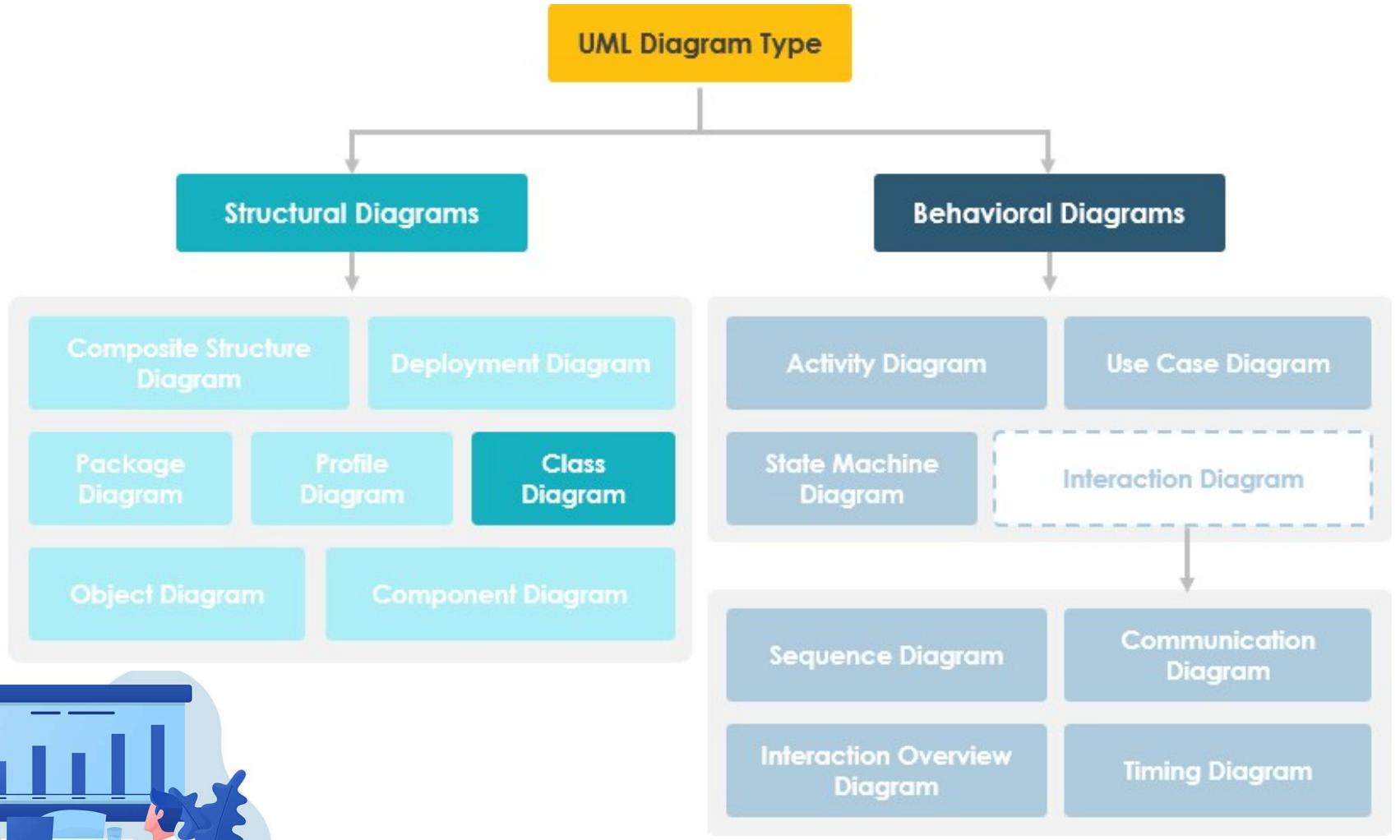
Class Diagram

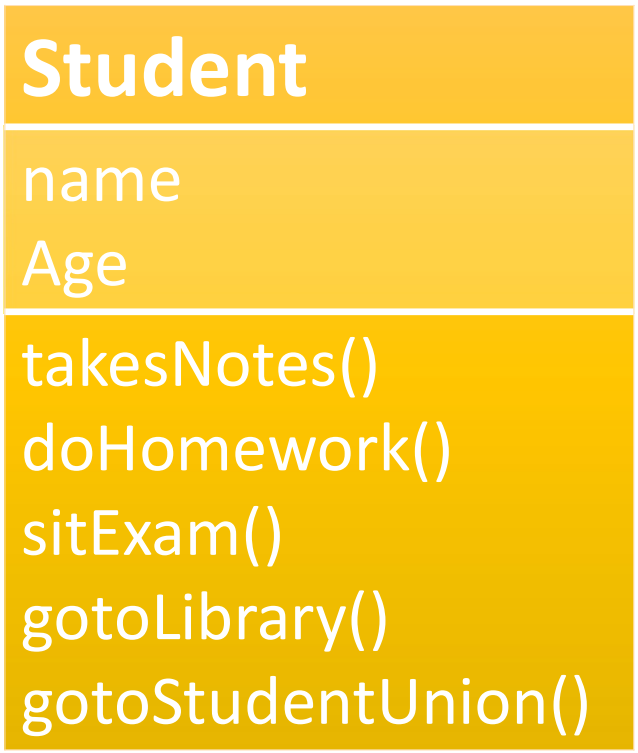
kst@ucsc.cmb.ac.lk
ays@ucsc.cmb.ac.lk


UML Diagram Type
Structural Diagrams
Behavioral Diagrams
Composite Structure Diagram
Deployment Diagram
Package Diagram
Profile Diagram
Class Diagram
Object Diagram
Component Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram

What is Class Diagram?

- The class diagram represents a static view of an application. It represents the different types of objects in the system, as well as their relationships. A class is formed up of objects and can inherit from other classes. A class diagram is used to visualize, describe, and document various aspects of the system, as well as to generate executable software code.

Class Notation



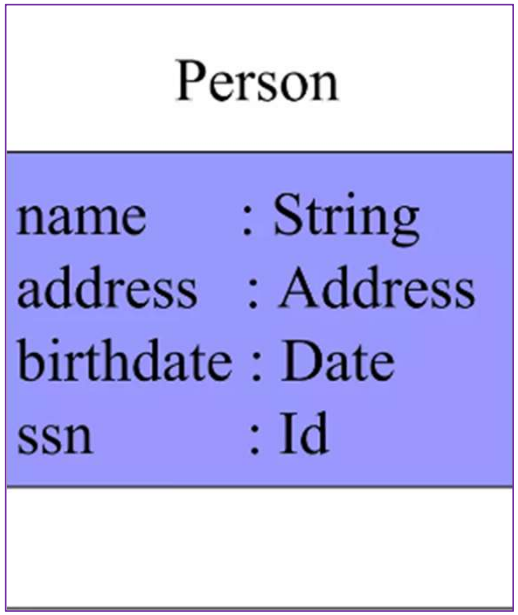
Class Name

Class Attributes

Class Operations

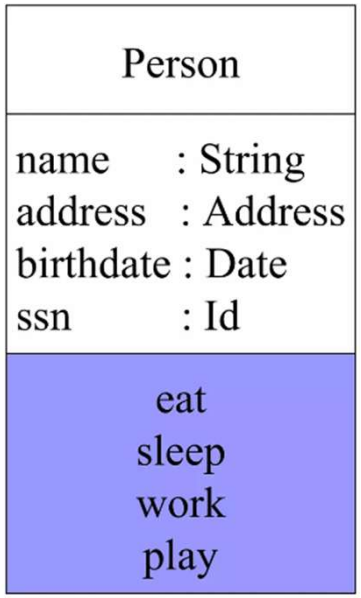
Class Attributes

- An attribute is a named property of a class that describes the object being represented. In the class diagram, attributes are in the second compartment, just below the name compartment.



Class Operations

- Operations describe the class behavior and appear in the third compartment. They are services the class provides.
- The return type of a method is shown after the colon at the end of the method signature.



Visibility

Visibility	Description	Symbol
Public	Is directly accessible by an instance of any class	+
Private	May only be used by an instance of the class that include it	-
Protected	May be used either by an instance of the class that includes it or by a subclass of that class	#
Package	Is directly accessible only by instances of a class in the same package	~

Exercise

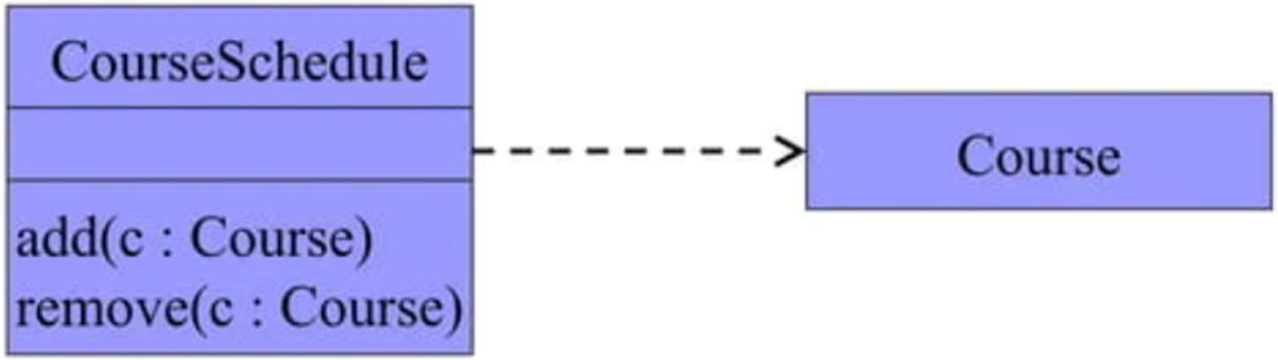
- For example: The Student class encapsulates student information such as student id #, student name, and total student. Student id, student name, and so on are the attributes of the Student class. The Student class also exposes functionality to other classes by using methods such as getStudentName(), getStudentId(), getEmailAddress(), and getTotalStudents(). Draw a student class using both attributes and methods.

Exercise

Student
-name: String -id: int -totalStudents: int
#getID(): int +getName(): String ~getEmailAddress(): String +getTotalStudents(): int

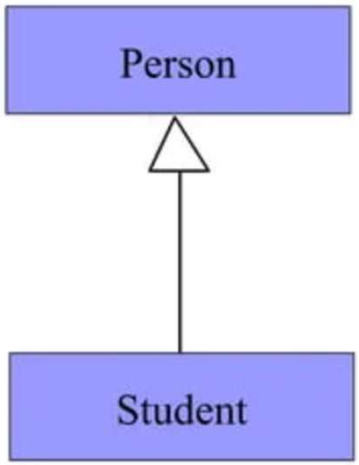
Dependency Relationships

- A dependency indicates a semantic relationship between two or more elements. The dependency form courseSchedule to Course exists because Course is used in both the add and remove operations of CourseSchedule.



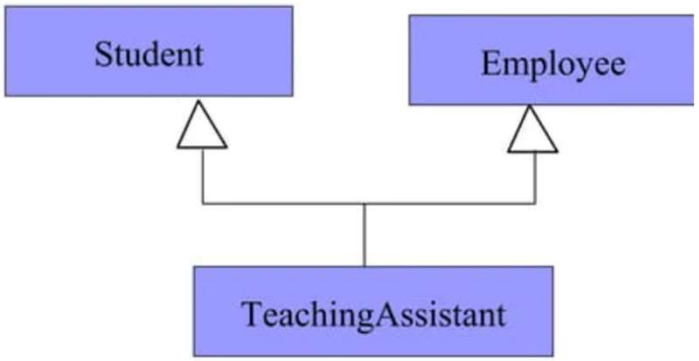
Generalization Relationships

- A generalization connects a subclass to its superclass. It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.



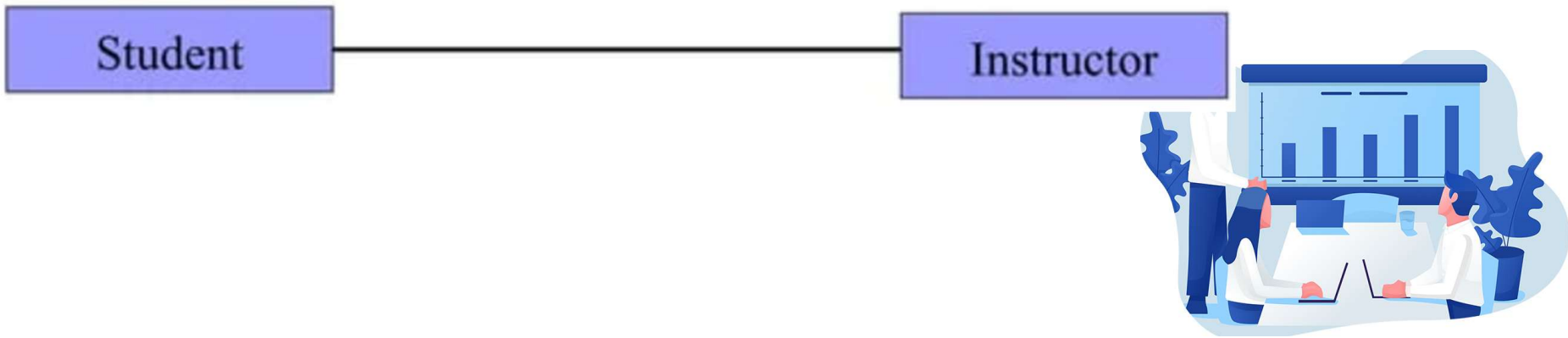
Generalization Relationships

- UML permits a class to inherit from multiple super classes, although some programming languages (eg: Java) do not permit multiple inheritance.



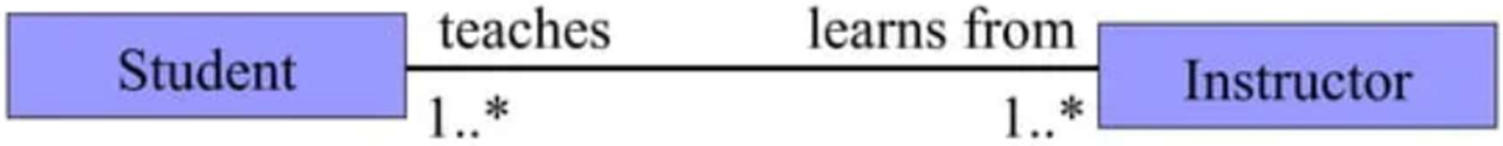
Association Relationships

- An association between two classes indicates that objects at one end of an association “recognize” objects at the other end and may send messages to them.



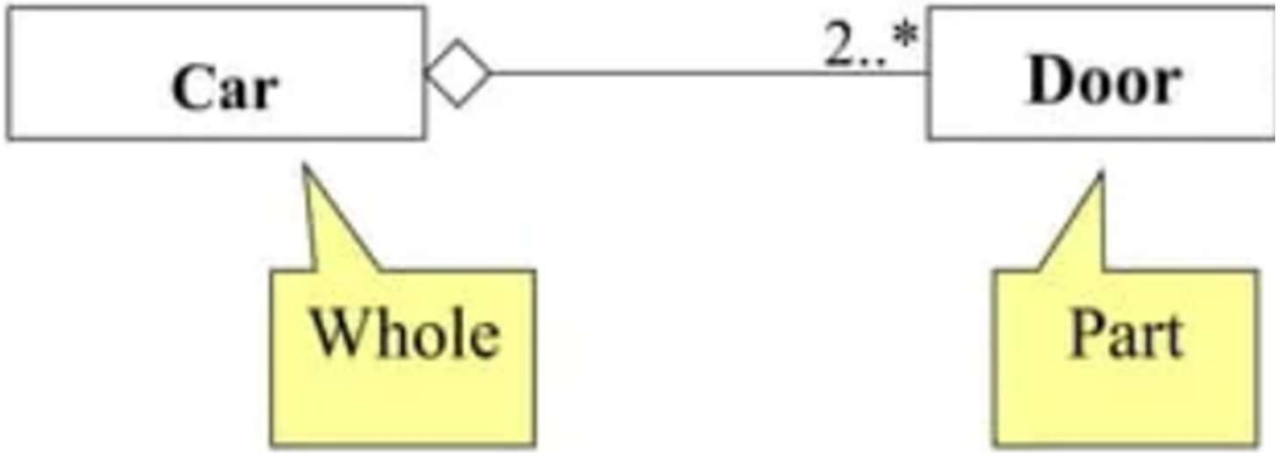
Association Relationships

- If two classes in a model need to communicate with each other, their must be link between them.
- Example: “A student has one or more instructor”
- “Every Instructor has one or more students”



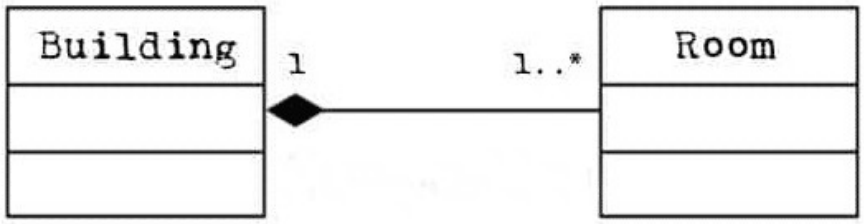
Aggregation

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.



Composition

- A strong form of aggregation:
  - The whole is the sole owner of its part. The part object may belong only one whole.
  - Multiplicity on the whole side must be zero or one.
  - The life time of the part is dependent upon the whole. The composite must manage the creation and destruction of its parts.



Multiplicity

- The number of objects that participate in the association.
- Indicates whether or not an association is mandatory

Multiplicity	Indicators
Exactly one	1
Zero or more(unlimited)	*(0..*)
One or more	1..*
Zero or one(optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8