COMP2511
Domain Modelling using UML

Prepared by Dr. Robert Clifton-Everest

Updated by Dr. Ashesh Mahidadia

Domain Models

- Domain Models are used to visually represent important domain concepts and relationships between them.
- Domain Models help clarify and communicate important domain specific concepts and are used during the requirements gathering and *designing phase*.
- Domain modeling is the activity of expressing related domain concepts into a domain model.
- Domain models are also often referred to as *conceptual models* or *domain object models*.
- We will be use Unified Modeling Language (UML) class diagrams to represent domain models.
- There are many different modelling frameworks, like: UML, Entity-Relationship, Mind maps, Context maps, Concept diagrams. etc.

Requirements Analysis vs Domain modelling

- Requirements analysis determines *external behaviour*
  "*What are the features of the system-to-be and who requires these features (actors)*"
- Domain modelling determines (internal behavior) –
  "*how elements of system-to-be interact to produce the external behaviour*"
- Requirements analysis and domain modelling are mutually dependent - domain modelling supports clarification of requirements, whereas requirements help building up the model.

What is a domain?

- *Domain* – A sphere of knowledge particular to the problem being solved
- *Domain expert* – A person expert in the domain
- For example, in the domain of cake decorating, cake decorators are the domain experts

Problem

A motivating example:

- Tourists have schedules that involve at least one and possibly several cities
- Hotels have a variety of rooms of different grades: standard and premium
- Tours are booked at either a standard or premium rate, indicating the grade of hotel room
- In each city of their tour, a tourist is booked into a hotel room of the chosen grade
- Each room booking made by a tourist has an arrival date and a departure date
- Hotels are identified by a name (e.g. Melbourne Hyatt) and rooms by a number
- Tourists may book, cancel or update schedules in their tour

Ubiquitous language

- **Things in our design must represent real things in the domain expert's mental model.**
- For example, if the domain expert calls something an "order" then in our domain model (and ultimately our implementation) we should have something called an Order.
- Similarly, our domain model should not contain an OrderHelper, OrderManager, etc.
- Technical details do not form part of the domain model as they are not part of the design.

Noun/verb analysis

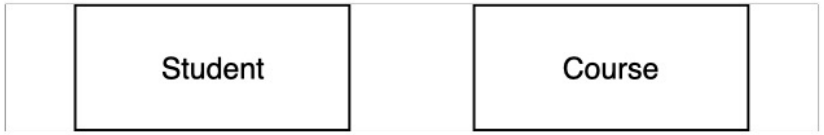
- Finding the ubiquitous language of the domain by finding the nouns and verbs in the requirements
- The nouns are possible entities in the domain model and the verbs possible behaviours

Problem

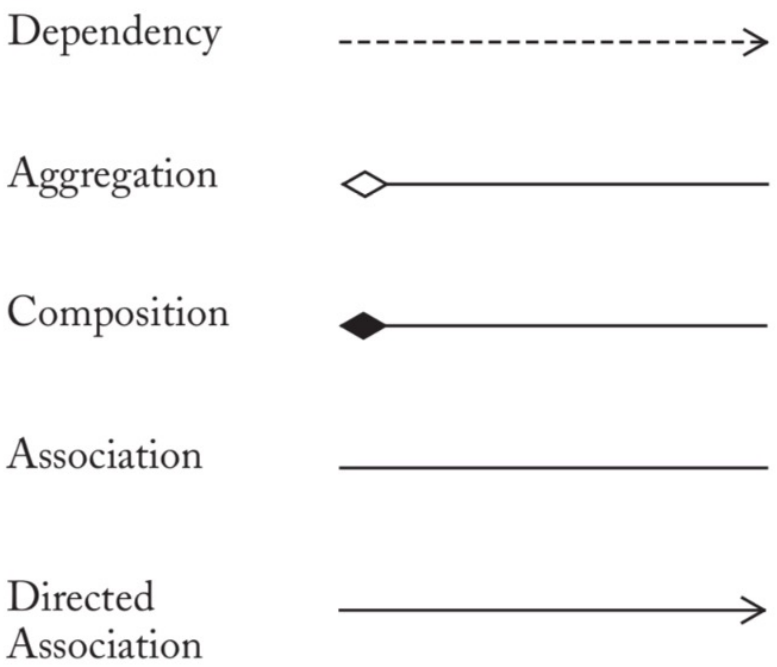
- The **nouns** and verbs:
  - **Tourists** have **schedules** that involve at least one and possibly several **cities**
  - **Hotels** have a variety of **rooms** of different **grades**: standard and premium
  - **Tours** are booked at either a standard or premium rate, indicating the **grade** of hotel **room**
  - In each **city** of their **tour**, a **tourist** is booked into a hotel **room** of the chosen **grade**
  - Each room **booking** made by a tourist has an arrival **date** and a departure **date**
  - **Hotels** are identified by a **name** (e.g. Melbourne Hyatt) and **rooms** by a **number**
  - **Tourists** may book, cancel or update **schedules** in their **tour**

UML Class diagrams

- Classes



- Relationships



UML Class diagrams

Dependency 

- The loosest form of relationship. A class in some way *depends* on another.

Association

- A class "uses" another class in some way. When undirected, it is not yet clear in what direction dependency occurs.

Directed Association 

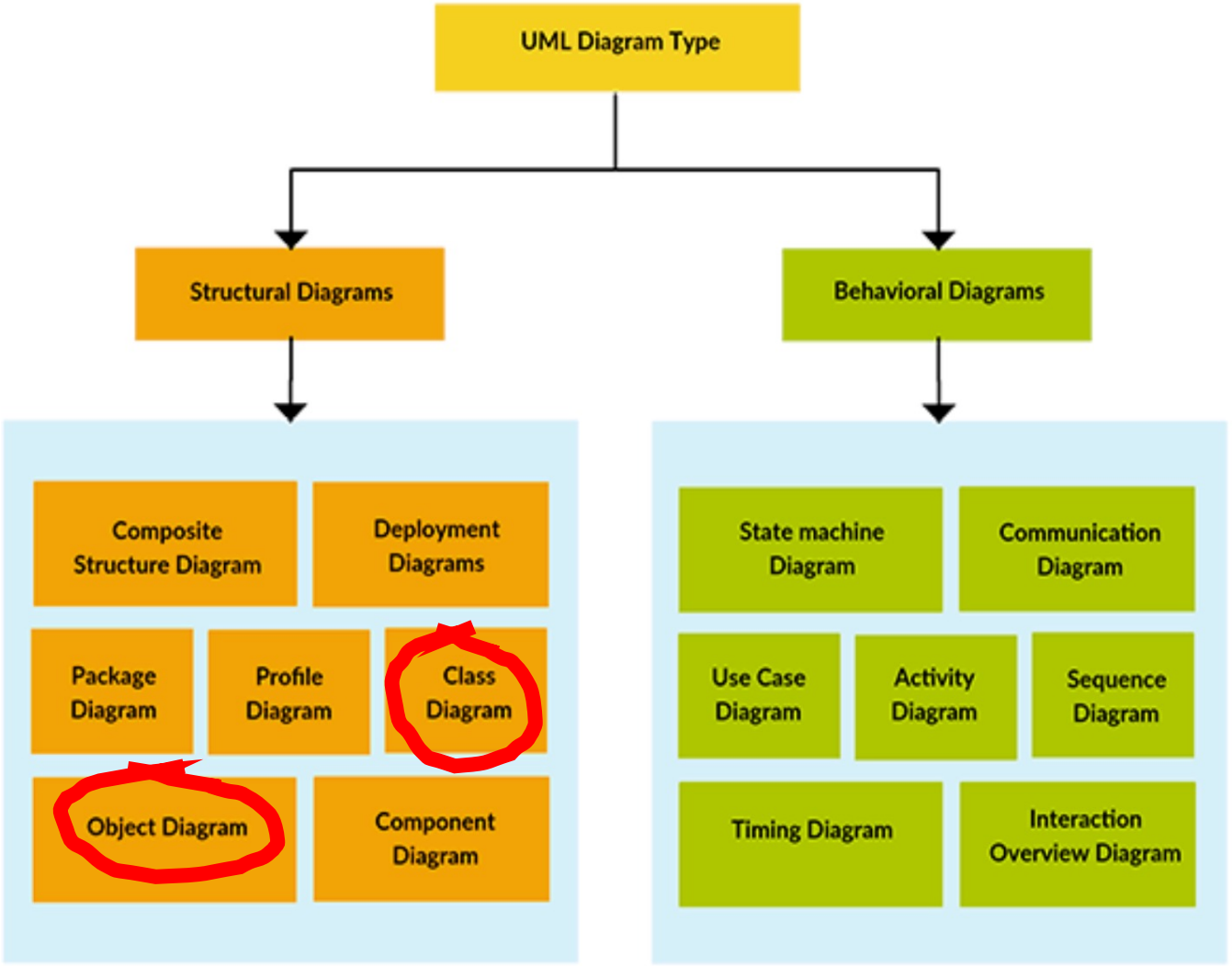
- Refines association by indicating which class has knowledge of the other

UML Class diagrams

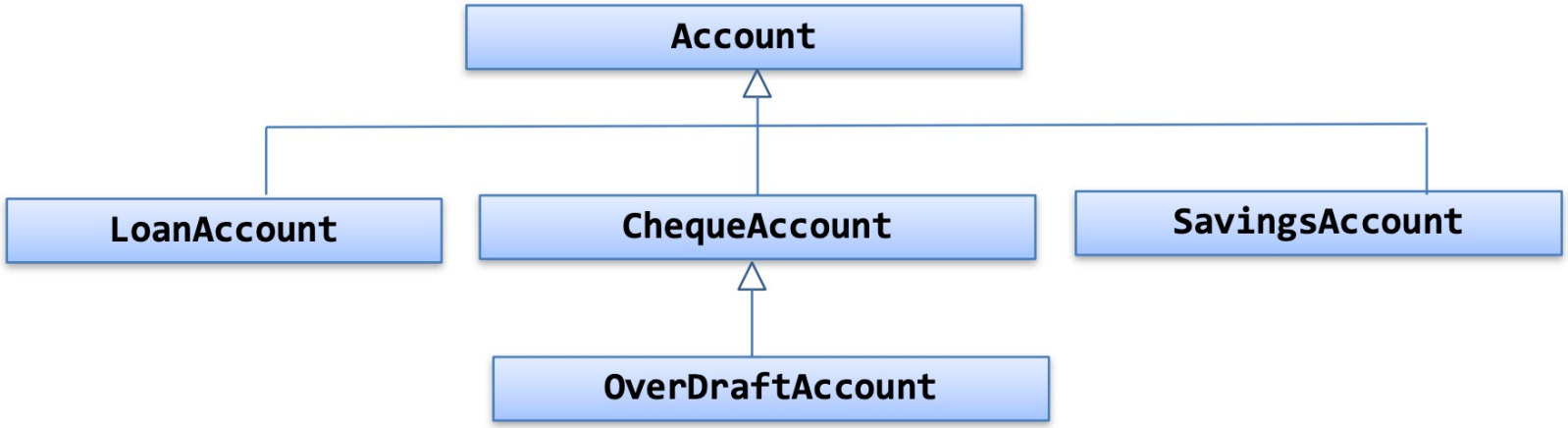
- A class contains another class (e.g. a course contains students). Note that the diamond it at the end with the containing class.

- Like aggregation, but the contained class is integral to the containing class. The contained class cannot exist outside of the container (e.g. the leg of a chair)

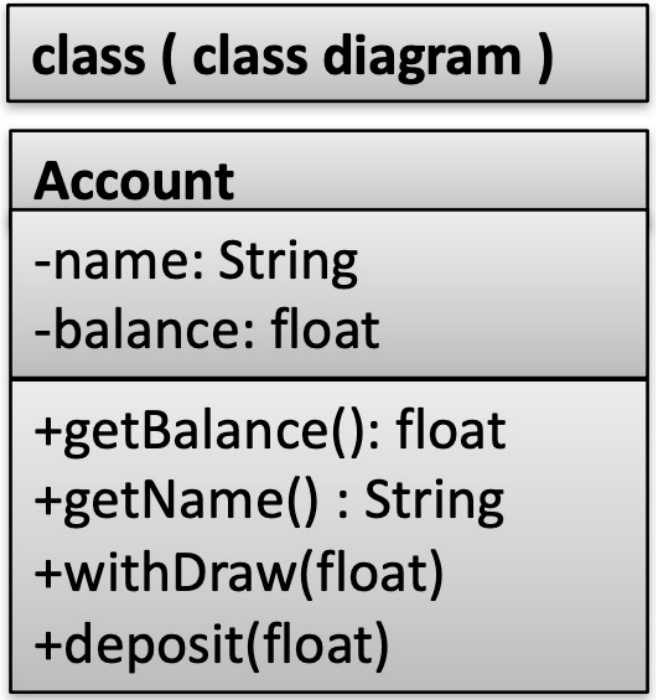
UML Diagram Types

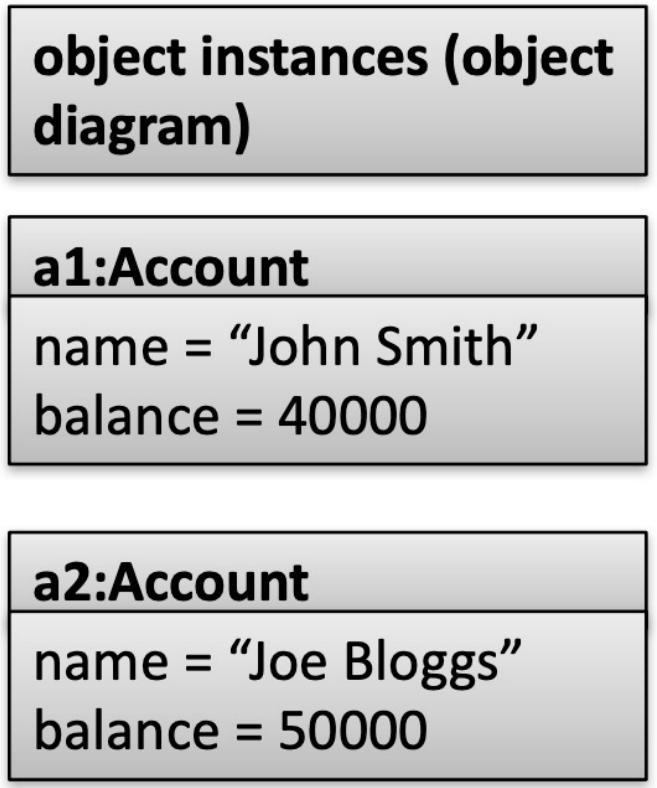


Examples

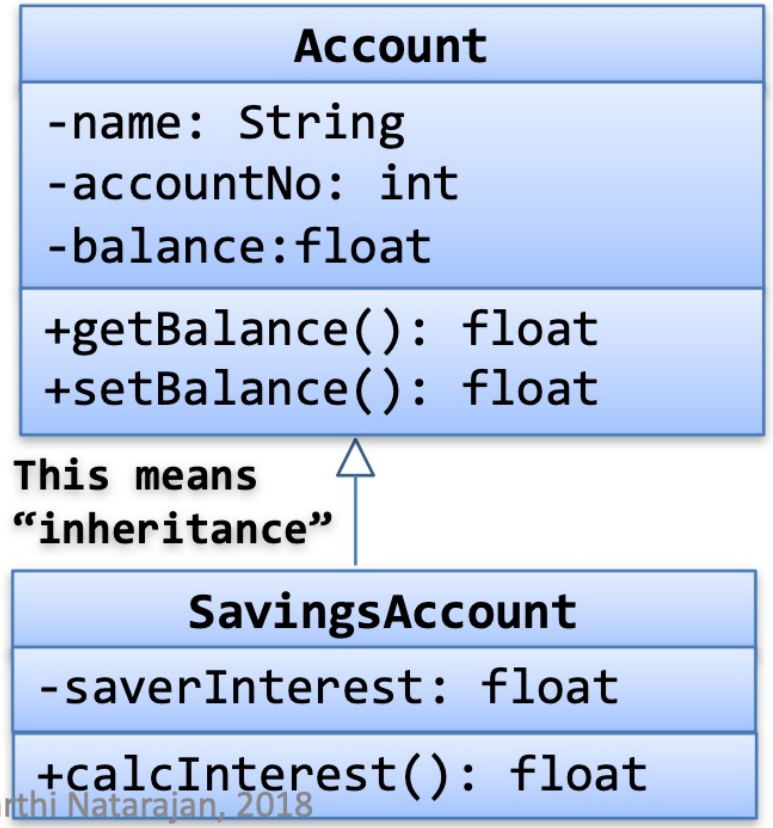


Representing classes in UML

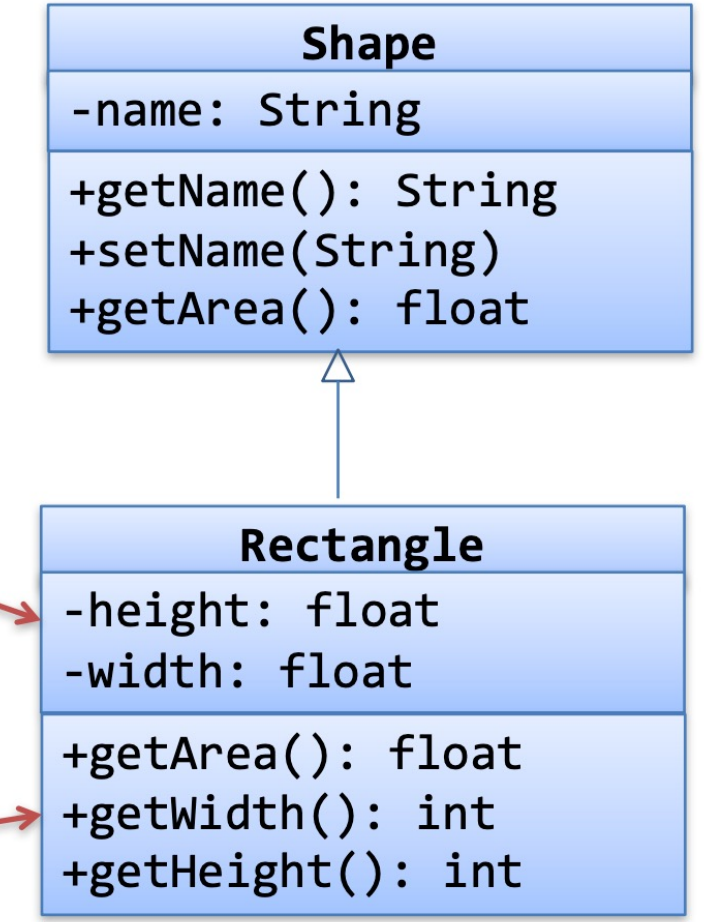




Representing classes in UML



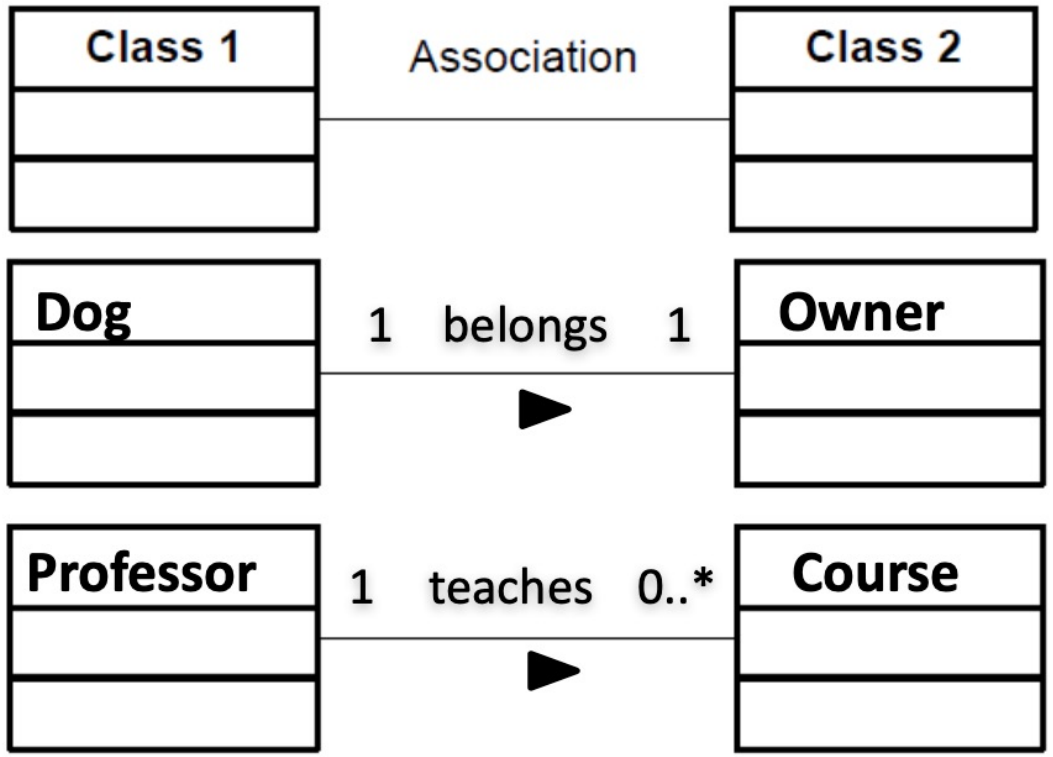
Representing classes in UML



class Rectangle *extends* Shape adding attributes *height* , *width*

class Rectangle *overrides* method *getArea()* to provide its own implementation

Representing Association in UML



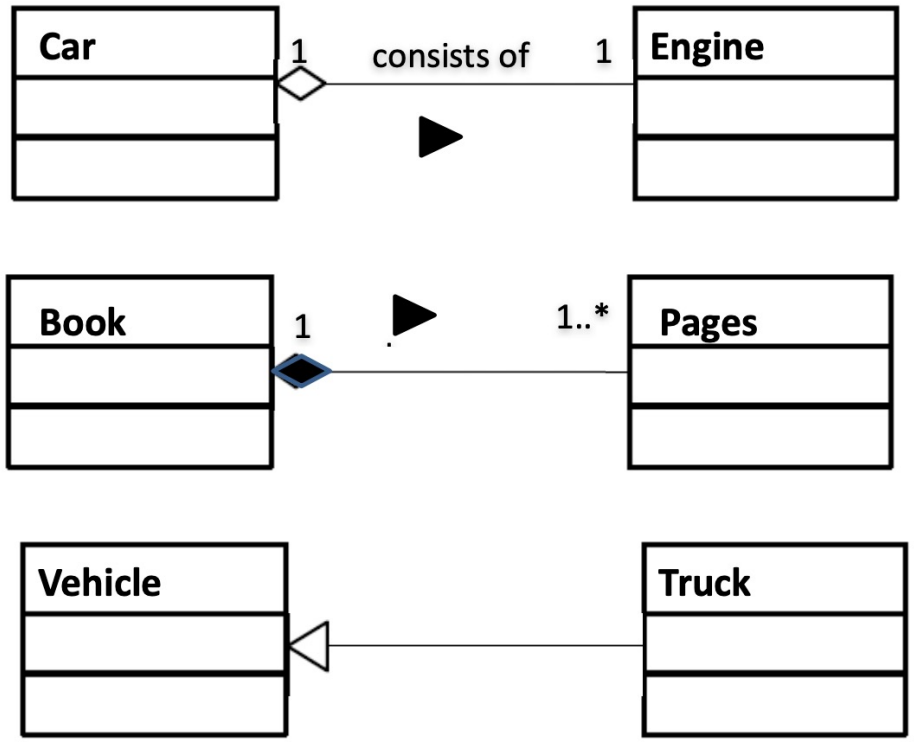
Representing Association in UML

- Associations can model a ***"has-a"*** relationship where one class "contains" another class
- Associations can further be refined as:

Aggregation relationship (hollow diamond symbol ◊): The contained item is an element of a collection but it can also exist on its own, e.g., a lecturer in a university or a student at a university

Composition relationship (filled diamond symbol ♦ in UML diagrams): The contained item is an integral part of the containing item, such as a leg in a desk, or engine in a car

Representing Association in UML



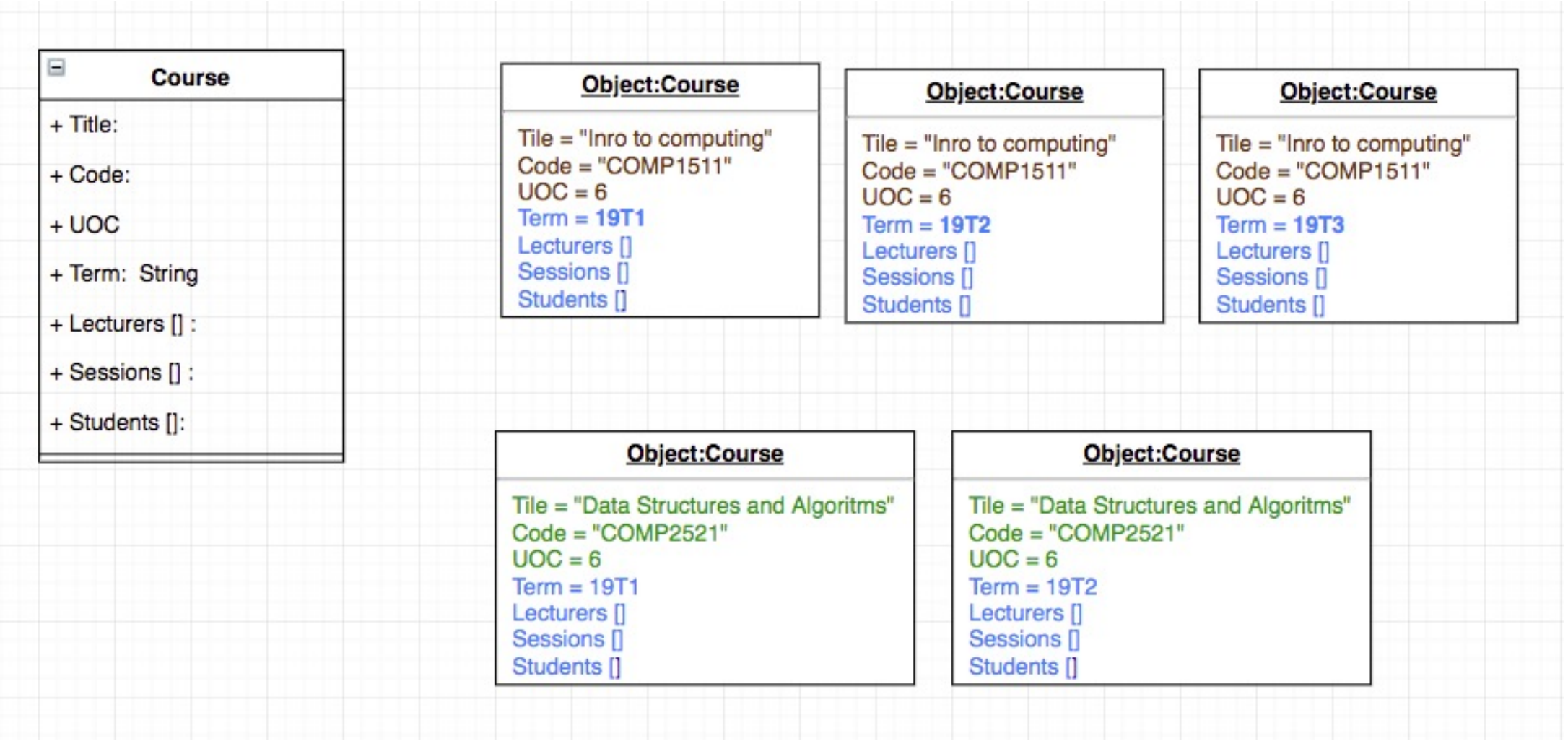
- Aggregation - “has-a” relationship where the part can exist without container
- Composition – “is-composed-of” relationship where part cannot live without container
- Inheritance – “is-a-kind-of” relationship

Attributes vs. Classes

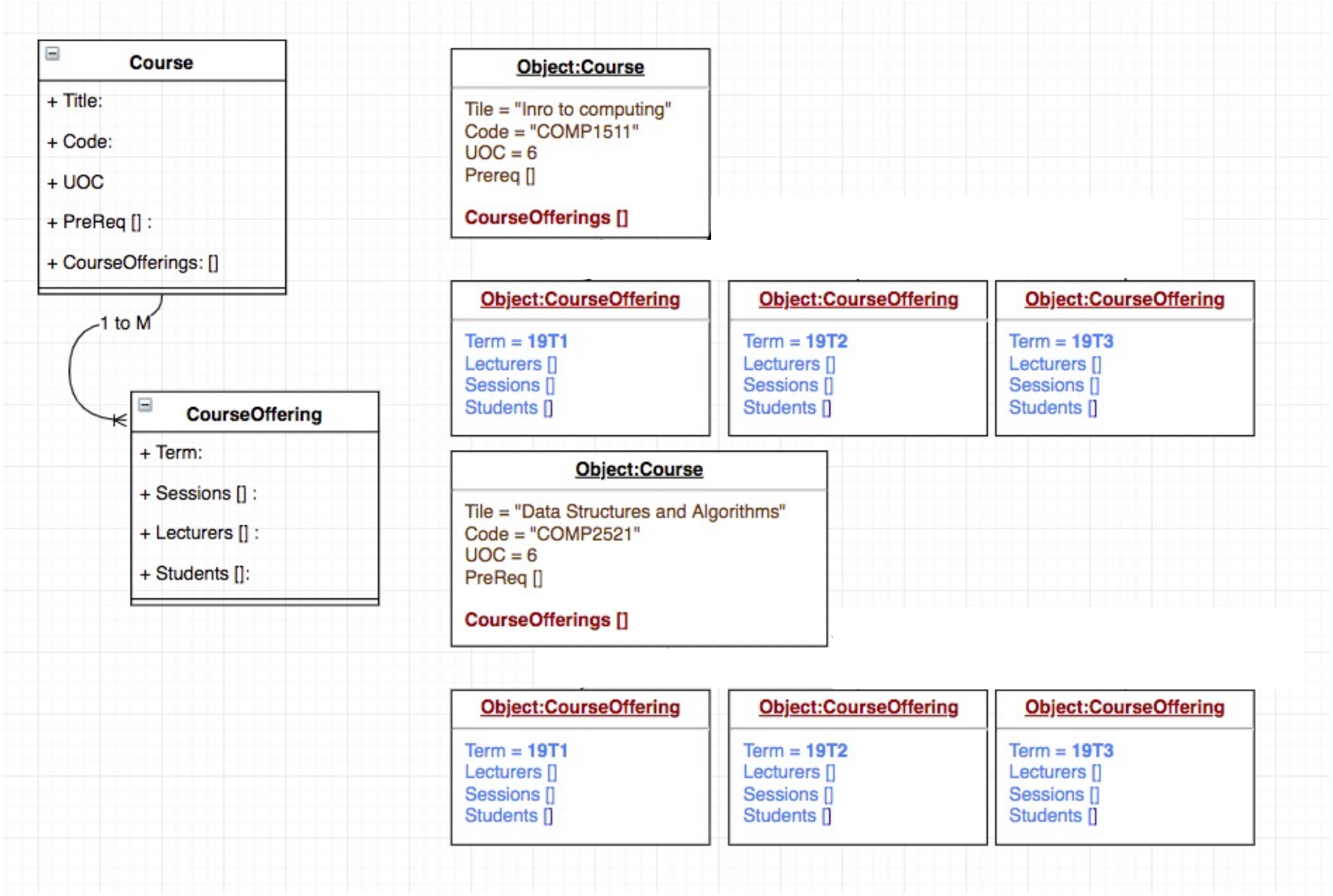
- The most common confusion – *should it be an attribute or a class?*
  - when creating a domain model, often we need to decide whether to represent *something* as an *attribute* or a conceptual *class*.
- If a concept is **not** representable by a *number* or a *string*, most likely it is a *class*.
- For example:
  - a *lab mark* **can** be represented by a *number*, so we should represent it as an *attribute*
  - a *student* **cannot** be represented by a *number or a string, so we should represent it as a class*

Attributes vs. Classes

What wrong with the following?



A Possible solution



References

- A very detailed description of UML
  - https://www.uml-diagrams.org/
- Books that go into detail on Domain Driven Design
  - *Domain-Driven Design: Tackling Complexity in the Heart of Software* by Eric Evans.
  - *Domain Modeling Made Functional: Tackle Software Complexity with Domain-Driven Design and F#* by Scott Wlaschin.