**Godavari College Of Engineering, Jalgaon.**

**Subject Name:** PDE **Teacher Name:** Prof. Nilesh Chaudhari Sir.

**Practical No**. : 3 **Date:**

**Class: S**.E **Roll No: 1951711245011**

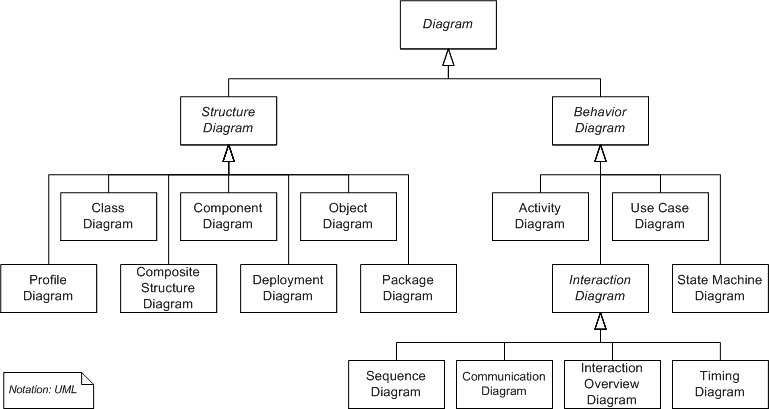
**Title:**   Explain Class Diagram in Software Engineering.

**Class Diagram. :-** In software engineering , a **class diagram** in the Unified Modeling Language(UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the structure of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

In the diagram, classes are represented with boxes that contain three compartments:

* The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
* The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
* The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.



## **Purpose of Class Diagrams**

1. Shows static structure of classifiers in a system
2. Diagram provides a basic notation for other structure diagrams prescribed by UML
3. Helpful for developers and other team members too
4. Business Analysts can use class diagrams to model systems from a business perspective

A UML class diagram is made up of:

* A set of classes and
* A set of relationships between classes

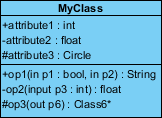
## What is a Class

A description of a group of objects all with similar roles in the system, which consists of:

* Structural features (attributes) define what objects of the class "know"
  + Represent the state of an object of the class
  + Are descriptions of the structural or static features of a class
* Behavioral features (operations) define what objects of the class "can do"
  + Define the way in which objects may interact
  + Operations are descriptions of behavioral or dynamic features of a class

## Class Notation

A class notation consists of three parts:

1. Class Name
   * The name of the class appears in the first partition.
2. Class Attributes
   * Attributes are shown in the second partition.
   * The attribute type is shown after the colon.
   * Attributes map onto member variables (data members) in code.
3. Class Operations (Methods)
   * Operations are shown in the third partition. They are services the class provides.
   * The return type of a method is shown after the colon at the end of the method signature.
   * The return type of method parameters is shown after the colon following the parameter name.
   * Operations map onto class methods in code
   * 

The graphical representation of the class - MyClass as shown above:

* MyClass has 3 attributes and 3 operations
* Parameter p3 of op2 is of type int
* op2 returns a float
* op3 returns a pointer (denoted by a \*) to Class6

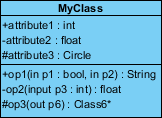
## **Visibility of Class attributes and Operations**

In object-oriented design, there is a notation of visibility for attributes and operations. UML identifies four types of visibility: public, protected, private, and package.

The +, -, # and ~ symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

* + denotes public attributes or operations
* - denotes private attributes or operations
* # denotes protected attributes or operations
* ~ denotes package attributes or operations

### Class Visibility Example



In the example above:

* attribute1 and op1 of MyClassName are public
* attribute3 and op3 are protected.
* attribute2 and op2 are private.

Access for each of these visibility types is shown below for members of different classes.

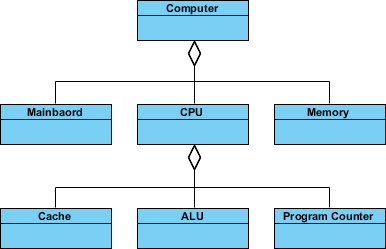
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Right | public (+) | private (-) | protected (#) | Package (~) |
| Members of the same class | yes | yes | yes | yes |
| Members of derived classes | yes | no | yes | yes |
| Members of any other class | yes | no | no | in same package |

## **Class Relationships**.

|  |  |
| --- | --- |
| Relationship Type | Graphical Representation |
| Inheritance (or Generalization):   * Represents an "is-a" relationship. * An abstract class name is shown in italics. * SubClass1 and SubClass2 are specializations of Super Class. * A solid line with a hollow arrowhead that point from the child to the parent class |  |
| Simple Association:   * A structural link between two peer classes. * There is an association between Class1 and Class2 * A solid line connecting two classes |  |
| Aggregation:  A special type of association. It represents a "part of" relationship.   * Class2 is part of Class1. * Many instances (denoted by the \*) of Class2 can be associated with Class1. * Objects of Class1 and Class2 have separate lifetimes. * A solid line with an unfilled diamond at the association end connected to the class of composite |  |
| Composition:  A special type of aggregation where parts are destroyed when the whole is destroyed.   * Objects of Class2 live and die with Class1. * Class2 cannot stand by itself. * A solid line with a filled diamond at the association connected to the class of composite |  |
| Dependency:   * Exists between two classes if the changes to the definition of one may cause changes to the other (but not the other way around). * Class1 depends on Class2 * A dashed line with an open arrow |  |

## **Class Digram Aggregation Example - Computer and parts**

* An aggregation is a special case of association denoting a "consists-of" hierarchy
* The aggregate is the parent class, the components are the children classes



**Conclusion:-** In this Assignment i learn what is Class Diagram and how to implemen**t it.**