**INTRODUCTION TO UML**

**UML DEFINITION:-**

The Unified Modeling Language (UML) is a standard language for writing software blueprints. The UML uses mostly graphical notations to express the design of software projects.

The UML is a language for

* Visualizing
* Specifying
* Constructing
* Documenting

the artifacts of a software intensive system.

Uml is a very expressive language, addressing all the views needed to develop and the deploying such system. It is not difficult to understand and to use.

Learning to apply UML starts with forming a conceptual model of the language which requires learning UML’s basic building blocks, the rules that dictate how these building blocks may be together and some common mechanisms that apply throughout the language.

**Application of UML:-**

The UML is intended primarily for software intensive system. It has been used effectively for such domains as

* + Enterprise information system.
  + Banking and financial services.
  + Telecommunication.
  + Defense/aerospace.
  + Retail
  + Medical electronics.
  + Scientific.
  + Distributed web based services.

The UML is not limited to modeling software. It is expressive enough to model non software system, such as workflow in the legal system, the structure and behaviors of a patient health care system, and the design of hardware.

# Goals of UML:

The primary goals in the design of the UML were:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

**Definition of Modeling:-**

Modeling is overall design of the system.

* Modeling captures essential parts of the system.
* Modeling is a central part of all the activities that lead up to the deployment of good software.
* We build models to communication the desired structure and behavior of our system.
* We build models to visualize and controls the system’s architecture
* We build models to better understanding the system we are building, after exposing opportunities for simplification and reuse.
* We build models to manage risk.

**Principles of Modeling:-**

The use of modeling has a rich history in all the engineering disciplines. That experience suggests four basic principles of Modeling.

1. The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.
2. Every model may be expressed at different levels of precision.
3. The best models are connected to reality.
4. No single is sufficient every nontrivial system is best approach through a small set of nearly independent models.

**A conceptual model of the UML:--**

To understand UML, you need to form a conceptual model of the languages, and this request learning there major elements, the uml’s basic building blocks, the rules that dictate how those building blocks may be put together, and some common mechanisms that apply throughout the UML.

**Building blocks of UML:**

The vocabulary of the uml encompasses three kinds of building blocks.

1. Things

2. Relationships

3. Diagrams.

Things are the abstractions that are first class citizens in a model; relationships these things together; diagrams group interesting collections of things.

**Things in uml**:

There are four kinds of things in Uml. They are

1. Structural things

2. Behavioral things

3. Grouping things

4. Annotational things.

**Structural things:**

These are the nouns of uml models. These are mostly the static parts of model. In all these are seven kinds of structural things. They are

1. class 5. Active class
2. Interface 6. Component
3. collaboration 7. Node
4. usecase

**1. Class:-**

A class is description of a set of objects that share the same attributes, operations, relationship, and semantics.

A class implements one or more interface classes are the most important building blocks of any objects-oriented system. We can use classes to represent software thing, hard ware thing, and even things that are purely conceptual.

Graphically, a class is rendered as a rectangle, usually including

ClassName

Attributes

Operations

Every class must have distinguished it from other classes. A name is a textual string name is a simple name or a path name path name is the class name prefixed by name of the package in which that class lies.

**Attributes:-**

An attributes is a named property of a class that describes a range of value that instances of the property may hold, A class may have any number of attributes or no attributes at all An attributes represents some properly of the thing we are modeling that is shared by all objects of that class. Graphically, attributes are listed in a compartment just below the class name.

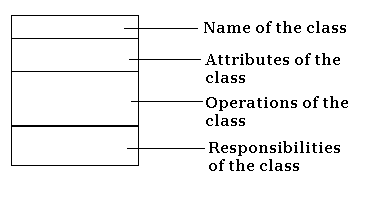
**Operations:-**

An operation is the implementation of a service that can be requested from any object of the class to affect behavior. In other words, an operation is an abstraction of something you can do to an object and that is shared by all objects of that class. A class may have any number of operation or no operation at all graphically operations are listed in a compartment just the class attributes.

**Responsibilities:-**

A responsibility is a contract or an obligation of a class when create a class, we are making a statement that all objects of behavior. At a more abstract level, these corresponding attributes and operations are just the features by which the class’s responsibilities are carried out.

A class may have any number of responsibility. As we refine our models, we will translate these responsibilities into a set of attributes and operations that best fulfill the class’s responsibilities. Graphically, responsibilities can be drawn in a separate compartment at the bottom of the class icon.



**2. Interface :-**

An interface is a collection of operation that specify a service of a class or component. An interface therefore describes the externally visible behavior of a class or component or only a part of that behavior. An interface is rendered as a circle together with its name.

Is spelling

**3. Collaboration:-**

To provide some co-operative It is an interaction and is a society of roles and other elements that work together behavior that’s bigger than the sum of all its elements. Graphically it is represented as ellipse with dashed lines usually including only its name .

**4. Usecase:**

It is a description of set of sequence of actions that a system performs yields on observable result of value to a particular actor.Graphically it represented as an ellipse with solid lines usually including only its names.

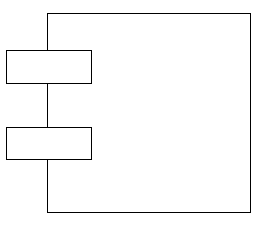
**5. Active class:**

It is a class whose objects own one or more processes or threads and therefore can initiate control actively.It is graphically represented as just like a class but with heavy lines, usually including its name, attributes and operations.

|  |
| --- |
| EVENT MANAGER |
|  |
| Suspend()  Fluh() |

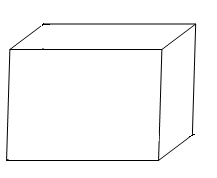
6. **Component:-**

A component is a physical and replaceable part of a system that conforms that conforms to and provides the realization of a set of interface. A component typically represents the physical package of otherwise logical elements such as class interfaces, and collaborations. Graphically, a component is rendered as a rectangle with tabs, usually including only its name.



7. **Node:-**

A node is a physical element that exits at runtime and represents a computational resource, generally having at least some memory and, often, processing capability. A set of components may reside on a node and may also migrate from node to node. Graphically , node is rendered as a cube, usually including only its name.



**Behavioral things:**

These are the dynamic parts of uml models. These are the verbs of a model, representing behavior over time and space, these are two types.

1.Interaction

2.State machine

**1.** **Interaction:**

. It is a behavior that compare a set of messages exchanged among a set of objects with in a particular context to accomplish a specific purpose graphically represented as a directed line, almost always including the name of its operation

Display

**2. State machine:**

It is a behavior that specifies the sequence of states an object or an interaction goes through during its lifetime in response to events, together with its responses to these events. A state machine involves a number of other elements, including states, transitions events and activities graphically it is represented as a rounded

Rectangle , including its name and its subsets.

Waiting

**Grouping things:**

These are organizational parts of uml models. These are the boxes into which a model can be decomposed. In all, there is one primary kinds of groping thing namely packages.

**package:**

It is a general purpose mechanism for organizing elements into groups structural, behavioral and even other grouping things may be placed in a package it is purely conceptual. Graphically represented as a tabbed folder including only its name and sometimes, its contents.

Business rules

**Annotational things:**

Those are the explanatory parts of uml models. These are the comments you may apply to describe illuminate and remark about any element in a model. There is one primary kinds of Annotational thing called a note.

**Note:**

It is a simply symbol for rendering or a collection of constructs and comments attached to an element or a collection of elements, graphically it is represented as a rectangle with a dog eared corner together with a or a graphical comment.

Return copy of self

**Relationships in UML :**

A relationship is a connection among things. In object oriented modeling, there are three kinds of relationships that are especially important.

* **Dependencies** which represent using relationships among classes
* **Generalization** which link generalized classes to their specialization
* **Associations**  which represent structural relations among object.
* **Realization**  which represent relation among class and interface or

Usecase and collaboration

Each of these relationships provides a different way of combining the abstraction. Graphically, a relationship is rendered as a path, with different kinds of lines used to distinguish the kinds of relationships.

**1. Dependency:-**

A dependency is a using relationship that states that a change in specification of one thing may affect another thing that uses it, but not necessarily the reverse. Graphically, a dependency is rendered as a dashed directed line, directed to the thing being depended on. We use dependencies when we want to show out thing using another thing.

**2**. **Generalization:-**

A generalization is a relationship between a general thing and a more specific kind of that thing. It is something called an “is-a-kind-of” relationship. Generalization means that objects of the child may be used anywhere the parent may appear, but not the reverse i.e., the child is substitutable for the parent. A child inherits the properties of its parents, especially their attributes and operations. An operation of a child that has the same signature as an operation in a parent. Graphically, generalization when we want to show parent/child relationships.

A class may have zero, one or more parents. A class that has next parents and one or more children is called a root class or a base class. A class that has no children is called a leaf class. A class that has exactly one parent is said to use single inheritance.

**3. Association:-**

An association is a structured relationship that specifies that object of one thing are connected to objected of another. Given an association connecting two class, we can navigate from an object of one class to an object of the other class, and vice versa. It’s quite legal to have both ends of an association circles back to the same class. This means that, given an object of the class, we can link to other object of the same class. An association that connect exactly two classes is called a binary association. Graphically an association is rendered as a solid line connecting the same or different classes. Use association when we want to shoe structural relationship.

Beyond this basic form, There are four adornments that apply to associations.

**Name:-**

An association can have a name, and we use that name to describe the nature of the relationship. So that is no ambiguity about its meaning, we can give a direction to the name by providing a direction triangle.

**Role:-**

When a class participates in an association it has a specific role that it plays in that relationship, It has a specific role that it plays in that relationship. Role is just the face the class at the near end of the association presents to the class at the other end of the association. We can explicitly name the role of a class plays in an association.

**Multiplicity:-**

An association represents a structural relationships among objects. In many modeling situation, it’s important for us to state how many objects may be connected across an instance of an association. This “how many”, is called the multiplicity of an associations role, and is written as an expression that evaluates to a range of values or an explicit value.

When we state a multiplicity at one end of an association, we are specifying that, for each object of the class at the opposite end, there must be that many object at the near end.

**Aggregation:-**

A plain association between two classes represents a structural relationship between peers, meaning that both classes are conceptually at the same level, no one more important than the other. Sometimes, we will want to model a “whole/part” relationships in which one class represents a larger thing, which consists of smaller things. This kind of relationship is called aggregation, which represents a “has-a” Relationship, meaning that an object of the whole has object of the part.

**Composition:-**

Composition specifies a strong form of aggregation in which every part also get deleted when we want to delete whole thing. Graphically it is represented with solid diamond with a line.

**4. Realization:**

It is a semantic relationship between classifier where in one classifies a contract that another classifier guarantees to carry out this relationship encounters that realize them graphically it is represented as a cross between a generalization and a dependency relationship.

**Diagrams in uml:**

Diagram is the graphical presentation of a set of elements most represented as a connected graph of vertices (things)&Arcs (relationships). Draw diagrams to visualize a system from different perspectives. The uml includes nine such diagrams they are:

Structural Diagrams are:

1. Class diagrams

2. Object diagrams

3. Usecase diagrams

4. Sequence diagrams

Behavioral Diagrams are

1. Collaboration diagrams.

2. State chart diagrams.

3. Activity diagrams

4. Component diagrams

5. Deployment diagrams.

**Class diagram:**

It shows set of classes’ interfaces and collaborations & their relationships. These address the static design view of a system these that include active classes address the static process view of a system. Class diagrams models class structure and contents using design elements such as classes, packages and objects. It also displays relationships such as containment, inheritance, associations and others.

**Object diagram:**

It shows a set objects and their relationships. These represents static snapshots of instances of the things found in class diagram.

**Use case diagram:**

It shows a set of use cases and actors and their relationships use case diagrams address the static (view)use case view of a system importance in organizing and Modeling the behavior of a system.

**Sequence diagram:**

Sequence diagram emphasizes the time ordering of messages. It mainly shows set of objects and the messages send /receive by those objects which is concerned with time ordering of messages. It displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension (time) and horizontal dimension (different objects).

**Collaboration diagram:**

It is an interaction diagram. Focuses on set of objects and messages, which is related to the structural organization of objects that send and receive messages. It displays an interaction organized around the objects and their links to one another. Numbers are used to show the sequence of messages.

**State chart diagram:**

It shows a state machine, consisting of states, transitions, events and activities. This is dynamic view of system and emphasize the event ordered behavior of an object, which is especially useful in modeling reactive system. It displays the sequences of states that an object of an interaction goes through during its life in response to received stimuli, together with its responses and actions.

**Activity diagram:**

It is a special kind of state chart diagram that shows the flow from activity to activity within a system. it address the dynamic view of a system. Emphasize flow of control among objects. It displays a special state diagram where most of the states are action states and most of the transitions are triggered by completion of the actions in the source states. This diagram focuses on flows driven by internal processing.

**Component diagram:**

It shows the organizations and dependencies among set of compo nets. The static(view) implementation view of a system. It displays the high level packaged structure of the code itself. Dependencies among components are shown, including source code components, binary code components, and executable components. Some components exist at compile time, at link time, at run times well as at more than one time.

**Deployment diagram:**

It shows the configuration of runtime processing nodes and the components that live on them it address the static deployment view of architecture. Itdisplays the configuration of run-time processing elements and the software components, processes, and objects that live on them. Software component instances represent run-time manifestations of code units.

***The common mechanisms in the UML***

Usually in the human languages the vocabulary extends itself as time passes, therefore dictionaries are printed every year. Similarly in UML, when a system is designed, not all connects and terms can be expressed through the language. Therefore there are mechanisms provided by the UML to cover the details in the form of comments and constraints, to better understand the system. notes capture the comments and constraints, stereotypes, tagged values and constraints are mechanisms which add new building blocks, new properties and semantics.

There are four common mechanisms that are used in UML.

**Specification:** These provide a semantic backplane that contains all the parts of all modes of a system.

**Adornments:** These are the additional details of a class whether it is an abstr4act class, or to show its visibility of the attributes and operations.

**Command divisions**: these are the divisions between class and its objects and between interfaces and implementation.

**Extensibility mechanism**: The UML can be extended in controlled ways

The extensibility mechanisms are:

* **Stereotypes**
* **Tagged values**
* **Constraints**

**Stereotypes**:

The stereotypes allow the creation of new types of building blocks similar to the existing blocks but may be specific to the current problem. Stereotyped nodes can be used to make thing appear as primitive’s building blocks, but with their own special properties, semantics and notations or icon.

**Tagged Values**:

The tagged value extends the properties of a UML building block, allowing you to create new information in that element’s. Specification. For example, if you are working on a shrink-wrapped product that undergoes many releases over time, you often want to track the version and author of certain critical abstractions. Version and author are not primitive UML concepts. They can be added to any building block, such as class, by introducing new tagged values to that building block.

**Constraints:**

A constraint extends the semantics of a UML building block, allowing you to add new rules or modify existing ones. For example, you might want to constrain the Events Quest class so that all additions are done in order.

Generally, these three extensibility mechanism allow you to shape and grow the UML to your project’s needs. These mechanisms also let the UML adapt to new software technology, such as the likely emergence of more powerful distributed existing ones, and even change their semantics.

Naturally tits important that you do so in controlled ways so that through these extensions, you remain true to the UML’