Lecture Notes

On

UML

UNIT-1

OOAD-INTRODUCTION, REVIEW OF OBJECT MODELLING ; NEW PARADIGM, OBJECT ORIENTED THINKING - RETHINKING, OBJECTS AND CLASSES, LINKS AND ASSOCIATION, GENERALIZATION AND SPECIALIZATION, INHERITANCE, GROUPING CONCEPTS, AGGREGATION, ABSTRACTS CLASSES, POLYMORPHISM, METADATA, CONSTRAINTS, DYNAMIC MODELLING EVENTS STATES, OPERATIONS, CONCURRENCY.

OOAD-INTRODUCTION:

"A method to design and build large programs with a long lifetime" e.g.

- 50k loc C++ with O(a) lifetime

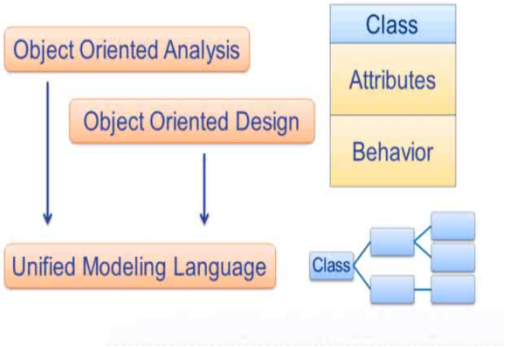
-Blueprints of systems before coding

-Development process

-Maintenance and modifications

-Control of dependencies

-Separation into components



During object-oriented analysis, there is an emphasis on finding and describing the objects—or concepts—in the problem domain. Eg library information system--Book, Library, and Patron.

During object-oriented design, there is an emphasis on defining software objects and how they collaborate to fulfill the requirements.eg Book software object may have a title attribute and a getChap-ter method.

Object Modeling

Object modeling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and operations that characterize each class.

The process of object modeling can be visualized in the following steps:

- Identify objects and group into classes
- Identify the relationships among classes
- Create user object model diagram
- Define user object attributes
- Define the operations that should be performed on the classes
- Review glossary

What is an object?

- An object is a concept, abstraction, or thing with identity that has meaning for application.
- An object is an entity that has well-defined structure and behavior
- It is combination of data and logic that represents some real world entity.
- An object can be tangible, intangible or conceptual .

Characteristics of an Object

- Object = State + Behavior + Identity + Responsibility
- State
- Behavior
- Identity
- Responsibility

State

- The state of the object is one of the possible condition in which an object may exists.
- The state of an object encompasses the current values of all its attributes.
- The state of the object normally changes over time.
- An attribute can be static or dynamic
- Attributes of a car Constant: Color, Average, Make, Power, Fuel Type
  Dynamic: Speed ,Fuel level, Tyre pressure

Behavior

- Behavior groups all the abilities of an object & describes the actions or reactions of the object.
- Behavior is how an object acts or reacts, in terms of its state changes and operations performed upon it.
- The visible behavior of an object is modeled by the set of messages.
- Window operations Open, Close, Maximize, Minimize , Resize, Move
- Totality of operations we can perform upon window and consequent changes in attributes defines behavior of the window

Identity

- Identity is that property of an object which distinguishes it from all other objects.
- Each object has unique identity, even if the state is identical to that of another object.
- Bank Account Balance,Interest Rate,Account no.,Customer, Account no. uniquely identifies an account among all others.

Responsibility

• The responsibility of an object is the role it serves within the system Responsibility
• Responsibility of the sensor object is to sense the temperature and initiate controller action.
• Responsibility of controller is to control the current flowing through the heater

What is a class?

• A class characterizes a set of objects that share a common structure and a common behavior.
• A class captures the essence of an object
• A class is a template for the creation of like objects.
• An object is the instantiation of a class
• The Interface of a class provides its outside view and therefore emphasizes the abstraction while hiding its structure and secrets of its behavior.
• The implementation of a class is its inside view, encompassing secrets of its behavior.

Links and association:

Link

A link represents a connection through which an object collaborates with other objects. Rumbaugh has defined it as “a physical or conceptual connection between objects”. Through a link, one object may invoke the methods or navigate through another object. A link depicts the relationship between two or more objects.

Association

Association is a group of links having common structure and common behavior. Association depicts the relationship between objects of one or more classes. A link can be defined as an instance of an association.

Degree of an Association

Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, or ternary.

- A **unary relationship** connects objects of the same class.
- A **binary relationship** connects objects of two classes.
- A **ternary relationship** connects objects of three or more classes.

Cardinality Ratios of Associations

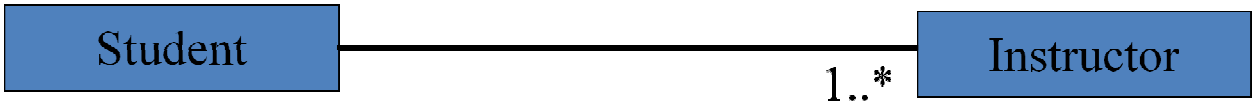
Cardinality of a binary association denotes the number of instances participating in an association. There are three types of cardinality ratios, namely:

- **One–to–One** : A single object of class A is associated with a single object of class B.
- **One–to–Many** : A single object of class A is associated with many objects of class B.
- **Many–to–Many** : An object of class A may be associated with many objects of class B and conversely an object of class B may be associated with many objects of class A.

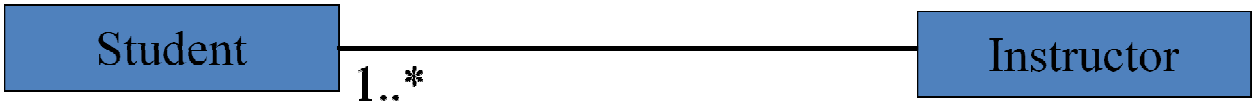
(OR)

(Association is a group of links having common structure and common behavior. Association depicts the relationship between objects of one or more classes.
A link can be defined as an instance of an association.)

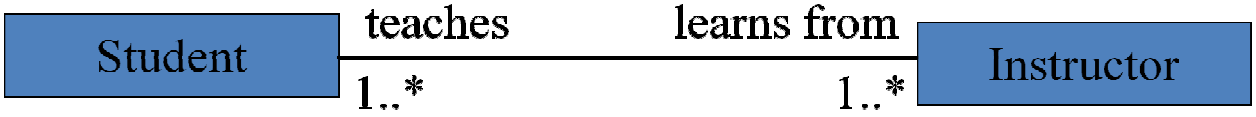
We can indicate the multiplicity of an association by adding multiplicity adornments to the line denoting the association. The example indicates that a Student has one or more Instructors:



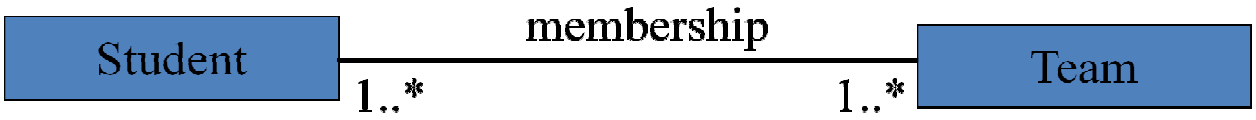
The example indicates that every Instructor has one or more Students:



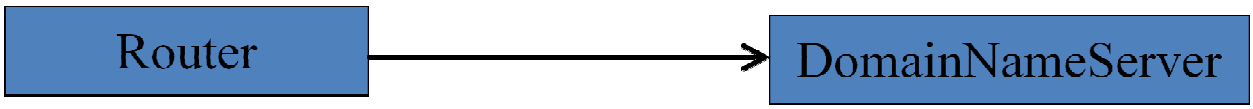
We can also indicate the behavior of an object in an association (i.e., the role of an object) using rolenames.



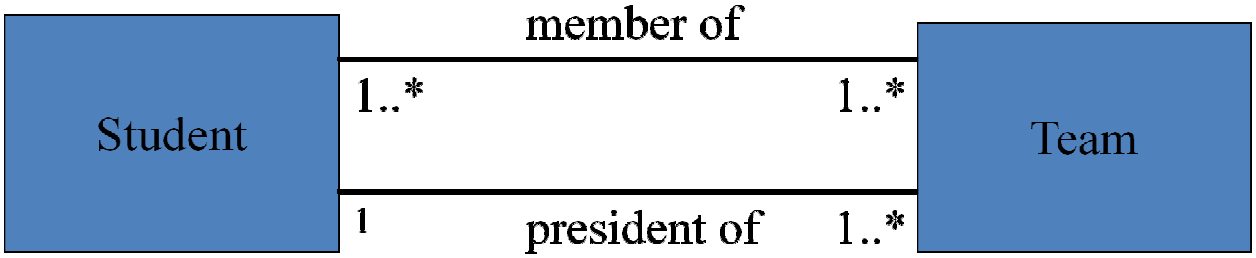
We can also name the association



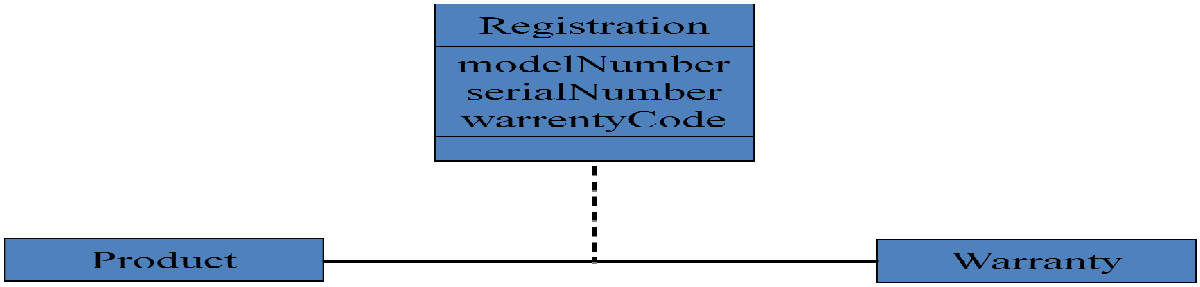
We can constrain the association relationship by defining the navigability of the association. Here, a Router object requests services from a DNS object by sending messages to (invoking the operations of) the server. The direction of the association indicates that the server has no knowledge of the Router.



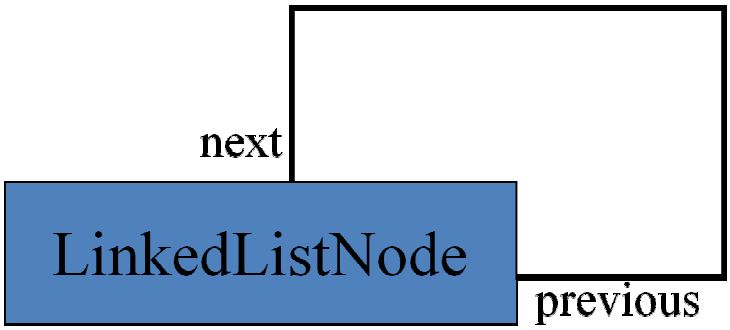
We can specify dual associations.



Associations can also be objects themselves, called link classes or an association classes.



A class can have a *self association*.

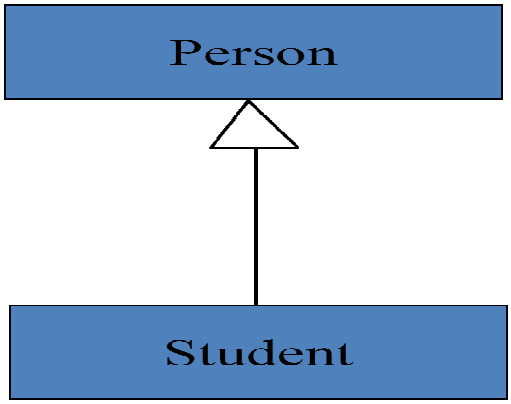


Common *multiplicity* values

Symbol	Meaning
1	One and only one
0...1	Zero o r one
*	Zero to any positive no.
1..*	One to any positive no.
m..n	From m to n, m and n are integers and m<n

Generalization and specialization:

- A generalization connects a subclass to its superclass. It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.



(or)

Generalization is the process of extracting shared characteristics from two or more classes, and combining them into a generalized superclass. Shared characteristics can be attributes, associations, or methods.

(OR)

Generalization

In the generalization process, the common characteristics of classes are combined to form a class in a higher level of hierarchy, i.e., subclasses are combined to form a generalized super-class. It represents an "is – a – kind – of" relationship. For example, "car is a kind of land vehicle", or "ship is a kind of water vehicle".

Specialization

Specialization is the reverse process of generalization. Here, the distinguishing features of groups of objects are used to form specialized classes from existing classes. It can be said that the subclasses are the specialized versions of the super-class.

Inheritance:

- Inheritance is the sharing of attributes and operations among classes based on hierarchical relationship.
- Inheritance is a property of a class hierarchy whereby each subclass inherits attributes and methods of its super-class.
- The subclass can have additional specific attributes and methods.

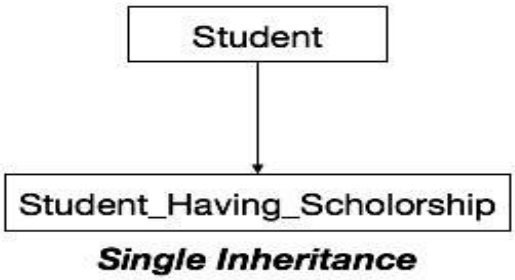
Inheritance is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities.

Example

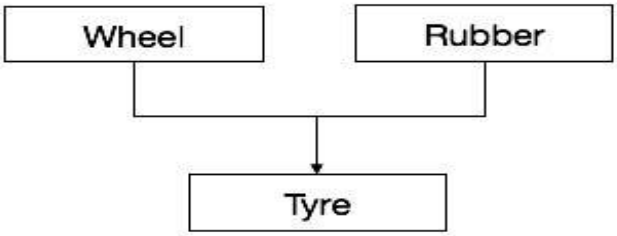
From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc. Humans, cats, dogs, and cows all have the distinct characteristics of mammals. In addition, each has its own particular characteristics. It can be said that a cow "is – a" mammal.

Types of Inheritance:

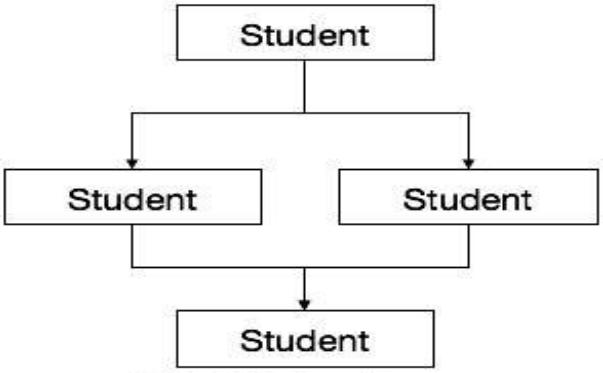
- **Single Inheritance** : A subclass derives from a single super-class.
- **Multiple Inheritance** : A subclass derives from more than one super-classes.
- **Multilevel Inheritance** : A subclass derives from a super-class which in turn is derived from another class and so on.
- **Hierarchical Inheritance** : A class has a number of subclasses each of which may have subsequent subclasses, continuing for a number of levels, so as to form a tree structure.
- **Hybrid Inheritance** : A combination of multiple and multilevel inheritance so as to form a lattice structure.



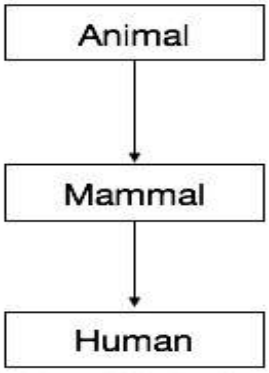
Single Inheritance



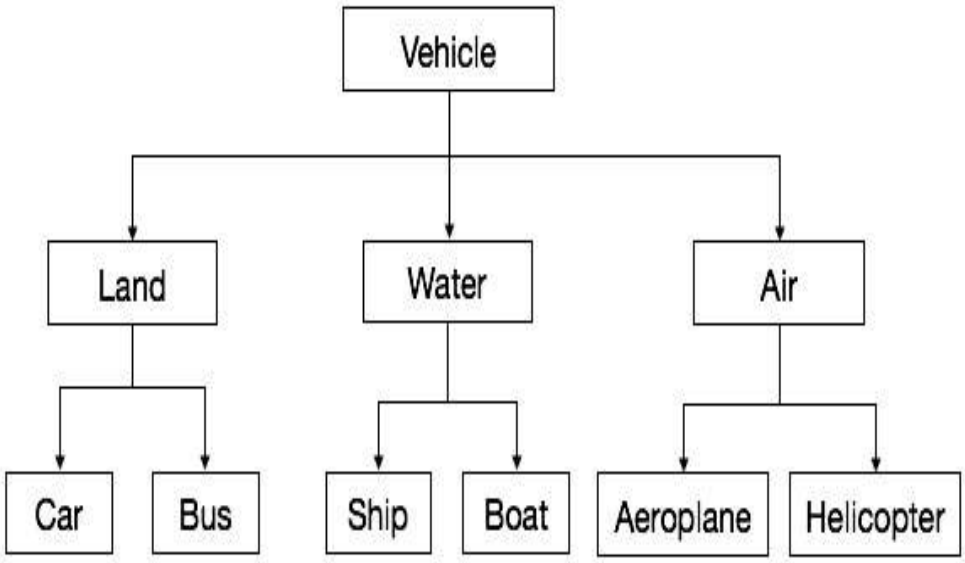
Multiple Inheritance



Hybrid Inheritance



Multi-level Inheritance



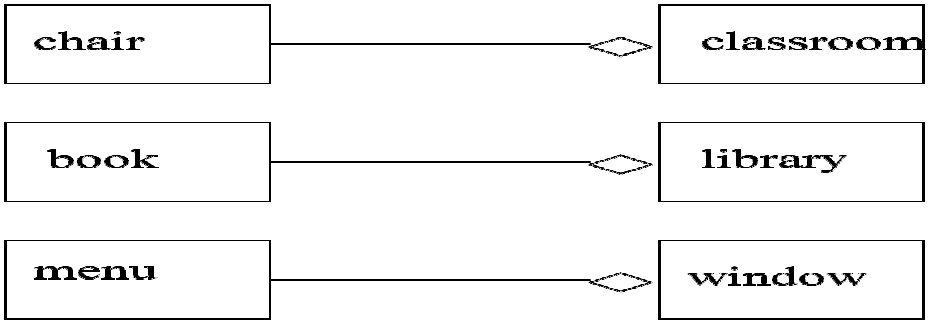
Hierarchical Inheritance

Grouping Things: They comprise the organizational parts of the UML models. There is only one kind of grouping thing, i.e., package.

Aggregation:

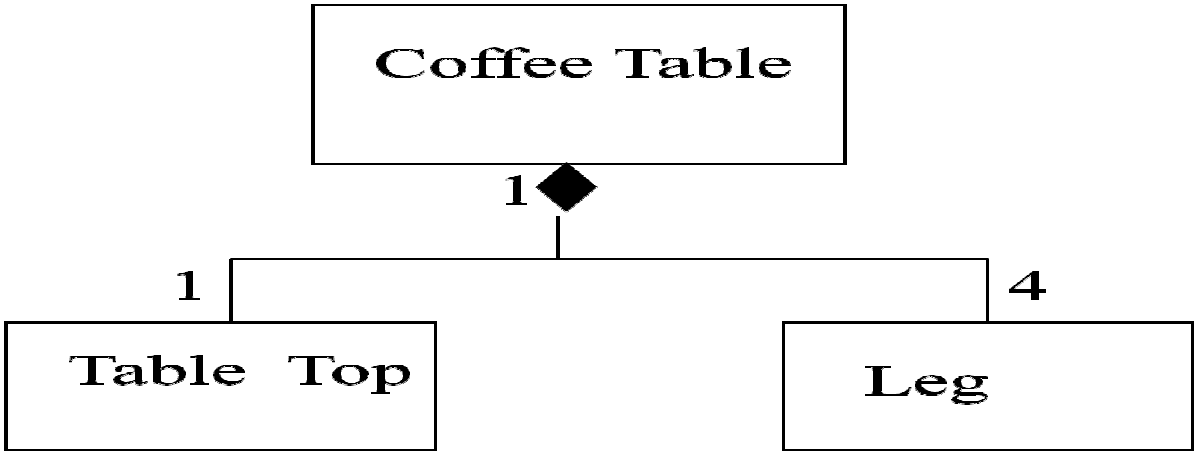
If two classes share association relationship and one class can be said to be a part of the other, the relationship is called aggregation.

Aggregation expresses a relationship between a 'whole' and its ' part'.



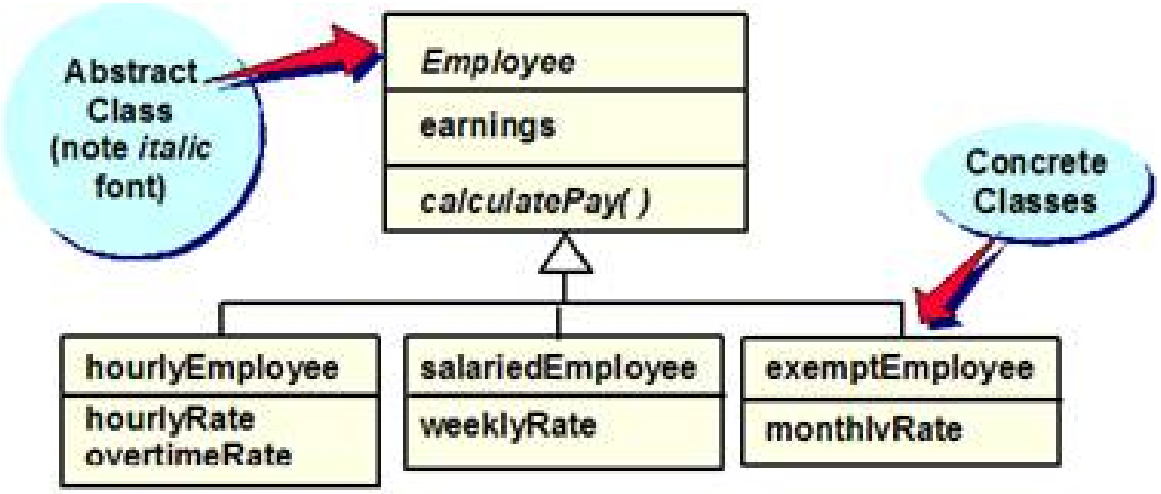
Aggregation

Composition is a strong type of aggregation. Each component in a composite can belong to just one whole. The composite is responsible for creation or deletion of its part. e.g. building and rooms, purchase order and order line.



Abstracts classes:

Abstract means that a class cannot have its own (direct) instances. Concrete classes can be instantiated but abstract classes cannot directly be instantiated. Abstract classes exist purely to generalize common behavior that would otherwise be duplicated across (sub)classes. We will illustrate this with the following diagram.



Polymorphism:

- The ability of the objects to respond to the same message in different ways is called Polymorphism
- Polymorphism means that the same operation may behave differently for different classes.
- The ability to hide many different implementation behind a single interface.

  Eg: Given the command to 'SPEAK';

  A man : How do you do?

  A dog : Woof!!!

  A cat : Meow!!!

(OR)

Polymorphism:

Polymorphism is originally a Greek word that means the ability to take multiple forms. In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instance they are operating upon. Polymorphism allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing inheritance.

Example

Let us consider two classes, Circle and Square, each with a method findArea(). Though the name and purpose of the methods in the classes are same, the internal implementation, i.e., the procedure of calculating area is different for each class. When an object of class Circle invokes its findArea() method, the operation finds the area of the circle without any conflict with the findArea() method of the Square class.

Metadata:

Metadata is such set of data which describes other data. For example, if you have to describe an object, you must have a description of the class from which it is instantiated. Here, the data used to describe class will be treated as metadata. You may observe that every real-world thing may have meta data, because every real world thing has a description for them. Let us take the example of institutes. and their directors. You can store that school A is having X as its direct, School B is having Y as its director, and so on. Now, you have concrete information to keep in metadata that is every institute is having a director. (OR)

Meta data is “data about the data”, “model about models”, “definitions of computing and business assets” and more:

- Technical: Design and Middleware meta data

-Administrative and process meta data

-Business meta data

End users, Software designers and IT personnel need to know:

- What does the data mean? What is its structure and format?

- Where did it come from? How was it calculated?

- When was it loaded? Who owns it? Where is it used?

-Etc.

Most metadata undocumented and usually locked up causing serious interchange and interoperability problems

Constraints:

UML Constraint. A constraint is a packageable element which represents some condition, restriction or assertion related to some element (that owns the constraint) or several elements. Constraint is usually specified by a Boolean expression which must evaluate to a true or false. Constraint must be satisfied (i.e. evaluated to **true**) by a correct design of the system. Constraints are commonly used for various elements on class diagrams.

Dynamic modeling:

The dynamic model represents the time–dependent aspects of a system. It is concerned with the temporal changes in the states of the objects in a system. The main concepts are:

- State, which is the situation at a particular condition during the lifetime of an object.
- Transition, a change in the state
- Event, an occurrence that triggers transitions
- Action, an uninterrupted and atomic computation that occurs due to some event, and
- Concurrency of transitions.

A state machine models the behavior of an object as it passes through a number of states in its lifetime due to some events as well as the actions occurring due to the events. A state machine is graphically represented through a state transition diagram.

States and State Transitions

State

The state is an abstraction given by the values of the attributes that the object has at a particular time period. It is a situation occurring for a finite time period in the lifetime of an object, in which it fulfils certain conditions, performs certain activities, or waits for certain events to occur. In state transition diagrams, a state is represented by rounded rectangles.

Parts of a state

- **Name** : A string differentiates one state from another. A state may not have any name.
- **Entry/Exit Actions** : It denotes the activities performed on entering and on exiting the state.
- **Internal Transitions** : The changes within a state that do not cause a change in the state.
- **Sub–states** : States within states.

Initial and Final States

The default starting state of an object is called its initial state. The final state indicates the completion of execution of the state machine. The initial and the final states are pseudo-states, and may not have the parts of a regular state except name. In state transition diagrams, the initial state is represented by a filled black circle. The final state is represented by a filled black circle encircled within another unfilled black circle.

Transition

A transition denotes a change in the state of an object. If an object is in a certain state when an event occurs, the object may perform certain activities subject to specified conditions and change the state. In this case, a state–transition is said to have occurred. The transition gives the relationship between the first state and the new state. A transition is graphically represented by a solid directed arc from the source state to the destination state.

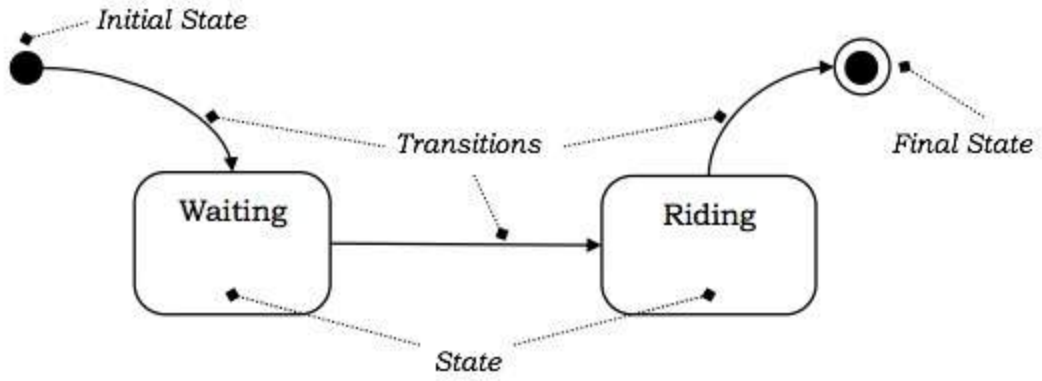
The five parts of a transition are:

- **Source State** : The state affected by the transition.
- **Event Trigger** : The occurrence due to which an object in the source state undergoes a transition if the guard condition is satisfied.
- **Guard Condition** : A Boolean expression which if True, causes a transition on receiving the event trigger.

- **Action** : An un-interruptible and atomic computation that occurs on the source object due to some event.
- **Target State** : The destination state after completion of transition.

Example

Suppose a person is taking a taxi from place X to place Y. The states of the person may be: Waiting (waiting for taxi), Riding (he has got a taxi and is travelling in it), and Reached (he has reached the destination). The following figure depicts the state transition.



Events

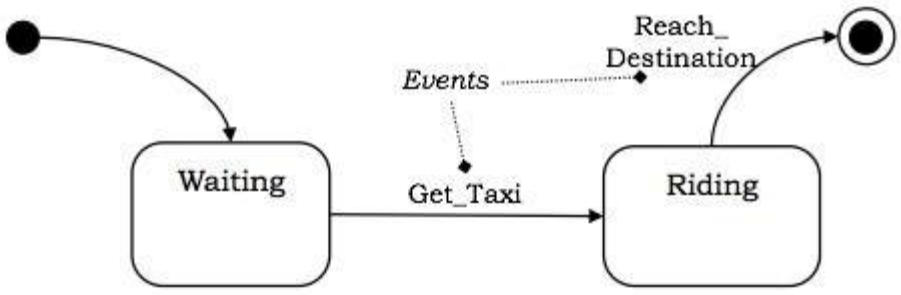
Events are some occurrences that can trigger state transition of an object or a group of objects. Events have a location in time and space but do not have a time period associated with it. Events are generally associated with some actions.

Examples of events are mouse click, key press, an interrupt, stack overflow, etc.

Events that trigger transitions are written alongside the arc of transition in state diagrams.

Example

Considering the example shown in the above figure, the transition from Waiting state to Riding state takes place when the person gets a taxi. Likewise, the final state is reached, when he reaches the destination. These two occurrences can be termed as events Get_Taxi and Reach_Destination. The following figure shows the events in a state machine.



External and Internal Events

External events are those events that pass from a user of the system to the objects within the system. For example, mouse click or key−press by the user are external events.

Internal events are those that pass from one object to another object within a system. For example, stack overflow, a divide error, etc.

Deferred Events

Deferred events are those which are not immediately handled by the object in the current state but are lined up in a queue so that they can be handled by the object in some other state at a later time.

Event Classes

Event class indicates a group of events with common structure and behavior. As with classes of objects, event classes may also be organized in a hierarchical structure. Event classes may have attributes associated with them, time being an implicit attribute. For example, we can consider the events of departure of a flight of an airline, which we can group into the following class:

Flight_Departs (Flight_No, From_City, To_City, Route)

Concurrency:

Concurrency allows more than one objects to receive events at the same time and more than one activity to be executed simultaneously. Concurrency is identified and represented in the dynamic model.

To enable concurrency, each concurrent element is assigned a separate thread of control. If the concurrency is at object level, then two concurrent objects are assigned to two different threads of control. If two operations of a single object are concurrent in nature, then that object is split among different threads.

Concurrency is associated with the problems of data integrity, deadlock, and starvation. So a clear strategy needs to be made whenever concurrency is required. Besides, concurrency requires to be identified at the design stage itself, and cannot be left for implementation stage.

Reference Books:

1. Rambaugh: Object Oriented modeling and design PHI
2. Craig larman--Prentice Hall - Applying UML And Patterns 2nd Edition(2001)
3. Booch, Object Oriented analysis and design with applications, Addison Wesley

Online References:

1. http://elearning.vtu.ac.in/newvtuelc/courses/06CS-IS71.html
2. https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_analysis.htm
3. http://www.uml-diagrams.org/constraint.html