**2069 Bhadra**

1. **What is the meaning of Encapsulation from the viewpoint of structured systems analysis and design? Explain how encapsulation and Abstraction concept work together in object-orientation? [4+6]**

Encapsulation is the mechanism used to hide the data, internal structure, and implementation details of some element, such as an object or subsystem. Encapsulation is most often achieved through information biding, which is the process of hiding all the secrets of an object that do not contribute to its essential characteristics; typically, the structure of an object is hidden, as well as the ,implementation of its methods.

Encapsulation is the process of compartmentalizing (classifying) the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation. The purpose is to achieve potential for change: the internal mechanisms of the component can be improved without impact on other components, or the component can be replaced with a different one that supports the same public interface. Encapsulation also protects the integrity of the component, by preventing users from setting the internal data of the component into an invalid or inconsistent state. Another benefit of encapsulation is that it reduces system complexity and thus increases robustness, by limiting the interdependencies between software components.

For example:- a relational database is encapsulated in the sense that its only public interface is a Query language (SQL for example), which hides all the internal machinery and data structures of the database management system.

Encapsulation and Abstraction concept works together in object-orientation. Abstraction and encapsulation are complementary concepts: abstraction focuses upon the observable behavior of an object, whereas *encapsulation* focuses upon the implementation that gives rise to this behavior..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. Encapsulation is the mechanism used to hide the data, internal structure, and implementation details of some element, such as an object or subsystem. Abstraction is simplified description, or specification, of a system that emphasizes some of the system's details or properties while suppressing others. An abstraction focuses on the outside view of an object, and so serves to separate an object's essential behavior from its implementation

Deciding upon the right set of abstractions for a given domain is the central problem in object-oriented design. Abstraction is the thought process whereas encapsulation is implementation process.

1. **Here are some of the requirements for a system that you are going to build for the Grand Care Hospital, which is coming in operation soon. Identify which of the following information are used in building Conceptual, Specification and Implementation model. [6]**
2. **Each out-patient registration process takes an average of 90 seconds.**
3. **Liver unit will treat the Jaundice patients.**
4. **All bio-chemistry tests are carried out in Pathology department.**
5. **The newly installed GE USG machine can record diagnostic video too. Extended computer interfacing is required with Windows OS and mpeg-4 application.**
6. **Each Gastro-patient going to operation theater (OT) is to be re-evaluated for bowel status exactly 30 minutes before their OT schedule.**
7. **Some of the gastro-patients visit Lever unit too.**

**Solution:**

a) Specification Model

b) Implementation Model

c) Implementation Model

d) Conceptual Model

e) Specification Model

f) Conceptual Model

1. **Explain four different types of relationships that we model in object-oriented analysis, which exists between two different classes represented as shown in below figure: [8]**

**Solution:**

**Dependancy:-** It is relationship between two things in which change in one element also effects other. Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time. One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class. This is different from an association, where an attribute of the dependent class is an instance of the independent class.



**Association:-** It is a set of links that connects elements of an UML model. An *association* represents a family of links. Binary associations (with two ends) are normally represented as a line. An association can be named, and the ends of an association can be adorned with role names, ownership indicators, multiplicity, visibility, and other properties.  
There are four different types of association: bi-directional, uni-directional, Aggregation (includes Composition aggregation) and Reflexive.

http://upload.wikimedia.org/wikipedia/commons/4/4d/UML_role_example.gif

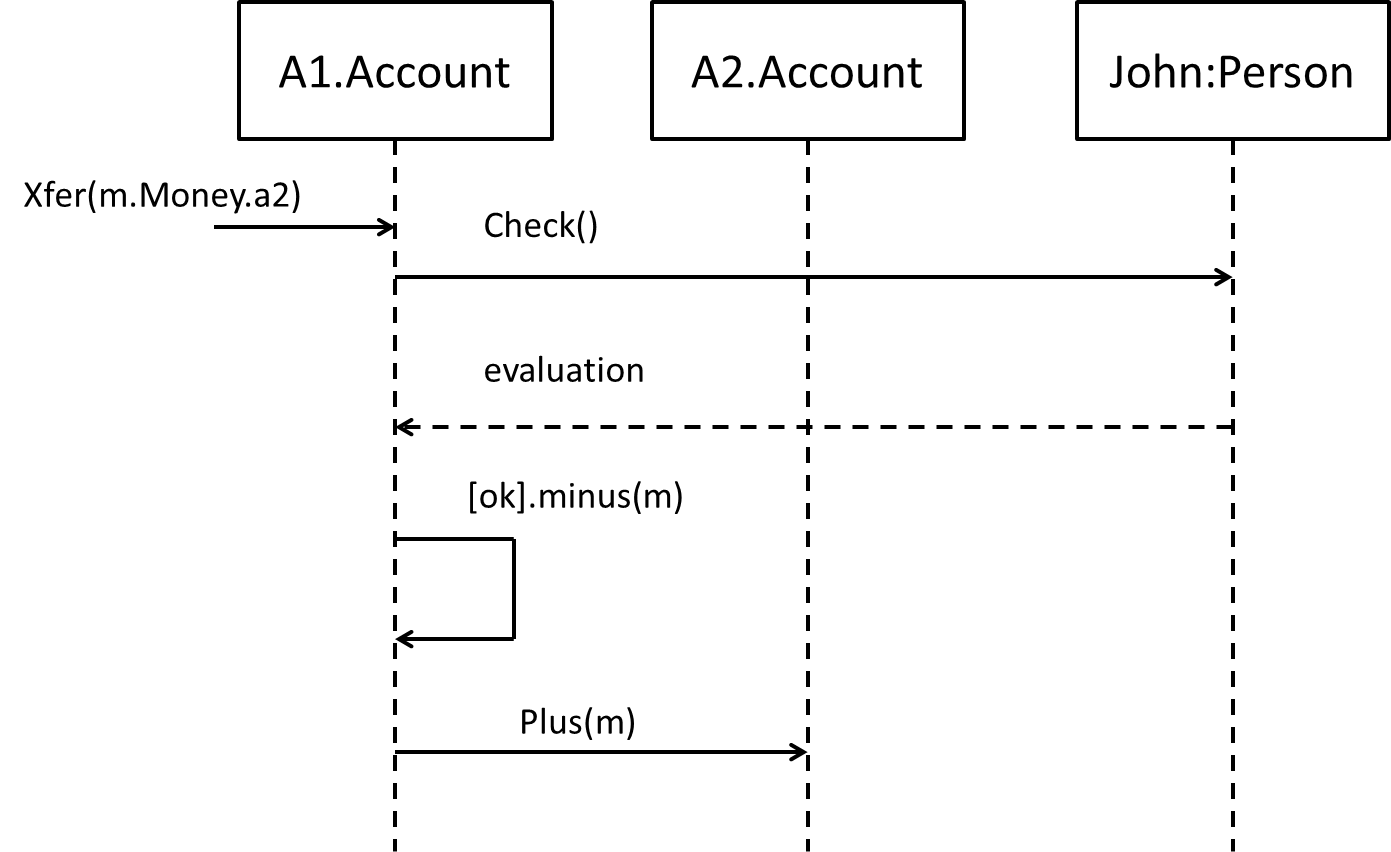
**Realization:-** In UML modeling, a realization relationship is a relationship between two model elements, in which one model element (the client) realizes (implements or executes) the behavior that the other model element . The UML graphical representation of a Realization is a hollow triangle shape on the interface end of the *dashed* line (or tree of lines) that connects it to one or more implementers. A plain arrow head is used on the interface end of the dashed line that connects it to its users. A realization is a relationship between classes, interfaces, components, and packages that connects a client element with a supplier element.



**Generalization:-** The Generalization relationship ("is a") indicates that one of the two related classes (the *subclass*) is considered to be a specialized form of the other (the *super type*) and superclass is considered as '***Generalization'*** of subclass. The UML graphical representation of a Generalization is a hollow triangle shape on the superclass end of the line (or tree of lines) that connects it to one or more subtypes. The [superclass](http://en.wikipedia.org/wiki/Superclass) (base class) in the generalization relationship is also known as the "parent", superclass, base class, or base type. The *subtype* in the specialization relationship is also known as the "child", subclass, derived class, derived type, inheriting class, or inheriting type.



1. **Based on the below diagram for model action of money transfer, answer the following questions: [3x3]**
2. **How many numbers of classes are involved in this transfer activity? What are they?**
3. **What are the events followed for transfer complete?**
4. **Why this check() function is required for this transfer?**

****

**Solution:**

1. Two classes are involved in this transfer activity. They are Account and Person.
2. The events followed for transfer function are check, evaluation, minus and plus.
3. Check() function is required for transfer function because in this sequence diagram we are transferring money from one account to another. In order to transfer the money it should first check whether there is sufficient amount of money(m) in A1.Account .After only evaluating money in A1.Account it will be able to transfer the money in A2.Account.
4. **A new digital clock recently available in the market, simultaneously displays the time and date. The time and date displays can be adjusted by the buttons available within the unit. The clock has two buttons, "mode" and "forward". If you wish to change the time you should first press the mode button, after which the time may be changed by the forward-button. If the mode button is pressed again, you can change the date(by the forward-button). If the mode button is pressed once again, you will go back to normal state. When the forward-button is pressed, the display will go a single unit(seconds or days) ahead. If the button is held down for more than two second, the display will change rapidly (once in every 0.2 second) ahead so long as the button is pressed.**

**Draw a state diagram for this clock control unit. [10]**

**Solution:**

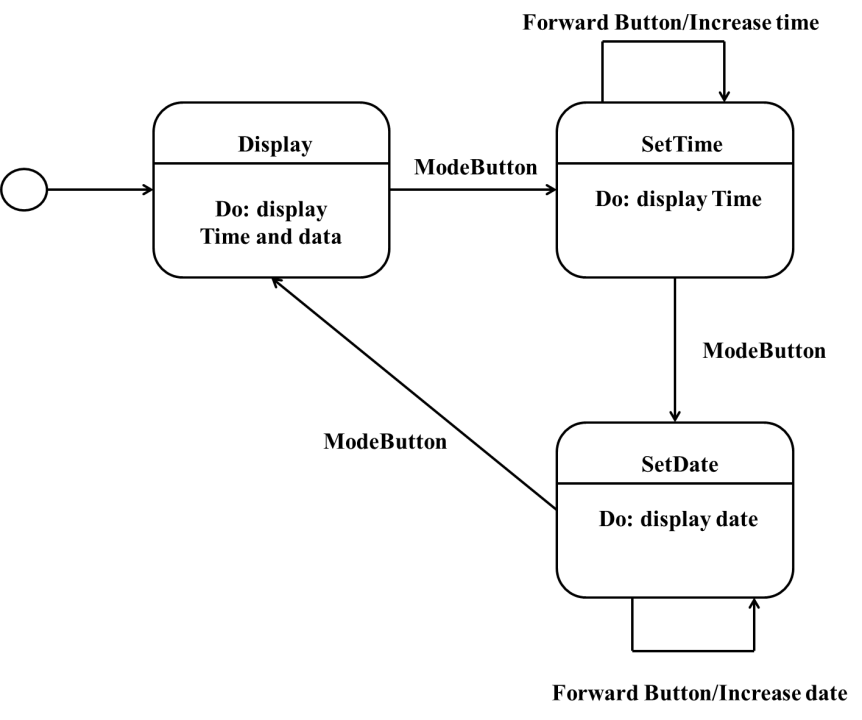
****

Fig: State diagram for clock

1. **Explain the forward and reverse engineering processes with outlining their merits and demerits in object-oriented implementation. [10]**

**Solution:**

Forward Engineering:- The traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system. Forward Engineering implies converting model into software code. The designs illustrated in UML diagrams will be incomplete, and only serve as a "springboard" to the programming. The interesting question is "How much diagramming before programming?" In part, the answer is a function of the experience and cognitive style of the designers. Some people are very spatial/visual thinkers, and expressing their software design thoughts in a visual language complements their nature; others aren't. A large percentage of the brain is dedicated to visual or iconic thinking and processing, rather than textual processing (code). Visual languages such as the UML play to a natural mental strength of most people. Those educated in the UML obviously have an easier time at it than those who are not. And in general, more experienced object designers can effectively design by drawing without straying into unrealistic speculation, because of their experience and judgment. Applied by adepts(skilled person), diagrams can help a group move more quickly toward a skillful design, due to the ability to ignore details and focus on the big picture. Now the step is towards implementing design to code. Using the case tools UML diagrams are used to generate the code. The process of implementing design to code is called forward engineering. Rational rose is one of the case tools for generating code from UML diagram. This process is done my mapping design to code.

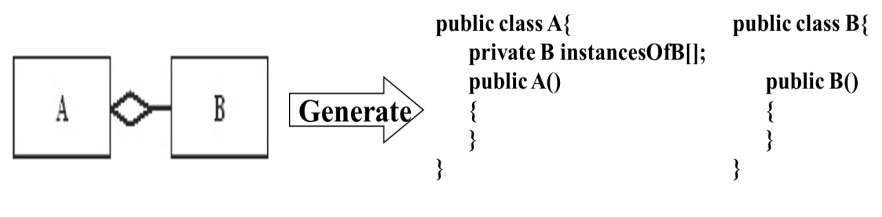
****

Fig:- Forward Engineering

Reverse Engineering:- Reverse engineering means generation of UML diagrams from code. It’s the reverse of forward engineering. In reverse engineering process the programmed code is used to generate the UML diagrams. Reverse Engineering of Object Oriented Code presents a unifying framework for the analysis of object oriented code.

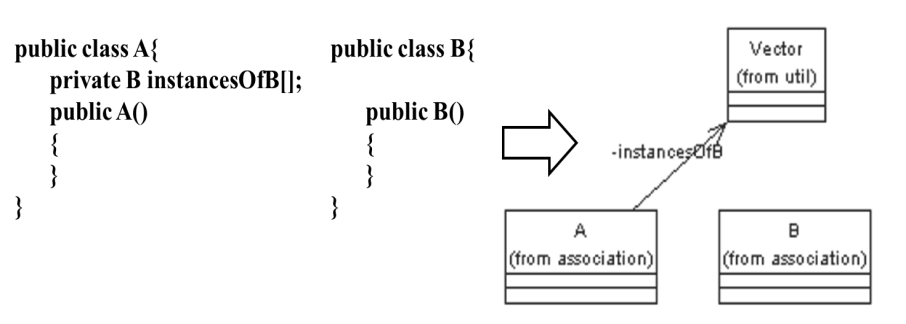
****

Fig:- Reverse Engineering

Reverse Engineering aims at supporting program comprehension, by exploiting the source code as the major source of information about organization and behavior of the program, and by extracting set of potentially useful views provided to programmers in the form of diagrams. Single diagram recovered from code through reverse engineering is insufficient. Rather, then set of complementary views should be obtained addressing different program understanding and needs. Reverse engineering is the difficult process. Software is among the most complex human artifacts. The way it is produced and maintained is thus an interesting object of study in itself. Among the various phases of software development, maintenance is by far the most demanding and expensive along the whole software's life cycle. Difficulties are mainly related to the understanding of extant systems and the estimate of the impact of changes. Reverse engineered diagrams provide useful information about the system being maintained. In this respect, they can support the program understanding activities, drive refactoring and restructuring interventions. Finally, they can be employed to trace back the compliance of the actual code with the intended design.

Merits of Forward Engineering

1. Once the design is created it is easy to generate code.

2. It also saves time during coding.

3. Since UML provides overall system view it will easy for mapping design to code.

4. Case Tools is used to generate code from UML diagrams so helps in saving time.

Demerits of Forward Engineering

1. Since code is generated using case tools it always does not guarantee correct code.

2. If UML diagrams are incorrect then code generated will be wrong.

3. Developing UML diagrams is time consuming process.

Merits of Reverse Engineering

1. In reverse engineering we just have to observe the code for generating UML diagrams, it will help to save the time because we already have the code.

2. Case Tools is used to generate design from code which makes easy for generating UML diagrams.

3. It also helps in reducing the costs.

Demerits of Reverse Engineering

1. Generating design from code is difficult process.

2. Although we use case tools it always does not guarantee the correct design.

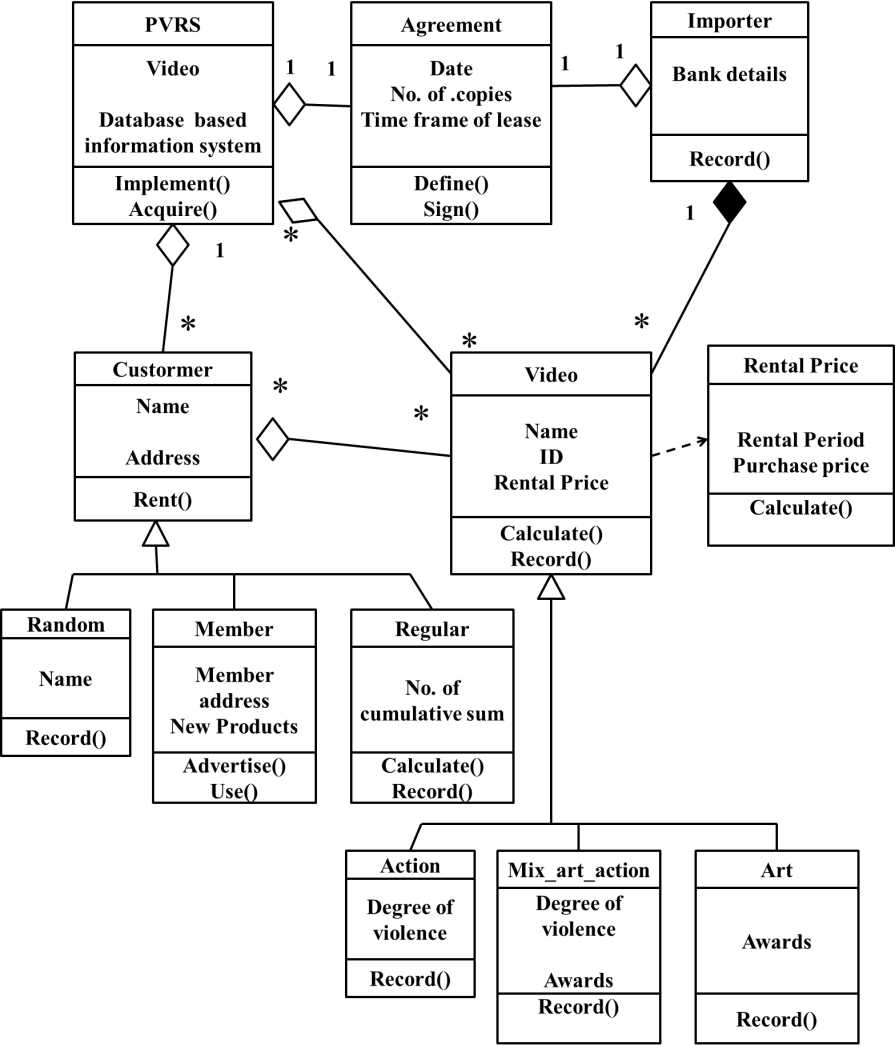
3. Lots of time will be spend in analyzing code.

1. **The Premier Video Rental shop (PVRS) decides to implement a database-based information system. PVRS acquires the video from the importer or chain trade. An agreement will be signed with both partners and it defines the date, number of copies, time frame of the lease and purchase price. As an additional info of importer also the address and bank details will be recorded. The customer rents a video from the PVRS. From each video the name, ID and rental price information will be recorded. The rental price is calculated from the rental period, the purchase price and customer relationship. Video types include action, art and children's video. Video may also be blend of action and art videos. As an action video info the degree of violence will be recorded, and from art videos the awards and from children's video the age limit. The customer relationship can be random customer, regular or member of PVRS club. As an overarching customer info the name will be recorded and from regular customer the cumulative sum of the number of rental events.**

**From the members of PVRS club information, the member address is used in order to advertise new products and offers.**

**Draw a class diagram, which presents the main classes, properties, methods and relationships between classes. [12]**

**Solution:**

****

1. **Write short notes: [3x5]**
2. **Sequence diagram**
3. **Swim lanes**
4. **Polymorphic signal**
5. **Sequence Diagram:-**

Sequence diagram models the flow of control by time-ordering; depicts the interaction between various objects by of messages passed, with a temporal dimension to it. A sequence diagram shows a particular sequence of messages exchanged between a number of objects. Sequence diagrams also show behavior by showing the ordering of message exchange. A sequence diagram shows some particular communication sequences in some run of the system .It is not characterizing all possible runs. Sequence Diagrams System display object interactions arranged in time sequence. A sequence diagram shows a set of objects and the messages sent and received by those objects. The objects are typically named or anonymous instances of classes, but may also represent instances of other things, such as collaborations, components, and nodes. You use sequence diagrams to illustrate the dynamic view of a system.

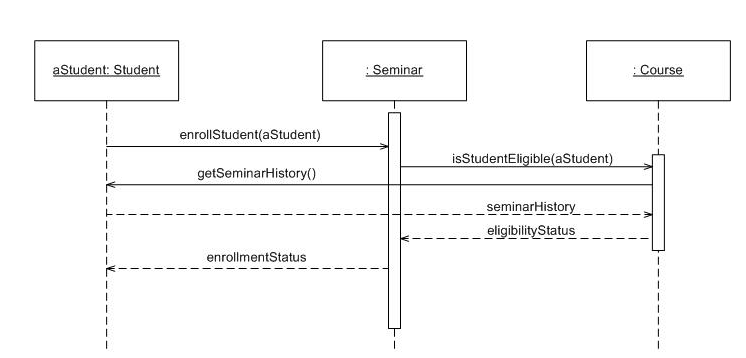


Fig:-Sequence Diagram for Enrolling Seminar

**b) Swim Lanes:-**

Swim Lanes represents the column in activity diagram to group the related activities. These are represented in form of partitioned region. Swim lanes are useful when modeling business workflow because they can represent organizational units or role within a business model. Swim lanes are very similar to object because they provide a way to tell who is performing a certain role. When used to diagram a business process that involves more than one department, swim lanes often serve to clarify not only the steps and who is responsible for each one, but also how delays, mistakes or cheating are most likely to occur.

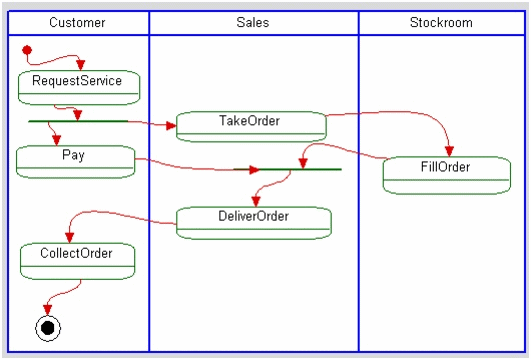


Fig:-Activity diagram with swim-lanes

Swim-lanes have some features. They are:-

1. Each action is assigned to one swim-lane.
2. Activity flows can cross lanes.
3. Swim-lanes do not change ownership hierarchy.
4. Swim-lane association (representing field population) to a class (only) can be created by dragging the class from the browser to the Swim-lane name compartment.
5. The relative ordering of swim-lanes has no semantic significance.
6. There is no significance to the routing of an activity flow path.
7. Parts representing internal behavior can be specified on swim-lanes.

**c) Polymorphic signal:-**

A polymorphic signal is a signal delivered at run time to a specific state machine for a class in a generalization hierarchy. The polymorphic signal must be able to be received in every branch in the superclass's hierarchy, so that the polymorphic signal occurrence always has a receiving state machine instance. When a supervisor tells a clerk to go off-duty, the supervisor does not care whether the clerk is a shipping clerk or a stock clerk. Similarly, when an action sends a signal to an object in a class hierarchy, the sender should not need to know the subclass of the object. Rather, the signal is polymorphic. To keep distinction between signal and event we have polymorphic event. A polymorphic event is an event that has many potential receiving state machines in a generalization hierarchy.

There are certain rules for polymorphic signal. They are:-

1. A polymorphic event specification is declared for an abstract superclass, and the corresponding event may be received by any of the superclass's subclasses.
2. A polymorphic signal cannot be received by the superclass in which the event is specified, only in the subclasses.
3. The polymorphic signal must be able to be received in every branch in the superclass's hierarchy, so that the polymorphic signal occurrence always has a receiving state machine instance.
4. At run time, the polymorphic signal occurrence is received as an event by exactly one state machine instance in a given hierarchy.
5. If a superclass has repeated specializations, each class in the specialization hierarchy must be able to receive an occurrence of the event.
6. If a superclass has multiple specializations, the polymorphic signal occurrence is received by at most one state machine instance in each specialization hierarchy.

**2069 Poush**

1. **Explain object oriented system with reference to class, object, encapsulation, abstraction message, inheritance, interface and polymorphism with suitable examples. [8]**

**Soluton:**

Object Oriented System are the system where both data and procedures combine in software objects, message passing is used to communicate digitally with and between objects, similar objects are grouped into class structure, and both data and procedures are inherited through the class structure to specific instances of objects. An object-oriented system is composed of objects. The behavior of the system results from the collaboration of those objects. Collaboration between objects involves them sending messages to each other. Sending a message differs from calling a function in that when a target object receives a message, it decides on its own what function to carry out to service that message. The same message may be implemented by many different functions, the one selected depending on the state of the target object. Object-oriented analysis and design (OOAD) is a software engineering approach that models a system as a group of interacting objects. Each object represents some entity of interest in the system being modeled, and is characterized by its class, its state (data elements), and its behavior. Various models can be created to show the static structure, dynamic behavior, and run-time deployment of these collaborating objects. There are a number of different notations for representing these models, such as the Unified Modeling Language (UML).

In object oriented system a class is used to describe characteristics of any entity of the real world. An object is a pattern of the class. An actual object created at runtime is called as an instance. A class, apart from characteristics has some functions to perform called as methods.

Object oriented system has different features such as class, object, encapsulation, abstraction message, inheritance, interface and polymorphism. Let us take an example:-

A class named “Food” has attributes like ‘price’, ‘quantity’. “Food” class has methods like Serve\_food(), bill\_food().

Object is runtime entity in Object Oriented Systems interact through messages. It is a particular case of a class.

Inheritance: The main class or the root class is called as a Base Class. Any class which is expected to have ALL properties of the base class along with its own is called as a Derived class. The process of deriving such a class is Derived class. For the “Food” class, a Derived class can be “Class Nepalese Food”.

Abstraction:- Abstraction is creating models or classes of some broad concept. Abstraction can be achieved through Inheritance or even Composition.

Interface:- An interface is a description of the actions that an object can do... for example when you flip a light switch, the light goes on, you don't care how, just that it does.

Encapsulation:- Encapsulation is a collection of functions of a class and object. The “Food” class is an encapsulated form. It is achieved by specifying which class can use which members (private, public, protected) of an object.

Polymorphism:- Polymorphism means existing in different forms. Inheritance is an example of Polymorphism. A base class exists in different forms as derived classes. Operator overloading is an example of Polymorphism in which an operator can be applied in different situations.

1. **IOE is willing to develop a system for the student result management of its BE program. Now, prepare the problem statement from the side of examination control division. What are building blocks of UML? Explain with suitable examples and notations. [4+6]**

**Solution:**

The building blocks of UML are:-

* Things
* Relationships
* Diagrams

1. **Things:-Things** are the most important building blocks of UML. Things can be:
   1. **Structural**
   2. **Behavioral**
   3. **Grouping**
   4. **Annotation**

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

## Class: Class represents set of objects having similar responsibilities.

class

## Interface: Interface defines a set of operations which specify the responsibility of a class.

Interface

## Collaboration: Collaboration defines interaction between elements.

Collaboration

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

Use case

## Component: Component describes physical part of a system.

Component

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



# b)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.

Interaction

**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.



# Annotational things: Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. Note is the only one Annotational thing available.

1. **Relationship:-** Relationship is another most important building block of UML. It shows how 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 one.

Dependency

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

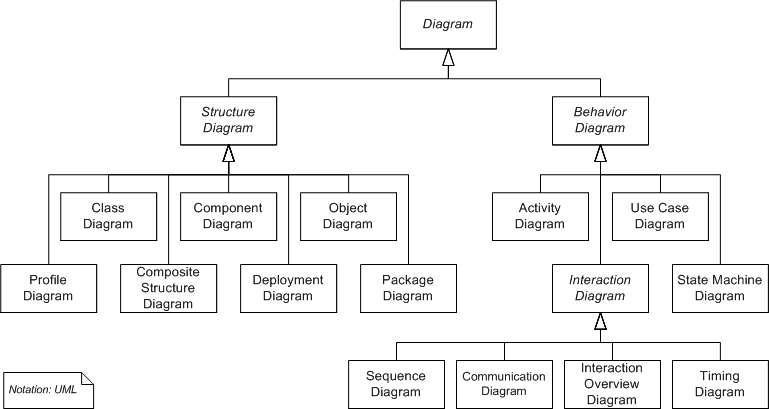
Association

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

Generalization   
**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.

Realization

1. **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.



1. **A web-based online store has "Buy a Product" scenario as follows:**

**The customer browses the catalog and adds desired items to the shopping basket. When the customer wishes to pay, the customer describes the shipping and credit card information and confirms the sale. The system checks the authorization on the credit card and confirms the sale both immediately and with a follow-up email. Now construct conceptual model for the scenario. [6]**

**Solution:**

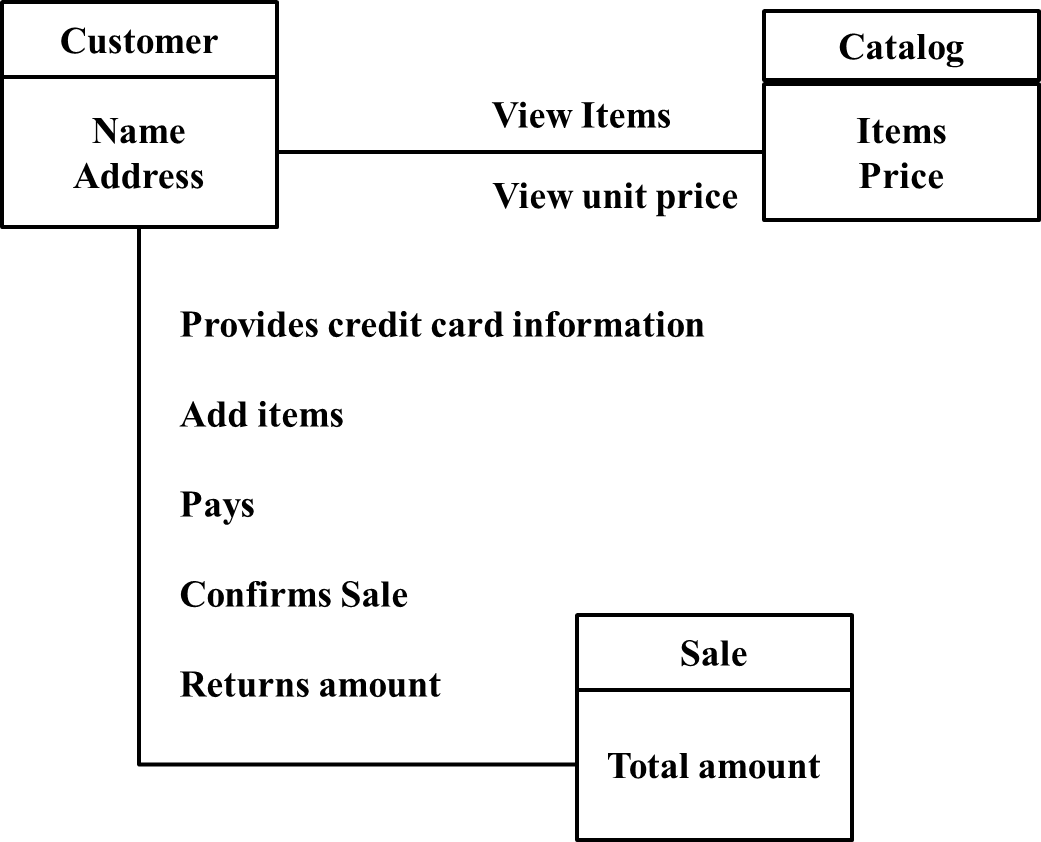
****

Fig:-Conceptual Scenario

1. **Draw a class diagram for point of sale system with association and multiplicity.[6]**

**Solution:**

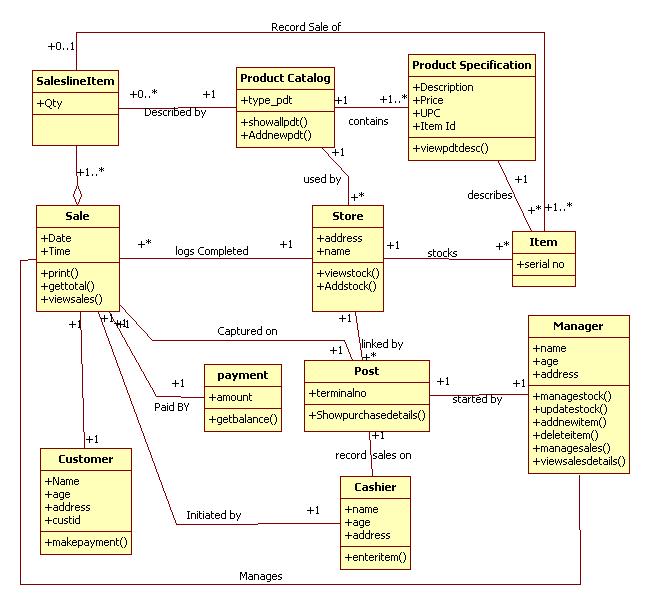


Fig: Class diagram of Point of sale

1. **Read the following case study carefully and answer the given questions.**

**Ministry of Health and Population is willing to computerize the system. This new system will be able to tell the population of the country, zone and district and even of the word of specific place. The system will update its data in monthly basis so that the birth rate and death rate can be easily seen. The home page is displayed when a person enters to the system. Administrators can enter to the admin panel by logging in with a an ID and password. He/she has privileges to enter and modify the data into the database. On the other hand, normal users can view the data but not modify them. They can also visualize the data in graphical form with animated charts, maps as well as in tabular form based on their selection of data. Besides, they can also view the forecasted data. (Make your assumptions if necessary)**

**Draw collaboration diagram and use case diagram. [6+6]**

**Solution:**

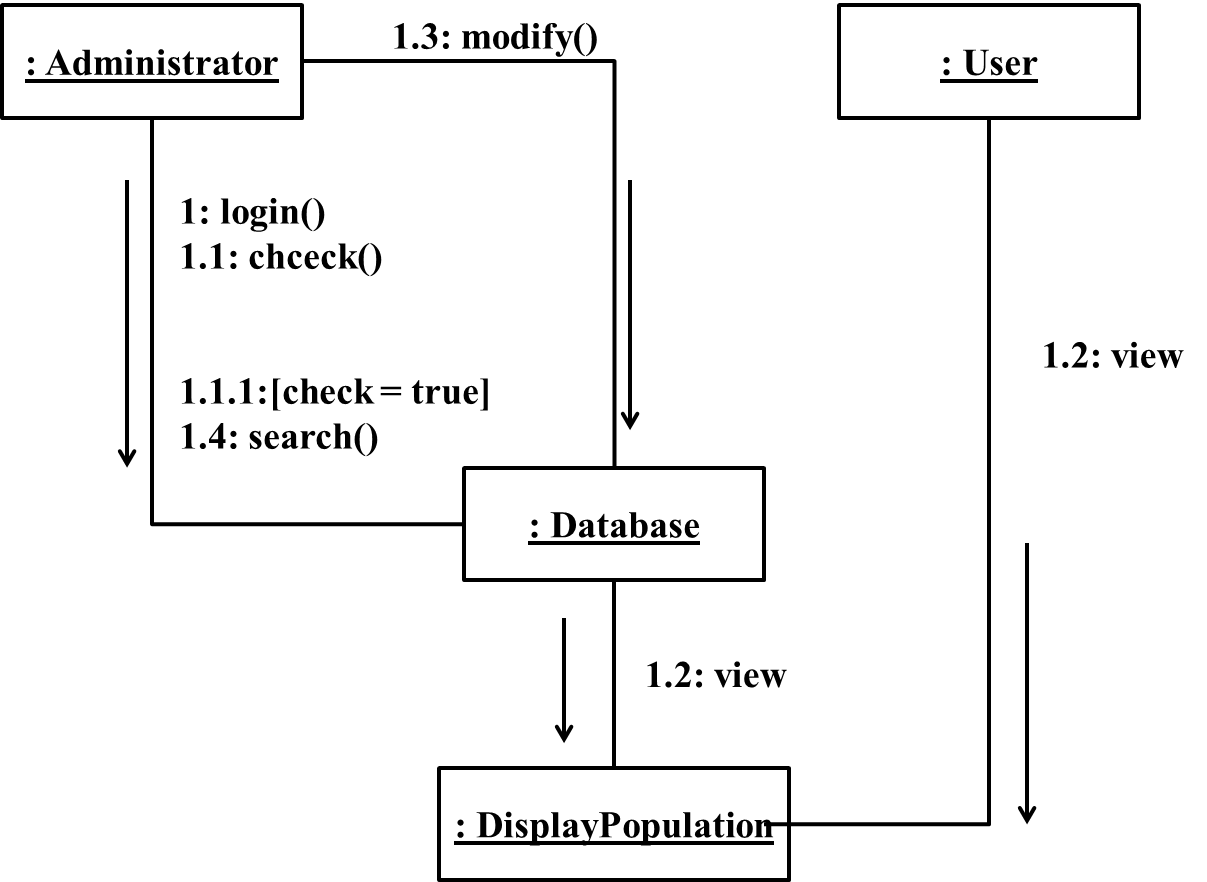
****

Fig: Collaboration diagram

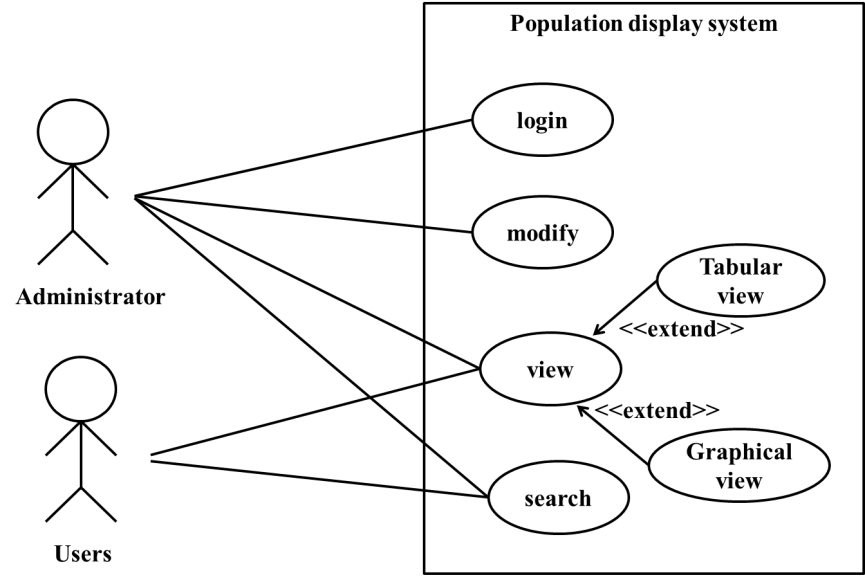
****

Fig:-Use Case Diagram

1. **What is framework? How design pattern is useful? Explain any one pattern in detail with suitable example. [6]**

Object-oriented frameworks are established tools for domain-specific reuse. Many framework design patterns have been documented, e.g. reverse engineering framework architectures from conventionally built applications for a given domain. The framework development cycle generally evolves from open framework to closed application. We describe a more flexible component-based approach to framework design that stresses a common interface for ’plugging-in’ new components at different lifecycle stages.

In OOD, a pattern is a named description of a problem & solution that can be applied to new contexts. Ideally a pattern advises us on how to apply its solution in varying circumstances. Design patterns are useful because

* Patterns, given a specific category of problem, guide the assignment of responsibilities to objects.
* Pattern has name which **supports chunking (grouping)** and incorporating that concept into our understanding and memory.
* It **facilitates communication**. Naming a complex idea such as a pattern is an example of the power of abstraction.
* **It reduces a complex form** to a simple one by eliminating detail.

**Creator pattern:-** Creation of objects is one of the most common activities in an object-oriented system. Which class is responsible for creating objects is a fundamental property of the relationship between objects of particular classes. Simply, "Creator pattern is responsible for creating an object of class".

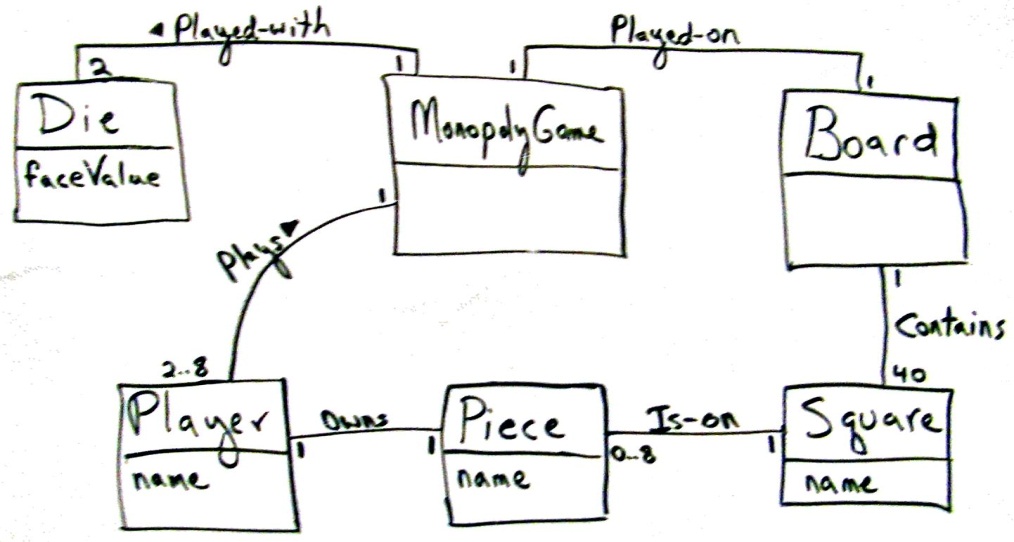
Name: Creator

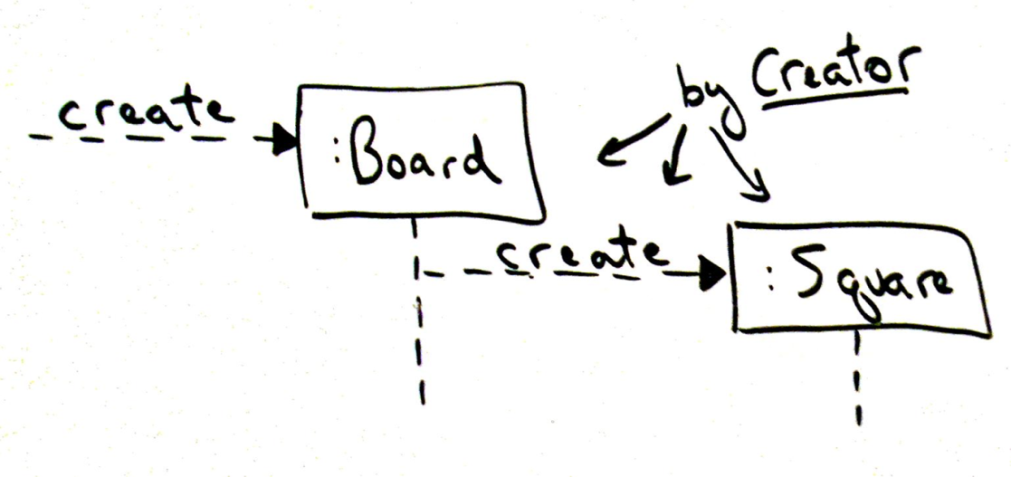
Problem: Who creates an A? (doing responsibility)

Solution: Assign class B the responsibility   
to create an instance of class A if one of these is true (the more the better):

* B contains or aggregates A (in a collection)
* B records A
* B closely uses A
* B has the initializing data for A

Here, in the square creation problem since no classes has been defined yet we should refer to the domain model of it as shown in the figure below. We see that the board contains squares which are a conceptual perspective, not a software one. But we can derive from it. In this example the board creates squares also the squares will always be a part of board & Board manages their creation & destruction. Thus, they are in a composite aggregation association with a board.





1. **Explain development process with suitable example. How can you map design into code? Illustrate with diagrams produced in question number five by using any object oriented languages like C++, Java, C# etc. [4+7]**

**Solution:**

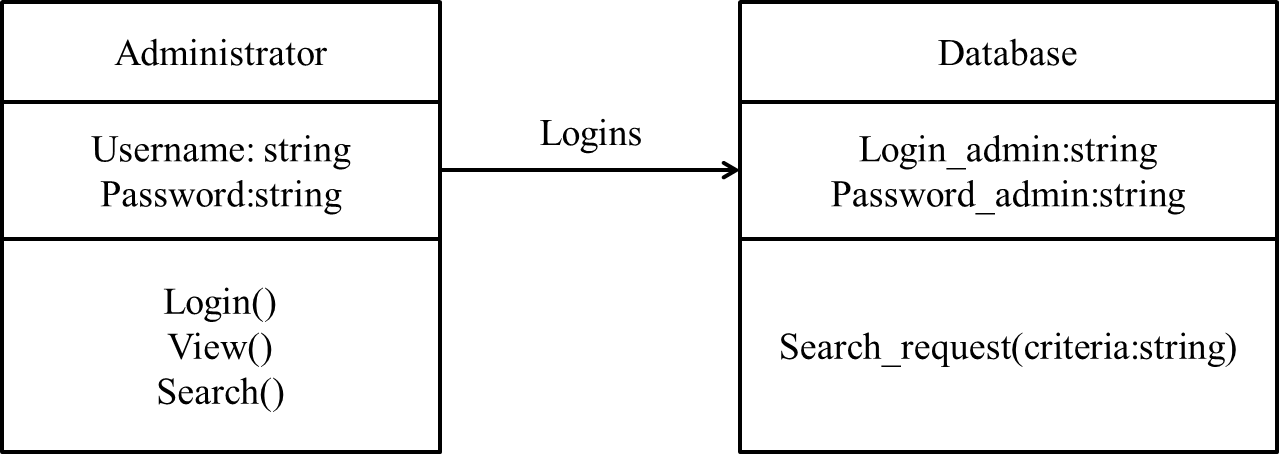
System development can be viewed as a process. The development itself, in essence, is a process of change, refinement, transformation, or addition to existing product. In object oriented analysis and design **unified process** is the development process which is iterative and incremental. Unified Development process includes 4 phases. They are:-

* **Inception Phase:-** Inception is the smallest phase in the project, and ideally it should be quite short. The following are typical goals for the Inception phase.
  + - Establish a justification or business case for the project
    - Establish the project scope and boundary conditions
    - Outline the use cases and key requirements that will drive the design tradeoffs
    - Outline one or more candidate architectures
    - Identify risks
* **Elaboration Phase:-** The primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture. Common processes undertaken in this phase include the creation of use case diagrams, conceptual diagrams (class diagrams with only basic notation) and package diagrams (architectural diagrams).
* **Construction Phase:-** Construction is the largest phase in the project. In this phase the remainder of the system is built on the foundation laid in Elaboration.  Common UML (Unified Modeling Language) diagrams used during this phase include Activity, Sequence, Collaboration, State (Transition) and Interaction Overview diagrams.
* **Transition Phase:-** In this phase the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations.

**Mapping design into code involves following step:-**

1. With the completion of interaction diagram & DCD(Design class diagram) , one can start generation of code where in the interaction diagram & DCD can be used as the input.
2. At the very least, DCDs depict the class or interface name, superclass’s, operation signatures, and attributes of a class.
3. The sequence of messages in the interaction diagram translates to a series of statements in the method definitions.
4. With the help of DCD, basic class definition can be generated.
5. With the help of Interaction diagram, method definitions are created.

With reference to Q.no 5 we first have to draw DCD.



public class Administrator{

private string username;

private string password;

public void login(username, password)

{….. }

Public void view(){.. }

Public string search(){..}

}

Using interaction diagram which is already done in Q.no 5. Method definition can be generated.

{ Display population=administrator.search(string country);

}

1. **Construct a system sequence diagram for customer from a "Food ordering system" of a very busy restaurant where seating and ordering is regulated by seating manager. [6]**

**Solution:**

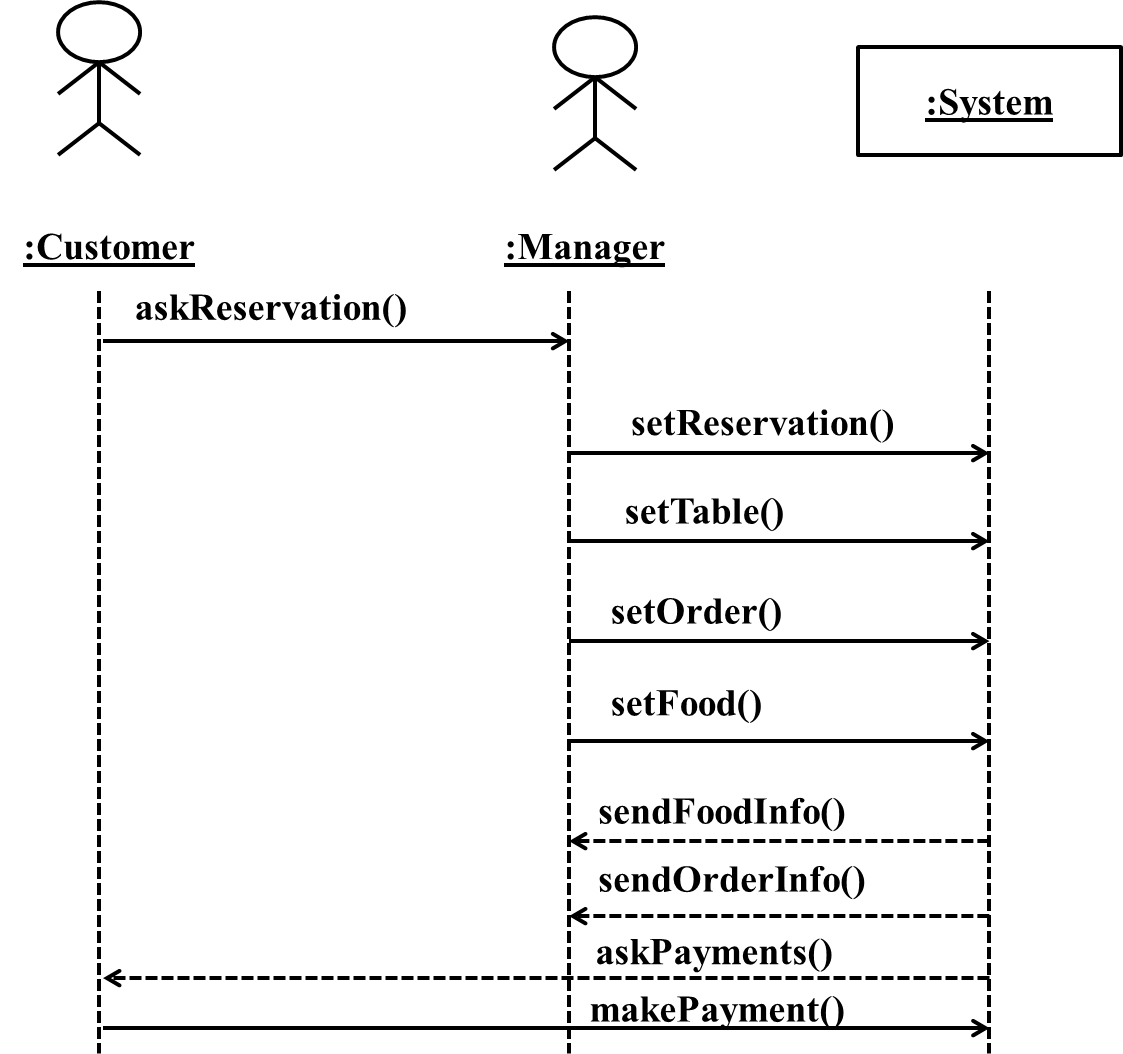
****

Fig: System Sequence Diagram of Food ordering system

1. **Illustrate how can you create classes from design class diagrams and methods from interaction diagrams. (Use C#, Java etc.) [6]**

**Solution:**

With the completion of interaction diagram & DCD one can start generation of code where in the interaction diagram & DCD can be used as the input. Implementation of OO language requires writing source code for:

* Class & interface definitions
* Method definitions

At the very least, DCDs depict the class or interface name, super classes, operation signatures, and attributes of a class. This is sufficient to create a basic class definition in an OO language. If the DCD was drawn in a UML tool, it can generate the basic class definition from the diagrams.

**Defining a Class with Methods and Simple Attributes**

From the DCD,

* a mapping to the attribute definitions(java fields) and
* method signatures for the Java definition of SalesLineItem is straightforward as shown in fig below.
* The create method is often excluded from the class diagram because of its commonality and multiple interpretations, depending on the target language.

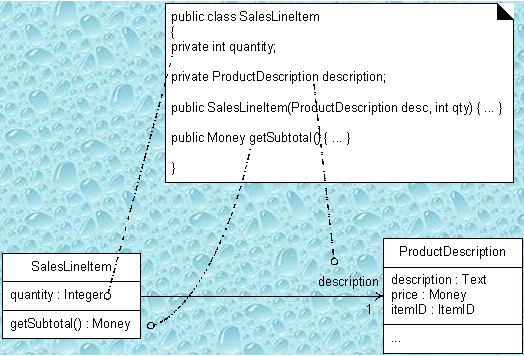
****

Fig:-From DCD to Java class

**Creating Methods from Interaction Diagrams**

* The sequence of messages in the interaction diagram translates to a series of statements in the method definitions
* The enterltem interaction diagram in fig below illustrates the Java definition of the enterltem method.

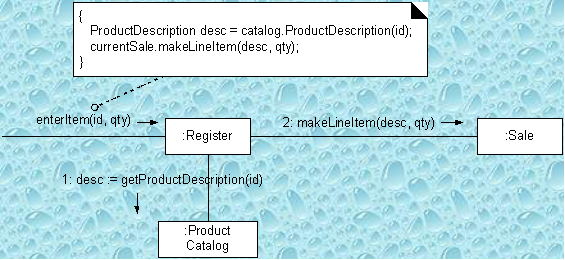


Fig:- Interaction Diagram to Method

For this example we will explore the implementation of the Register & its enterItem method.

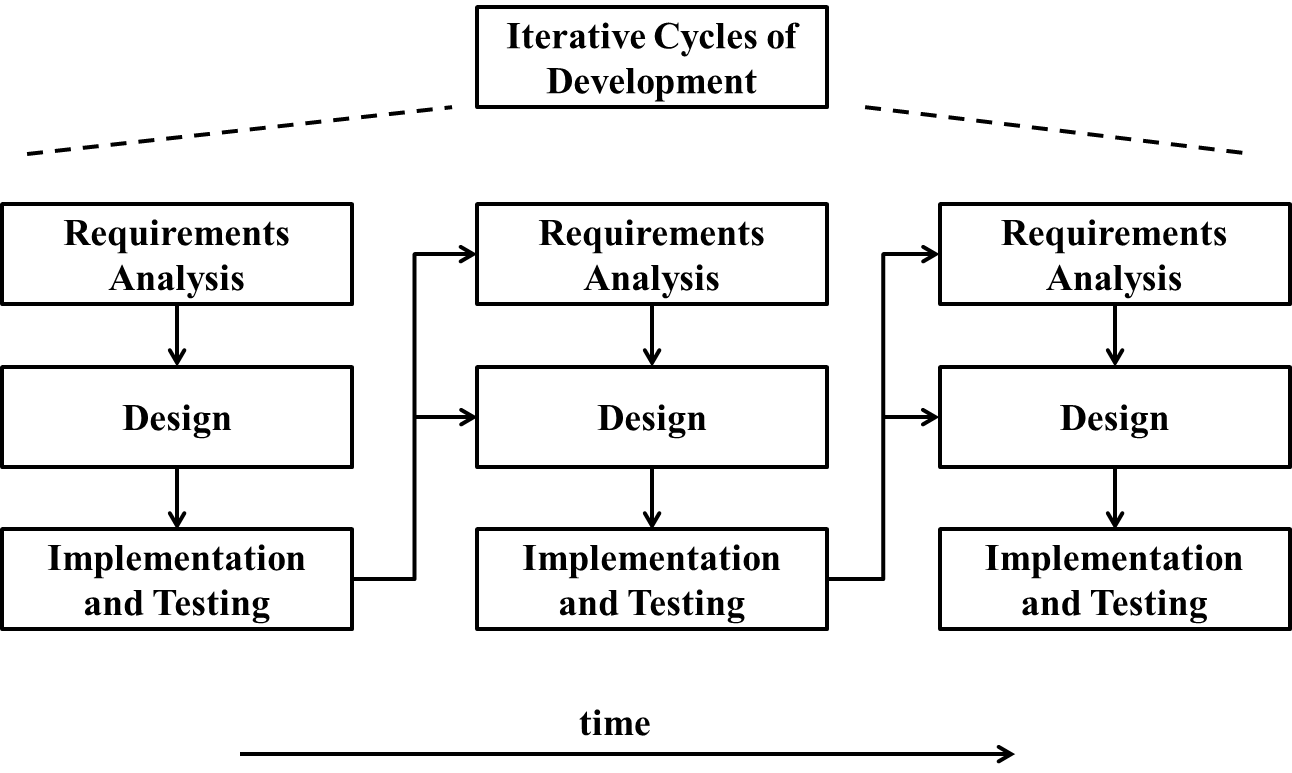
* A Java definition of Register class is shown below
* The enterltem message is sent to a Register instance; therefore, the enterltem method is defined in class Register.

public void enterltem( ItemID itemID, int qty);

1. **Write short notes on: [3x3]**
2. **Iterative cycles of development**

The word "iterative" means that it involves repetition. Iterative Development is a development approach that "cycles" through the development phases, from gathering requirements to delivering functionality in a working release. Iterative and Incremental development is any combination of both iterative design or iterative method and incremental build model for development. The combination is of long standing and has been widely suggested for large development efforts. During software development, more than one iteration of the software development cycle may be in progress at the same time." and "This process may be described as an "evolutionary acquisition" or "incremental build" approach." The relationship between iterations and increments is determined by the overall software development methodology and software development process. Incremental development slices the system functionality into increments (portions). In each increment, a slice of functionality is delivered through cross-discipline work, from the requirements to the deployment. The unified process groups increments/iterations into phases: inception, elaboration, construction, and transition.

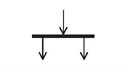
* **Inception** identifies project scope, requirements (functional and non-functional) and risks at a high level but in enough detail that work can be estimated.
* **Elaboration** delivers a working architecture that mitigates the top risks and fulfills the non-functional requirements.
* **Construction** incrementally fills-in the architecture with production-ready code produced from analysis, design, implementation, and testing of the functional requirements.
* **Transition** delivers the system into the production operating environment.

****

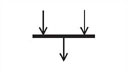
1. **Synchronization bar**

A synchronization bar helps illustrate parallel transitions used in activity diagram. Synchronization is also called forking and joining.

Fork:-For the branching of flows in two or more parallel flows we use a synchronization bar, which is depicted as a thick horizontal or vertical line:



Join:- For the consolidation of two or more parallel flows we also use a synchronization bar, which is depicted as a thick horizontal or vertical line:



During consolidation synchronization takes place, meaning the flow proceeds only after all incoming flows have reached the consolidation point. Join has two or more inputs and one output.

**c) Flow of object**

Activity diagrams are a well-known means to model the control flow of system behavior. Their expressiveness can be enhanced by using their object flow notation. In addition, we refine activities by pairs of pre- and post-conditions formulated by interrelated object diagrams. To define a clear semantics for refined activity diagrams with object flow, we use a graph transformation approach. Control flow is formalized by sets of transformation rule sequences, while object flow is described by partial dependencies between transformation rules. **An object flow is a path along which objects or data can pass.** An object is shown as a rectangle. An object flow is shown as a connector with an arrowhead denoting the direction the object is being passed. An object flow must have an object on at least one of its ends. A shorthand notation for the above diagram would be to use input and output pins.

