MCS-219: Object Oriented Analysis and Design
Guess Paper-I

Q1. Explain the principles of object Orientation?

Ans. The conceptual framework of object–oriented systems is based upon the object model. There are two categories of elements in an object-oriented system –

Major Elements – By major, it is meant that if a model does not have any one of these elements, it ceases to be object oriented. The four major elements are –

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

Minor Elements – By minor, it is meant that these elements are useful, but not indispensable part of the object model. The three minor elements are –

- Typing
- Concurrency
- Persistence

Abstraction: Abstraction means to focus on the essential features of an element or object in OOP, ignoring its extraneous or accidental properties. The essential features are relative to the context in which the object is being used. Grady Booch has defined abstraction as follows –

"An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer."

Example – When a class Student is designed, the attributes enrolment number, name, course, and address are included while characteristics like pulse rate and size_of_shoe are eliminated, since they are irrelevant in the perspective of the educational institution.

Encapsulation: Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. The class has methods that provide user interfaces by which the services provided by the class may be used.

Modularity: Modularity is the process of decomposing a problem (program) into a set of modules so as to reduce the overall complexity of the problem. Brooch has defined modularity as –

"Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules."

Modularity is intrinsically linked with encapsulation. Modularity can be visualized as a way of mapping encapsulated abstractions into real, physical modules having high cohesion within the modules and their inter–module interaction or coupling is low.

Hierarchy: In Grady Booch's words, "Hierarchy is the ranking or ordering of abstraction". Through hierarchy, a system can be made up of interrelated subsystems, which can have their own subsystems and so on until the smallest level components are reached. It uses the principle of "divide and conquer". Hierarchy allows code reusability.

The two types of hierarchies in OOA are –

"IS–A" hierarchy – It defines the hierarchical relationship in inheritance, whereby from a super-class, a number of subclasses may be derived which may again have subclasses and so on. For example, if we derive a class Rose from a class Flower, we can say that a rose "is–a" flower.

"PART–OF" hierarchy – It defines the hierarchical relationship in aggregation by which a class may be composed of other classes. For example, a flower is composed of sepals, petals, stamens, and carpel. It can be said that a petal is a "part–of" flower.

Typing: According to the theories of abstract data type, a type is a characterization of a set of elements. In OOP, a class is visualized as a type having properties distinct from any other types. Typing is the enforcement of the notion that an object is an instance of a single class or type. It also enforces that objects of different types may not be generally interchanged; and can be interchanged only in a very restricted manner if absolutely required to do so.

The two types of typing are –

- **Strong Typing:** Here, the operation on an object is checked at the time of compilation, as in the programming language Eiffel.
- **Weak Typing:** Here, messages may be sent to any class. The operation is checked only at the time of execution, as in the programming language Smalltalk.

Concurrency: Concurrency in operating systems allows performing multiple tasks or processes simultaneously. When a single process exists in a system, it is said that there is a single thread of control. However, most systems have multiple threads, some active, some waiting for CPU, some suspended, and some terminated. Systems with multiple CPUs inherently permit concurrent threads of control; but systems running on a single CPU use appropriate algorithms to give equitable CPU time to the threads so as to enable concurrency.

In an object-oriented environment, there are active and inactive objects. The active objects have independent threads of control that can execute concurrently with threads of other objects. The active objects synchronize with one another as well as with purely sequential objects.

Persistence

An object occupies a memory space and exists for a particular period of time. In traditional programming, the lifespan of an object was typically the lifespan of the execution of the program that created it. In files or databases, the object lifespan is longer than the duration of the process creating the object. This property by which an object continues to exist even after its creator ceases to exist is known as persistence.

Q2. Explain the different types of Modelling in UML?

Ans. There are three important types of modelling in UML, mentioned below:-

Structural Modeling: Structural modeling captures the static features of a system. They consist of the following −

- Classes diagrams.
- Objects diagrams.
- Deployment diagrams.
- Package diagrams.
- Composite structure diagram.
- Component diagram.

Structural model represents the framework for the system and this framework is the place where all other components exist. Hence, the class diagram, component diagram and deployment diagrams are part of structural modeling. They all represent the elements and the mechanism to assemble them.

The structural model never describes the dynamic behavior of the system. Class diagram is the most widely used structural diagram.

Behavioral Modeling: Behavioral model describes the interaction in the system. It represents the interaction among the structural diagrams. Behavioral modeling shows the dynamic nature of the system. They consist of the following −

- Activity diagrams
- Interaction diagrams
- Use case diagrams

All the above show the dynamic sequence of flow in a system.

Architectural Modeling: Architectural model represents the overall framework of the system. It contains both structural and behavioral elements of the system. Architectural model can be defined as the blueprint of the entire system. Package diagram comes under architectural modeling.

Q3. What is use cases and use case Diagram?

Ans. A use case describes a function that a system performs to achieve the user's goal. A use case must yield an observable result that is of value to the user of the system.

Use cases contain detailed information about the system, the system's users, relationships between the system and the users, and the required behavior of the system. Use cases do not describe the details of how the system is implemented.

Each use case describes a particular goal for the user and how the user interacts with the system to achieve that goal. The use case describes all possible ways that the system can achieve, or fail to achieve, the goal of the user.

You can use use cases for the following purposes:

- Determine the requirements of the system
- Describe what the system should do
- Provide a basis for testing to ensure that the system works as intended
- In models that depict businesses, use cases represent the processes and activities of the business. In models that depict software systems, use cases represent the capabilities of the software.
- Each use case must have a unique name that describes the action that the system performs. Use case names are often short phrases that start with a verb, such as Place Order Online.
- As the following figure illustrates, a use case is displayed as an oval that contains the name of the use case.

UseCase1

We can add the following features to use cases:

- Attributes that identify the properties of the objects in a use case
- Operations that describe the behavior of objects in a use case and how they affect the system
- Documentation that details the purpose and flow of events in a use case
- In UML, use-case diagrams model the behavior of a system and help to capture the requirements of the system.

Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors. The use cases and actors in use-case diagrams describe what the system does and how the actors use it, but not how the system operates internally.

Use-case diagrams illustrate and define the context and requirements of either an entire system or the important parts of the system. You can model a complex system with a single use-case diagram, or create many use-case diagrams to model the components of the system. You would typically develop use-case diagrams in the early phases of a project and refer to them throughout the development process.

Use-case diagrams are helpful in the following situations:

- Before starting a project, you can create use-case diagrams to model a business so that all participants in the project share an understanding of the workers, customers, and activities of the business.
- While gathering requirements, you can create use-case diagrams to capture the system requirements and to present to others what the system should do.
- During the analysis and design phases, you can use the use cases and actors from your use-case diagrams to identify the classes that the system requires.
- During the testing phase, you can use use-case diagrams to identify tests for the system.

The following topics describe model elements in use-case diagrams:

(1) **Use cases**: A use case describes a function that a system performs to achieve the user's goal. A use case must yield an observable result that is of value to the user of the system.

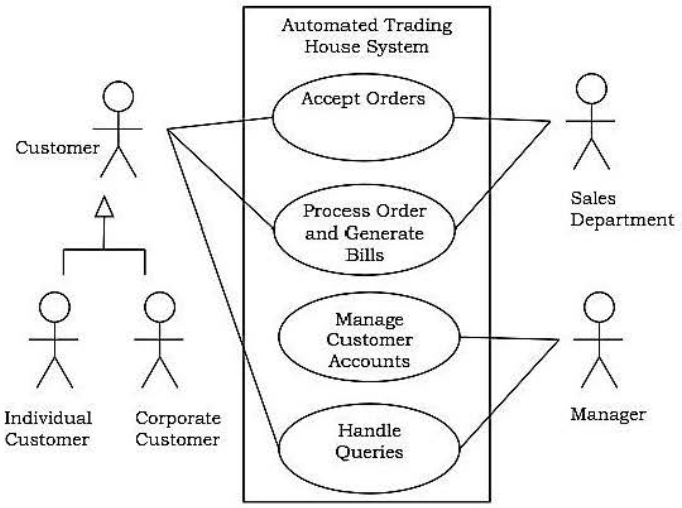
(2) **Actors:** An actor represents a role of a user that interacts with the system that you are modeling. The user can be a human user, an organization, a machine, or another external system.

(3) **Subsystems:** In UML models, subsystems are a type of stereotyped component that represent independent, behavioral units in a system. Subsystems are used in class, component, and use-case diagrams to represent large-scale components in the system that you are modeling.

(4) **Relationships in use-case diagrams**: In UML, a relationship is a connection between model elements. A UML relationship is a type of model element that adds semantics to a model by defining the structure and behavior between the model elements.

Example: Let us consider an Automated Trading House System. We assume the following features of the system –

- The trading house has transactions with two types of customers, individual customers and corporate customers.
- Once the customer places an order, it is processed by the sales department and the customer is given the bill.
- The system allows the manager to manage customer accounts and answer any queries posted by the customer.

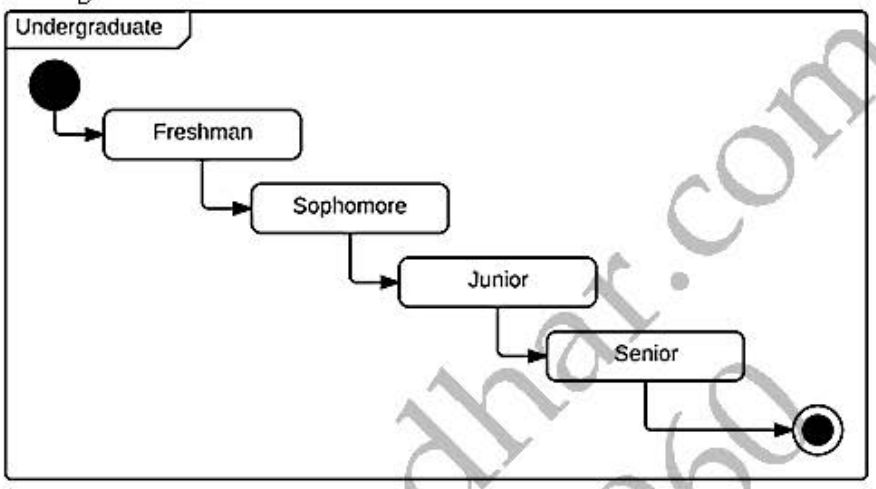


Q4. What is state Machines?

Ans. A state machine is any device that stores the status of an object at a given time and can change status or cause other actions based on the input it receives. States refer to the different combinations of information that an object can hold, not how the object behaves. In order to understand the different states of an object, you might want to visualize all of the possible states and show how an object gets to each state, and you can do so with a UML state diagram.

Each state diagram typically begins with a dark circle that indicates the initial state and ends with a bordered circle that denotes the final state. However, despite having clear start and end points, state diagrams are not necessarily the best tool for capturing an overall progression of events. Rather, they illustrate specific kinds of behavior—in particular, shifts from one state to another.

State diagrams mainly depict states and transitions. States are represented with rectangles with rounded corners that are labeled with the name of the state. Transitions are marked with arrows that flow from one state to another, showing how the states change.

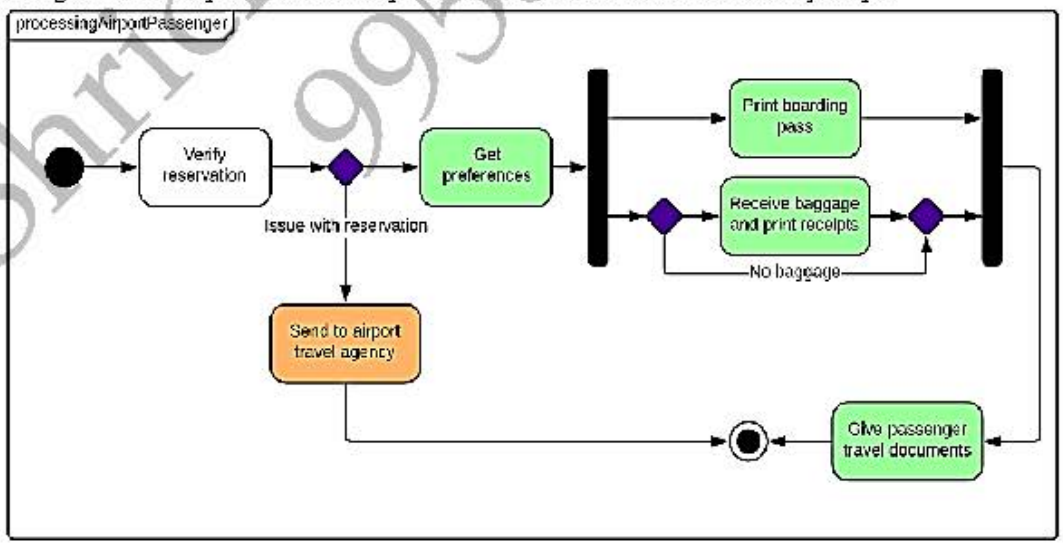


State diagram applications: Like most UML diagrams, state diagrams have several uses. The main applications are as follows:

- Depicting event-driven objects in a reactive system.
- Illustrating use case scenarios in a business context.
- Describing how an object moves through various states within its lifetime.
- Showing the overall behavior of a state machine or the behavior of a related set of state machines.

Example of State Diagram:

Airport check-in state diagram: The following example simplifies the steps required to check in at an airport. For airlines, a state diagram can help to streamline processes and eliminate unnecessary steps.



Q5. Explain the concept of architectural Modelling?

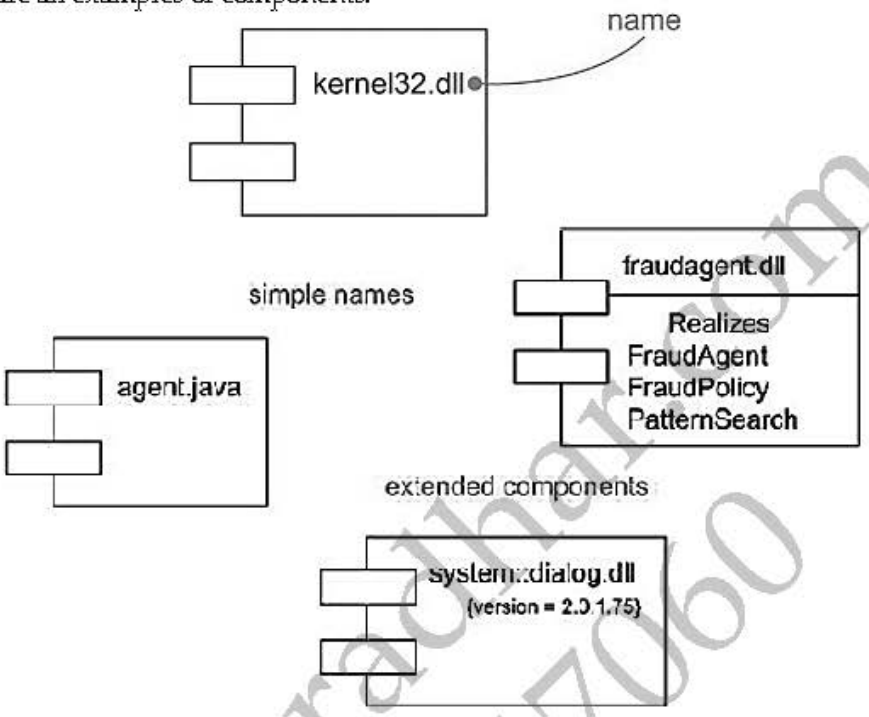
Ans. Architectural model represents the overall framework of the system. It contains both structural and behavioral elements of the system. Architectural model can be defined as the blueprint of the entire system. Package diagram comes under architectural modeling.

Components:

A component is a physical replaceable part of a system that complies with and provides the realization of a set of interfaces. We use components to model the physical things that may reside on a node, such as executables, libraries, tables, files and documents.

A component typically represents the physical packaging of otherwise logical elements such as classes, interfaces and collaborations. We do logical modeling to visualize, specify, and document our decisions about the vocabulary of our domain and the structural and behavioral way those things collaborate.

We do physical modeling to construct the executable system. Object libraries, executables, COM+ components and Enterprise Java Beans are all examples of components.

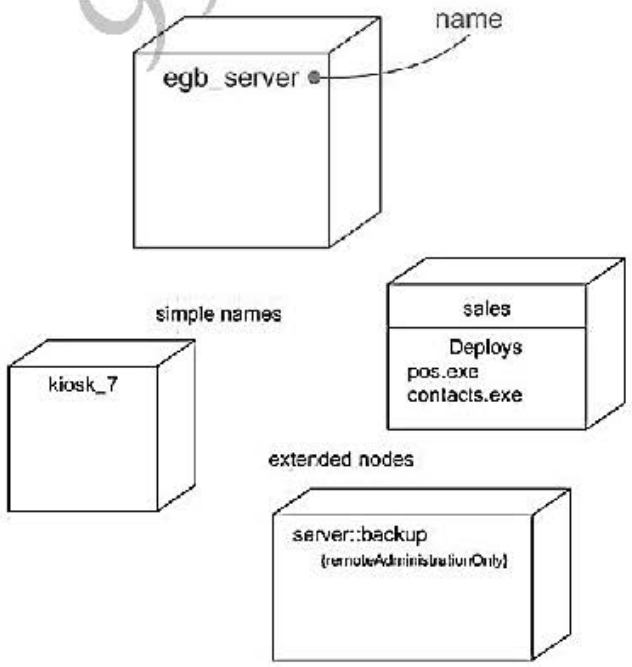


Deployment: A node is a physical element that exists at runtime and represents a computational resource, generally having atleast some memory and, often, processing capability.

We use nodes to model the topology of the hardware on which our system executes. A node typically represents a processor or a device on which components may be deployed.

When we architect a software-intensive system, we have to consider both its logical and physical dimensions. On the logical side, you'll find things such as classes, interfaces, collaborations, interactions and state machines. On the physical side you'll find components and nodes.

In UML, a node is represented as a cube as shown below. Using stereotypes we can tailor this notation to represent specific kinds of processors and devices.



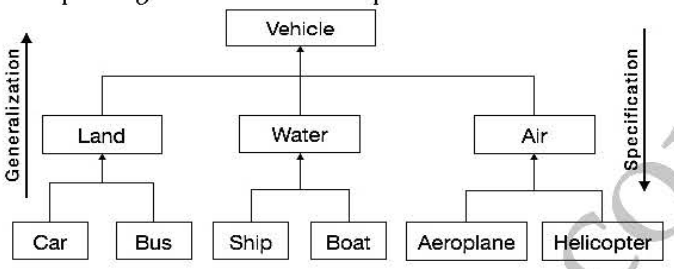
Q6. What is generalisation and Specialisation?

Ans. Generalization and specialization represent a hierarchy of relationships between classes, where subclasses inherit from super-classes.

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

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

The following figure shows an example of generalization and specialization.



Q7. Describe the concepts of advanced dynamic Modelling?

Ans. Entry and exit actions are features of advanced dynamic modelling. Actions can be associated with entering an exiting a state as an alternative to connecting them to transactions. An entry action is performed when any transition enters the state and an exit action is performed when a state is exited. This allows a state to be expressed in terms of matched entry and exit actions without regard to what happens before a state becomes active. An internal action does not change the state it executes within the state. Automatic transactions fire and change the state when their conditions are met and any activity in the current state is terminated. An object can send an event to another object together with an attribute. A race condition occurs when a state may accept events from more then one object. In this case the order of the events becomes important since it might affect the final state of the object.

Activity: Activity is an operation upon the states of an object that requires some time period. They are the ongoing executions within a system that can be interrupted. Activities are shown in activity diagrams that portray the flow from one activity to another.

Action: An action is an atomic operation that executes as a result of certain events. By atomic, it is meant that actions are un-interruptible, i.e., if an action starts executing, it runs into completion without being interrupted by any event. An action may operate upon an object on which an event has been triggered or on other objects that are visible to this object. A set of actions comprise an activity.

Entry and Exit Actions: Entry action is the action that is executed on entering a state, irrespective of the transition that led into it.

Likewise, the action that is executed while leaving a state, irrespective of the transition that led out of it, is called an exit action.

Scenario: Scenario is a description of a specified sequence of actions. It depicts the behavior of objects undergoing a specific action series. The primary scenarios depict the essential sequences and the secondary scenarios depict the alternative sequences.

Q8. Draw functional model for Wholesale Software?

Ans. Let us consider a software system, Wholesaler Software, that automates the transactions of a wholesale shop. The shop sells in bulks and has a clientele comprising of merchants and retail shop owners. Each customer is asked to register with his/her particulars and is given a unique customer code, C_Code. Once a sale is done, the shop registers its details and sends the goods for dispatch. Each year, the shop distributes Christmas gifts to its customers, which comprise of a silver coin or a gold coin depending upon the total sales and the decision of the proprietor.

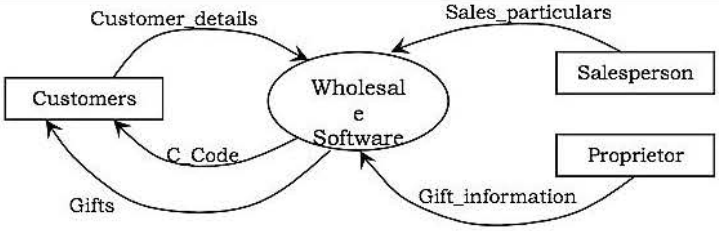
The functional model for the Wholesale Software is given below. The figure below shows the top-level DFD. It shows the software as a single process and the actors that interact with it.

The actors in the system are –

Customers

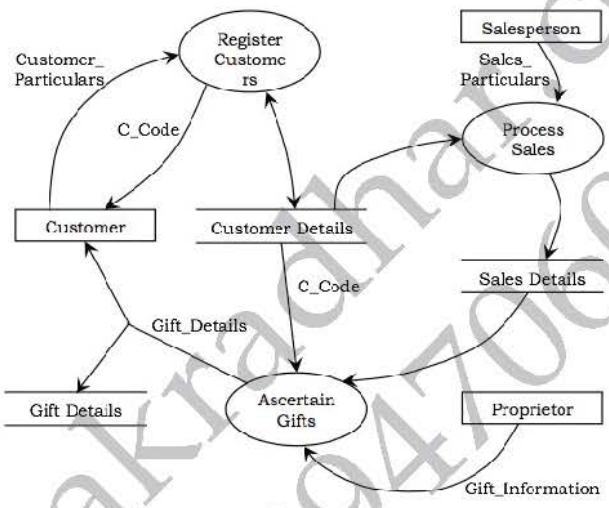
Salesperson

Proprietor



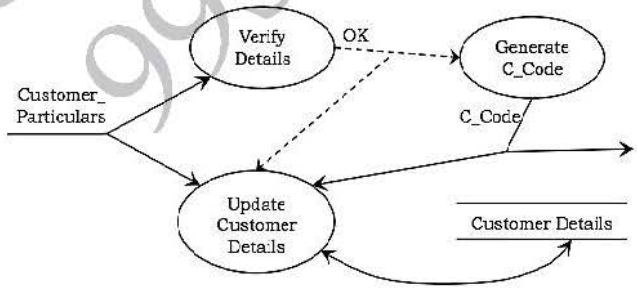
In the next level DFD, as shown in the following figure, the major processes of the system are identified, the data stores are defined and the interaction of the processes with the actors, and the data stores are established.
In the system, three processes can be identified, which are –

- Register Customers
- Process Sales
- Ascertain Gifts
- The data stores that will be required are –
- Customer Details
- Sales Details
- Gift Details



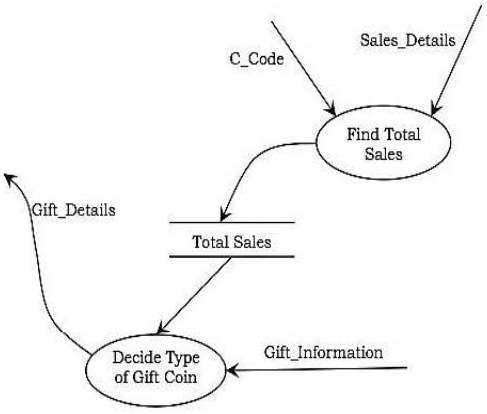
DFD of Wholesale Software

The following figure shows the details of the process Register Customer. There are three processes in it, Verify Details, Generate C_Code, and Update Customer Details. When the details of the customer are entered, they are verified. If the data is correct, C_Code is generated and the data store Customer Details is updated.



DFD of Customer Process

The following figure shows the expansion of the process Ascertain Gifts. It has two processes in it, Find Total Sales and Decide Type of Gift Coin. The Find Total Sales process computes the yearly total sales corresponding to each customer and records the data. Taking this record and the decision of the proprietor as inputs, the gift coins are allotted through Decide Type of Gift Coin process.



DFD of Gift Process

Q9. Describe Handling Boundary Conditions?

Ans. The system design phase needs to address the initialization and the termination of the system as a whole as well as each subsystem. The different aspects that are documented are as follows –

- The start–up of the system, i.e., the transition of the system from non-initialized state to steady state.
- The termination of the system, i.e., the closing of all running threads, cleaning up of resources, and the messages to be sent.
- The initial configuration of the system and the reconfiguration of the system when needed.
- Foreseeing failures or undesired termination of the system.

Boundary conditions are modelled using boundary use cases.

Q10. What is control as state within Program?

Ans. The term control literally means to check the effect of input within a program. In ATM, after the ATM card is inserted (as input) the control of the program is transferred to the next state (i.e., to request password state).

This is the traditional approach to represent control within a program. The location of control within a program implicitly defines the program state. Each state transition corresponds to an input statement. After input is read, the program branches depending on the input event produce some result. Each input statement handles any input value that could be received at that point. In case of highly nested procedural code, low-level procedures must accept inputs that may be passed to upper level procedures. After receiving input they pass them up through many levels of procedure calls. There must be some procedure prepared to handle these lower level calls. The technique of converting a state diagram to code is given as under:

- Identify all the main control paths. Start from the initial state; choose a path through the diagram that corresponds to the normally expected sequence of events. Write the names of states along the selected path as a linear sequence. This will be a sequence of statements in the program.
- Choose alternate paths that branch off the main path of the program and rejoin it later. These could be conditional statements in the program.
- Identify all backward paths that branch off the main loop of the program and rejoin it earlier. This could be the loop in the program. All non-intersecting backward paths become nested loops in the program.
- The states and transitions that remain unchecked correspond to exception conditions. These can be handled by applying several techniques, like error subroutines, exception handling supported by the language, or setting and testing of status flags. To understand control as a state within a program, let us take the state model for the ATM class, the state model of the ATM class and the pseudo code derived from it. In this process first, we choose the main path of control, this corresponds to the reading of a card querying the user for transaction information, processing the transaction, printing a receipt, and ejecting the card. If the customer wants to process for some alternates control that should be provided. For example, if the password entered by the customer is bad, then the customer is asked to try again

Pseudo code of ATM control: The pseudocode for the ATM is given as under:

- do forever
- display main screen
- read card
- repeat
- ask for password
- read password
- verify account

- until account verification is OK
- repeat
- repeat
- ask for type of transaction
- read type ask for amount
- read amount
- start transaction
- wait for it to complete
- until transaction is OK
- dispense cash
- wait for customer to take it
- ask whether to continue
- until user
- asks to terminate eject card
- wait for customer to take card.

MCS-219: Object Oriented Analysis and Design
Guess Paper-II

Q1. Describe Design Optimization?

Ans. The examination demonstrate catches the intelligent data about the system, while the outline show adds points of interest to help productive data get to. Prior to a plan is actualized, it ought to be upgraded in order to make the execution more effective. The point of streamlining is to limit the cost regarding time, space, and different measurements.

Be that as it may, outline improvement ought not be overabundance, as simplicity of usage, practicality, and extensibility are likewise imperative concerns. It is frequently observed that an impeccably streamlined outline is more productive yet less intelligible and reusable. So the architect must strike a harmony between the two.

The various things that may be done for design optimization are –

- Add redundant associations
- Omit non-usable associations
- Optimization of algorithms
- Save derived attributes to avoid re-computation of complex expressions
- Expansion of Redundant Associations

Addition of Redundant Associations: Amid plan streamlining, it is checked if determining new affiliations can diminish get to costs. In spite of the fact that these repetitive affiliations may not include any data, they may build the productivity of the general model.

Omission of Non-Usable Associations: Nearness of an excessive number of affiliations may render a framework unintelligible and consequently diminish the general productivity of the framework. Along these lines, amid advancement, all non-usable affiliations are evacuated.

Optimization of Algorithms: In protest situated frameworks, improvement of information structure and calculations are done in a community oriented way. Once the class configuration is set up, the operations and the calculations should be advanced.

Streamlining of calculations is acquired by –

- Improvement of the request of computational undertakings
- Inversion of execution request of circles from that set down in the useful model
- Evacuation of dead ways inside the calculation

Sparing and Storing of Derived Attributes: Determined characteristics are those traits whose esteems are registered as an element of different properties (base qualities). Re-calculation of the estimations of inferred traits each time they are required is a time– expending method. To maintain a strategic distance from this, the qualities can be processed and put away in their figured structures.

In any case, this may posture refresh inconsistencies, i.e., an adjustment in the estimations of base characteristics with no comparing change in the estimations of the determined properties. To stay away from this, the accompanying advances are taken –

With each refresh of the base characteristic esteem, the determined trait is additionally re-figured.

All the determined characteristics are re-registered and refreshed occasionally in a gathering as opposed to after each refresh.

Q2. What is Implementing Constraints?

Ans. Implementing Constraints: Constraints in classes restrict the range and type of values that the attributes may take. In order to implement constraints, a valid default value is assigned to the attribute when an object is instantiated from the class. Whenever the value is changed at runtime, it is checked whether the value is valid or not. An invalid value may be handled by an exception handling routine or other methods.

Example: Consider an Employee class where age is an attribute that may have values in the range of 18 to 60. The following C++ code incorporates it –

```
class Employee {
  private: char * name;
  int age;
  // other attributes
  public:
  Employee() {              // default constructor
```

```
    strcpy(name, "");
    age = 18;              // default value
  }
  class AgeError {};       // Exception class
  void changeAge( int a) {  // method that changes age
    if ( a < 18 || a > 60 )  // check for invalid condition
    throw AgeError();      // throw exception
    age = a;
  }};
```

Q3. What is DBMS?

Ans. The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently. It is also used to organize the data in the form of a table, schema, views, and reports, etc.

For example: The college Database organizes the data about the admin, staff, students and faculty etc.

Using the database, you can easily retrieve, insert, and delete the information.

Database Management System

- Database management system is a software which is used to manage the database. For example: MySQL, Oracle, etc are a very popular commercial database which is used in different applications.
- DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.
- It provides protection and security to the database. In the case of multiple users, it also maintains data consistency.

DBMS allows users the following tasks:

- **Data Definition:** It is used for creation, modification, and removal of definition that defines the organization of data in the database.
- **Data Updation:** It is used for the insertion, modification, and deletion of the actual data in the database.
- **Data Retrieval:** It is used to retrieve the data from the database which can be used by applications for various purposes.
- **User Administration:** It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

Characteristics of DBMS

- It uses a digital repository established on a server to store and manage the information.
- It can provide a clear and logical view of the process that manipulates data.
- DBMS contains automatic backup and recovery procedures.
- It contains ACID properties which maintain data in a healthy state in case of failure.
- It can reduce the complex relationship between data.
- It is used to support manipulation and processing of data.
- It is used to provide security of data.
- It can view the database from different viewpoints according to the requirements of the user.

Advantages of DBMS:

- **Controls database redundancy:** It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.
- **Data sharing:** In DBMS, the authorized users of an organization can share the data among multiple users.
- **Easily Maintenance:** It can be easily maintainable due to the centralized nature of the database system.
- **Reduce time:** It reduces development time and maintenance need.
- **Backup:** It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.
- **multiple user interface:** It provides different types of user interfaces like graphical user interfaces, application program interfaces

Disadvantages of DBMS:

- **Cost of Hardware and Software:** It requires a high speed of data processor and large memory size to run DBMS software.

- **Size:** It occupies a large space of disks and large memory to run them efficiently.
- **Complexity:** Database system creates additional complexity and requirements.
- **Higher impact of failure:** Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

Q4. Differentiate between RDBMS and DBMS?

Ans.

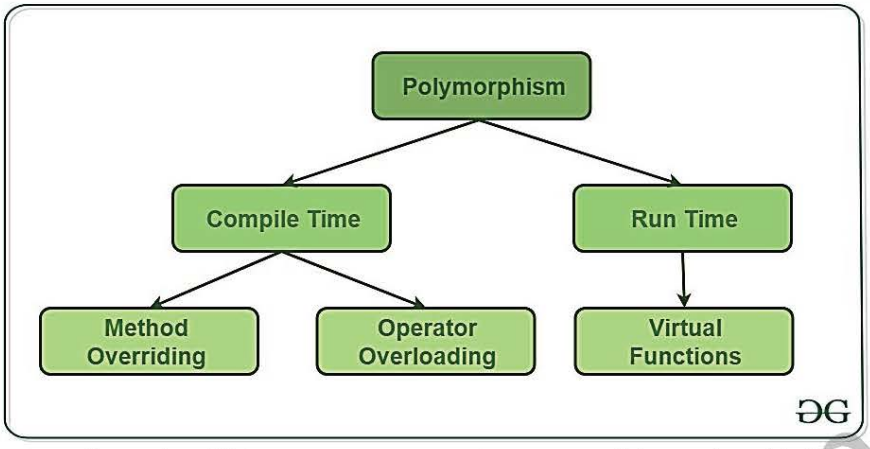
DBMS	RDBMS
DBMS stores data as file.	RDBMS stores data in tabular form.
Data elements need to access individually.	Multiple data elements can be accessed at the same time.
No relationship between data.	Data is stored in the form of tables which are related to each other.
Normalization is not present.	Normalization is present.
DBMS does not support distributed database.	RDBMS supports distributed database.
It stores data in either a navigational or hierarchical form.	It uses a tabular structure where the headers are the column names, and the rows contain corresponding values.
It deals with small quantity of data.	It deals with large amount of data.
Data redundancy is common in this model.	Keys and indexes do not allow Data redundancy.
It is used for small organization and deal with small data.	It is used to handle large amount of data.
It supports single user.	It supports multiple users.
Data fetching is slower for the large amount of data.	Data fetching is fast because of relational approach.
The data in a DBMS is subject to low security levels with regards to data manipulation.	There exists multiple levels of data security in a RDBMS.
Low software and hardware necessities.	Higher software and hardware necessities.
Examples: XML, Window Registry, etc.	Examples: MySQL, PostgreSQL, SQL Server, Oracle, Microsoft Access etc.

Q5. Describe Polymorphism?

Ans. Polymorphism is the ability of any data to be processed in more than one form. The word itself indicates the meaning as poly means many and morphism means types. Polymorphism is one of the most important concept of object oriented programming language. The most common use of polymorphism in object-oriented programming occurs when a parent class reference is used to refer to a child class object. Here we will see how to represent any function in many types and many forms.

Real life example of polymorphism, a person at the same time can have different roles to play in life. Like a woman at the same time is a mother, a wife, an employee and a daughter. So the same person has to have many features but has to implement each as per the situation and the condition. Polymorphism is considered as one of the important features of Object Oriented Programming.

Polymorphism is the key power of object-oriented programming. It is so important that languages that don't support polymorphism cannot advertise themselves as Object-Oriented languages. Languages that possess classes but have no ability of polymorphism are called object-based languages. Thus it is very vital for an object-oriented programming language. It is the ability of an object or reference to take many forms in different instances. It implements the concept of function overloading, function overriding and virtual functions.



Polymorphism is a property through which any message can be sent to objects of multiple classes, and every object has the tendency to respond in an appropriate way depending on the class properties.
This means that polymorphism is the method in an object-oriented programming language that does different things depending on the class of the object which calls it. For example, $square->area() will return the area of a square, but $triangle->area() might return the area of a triangle. On the other hand, $object->area() would have to calculate the area according to which class $object was called.

Q6. What is building blocks of an UML?

Ans. The conceptual model of UML can be mastered by learning the following three major elements –

- UML building blocks
- Rules to connect the building blocks
- Common mechanisms of UML
- The building blocks of UML can be defined as –
- Things
- Relationships
- Diagrams
- Things

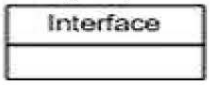
Things are the most important building blocks of UML. Things can be –

- Structural
- Behavioral
- Grouping
- Annotational
- Structural Things

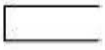
Structural things define the static part of the model. They represent the physical and conceptual elements. Following are the brief descriptions of the structural things.

Class – Class represents a set of objects having similar responsibilities.

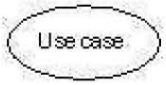
Interface – Interface defines a set of operations, which specify the responsibility of a class.



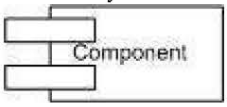
Collaboration –Collaboration defines an interaction between elements.



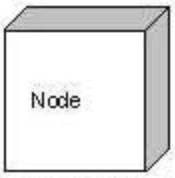
Use case –Use case represents a set of actions performed by a system for a specific goal.



Component –Component describes the physical part of a system.

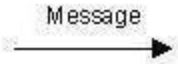


Node – A node can be defined as a physical element that exists at run time.



Behavioral Things: A behavioral thing consists of the dynamic parts of UML models. Following are the behavioral things –

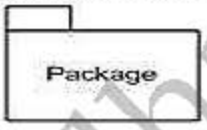
Interaction: Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.



State machine: State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change

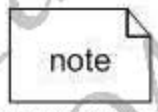
Grouping Things: Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available –

Package – Package is the only one grouping thing available for gathering structural and behavioral things.



Annotational Things: Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements.

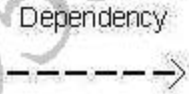
Note - It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element.



Relationship: Relationship is another most important building block of UML. It shows how the elements are associated with each other and this association describes the functionality of an application.

There are four kinds of relationships available.

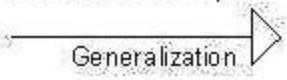
Dependency: Dependency is a relationship between two things in which change in one element also affects the other.



Association: Association is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.

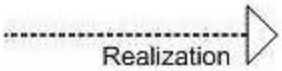
Generalization

Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the world of objects.



Realization

Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces.



UML Diagrams: UML diagrams are the ultimate output of the entire discussion. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system.

The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it complete.

UML includes the following nine diagrams, the details of which are described in the subsequent chapters.

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Statechart diagram
- Deployment diagram
- Component diagram

Q7. What is behavioural Modelling?

Ans. Behavioral (or Dynamic) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams, and state machine diagrams.

UML's five behavioral diagrams are used to visualize, specify, construct, and document the dynamic aspects of a system. It shows how the system behaves and interacts with itself and other entities (users, other systems). They show how data moves through the system, how objects communicate with each other, how the passage of time affects the system, or what events cause the system to change internal states. Since behavior diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems. As an example, the activity diagram describes the business and operational step-by-step activities of the components in a system.

In other words, a behavioral diagram shows how the system works 'in motion', that is how the system interacts with external entities and users, how it responds to input or event and what constraints it operates under.

There are seven behavioral diagrams that you can model the dynamics of a system as listed in the Table below:

Behavioral Diagram	**Brief Description**
Activity Diagram	It is a graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency
Use Case Diagram	It describes a system's functional requirements in terms of use cases that enable you to relate what you need from a system to how the system delivers on those needs.
State Machine Diagram	It shows the discrete behavior of a part of a designed system through finite state transitions.
Sequence Diagram	It shows the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.
Communication Diagram	It shows interactions between objects and/or parts (represented as lifelines) using sequenced messages in a free-form arrangement.
Interaction Overview Diagram	It depicts a control flow with nodes that can contain other interaction diagrams.
Timing Diagram	It shows interactions when the primary purpose of the diagram is to reason about time by focusing on conditions changing within and among lifelines along a linear time axis.

Q8. Difference between process and Threads.

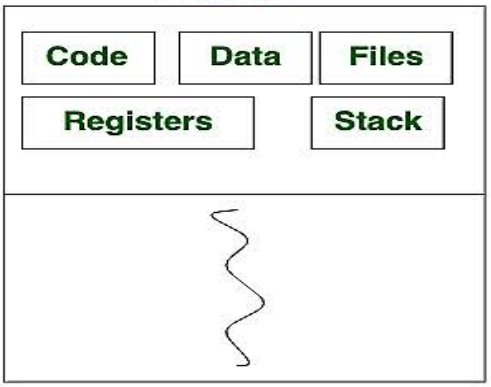
Ans. Process: Process are basically the programs which are dispatched from the ready state and are scheduled in the CPU for execution. PCB(Process Control Block) holds the concept of process. A process can create other processes which are known as Child Processes. The process takes more time to terminate and it is isolated means it does not share the memory with any other process.

The process can have the following states like new, ready, running, waiting, terminated, suspended.

Thread: Thread is the segment of a process means a process can have multiple threads and these multiple threads are contained within a process. A thread has three states: Running, Ready, and Blocked.

Thread takes less time to terminate as compared to process but unlike process threads do not isolate.

Process



Thread

S.NO	Process	Thread
1.	Process means any program is in execution.	Thread means segment of a process.
2.	Process takes more time to terminate.	Thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	Process is less efficient in term of communication.	Thread is more efficient in term of communication.
6.	Multi programming holds the concepts of multi process.	We don't need multi programs in action for multiple threads because a single process consists of multiple threads.
7.	Process is isolated.	Threads share memory.
8.	Process is called heavy weight process.	A Thread is lightweight as each thread in a process shares code, data and resources.
9.	Process switching uses interface in operating system.	Thread switching does not require to call a operating system and cause an interrupt to the kernel.
10.	If one process is blocked then it will not effect the execution of other process	Second thread in the same task could not run, while one server thread is blocked.
11.	Process has its own Process Control Block, Stack and Address Space.	Thread has Parents' PCB, its own Thread Control Block and Stack and common Address space.

S.NO	Process	Thread
12.	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
13.	Changes to the parent process does not affect child processes.	Since all threads of the same process share address space and other resources so any changes to the main thread may affect the behavior of the other threads of the process.

Q9. Describe integrity constraints in object modelling?

Ans. Integrity constraints provide a way of ensuring that changes made to the database by authorized users do not result in a loss of data consistency.

We saw a form of integrity constraint with E-R models:-

- **key declarations:** stipulation that certain attributes form a candidate key for the entity set.
- **form of a relationship:** mapping cardinalities 1-1, 1-many and many-many.

An integrity constraint can be any arbitrary predicate applied to the database. They may be costly to evaluate, so we will only consider integrity constraints that can be tested with minimal overhead. Often we wish to ensure that a value appearing in a relation for a given set of attributes also appears for another set of attributes in another relation. This is called referential integrity.

Triggering Operation Integrity Rules: Triggering operation integrity rules govern insert, delete, update and retrieval validity. These rules involve the effects of operations on other classes, or on other attributes within class, and include domains, and insert/delete, and other attribute within a class, and include domains and insert/delete and other attributes business rules.

Triggering operation constraints involve:

- Attributes across multiple classes or instances
- Two or more attributes within a class
- One attribute or class and an external parameter.

Example triggering constraints include:

- An employee may only save up to three weeks time off
- A customer may not exceed a predetermined credit limit
- All customer invoices must include at least one line items
- Order dates must be current, or future dates.

Triggering operations have two components. These are:-

- The event or condition that causes an operation to execute
- The action set in motion by the event or condition.

Q10. Explain state and state Transitions?

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

Parts of a state

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

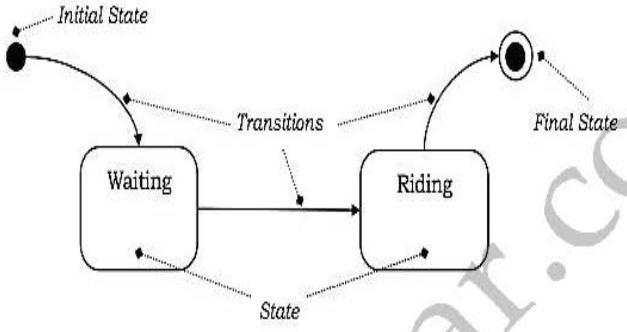
Initial and Final States: The default starting state of an object is called its initial state. The final state indicates the completion of execution of the state machine. The initial and the final states are pseudo-states, and may not have the parts of a regular state except name. In state transition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

Transition: A transition denotes a change in the state of an object. If an object is in a certain state when an event occurs, the object may perform certain activities subject to specified conditions and change the state. In this case, a

state–transition is said to have occurred. The transition gives the relationship between the first state and the new state. A transition is graphically represented by a solid directed arc from the source state to the destination state.
The five parts of a transition are –

- **Source State:** The state affected by the transition.
- **Event Trigger:** The occurrence due to which an object in the source state undergoes a transition if the guard condition is satisfied.
- **Guard Condition:** A Boolean expression which if True, causes a transition on receiving the event trigger.
- **Action:** An un-interruptible and atomic computation that occurs on the source object due to some event.
- **Target State:** The destination state after completion of transition.

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



MCS-219: Object Oriented Analysis and Design
Guess Paper-III

Q1. What is DFD?

Ans. Functional Modelling is represented through a hierarchy of DFDs. The DFD is a graphical representation of a system that shows the inputs to the system, the processing upon the inputs, the outputs of the system as well as the internal data stores. DFDs illustrate the series of transformations or computations performed on the objects or the system, and the external controls and objects that affect the transformation.

Rumbaugh et al. have defined DFD as, "A data flow diagram is a graph which shows the flow of data values from their sources in objects through processes that transform them to their destinations on other objects."

The four main parts of a DFD are –

- Processes,
- Data Flows,
- Actors, and
- Data Stores.

The other parts of a DFD are –

- Constraints, and
- Control Flows.

Q2. What is concurrency Identification?

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

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

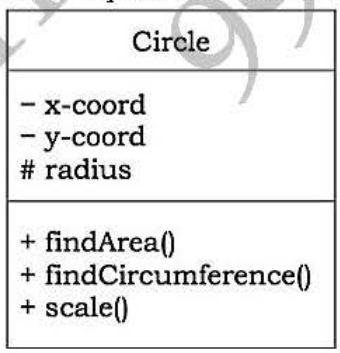
Concurrency is associated with the problems of data integrity, deadlock, and starvation. So a clear strategy needs to be made whenever concurrency is required. Besides, concurrency requires to be identified at the design stage itself, and cannot be left for implementation stage.

Q3. Explain the Implementation Strategies?

Ans. Implementing an object-oriented design generally involves using a standard object oriented programming language (OOPL) or mapping object designs to databases. In most cases, it involves both.

Implementation using Programming Languages: Usually, the task of transforming an object design into code is a straightforward process. Any object-oriented programming language like C++, Java, Smalltalk, C# and Python, includes provision for representing classes. In this chapter, we exemplify the concept using C++.

The following figure shows the representation of the class Circle using C++.




```
class Circle
{
   private:
     double x_coord;
     double y_coord;
   protected:
     double radius;
   public:
     double findArea();
     double findCircumference();
     void scale();
};
```

Implementing Associations: Most programming languages do not provide constructs to implement associations directly. So the task of implementing associations needs considerable thought.

Associations may be either unidirectional or bidirectional. Besides, each association may be either one–to–one, one–to–many, or many–to–many.

Unidirectional Associations: For implementing unidirectional associations, care should be taken so that unidirectionality is maintained. The implementations for different multiplicity are as follows –

Optional Associations: Here, a link may or may not exist between the participating objects. For example, in the association between Customer and Current Account in the figure below, a customer may or may not have a current account.

For implementation, an object of Current Account is included as an attribute in Customer that may be NULL. Implementation using C++ –

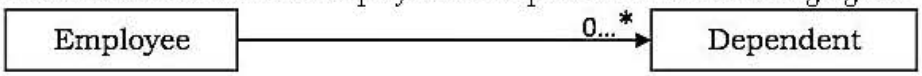
```
class Customer {
  private:
  // attributes
  Current_Account c; //an object of Current_Account as attribute
  public:
  Customer() {
    c = NULL;
  } // assign c as NULL

  Current_Account getCurrAc() {
    return c;
  }
  void setCurrAc( Current_Account myacc) {
    c = myacc;
  }
  void removeAcc() {
    c = NULL;
  }
};
```

One–to–one Associations – Here, one instance of a class is related to exactly one instance of the associated class. For example, Department and Manager have one–to–one association as shown in the figure below.

This is implemented by including in Department, an object of Manager that should not be NULL. Implementation using C++ –

```
class Department {
  private:
  // attributes
  Manager mgr; //an object of Manager as attribute
  public:
  Department (/*parameters*/, Manager m) { //m is not NULL
    // assign parameters to variables
    mgr = m;
  }
  Manager getMgr() {
    return mgr;
  }
};
```

One–to–many Associations – Here, one instance of a class is related to more than one instances of the associated class. For example, consider the association between Employee and Dependent in the following figure.



This is implemented by including a list of Dependents in class Employee. Implementation using C++ STL list container –

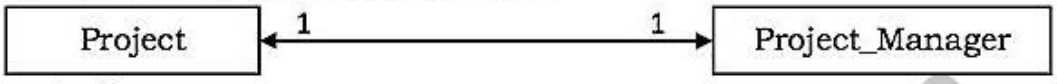
```
class Employee {
  private:
  char * deptName;
  list <Dependent> dep; //a list of Dependents as attribute
  public:
  void addDependent ( Dependent d) {
    dep.push_back(d);
  } // adds an employee to the department
```

```
  void removeDeoendent( Dependent d) {
    int index = find ( d, dep );
    // find() function returns the index of d in list dep
    dep.erase(index);
  }
};
```

Bi-directional Associations

To implement bi-directional association, links in both directions require to be maintained.

Optional or one–to–one Associations – Consider the relationship between Project and Project Manager having one–to–one bidirectional association as shown in the figure below.



Implementation using C++ –

```
Class Project {
  private:
  // attributes
  Project_Manager pmgr;
  public:
  void setManager ( Project_Manager pm);
  Project_Manager changeManager();
};
class Project_Manager {
  private:
  // attributes
  Project pj;
  public:
  void setProject(Project p);
  Project removeProject();
};
```

One–to–many Associations – Consider the relationship between Department and Employee having one–to–many association as shown in the figure below.

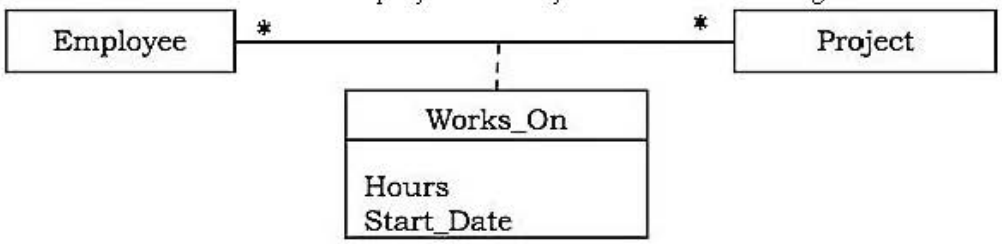
Implementation using C++ STL list container

```
class Department {
  private:
  char * deptName;
  list <Employee> emp; //a list of Employees as attribute
  public:
  void addEmployee ( Employee e) {
    emp.push_back(e);
  } // adds an employee to the department
  void removeEmployee( Employee e) {
    int index = find ( e, emp );
    // find function returns the index of e in list emp
    emp.erase(index);
  }
};
class Employee {
  private:
  //attributes
  Department d;
  public:
  void addDept();
  void removeDept();
```

};

Implementing Associations as Classes

If an association has some attributes associated, it should be implemented using a separate class. For example, consider the one–to–one association between Employee and Project as shown in the figure below.



Q4. Explain mapping design to code?

Ans. The interaction diagram and the class diagrams can be used as input to the code generation process. The implementation model is a model. Which consists of several implementation artifacts such as source code, database definition, HTML pages and so on.

Various object oriented languages such as java, c++, c#, small talk, python and so on can be used as the languages of implementation. Following are the approaches used for generating code from the design.

Defining class with methods and attributes from the class diagram.

The class diagram consists of classes, interfaces, superclasses, methods and attributes. These elements are sufficient to create a basic class definition of any object oriented languages like java. Mapping for methods and attributes signature to the code from the class diagram is straight forwarded. It is shown in the below code.

- public class ATMSystem{
- private int ATMID;
- private string BankName;
- private CardReader cardReader;
- private CashDispenser cashDispenser;
- private Console console;
- private MessageLog messageLog;
- private int SystemState;
- private NetworkToBank networkToBank;
- public void ATM(object BankName, object Address){
- .
- .
- }
- public void switchon(){...}
- public void switchoff(){...}
- public void startup(){...}
- }
- public class CardReader{
- private Object ATMID;
- public void CardReader(){..}
- public void Readard(){..}
- public void EjectCard(){..}
- public void RetainCard(){...}
- }
- public class Card{
- private Object Number;
- public void Card(Object int number){
- .
- .
- }
- public void getCardNo(){...}
- }

Mapping concepts

Forward engineering: Forward Engineering is a method of creating or making an application with the help of the given requirements. Forward engineering is also known as Renovation and Reclamation. Forward engineering requires high proficiency skills. It takes more time to construct or develop an application.

Reverse engineering: Reverse Engineering is also known as backward engineering, is the process of forward engineering in reverse. In this, the information is collected from the given or existing application. It takes less time than forward engineering to develop an application. In reverse engineering, the application is broken to extract knowledge or its architecture.

Model transformation: Model transformation is the technology that is used in the area of MDE to convert models to other software artifacts

Refactoring: It is the process of restructuring the existing component code without changing its external behavior. It is intended to improve the non-functional attributes of the software.

Round tripping = forward engineering + reverse engineering

Other mapping activities:

- Optimizing object design model
- Mapping associations
- Mapping contracts to exceptions
- Mapping object models to tables.

Q5. Explain the concept of RDBMS?

Ans. RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

Table: The data in an RDBMS is stored in database objects which are called as tables. This table is basically a collection of related data entries and it consists of numerous columns and rows.

Remember, a table is the most common and simplest form of data storage in a relational database. The following program is an example of a CUSTOMERS table –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Field: Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to maintain specific information about every record in the table.

Record or a Row: A record is also called as a row of data is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table –

```
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
+----+----------+-----+-----------+----------+
```

A record is a horizontal entity in a table.

Column: A column is a vertical entity in a table that contains all information associated with a specific field in a table. For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would be as shown below –

```
+-----------+
| ADDRESS   |
+-----------+
| Ahmedabad |
| Delhi     |
| Kota      |
| Mumbai    |
| Bhopal    |
```

| MP |
| Indore |
+---+---+

NULL value: A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is the one that has been left blank during a record creation.

SQL Constraints: Constraints are the rules enforced on data columns on a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints can either be column level or table level. Column level constraints are applied only to one column whereas, table level constraints are applied to the entire table.

Following are some of the most commonly used constraints available in SQL –

- NOT NULL Constraint – Ensures that a column cannot have a NULL value.
- DEFAULT Constraint – Provides a default value for a column when none is specified.
- UNIQUE Constraint – Ensures that all the values in a column are different.
- PRIMARY Key – Uniquely identifies each row/record in a database table.
- FOREIGN Key – Uniquely identifies a row/record in any another database table.
- CHECK Constraint – The CHECK constraint ensures that all values in a column satisfy certain conditions.
- INDEX – Used to create and retrieve data from the database very quickly.

Data Integrity: The following categories of data integrity exist with each RDBMS –

- **Entity Integrity:** There are no duplicate rows in a table.
- **Domain Integrity:** Enforces valid entries for a given column by restricting the type, the format, or the range of values.
- **Referential integrity:** Rows cannot be deleted, which are used by other records.
- **User-Defined Integrity:** Enforces some specific business rules that do not fall into entity, domain or referential integrity.

Database Normalization: Database normalization is the process of efficiently organizing data in a database. There are two reasons of this normalization process:-

- Eliminating redundant data, for example, storing the same data in more than one table.
- Ensuring data dependencies make sense.

Both these reasons are worthy goals as they reduce the amount of space a database consumes and ensures that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure.

Normalization guidelines are divided into normal forms; think of a form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure, so that it complies with the rules of first normal form, then second normal form and finally the third normal form.

It is a matter of choice to take it further and go to the fourth normal form, fifth normal form and so on, but in general, the third normal form is more than enough.

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

Q6. Explain binary Relationships in Tables?

Ans. A Binary Relationship is the relationship between two different Entities i.e. it is a relationship of role group of one entity with the role group of another entity.

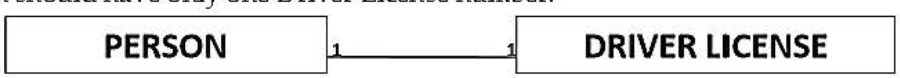
There are three types of cardinalities for Binary Relationships:

(1) **One-to-One:** Here one role group of one entity is mapped to one role group of another entity. In simple terms one instance of one entity is mapped with only one instance of another entity. In this type the primary key of one entity must be available as foreign key in other entity.

For example: consider two entities Person and Driver_License.

Person has the information about an individual and Driver_License has information about the Driving License for an individual. The relationship from Driver_License to Person is optional as not all People have driving license. Whereas the relationship from Person to Driver_License is compulsory i.e. every instance of Driver_License must be related to one Person.

One Person should have only one Driver License number.



(2) **One-to-Many:** One role group of one entity is mapped with many role groups of second entity and one role group of second entity is mapped with one role group of first entity.

For Example: consider two entities Project and Employee.
One Project can have many Employees working on it but one Employee will always be engaged in only one Project.

(3) **Many-to-Many:** One role group of one entity is mapped with many role groups of second entity and one role group of second entity is mapped with many role groups of first entity. In these kind of relationships a third table is always associated that defines the relationship between the two entities.
For example: Consider two entities Student and Books.
Many Students can have a Book and many Books can be issued to a Student so in this way this is a many-to-many relationship.
Now in between there would be a third relation Book_Issue that defines the relationship between Student and Book entities. It will contain the information for every Student that is issued a Book and for how many days i.e. it will keep track for all the Books issued.

Q7. What is object Orientation?

Ans. In the object-oriented approach, the focus is on capturing the structure and behavior of information systems into small modules that combines both data and process. The main aim of Object Oriented Design (OOD) is to improve the quality and productivity of system analysis and design by making it more usable.
In analysis phase, OO models are used to fill the gap between problem and solution. It performs well in situation where systems are undergoing continuous design, adaption, and maintenance. It identifies the objects in problem domain, classifying them in terms of data and behaviour.

The OO model is beneficial in the following ways –

- It facilitates changes in the system at low cost.
- It promotes the reuse of components.
- It simplifies the problem of integrating components to configure large system.
- It simplifies the design of distributed systems.

Elements of Object-Oriented System:

- **Objects:** An object is something that is exists within problem domain and can be identified by data (attribute) or behavior. All tangible entities (student, patient) and some intangible entities (bank account) are modeled as object.
- **Attributes:** They describe information about the object.
- **Behavior:** It specifies what the object can do. It defines the operation performed on objects.
- **Class**: A class encapsulates the data and its behavior. Objects with similar meaning and purpose grouped together as class.
- **Methods**: Methods determine the behavior of a class. They are nothing more than an action that an object can perform.
- **Message**: A message is a function or procedure call from one object to another. They are information sent to objects to trigger methods. Essentially, a message is a function or procedure call from one object to another.

Features of Object-Oriented System: An object-oriented system comes with several great features which are discussed below.

Encapsulation: Encapsulation is a process of information hiding. It is simply the combination of process and data into a single entity. Data of an object is hidden from the rest of the system and available only through the services of the class. It allows improvement or modification of methods used by objects without affecting other parts of a system.

Abstraction: It is a process of taking or selecting necessary method and attributes to specify the object. It focuses on essential characteristics of an object relative to perspective of user.

Relationships: All the classes in the system are related with each other. The objects do not exist in isolation, they exist in relationship with other objects.

There are three types of object relationships

- **Aggregation:** It indicates relationship between a whole and its parts.
- **Association:** In this, two classes are related or connected in some way such as one class works with another to perform a task or one class acts upon other class.
- **Generalization:** The child class is based on parent class. It indicates that two classes are similar but have some differences.

Inheritance: Inheritance is a great feature that allows to create sub-classes from an existing class by inheriting the attributes and/or operations of existing classes.

Polymorphism and Dynamic Binding: Polymorphism is the ability to take on many different forms. It applies to both objects and operations. A polymorphic object is one who true type hides within a super or parent class.

In polymorphic operation, the operation may be carried out differently by different classes of objects. It allows us to manipulate objects of different classes by knowing only their common properties.

Q8. Explain the concept of advanced class in UML?

Ans. A classifier is a mechanism which describes structural and behavioral features in a system. Other classifiers in UML are: interfaces, datatypes, signals, components, nodes, use cases and subsystems.

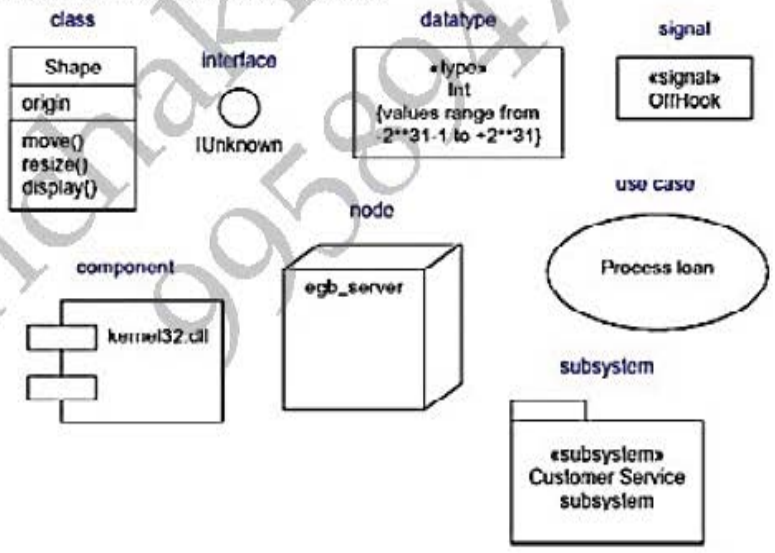
Classifiers

A classifier is a mechanism which describes structural and behavioral features. Class is the frequently used classifier. Every classifier represents structural aspects in terms of properties and behavioral aspects in terms of operations. Beyond these basic features, there are several advanced features like multiplicity, visibility, signatures, polymorphism and others.

In general, all the modeling elements that can have instances are called as classifiers. For example, classes have instances known as objects. Some elements like packages and a generalization does not have instances. Classifiers in UML are as follows:

Classifier	Description
Class	A class is a blueprint or a template for its objects
Interface	An interface is a collection of operations that must be realized by another element
Datatype	A type whose values have no identity
Signal	An asynchronous event communicated between objects
Component	Physical replaceable part of the system
Node	A physical existing at runtime which represents a computational resource
Use case	Collection of actions that a role performs on the system or the system does for a user.
Sub system	A grouping of elements which specify the behavior of a part of the system

The graphical representation of the classifiers is as follows:



Let's look at the advanced features (specific information that can be represented) of classes in UML.

Visibility: One of the important features that can be applied to the classifier's attributes and operations is the visibility. The visibility feature specifies whether a classifier can be used or accessed by the other classifiers. The four access specifiers in UML are as shown below:

Symbol	Name	Description
+	Public	Accessible by all other classifiers
#	Protected	Accessible only by the descendents and the classifier itself
~	Package	Accessible by all the classifiers with the same package
-	Private	Accessible only by the classifier in which they are available

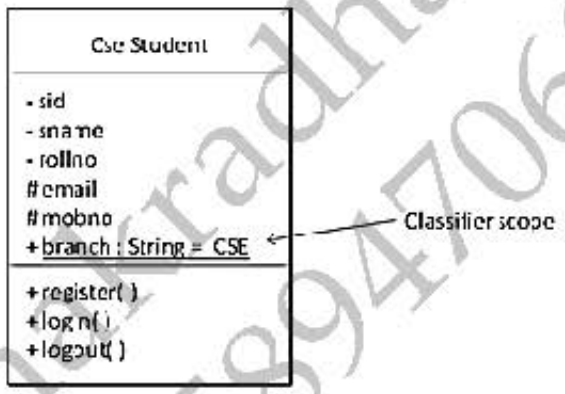
Example:

Scope: Another feature that can be applied to the classifier's attributes and operations is the scope. The scope of an attribute or an operation denotes whether they have their existence in all the instances of the classifier or only one copy is available and is shared across all the instances of the classifier. The scope specifiers in UML are:

Scope	Description
instance	Each instance of the classifier contains a value of the feature
classifier	One copy of the feature's value is shared by all the instances of a classifier

A classifier scoped feature is graphically represented by underlining the feature within the classifier. All other normal features are treated as instanced scope.

Example:



Q9. Explain interaction diagrams?

Ans. Interaction diagrams depict interactions of objects and their relationships. They also include the messages passed between them. There are two types of interaction diagrams –

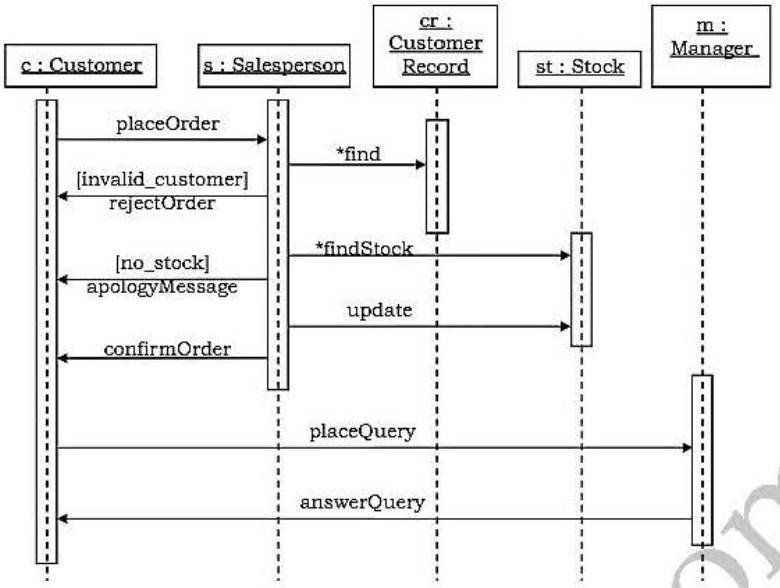
- Sequence Diagrams
- Collaboration Diagrams

Interaction diagrams are used for modeling –

- the control flow by time ordering using sequence diagrams.
- the control flow of organization using collaboration diagrams.

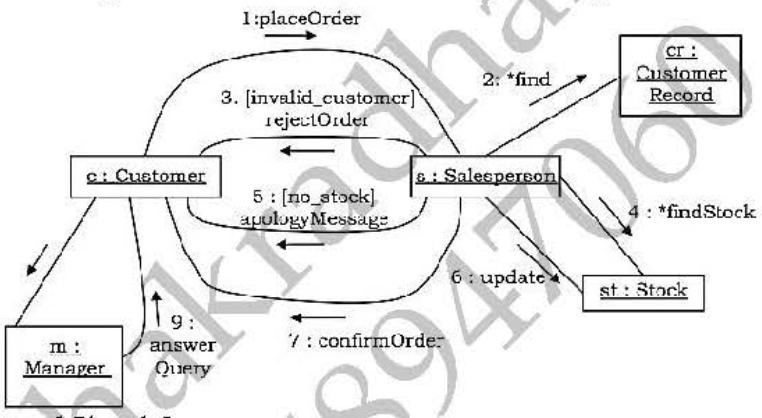
Sequence Diagrams: Sequence diagrams are interaction diagrams that illustrate the ordering of messages according to time.

- **Notations**: These diagrams are in the form of two-dimensional charts. The objects that initiate the interaction are placed on the x–axis. The messages that these objects send and receive are placed along the y–axis, in the order of increasing time from top to bottom.
- **Example**: A sequence diagram for the Automated Trading House System is shown in the following figure.



Collaboration Diagrams: Collaboration diagrams are interaction diagrams that illustrate the structure of the objects that send and receive messages.

- **Notations:** In these diagrams, the objects that participate in the interaction are shown using vertices. The links that connect the objects are used to send and receive messages. The message is shown as a labeled arrow.
- **Example:** Collaboration diagram for the Automated Trading House System is illustrated in the figure below.

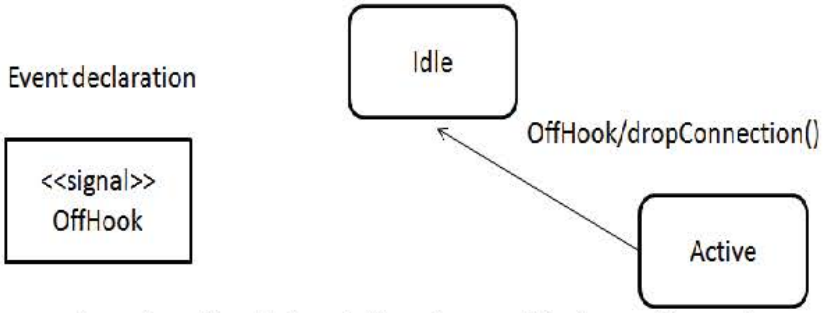


Q10. Briefly explain events and Signals?

Ans. In state machines (sequence of states), we use events to model the occurrence of a stimulus that can trigger an object to move from one state to another state. Events may include signals, calls, the passage of time or a change in state.

In UML, each thing that happens is modeled as an event. An event is the specification of a significant occurrence that has a location in time and space. A signal, passing of time and change in state are asynchronous events. Calls are generally synchronous events, representing invocation of an operation.

UML allows us to represent events graphically as shown below. Signals may be represented as stereotyped classes and other events are represented as messages associated with transitions which cause an object to move from one state to another.

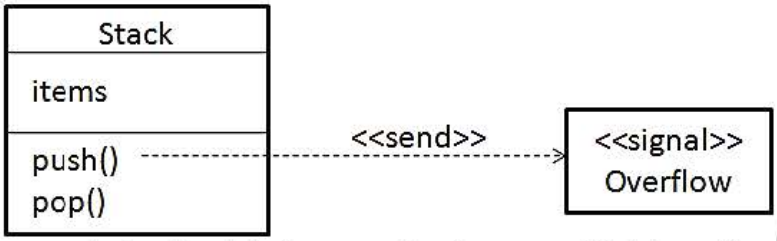


Types of Events: Events may be external or internal. Events passed between the system and its actors are external events. For example, in an ATM system, pushing a button or inserting a card are external events. Internal events are those that are passed among objects living inside the system. For example, a overflow exception generated by an object is an internal event.

In UML, we can model four kinds of events namely: signals, calls, passing of time and change in state.

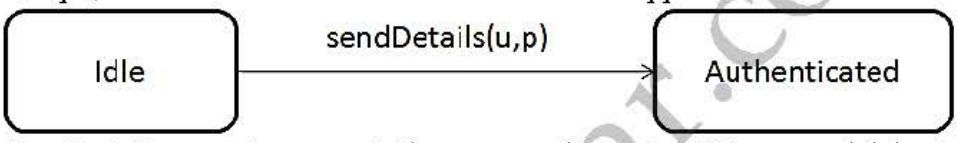
Signals: A signal is a named object that is sent asynchronously by one object and then received by another. Exceptions are the famous examples for signals. A signal may be sent as the action of a state in a state machine or as a message in an interaction. The execution of an operation can also send signals.

In UML, we model the relationship between an operation and the events using a dependency stereotyped with "send", which indicates that an operation sends a particular signal.



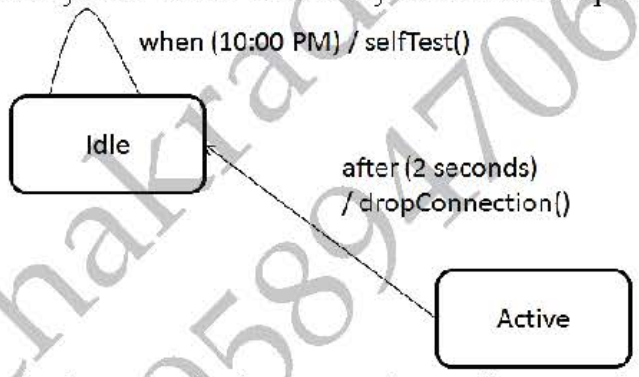
Call Events: A call event represents the dispatch of an operation from one object to another. A call event may trigger a state change in a state machine. A call event, in general, is synchronous.

This means that the sender object must wait until it gets an acknowledgment from the receiver object which receives the call event. For example, consider the states of a customer in an ATM application:



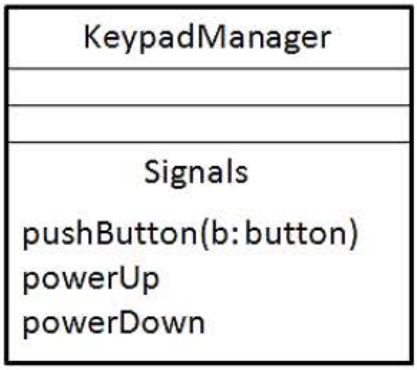
Time and Change Events: A time event represents the passage of time. In UML, we model the time event using the "after" keyword followed by an expression that evaluates a period of time.

A change event represents an event that represents a change in state or the satisfaction of some condition. In UML, change event is modeled using the keyword "when" followed by some Boolean expression.



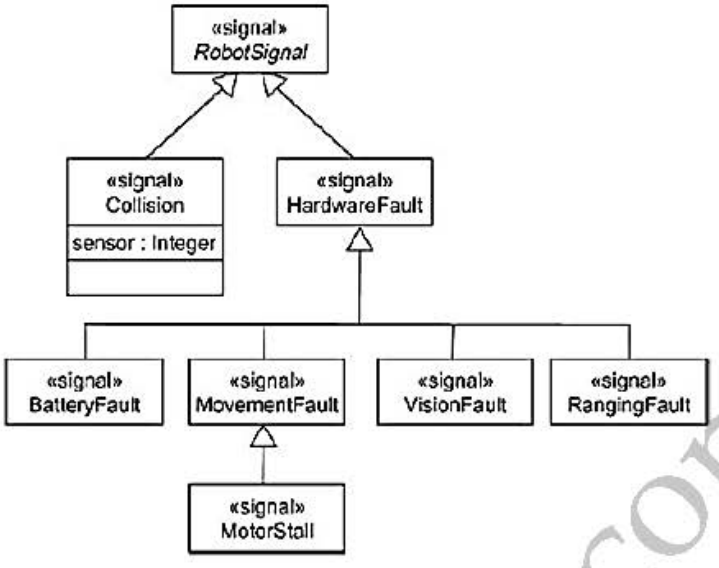
Sending and Receiving Events: Any instance of a class can receive a call event or signal. If this is a synchronous call event, the sender is in locked state with receiver. If this is a signal, then the sender is free to carry its operations without any concern on the receiver.

In UML, call events are modeled as operations on the class of an object and signals that an object can receive are stored in an extra component in the class as shown below:



- Common Modeling Techniques
- Modeling a family of signals
- To model a family of signals,
- Consider all the signals to which a set of objects can respond.
- Arrange these signals in a hierarchy using generalization-specialization relationship.

Look out for polymorphism in the state machine of the active objects. When polymorphism is found, adjust the hierarchy by introducing intermediate abstract signals.



- Modeling Exceptions
- To model exceptions,

For each class and interface and for each operation of such elements, consider the exceptional conditions that might arise.

Arrange these exceptions in a hierarchy.

For each operation, specify the exceptions that it may rise.