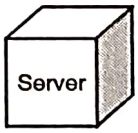
2.1 Advanced Object and Class

GQ. Explain the following terms with respect to class modeling :
(a) Metadata (b) Reification (c) Constraints (d) Derived data

- Objects and classes are classified on the basis of their functionality in a particular application.
- Basically, there are four types of objects: Control object, Entity object, Boundary object and application logic object.
- ***Control objects*** are the objects which act as a controller in overall collection of objects. They are also known as coordinator objects.
- ***Entity objects*** envelopes information and offers an access to the stored information.
- The objects that are dedicated for communication and coordination with external environment or outside world are known as ***Boundary objects***.
- Furthermore, Boundary objects are categorized into three types: Proxy object, User interaction object and Device I/O boundary object. Proxy object communicates with a peripheral subsystem. User interaction object communicates and coordinates with end user (human being). Device I/O boundary object handles input and output received from hardware I/O device.
- The detailed application logic of a system is comprised by ***Application logic object***.
- Classes are the important building block of an object-oriented system. Beyond the characteristics and features of a simple class, advanced classes have a number of advanced properties.
- A special class in Unified Modeling Language (UML) which is even more fundamental and dedicated for detailed description of behavioral and structural properties of a set of objects are considered as an advanced class.
- Advanced class is a best type of classifier in UML. Different types of classifiers are offered by UML in order to support designers to design an archetype n object oriented modeling.
- Following is the list of classifiers offered by UML.

Table 2.1.1 : Classifiers in UML

Classifier	Description	UML Notation
Node	A stationary element having memory and processor along with it and it denotes the computational resource.	
Component	A static element that offers the detailing of a group of interfaces. It is a expendable part of a system.	
Data type	A type whose values do not have any identity like built in types (string, char) and enumeration types (Boolean).	

Classifier	Description	UML Notation
Interface	A set of operations required to specify a service of a component.	
Use case	An explanation of a group of activities performed by a software system in some proper sequence.	
Signal	Asynchronous communication between components can be represented with the help of a signal.	

- We can identify and state the visibility of the attributes of a classifier. Three levels of visibility are :
  - **Public :** any external classifier can make use of the attribute. It is shown by symbol '+'.
  - **Private :** the classifier itself only can make use of the attribute. Symbol used is '–'.
  - **Protected :** any descendent of the classifier may use the attribute and presented with a symbol '#'.
- Fig. 2.1.1 shows a combination of three levels of visibility.

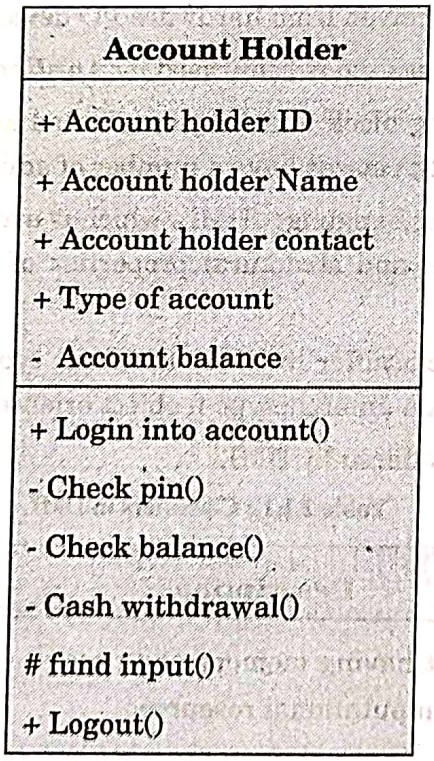


Fig. 2.1.1 : Three levels of Visibility

- In case of an advanced class, we can specify multiplicity, scope and visibility of each and every attribute.

Guidelines for designing a classifier

- A classifier should be loosely coupled and should possess both behavioral and structural aspects.
- It should be unambiguous in semantics.
- For better understanding, only important characteristics (attributes) should be mentioned within a classifier.

2.1.1 Association Ends and n-ary Association

GQ. What so you mean by Association ends? Explain n-ary association in short.

- Association Ends are simply nothing but the endpoints of an association relationship. ***Association*** is a structural relationship amongst two classes and is static in nature. It shows fixed relationships between classes involved in a system.
- Association is shown by a solid line between two classes and it can be called as a binary association. Association relationship can be of the type binary, ternary and onwards.
- Binary association refers to two association ends, ternary association denotes three association ends and so on. For every association end, we can mention multiplicity.
- Association amongst three or more classes can be achieved by means of n-ary association.
- We can show a multiplicity of exactly one (1), zero or one (0...1), one or more (1...*) or many (0...*) at association ends.
- Along with the multiplicity, we can have aggregation amongst association ends. Aggregation indicates ownership along with a relationship amongst lifelines.
- Furthermore, association ends can be public, private or protected.

2.1.2 Abstract Class

GQ. Write a short note on : Abstract class.

- A class without instances is an abstract class. An abstract class cannot be directly instantiated. An abstract class can be used as a template for design of subclasses since, it has no instances. It is not used for creation of objects. It defines a shared interface for its subclasses. Most of the super classes are abstract classes.
- An abstract class should have at least one abstract operation. Abstract operation is an operation that is stated in an abstract class but not executed. Interface can be defined in the form of abstract operations in case of an abstract class and it can be stretched furthermore by means of inclusion of number of operations in a class.

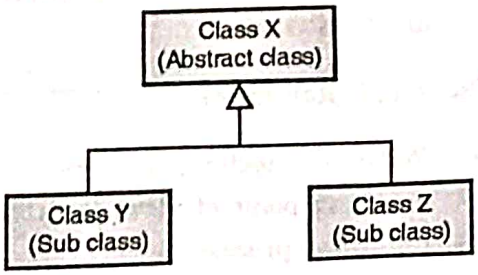


Fig. 2.1.2 : Abstract Class and Sub class

- An abstract class initiate code sharing since software developers can define concrete method that makes use of abstract methods. An abstract class allows for flexible object oriented modeling.
- Fig. 2.1.2 depicts example of abstract class and subclasses.
- Class X is an abstract class while Class Y and Class Z are sub classes. Class X can encapsulates number of generalized attributes that can be accessed by class Y and Z.

2.1.3 Multiple Inheritance

- It is possible for a class to have more than one superclass in Unified Modeling Language. That means, a child can have more than one parent and this property is known as Multiple Inheritance.
- Multiple inheritance is not supported by all of the object oriented languages. C# and Java permits only single inheritance. A child class can inherit properties of its superclasses in multiple inheritance.
- Implementation of multiple inheritance is possible in object oriented programming only if it is supported by the target implementation language. This can be considered as one of the design issue in object oriented programming and hence, implementation of multiple inheritance is totally dependent on the target object oriented language.
- The "***is kind of***"relationship should be applied between the superclasses and subclasses.
- Normally, superclasses should not have a parent class in common in order to avoid cycles in the inheritance hierarchy. All the superclasses involved in the scenario should be semantically disjoint.
- Sometimes, classes are defined to be "mixed in" with other classes with the help of the inheritance. Such types of classes are known as ***mixin classes***. Mixin classes supports multiple inheritance effectively.

2.1.4 Metadata

- Metadata is data about data. In software modeling and design, a class archetype is a good example of metadata. Classes involved in a class model are termed as meta objects.
- We have already discussed an example of bicycle in section 1.5 of this unit. A bicycle class may have different attributes like manufacturer, color, cost, number of gears and many more.
- Furthermore, a bicycle can be manufactured by several manufacturers, in different colors and shapes and with many other features.
- So, this detailed data is a kind of metadata and in this way, several classes involved in the class model comes with their own metadata.
- Metadata helps us to explain a particular class in brief for better understanding the class archetype and hence to get basic idea about overall structure and behavior of the proposed software system application.

2.1.5 Reification

- Reification technique offers a facility to a particular thing to make it real. From object oriented modeling point of view, the thing which is not an object can be endorsed as an object by means of reification process.
- So, the data constraints, methods and control information can be manipulated as a data or information and the same can be used as objects in object oriented modeling and design.

2.1.6 Constraints

- As it is discussed earlier, the elements of class model are object, class, association, aggregation, generalization and multiplicity. These elements of a class archetype are managed and monitored by means of values associated with themselves. These values associated with elements can be of the type Boolean.
- These Boolean values associated with elements of class model can be termed as constraints. Different constraints can be assumed depending on the situation and elements involved within the class model.
- Constraints are quite helpful and provide support in assessing the quality of the class archetype.

2.1.7 Derived Data

1. Derived data is a kind of data that can be derived by any element involved in the software system scenario.
2. Derived data can be completely determined by any of the components of a class model and hence it is redundant.
3. In class model, components like classes, attributes and relationships can be derived.
4. So, as far as class model is concerned; we may have derived classes, derived attributes and derived relationships for instance, derived associations.

2.2 Packages

GQ. Explain the concept of package in detail. Draw UML notation for a package.

- The building blocks of Unified Modeling Language comprises of relationships, interfaces, objects and UML diagrams. When we have to extend the system to some more extent, it is quite necessary to organize all of these contents collectively into large containers.
- The package is defined as a container that organizes a model by grouping things. Hence, packages help us to arrange all of the elements collectively for better understanding of the system archetype.
- The package is a widespread mechanism dedicated for organizing all of the building blocks of UML into groups. Normally, packages are used for presenting behavioural and structural views of the software system.
- Semantically associated elements are grouped by means of concept of package. The UML notation for package is a folder and the name of a package can be given on the tab in case the package contents are shown, otherwise package name is mentioned on the body of the folder itself as shown in Fig. 2.2.1.

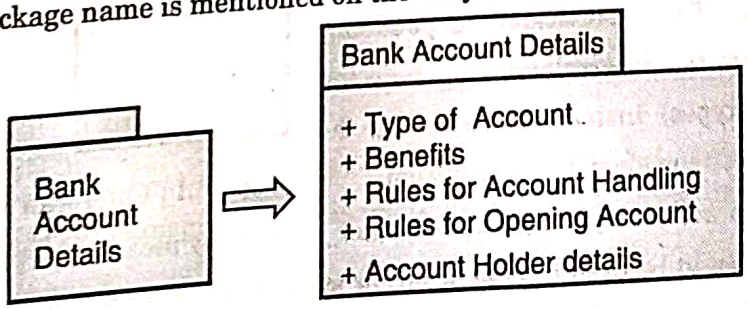


Fig. 2.2.1 : UML Notation for Package

2.3 Package Diagram

GQ. List and explain types of dependencies in package.
GQ. Draw a package diagram for ATM system scenario. Explain the scenario and your assumptions in brief.

- A package diagram represents a collection of classes and their interrelationships. For larger systems, drawing a package diagram can be one of the valuable things in order to control the design of a system archetype.
- Preferably, the package diagram should be prepared from the code of the subsystem. In package diagram, a good package can be identified on the basis of the perfect flow of dependencies among packages involved in the scenario along with the overall flow of the system. We can have nested packages inside other packages.
- By rule, there should not be cycles in dependency relationships amongst packages involved in the scenario.
- The interfaces of the packages should be stable in case of the numerous dependencies in between packages.
- There are five categories of dependencies amongst packages.
  - ***<<import>>*** : The <<import>> dependency combine client and supplier namespaces and it implements a public merge.
  - ***<<access>>*** : The <<access>> dependency also combine client and supplier namespaces. Only the difference is that, it implements a private merge.
  - ***<<use>>*** : The <<use>> dependency represents the interrelationship between the elements within the packages and not the packages themselves.
  - ***<<merge>>*** : The <<merge>> dependency is used in metamodeling only. We should not take into consider this kind of dependency in case of object oriented modeling and design.
  - ***<<trace>>*** : The <<trace>> dependency typically shows the interrelationship between models (archetypes) instead of depicting the relationship between elements of a system.

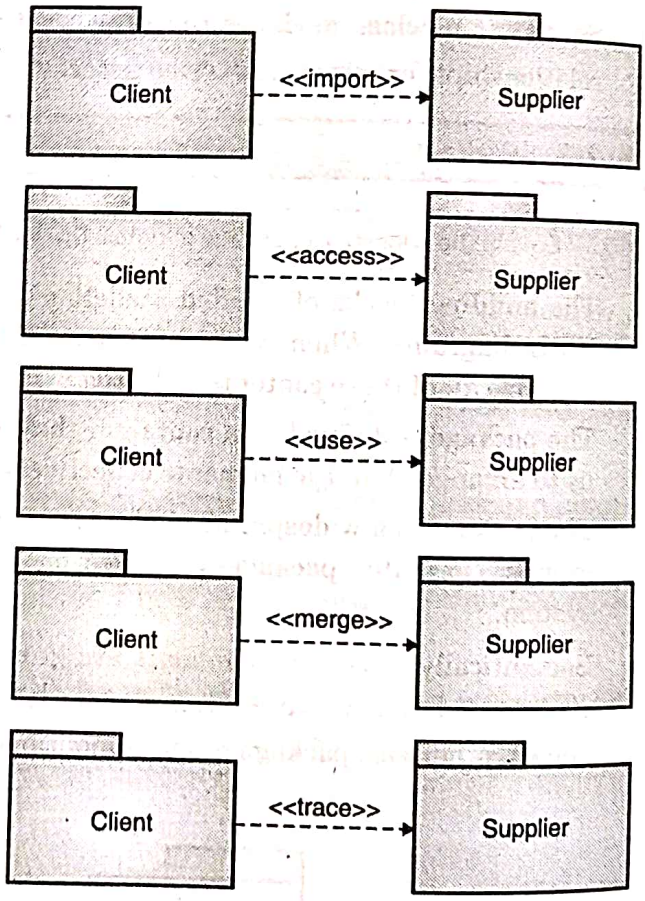


Fig. 2.3.1 : Categories of package dependencies

- We can merge classes from one package into another package by means of dependency.
- Package diagrams should be used in case of larger systems in order to get a detailed representation of the dependencies amongst key elements of a system.

- Dependencies within a proposed system can be handled effectively with the help of package diagram. Let us have a look at package diagram for ATM system.
- As we have already discussed, the package diagram shows the collection of classes and their relationships.
- In the example of ATM system, two top level classes are involved: Main ATM & Accounting ATM. Both classes unitedly coordinate the activities involved in the proper execution of an ATM system.

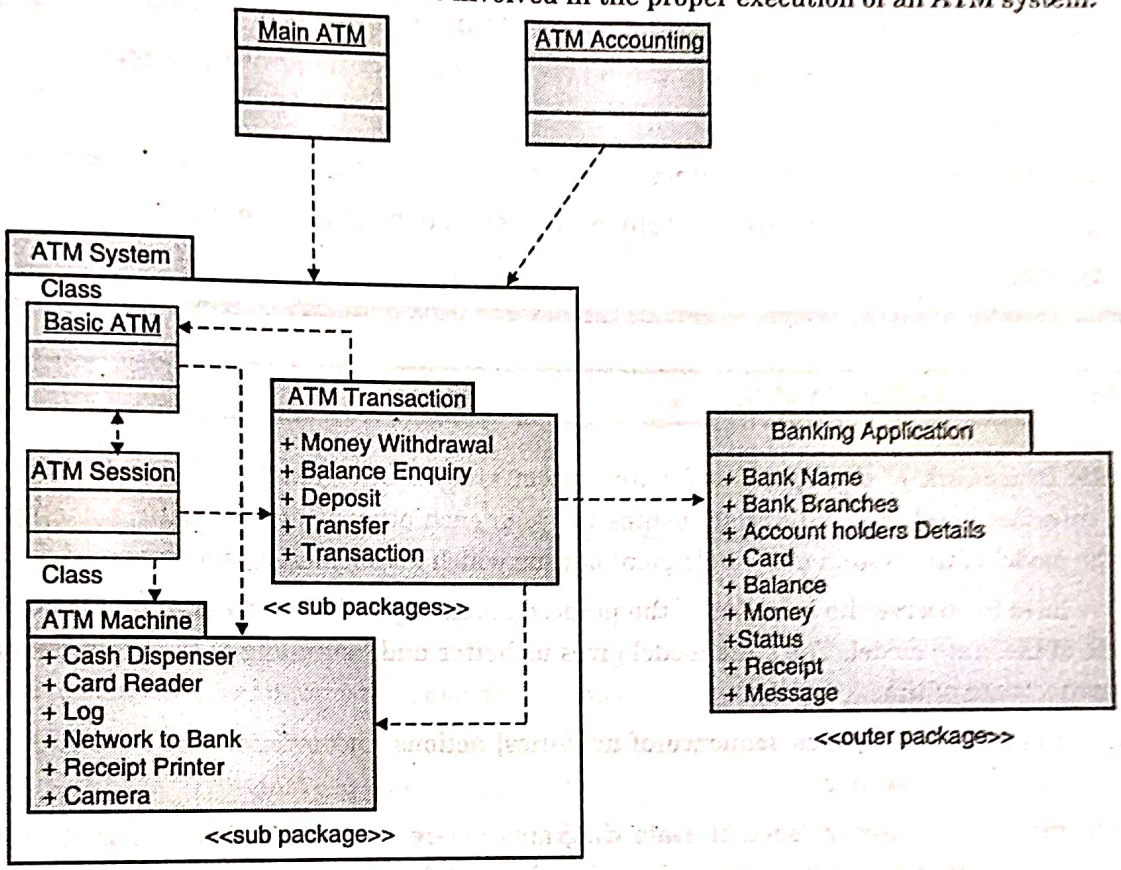


Fig. 2.3.2 : Package Diagram for ATM System

- Inside a package ATM, Basic ATM sub class is dependent on ATM Session class since, ATM system works in sessions and each session makes use of a basic class ATM for successful execution of that particular session.
- Main ATM package comes with two sub packages - ATM Transaction and ATM Machine.
- The sub package ATM Transaction is dedicated for execution of individual transactions originated by a customer.
- The ATM Session class is dependent on subpackage ATM Transaction since ATM Session class produces specific transaction objects. Furthermore, ATM Machine subpackage depicts all of the physical elements of the ATM system.
- The outer package Banking Application comprises of the classes that represents banking enterprise itself which is dedicated for managing actual communication and coordination between ATM Machines and Bank Branches.

Guidelines for designing a Package Diagram

- A well-organized package is loosely coupled and normally less nested.
- Packages in package diagram should not be too large for better understanding of the system archetype.
- Simple form of the package (i.e. folder) should be used for representation of a package. Detailed contents of the package should be given on the body of the package in extreme cases only.
- Only meaningful contents (objects, classes, relationships and interfaces) which are important to understand the system model should be shown explicitly in the package diagram.

2.4 Introduction to State Modeling

- The static framework of a proposed software system gives a brief idea about a collection of objects, classes, interfaces and their interrelationships between each other. But, the static archetype typically shows the model of the system at a single point of time which is considered as a class model.
- When, we have to observe the behavior of the proposed archetype of the system in real time, we have to make use of the state model. The state model gives us better understanding of the proposed system at a particular instance of time.
- In fact, the state model defines sequence of activities, actions and events which are involved in the proper execution of the system.
- The state model comprises of several state diagrams where each state diagram is dedicated for a particular class within the system scenario and is devoted for showing the behavior of the class at a particular instance of time.
- The state diagram can be considered as a graphical representation of finite state machines. Events and states are the important building blocks of the state diagram.

2.5 Events

GQ. Define event and explain in brief different types of events.
GQ. Explain in brief :
(a) Change in a state event (b) Passing of time event

- In the real world, many things might occur at the same point of time. In simple manner, things that occur at a particular instance of time are known as events.
- Typically, all real systems are dynamic in nature. That is, real systems accepts an input and generates an output by means of a series of events and actions at a particular point of time. For instance, in case of an ATM machine, a customer initiates a transaction by pressing the button on the screen.

- Every single thing or action occurred at a particular time instance is termed as an event in UML. An event owns an activity or operation along with its time and location.
- A transformation of a state and a signal are asynchronous events while calls are synchronous events. Asynchronous events are the events that may occur in random time and synchronous events depict the invocation of an operation.
- Fig. 2.5.1 represents a graphical representation of an event in UML.

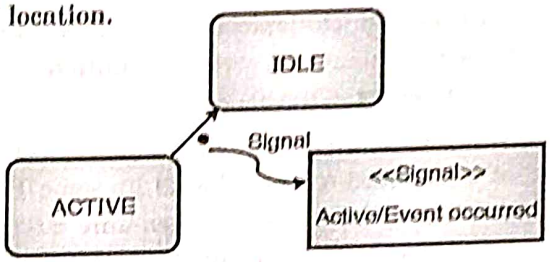


Fig. 2.5.1 : Event

- We can declare an event as shown in the Fig. 2.5.1 in order to imagine and understand the event scenario.
- Events are broadly categorized into two types: internal events and external events. ***Internal events*** are communicated in between the objects which are situated inside the system. ***External events*** are communicated among the subsystem and its actors. So, we can represent events internally within states and externally on transitions.
- With the help of UML, we can classify events into four types :

1.	Signal event	2.	Change in a state event
3.	Passing of time event	4.	Call event

2.5.1 Signal Event

- A signal event is a type of event meant for description of an asynchronous event communicated amongst instances.
- A signal can be considered as a bundle of information that is communicated between instances asynchronously.
- In UML, a signal event is shown as a stereotyped class. The information which is to be communicated in between instances of a class is stored in the form of attributes in a stereotyped class as shown in Fig. 2.5.2.

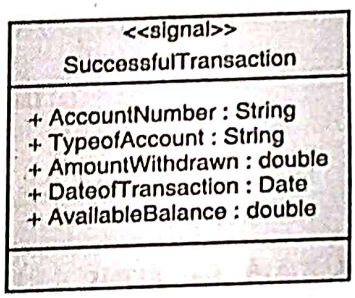


Fig. 2.5.2 : Signal event

- A signal can be directed as the message passing action in an interaction or the act of a state transition in a state machine.
- In UML, when we design a particular element (object, class, interface, etc.); it is quite important to define the behavior of the element comprising of attributes, signal events and operations handled by that particular element.
- A signal sent is represented by convex pentagon containing the signal name on its body while signal received is shown by a concave pentagon shown in Fig. 2.5.3.

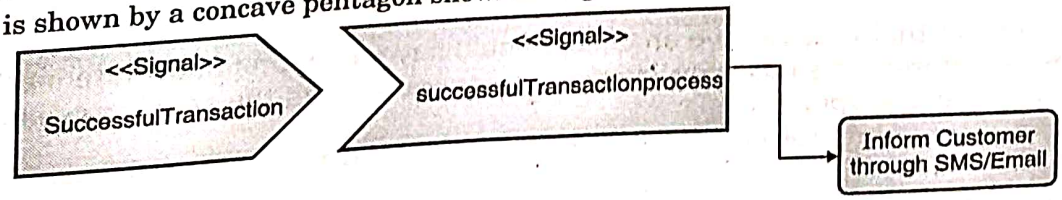


Fig. 2.5.3 : UML notation for signal event sent and received

2.5.2 Change in a State Event

- An event which depicts a change or transformation in state after fulfilment of a certain condition is called as a change in a state event.
- We can show a change in a state type of event by making use of a keyword *when* followed by some Boolean expression or condition. For instance, *when date = 03/05/1989*. For better understanding, refer Fig. 2.5.4.

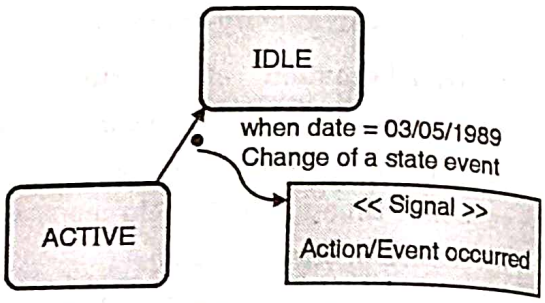
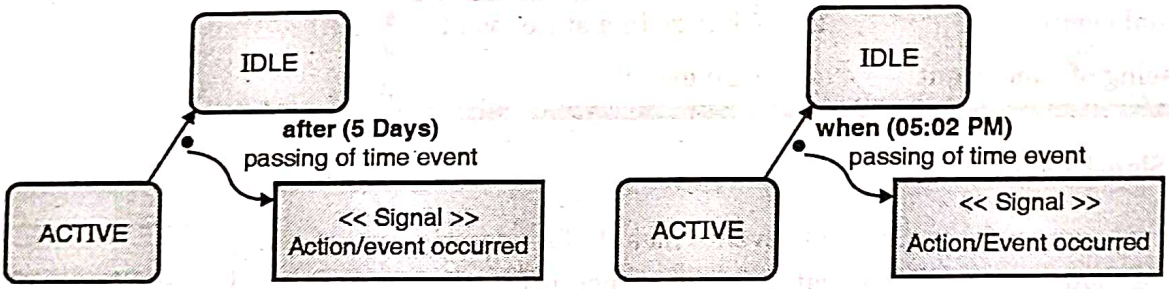


Fig. 2.5.4 : Change of a state event

- A change of a state event is initiated and triggered after each and every single change in a Boolean expression.

2.5.3 Passing of Time Event

- An event that shows a passage of time is known as passing of time event. Normally, this type of event is represented with the help of keywords ***after*** and ***when***.
- The keyword ***after*** states ***a threshold time*** after which an event is initiated and executed and the keyword ***when*** specifies ***a particular instance of time*** at which an event is initiated. We should always make sure that, the units of time for e.g., hours, minutes, seconds, etc. are noted in the diagram.



(a) Use of key word "after" **(b) Use of key word "when"**

Fig. 2.5.5 : Passing of time event

- Fig. 2.5.5 (a) and (b) depicts a passing of time event in UML using keywords ***after*** and ***when*** respectively.

2.5.4 Call Event

- A call event illustrates the actual dispatch and report of a particular operation.
- It is a request for a particular operation to be executed on an object of the class.
- Usually, a call event is synchronous event. That is, when an instance of a class invokes a specific operation on another instance that has a state machine, control switches from the sender to the receiver, the operation is accomplished, the transition is initiated by the call event, then control returns back to the sender and receiver transits to the new state.

Guidelines for Designing an Event

- Each element that may receive an event should have an appropriate state machine.
- Define and design the receiving event elements as well as the sending event elements in a system archetype.
- Ordinary flow of control in a system archetype should not be replaced by the sending event signals.

2.6 States

GQ. What is state ? Explain the contents of a state in detail.

- *In UML Reference Manual by Rumbaugh*, a state is defined as, "a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event".
- An instance of a class holds a specific state for a particular instance of time. The state of an object may correspond to a continuous activity and it fluctuates over time.
- Let us, consider an example of a washing machine in a home. Basically, it works in four states: Idle state, Activating State, Active state and End state.
- In Idle state, washing machine is in ready to work state and is waiting for a command from end user. In its Activating state, machine is ON but it is again waiting for a particular action or activity by end user. An Active state is a proper working state of a washing machine while in End state, machine is OFF.

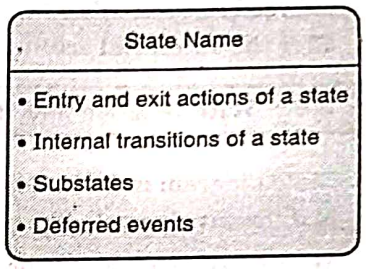


Fig. 2.6.1 : UML Notation of a State

- A state has five parts as shown in Fig. 2.6.1.
  1. Each state in a state diagram contains zero or more activities and actions.
  2. A ***state name*** is a textual string that differentiates a particular state from other state. We can declare a state without a name.
  3. ***Entry and Exit actions*** are the actions performed on entering and exiting the state respectively.
  4. ***Internal transition of a state*** is the transition that is controlled deprived of any change in a state.
  5. ***Substates*** are the nested structure of states and shows a set of substates.
  6. ***Deferred events*** are the events which are not handled in the state but are suspended and are waiting in line of another object for further treatment.

2.7 Transitions and Conditions

- A transition is nothing but the path or relationship between two states.
- It is responsible for showing the actual transformation or change from one state to any other state in the state machine configuration.
- Each transaction comes with a guard condition which specifies if the transition can be assisted, a trigger that causes the transition to execute if it is enabled and any effect the transition may have when it happens.
- Transitions are shown as a line between two states or pseudostates with a solid arrowhead pointing to the destination state.
- A single transition has :
  - Source state
  - Guard condition
  - Target state
  - Event trigger
  - Action

- Source state is the state affected by the transition; if an instance is in this state and the guard condition if any is satisfied.
- Event trigger designates what condition may cause the particular transition to happen. Usually, an event trigger is the name of an event. It is relatively possible to have a trigger less transition.
- Guard condition is a control condition which is estimated when an event is fired by the state machine in order to decide if the transition should be enabled. Guard conditions will continually be assessed before a transition is fired.
- Action is an executable thing that may straightway act on the instance that owns the state machine and indirectly act on other instances that are visible to the object. Target state is the state which is active after the successful accomplishment of the transition.

2.8 State Diagram and Its Behavior

- State diagram in UML depicts the behavior of a software system and can be used to model the behavior of different elements like a class, a subsystem or entire software system.
- We use state diagram for designing the dynamic aspects of a software system. Well organized state diagram are simple, easily understandable and just like a well-planned and well-controlled algorithms.
- The state diagram depicts implementation of each specific element involved in the software system scenario.
- As we have already discussed, the state diagram comprises of three fundamental elements :
  - Event
  - State
  - Transition
- Every single thing or action occurred at a particular time instance is termed as an event. The state of an object may correspond to a continuous activity and it fluctuates over time.
- Transition for showing the actual transformation or change from one state to any other state in the state machine configuration.

Key Concepts

• Advanced Object	• Signal event
• Advanced Class	• Change in a state event
• Classifier	• Passing of time event
• Abstract Class	• Call event
• Multiple Inheritance	• State
• Package	• State Transition
• Package Diagram	• State diagram
• Event	• Nested State

Summary

- A class without instances is an ***abstract class.***
- A child can have more than one parent and this property is known as ***multiple inheritance.***
- The ***package*** is defined as a container that organizes a model by grouping things. Hence, packages help us to arrange all of the elements collectively for better understanding of the system archetype.
- A ***package diagram*** represents a collection of classes and their interrelationships.
- The ***state model*** comprises of several state diagrams where each state diagram is dedicated for a particular class within the system scenario and is devoted for showing the behavior of the class at a particular instance of time.
- The ***state diagram*** can be considered as a graphical representation of finite state machines. Events and states are the important building blocks of the state diagram.
- In the real world, many things might occur at the same point of time. In simple manner, things that occur at a particular instance of time are known as ***events.***
- Every single thing or action occurred at a particular time instance is termed as an ***event*** in UML.
- A transformation of a state and a signal are asynchronous events while calls are ***synchronous events.***
- ***Asynchronous events*** are the events that may occur in random time and synchronous events depicts the invocation of an operation.
- ***Internal events*** are communicated in between the objects which are situated inside the system.
- ***External events*** are communicated among the subsystem and its actors. So, we can represent events internally within states and externally on transitions.
- A ***signal event*** is a type of event meant for description of an asynchronous event communicated amongst instances.
- A ***signal*** can be considered as a bundle of information that is communicated between instances asynchronously.
- An event which depicts a change or transformation in state after fulfilment of a certain condition is called as a ***change in a state event.***
- A ***call event*** illustrates the actual dispatch and report of a particular operation.
- A ***transition*** is nothing but the path or relationship between two states.
- ***Guard condition*** is a control condition which is estimated when an event is fired by the state machine in order to decide if the transition should be enabled. Guard conditions will continually be assessed before a transition is fired.
- ***State diagram*** in UML depicts the behavior of a software system and can be used to model the behavior of different elements like a class, a subsystem or entire software system.
- The basic difference between a simple state and a composite state is that, there is no substructure in case of a simple state but, a composite state might hold sequential concurrent substates.

Chapter Ends...

❑❑❑