OOAD-WEEK 4

UML Class Diagram

The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

Purpose of Class Diagrams

The main purpose of class diagrams is to build a static view of an application. It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages. It is one of the most popular UML diagrams. Following are the purpose of class diagrams given below:

1. It analyses and designs a static view of an application.
2. It describes the major responsibilities of a system.
3. It is a base for component and deployment diagrams.
4. It incorporates forward and reverse engineering.

Benefits of Class Diagrams

1. It can represent the object model for complex systems.
2. It reduces the maintenance time by providing an overview of how an application is structured before coding.
3. It provides a general schematic of an application for better understanding.
4. It represents a detailed chart by highlighting the desired code, which is to be programmed.
5. It is helpful for the stakeholders and the developers.

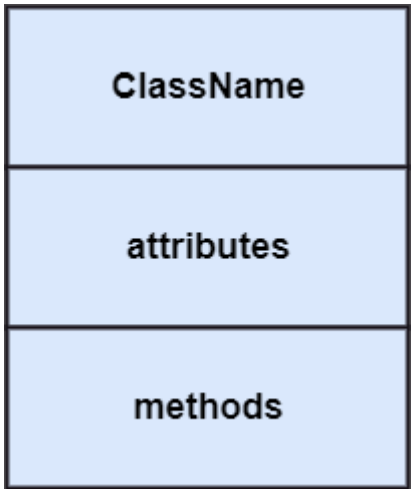
Vital components of a Class Diagram

The class diagram is made up of three sections:

- **Upper Section:** The upper section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics. Some of the following rules that should be taken into account while representing a class are given below:
    a. Capitalize the initial letter of the class name.
    b. Place the class name in the center of the upper section.
    c. A class name must be written in bold format.
    d. The name of the abstract class should be written in italics format.
- **Middle Section:** The middle section constitutes the attributes, which describe the quality of the class. The attributes have the following characteristics:

. The attributes are written along with its visibility factors, which are public (+), private (-), protected (#), and package (~).
    a. The accessibility of an attribute class is illustrated by the visibility factors.
    b. A meaningful name should be assigned to the attribute, which will explain its usage inside the class.

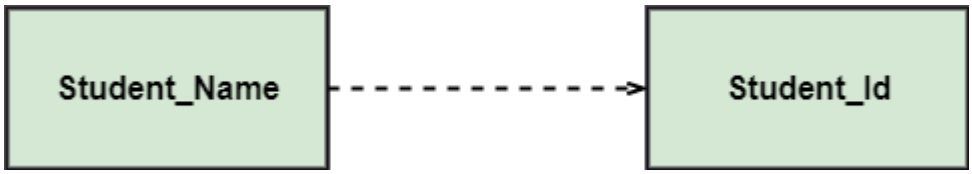
- **Lower Section:** The lower section contain methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.



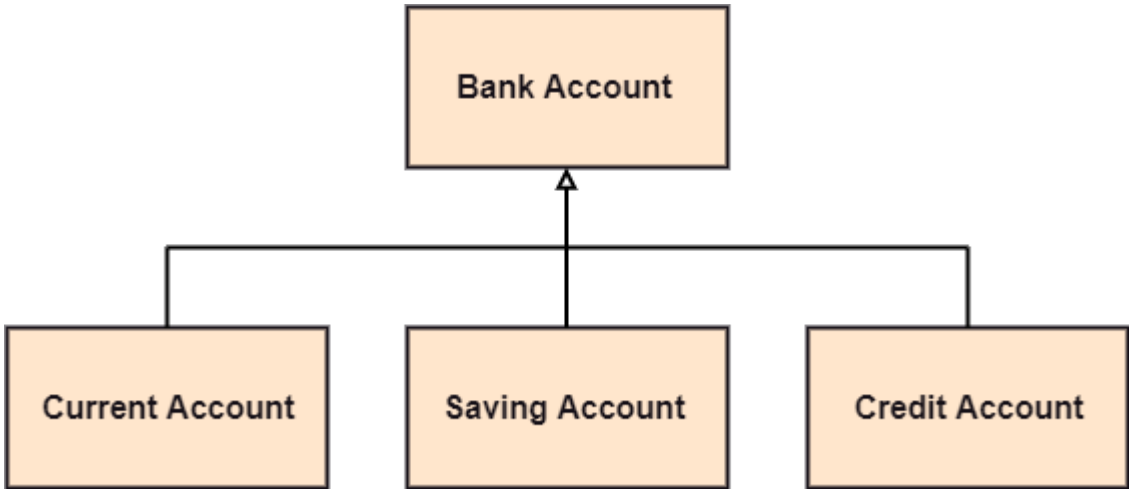
Relationships

In UML, relationships are of three types:

- **Dependency:** A dependency is a semantic relationship between two or more classes where a change in one class cause changes in another class. It forms a weaker relationship.
  In the following example, Student_Name is dependent on the Student_Id.



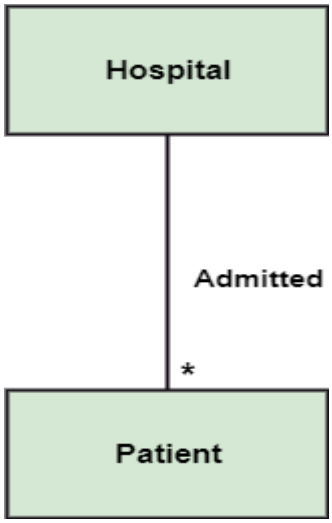
- **Generalization:** A generalization is a relationship between a parent class (superclass) and a child class (subclass). In this, the child class is inherited from the parent class. For example, The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account.



- **Association:** It describes a static or physical connection between two or more objects. It depicts how many objects are there in the relationship.
  For example, a department is associated with the college.

Multiplicity: It defines a specific range of allowable instances of attributes. In case if a range is not specified, one is considered as a default multiplicity.

For example, multiple patients are admitted to one hospital.



Aggregation: An aggregation is a subset of association, which represents has a relationship. It is more specific then association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class can exist independently of its parent class.

The company encompasses a number of employees, and even if one employee resigns, the company still exists.

Composition: The composition is a subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleted, then the other part also gets discarded. It represents a whole-part relationship.

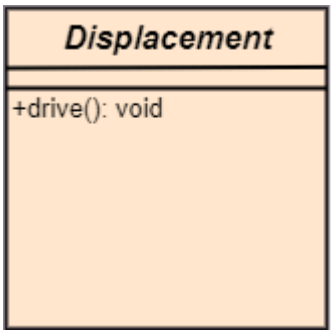
A contact book consists of multiple contacts, and if you delete the contact book, all the contacts will be lost.

Abstract Classes

In the abstract class, no objects can be a direct entity of the abstract class. The abstract class can neither be declared nor be instantiated. It is used to find the functionalities across the classes. The notation of the abstract class is similar to that of class; the only difference is that the name of the

class is written in italics. Since it does not involve any implementation for a given function, it is best to use the abstract class with multiple objects.

Let us assume that we have an abstract class named **displacement** with a method declared inside it, and that method will be called as a **drive ()**. Now, this abstract class method can be implemented by any object, for example, car, bike, scooter, cycle, etc.



How to draw a Class Diagram?

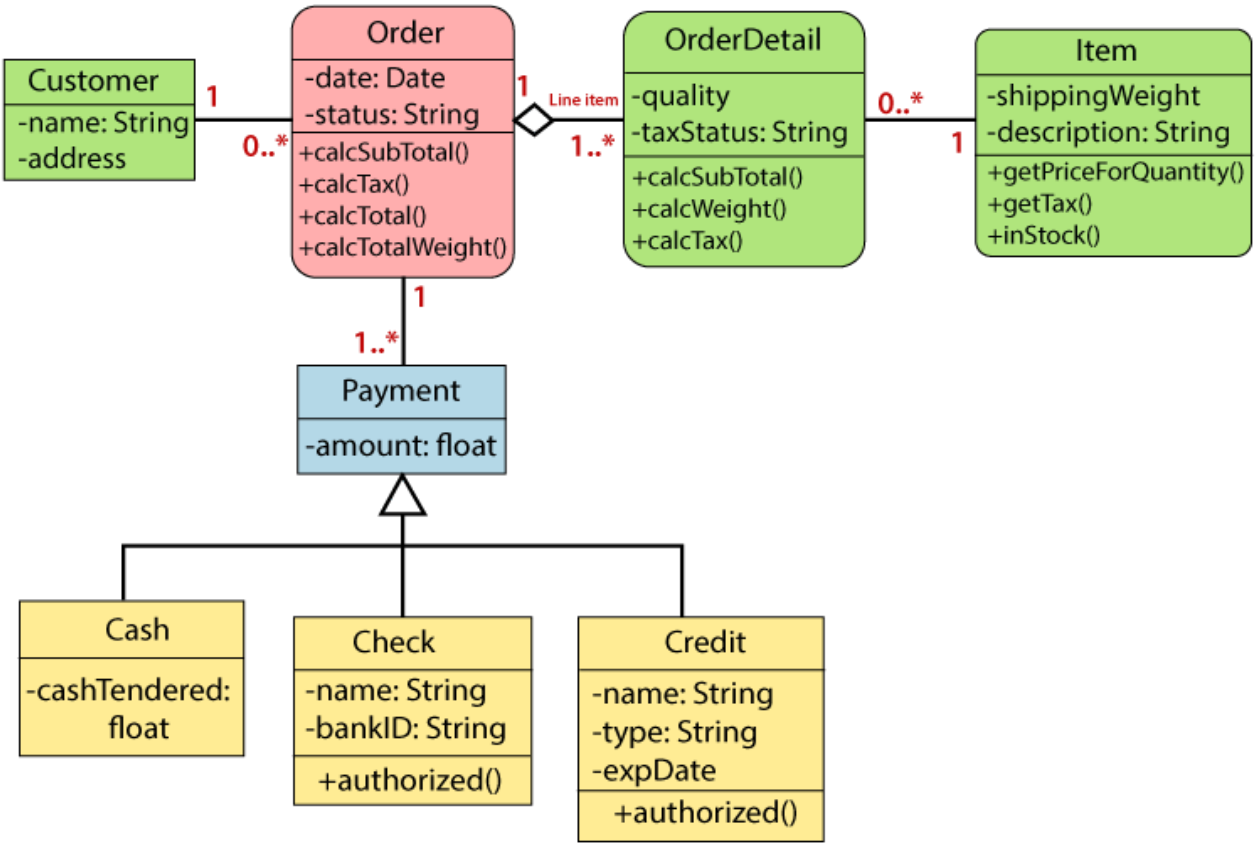
The class diagram is used most widely to construct software applications. It not only represents a static view of the system but also all the major aspects of an application. A collection of class diagrams as a whole represents a system.

Some key points that are needed to keep in mind while drawing a class diagram are given below:

1. To describe a complete aspect of the system, it is suggested to give a meaningful name to the class diagram.
2. The objects and their relationships should be acknowledged in advance.
3. The attributes and methods (responsibilities) of each class must be known.
4. A minimum number of desired properties should be specified as more number of the unwanted property will lead to a complex diagram.
5. Notes can be used as and when required by the developer to describe the aspects of a diagram.
6. The diagrams should be redrawn and reworked as many times to make it correct before producing its final version.

Class Diagram Example

A class diagram describing the sales order system is given below.



Usage of Class diagrams

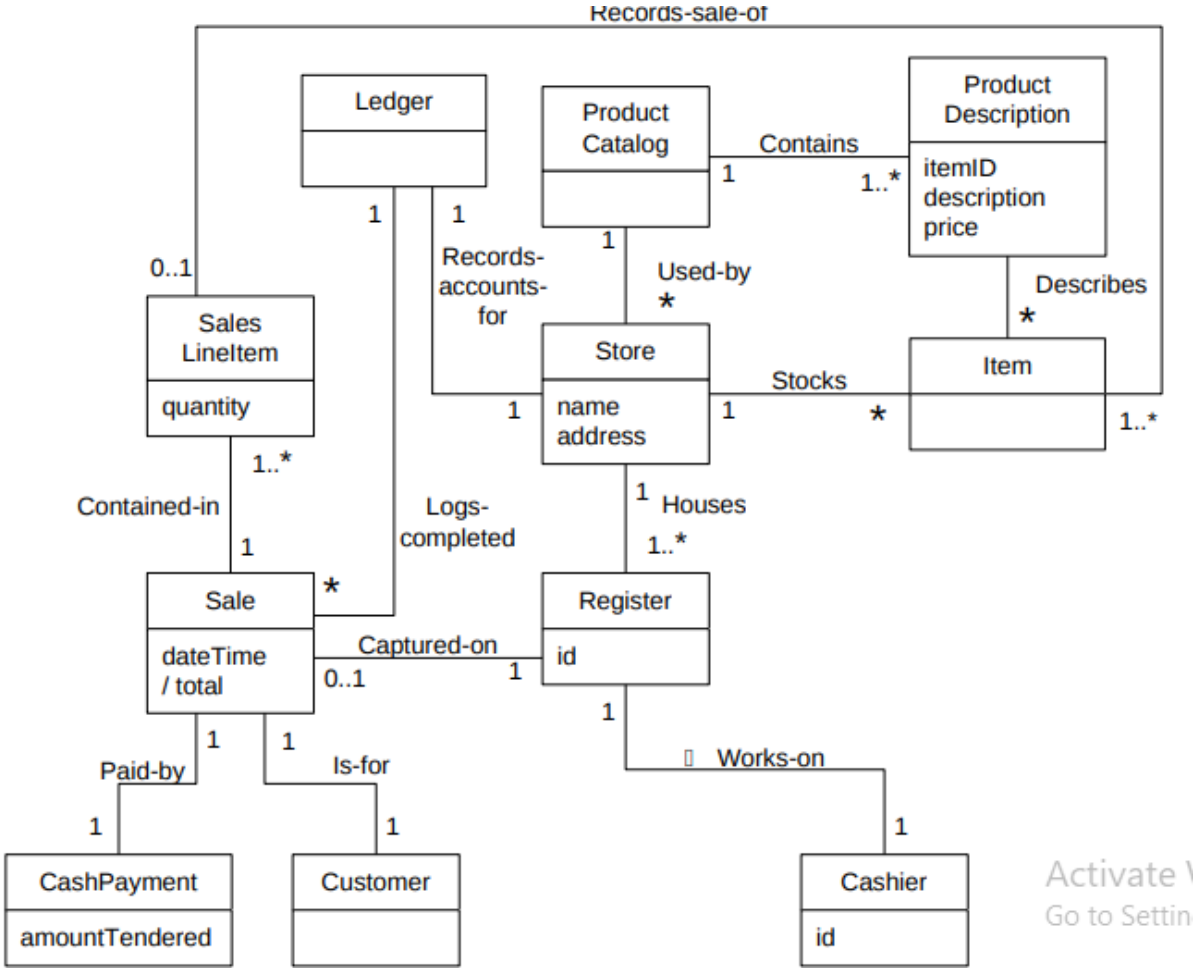
The class diagram is used to represent a static view of the system. It plays an essential role in the establishment of the component and deployment diagrams. It helps to construct an executable code to perform forward and backward engineering for any system, or we can say it is mainly used for construction. It represents the mapping with object-oriented languages that are C++, Java, etc. Class diagrams can be used for the following purposes:

1. To describe the static view of a system.
2. To show the collaboration among every instance in the static view.
3. To describe the functionalities performed by the system.
4. To construct the software application using object-oriented languages.

Domain Model:

A domain model is not a description of software objects; it is a visualization of the concepts or mental models of a real-world domain. Thus, it has also been called a conceptual object model.

Example: NextGen POS



Question:

1. Draw Domain and Design model for your project