# Assignment no 5

**Aim**:
Design and implementation design model from analysis model.

**Problem Statement**:
- Prepare a Design Model from Analysis Model
- Study in detail working on systems/projects.
- Identify Design classes/ Evolve Analysis Model. Use advanced relationships. Draw Design class Model using OCL and UML2.0 Notations. Implement the design model with a suitable object-oriented language.

**Objective**:
- To Identify Design level Classes.
- To Draw Design level class Model using analysis model.
- To Implement Design Model-class diagrams.

**Theory**:
The analysis model is refined and formalized to get a design model.
Design modeling, means refinement to analysis level models to adapt to the actual implementation environment. In the design space, yet another new dimension has been added to the analysis space to include the implementation environment. This means that we want to adopt our analysis model to fit in the implementation model at the same time as we refine it.

**Creating Design level Class Diagrams**
Class diagrams model the static structure of a package or of a complete system. As the blueprints of the system, class diagrams model the objects that make up the system, allowing to display the relationships among those objects and to describe what the objects can do and the services they can provide.

**Class diagrams**
In UML, class diagrams are one of six types of structural diagram. Class diagrams are fundamental to the object modeling process and model the static structure of a system. Depending on the complexity of a system, you can use a single class

diagram to model an entire system, or you can use several class diagrams to model the components of a system.
Class diagrams are the blueprints of your system or subsystem. You can use class diagrams to model the objects that make up the system, to display the relationships between the objects, and to describe what those objects do and the services that they provide.

**Abstraction relationships**
An abstraction relationship is a dependency between model elements that represents the same concept at different levels of abstraction or from different viewpoints.
In an abstraction relationship, one model element, the client, is more refined or detailed than the other, the supplier. The different types of abstraction relationships include derivation, realization, refinement, and trace relationships.

**Aggregation relationships**
In UML models, an aggregation relationship shows a class as a part of or subordinate to another class.
An aggregation is a special type of association in which objects are assembled or configured together to create a more complex object. Data flows from the whole classifier, or aggregate, to the part. A part classifier can belong to more than one aggregate classifier and it can exist independently of the aggregate.
For example, a Department class can have an aggregation relationship with a Company class, which indicates that the department is part of the company. Aggregations are closely related to compositions.
You can name an association to describe the nature of the relationship between two classifiers; however, names are unnecessary if you use association end names.
As the following figure illustrates, an aggregation association appears as a solid line with an unfilled diamond at the association end, which is connected to the classifier that represents the aggregate. Aggregation relationships do not have to be unidirectional.

**Association relationships**
In UML models, an association is a relationship between two classifiers, such as classes, that describes the reasons for the relationship and the rules that govern the relationship.

An association represents a structural relationship that connects two classifiers.
Like attributes, associations record the properties of classifiers.
For example, in relationships between classes, you can use associations to show the design decisions that you made about classes in your application that contain data, and to show which of those classes need to share data. You can use an association's navigability feature to show how an object of one class gains access to an object of another class or, in a reflexive association, to an object of the same class.
The name of an association describes the nature of the relationship between two classifiers and should be a verb or phrase.
In the diagram editor, an association appears as a solid line between two classifiers.

**Association classes**
In UML diagrams, an association class is a class that is part of an association relationship between two other classes.
You can attach an association class to an association relationship to provide additional information about the relationship. An association class is identical to other classes and can contain operations, attributes, as well as other associations.

**Composition Association Relationships**
A composition association relationship represents a whole–part relationship and is a form of aggregation. A composition association relationship specifies that the lifetime of the part classifier is dependent on the lifetime of the whole classifier.
In a composition association relationship, data usually flows in only one direction (that is, from the whole classifier to the part classifier). For example, a composition association relationship connects a Student class with a Schedule class, which means that if you remove the student, the schedule is also removed.

**Generalization Relationships**
In UML modeling, a generalization relationship is a relationship in which one model element (the child) is based on another model element (the parent).
You can add generalization relationships to capture attributes, operations, and relationships in a parent model element and then reuse them in one or more child model elements. Because the child model elements in generalizations inherit the attributes, operations, and relationships of the parent, you must only define for the child the attributes, operations, or relationships that are distinct from the parent.

The parent model element can have one or more children, and any child model element can have one or more parents. It is more common to have a single parent model element and multiple child model elements. Generalization relationships do not have names.

**Interface Realization Relationships**
In UML diagrams, an interface realization relationship is a specialized type of implementation relationship between a classifier and a provided interface. The interface realization relationship specifies that the realizing classifier must conform to the contract that the provided interface specifies.
Typically, interface realization relationships do not have names. If you name an interface realization, the name is displayed beside the connector in the diagram.

**Realization Relationships**
In UML modeling, a realization relationship is a relationship between two model elements, in which one model element (the client) realizes the behavior that the other model element (the supplier) specifies. Several clients can realize the behavior of a single supplier.

**Attributes in Class**
In UML models, attributes represent the information, data, or properties that belong to instances of a classifier.
A classifier can have any number of attributes or none at all. Attributes describe a value or a range of values that instances of the classifier can hold. You can specify an attribute's type, such as an integer or Boolean, and its initial value. You can also attach a constraint to an attribute to define the range of values it holds.
Attribute names are short nouns or noun phrases that describe the attribute. The UML syntax for an attribute name incorporates information in addition to its name

**Operations in Class**
In UML models, operations represent the services or actions that instances of a classifier might be requested to perform.
A classifier can have any number of operations or none at all. Operations define the behavior of an instance of a classifier

You can add operations to identify the behavior of many types of classifiers in your model. In classes, operations are implementations of functions that an object might be required to perform. Well-defined operations perform a single task.
Operations can have exceptions, elements that are created when the operation encounters an error.
Every operation in a classifier must have a unique signature. A signature comprises the operation's name and its ordered list of parameter types. The UML syntax for an operation name is as follows:

Visibility «stereotype» name(parameter list) : return-type

**<u>Conclusion</u>:**
We designed and implemented the design model from the analysis model.