**Lab 4: Class diagram**

## **Purpose**

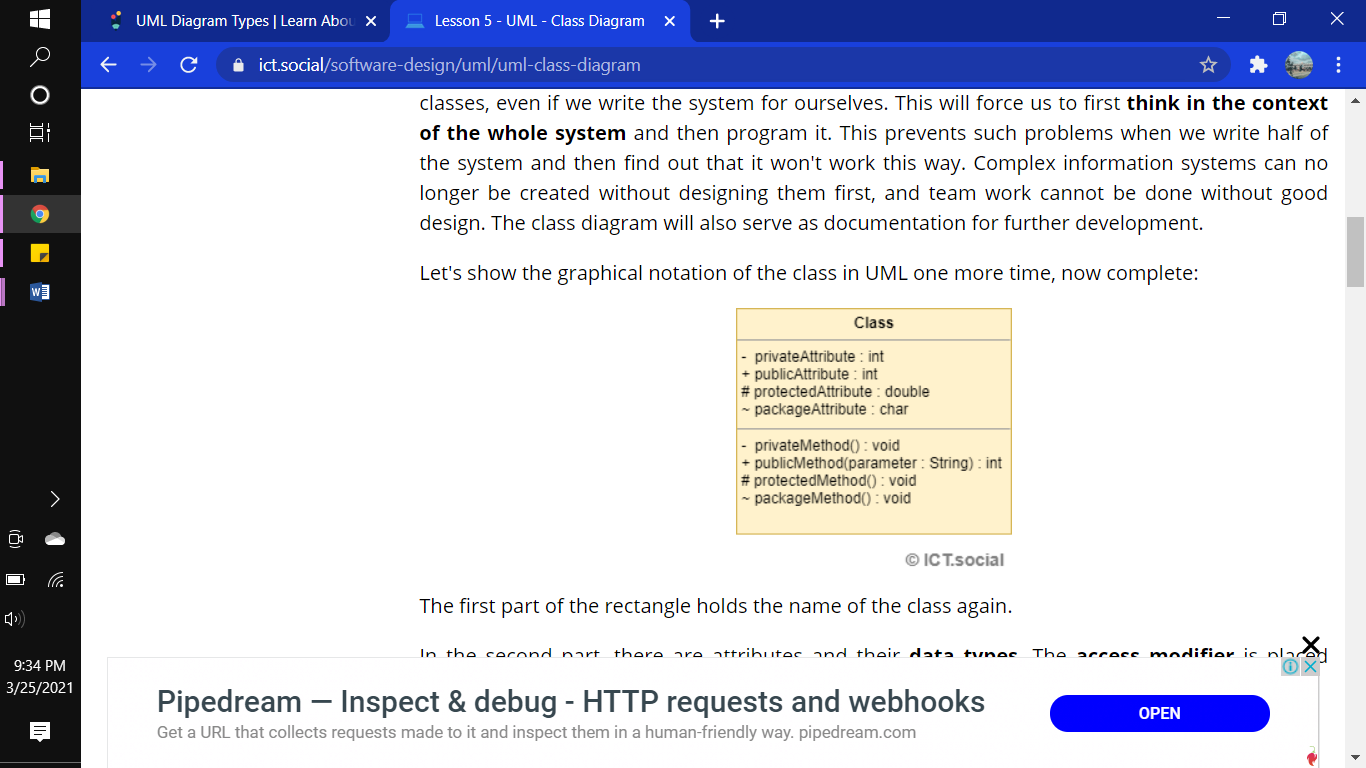
The purpose of this lab is to identify the syntax class diagram attributes and functions of each conceptual class.

## **Theoretical Description**

Describe the structure of the system in terms of classes and objects. It shows the classes in a system, attributes, and operations of each class and the relationship between each class.

In most modeling tools, a class has three parts. Name at the top, attributes in the middle and operations or methods at the bottom. In a large system with many related classes, classes are grouped together to create class diagrams. Different relationships between classes are shown by different types of arrows.

The below figure shows how a class diagram can be created.

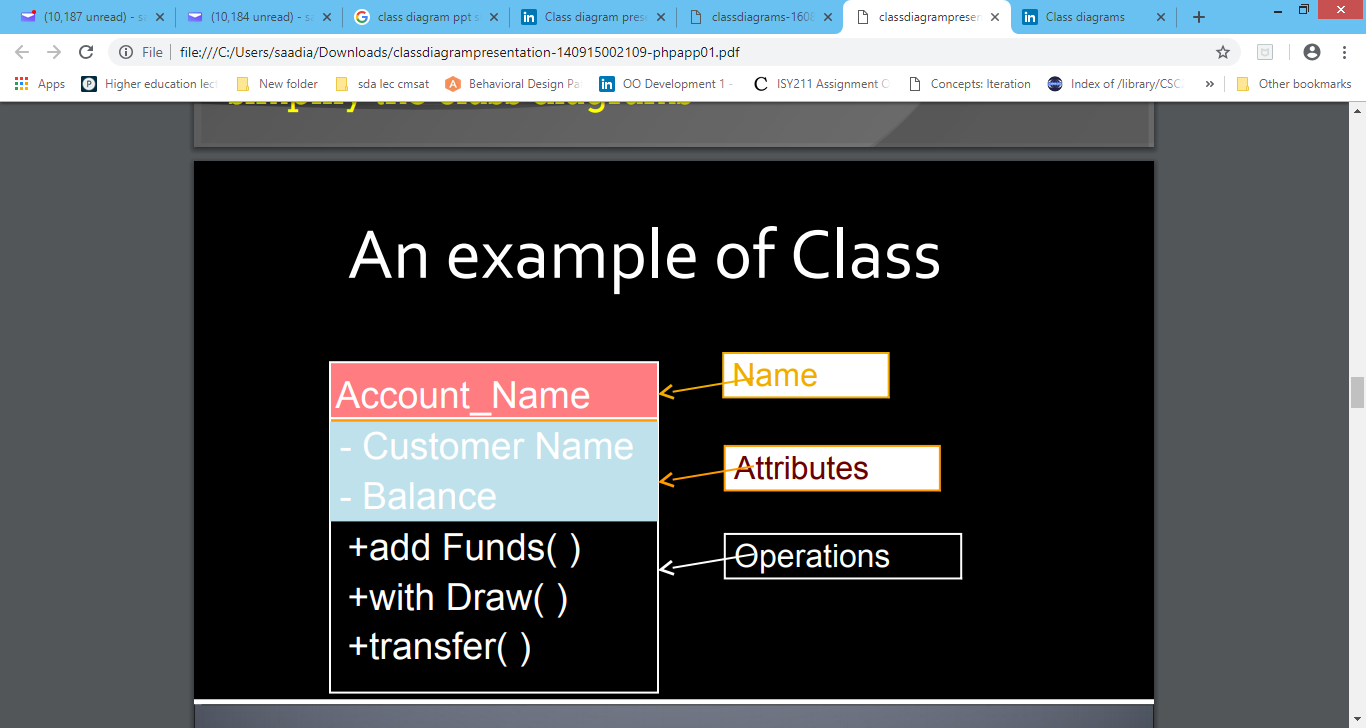


The first part of the rectangle holds the name of the class again.

In the second part, there are attributes and their **data types**. The **access modifier** is placed before each attribute. We have 4 options:

* **- (minus)** - Private attribute.
* **+ (plus)** - Public attribute.
* **# (hash cross)** - Protected attribute.
* **~ (tilde)** - Attribute visible from within the package.

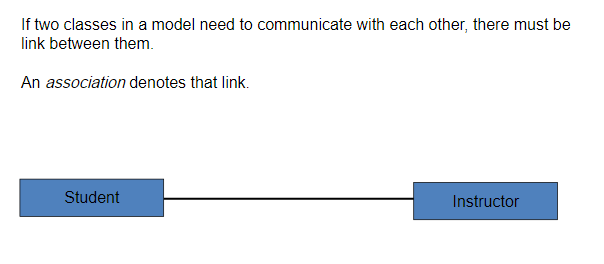
**Example of class**

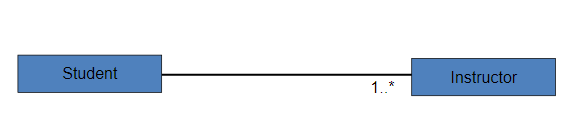


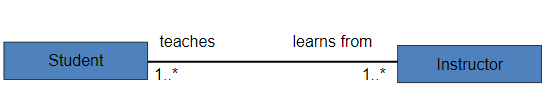
Four types of **relations** occur between different classes. These relations are:

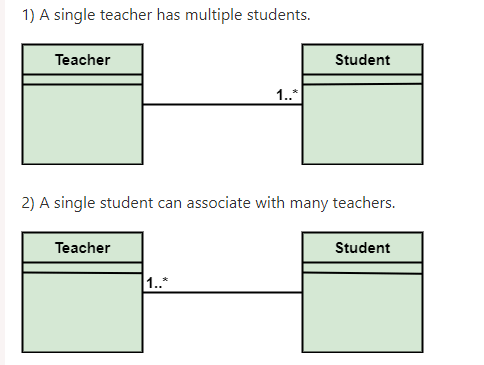
* **Association**

An association is the basic relationship between two entities. These entities can exist independently of each other. We draw it as a simple solid line. The association is bi-directional by default. That means that the first entity has a reference to the other, and the other to the first. We can change this behavior by adding a simple arrow specifying the direction of the relationship. Only the instance from which the arrow points stores the reference to the other entity in these cases.







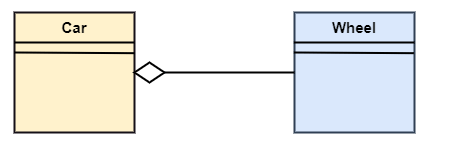


* **Aggregation**

Aggregation represents the relationship between a whole and its parts. We draw it as a solid line with an empty diamond shape. The diamond is drawn at the class representing the whole (e.g. an article section). From the implementation point of view, this is the entity that holds the item collection. An entity representing the part can exist independently and be part of other collections.

**Example**

consider a car and a wheel example. A car cannot move without a wheel. But the wheel can be independently used with the bike, scooter, cycle, or any other vehicle. The wheel object can exist without the car object, which proves to be an aggregation relationship.

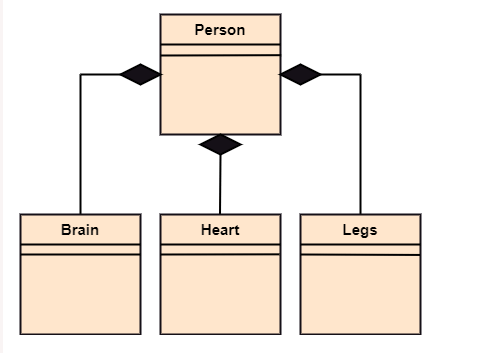
****

* **Composition**

A composition is similar to aggregation, but it represents a stronger relationship. The entity representing the part has no sense without the entity representing the whole. If the entity representing the whole is removed, its parts are automatically removed as well. We draw the composition relationship like the aggregation, but the diamond shape is filled. The multiplicity of the entity representing the whole must always be 1. This relationship is confusing and I'd rather avoid it and replace it with aggregation.

**Example**

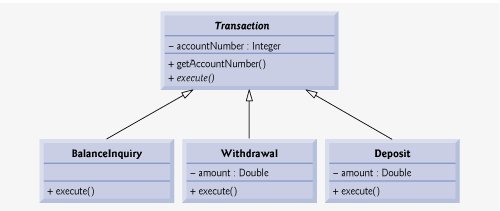
The composition association relationship connects the Person class with Brain class, Heart class, and Legs class. If the person is destroyed, the brain, heart, and legs will also get discarded.

****

* **Generalization**

In terms of implementation, it represents inheritance. One entity inherits the properties and behavior from another entity. We've already seen this kind of relationship in the use case diagram. We draw the generalization as a solid line with an empty arrow on one side (a triangle if you like). The arrow is on the side of the entity from which it's inherited.

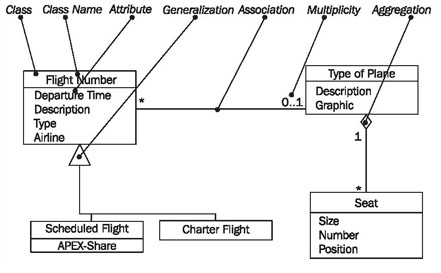
EXAMPLE

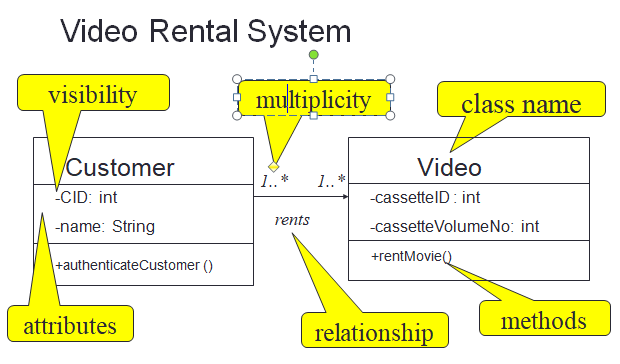


* **Multiplicity**

We can specify the multiplicity for association, aggregation, and composition (for composition on one side only). Let's go back to the example with a section and an article: We read the multiplicity here as follows: A section can contain any number of articles (this is shown by the asterisk at the Article class). An Article belongs to 1 or more sections (it's shown by 1..\* at the Section). Let's list available multiplicity syntax:

1. **1 (number)** - Indicates a specific value (1 in this example).
2. **\* (asterisk)** - Indicates any number (even 0). Instead of an asterisk, we can find the N symbol in some diagrams.
3. **1..\* (interval)** - We can specify an interval with 2 dots. Then we use the symbols we already known, such as:2..5,  1..\* or 0..1.

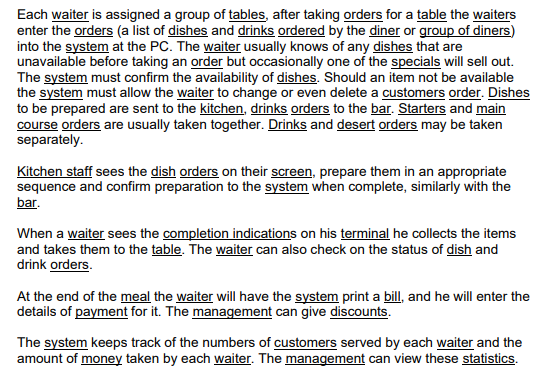


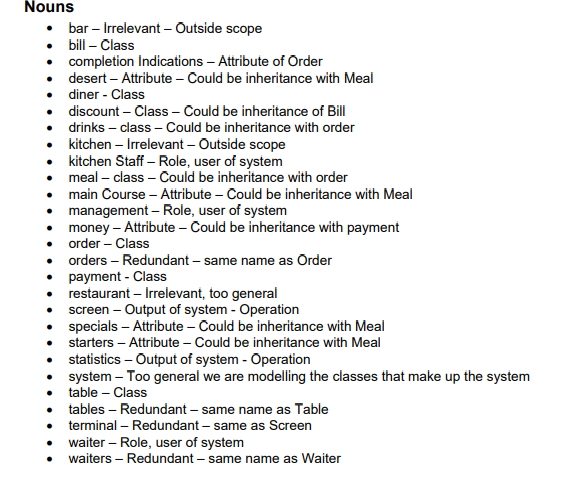
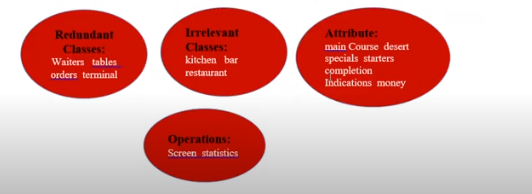
**EXAMPLE OF CLASS DIAGRAM**

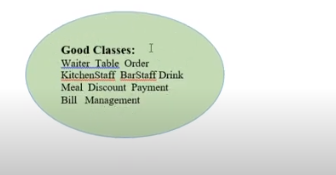
Let’s take a simple example of ‘Order Processing System’ for making class diagram

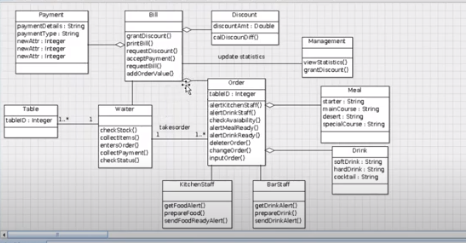


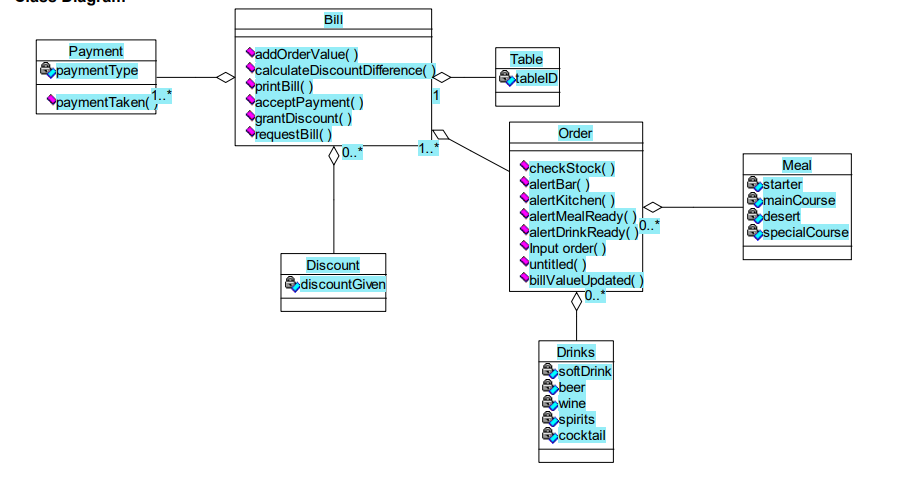
**EXAMPLE**

****

****

****

****

****

**LAB TASK**

**Q1** Draw a class diagram for a scenario of a test system. The process flow is outlined below. Use association according to your own understanding

**i. Students attend Classes;**

**ii. Teachers mark attendance;**

**iii. Teachers prepare test papers;**

**iv. Students write test papers;**

**v. Teachers check test papers;**

**vi. Teachers declare results;**

**Q2 Draw a class diagram for the following scenario.**

A company has a number of employees. The attributes of Employee include employee ID (primary key), name, address, and birth date. The company also has several projects. Attributes of Project include project ID, project name and start date. Each employee may be assigned to one or more projects, or may not be assigned to a project, a project must have at least one employee assigned, and it may have any number of employees assigned. As employee's billing rate may vary by project, and the company wishes to record the applicable billing rate for each employee when assigned to a particular project. At the end of each month, the company mails a check to each employee who has worked on a project during that month. The check amount is based on the billing rate and the hours logged for each project assigned to the employ

**Q3 Select smart draw or visio and make a list of UML classifiers, UML relationships and UMLenhancing features**