests - Unlike other subjects, this one does not require unit tests in your final project. **This does not mean you shouldn't have them**. You project is evaluated based on correct functioning, you knowledge of the system, the code and how it runs under the hood. Writing unit tests will help you a lot during the defense of the final project.
As for the mini projects - don't bother. Write them as quickly as you can, you can ensure their correctness by some manual testing.

Get to know the teachers - Your grade depends quite a lot on how the teacher views you. Some are stuck in their ways and would like to see your system developed how **they** want. Contact the teacher **before** the first defense and ask them if your code is up to their standard.
Make sure that you project confidence and professionalism during your first defense. Try to go first if you can. If you lose the teacher's respect, you will fail the subject. Unfair? Maybe. Worth fighting? Unfortunately, no.

Know what you wrote - The teacher might delete a part of your code as a part of the defense and ask if and/or how the system will break. You might also have to rewrite the deleted part during the defense to prove that you know your system. If the teacher suspects that you did not write the code yourself (regardless if you actually did), you may get a 0 from the project and fail the whole course in the process. Revise before class and make sure that you know what you are presenting. Inline docs are always a plus. Making full blown documentation pages is a way to gain some "wow points" during the defense.

Good luck!