Module 7

Object Modeling using UML

Lesson 17

Activity and State Chart Diagram

Specific Instructional Objectives

At the end of this lesson the student will be able to:

- Draw activity diagrams for any given problem.
- Differentiate between the activity diagrams and procedural flow charts.
- Develop the state chart diagram for any given class.
- Compare activity diagrams with state chart diagrams.

Activity diagrams

The activity diagram is possibly one modeling element which was not present in any of the predecessors of UML. No such diagrams were present either in the works of Booch, Jacobson, or Rumbaugh. It is possibly based on the event diagram of Odell [1992] through the notation is very different from that used by Odell. The activity diagram focuses on representing activities or chunks of processing which may or may not correspond to the methods of classes. An activity is a state with an internal action and one or more outgoing transitions which automatically follow the termination of the internal activity. If an activity has more than one outgoing transitions, then these must be identified through conditions. An interesting feature of the activity diagrams is the swim lanes. Swim lanes enable you to group activities based on who is performing them, e.g. academic department vs. hostel office. Thus swim lanes subdivide activities based on the responsibilities of some components. The activities in a swim lane can be assigned to some model elements, e.g. classes or some component, etc.

Activity diagrams are normally employed in business process modeling. This is carried out during the initial stages of requirements analysis and specification. Activity diagrams can be very useful to understand complex processing activities involving many components. Later these diagrams can be used to develop interaction diagrams which help to allocate activities (responsibilities) to classes.

The student admission process in IIT is shown as an activity diagram in fig. 7.15. This shows the part played by different components of the Institute in the admission procedure. After the fees are received at the account section, parallel activities start at the hostel office, hospital, and the Department. After all these activities complete (this synchronization is represented as a horizontal line), the identity card can be issued to a student by the Academic section.

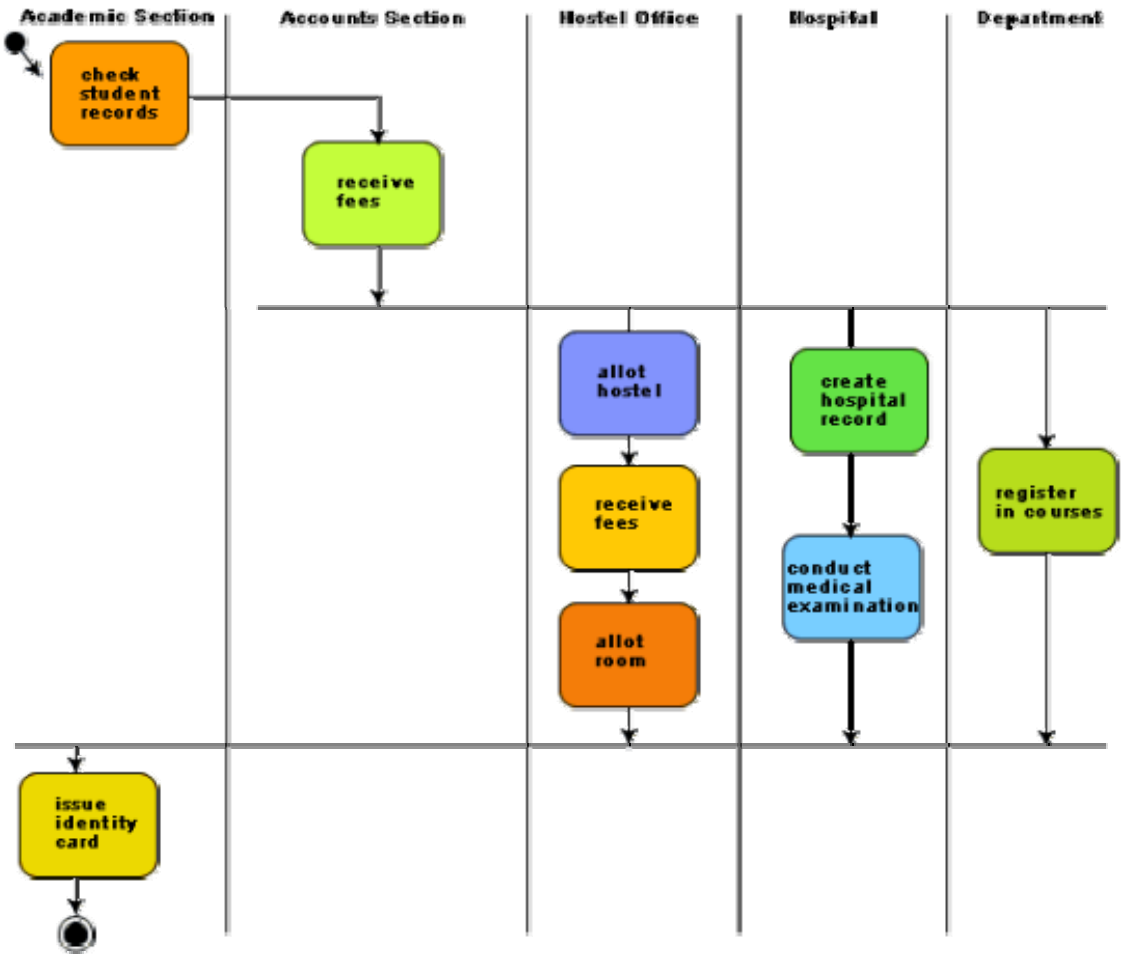


Fig. 7.15: Activity diagram for student admission procedure at IIT

Activity diagrams vs. procedural flow charts

Activity diagrams are similar to the procedural flow charts. The difference is that activity diagrams support description of parallel activities and synchronization aspects involved in different activities.

State chart diagram

A state chart diagram is normally used to model how the state of an object changes in its lifetime. State chart diagrams are good at describing how the behavior of an object changes across several use case executions. However, if we are interested in modeling some behavior that involves several objects collaborating with each other, state chart diagram is not appropriate. State chart diagrams are based on the finite state machine (FSM) formalism.

An FSM consists of a finite number of states corresponding to those of the object being modeled. The object undergoes state changes when specific events occur. The FSM formalism existed long before the object-oriented technology and has been used for a wide variety of applications. Apart from modeling, it has even been used in theoretical computer science as a generator for regular languages.

A major disadvantage of the FSM formalism is the state explosion problem. The number of states becomes too many and the model too complex when used to model practical systems. This problem is overcome in UML by using state charts. The state chart formalism was proposed by David Harel [1990]. A state chart is a hierarchical model of a system and introduces the concept of a composite state (also called nested state).

Actions are associated with transitions and are considered to be processes that occur quickly and are not interruptible. Activities are associated with states and can take longer. An activity can be interrupted by an event.

The basic elements of the state chart diagram are as follows:

- **Initial state.** This is represented as a filled circle.
- **Final state.** This is represented by a filled circle inside a larger circle.
- **State.** These are represented by rectangles with rounded corners.
- **Transition.** A transition is shown as an arrow between two states. Normally, the name of the event which causes the transition is places along side the arrow. A guard to the transition can also be assigned. A guard is a Boolean logic condition. The transition can take place only if the grade evaluates to true. The syntax for the label of the transition is shown in 3 parts: event[guard]/action.

An example state chart for the order object of the Trade House Automation software is shown in fig. 7.16.

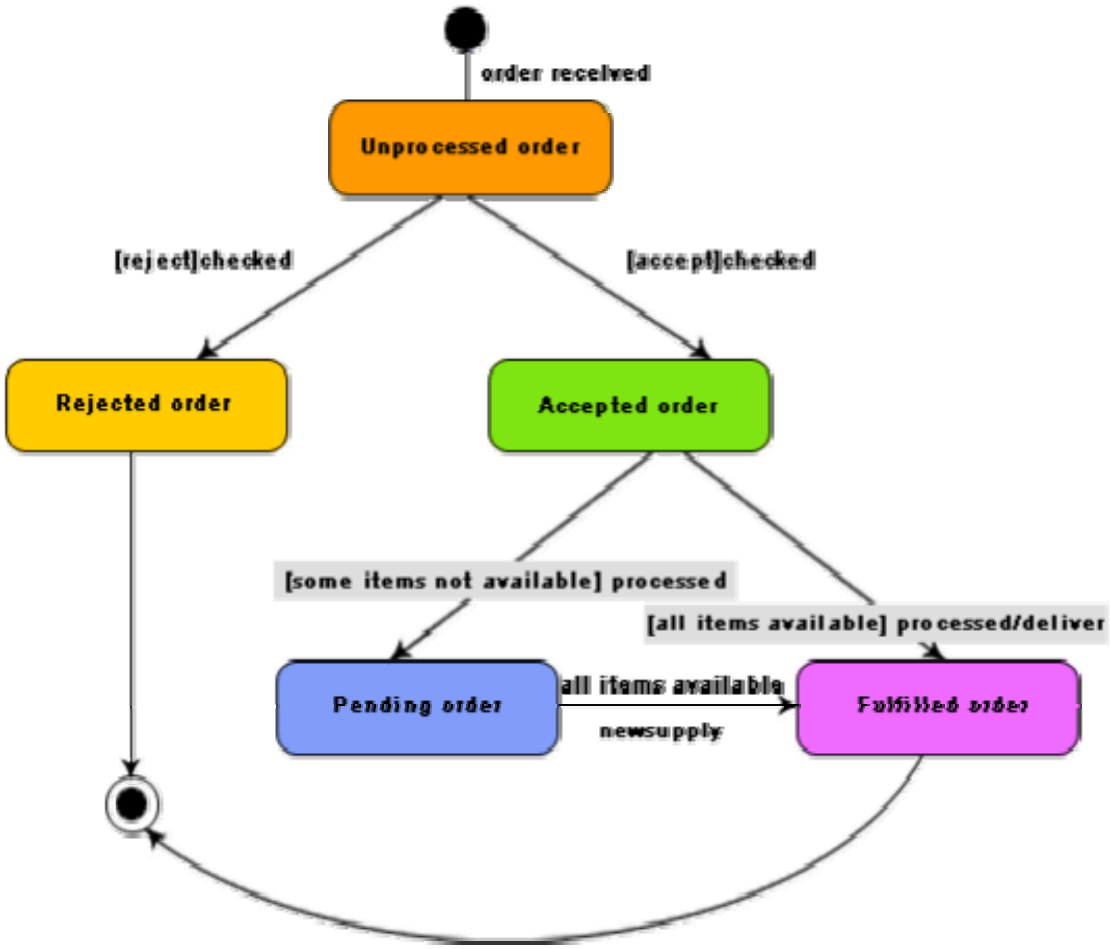


Fig. 7.16: State chart diagram for an order object

Activity diagram vs. State chart diagram

- Both activity and state chart diagrams model the dynamic behavior of the system. Activity diagram is essentially a flowchart showing flow of control from activity to activity. A state chart diagram shows a state machine emphasizing the flow of control from state to state.
- An activity diagram is a special case of a state chart diagram in which all or most of the states are activity states and all or most of the transitions are triggered by completion of activities in the source state (An activity is an ongoing non-atomic execution within a state machine).

- Activity diagrams may stand alone to visualize, specify, and document the dynamics of a society of objects or they may be used to model the flow of control of an operation. State chart diagrams may be attached

to classes, use cases, or entire systems in order to visualize, specify, and document the dynamics of an individual object.

The following questions have been designed to test the objectives identified for this module:

1. Explain why is it necessary to create a model in the context of good software development.

Ans.: - An important reason behind constructing a model is that it helps manage complexity. Once models of a system have been constructed, these can be used for a variety of purposes during software development, including the following**:**

- Analysis
- Specification
- Code generation
- Design
- Visualize and understand the problem and the working of a system
- Testing, etc.

Since a model can be used for a variety of purposes, it is reasonable to expect that the model would vary depending on the purpose for which it is being constructed. For example, a model developed for initial analysis and specification should be very different from the one used for design. A model that is being used for analysis and specification would not show any of the design decisions that would be made later on during the design stage. On the other hand, a model used for design purposes should capture all the design decisions. Therefore, it is a good idea to explicitly mention the purpose for which a model has been developed, along with the model.

2. Identify different types of views of a system captured by UML diagrams.

Ans.: - UML can be used to construct nine different types of diagrams to capture five different views of a system. Different UML diagrams provide different perspectives of the software system to be developed and facilitate a comprehensive understanding of the system. Such models can be refined to get the actual implementation of the system.

The UML diagrams can capture the following five views of a system**:**

- User's view
- Structural view

- Behavioral view
- Implementation view
- Environmental view

User's view: This view defines the functionalities (facilities) made available by the system to its users. The users' view captures the external users' view of the system in terms of the functionalities offered by the system. The users' view is a black-box view of the system where the internal structure, the dynamic behavior of different system components, the implementation etc. are not visible. The users' view is very different from all other views in the sense that it is a functional model compared to the object model of all other views. The users' view can be considered as the central view and all other views are expected to conform to this view. This thinking is in fact the crux of any user centric development style.

Structural view: The structural view defines the kinds of objects (classes) important to the understanding of the working of a system and to its implementation. It also captures the relationships among the classes (objects). The structural model is also called the static model, since the structure of a system does not change with time.

Behavioral view: The behavioral view captures how objects interact with each other to realize the system behavior. The system behavior captures the time-dependent (dynamic) behavior of the system.

Implementation view: This view captures the important components of the system and their dependencies.

Environmental view: This view models how the different components are implemented on different pieces of hardware.

3. What is the purpose of a use case?

Ans.: - The purpose of a use case is to define a piece of coherent behavior without revealing the internal structure of the system. The use cases do not mention any specific algorithm to be used or the internal data representation, internal structure of the software, etc. A use case typically represents a sequence of interactions between the user and the system. These interactions consist of one mainline sequence. The mainline sequence represents the normal interaction between a user and the system. The mainline sequence is the most occurring sequence of interaction. For example, the mainline sequence of the withdraw cash use case supported by a bank ATM drawn, complete the transaction, and get the amount. Several variations to the main line sequence may also exist. Typically, a variation from the mainline sequence occurs when some specific conditions hold. For the bank ATM example, variations or alternate scenarios may occur, if the password is invalid or the amount to be withdrawn

exceeds the amount balance. The variations are also called alternative paths. A use case can be viewed as a set of related scenarios tied together by a common goal. The mainline sequence and each of the variations are called scenarios or instances of the use case. Each scenario is a single path of user events and system activity through the use case.

4. Which diagrams in UML capture the behavioral view of the system?

Ans.: - The behavioral view is captured by the following UML diagrams:

- Sequence diagrams
- Collaboration diagrams
- State chart diagrams
- Activity diagrams

5. Which UML diagrams capture the structural aspects of a system?

Ans.: - Structural aspects of a system are captured by the following UML diagrams:

- Class diagrams
- Object diagrams

6. Which UML diagrams capture the important components of the system and their dependencies?

Ans.: - Implementation view captures the important components of the system and their dependencies.

7. Represent the following relations among classes using UML diagram.

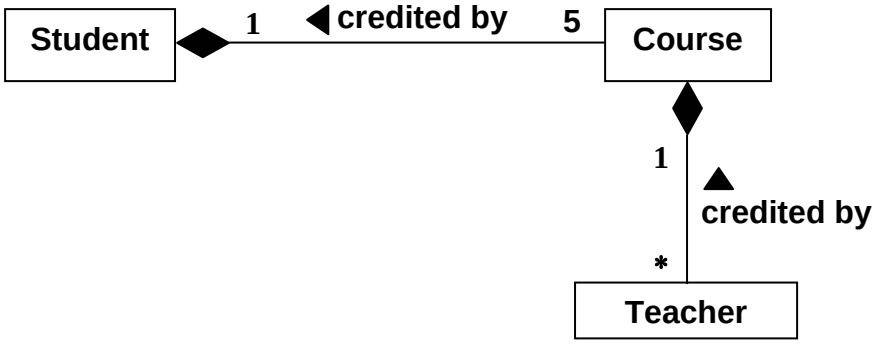
1. *Students credit 5 courses each semester. Each course is taught by one or more teachers.*

2. *Bill contains number of items. Each item describes some commodity, the price of unit, and total on this price.*

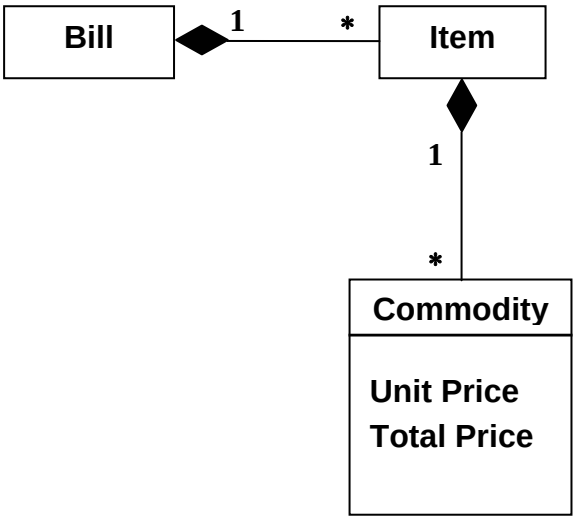
3. *An order consists of one or more order items. Each order item contains the name of the item, its quantity and the date by which it is required. Each order item is described by an item type specification object having details such as its vendor addresses, its unit price, and the manufacturer.*

Ans.: -

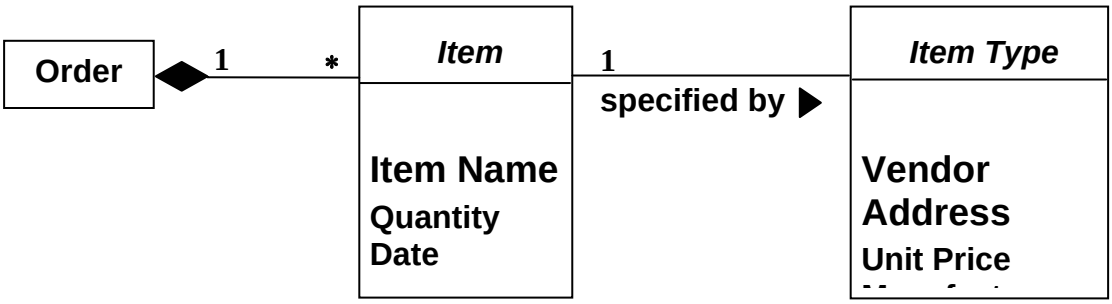
1)


Student
1
credited by
5
Course
1
credited by
*
Teacher

2)


Bill
1
*
Item
1
*
Commodity
Unit Price
Total Price

3)


Order
1
*
Item
1
specified by
Item Type
Item Name
Quantity
Date
Vendor
Address
Unit Price

8. What is the necessity for developing use case diagram?

Ans.: - From use case diagram, it is obvious that the utility of the use cases are represented by ellipses. They along with the accompanying text description serve as a type of requirements specification of the system and form the core model to which all other models must conform. But, what about the actors (stick person icons)? One possible use of identifying the different types of users (actors) is in identifying and implementing a security mechanism through a login system, so that each actor can involve only those functionalities to which he is entitled to. Another possible use is in preparing the documentation (e.g. users' manual) targeted at each category of user. Further, actors help in identifying the use cases and understanding the exact functioning of the system.

9. How to identify use cases of a system?

Ans.: - The use case model for any system consists of a set of "use cases". Intuitively, use cases represent the different ways in which a system can be used by the users. A simple way to find all the use cases of a system is to ask the question: "What the users can do using the system?" Thus for the Library Information System (LIS), the use cases could be**:**

- issue-book
- query-book
- return-book
- create-member
- add-book, etc.

Use cases correspond to the high-level functional requirements. The use cases partition the system behavior into transactions, such that each transaction performs some useful action from the user's point of view. To complete each transaction may involve either a single message or multiple message exchanges between the user and the system to complete.

10. What is the difference between an operation and a method in the context of object-oriented design technique?

Ans.: - There is a distinction between the terms operation and method. An operation is something that is supported by a class and invoked by objects of other classes. There might be multiple methods implementing the same operation. This is called static polymorphism. The method names can be the same; however, it should be possible to distinguish the methods by examining their parameters. Thus, the terms operation and the method are distinguishable only when there is polymorphism.

11. What does the association relationship among classes represent? Give examples of the association relationship.

Ans.: - Associations are needed to enable objects to communicate with each other. An association describes a connection between classes. The association relation between two objects is called object connection or link. Links are instances of associations. A link is a physical or conceptual connection between object instances. For example, suppose Amit has borrowed the book Graph Theory. Here, borrowed is the connection between the objects Amit and Graph Theory book. Mathematically, a link can be considered to be a tuple, i.e. an ordered list of object instances. An association describes a group of links with a common structure and common semantics. For example, consider the statement that Library Member borrows Books. Here, borrows is the association between the class LibraryMember and the class Book. Usually, an association is a binary relation (between two classes). However, three or more different classes can be involved in an association. A class can have an association relationship with itself (called recursive association). In this case, it is usually assumed that two different objects of the class are linked by the association relationship.

Association between two classes is represented by drawing a straight line between the concerned classes. Fig. 7.9 illustrates the graphical representation of the association relation. The name of the association is written along side the association line. An arrowhead may be placed on the association line to indicate the reading direction of the association. The arrowhead should not be misunderstood to be indicating the direction of a pointer implementing an association. On each side of the association relation, the multiplicity is noted as an individual number or as a value range. The multiplicity indicates how many instances of one class are associated with each other. Value ranges of multiplicity are noted by specifying the minimum and maximum value, separated by two dots, e.g. 1.5. An asterisk is a wild card and means many (zero or more). The association of fig. 7.9 should be read as “Many books may be borrowed by a Library Member”. Observe that associations (and links) appear as verbs in the problem statement.

12. What does aggregation relationship between classes represent? Give examples of aggregation relationship between classes.

Ans.: - Aggregation is a special type of association where the involved classes represent a whole-part relationship. The aggregate takes the responsibility of forwarding messages to the appropriate parts. Thus, the aggregate takes the responsibility of delegation and leadership. When an instance of one object contains instances of some other objects, then aggregation (or composition) relationship exists between the composite object and the component object. Aggregation is represented by the diamond symbol at the composite end of a relationship.

A document may consist of several paragraphs and each paragraph consists of many lines. Aggregation is represented by the diamond symbol (as shown in the fig. 7.10) at the composite end of a relationship.

13. Why are objects always passed by reference in all popular programming languages?

Ans.: - The size of objects may be large. Unless they are passed by reference, there can be overflow of the method called run-time stack.

Mark the following as either True or False. Justify your answer.

1. For any given problem, one should construct all the views using all the diagrams provided by UML.

Ans.: - False.

Explanation: - For a system in which the objects undergo many state changes, a state chart diagram may be necessary. For a system, which is implemented on a large number of hardware components, a deployment diagram may be necessary. So, the type of models to be constructed depends on the problem at hand.

2. Use cases are explicitly dependent among themselves.

Ans.: - False.

Explanation: - Normally, each use case is independent of the other use cases. However, implicit dependencies among use cases may exist because there might exist dependencies among the use cases at the implementation level due to shared resources, objects, or functions. For example, in the Library Automation System example, renew-book and reserve-book are two independent use cases.

3. Each actor can participate in one and only one use case.

Ans.: - False.

Explanation: - In case of use case diagram, different users of the system are represented by using the stick person icon. Each stick person icon is normally referred to as an actor. An actor is a role played by a user with respect to the system use. It is possible that the same user may play the role of multiple actors. Each actor can participate in one or more use cases.

4. **Class diagrams developed using UML can serve as the functional specification of a system.**

Ans.: - False.

Explanation: - A class diagram describes the static structure of a system. It shows how a system is structured rather than how it behaves. The static structure of a system comprises of a number of class diagrams and their dependencies.

5. The terms method and operation are equivalent concepts and can be used interchangeably.

Ans.: - False.

Explanation: - An operation is something that is supported by a class and invoked by objects of other classes. There might be several methods in a class implementing the same operation. This is called the static polymorphism. The method names can be the same; however, it is possible to distinguish the methods by examining their parameters. Thus, the terms operation and method are distinguishable only when there is polymorphism.

6. A class can have an association relationship with itself.

Ans.: - A class can have an association relationship with itself (called recursive association). In this case, it is usually assumed that two different objects of the class are linked by the association relationship.

7. The Aggregation relationship can be recursively defined, i.e. an object can contain instances of itself.

Ans.: - False.

Explanation: - The aggregation relationship cannot be reflexive (i.e. recursive). That is, an object cannot contain objects of the same class as itself.

8. In a UML class diagram, the aggregation relationship defines an equivalence relationship among objects.

Ans.: - False.

Explanation: - The aggregation relationship cannot be reflexive (i.e. recursive). That is, an object cannot contain objects of the same class as itself. Also, the aggregation relation is not symmetric. That is, two classes A and B cannot contain instances of each other. But the aggregation relationship can be transitive because aggregation may consist of an arbitrary number of levels. For

those above reasons the aggregation relationship does not define an equivalence relationship among objects.

9. The aggregation relationship can be considered to be a special type of association relationship.

Ans.: - True.

Explanation: - Aggregation is a special type of association where the involved classes represent a whole-part relationship. The aggregate takes the responsibility of forwarding messages to the appropriate parts. Thus, the aggregate takes the responsibility of delegation and leadership.

10. The aggregation relationship can be reflexive.

Ans.: - The aggregation relationship cannot be reflexive (i.e. recursive). That is, an object cannot contain objects of the same class as itself.

11. The aggregation relationship cannot be reflexive and symmetric but is transitive.

Ans.: - True.

Explanation: - The aggregation relationship cannot be reflexive (i.e. recursive). That is, an object cannot contain objects of the same class as itself. Also, the aggregation relation is not symmetric. That is, two classes A and B cannot contain instances of each other. But the aggregation relationship can be transitive because, aggregation may consist of an arbitrary number of levels.

12. Normally, you use an interaction diagram to represent how the behavior of an object changes over its life time.

Ans.: - False.

Explanation: - Interaction diagrams are models that describe how groups of objects collaborate to realize some behavior. Typically, each interaction diagram realizes the behavior of a single use case. An interaction diagram shows a number of example objects and the messages that are passed between the objects within the use case. On the other hand, a state chart diagram is normally used to model how the state of an object changes over its life time.

13. The chronological order of the messages in an interaction diagram cannot be determined from an inspection of the diagram.

Ans.: - False.

Explanation: - A sequence diagram shows interaction among objects as a two dimensional chart. In a sequence diagram, a vertical dashed line is used to represent an object's lifeline. Each message is indicated as an arrow between the lifelines of two objects. The messages are shown in chronological order from the top to the bottom. That is, reading the diagram from the top to the bottom it is possible to see the sequence of messages in order.

14. The interaction diagrams can be effectively used to describe how the behavior of an object changes across several use case.

Ans.: - False.

Explanation: - Interaction diagrams are models that describe how groups of objects collaborate to realize some behavior. Typically, each interaction diagram realizes the behavior of a single use case. An interaction diagram shows a number of example objects and the messages that are passed between the objects within the use case. On the other hand, state chart diagrams are good at describing how the behavior of an object changes across several use case executions.

15. State chart diagrams in UML are normally used to model how some behavior of a system is realized through the co-operative actions of several objects.

Ans.: - False.

Explanation: - A state chart diagram is normally used to model how the state of an object changes in its life time. State chart diagrams are good at describing how the behavior of an object changes across several use case executions.

16. A state chart diagram is good at describing behavior that involves multiple objects cooperating with each other to achieve some behavior.

Ans.: - False.

Explanation: - State chart diagrams are good at describing how the behavior of an object changes across several use case executions. On the other hand, interaction diagrams are models that describe how groups of objects collaborate to realize some behavior.

Mark all options which are true.

1. UML is a

□ a language to model syntax
□ an object-oriented development methodology
□ an automatic code generation tool
□ none of the above √

2. Which of the following view captured by UML diagrams can be considered as black box model of a system?

□ structural view
□ behavioral view
□ user's view √
□ environmental view
□ implementation view

3. In the context of use case diagram, the stick person icon is used to represent

□ human users √
□ external systems
□ internal systems
□ none of the above