SWEN30006 Software Modelling and Design
Workshop 3: Design Models and Coding

School of Computing and Information Systems
University of Melbourne
Semester 2, 2020

Assessment: You *should* complete the exercises ***as a team***. To receive a workshop mark, you **need** to demonstrate your active participation during the workshop. Your tutor will observe your participation for you to be eligible to receive a mark for this workshop. The active participation include (but not limit to) present your solutions, help your teammates, and actively engage with the team. Attending the workshop with only passive participation will *not* be given a mark. See the Workshop Overview under Subject Information on the LMS for more details.

Requisite Knowledge and Tools

To complete this workshop you will need to be familiar with the following terms and concepts:

- Use cases and use case diagrams
- Domain models and system sequence diagrams
- The representational gap
- Object Oriented Programming in Java

Introduction

This week workshop will move focus to the solution space and static and dynamic design modelling. We will be considering the relationship of the design models to the domain model and system sequence diagrams and the representational gap between the static models. We will also look at the transformation of the design models into Java code. Once again we will be using Draw.IO as our supported diagramming tool.

We suggest you use the textbook and resources provided in the "Pre-Workshop Learning" sub-module which include UML notation references for Design Class Diagram and Design Sequence Diagram.

Part 1 From Domain to Design Class Diagrams

Note: All tasks should be done as **a team** and demonstrate **active participation**.

Having now generated a Domain Model for a subset of our new student learning management system, lets look at the work involved in deriving a Design Model from this Domain Model. Focusing on the `Upload an Assignment Submission` use case from the previous workshop, create a Design Model that specifies the software classes required for this functionality. You may abstract concepts such as Databases to a higher level. Assume the implementation is in Java.

After completing version 1 of your Static Design Model, answer the following questions:

1. When deriving our Design Model from the Domain Model (and other constraints) it is common that we try to minimise the *Representational Gap*. That is, we translate a domain class `Exam` to a design class `Exam`. We could have easily called the design class `BaseBallGlove` or `AhKisuLwjMMn` but we don't. What advantages does this give?
2. What kind of assumptions did you need to make about the design to construct the Design Model from the Domain Model? Where would you normally get this kind of data?

Part 2 Creating a Dynamic Design Model

Starting from the `Submission` System Sequence Diagram, draw a `Submission` design sequence diagram. Ensure that your system sequence diagram, your design class diagram, and your design sequence diagram are consistent; you may need to iterate a few times making changes to achieve this.

After completing version 1 of your Dynamic Design Model, answer the following questions:

1. What does the dynamic model tell us about the static model?
2. Would it have been better to start with the dynamic model and use it to inform construction of the static model?

Part 3 From Design to Implementation

Coding collaboration: To demonstrate active participation for coding exercises, you can work individually or perform *pair programming*, where one of the members writes the code while others can observe via live screen share.

Suggestion: If you opt to perform *pair programming*, it is highly recommended that the one with low programming skills should write the code with a help of other members. You are also encouraged to take turns for the coding exercises throughout the semester.

The final step after you have finished your static modelling is to move to actual implementation. Often it is the case that developers will focus on Static models (including domain and design class diagrams), along with use cases, to specify how software is to be written. Figures 1 and 2 specify the domain and class diagrams for a basic program that validates student submissions to our new LMS system. The interface required have been provided in the workshop package for you, along with a sample mock implementation.

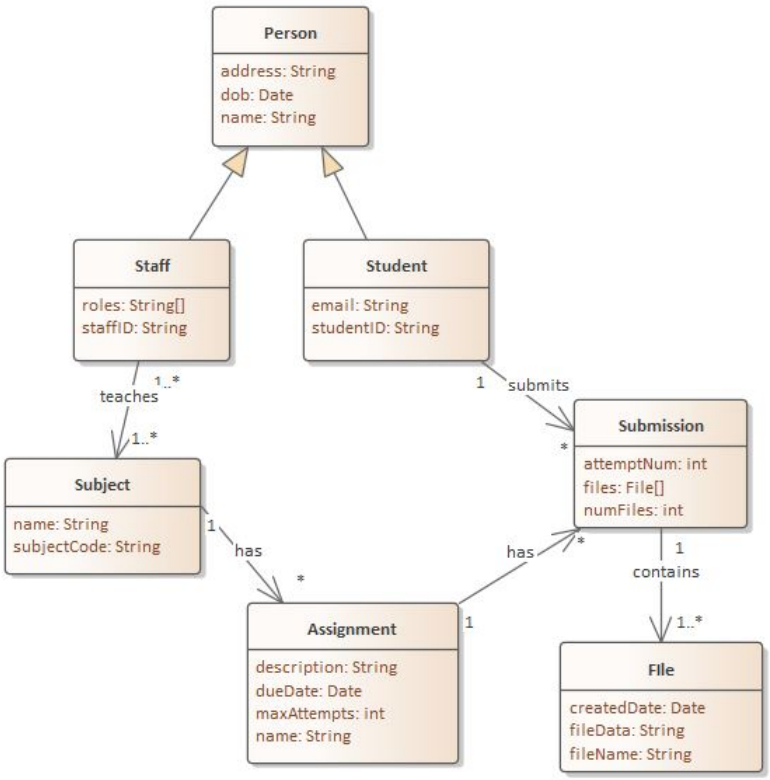

Figure 1: Incomplete Domain Model for Submission Validation

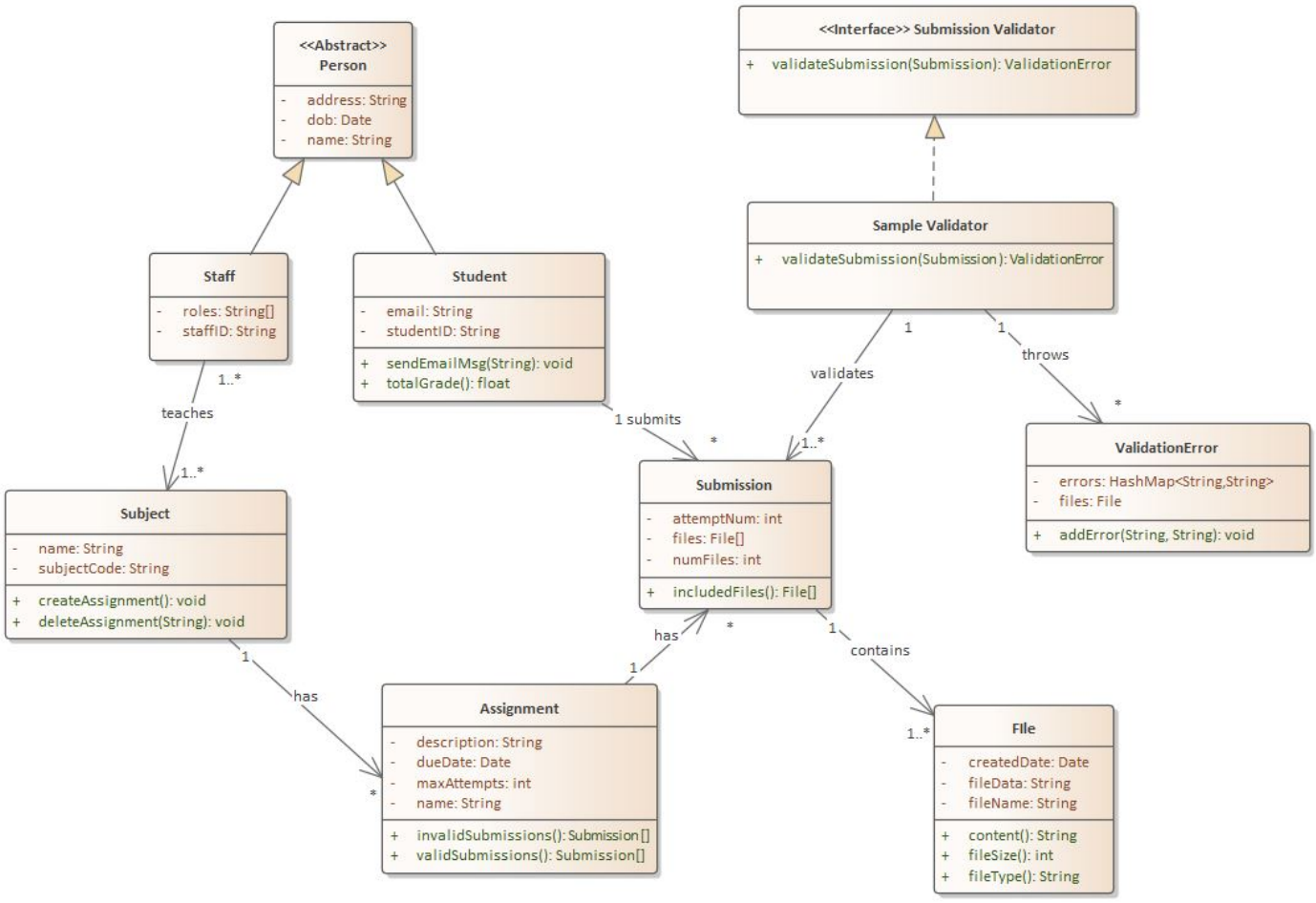

Figure 2: Incomplete Design Model for Submission Validation

Based on this information alone, create a Java implementation of the Design Model in Eclipse, along with a driver program to test your program. Once you have completed this, answer the following questions.

1. Did you have sufficient information to complete the implementation?
2. What assumptions, if any, did you have to make about certain functions or relationships within the program?
3. Does your program work the same as people's sitting near to you? If not, how does it differ?