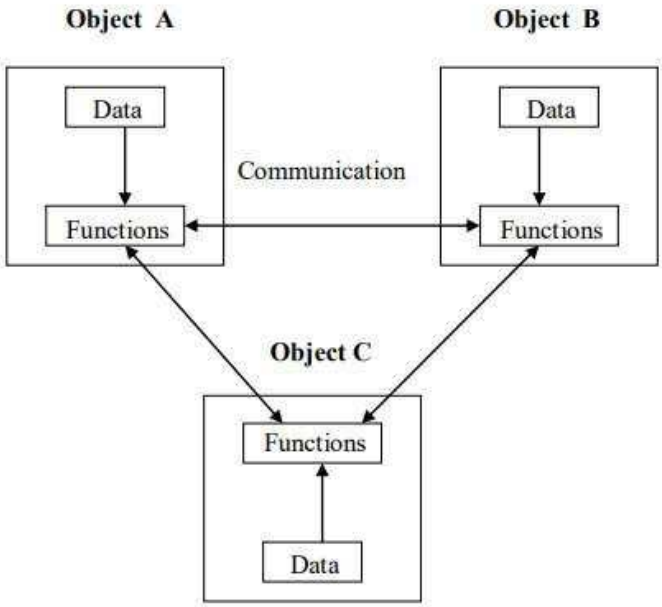
FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

Object Oriented Programming

"Object oriented programming as an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand".

Features of the Object-Oriented programming

1. Emphasis is on doing rather than procedure.
2. programs are divided into what are known as objects.
3. Data structures are designed such that they characterize the objects.
4. Functions that operate on the data of an object are tied together in the data
5. structure.
6. Data is hidden and can't be accessed by external functions.
7. Objects may communicate with each other through functions.
8. New data and functions can be easily added.
9. Follows bottom-up approach in program design.



What Is an Object?

Objects are key to understanding object-oriented (in the glossary) technology. You can look around you now and see many examples of real-world objects: your dog, your desk, your television set, your bicycle.

These real-world objects share two **characteristics:** They all have state and behaviour. For example, dogs have state (name, colour, breed, hungry) and

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

behaviour (barking, fetching, and wagging tail). Bicycles have state (current gear, current pedal cadence, two wheels, number of gears) and behaviour (braking, accelerating, slowing down, changing gears).

Definition: An object is a software bundle of variables and related methods.

Software objects are modelled after real-world objects in that they too have state and behaviour. A software object maintains its state in one or more variables (in the glossary). A variable is an item of data named by an identifier. A software object implements its behaviour with methods (in the glossary). A method is a function (subroutine) associated with an object.

CLASS:

Class is a blueprint of any object and A group of objects that share common properties for data part and some program parts are collectively called as class.

In C ++ a class is a new data type that contains member variables and member functions that operate on the variables.

Object Identity

1. **Object identity:** An object retains its identity even if some or all of the values of variables or definitions of methods change over time.

   This concept of object identity is necessary in applications but doe not apply to tuples of a relational database.

2. Object identity is a stronger notion of identity than typically found in programming languages or in data models not based on object orientation.
3. Several forms of identity:

   - **value**: A data value is used for identity (e.g., the primary key of a tuple in a relational database).
   - **name**: A user-supplied name is used for identity (e.g., file name in a file system).
   - **built-in**: A notion of identity is built-into the data model or programming languages, and no user-supplied identifier is required (e.g., in OO systems).

4. Object identity is typically implemented via a *unique*, *system-generated* OID. The value of the OID is not visible to the external user, but is used internally by the system to identify each object uniquely and to create and manage inter-object references.

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

5. There are many situations where having the system generate identifiers automatically is a benefit, since it frees humans from performing that task. However, this ability should be used with care. System-generated identifiers are usually specific to the system, and have to be translated if data are moved to a different database system. System-generated identifiers may be redundant if the entities being modelled already have unique identifiers external to the system, e.g., SIN#.

Basic Principle of Object-Oriented programming

1. **Data Abstraction (Information Hiding):** Abstraction refers to the act of representing essential features without including the back ground details or explanations. Classes use the concept of abstraction and are defined as size, width and cost and functions to operate on the attributes.

2. **Data Encapsulation:** The wrapping up of data and function into a single unit (called class) is known as encapsulation. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it. These functions provide the interface between the objects data and the program.

3. **Polymorphism:** Polymorphism means the ability to take more than one form. An operation may exhibit different instance. The behaviour depends upon the type of data used in the operation. A language feature that allows a function or operator to be given more than one definition. The types of the arguments with which the function or operator is called determines which definition will be used. Overloading may be operator overloading or function overloading

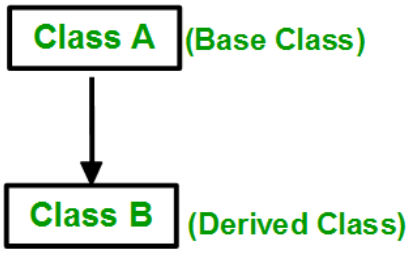
4. **Inheritance:** Inheritance is the process by which objects of one class acquire the properties of another class. In the concept of inheritance provides the idea of reusability. This mean that we can add additional features to an existing class without modifying it. This is possible by designing a new class will have the combined features of both the classes.

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

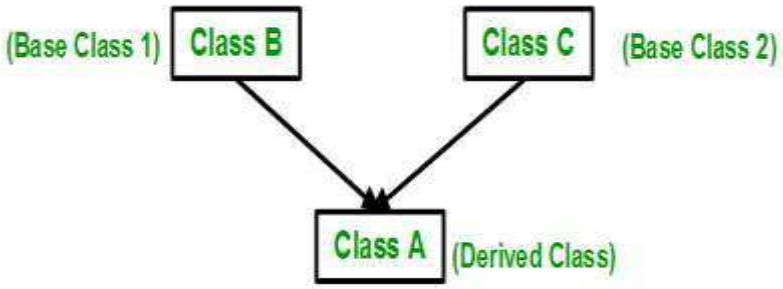
FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

Types of Inheritance in C++

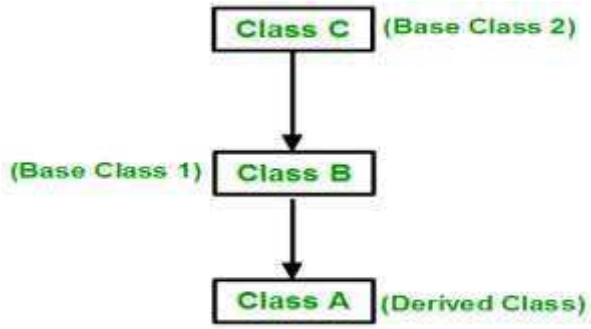
1. Single Inheritance: In single inheritance, a class is allowed to inherit from only one class. i.e., one sub class is inherited by one base class only.



2. Multiple Inheritance: Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes. i.e one **sub class** is inherited from more than one **base classes**.

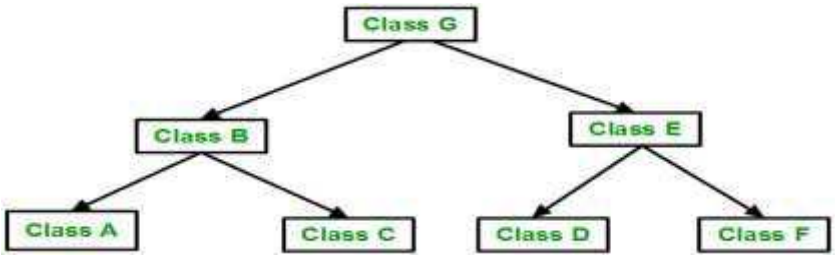


3. **Multilevel Inheritance**: In this type of inheritance, a derived class is created from another derived class.

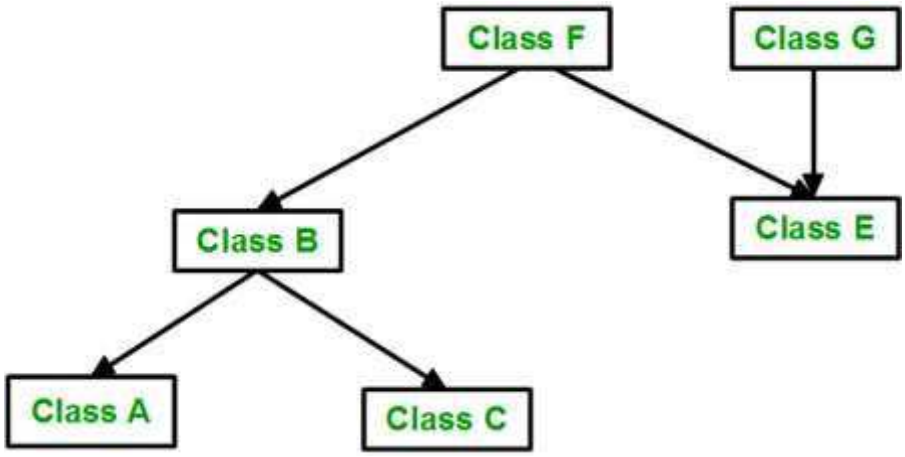


4. Hierarchical Inheritance: In this type of inheritance, more than one sub class is inherited from a single base class. i.e. more than one derived class is created from a single base class.

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM



5. Hybrid (Virtual) Inheritance: Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.
Below image shows the combination of hierarchical and multiple inheritance:



What is UML

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

The UML gives you a standard way to write a system's blueprints, covering conceptual things, such as business processes and system

functions, as well as concrete things, such as classes written in a specific programming language, database schemas, and reusable software components

UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

- UML stands for **Unified Modeling Language**.
- UML is different from the other common programming languages such as C++, Java, COBOL, etc.
- UML is a pictorial language used to make software blueprints.

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

- UML can be described as a general purpose visual modeling language to visualize, specify, construct, and document software system.
- Although UML is generally used to model software systems, it is not limited within this boundary. It is also used to model non-software systems as well. For example, the process flow in a manufacturing unit, etc.

UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object-oriented analysis and design. After some standardization, UML has become an OMG standard.

What is Model

A model is a simplification of reality. A model provides the blueprints of a system. A model may be structural, emphasizing the organization of the system, or it may be behavioural, emphasizing the dynamics of the system.

Why do we model

We build models so that we can better understand the system we are developing.

Through modelling, we achieve four aims.

1. Models help us to visualize a system as it is or as we want it to be.
2. Models permit us to specify the structure or behavior of a system.
3. Models give us a template that guides us in constructing a system.
4. Models document the decisions we have made.

Principles of Modelling

There are four basic principles of model

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 model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models.

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

Object Oriented Modelling

In software, there are several ways to approach a model. The two most common ways are

1. Algorithmic perspective
2. Object-oriented perspective

Algorithmic Perspective

The traditional view of software development takes an algorithmic perspective. In this approach, the main building block of all software is the procedure or function. This view leads developers to focus on issues of control and the decomposition of larger algorithms into smaller ones. As requirements change and the system grows, systems built with an algorithmic focus turn out to be very hard to maintain.

Object-oriented perspective

The contemporary view of software development takes an object-oriented perspective. In this approach, the main building block of all software systems is the object or class. A class is a description of a set of common objects. Every object has identity, state, and behaviour.

An Overview of UML

- The Unified Modelling Language is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system.
- The UML is appropriate for modelling systems ranging from enterprise information systems to distributed Web-based applications and even to hard real time embedded systems. It is a very expressive language, addressing all the views needed to develop and then deploy such systems.

The UML is a language for

- Visualizing
- Specifying
- Constructing
- Documenting

Visualizing The UML is more than just a bunch of graphical symbols. Rather, behind each symbol in the UML notation is a well-defined semantics. In this

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

manner, one developer can write a model in the UML, and another developer, or even another tool, can interpret that model unambiguously

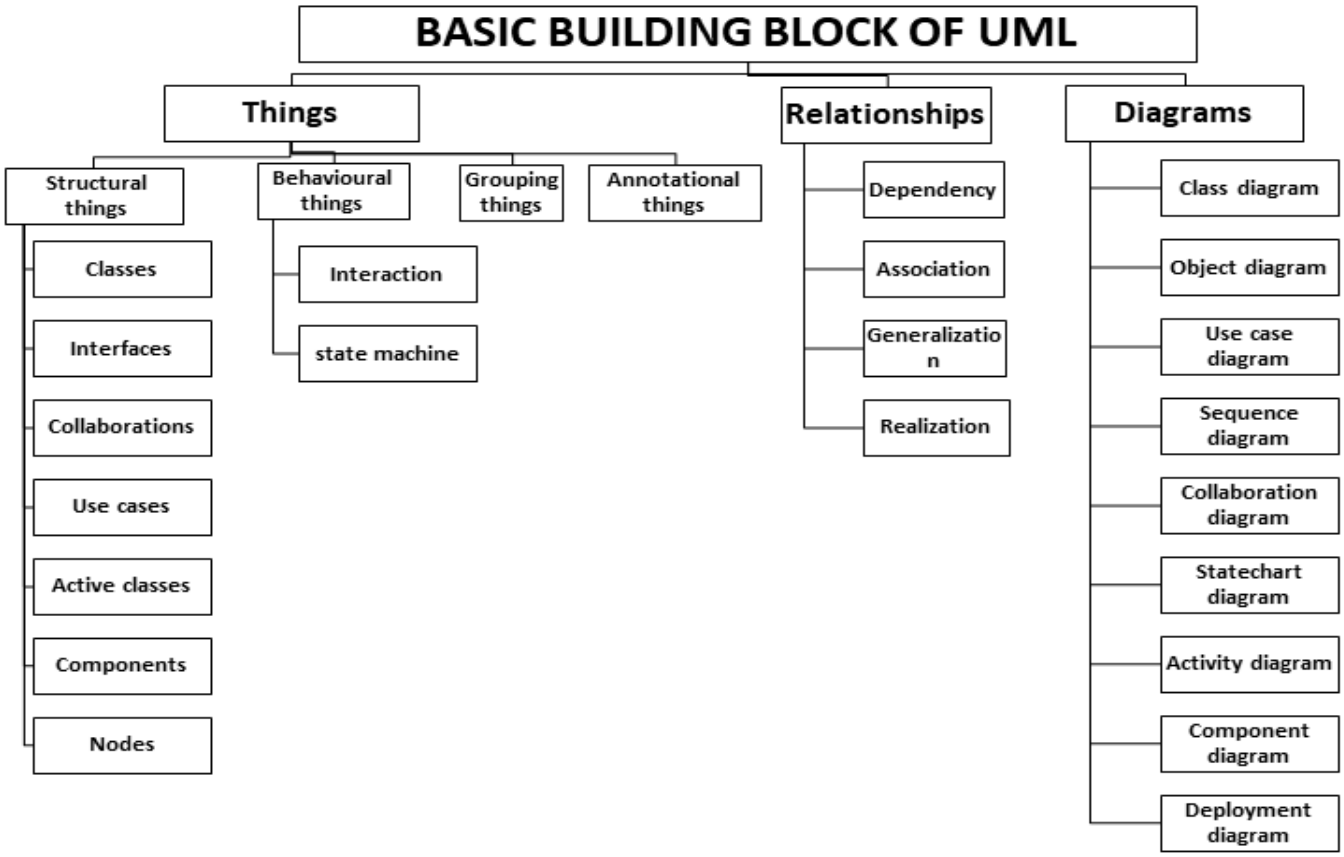
Specifying means building models that are precise, unambiguous, and complete. Constructing the UML is not a visual programming language, but its models can be directly connected to a variety of programming languages

Documenting a healthy software organization produces all sorts of artifacts in addition to raw executable code. These artifacts include

1.Requirements	2.Architecture	3.Design
4.Source code	5.Project plans	6.Tests
7.Prototypes	8.Releases	

Conceptual Model of UML

To understand the UML, you need to form a conceptual model of the language, and this requires learning three 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.



FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

Basic Building Block of UML

1. Things
2. Relationships
3. Diagrams

Things in the UML

There are four kinds of things in the UML:

1. Structural things
2. Behavioural things
3. Grouping things
4. Annotational things

Structural things are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical. In all, there are seven

kinds of structural things.

1. Classes
2. Interfaces
3. Collaborations
4. Use cases
5. Active classes
6. Components
7. Nodes

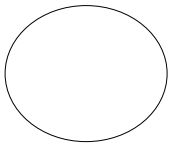
Class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class implements one or more interfaces. Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations.

Interface is a collection of operations that specify a service of a class or component. An interface therefore describes the externally visible behaviour of that element. An interface might represent the complete behaviour of a class or component or only a part of that behaviour. An interface is rendered as a circle

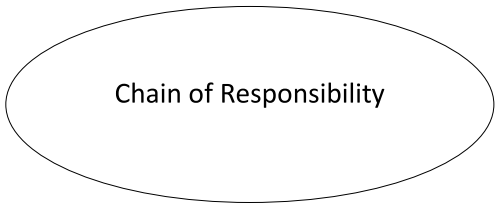
FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

together with its name. An interface rarely stands alone. Rather, it is typically attached to the class or component that realizes the interface

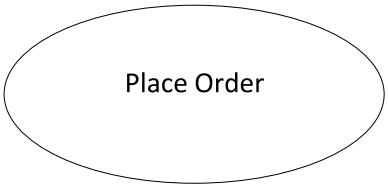


Collaboration defines an interaction and is a society of roles and other elements that work together to provide some cooperative behaviour that's bigger than the sum of all the elements. Therefore, collaborations have structural, as well as behavioural, dimensions. A given class might participate in several collaborations. Graphically, a collaboration is rendered as an ellipse with dashed lines, usually including only its name

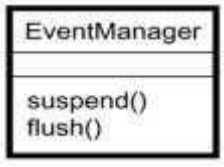


Use case

- Use case is a description of set of sequence of actions that a system performs that yields an observable result of value to a particular actor
- Use case is used to structure the behavioural things in a model.
- A use case is realized by a collaboration. Graphically, a use case is rendered as an ellipse with solid lines, usually including only its name

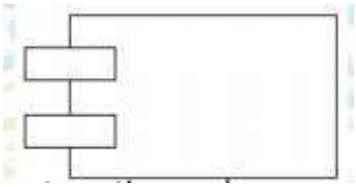


Active class is just like a class except that its objects represent elements whose behaviour is concurrent with other elements. Graphically, an active class is rendered just like a class, but with heavy lines, usually including its name, attributes, and operations

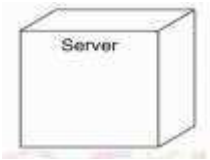


Component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. Graphically, a component is rendered as a rectangle with tabs

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM



Node is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capability. Graphically, a node is rendered as a cube, usually including only its name



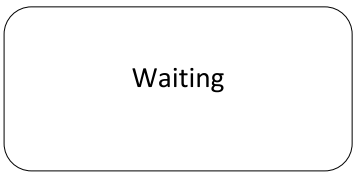
Behavioural Things are the dynamic parts of UML models. These are the verbs of a model, representing behaviour over time and space. In all, there are two primary kinds of behavioural things

1. Interaction
2. state machine

Interaction Interaction is a behaviour that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. An interaction involves a number of other elements, including messages, action sequences and links. Graphically a message is rendered as a directed line, almost always including the name of its operation

Display

State Machine State machine is a behaviour that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events. State machine involves a number of other elements, including states, transitions, events and activities. Graphically, a state is rendered as a rounded rectangle, usually including its name and its substates



FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

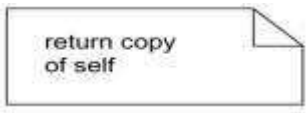
Grouping Things: -

1. are the organizational parts of UML models. These are the boxes into which a model can be decomposed
2. There is one primary kind of grouping thing, namely, packages.

Package: -

- A package is a general-purpose mechanism for organizing elements into groups. Structural things, behavioural things, and even other grouping things may be placed in a package
- Graphically, a package is rendered as a tabbed folder, usually including only its name and, sometimes, its contents

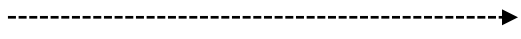
An notational things are the explanatory parts of UML models. These are the comments you may apply to describe about any element in a model. A note is simply a symbol for rendering constraints and comments attached to an element or a collection of elements. Graphically, a note is rendered as a rectangle with a dog-eared corner, together with a textual or graphical comment



Relationships in the UML: There are four kinds of relationships in the UML:

1. Dependency
2. Association
3. Generalization
4. Realization

Dependency: - Dependency is a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing. Graphically a dependency is rendered as a dashed line, possibly directed, and occasionally including a label



Association is a structural relationship that describes a set of links, a link being a connection among objects. Graphically an association is rendered as a solid

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

line, possibly directed, occasionally including a label, and often containing other adornments, such as multiplicity and role names

---

Aggregation is a special kind of association, representing a structural relationship between a whole and its parts. Graphically, a generalization relationship is rendered as a solid line with a hollow arrowhead pointing to the parent

Realization is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out. Graphically a realization relationship is rendered as a cross between a generalization and a dependency relationship

Diagrams in the UML

Diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships). In theory, a diagram may contain any combination of things and relationships.

For this reason, the UML includes nine such diagrams:

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram
6. Statechart diagram
7. Activity diagram
8. Component diagram
9. Deployment diagram

Class diagram A class diagram shows a set of classes, interfaces, and collaborations and their relationships. Class diagrams that include active classes address the static process view of a system.

Object diagram Object diagrams represent static snapshots of instances of the things found in class diagrams. These diagrams address the static design view or

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

static process view of a system. An object diagram shows a set of objects and their relationships

Use case diagram A use case diagram shows a set of use cases and actors and their relationships. Use case diagrams address the static use case view of a system. These diagrams are especially important in organizing and modelling the behaviours of a system.

Interaction Diagrams Both sequence diagrams and collaboration diagrams are kinds of interaction diagrams. Interaction diagrams address the dynamic view of a system

- **A sequence diagram** is an interaction diagram that emphasizes the time-ordering of messages
- **A collaboration** diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages

Sequence diagrams and collaboration diagrams are isomorphic, meaning that you can take one and transform it into the other

Statechart diagram A state chart diagram shows a state machine, consisting of states, transitions, events, and activities. Statechart diagrams address the dynamic view of a system. They are especially important in modelling the behaviour of an interface, class, or collaboration and emphasize the event-ordered behaviour of an object

Activity diagram an activity diagram is a special kind of a state chart diagram that shows the flow from activity to activity within a system. Activity diagrams address the dynamic view of a system. They are especially important in modelling the function of a system and emphasize the flow of control among objects

Component diagram A component diagram shows the organizations and dependencies among a set of components. Component diagrams address the static implementation view of a system. They are related to class diagrams in that a component typically maps to one or more classes, interfaces, or collaborations

Deployment diagram A deployment diagram shows the configuration of run-time processing nodes and the components that live on them. Deployment diagrams address the static deployment view of an architecture

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

Rules of the UML

The UML has semantic rules for

1. **Names** What you can call things, relationships, and diagrams
2. **Scope** The context that gives specific meaning to a name
3. **Visibility** How those names can be seen and used by others
4. **Integrity** How things properly and consistently relate to one another
5. **Execution** What it means to run or simulate a dynamic model

Models built during the development of a software-intensive system tend to evolve and may be viewed by many stakeholders in different ways and at different times. For this reason, it is common for the development team to not only build models that are well-formed, but also to build models that are

1. Elided Certain elements are hidden to simplify the view
2. Incomplete Certain elements may be missing
3. Inconsistent The integrity of the model is not guaranteed

Common Mechanisms in the UML

UML is made simpler by the presence of four common mechanisms that apply consistently throughout the language.

1. Specifications
2. Adornments
3. Common divisions
4. Extensibility mechanisms

Specification that provides a textual statement of the syntax and semantics of that building block. The UML's specifications provide a semantic backplane that contains all the parts of all the models of a system, each part related to one another in a consistent fashion

Adornments Most elements in the UML have a unique and direct graphical notation that provides a visual representation of the most important aspects of the element. A class's specification may include other details, such as whether it is abstract or the visibility of its attributes and operations. Many of these details can be rendered as graphical or textual adornments to the class's basic rectangular notation.

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM

Extensibility Mechanisms

The UML's extensibility mechanisms include

1. Stereotypes
2. Tagged values
3. Constraints

Stereotype

- Stereotype extends the vocabulary of the UML, allowing you to create new kinds of building blocks that are derived from existing ones but that are specific to your problem
- A tagged value extends the properties of a UML building block, allowing you to create new information in that element's specification
- A constraint extends the semantics of a UML building block, allowing you to add new rules or modify existing ones

Architecture

A system's architecture is perhaps the most important artifact that can be used to manage these different viewpoints and so control the iterative and incremental development of a system throughout its life cycle.

Architecture is the set of significant decisions about

1. The organization of a software system
2. The selection of the structural elements and their interfaces by which the system is composed
3. Their behaviour, as specified in the collaborations among those elements
4. The composition of these structural and behavioural elements into progressively larger subsystems

The architectural style that guides this organization: the static and dynamic elements and their interfaces, their collaborations, and their composition. Software architecture is not only concerned with structure and behaviour, but also with usage, functionality, performance, resilience, reuse, comprehensibility, economic and technology constraints and trade-offs, and aesthetic concerns.

Use case view the use case view of a system encompasses the use cases that describe the behaviour of the system as seen by its end users, analysts, and testers. With the UML, the static aspects of this view are captured in use case

FREE DOWNLOAD NOTES, EBOOK, STUDY MATERIAL VISIT:- WWW.TAIYARILIVE.COM