Parul® University

Object Oriented Concepts with UML (203105207)

Prof. Shaleen Shukla, Assistant Professor
Information Technology

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-2

Class Modelling: Part I

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-2

Class and State Modeling

Objects

- An object is the instance of the class, which helps programmers to use variables and methods from inside the class.
- ***Object*** acts like a variable of the class.
- ***Objects*** can be declared several times depending on the requirement.
- Each object is an instance of a particular class or subclass with the class's own methods or procedures and data variables.
- ***Objects have three responsibilities:***

- What they know about themselves – (e.g., Attributes)
- What they do – (e.g., Operations)
- What they know about other objects – (e.g., Relationships)

Object

An object, then, the Animal can have individual objects like cow, lion, dog etc. These are object not only because it has different names, but also because they have different values assigned to their properties.

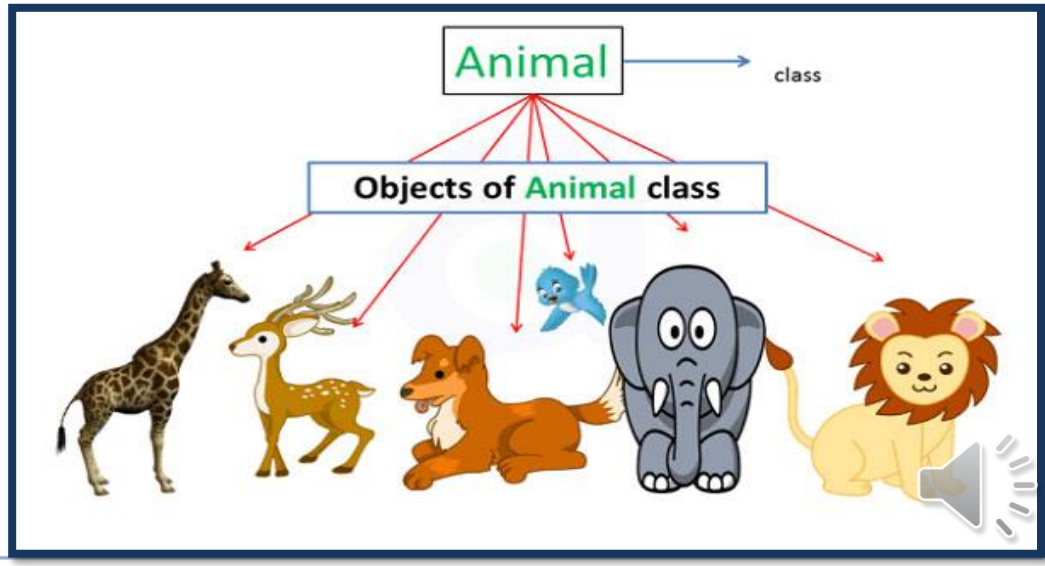


Image source : Google

Class

- A class is a blueprint from which you can create the instance, i.e., objects.
- A class is used to bind data as well as methods together as a single unit.
- The class has to be declared only once.
- Class diagrams provide a graphic notation for modeling classes and their relationships. The UML symbol for a class is a box.

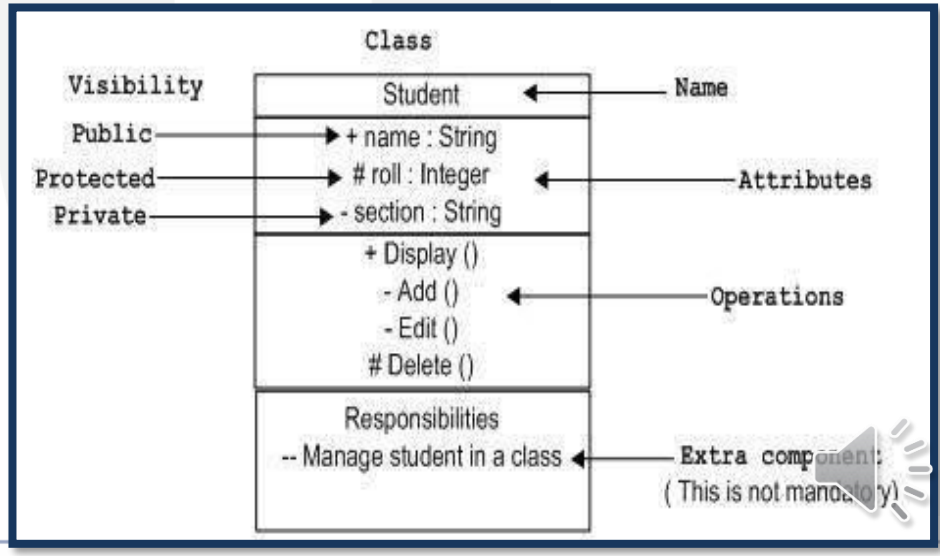


Image source : Google

Classes and Objects with an example

Stop here right now! List down the differences…..

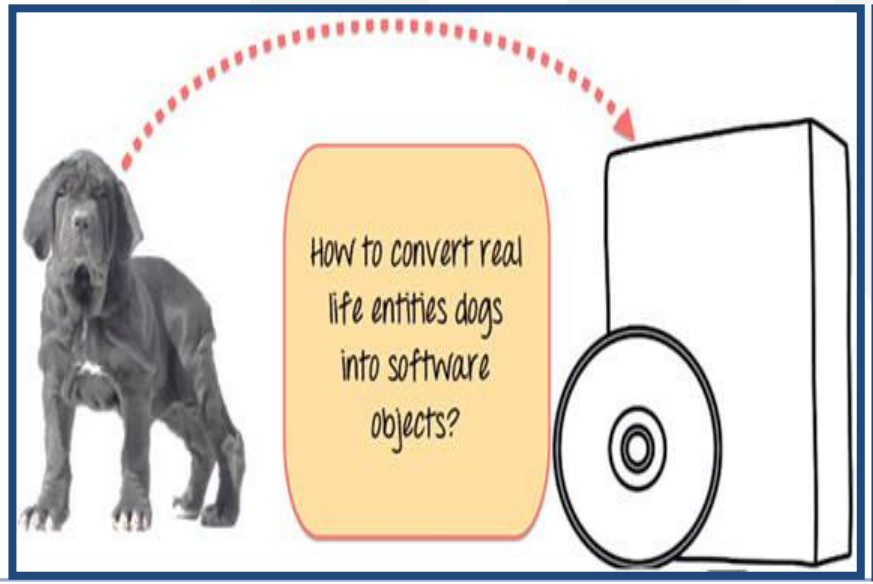
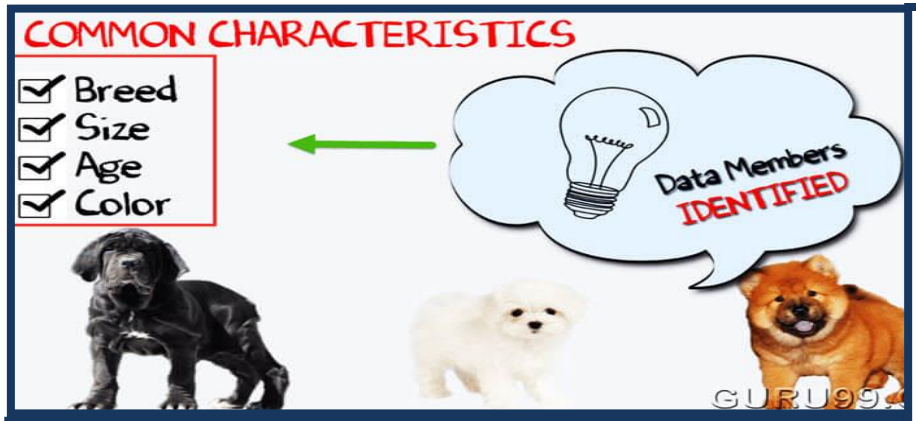
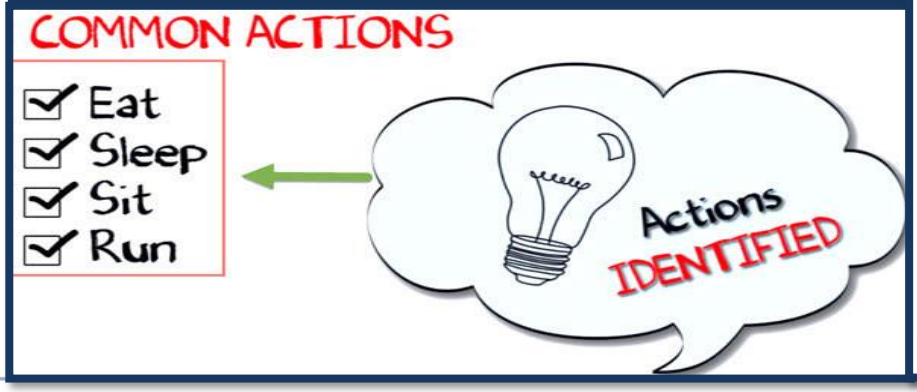


Image source : Google

List out! Difference…





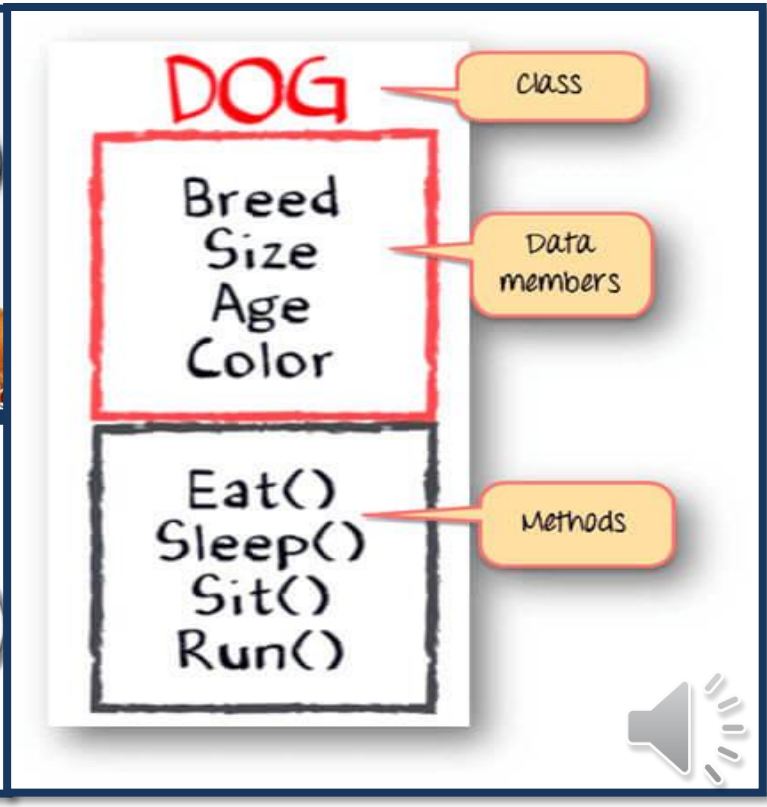


Image source : Google

Class

A class represent a concept which encapsulates state (attributes) and behavior (operations). Each attribute has a type. Each operation has a signature. *The class name is the only mandatory information*.

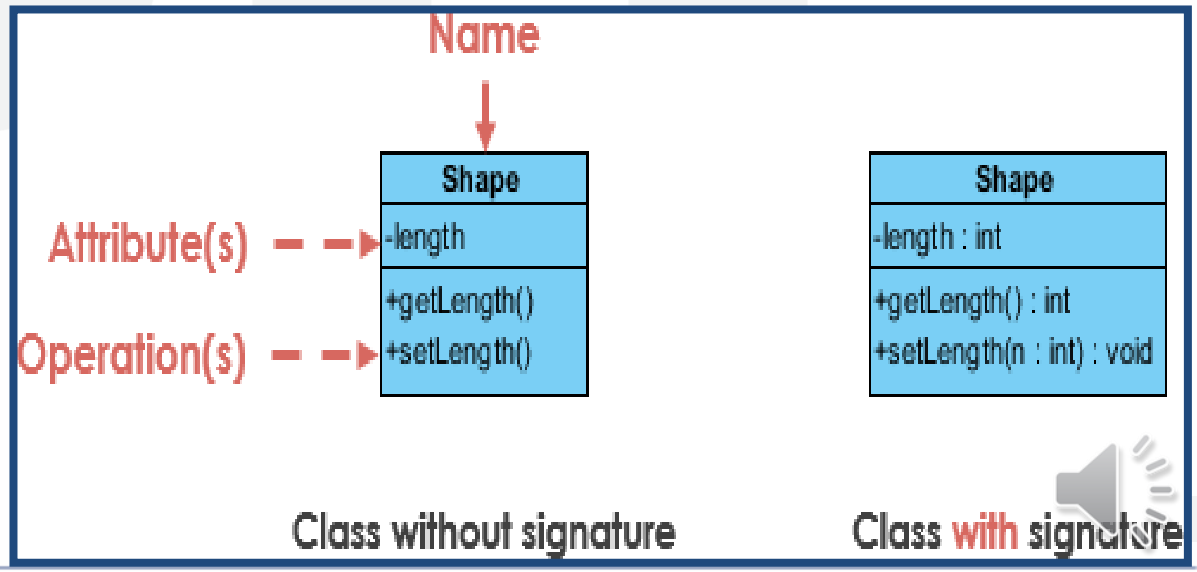


Image source : Google

Visibility

•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

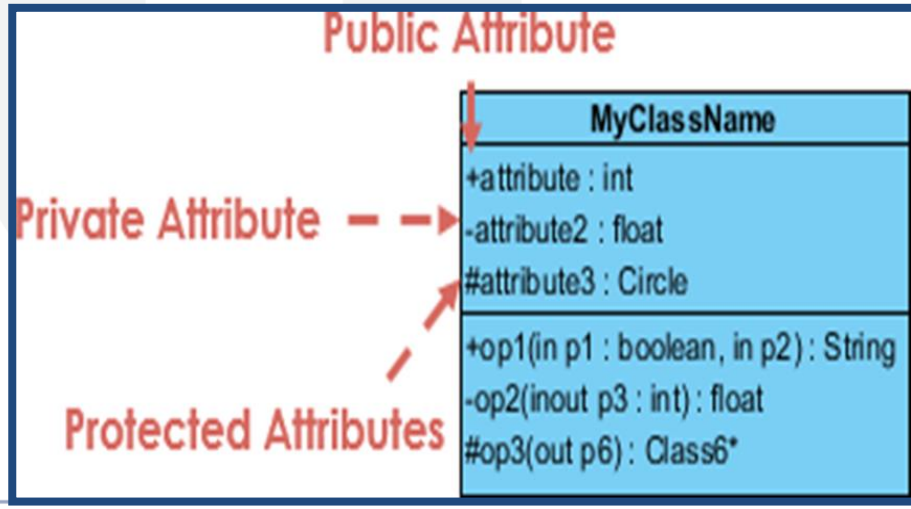


Image source : Google

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-2

RELATIONSHIP

Relationships

A Relationship is what a class or an object knows about another class or object.

Generalization (Class-to-Class) (Superclass/Subclass)

Inheritance

Ex: Person - FacultyPerson, StudentPerson, Staff..

[Object] Associations

FacultyInformation - CourseInformation

StudentInformation - CourseInformation

[Object] Aggregations & Composition (Whole-Part)

Assembly - Parts

Group - Members

Container - Contents

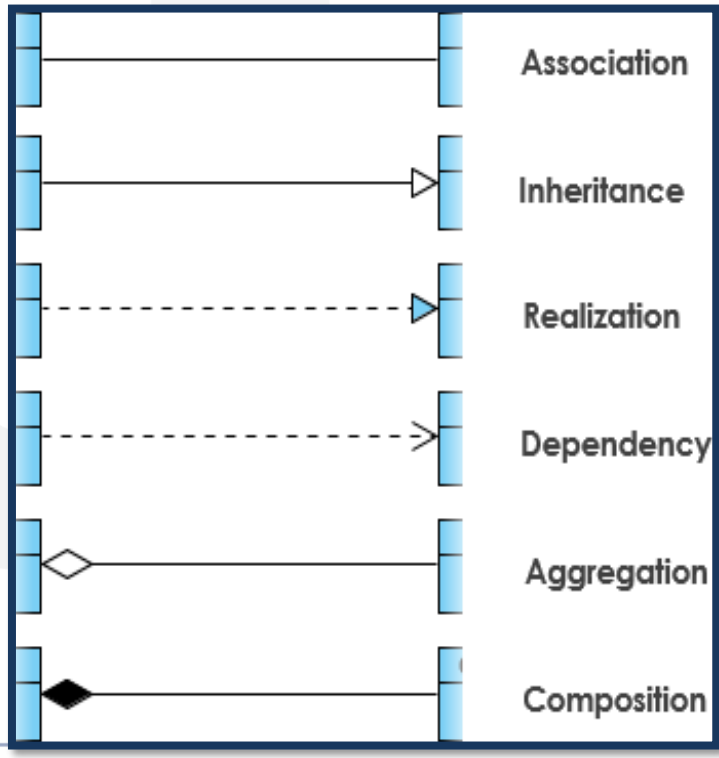


Image source : Google

Relationships

The logical or physical connection among objects is referred to as **link**.
These links are used to relate multiple objects and represent a relationship between objects.

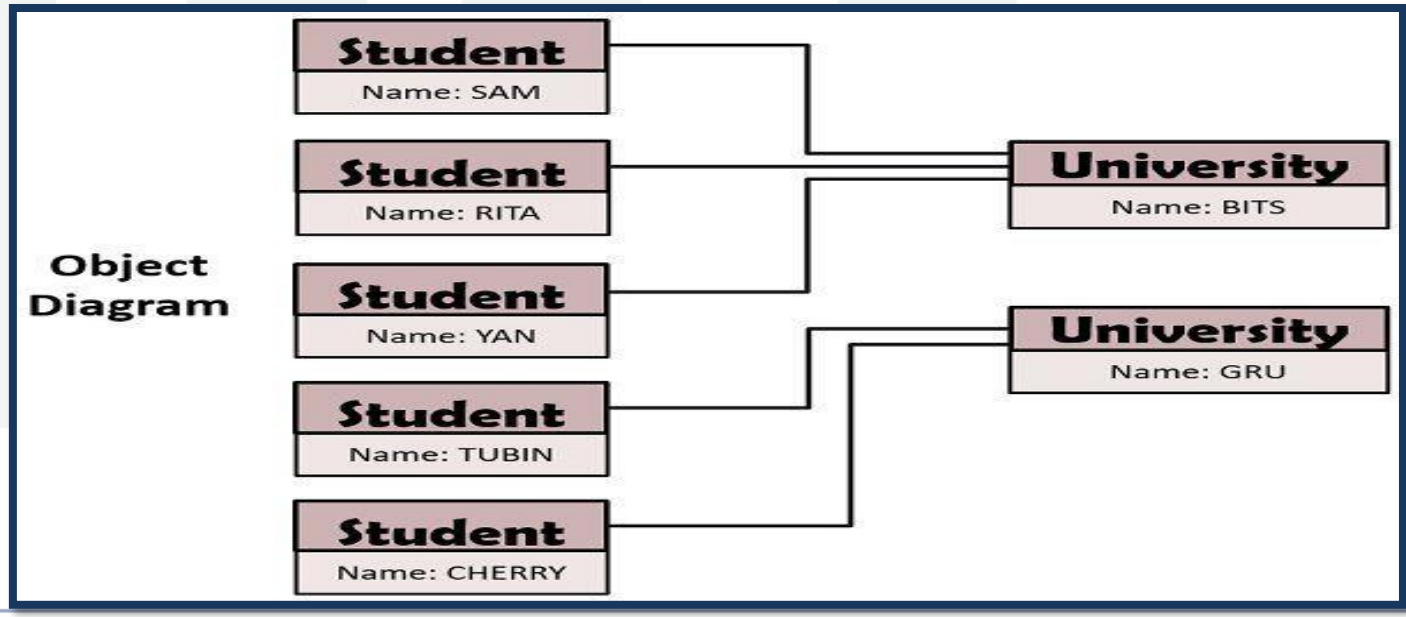


Image source : Google

Association

This kind of relationship represents static relationships between classes A and B. For example; an employee works for an organization.
Here are some rules for Association:

-Association is mostly verb or a verb phrase or noun or noun phrase.

-It should be named to indicate the role played by the class attached at the end of the association path.

-Mandatory for reflexive associations

-Also uses line segment but it shows the connection between classes.

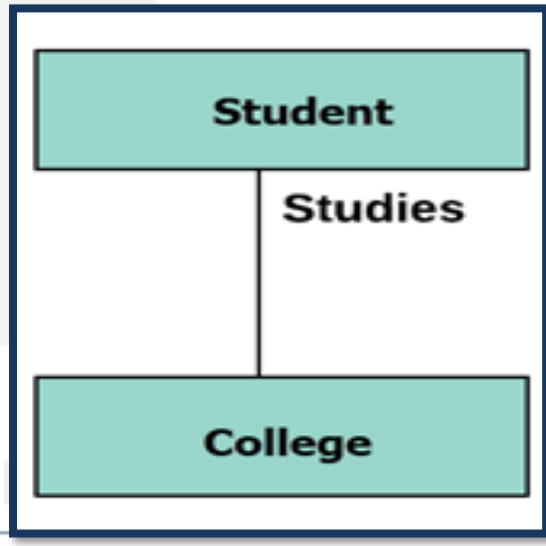


Image source : Google

Association End Names

An Association End Notation is a graphical notation used in a UML Class Diagram to represent connection properties of an association to a connected class.

Association End Name - A text placed near the connection point of an end of the association line and the connected class to describe the role the connected class played in the association.

Association End Multiplicity - A numeric range placed near the connection point of an end of the association line and the connected class to indicate how many instances of the connected class can be related to a single instance of the opposite class.

For example, the "Teacher" class and the "Class" class have an association that "A teacher teaches in a class".

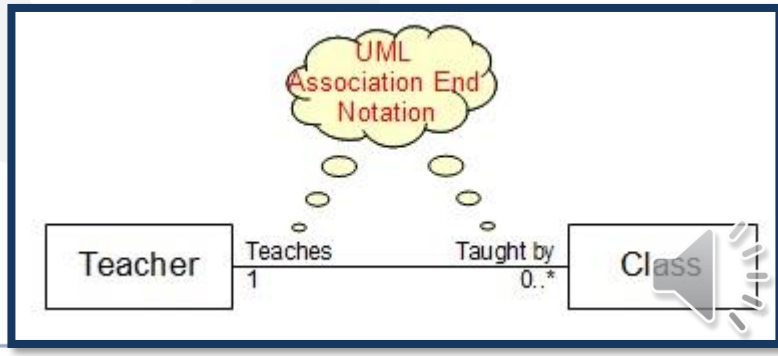


Image source : Google

Multiplicity

Specifies the number of instances of one class that may relate to a single instance of an associated class. It constraints the number of related objects.

It may be as “one” or “many”, but more generally it is a subset of none negative numbers.

UML diagrams explicitly list multiplicity at the end of the association

Such as

“1” (exactly one)

“1..*” (one or more)

“3..5” (three or five)

“*” (zero or more)

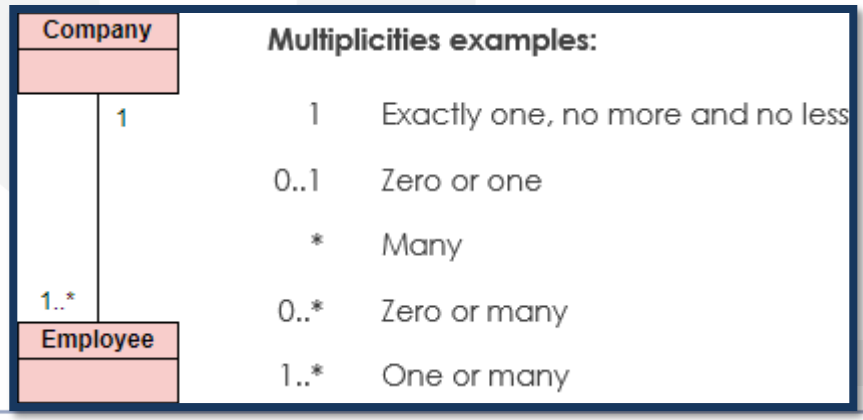


Image source : Google

Aggregation

Aggregation is a special type of association that models a whole- part relationship between aggregate and its parts.

For example, the class college is made up of one or more student.

In aggregation, the contained classes are never totally dependent on the lifecycle of the container. Here, the college class will remain even if the student is not available.

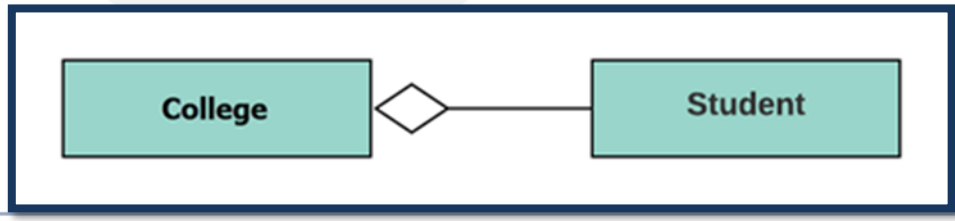


Image source : Google

Composition

The composition is a special type of aggregation which denotes strong ownership between two classes when one class is a part of another class.
For example, if college is composed of classes student. The college could contain many students, while each student belongs to only one college. So, if college is not functioning all the students also removed.

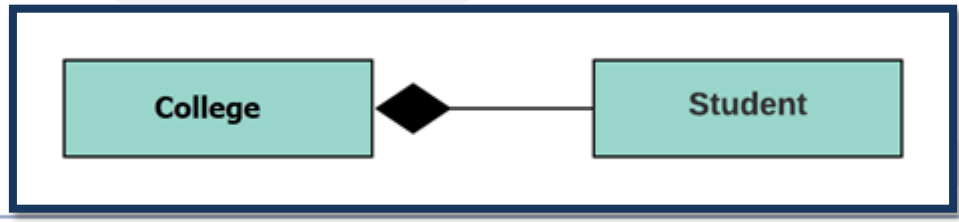
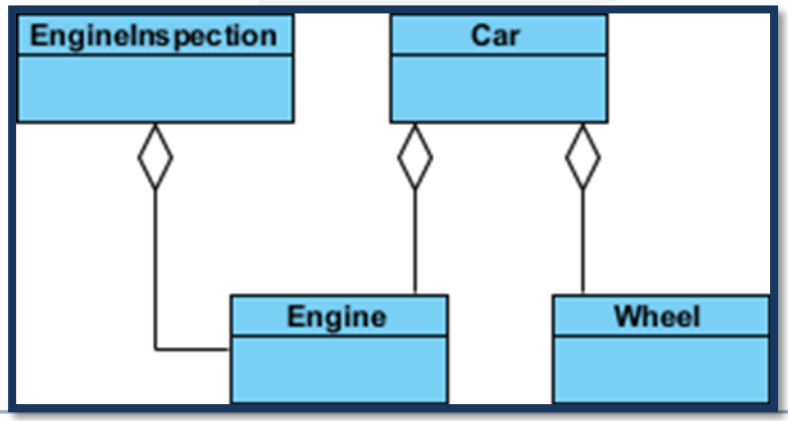


Image source : Google

Aggregation v/s Composition

Aggregation	Composition
Aggregation indicates a relationship where the child can exist separately from their parent class. Example: Automobile (Parent) and Car (Child). So, If you delete the Automobile, the child Car still exist.	Composition display relationship where the child will never exist independent of the parent. Example: House (parent) and Room (child). Rooms will never separate into a House
Aggregation is a **weak** Association.	Composition is a **strong** Association



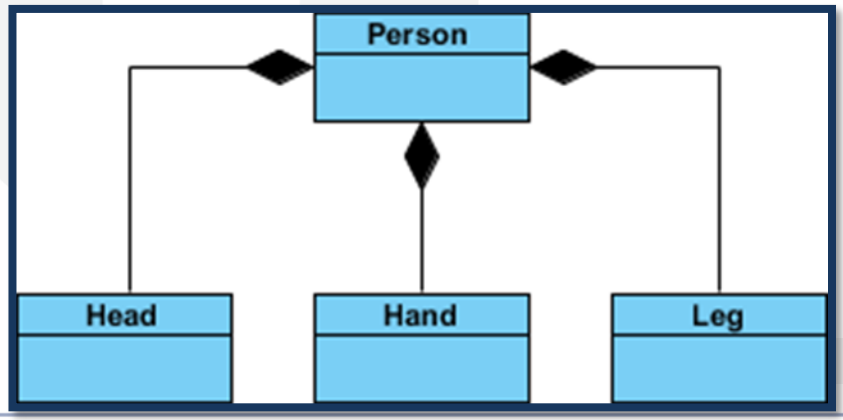


Image source : Google

Generalization

Generalization uses a “is-a” relationship from a specialization to the generalization class.

Common structure and behavior are used from the specialization to the generalized class.

At a very broader level you can understand this as inheritance.

Why I take the term inheritance is, you can relate this term very well.

Generalization is also called a “Is-a” relationship.

In other words, a superclass has the most general attributes, operations, and relationships that may be shared with subclasses. A subclass may have more specialized attributes and operations.

Image source : Google

Specialization

Specialization is the reverse process of Generalization means creating new sub-classes from an existing class.

For Example, a Bank Account is of two types - Savings Account and Credit Card Account. Savings Account and Credit Card Account inherit the common/ generalized properties like Account Number, Account Balance, etc. from a Bank Account and also have their specialized properties like unsettled payment etc.

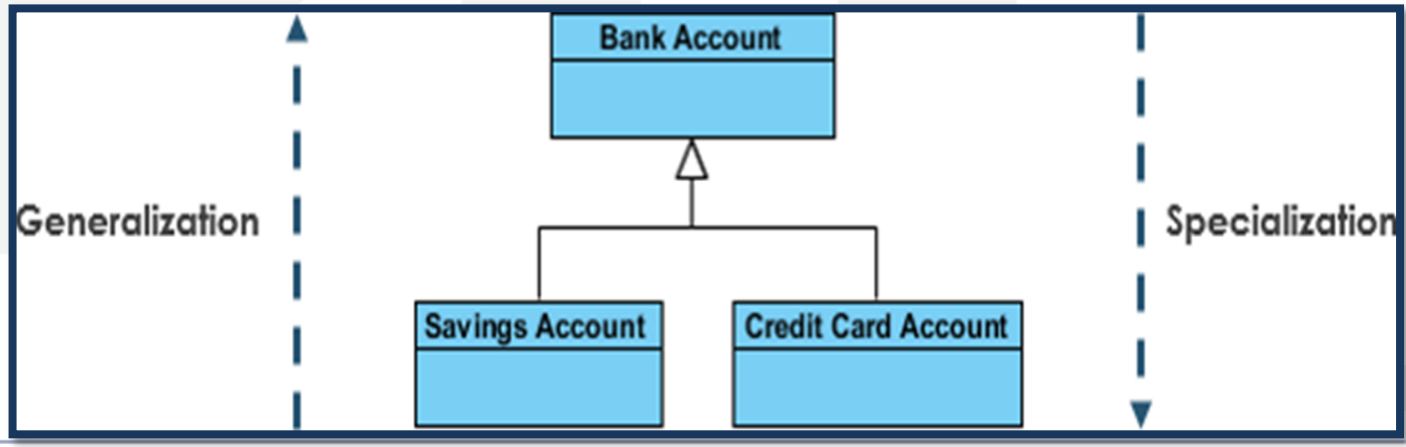


Image source : Google

Inheritance

In OOAD inheritance is usually defined as a mechanism by which more specific classes (called subclasses or derived classes) incorporate structure and behavior of more general classes (called super classes or base classes).

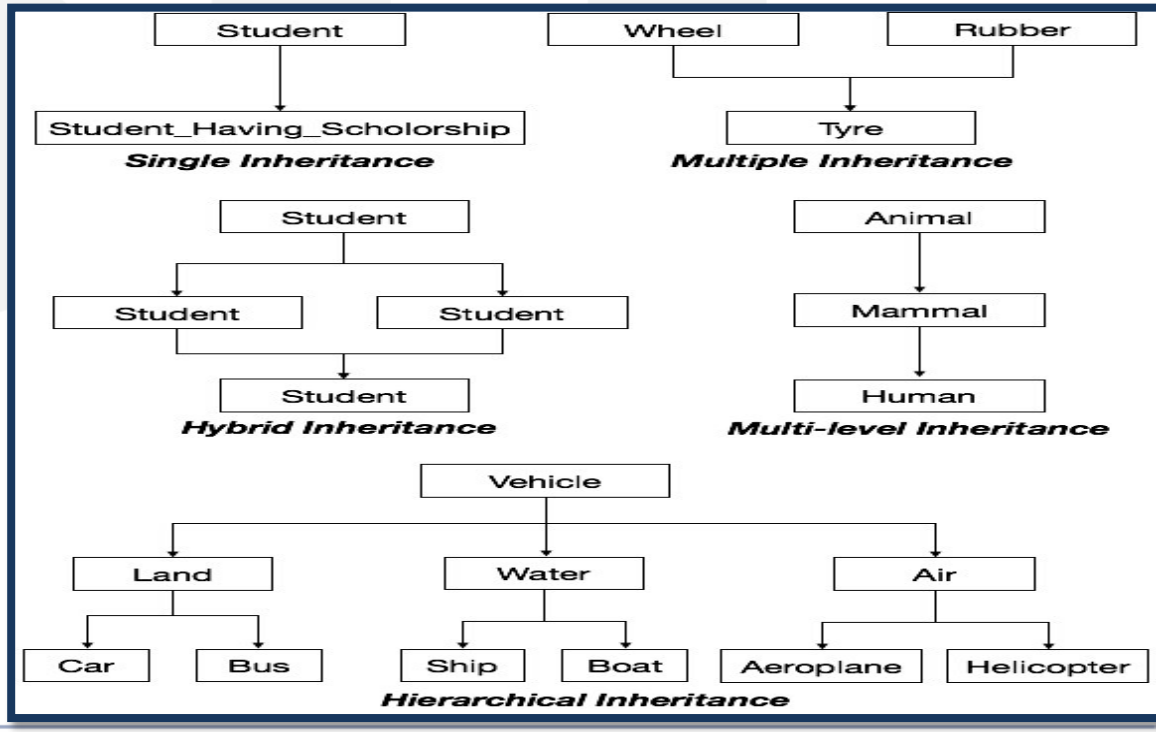


Image source : Google

Abstract Class

It is a class with an operation prototype, but not the implementation. It is also possible to have an abstract class with no operations declared inside of it.
An abstract is useful for identifying the functionalities across the classes.
Abstraction reduces the complexity by hiding low level details.
Suppose we have an abstract class called as a motion with a method or an operation declared inside of it. The method declared inside the abstract class is called a move ().
In UML, the abstract class has the same notation as that of the class. The only difference between a class and an abstract class is that the class name is strictly written in an italic font. An abstract class cannot be initialized or instantiated.

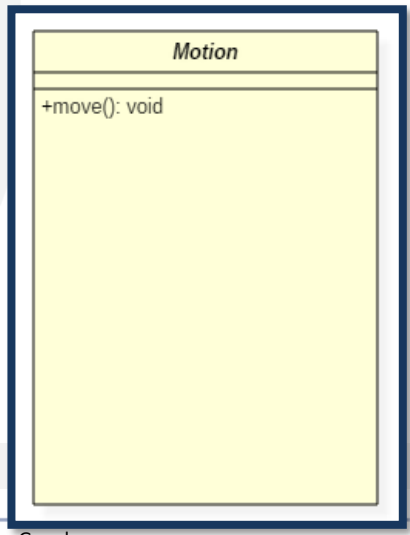


Image source : Google

Association Class

Association classes allow you to add attributes, operations, and other features to associations.

-You can't attach the same class to more than one association; an association class is the association.

-The name of the association is usually omitted since it is considered to be the same as that of the attached class.

-Distinguish between the use of an association class as a modeling technique and the implementation of the association class.

-There can be several ways to implement an association class.

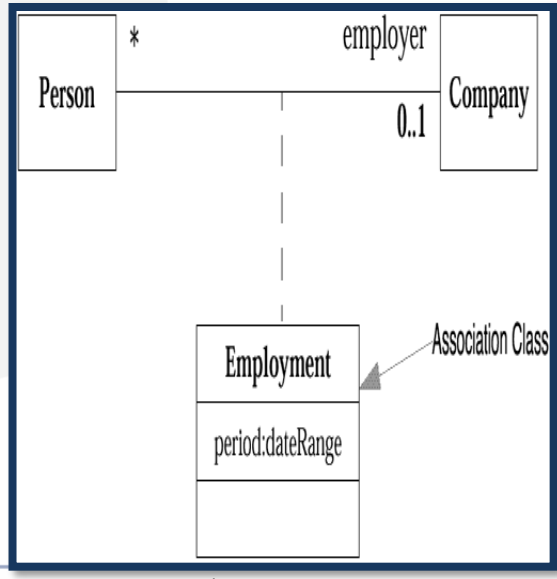


Image source : Google

Example

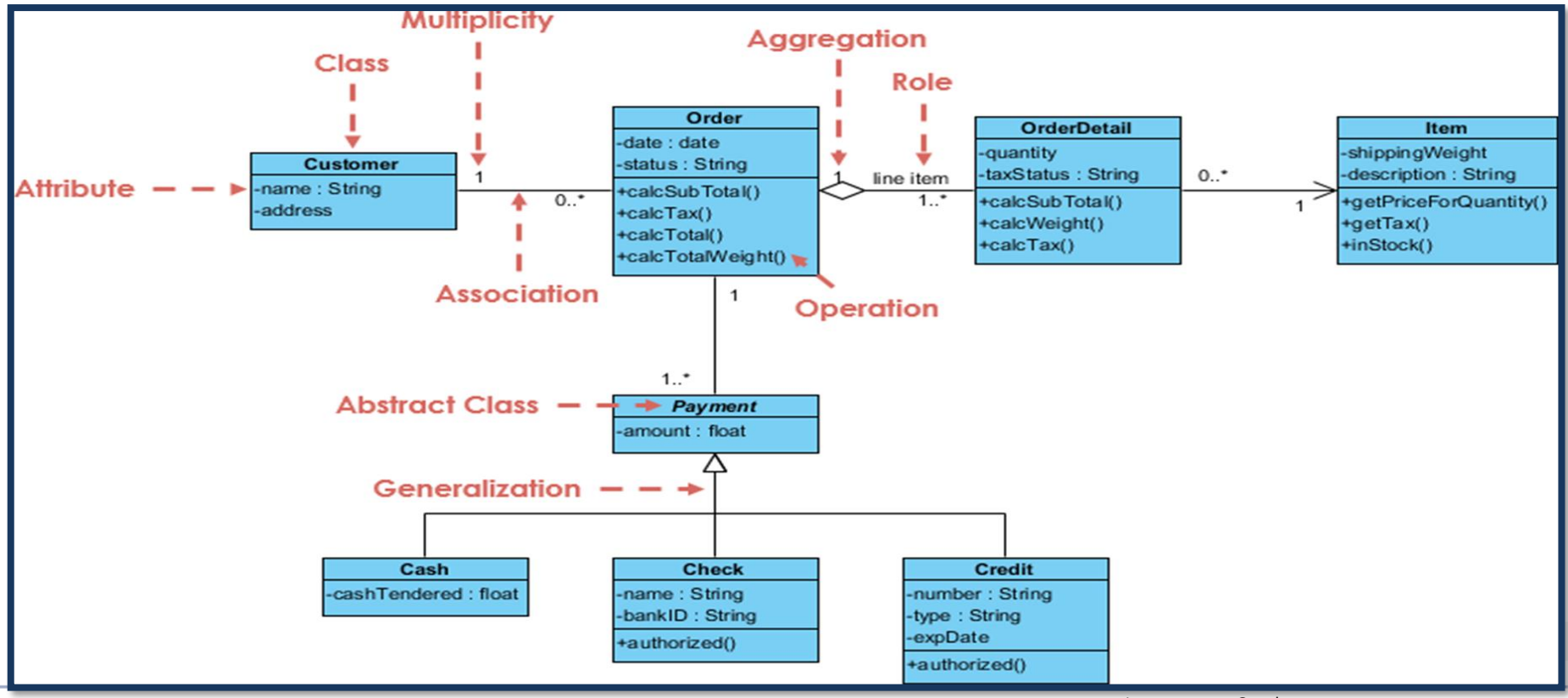


Image source : Google

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-2

VALUES AND ATTRIBUTES

Value And Attributes

- A value is a piece of data
- An attribute is a named property of a class that describes a value held by each object
- Ex. Name, birthdate and weight are attributes of Person objects
- Color, model Year and weight are attributes of Car objects
- Red is value for attribute Color
- An attribute should describe values, not objects
- Unlike objects, values lack identity
- Ex. All occurrences of the integer “17” are indistinguishable
- The country Canada is an object, whose name attribute has a value “Canada”.

Example

Attribute defines values that can be attached to instances of the class or interface.

Class with Attributes

Person
name: string birthdate: date

Objects with Values

JoeSmith:Person
name= “Joe Smith” birthdate= 21 Oct 1989

MarySharp:Person
name= “Mary Sharp” birthdate= 16 Mar 1979

Image source : Google

Operation Methods

An operation is a function or procedure that may be applied to or by objects in a class.

Open, close, hide, and redisplay are operations on class Window.

All objects in a class share the same operations.

A method is the implementation of an operation for a class.

Ex. The class *File* may have an operation *print.*

When an operation has methods on several classes, all the methods should have the same signature.

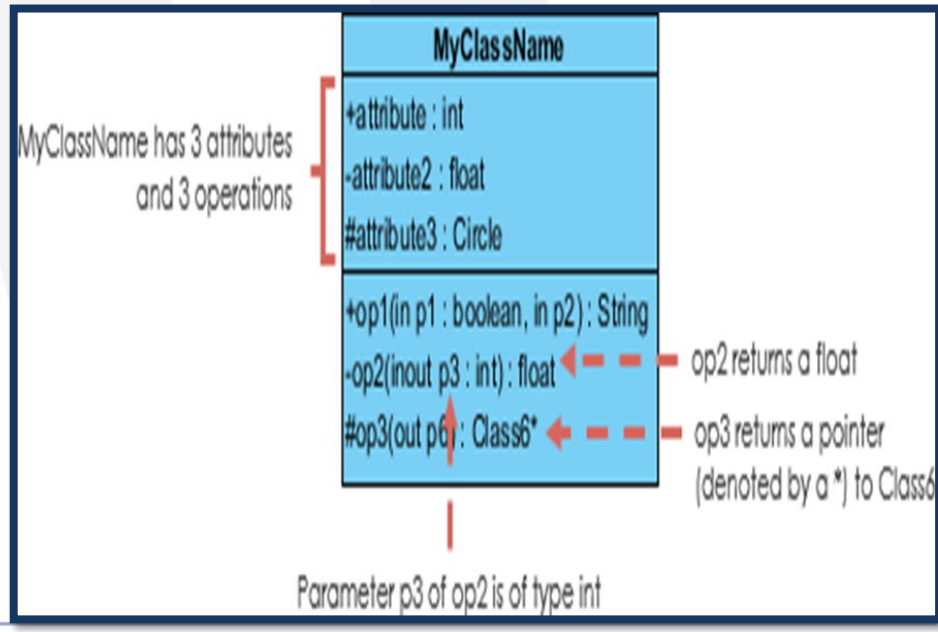


Image source : Google

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-2

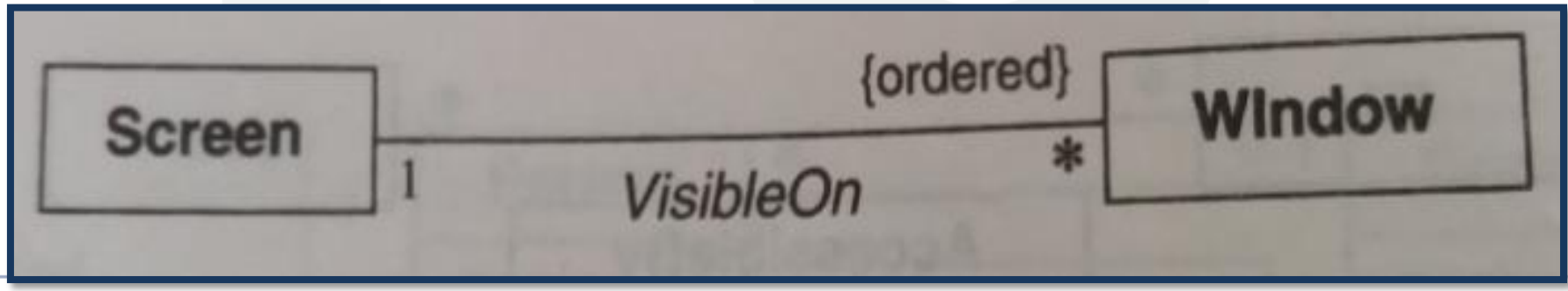
ORDERED ,BAG & SEQUENCE

Ordered

ordered means that the objects are ordered in the collection of associated elements. This means that there is a mapping between a positive integer to the elements in the collection. It doesn't say more, and in particular, it doesn't say if the elements in the associated collection are unique or not.

Ex. A workstation screen containing a number of overlapping windows, Each window on a screen occurs at most once

It can be indicates by writing “{ordered}” next to the appropriate association end.



Bag,Sequence

Ordinarily a binary association has at most one link for a pair of objects. You can permit multiple links for a pair of objects

Use an association end with {bag} or {sequence}

A bag is a collection of elements with duplicates allowed

A sequence is an ordered collection of elements with duplicate allowed

An itinerary may visit multiple airports, so you should use {sequence} and not {ordered}

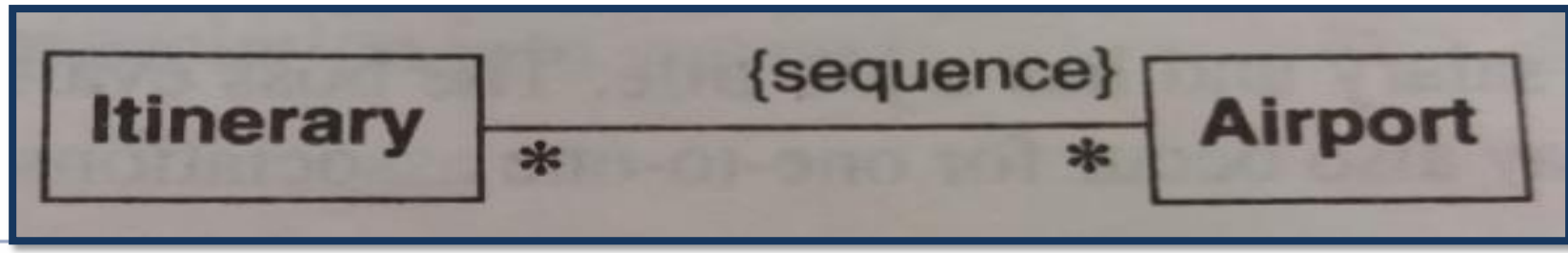


Image source : Google

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-2

Class Modelling: Part II

Association Classes

• **Association classes** allow you to add attributes, operations, and other features to associations.

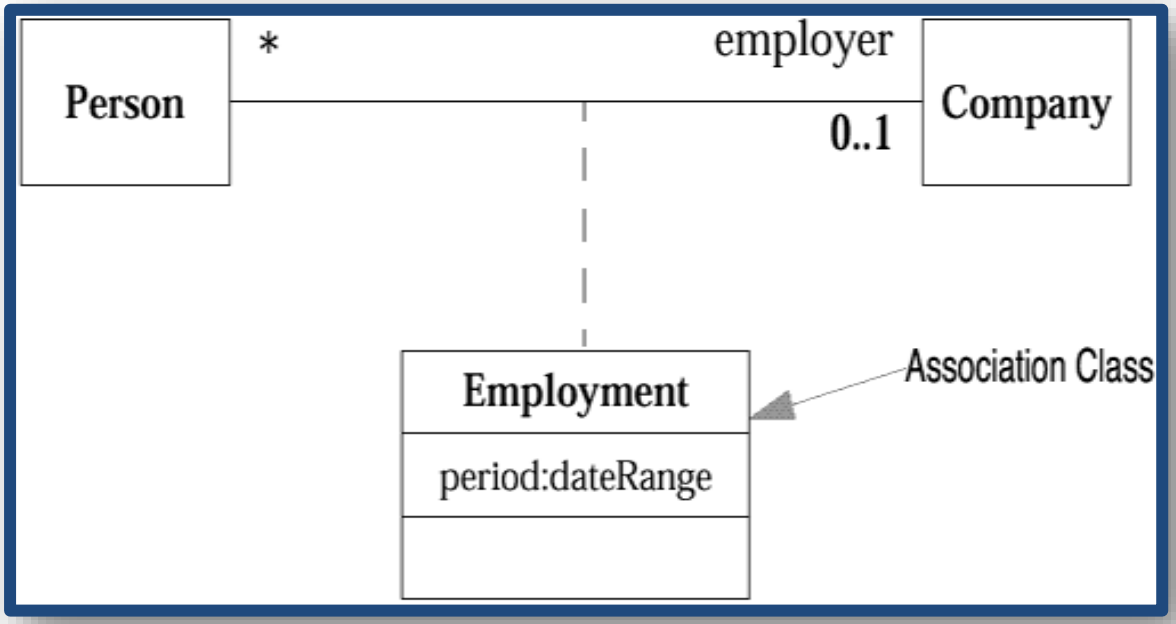


Image source : Google

Association vs Aggregation vs Composition

- owners feed pets, pets please owners (association)
- a tail is a part of both dogs and cats (aggregation / composition)
- a cat is a kind of pet (inheritance / generalization)

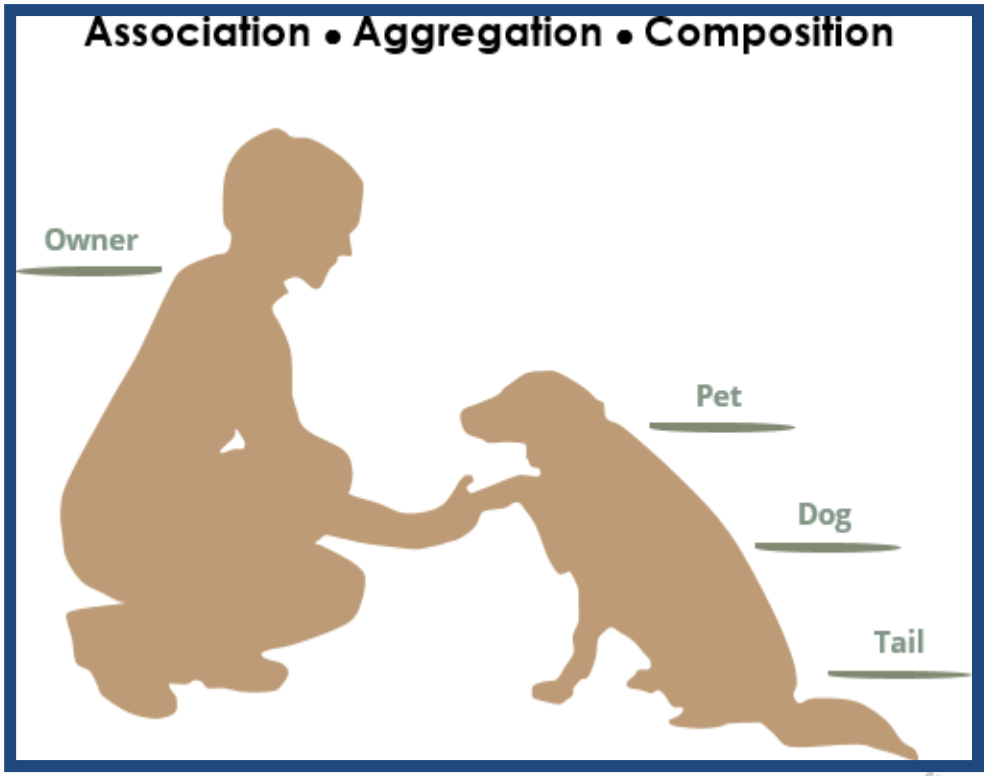


Image source : Google

Aggregation and Composition

Aggregation implies a relationship where the child can exist independently of the parent. Example: Class (parent) and Student (child). Delete the Class and the Students still exist.

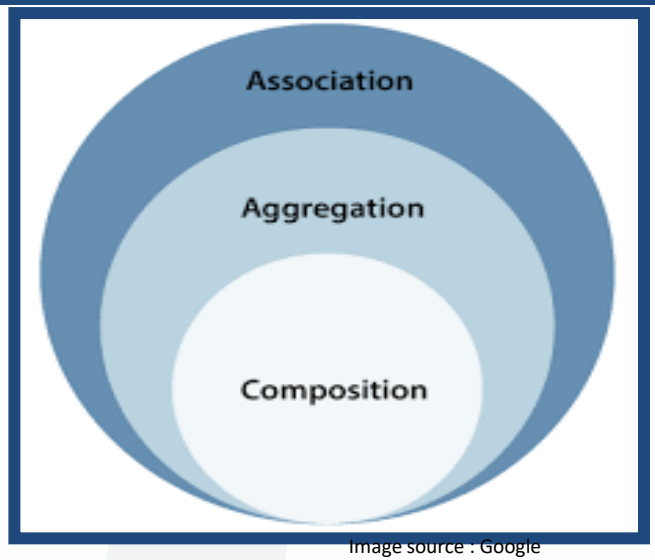
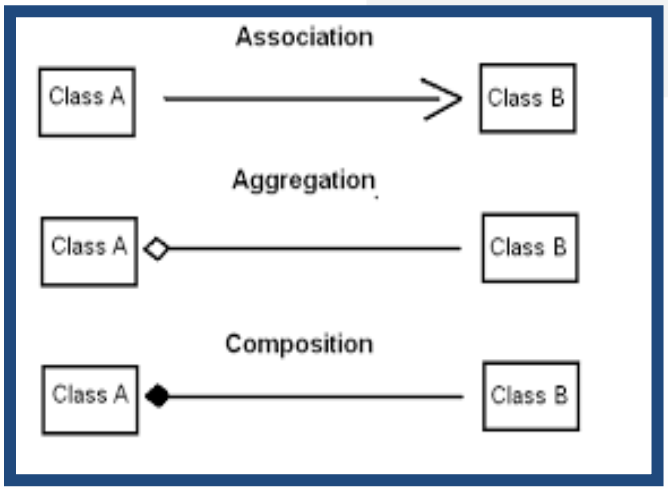


Image source : Google



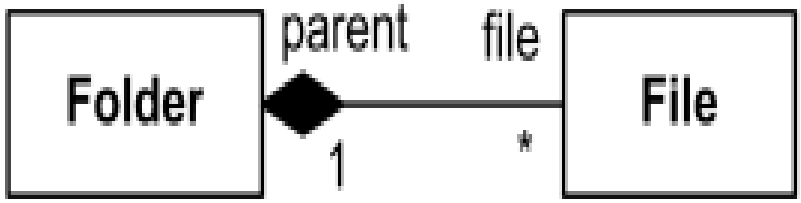
Composition implies a relationship where the child cannot exist independent of the parent. Example: House (parent) and Room (child). Rooms don't exist separate to a House.

Image source : Google

Composition

- Composite aggregation (composition) is a "strong" form of aggregation with the following characteristics:
- it is binary association,
- it is a whole/part relationship,
- a part could be included in at most one composite (whole) at a time, and
- if a composite (whole) is deleted, all of its composite parts are "normally" deleted with it.

Composition: Notation



Folder could contain many files, while each File has exactly one Folder parent.

If Folder is deleted, all contained Files are deleted as well.

Composition: Notation

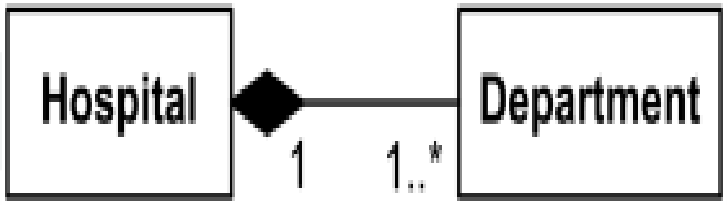


Image source : Google

Hospital has 1 or more Departments, and each Department belongs to exactly one Hospital. If Hospital is closed, so are all of its Departments.

Composition: Notation

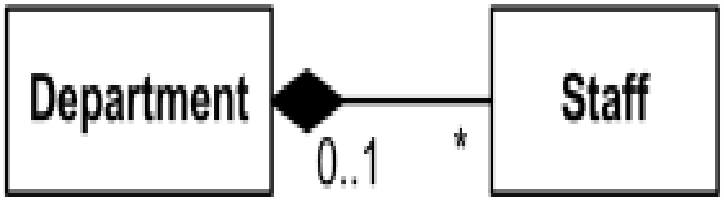


Image source : Google

Each Department has some Staff, and each Staff could be a member of one Department (or none). If Department is closed, its Staff is relieved (but excluding the "stand alone" Staff)

Abstract Class

- It is a class with an operation prototype, but not the implementation. It is also possible to have an abstract class with no operations declared inside of it.
- An abstract is useful for identifying the functionalities across the classes. Let us consider an example of an abstract class. Suppose we have an abstract class called as a motion with a method or an operation declared inside of it. The method declared inside the abstract class is called a move ().

Abstract Class

- This abstract class method can be used by any object such as a car, an animal, robot, etc. for changing the current position. It is efficient to use this abstract class method with an object because no implementation is provided for the given function. We can use it in any way for multiple objects.
- In UML, the abstract class has the same notation as that of the class. The only difference between a class and an abstract class is that the class name is strictly written in an italic font.
- An abstract class cannot be initialized or instantiated.

Abstract Class

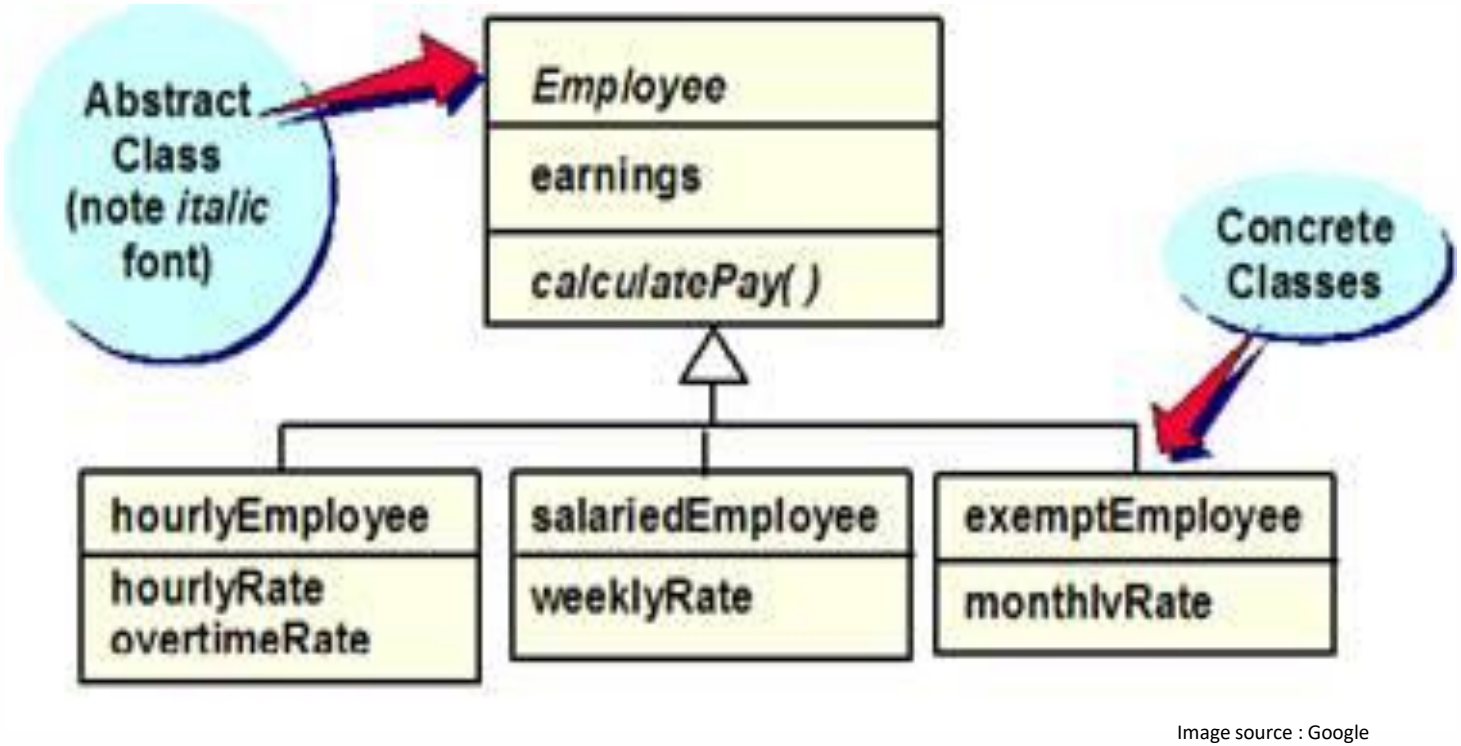


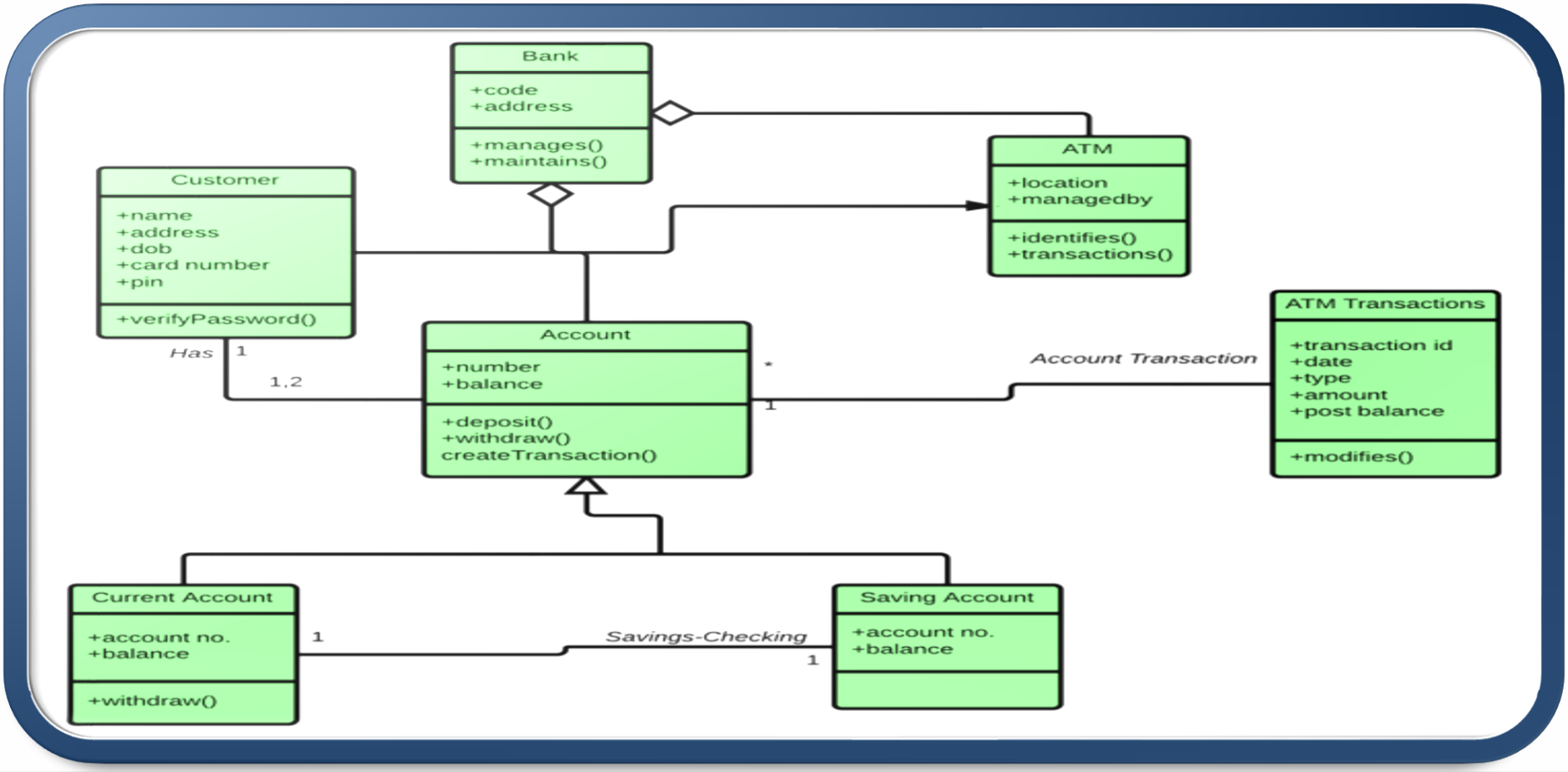
Image source : Google

Abstract Class

❖In summary these are the characteristics of abstract classes

1. Class must be declared as abstract.
2. Use abstract classes when you need to provide some functionality but also wish to defer some implementation.
3. In UML the method is shown in italics.
4. You will find that as you gain experience the concept of abstract classes provides an elegant solution to reuse and the delegation of responsibilities.

Example


Bank
+code
+address
+manages()
+maintains()
ATM
+location
+managedby
+identifies()
+transactions()
Customer
+name
+address
+dob
+card number
+pin
+verifyPassword()
Has
1
1,2
Account
+number
+balance
+deposit()
+withdraw()
createTransaction()
*
1
Account Transaction
ATM Transactions
+transaction id
+date
+type
+amount
+post balance
+modifies()
Current Account
+account no.
+balance
+withdraw()
1
Savings-Checking
1
Saving Account
+account no.
+balance

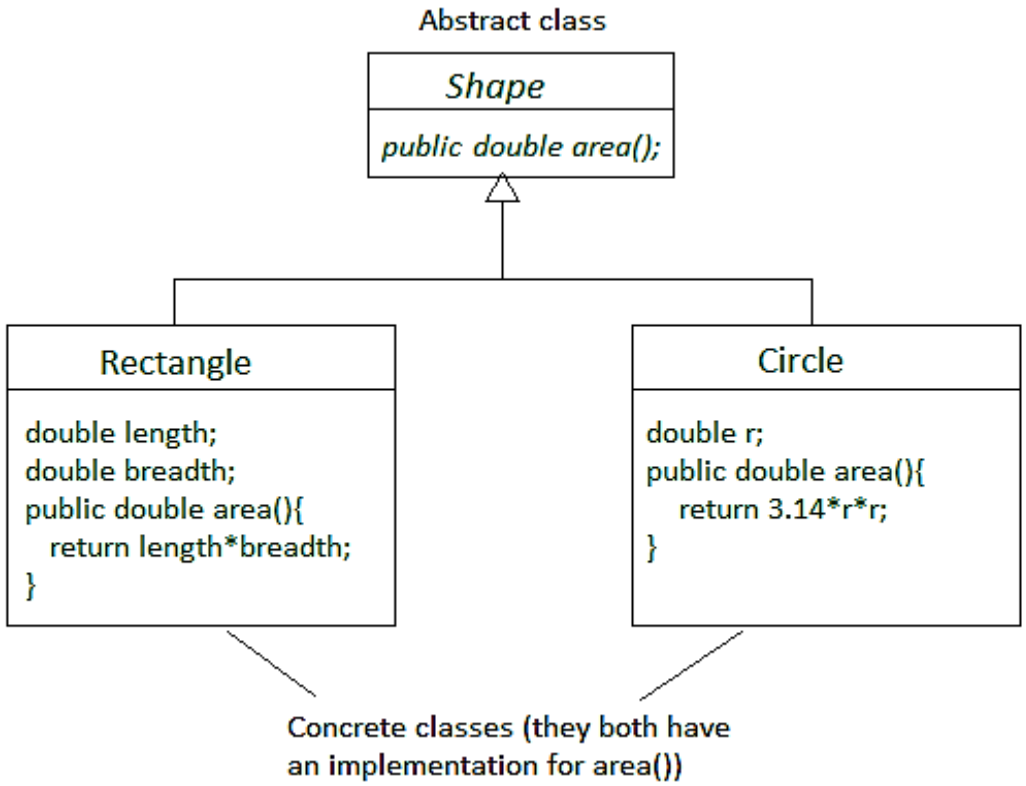
Concrete Class

Concrete is the *default*. This means that the class can have (direct) instances. In contrast, abstract means that a class cannot have its own (direct) instances.

❖In summary these are the characteristics of concrete classes

1. You can simply extend these.
2. You don't need to provide any additional information.
3. The bottom-level classes must be concrete, otherwise no instances will ever be created that exhibit the defined behavior!!
4. Concrete classes are the classes that you will have already encountered and worked with.

Concrete Class



Metadata

- "Metadata is "data about data". The term is ambiguous, as it is used for two fundamentally different concepts (types).
- *Structural metadata* is about the design and specification of data structures and is more properly called "data about the containers of data"; *descriptive metadata*, on the other hand, is about individual instances of application data, the data content. Metadata are traditionally found in the card catalogs of libraries.
- As information has become increasingly digital, metadata are also used to describe digital data using metadata standards specific to a particular discipline.
- By describing the contents and context of data files, the quality of the original data/files is greatly increased.
- For example, a webpage may include metadata specifying what language it is written in, what tools were used to create it, and where to go for more on the subject, allowing browsers to automatically improve the experience of users."

Metadata

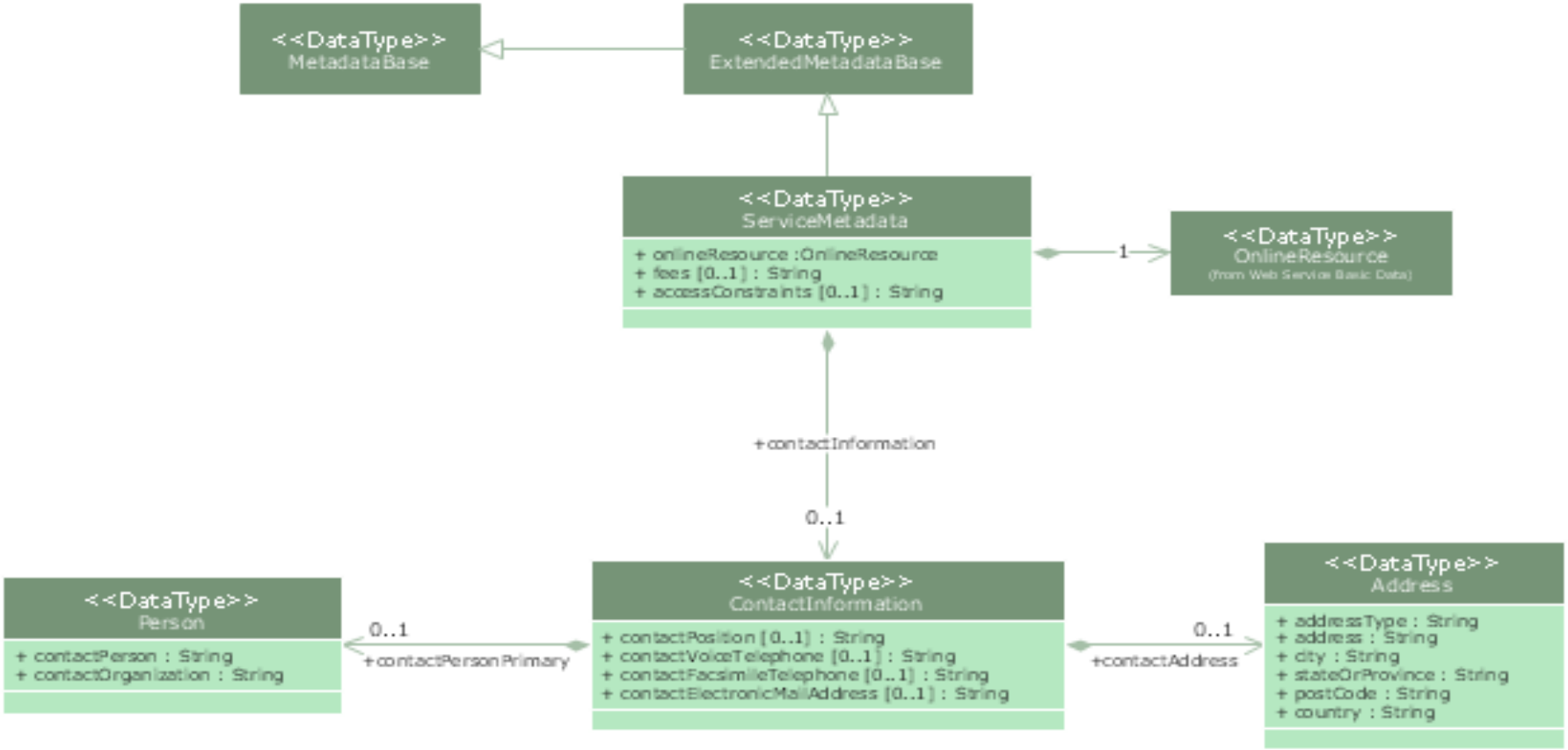


Image source : Google

Constraint

•A constraint is a package able 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.

•In general there are many possible kinds of owners for a constraint. Owning element must have access to the constrained elements to verify constraint. The owner of the constraint will determine when the constraint is evaluated. For example, operation can have pre-condition and/or a post-condition constraints.

•Constraint could have an optional name, though usually it is anonymous. A constraint is shown as a text string in curly braces according to the following syntax:

•*constraint* ::= '{' [***name*** ':'] ***Boolean-expression*** '}'

Constraint

•UML specification does not restrict languages which could be used to express constraint. Some examples of **constraint languages** are:

•OCL

•Java

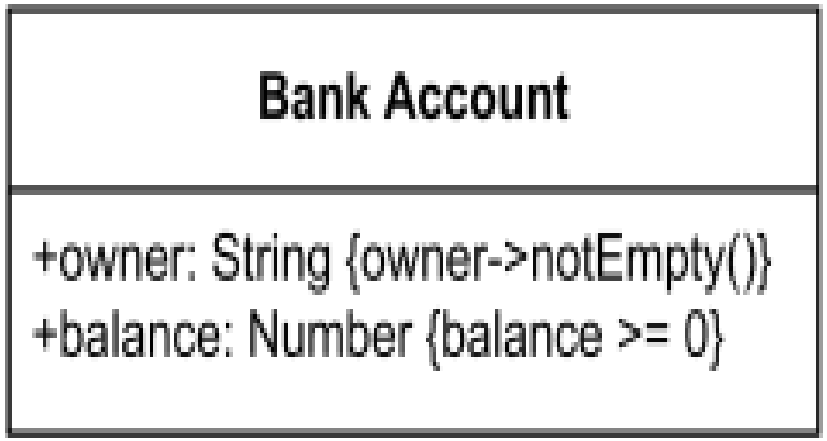
•some machine readable language

•natural language

Constraint

•OCL is a constraint language predefined in UML but if some UML tool is used to draw diagrams, any constraint language supported by that tool could be applied.

•For an element whose notation is a text string (such as a class attribute), the constraint string may follow the element text string in curly braces.



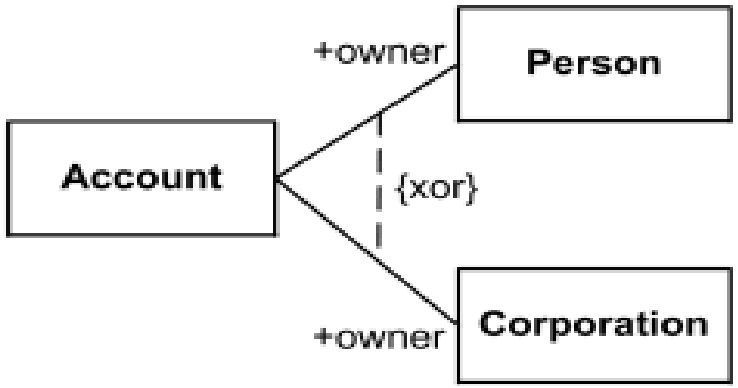
Bank account ***attribute constraints***
- non empty owner and positive balance.

Image source : Google

Constraint

•For a constraint that applies to a single element (such as a class or an association path), the constraint string may be placed near the symbol for the element, preferably near the name, if any. A UML tool must make it possible to determine the constrained element.

•For a constraint that applies to two elements (such as two classes or two associations), the constraint may be shown as a dashed line between the elements labeled by the constraint string in curly braces.



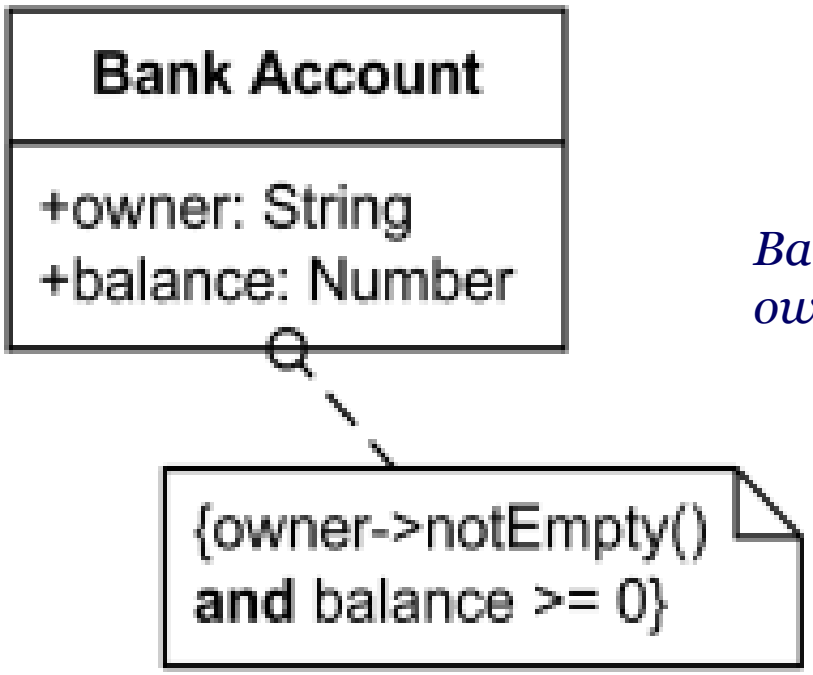
Account owner is either Person or Corporation,
***{xor}** is predefined UML constraint.*

Image source : Google

Constraint

- If the constraint is shown as a dashed line between two elements, then an arrowhead may be placed on one end. The direction of the arrow is relevant information within the constraint.
- The element at the tail of the arrow is mapped to the first position and the element at the head of the arrow is mapped to the second position in the constrainedElements collection.
- For three or more paths of the same kind (such as generalization paths or association paths), the constraint may be attached to a dashed line crossing all of the paths.
- The constraint string may be placed in a note symbol (same as used for comments) and attached to each of the symbols for the constrained elements by a dashed line.

Constraint



Bank account ***constraints*** *- non empty owner and positive balance.*

Image source : Google

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-2

State Modeling:

State Modeling

- State model describes the sequences of operations that occur in response to external stimuli.
- The state model consists of multiple state diagrams, one for each class with temporal behavior that is important to an application.
- The state diagram is a standard computer science concept that relates events and states.
- Events represent external stimuli and states represent values objects.

Elements of State Diagrams

The basic elements of state diagrams are

- **Events** – An event is an occurrence at a point in time
- **States** – the state in which the object finds itself at any moment
- **Transitions** – take the object from one state to another
- **Actions** – take place as a result of a transition

Events

An event is an occurrence at a point in time such as –

User presses left button
Indigo flight departs from Mumbai

An event happens instantaneously with regard to time scale of an application.

One event may logically precede or follow another, or the two events may be unrelated (concurrent; they have no effect on each other).

Types of Events

An event may be one of 3 types :

- Signal event
- Time event
- Change event

Signal Event

A signal is an explicit one-way transmission of information from one object to another.

An object sending a signal to another object may expect a reply, but the reply is a separate signal under the control of the second object, which may or may not choose to send it.

A signal event is the event of sending or receiving a signal (concern about receipt of a signal).

The difference between signal and signal event

a signal is a message between objects

a signal event is an occurrence in time.

Time Event

Time event is an event caused by the occurrence of an absolute time or the elapse of a time interval.

UML notation for an absolute time is the keyword **when** followed by a parenthesized expression involving time.

The notation for a time interval is the keyword after followed by a parenthesized expression that evaluates to a time duration.

when (date = Aug 1 , 2015)

Change Event

A change event occurs whenever a specified condition is met

- Event name is specified as keyword *when*
- Parameter list is a *Boolean expression*
- The event occurs when both of the following conditions are met, irrespective of the order when they happen
  - The expression evaluates to true
  - The object is in the required state

when (battery power < lower limit)

when (tire pressure < minimum pressure)

States

- State is a condition or situation during the life of an object within which it performs some activity, or waits for some events
- The objects in a class have a finite number of possible states.
- Each object can be in one state at a time.
- A state specifies the response of an object to input events.
- At any given point in time, the system is in one state.
- It will remain in this state until an event occurs that causes it to change state.

Event vs. States

Event represents points in time.
State represents intervals of time.

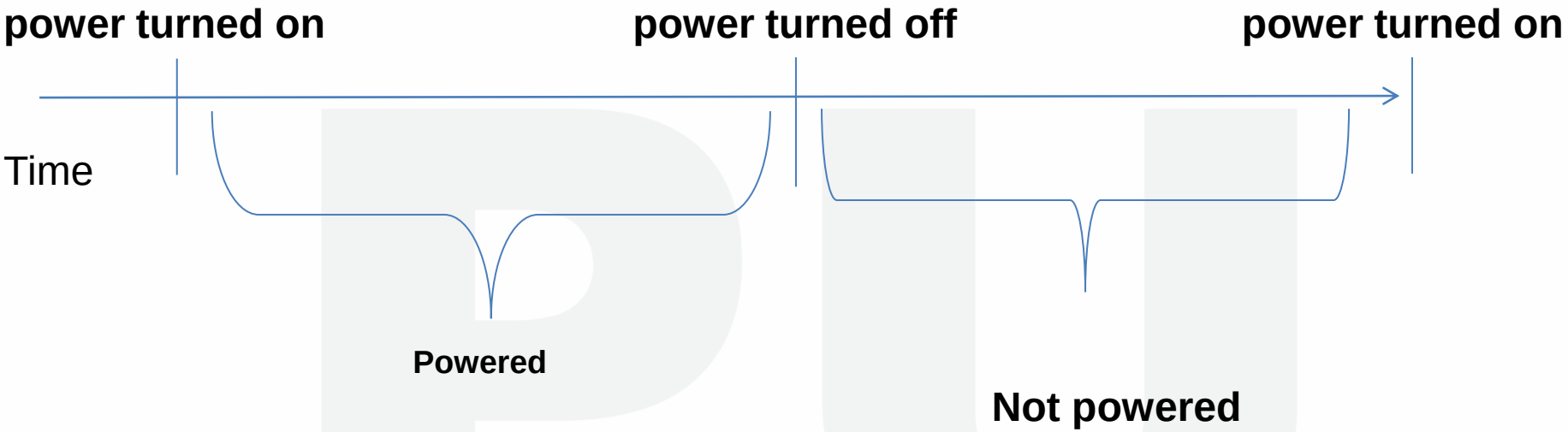
States

A **state** is when a system is:

Doing something – e.g., hea**ting** oven, mix**ing** ingredients, accelerat**ing** engine,

Waiting for something to happen – Wait**ing** for user to enter password, wait**ing** for sensor reading.

States

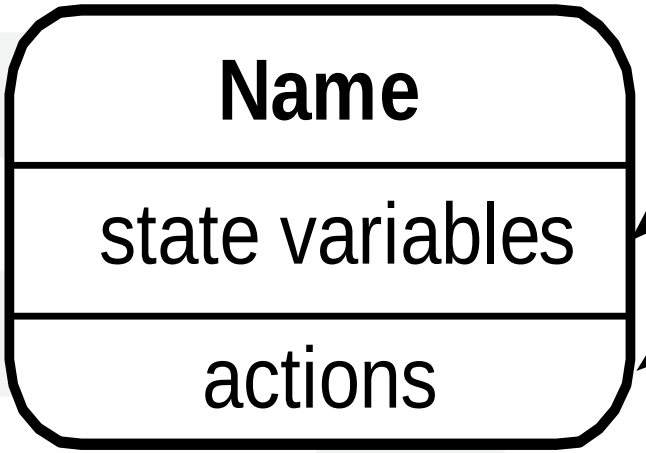


A state corresponds to the interval between two events received by an object.

The state of an object depends on past events.

Basic UML Syntax

- A state is drawn with a round box, with three compartments for
  - name
  - state variables (if any)
  - actions to be performed



sometimes left out when empty

Image source:https://slideplayer.com/slide/4964542/

Example: various characterizations of a state

State Alarm ringing on a watch

- **State** : *Alarm Ringing*
- **Description** : alarm on watch is ringing to indicate target time
- **Event sequence that produces the state** :

  setAlarm (targetTime)

  any sequence not including *clearAlarm*

  when (*currentTime = targetTime*)
- **Condition that characterrizes the state:**

  alarm = on, alarm set to *targetTime*, *targetTime <= currentTime <=targetTime+20 sec* , and no button has been pushed since *targetTime*
- **Events accepted in the state:**

event	response	next state
when (*currentTime = targetTime+20*)	*resetAlarm*	*normal*
buttonPushed(any button)	*resetAlarm*	*normal*

Image source:object oriented modeling and design with UML ,michael r blaha ,james r Rumbaugh

Transitions

- A transition is a relationship between two states indicating that an object in the first state will enter the second state.
- A transition is an instantaneous change from one state to another.
- The transition is said to fire upon the change from the source state to target state.
- A guard condition must be true in order for a transition to occur.
- A guard condition is checked only once, at the time the event occurs, and the transition fires if the condition is true.

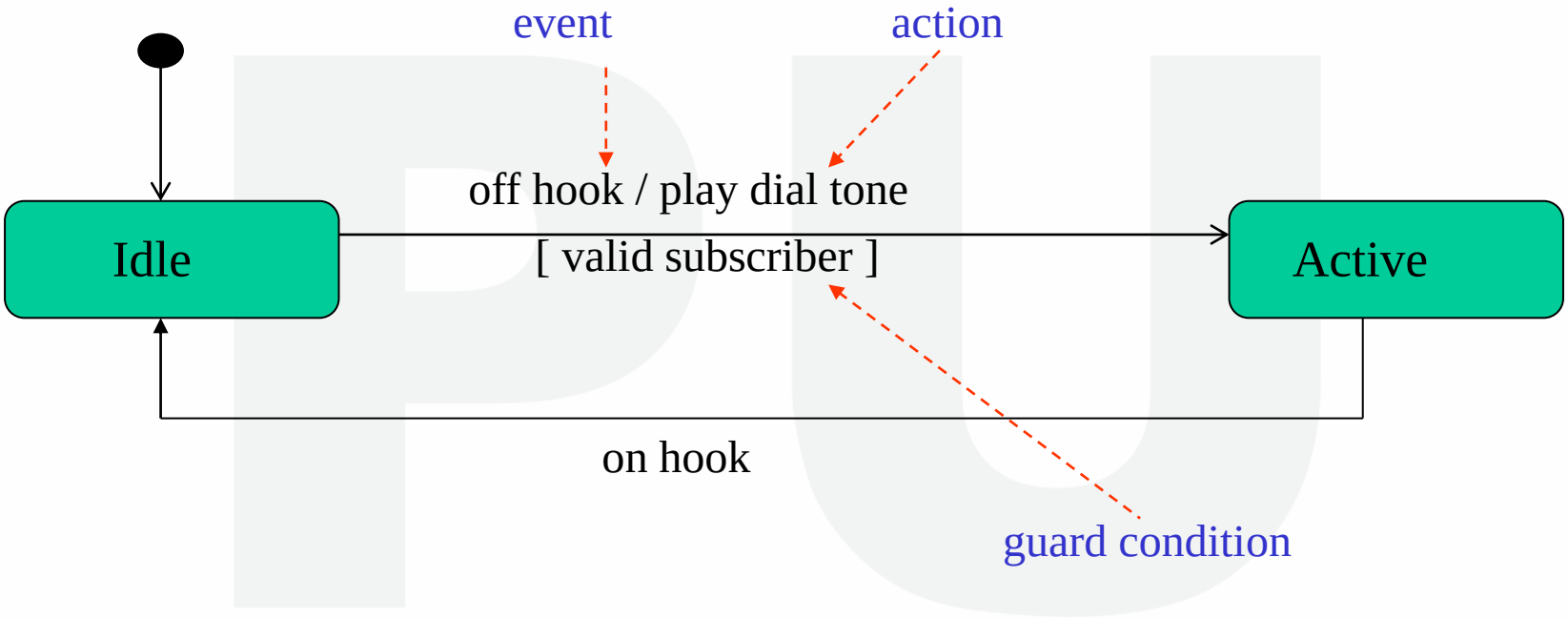
Transitions

A directed relationship between two states.

Contains four parts

- **Source state** - current state before transition fires.
- **Event trigger** - external stimulus that has the potential to cause a transition to fire.
- **Guard condition** - a condition that must be satisfied before a transition can fire.
- **Target state** - new state after transition fires.

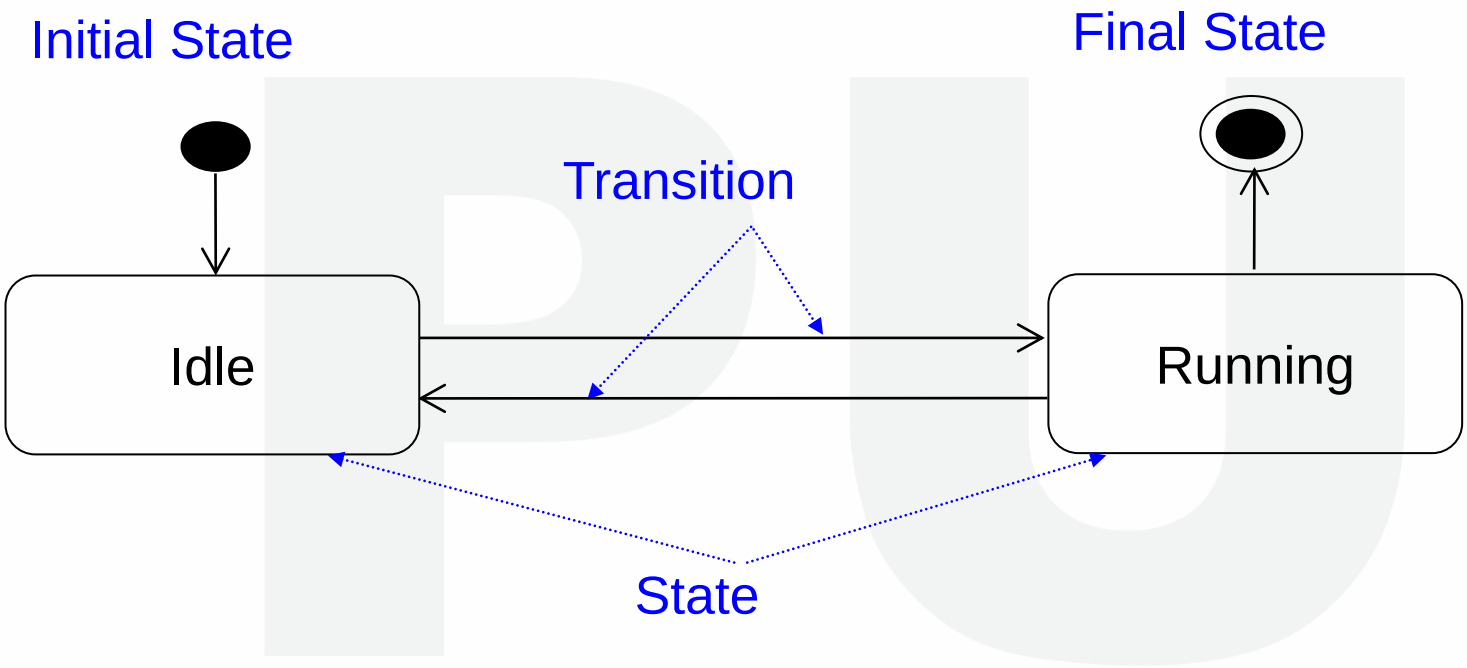
Example


event
action
off hook / play dial tone
[valid subscriber]
Idle
Active
on hook
guard condition

Basic UML Syntax

- A transition is drawn with an arrow, possibly labeled with
  - event causing the transaction
  - guard condition
  - Action to perform

State Diagram

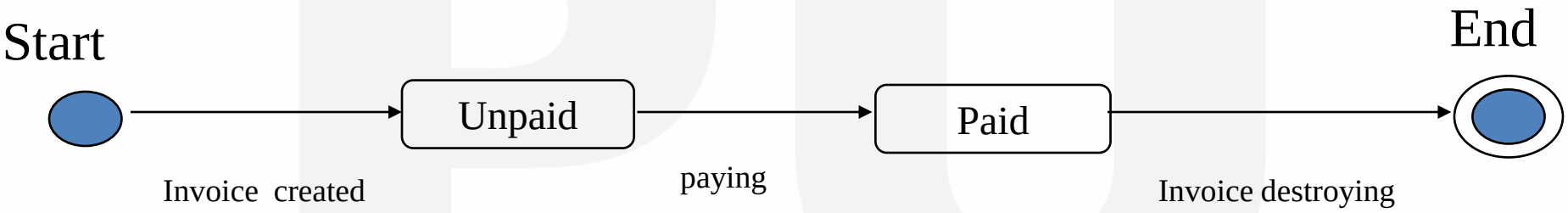


Parul® University

DIGITAL LEARNING CONTENT

Billing Example

State Diagrams show the sequences of states an object goes through during its life cycle in response to stimuli, together with its responses and actions; an abstraction of all possible behaviors.



Actions

- **Action**
  - is an executable atomic computation
  - includes operation calls, the creation or destruction of another object, or the sending of a signal to an object
  - associated with transitions and during which an action is not interruptible -- e.g., entry, exit

Predefined Action Labels

“entry/”

- identifies an action, specified by the corresponding action expression, which is performed upon entry to the state (entry action)

“exit/”

- identifies an action, specified by the corresponding action expression, that is performed upon exit from the state (exit action)

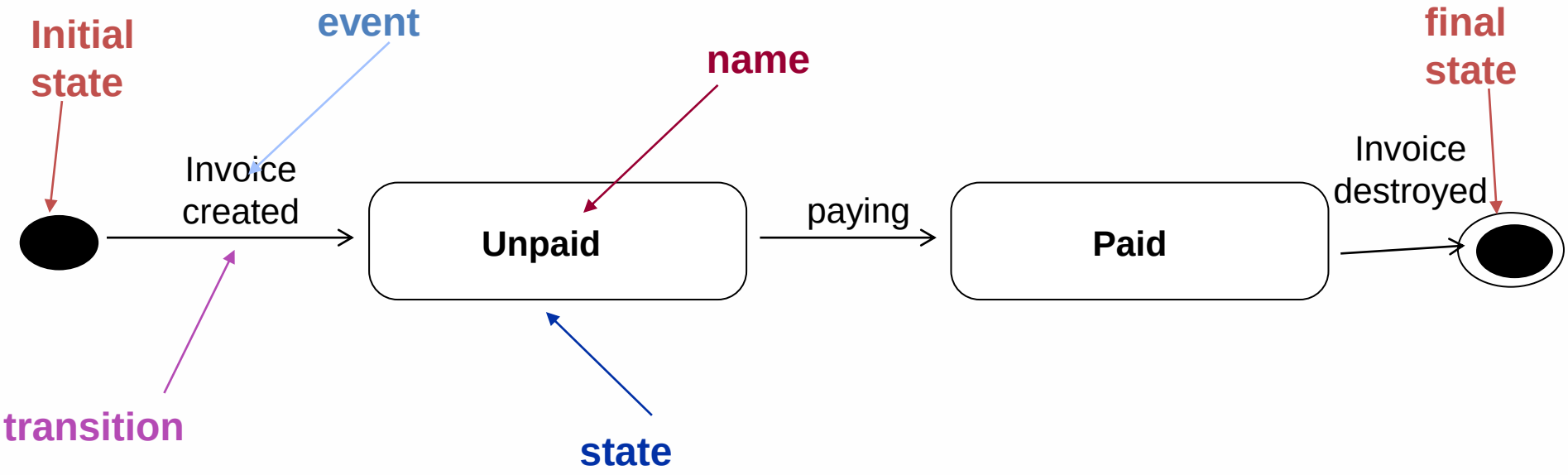
“do/”

- identifies an ongoing activity (“do activity”) that is performed as long as the modeled element is in the state or until the computation specified by the action expression is completed (the latter may result in a completion event being generated).

“include/”

- is used to identify a submachine invocation. The action expression contains the name of the submachine that is to be invoked.

State Diagrams notation



Example State Diagram for a washing machine.

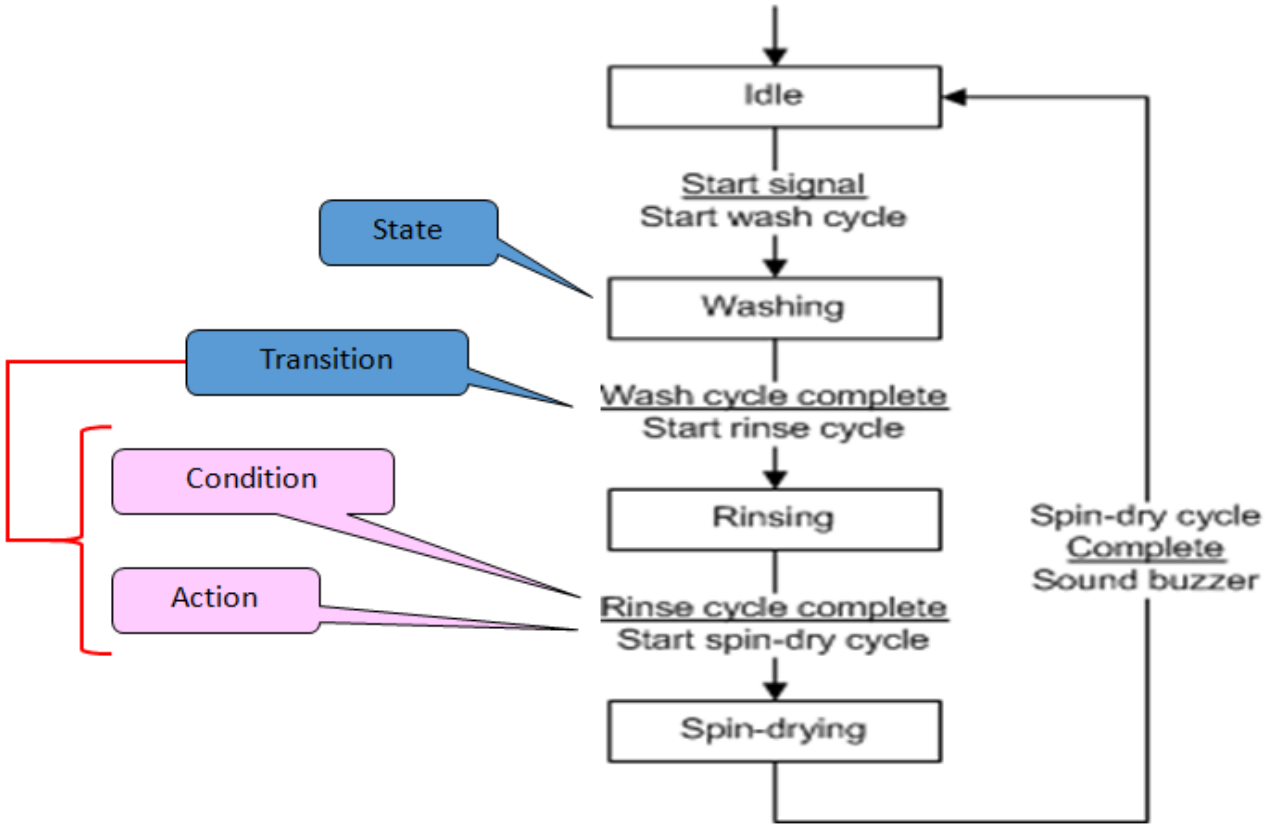


Image source: https://slideplayer.com/slide/6003291/

Seminar Registration

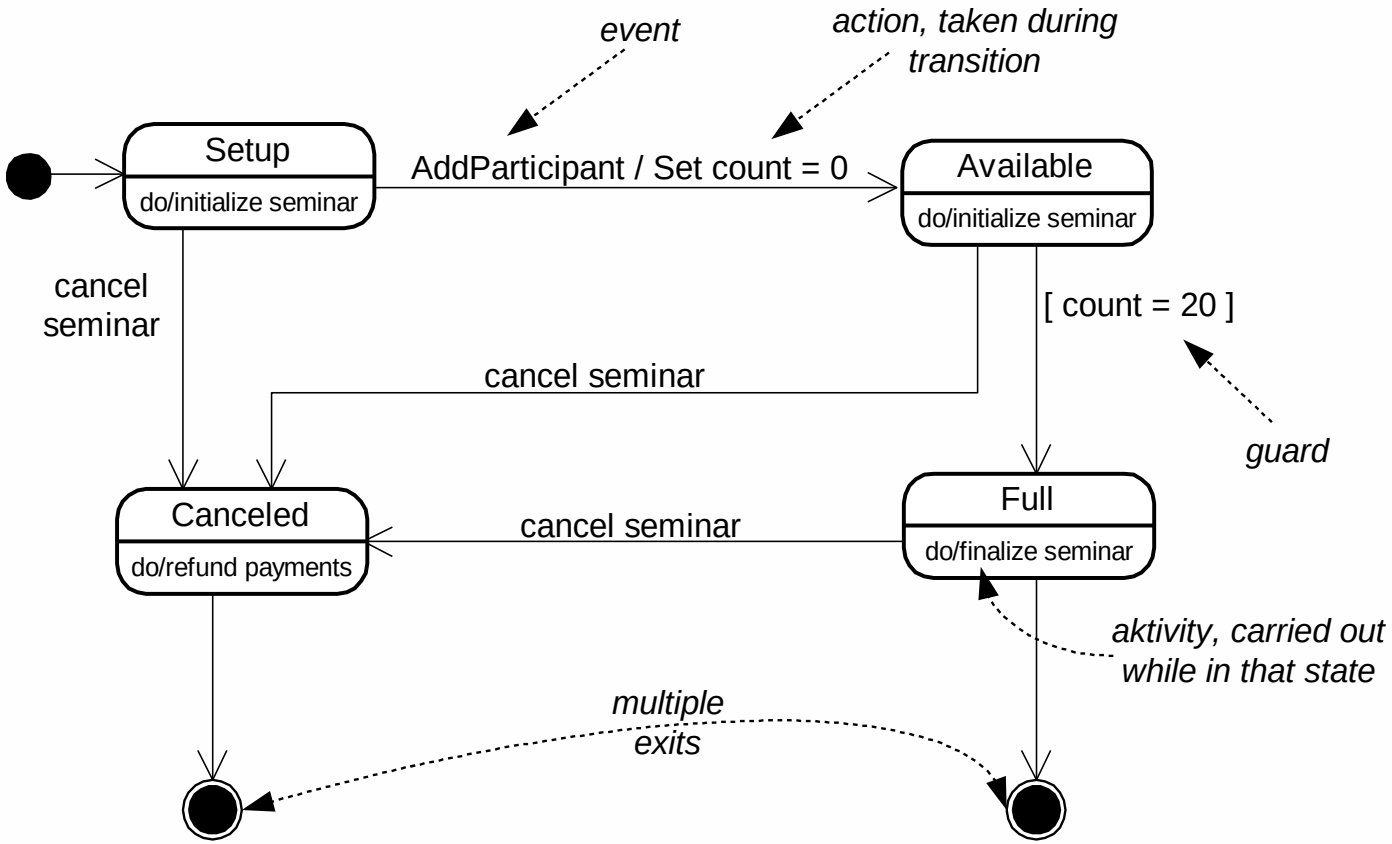


Image source:https://slideplayer.com/slide/4964542/

A **condition** is typically some kinc
event, e.g.:

- Signal
- Arrival of an object (data/material)

An **action** is the appropriate **out**
or **response** to the event, e.g.:

- Signal or message
- Transfer of an object,
- Calculation.

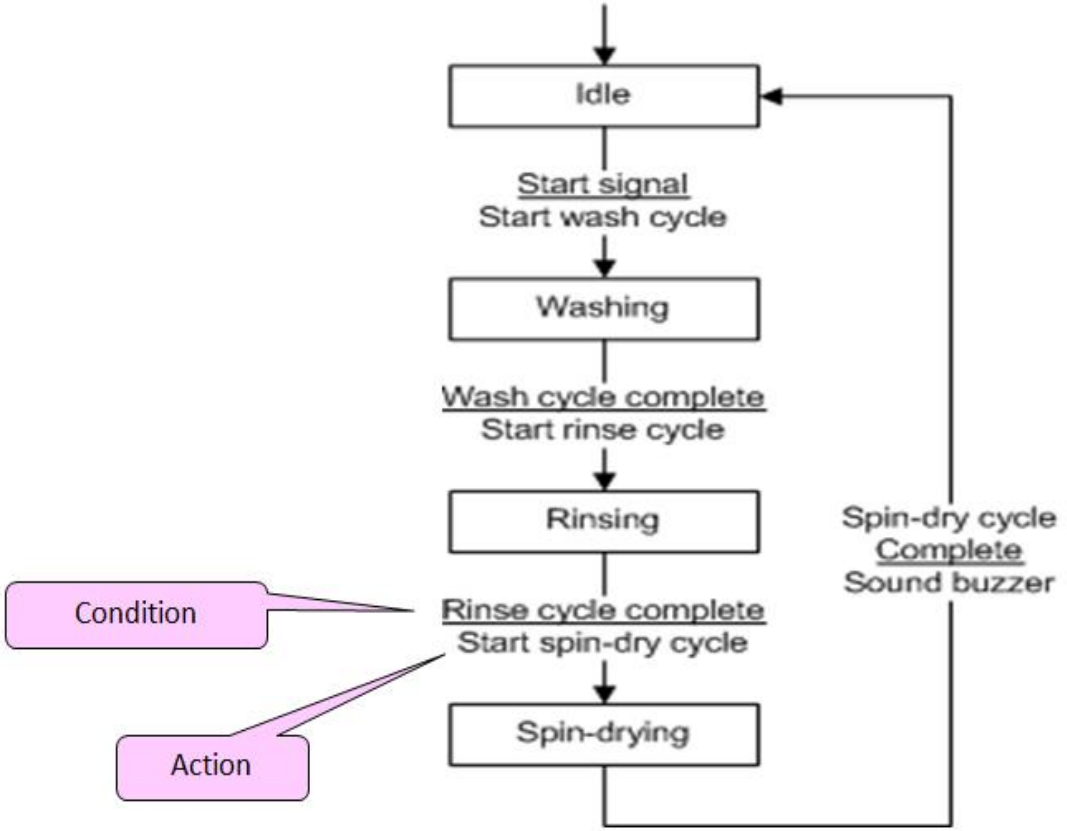
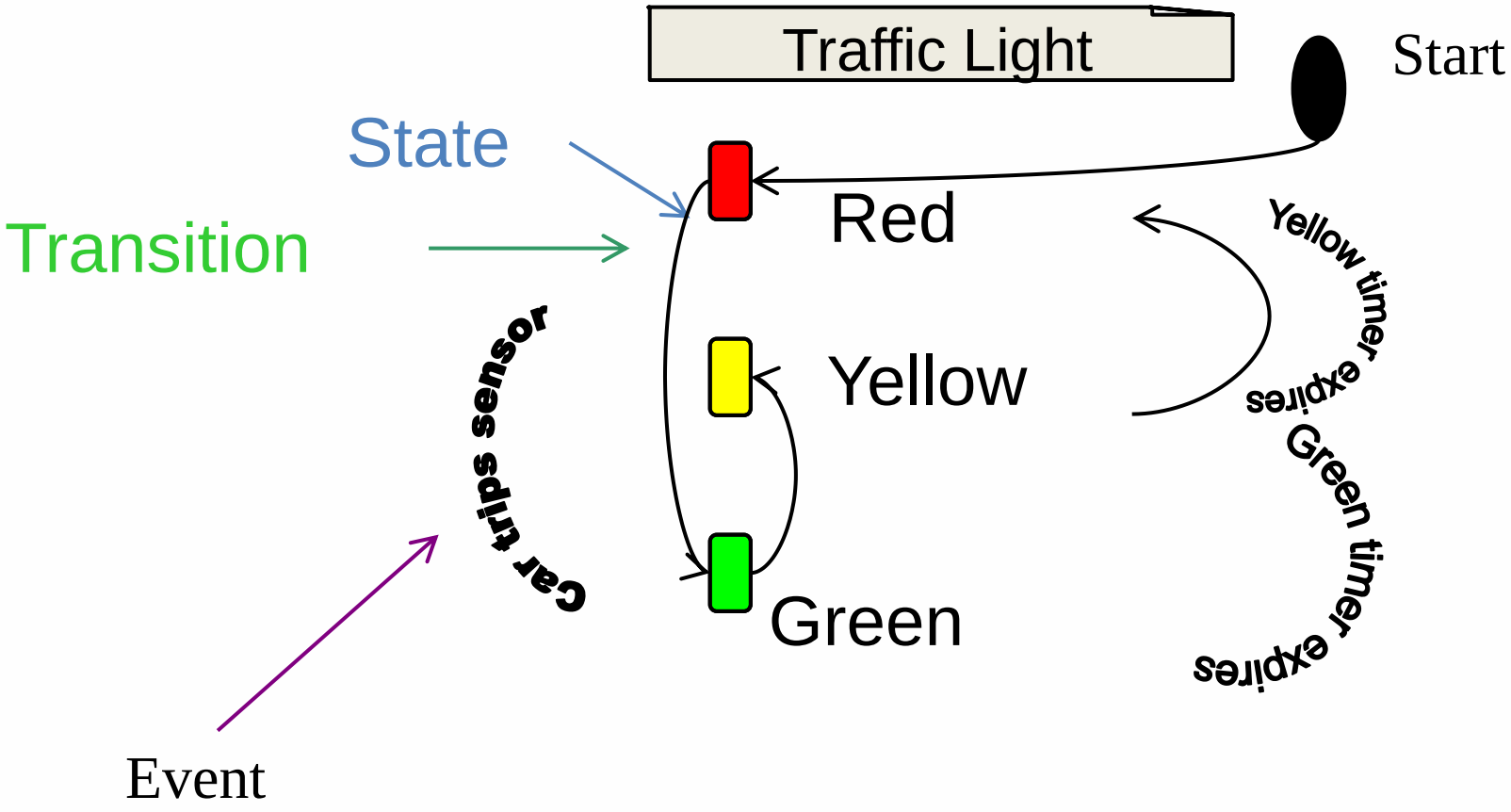


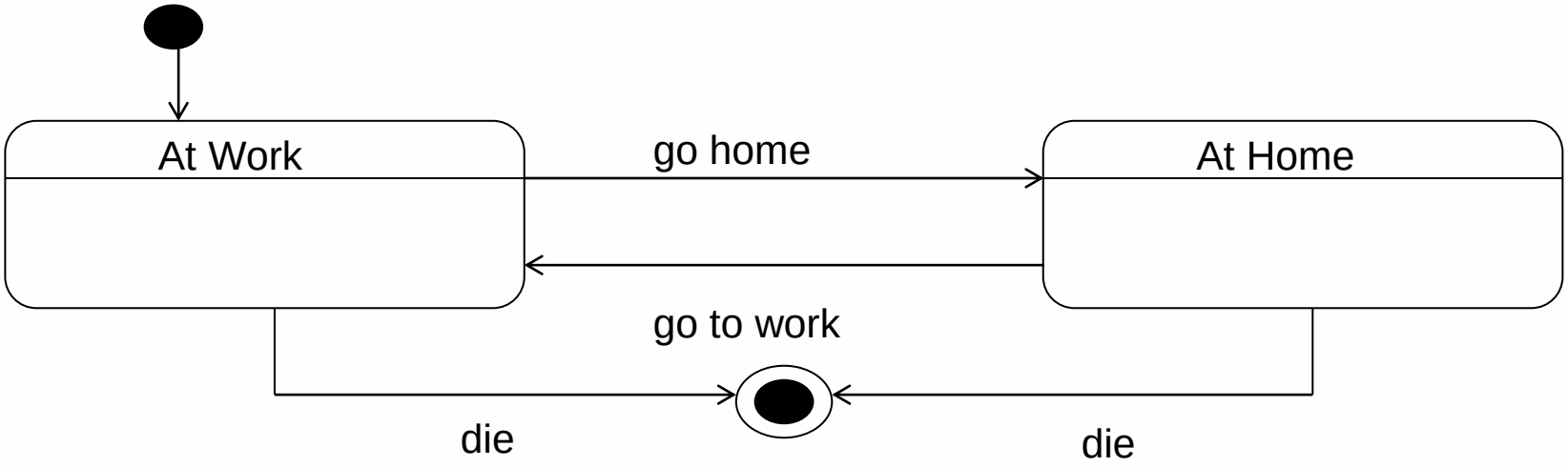
Image source: https://slideplayer.com/slide/6003291/

State Diagrams (Traffic light example)

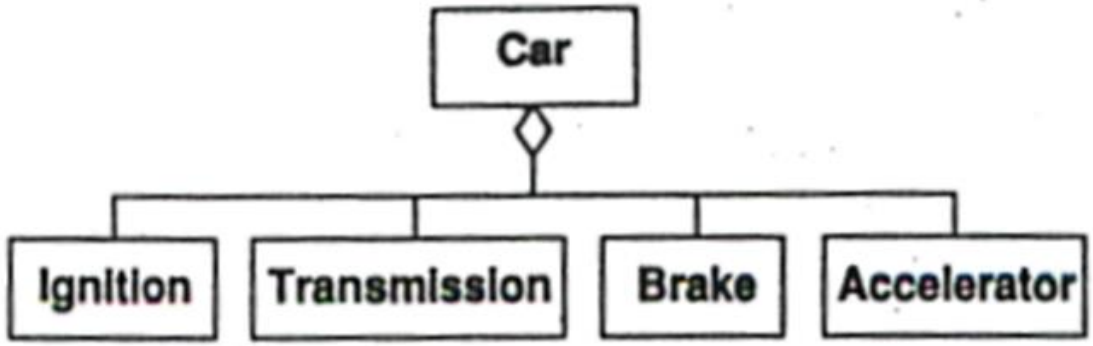


Initial and Final States

- An example:



State Diagrams



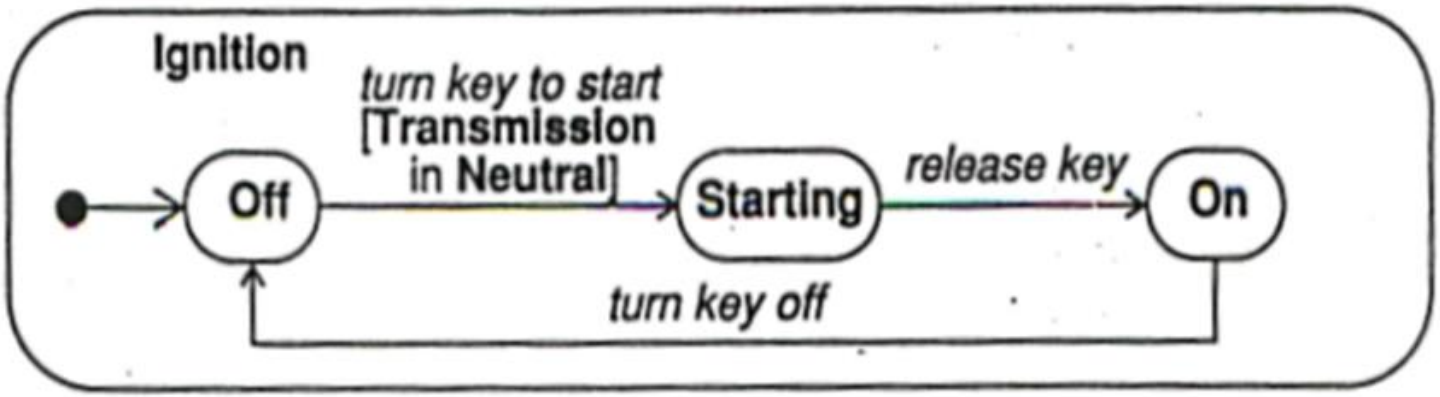
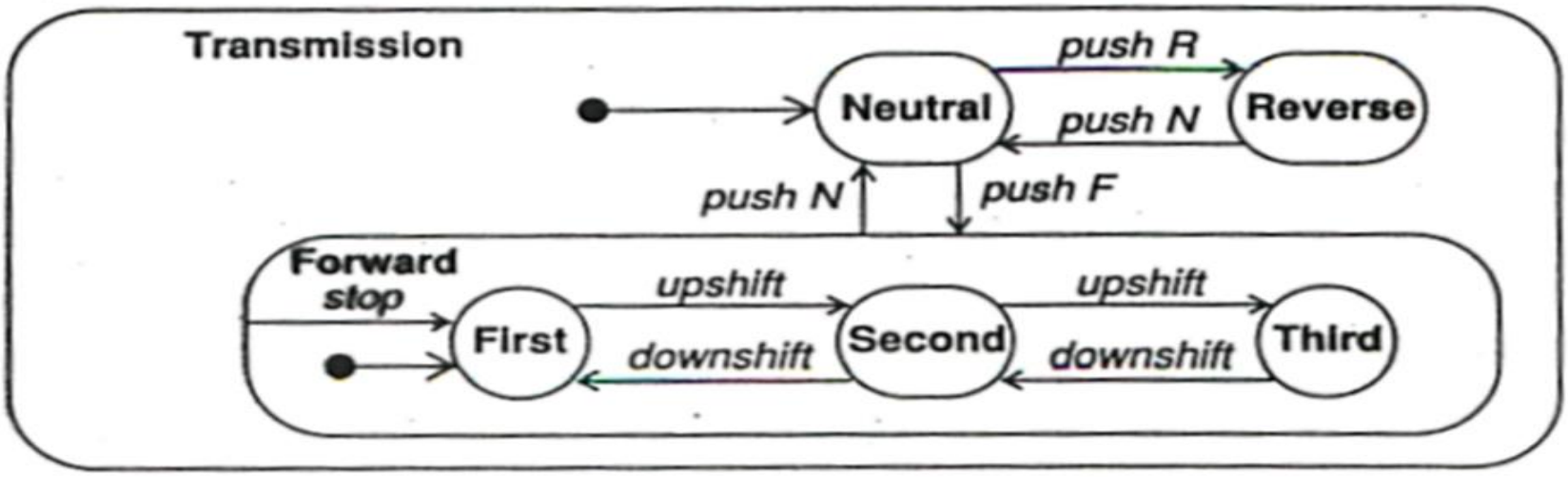
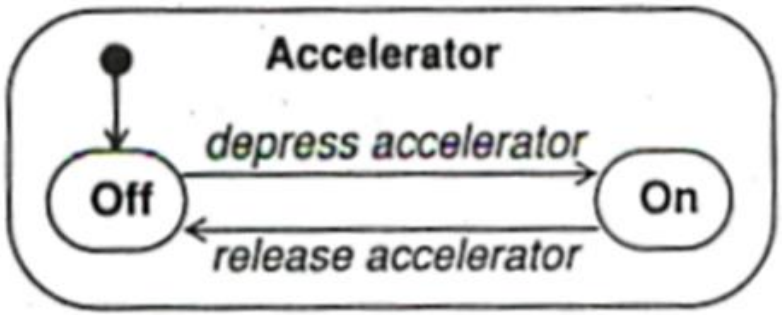


Image source:object oriented modeling and design with UML ,michael r blaha ,james r Rumbaugh

State Diagrams





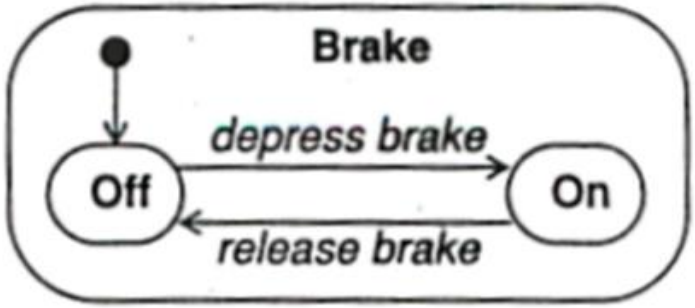
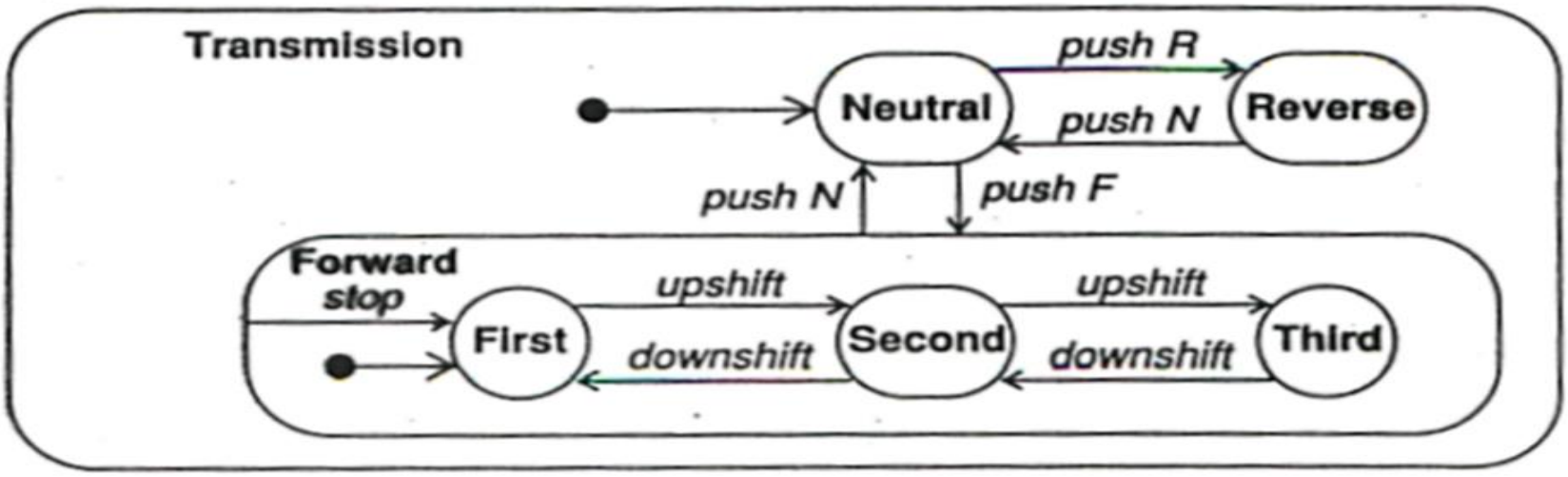
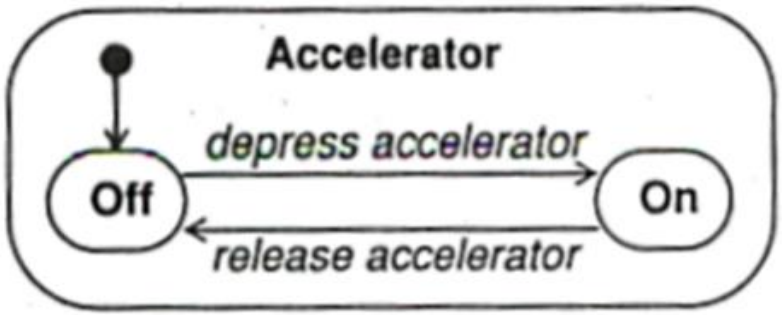


Image source:object oriented modeling and design with UML ,michael r blaha ,james r Rumbaugh

State Diagrams





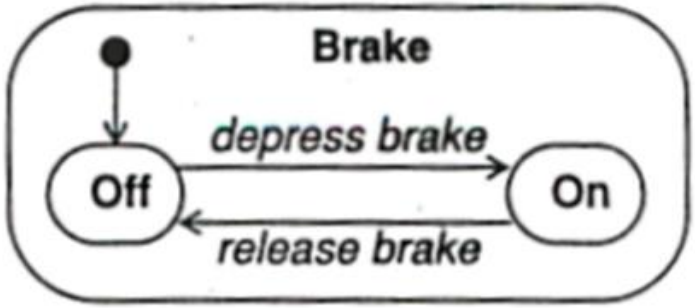
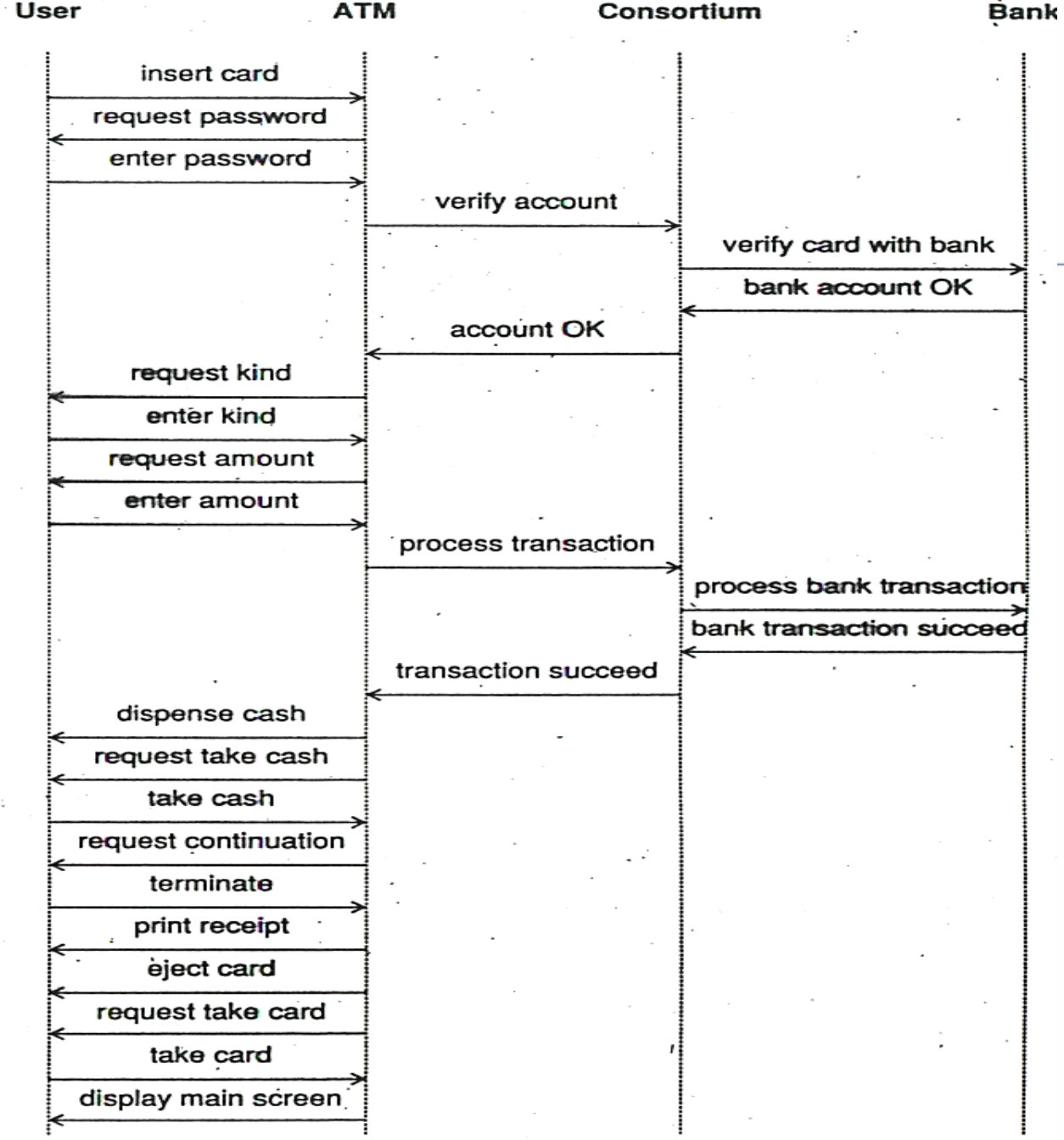


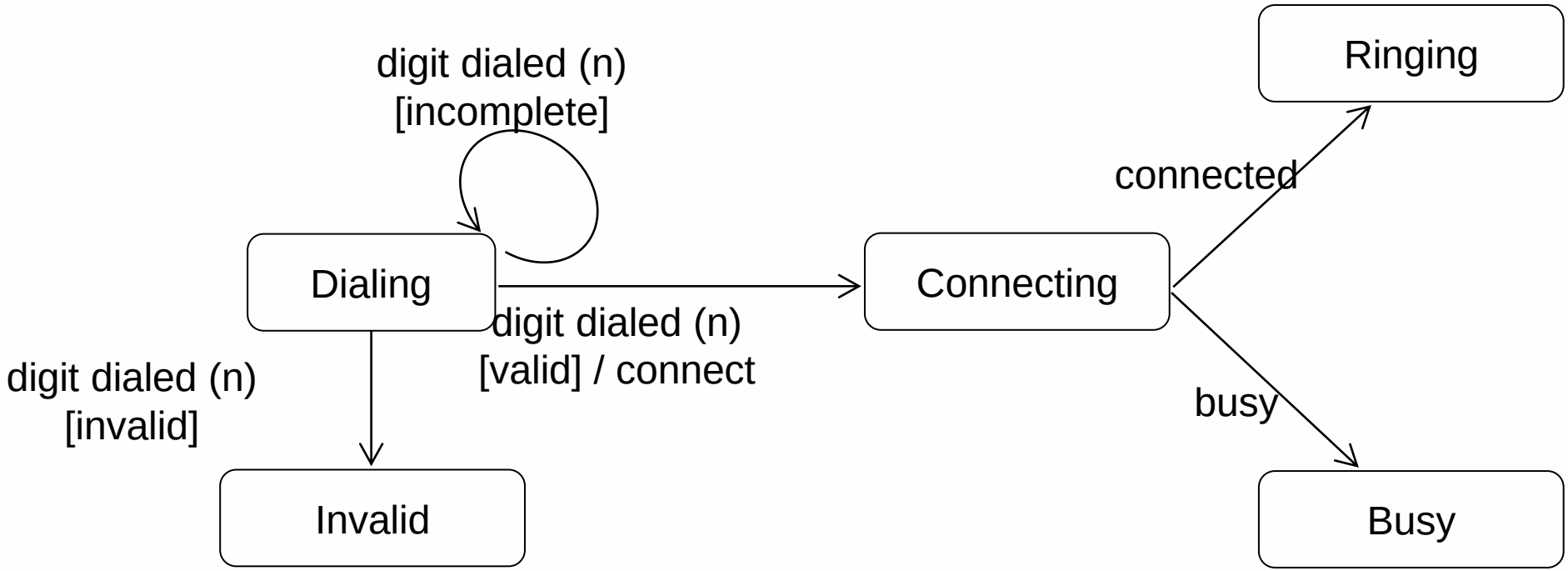
Image source:object oriented modeling and design with UML ,michael r blaha ,james r Rumbaugh



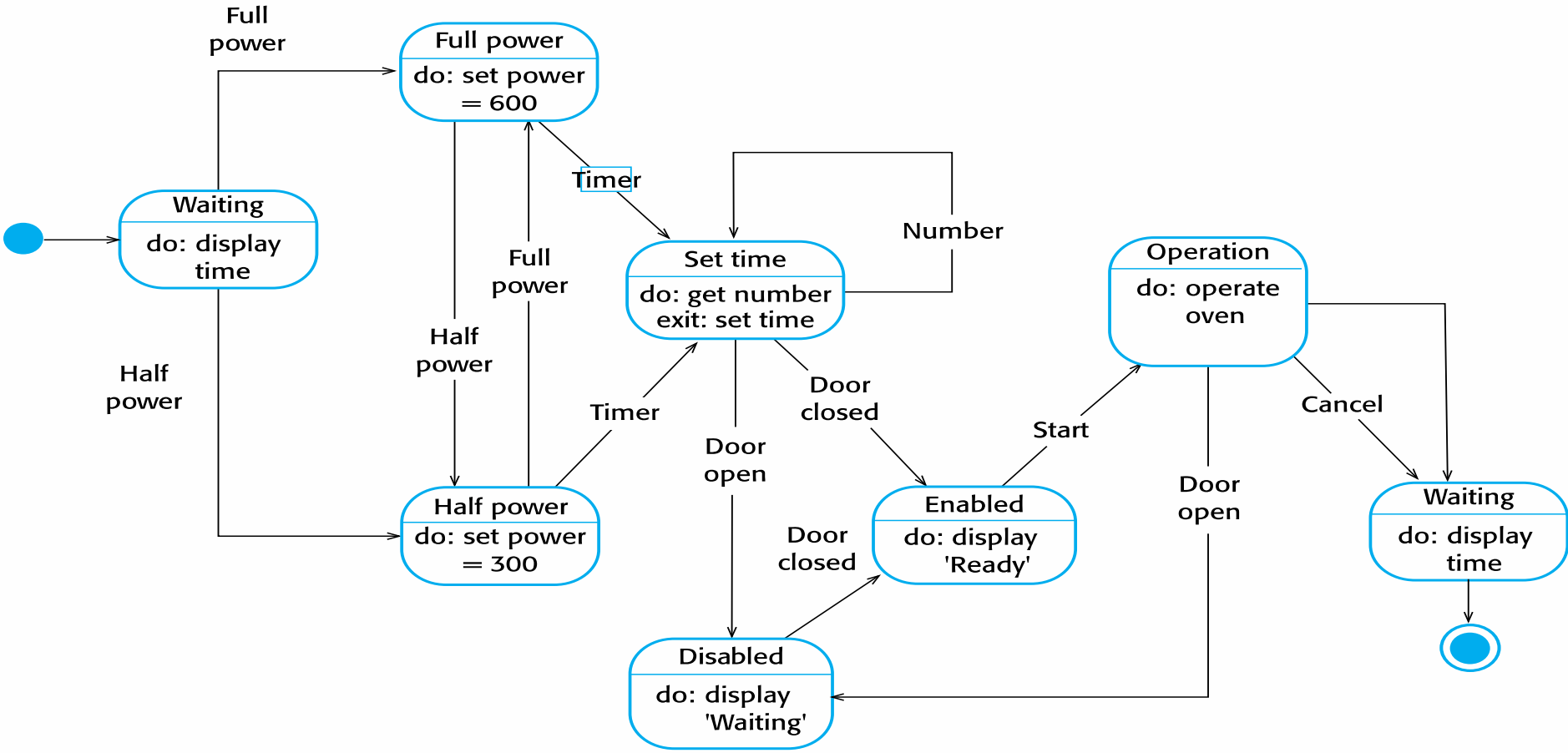
Event Trace for an ATM Transaction

Image source:object oriented modeling and design with UML ,michael r blaha ,james r Rumbaugh

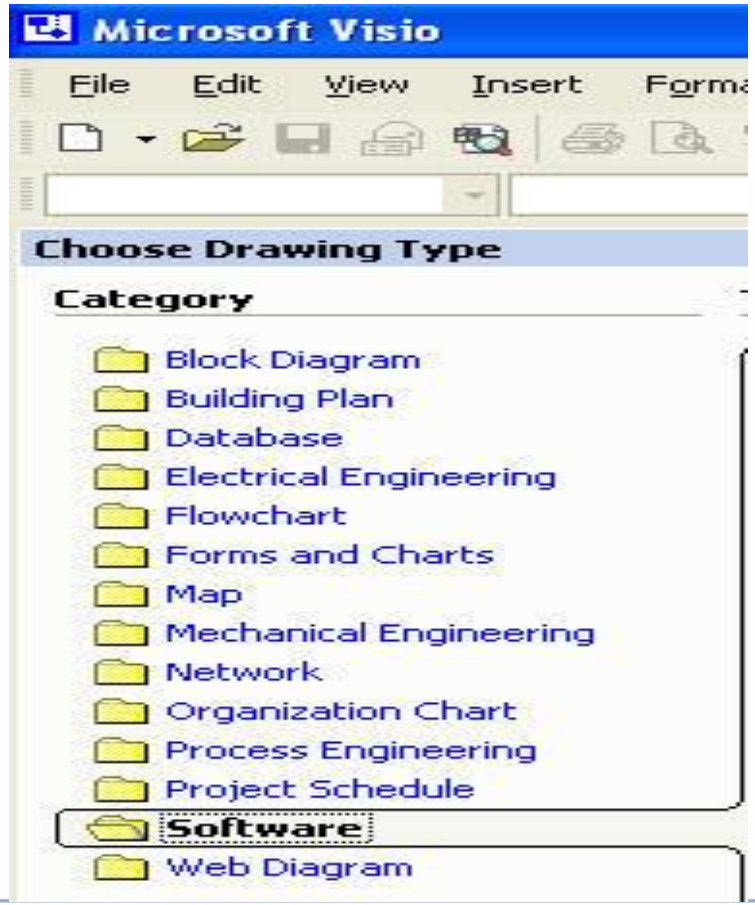
Phone Example



State diagram of a microwave oven



Using Microsoft Visio



Visio can be used to draw UML diagrams

It is component of Microsoft Office

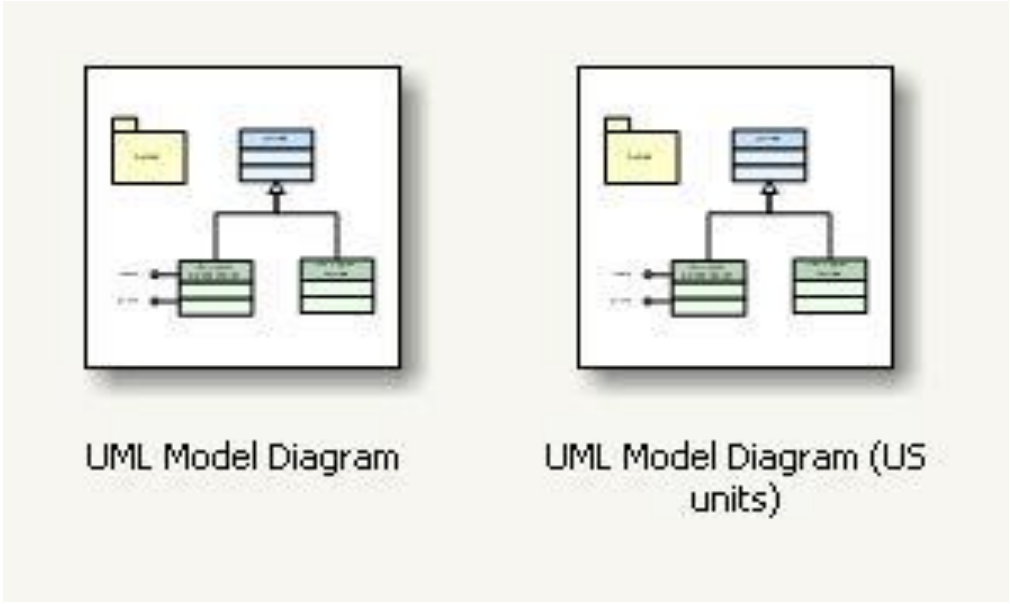


Image source: Microsoft Visio

Parul® University

DIGITAL LEARNING CONTENT

Using Microsoft Visio (continued)

Available Sets of Shapes in the UML Collection

- Activity Diagrams
- Collaboration Diagrams
- Components
- Deployment Diagrams
- Sequence Diagrams
- State Diagrams (State charts)
- Static Structures (shown) – include Packages and Classes
- Use Case Diagrams

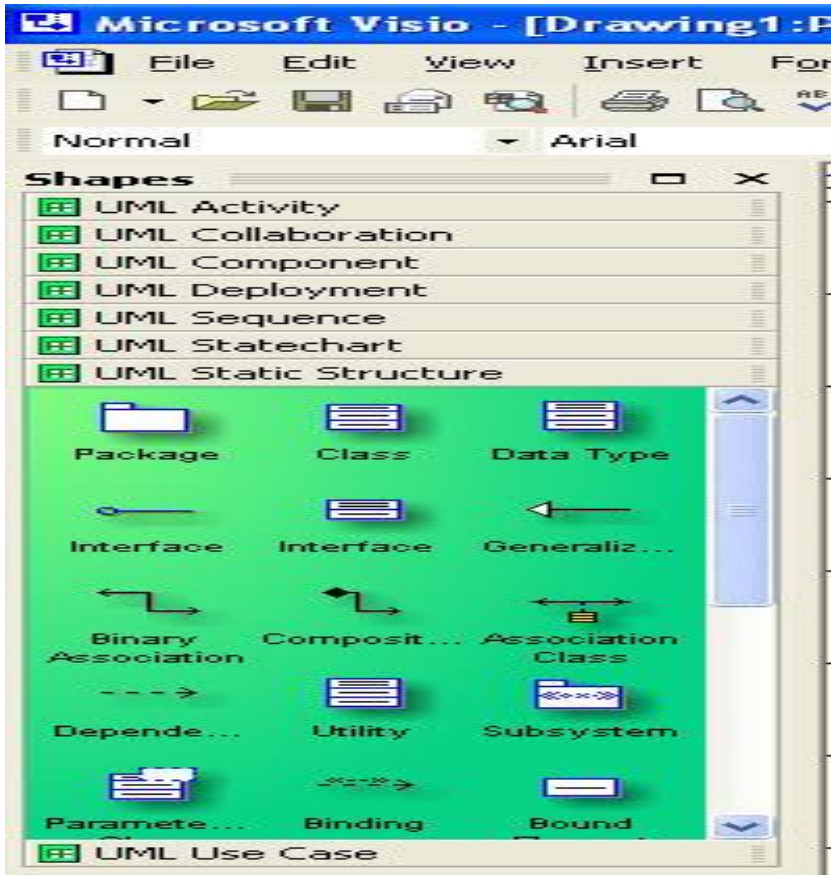


Image source: Microsoft Visio

Parul®
University

References

- https://softwarekno.blogspot.com/2016/09/event-driven-modeling.html
- https://slideplayer.com/slide/6003291/
- Object –Oriented modeling and Design with UML-Michael R. Blaha, James r Rumbaugh

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in