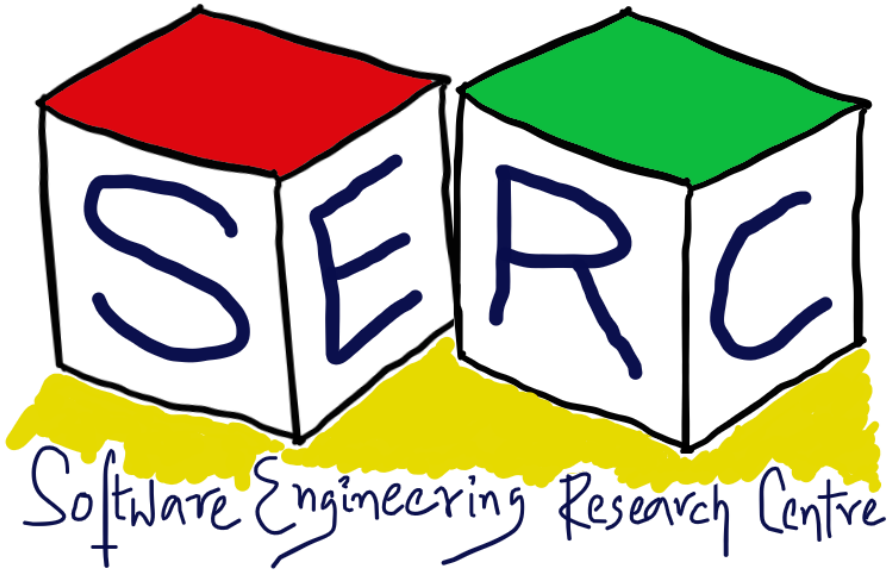
Software Modeling: An Overview

CS6.401 Software Engineering



Rudra Dhar

PhD Student

SERC, IIIT-H

Acknowledgements

Sources:

1. Introduction to MDE, Ludovico Iovino, GSSI, Italy
2. UML@Classroom, An Introduction to Object-Oriented Modeling by Martina Seidl, Marion Scholz, Christian Huemer and Gerti Kappel
3. UML Modelling lecture, Dr. Raghu, IIIT Hyderabad

Source: Googe images

UML

Unified Modeling Language (UML): Brief History

- No common language to model until 1996
- GPL developed by industry consortium in 1997
  - Introduction of OOP in IT dates back to 1960's
  - Required a standard representation: **OMG**
  - Three Amigos: Grady Booch, Ivar Jacobson and James Rumbaugh
- Based on multiple prior visual modeling languages
- Goal was to have a single language that could cover large number of SE tasks
- Current version of UML: 2.5.1 (as of Dec 2017)

https://www.omg.org/spec/UML/

Image source: Wikimedia commons

Unified Modeling Language (UML)

- Notation for OO Modeling
  - Use object orientation as basis
  - Model a system as collection of objects that interact with each other

- Graphical diagrams as a way to model systems
  - More clear (imprecise) than natural language (too detailed)
  - Capture an overall view of the system
  - Independent of language or technology

What UML is not?

- Not an OO Method or Process
- Not a visual programming language
- Not a tool specification

UML Diagrams

- 14 different diagrams
- Structure diagrams

for capturing static aspects of system

- Behavior diagrams

for capturing dynamic aspect of system

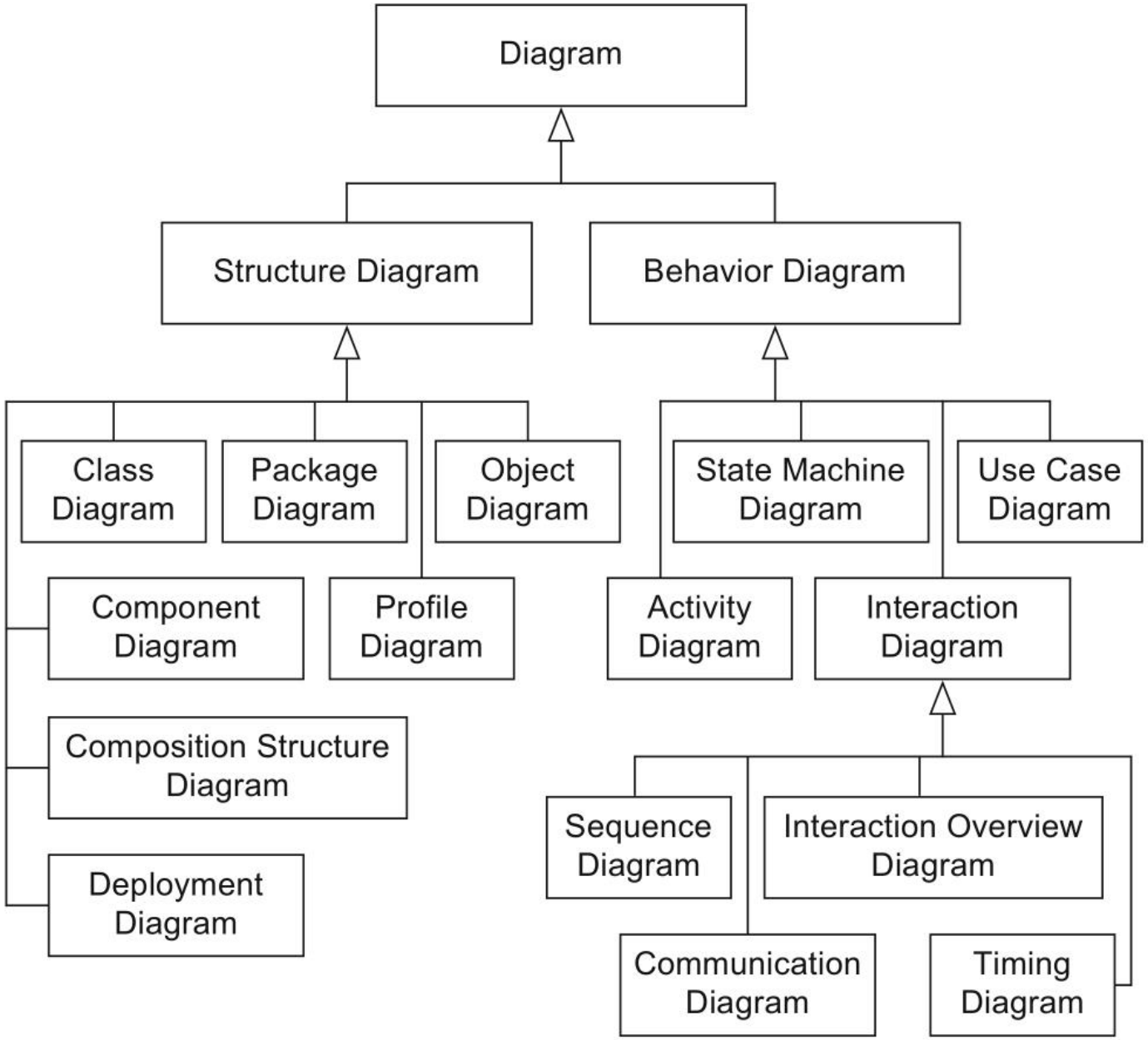


Image source: UML@classroom

Static Vs Dynamic Models

Static Model

- Describes the static structure of a system
- One of the most common diagrams: *class diagrams*

Dynamic Model

- Captures the dynamic behavior of a system
- Developed with help of state chart diagrams, sequence diagrams, etc.

In this unit: class diagram (static) and sequence diagram (dynamic)

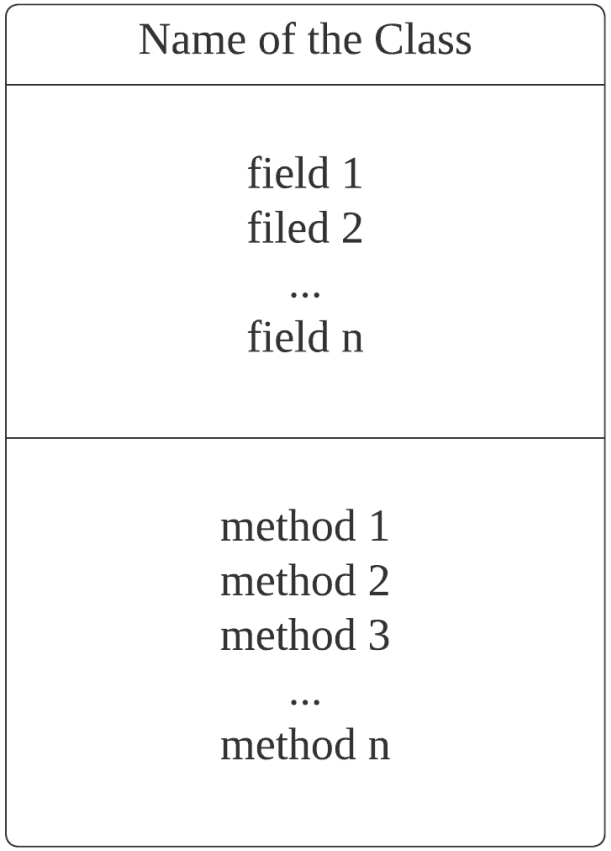
UML Class Diagram

UML Class Diagram

- Most common diagram in OO modeling
- Captures the static structure of a system
- Intuitively it is like a graph
  - Nodes represent the classes
  - Links represent the relationship among classes
    - Inheritance
    - Association (aggregation, composition)
    - Dependency

UML Class Diagram: Notation

Consists of three compartments



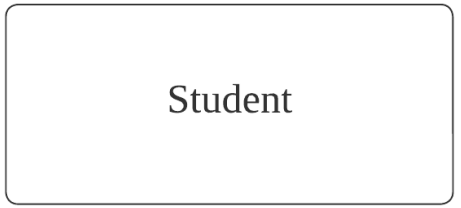
Class name - Pascal Casing, Singular noun, domain vocabulary

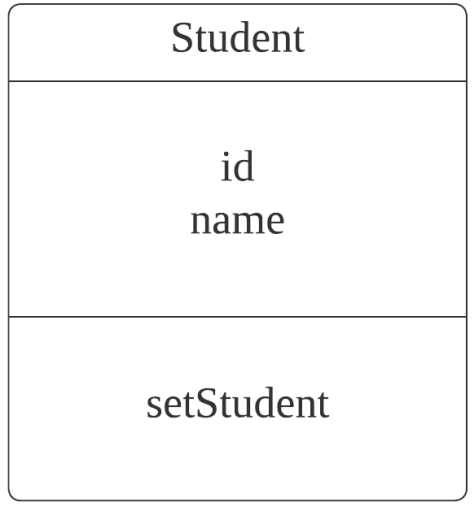
Fields/Attributes (state) - camel casing, name and type at basic level

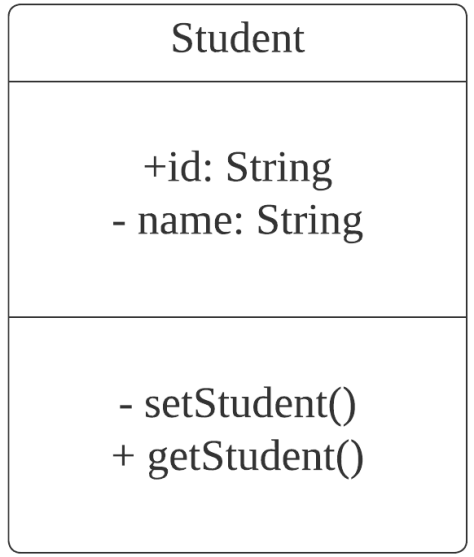
Methods/operations (behavior) – camel casing, name, parameters, return value

UML Class Diagram: Always make use of abstraction

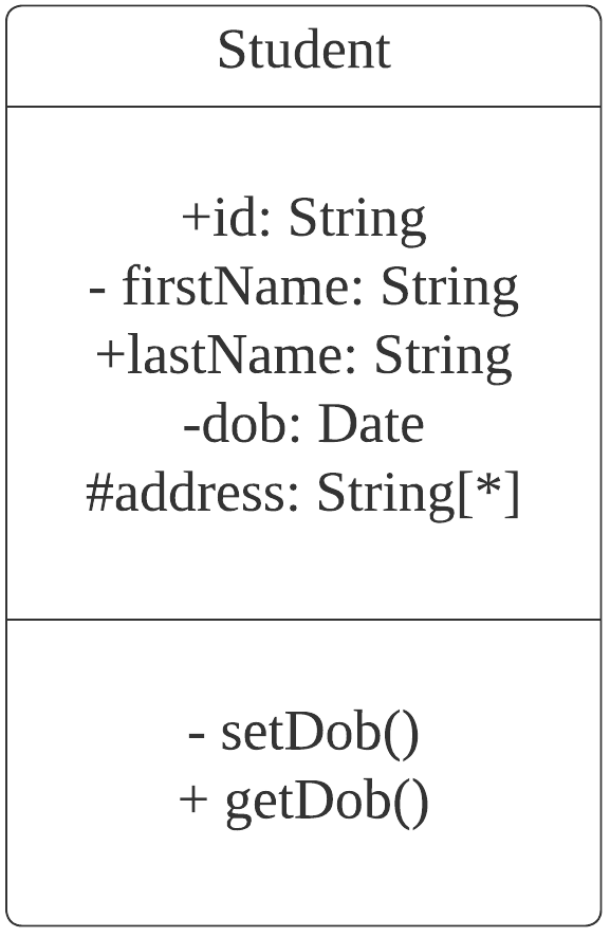
- Model has to be clear and understandable
- Detail with respect to the stage of software development process
- More low-level analysis and development requires detailed information







UML Class Diagram: Specifying Attributes and Methods



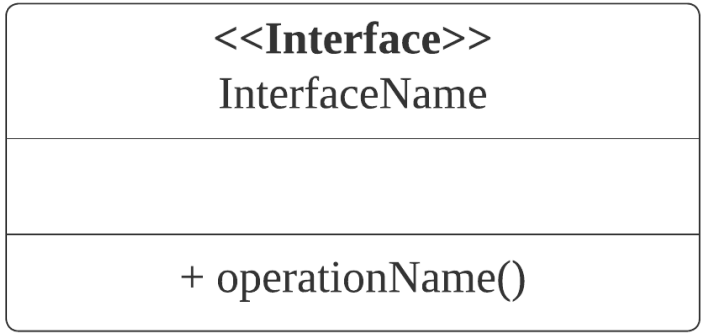
Name and Symbol	Description
public (+)	Access by objects of any class
Private (-)	Access only within the object
Protected (#)	Access by objects of same classes or sub-classes
Package (~)	Access by objects of the classes which are in same package

Create a class diagram for the following code

```java
public class Course {
    public String courseName;
    public String courseId;
    private String roomNumber;
    protected int count;

    public String getCourseName() {
        return courseName;
    }

    public String getCourseId() {
        return courseId;
    }

    private String getRoomNumber() {
        return roomNumber;
    }
```

Interface and Notation for Interfaces

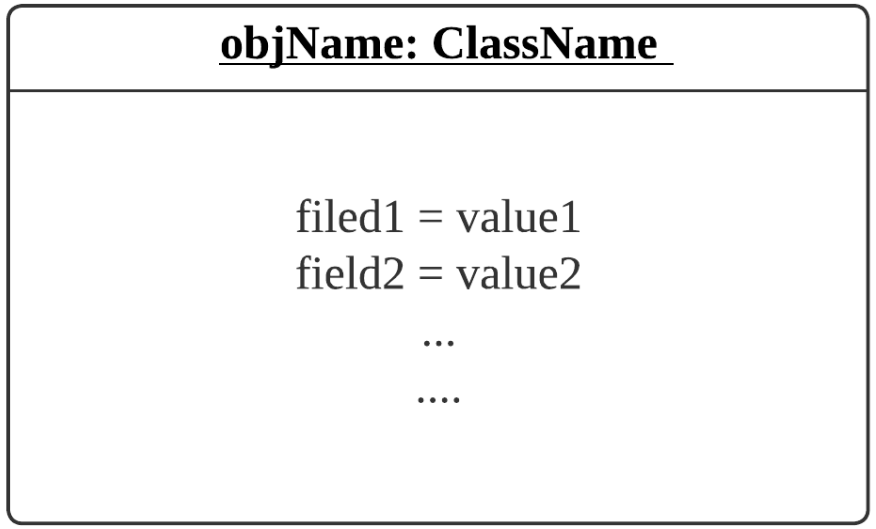
- In simple terms it's a contract mechanism – What to do!
- Mechanism to achieve abstraction, group classes, enforcer – No instance variables only constants
- Class can implement an interface – "implements" keyword (Java)

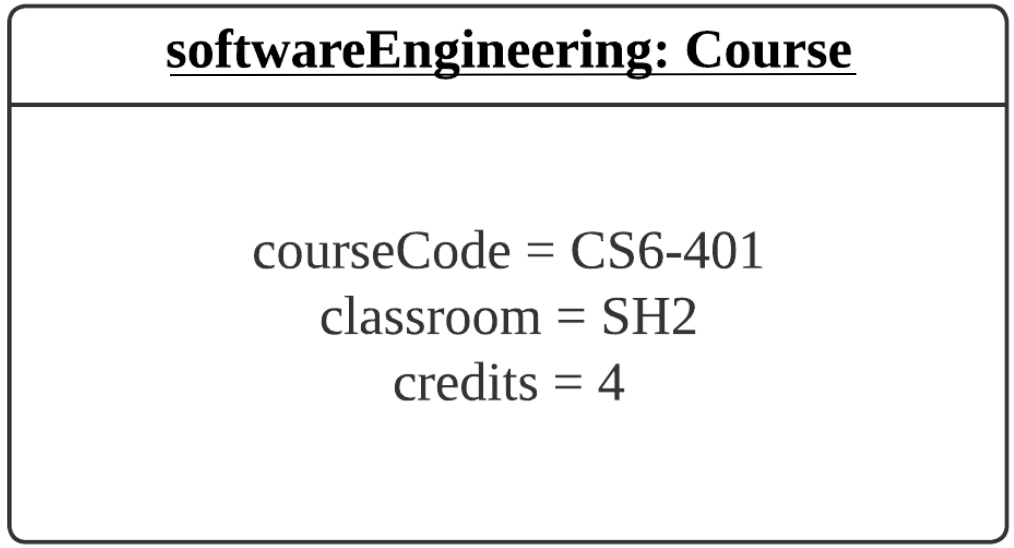


- Vehicles can implement Gear interface

Notation for Objects

- Box with one or two compartments
- Remember to mention the class name

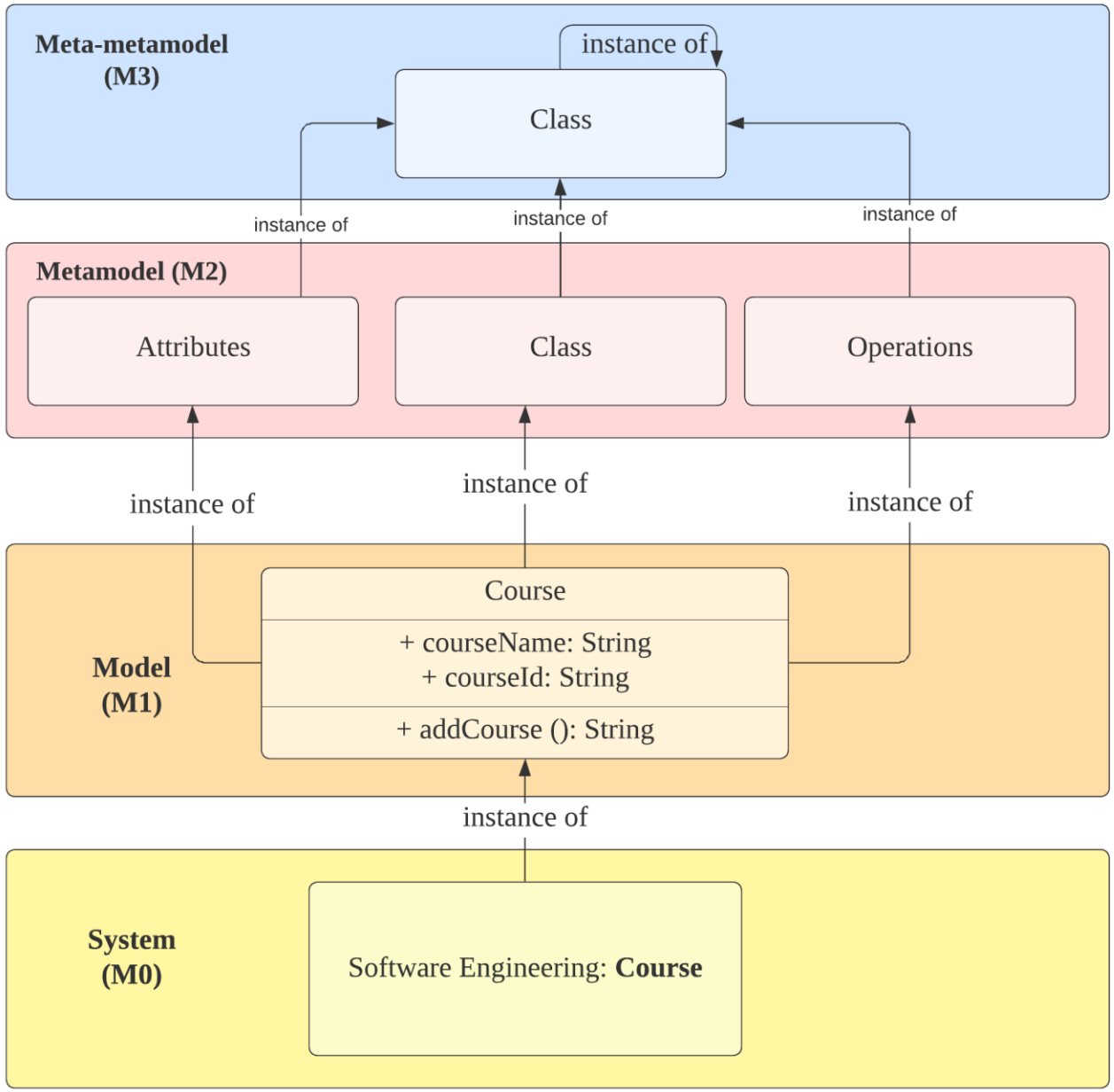




First part has object name and corresponding class name
Second part has list of fields and values

Models and Meta models

- Models of models
- Defines the rules for the different models
- For eg: a class needs to be defined in a particular way



Modeling Relationships using UML

Three main relationships between classes

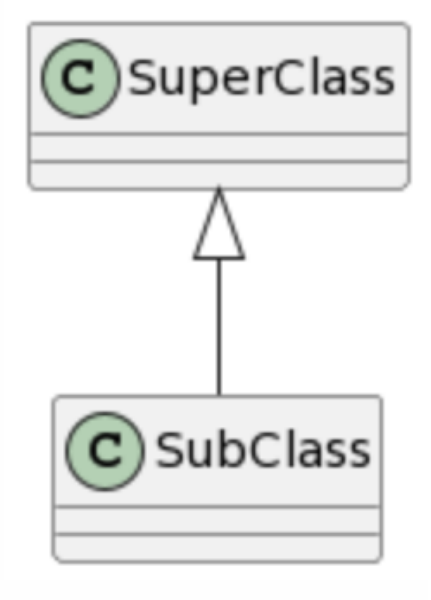
- Dependency
  - Class A uses Class B
- Associations (has-a)
  - Class A affects Class B
  - Types: Aggregation and Composition
- Generalization (Is-a)
  - Class A is a kind of Class B

Inheritance in Java

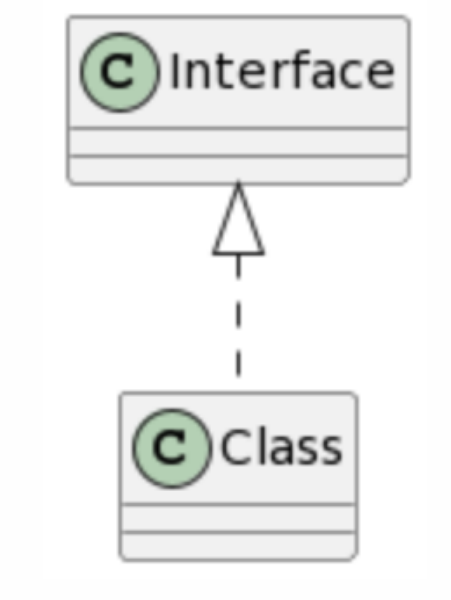
- Object acquires properties and behavior of parent object
- Create new classes based on existing classes
  - Derive classes from existing classes (”extends” keyword)
  - Parent class/super class – Class from which other classes are derived
  - Child class/sub class – Class that is derived from existing class
- **Object** class is the parent class for every class in java (java.lang.package)
- Eg: Vehicle class can be parent of car, bikes, etc.
  - Each car, bike can themselves be parent class for child classes – **How?**

Inheritance in UML

- UML provides easy ways to represent inheritance
  - Extension is called *specialization (sub class) and generalization (supper class)*
  - Implementation is called *realization*

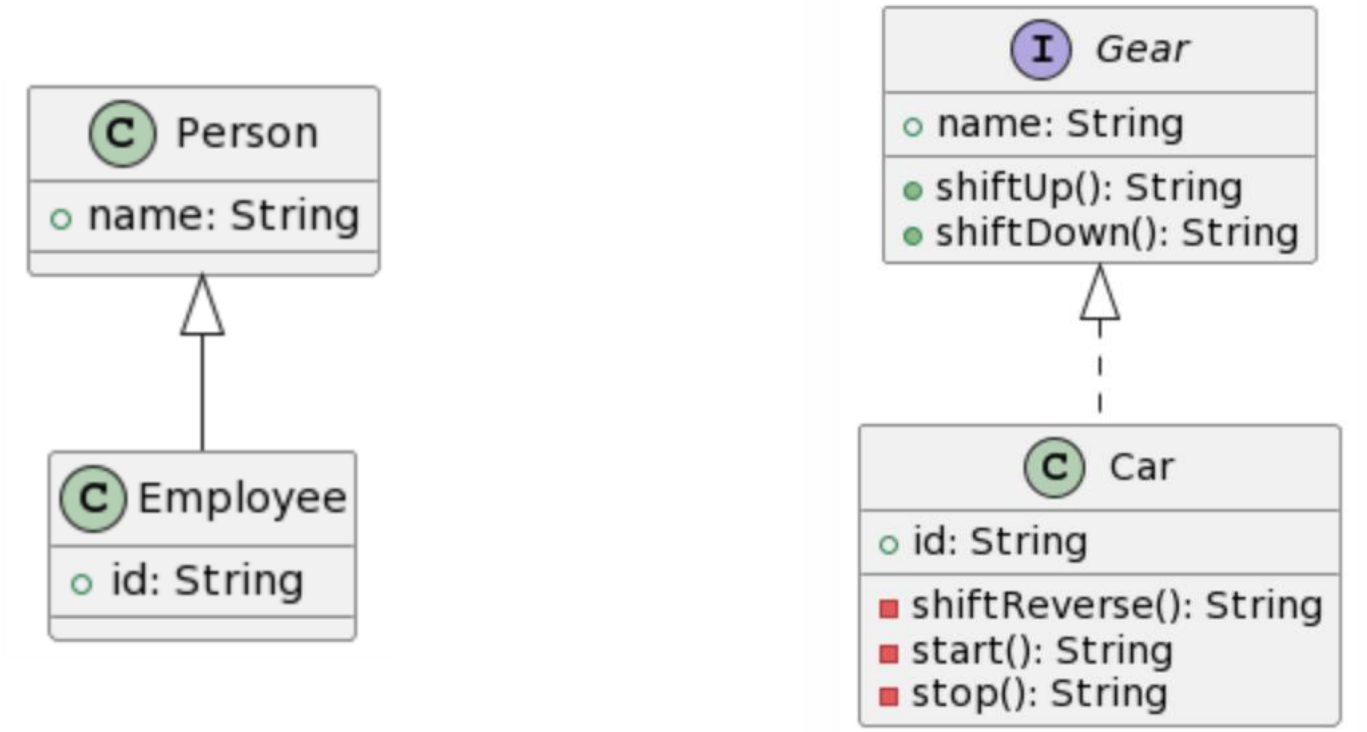


Extension of classes



Realization of interfaces

More Concrete Example



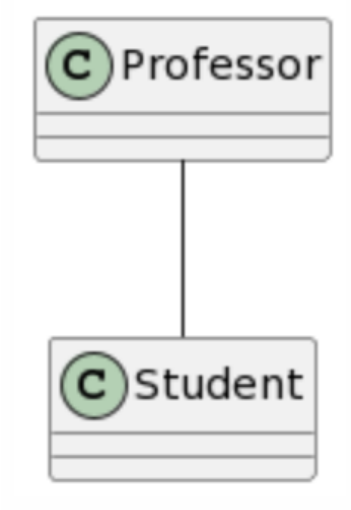
Time to be Creative

Draw a UML diagram showing possible inheritance relationship between different types of students in the class. What will be the abstract class (es)?

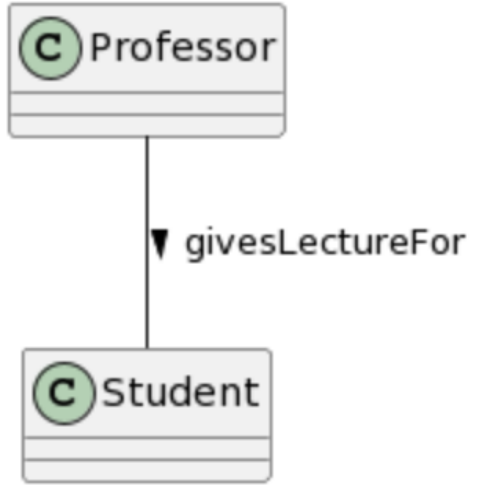
Hint: We have B.Tech, M.Tech, …..

Association

- Model links between instances of classes
- Identify the communication partners
- Use association names and reading directions (solid arrowhead) for labeling



What kind?

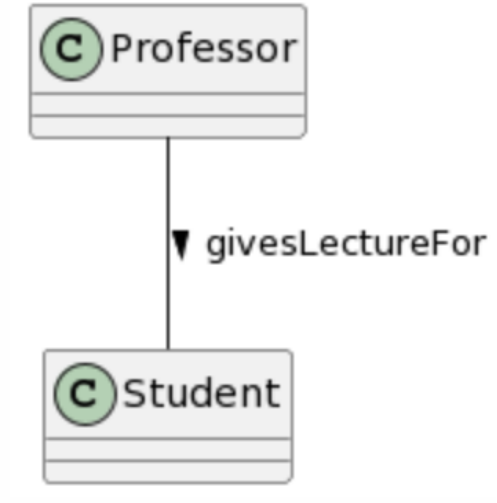


What about multiplicity?

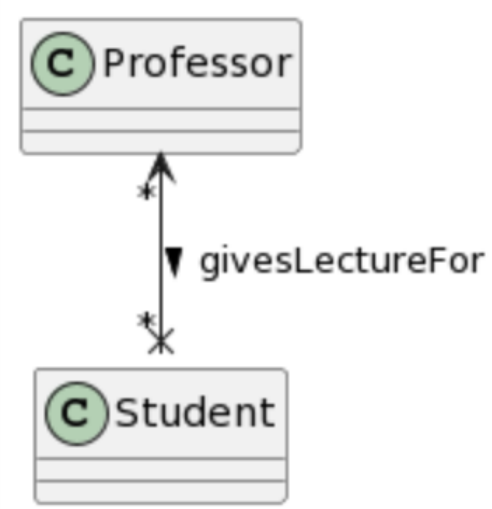
Professors gives lecture

Association – Navigability and Multiplicity

- Cardinality of the class in relation to the another - Multiplicity
- Navigation from one to another is possible – Navigability
- Navigability - Indicates who can access what (not reading direction)
- Usual assumption: Bidirectional navigability



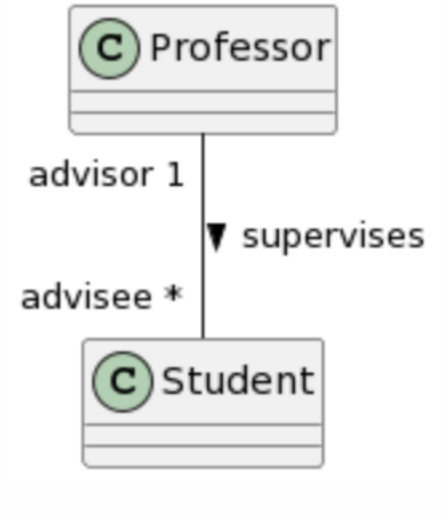
Bidirectional



Professor class can access public parameters/methods of student

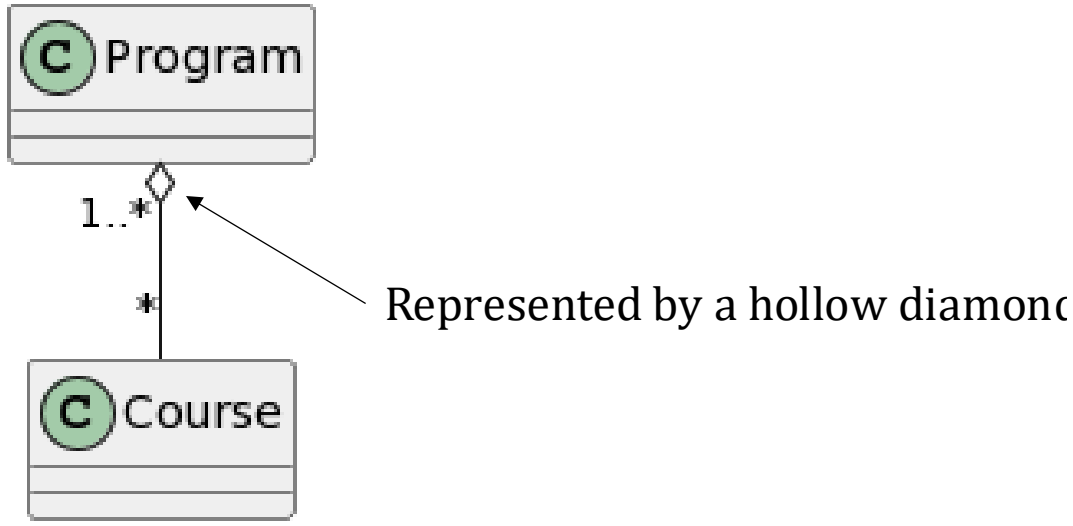
Association – Few more things

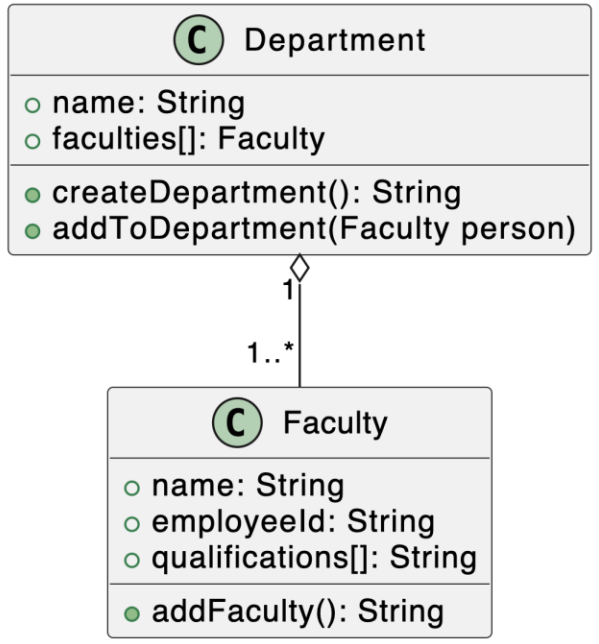
- May have optional role name
- Multiplicity specification is not always mandatory
  - min…max: closed (inclusive) range of integers
  - n: single integer
  - 0..*: entire set of non-negative integers



Aggregation

- Special form of association - Parts-whole relationship
- Used to express that a class is part of another (hollow diamond)
- Combination of independent objects (eg: Program and course)

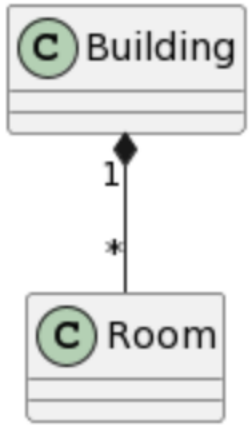




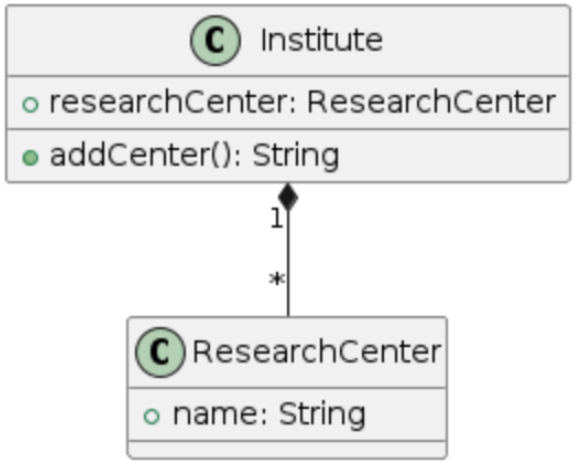
Another example

Composition

- Dependency between composite objects and its parts
- If the composite object is deleted, the parts are also deleted
- One part can be contained in at most one composite object at a time
  - Max multiplicity at the aggregating end is 1 (closed diamond representation)

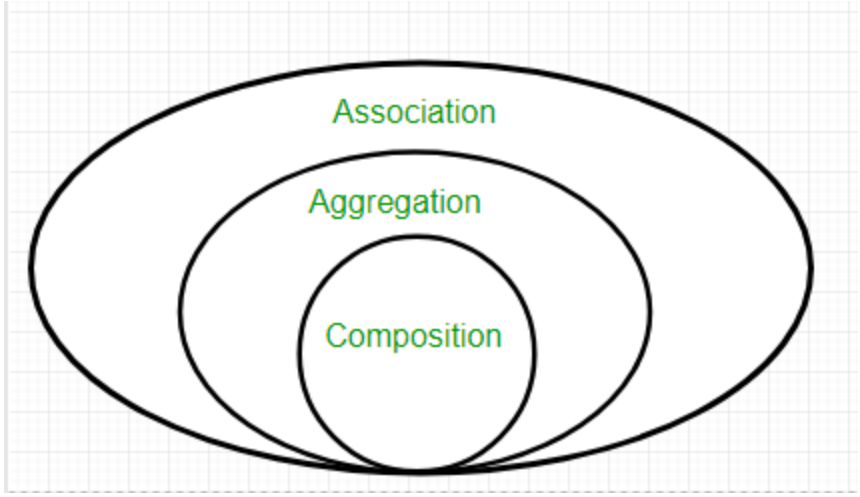


Building is composed of multiple rooms



Adding centers from Institute

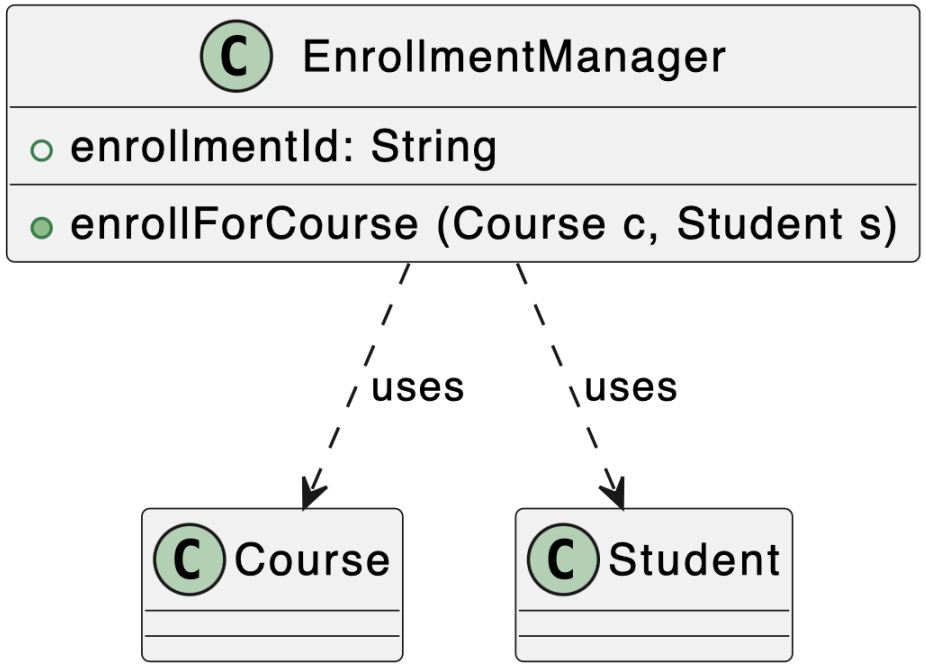
Association Aggregation Composition



https://www.geeksforgeeks.org/association-composition-aggregation-java/

Dependency

- One class uses another class <<uses> relationship
- There is no conceptual link between the objects of the classes
- One may refer the other or vice versa



Time to be Creative

Let’s revisit the case this time with class diagrams:

we want to build a course management portal (think of moodle), what could be some of the classes the corresponding attributes and methods? Can you think of some interfaces?

Thank You

Course website: karthikv1392.github.io/cs6401_se

Web: https://karthikvaidhyanathan.com