# CS8592 – OBJECT ORIENTED ANALYSIS AND DESIGN

**UNIT I**

**UNIFIED PROCESS AND USE CASE DIAGRAMS**

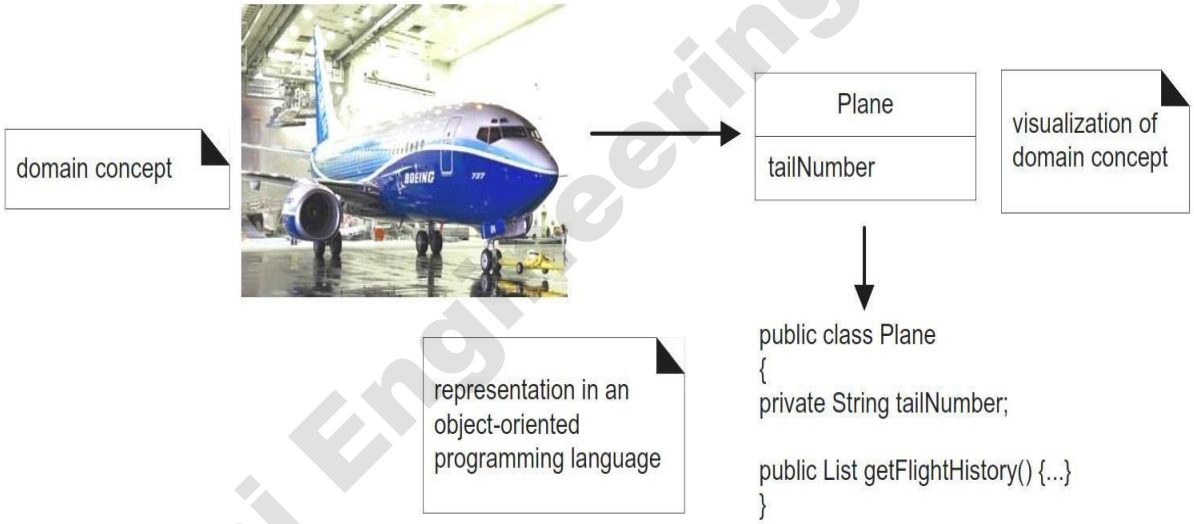
**PART- A**

1. **What** is Analysis and Design **(May/June 2013)**

- Analysis (do the right thing) emphasizes an investigation of the problem and requirements, rather than a solution. For example, if a new online trading system is desired, how will it be used? What are its functions?
- Design (do the thing right) emphasizes a conceptual solution (in software and hardware) that fulfills the requirements, rather than its implementation. For example, a description of a database schema and software objects. Design ideas often exclude low-level or "obvious" detail.

2. **What** is object oriented analysis and design? (**April/May 2011, Nov/Dec 2013, May/June 2014,Apr/May 2015,Apr/May 2017)**

- Object-oriented analysis is emphasizing on finding and describing the objects or concepts in the problem domain. For example, in the case of the flight information system, some of the concepts include Plane, Flight, and Pilot.
- Object-oriented design (or simply, object design) is emphasizing on defining software objects and how they collaborate to fulfill the requirements. For example, a Plane software object may have a tail Number attribute and a get Flight History method.




3. **What** is the Need for Modeling **(May/June 2014)?**

- **Efficient and effective communication:** Visual model diagrams can be more understandable and can allow users and stakeholders to give developers feedback on the appropriate requirements and structure of the system.
- **Useful and stable abstraction:** Modeling helps coding. A goal of most modern software methodologies is to first address "what" questions and then address "how" questions, i.e. first determine the functionality the system is to provide without consideration of implementation constraints, and then consider how to make specific solutions to these abstract requirements, and refine them into detailed designs and codes by constraints such as technology and budget.

4. **List** out any four reasons for the complexity of software? **(Nov/Dec 2011)**
   a. High level of abstraction.
   b. Seamless transition among different phases of software development.

**5. Why** do we need object oriented system development? **(Nov/Dec 2012)**

The system created using object oriented methods are easier to adapt changing requirements, easier to maintain, more robust, promote greater design. The reasons why object orientation works

1. High level of abstraction.
2. Seamless transition among different phases of software development.
3. Encourage of good programming techniques.
4. Promotion of reusability.

**6. What** is UML? **(May/June 2012, May/June 2013)**

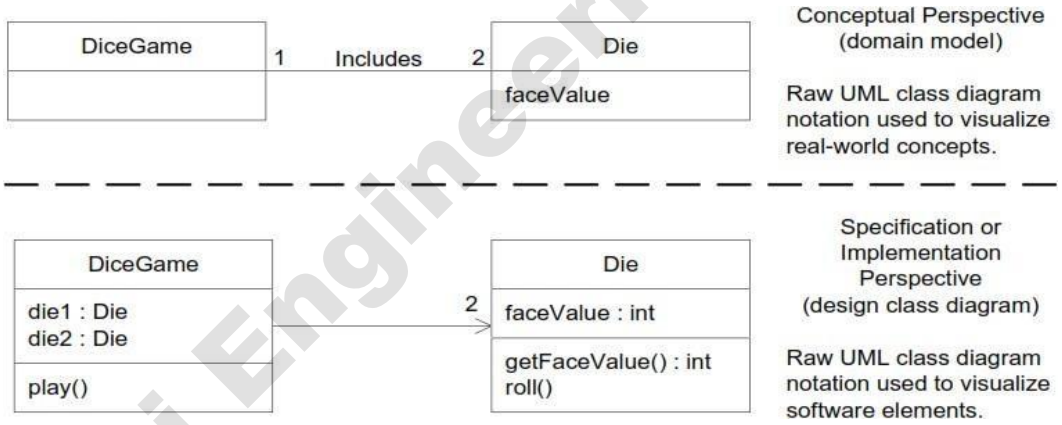
The Unified Modeling Language is a visual language for specifying, constructing and documenting the artifacts of systems. The word visual in the definition is a key point -the UML is the de facto standard diagramming notation for drawing or presenting pictures

**7. What** are the ways to Apply UML?

- UML as sketch
- UML as blueprint
- UML as programming language

**8. What** are the Perspectives to Apply UML? **(Nov/Dec 2015, Nov/Dec 2016, Apr/May 17)**

- Conceptual perspective
- Specification (software) perspective
- Implementation (software) perspective




**9. Define** Software class

**The Meaning of "Class" in Different Perspectives**

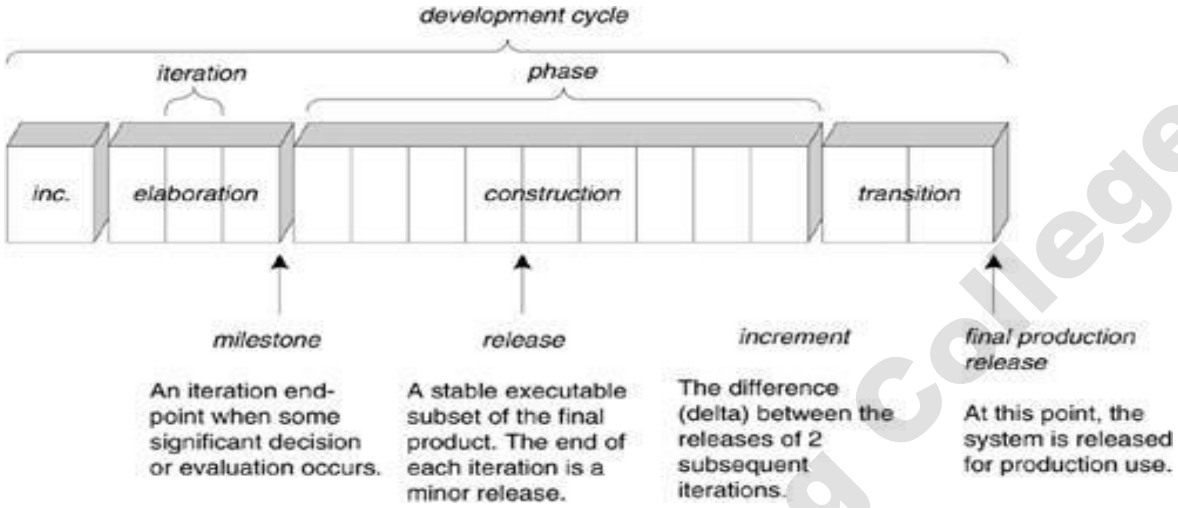
- **Conceptual class** real-world concept or thing. A conceptual or essential perspective. The UP Domain Model contains conceptual classes.
- **Software class** a class representing a specification or implementation perspective of a software component, regardless of the process or method.
- **Implementation class** a class implemented in a specific OO language such as Java.

**10. List** the UML Diagrams.

a. Use Case Diagram
b. Class Diagram
c. Collaboration Diagram
d. Sequence Diagram
e. Activity Diagram
f. Object Diagram
g. State chart Diagram
h. Component Diagram
i. Deployment Diagram

**11. What** is Unified Approach/ Process? **(NOV/DEC 2018)**

- A software development process describes an approach to building, deploying, and possibly maintaining software. The Unified Process has emerged as a popular iterative software development process for building object-oriented systems.
- The UP combines commonly accepted best practices, such as an **iterative lifecycle and risk-driven development, into a cohesive and well-documented process description.**




**12. State** the reasons for using Unified Process

1. The UP is an iterative process.
2. UP practices provide an example structure for how to do and thus how to explain OOA/D.
3. The UP is flexible, and can be applied in a lightweight and agile approach that includes practices from other agile methods

**13. List** the UP Phases?

A UP project organizes the work and iterations across four major phases:

1. **Inception** - approximate vision, business case, scope, vague estimates.
2. **Elaboration** - refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates.
3. **Construction -** iterative implementation of the remaining lower risk and easier elements, and preparation for deployment.
4. **Transition** - beta tests, deployment.

**14. What** do you mean by Use cases and actors **(Nov /Dec 2011) (Remember) / Define** Use case and Actor **(Nov/Dec 2013, Apr 2018)**

- **Use cases** are requirements, primarily functional or behavioral requirements that indicate what the system will do. A related viewpoint is that a use case defines a contract of how a system will behave.
- **An actor** is anything with behavior, including the system under discussion (SuD) itself when it calls upon the services of other systems.

**15. What** is UP disciplines?

Disciplines are set of activities (and related artifacts) in one subject area, such as the activities within requirements analysis. There are several disciplines in the UP:

- **Business Modeling** - The Domain Model artifact, to visualize noteworthy concepts in the application domain.

- **Requirements -** The Use-Case Model and Supplementary Specification artifacts to capture functional and non-functional requirements.
- **Design -** The Design Model artifact, to design the software objects.

**16. Define** the Development Case

The choice of practices and UP artifacts for a project may be written up in a short document called the Development Case (an artifact in the Environment discipline).

| **Discipline** | **Practice** | **Artifact** | **Incep.** | **Elab.** | **Const.** | **Trans.** |
|---|---|---|---|---|---|---|
| | | **Iteration** | **I1** | **E1..En** | **C1..Cn** | **T1..T2** |
| Business Modeling | Agile modeling req. Workshop | Domain Model | | s | | |
| Requirements | Req. Workshop vision box exercise dot voting | Use-Case Model | s | r | | |
| | | Vision | s | r | | |
| | | Supplementary Specification | s | r | | |
| | | Glossary | s | r | | |
| Design | Agile modeling test-driven dev. | Design Model | | s | r | |
| | | SW Architecture Document | | s | | |
| | | Data Model | | s | r | |

**17. What** is Use case Diagram**? (Remember)(Nov/Dec 2019)**

The UML provides use case diagram notation to illustrate the names of use cases and actors, and the relationships between them

- Use case diagram is an excellent picture of the system context
- It makes a good context diagram that is, showing the boundary of a system, what lies outside of it, and how it gets used.
- It serves as a communication tool that summarizes the behavior of a system and its actors

**18. Define** the following

a) Forward Engineering

b) Reverse Engineering

- In reverse engineering, a UML tool reads the source or binaries and generates (typically) UML package, class, and sequence diagrams. These "blueprints" can help the reader understand the big-picture elements, structure, and collaborations.
- In forward engineering, Before programming, some detailed diagrams can provide guidance for code generation (e.g., in Java), either manually or automatically with a tool. It's common that the diagrams are used for some code, and other code is filled in by a developer while coding (perhaps also applying UML sketching).

**19. State** the need for Use Cases **(Apr/May 2015)**

- Lack of user involvement in software projects is near the top of the list of reasons for project failure. Use cases are a good way to help keep it simple, and make it possible for domain experts or requirement donors to themselves write use cases.

- Another value of use cases is that they emphasize the user goals and perspective; we ask the question "Who is using the system, what are their typical scenarios of use, and what are their goals?" This is a more user-centric emphasis compared to simply asking for a list of system features.

**20. List** out the steps for finding Use cases **(Nov /Dec 2012)**

Use cases are defined to satisfy the goals of the primary actors. Hence, the basic procedure is:

- Choose the system boundary. Is it just a software application, the hardware and application as a unit, that plus a person using it, or an entire organization?
- Identify the primary actors those that have goals fulfilled through using services of the system.
- Identify the goals for each primary actor.
- Define use cases that satisfy user goals; name them according to their goal. Usually, user-goal level use cases will be one-to-one with user goals, but there is at least one exception, as will be examined.

**21. Define** inception step? (**April/May 2011, NOV/DEC 2017)**

Inception - Envision the product scope, vision, and business case. Inception is the initial short step to establish a common vision and basic scope for the project. It will include analysis of perhaps 10% of the use cases, analysis of the critical non-functional requirement, creation of a business case, and preparation of the development environment.

Sample inception artifacts: Vision and Business Case, Use-Case Model, Supplementary Specification, Glossary, Risk List & Risk Management Plan, Prototypes and proof-of-concepts, Iteration Plan, Phase Plan & Software Development Plan, Development Case

**22. List** the various Use case formats.

**Three Common Use Case Formats**

**Brief** - Terse one-paragraph summary, usually of the main success scenario. The prior Process Sale example was brief.
When? During early requirements analysis, to get a quick sense of subject and scope. May take only a few minutes to create.
**Casual** - Informal paragraph format. Multiple paragraphs that cover various scenarios. The prior Handle Returns example was casual.
When? As above.
**fully dressed** - All steps and variations are written in detail, and there are supporting sections, such as preconditions and success guarantees.
When? After many use cases have been identified and written in a brief format, then during the first requirements workshop a few (such as 10%) of the architecturally significant and high-value use cases are written in detail.

**23. List** the artifacts prepared during Inception Phase

inception artifacts: Vision and Business Case , Use-Case Model , Supplementary Specification, Glossary , Risk List & Risk Management Plan, Prototypes and proof-of-concepts , Iteration Plan , Phase Plan & Software Development Plan , Development Case

**24. List** the elements of fully dressed Use case Format**.**

| **Use Case Section** | **Comment** |
|---|---|
| Use Case Name | Start with a verb. |
| Scope | The system under design. |
| Level | "user-goal" or "subfunction" |
| Primary Actor | Calls on the system to deliver its services. |
| Stakeholders and Interests | Who cares about this use case, and what do they want? |
| Preconditions | What must be true on start, and worth telling the reader? |
| Success Guarantee | What must be true on successful completion, and worth telling the reader. |
| Main Success Scenario | A typical, unconditional happy path scenario of success. |
| Extensions | Alternate scenarios of success or failure. |
| Special Requirements | Related non-functional requirements. |
| Technology and Data Variations List | Varying I/O methods and data formats. |
| Frequency of Occurrence | Influences investigation, testing, and timing of implementation. |
| Miscellaneous | Such as open issues. |

**25. Define** the following: 1) Concrete Usecase 2) Abstract Usecase

A **concrete use case** is initiated by an actor and performs the entire behavior desired by the actor. These are the elementary business process use cases. For example, Process Sale is a concrete use case. An **abstract use case** is never instantiated by itself; it is a subfunction use case that is part of another use case. Handle Credit Payment is abstract; it doesn't stand on its own, but is always part of another story, such as Process Sale.

**26. How** to find the Useful Use cases?

To find useful usacases, there are several rules of thumb, including:

- The Boss Test - To check for achieving results of measurable value
- The EBP Test – EBP (business process engineering field) is similar to the term user task in usability engineering, although the meaning is less strict in that domain.
- The Size Test

**27. List** the relationships used in use cases **(May/June 2012)**

Relationships

1. The include Relationship
2. The extend Relationship
3. The generalize Relationship

**28. Define** Extend relationship between use cases **(Apr/May 2017)**

The idea is to create an extending or addition use case, and within it, describe where and under what condition it extends the behavior of some base use case.

This is an example of an extend relationship. The use of an extension point, and that the extending use case is triggered by some condition. Extension points are labels in the base use case which the extending use case references as the point of extension, so that the step numbering of the base use case can change without affecting the extending use case.

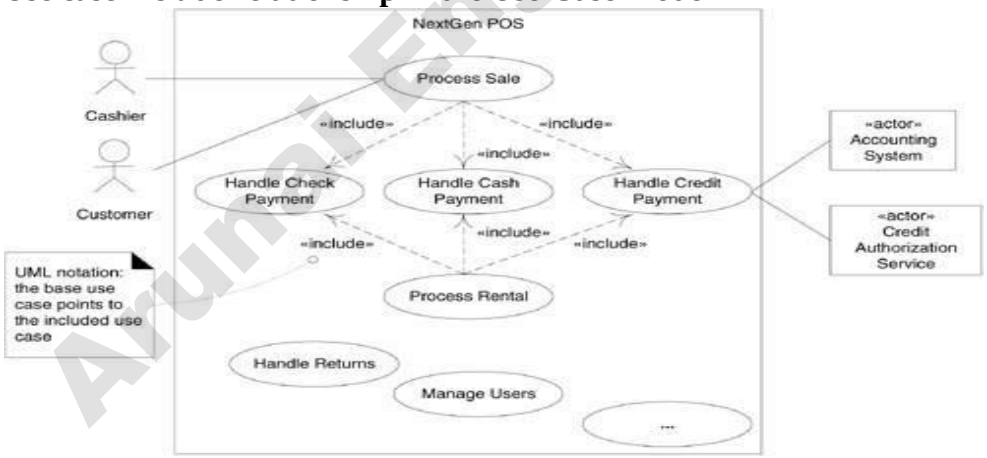

**29. Define** include relationship b/w use cases **(Apr/May 2017)**

**use cases use the include relationship when:**

- They are duplicated in other use cases.

A use case is very complex and long, and separating it into subunits aids comprehension.

**Use case include relationship in the Use-Case Model**




**30. List** out the components of POS system **(Apr 2018)**

POS hardware includes display screen, customer display screen, cash drawer, swiping device (for credit cards), printer, computer and a bar code reader. All of these are components of POS terminal, the crucial component of POS hardware.

**31. What** is an Object Modeling Language**? (NOV/DEC 2018)**

An object-modeling language is a standardized set of symbols used to model a software system using an object-oriented framework.

A modeling language is usually associated with a methodology for object-oriented development. The modeling language defines the elements of the model. E.g., that a model has classes, methods, object properties, etc.
Ex: the most important object modeling language standards: the Unified Modeling Language (UML).

**32. List** any two features of object based languages**. (Nov/Dec 2018)**

- Object-based language support all the features of OOPs (Classes , Objects , Abstraction, Encapsulation , message Passing) except Polymorphism and Inheritance.
- Object-based language has an in-built object like javascript has a window object.
  Ex: Object-based languages are Javascript, VB, etc.

**Part-B**

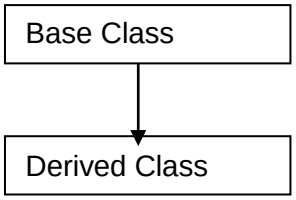
1. Briefly explain the different phases of unified process**?(April/May 2011, May/June 2012, May/June 2013, Nov/Dec 2013,Nov/Dec 2015,Nov/Dec 2016) (Understand)**
2. List various UML Diagrams and explain the purpose of each diagram(May/June 2014,Apr/May 2017) (Remember)
3. What do you mean by unified process in OOAD? Explain the phases with suitable diagrams **(Nov/Dec 2011, Nov/Dec 2012,Apr/May 2017)**
4. Explain about Use Case Model for case study for your choice. (**Nov/Dec 2015).**
5. i) What is UML. (2) **(Apr/May 2015)**
   **ii)** Explain the include, extend and generalization relationship with an example. (6)
6. What is the Unified process? Is the UP iterative and incremental? Explain. (8) **(Apr/May 2015)**
7. Present an outline of object oriented analysis and Object Oriented Design**. (NOV/DEC 2017)**
8. Why the Unified process has emerged as a popular and effective software development process? **(NOV/DEC 2017)**
9. Explain about Use Case Model for case study for your choice. (**Nov/Dec 2015).**
10. What is the Unified process? Is the UP iterative and incremental? Explain. (8) **(Apr/May 2015)**
11. i)Present an outline of object oriented analysis and Object Oriented Design**. (NOV/DEC 2017)**
    **ii)** Why the Unified process has emerged as a popular and effective software development process? **(NOV/DEC 2017)**
12. Explain about NEXTGEN POS SYSTEM?
13. Explain the relationship between use cases.
14. Explain the various use case formats.

15. Explain with an example generalization and specialization and write note on abstract class and abstract operation. (8) **(NOV/DEC 2017)**
16. i.Explain in detail about use case diagrams. (6) **(APR /MAY 2018)**
    **ii)** Explain the software development lifecycle of object oriented approach. (7) **(APR /MAY 2018)**
17. Explain briefly the elements of Use Case diagram. (13) **(Understand) (NOV /DEC 2018)**
18. How do you see the application of UML diagram for Iterative Software Development? Explain.) **(NOV /DEC 2018)**
19. **i)** Outline the steps to be followed to identify actors and use cases.
    **ii)** What is inception? Outline the tasks that a project team performs during inception? **(Nov/Dec 2019)**

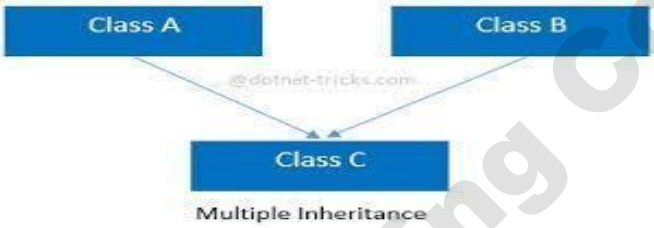
**UNIT – II**
**STATIC UML DIAGRAMS**
**PART – A**

1. **Differentiate single and multiple inheritance (Nov/Dec 2012)**

Single inheritance :A class derived from only one base class




Multiple inheritance : A class derived from more than one base class




2. **List** the relationship used in class diagram?**(April/May 2011, Nov/Dec 2013, May/June 2014)**

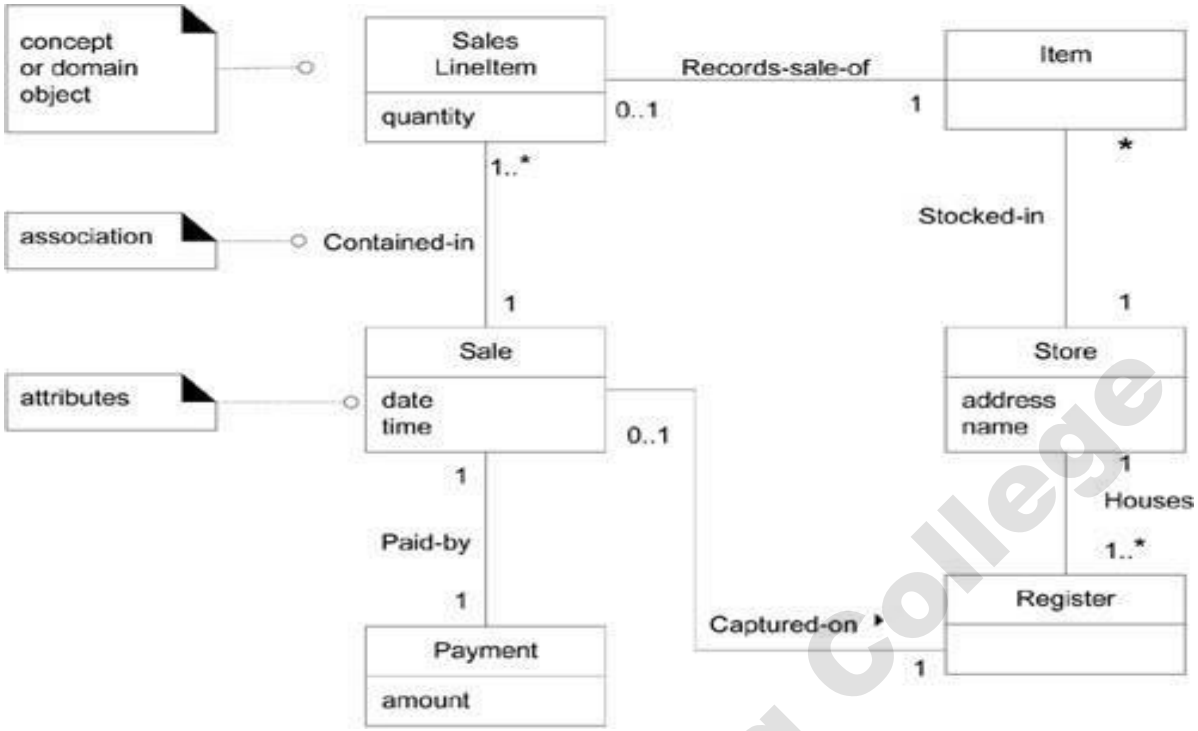
   Association, Generalization , Dependency ,Interface Realization, Composition and aggregation.

3. **What** is UML Method & Operation**?**

- A UML operation is a declaration, with a name, parameters, return type, exceptions list, and possibly a set of constraints of pre-and post-conditions. But, it is not an implementation rather, methods are implementations.
- A UML method is the implementation of an operation; if constraints are defined, the method must satisfy them. A method may be illustrated several ways, including:
  - In interaction diagrams, by the details and sequence of messages
  - In class diagrams, with a UML note symbol stereotyped with «method»

4. **What** is a domain model**?(April/May 2011, Nov/Dec 2013,Apr/May 2015)**

- A domain model is a visual representation of conceptual classes or real-world objects in a domain of interest. They have also been called conceptual models, domain object models, and analysis object models.
- A domain model is illustrated with a set of class diagrams in which no operations (method signatures) are defined. It provides a conceptual perspective. It may show:
  - domain objects or conceptual classes
  - associations between conceptual classes
  - attributes of conceptual classes




5. **Define** aggregation and composition**?(April/May 2011 , May/June 2012, May/June 2013, Nov/Dec 2013, May/June 2014)**

- Aggregation is a vague kind of association in the UML that loosely suggests whole-part relationships (as do many ordinary associations). It has no meaningful distinct semantics in the UML versus a plain association, but the term is defined in the UML.
- Composition, also known as composite aggregation, is a strong kind of whole-part aggregation and is useful to show in some models. A composition relationship implies that 1) an instance of the part belongs to only one composite instance at a time, 2) the part must always belong to a composite and 3) the composite is responsible for the creation and deletion of its parts either by itself creating/deleting the parts, or by collaborating with other objects.


6. **What is** elaboration? **(or) What** are the tasks performed in elaboration **(May/June 2012, May/June 2013, May/June 2014, Nov/Dec 2015, Apr 2018)**

Elaboration is the initial series of iterations during which, on a normal project:

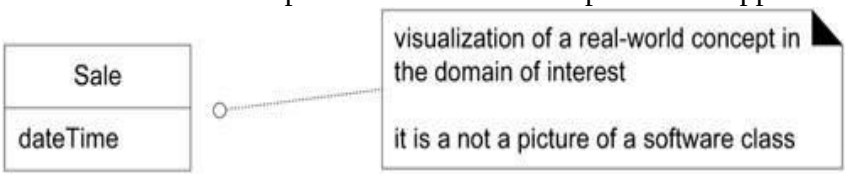
- The core, risky software architecture is programmed and tested
- The majority of requirements are discovered and stabilized
- The major risks are mitigated or retired

**Sample elaboration artifacts:** Domain Model , Design Model , Software Architecture Document, Data Model, Use-Case Storyboards, UI Prototypes.

7. **What** is Conceptual Class**?**

A conceptual class is an idea, thing, or object. It may be considered in terms of its symbol, intension, and extension

- Symbol words or images representing a conceptual class.
- Intension the definition of a conceptual class.
- Extension the set of examples to which the conceptual class applies.



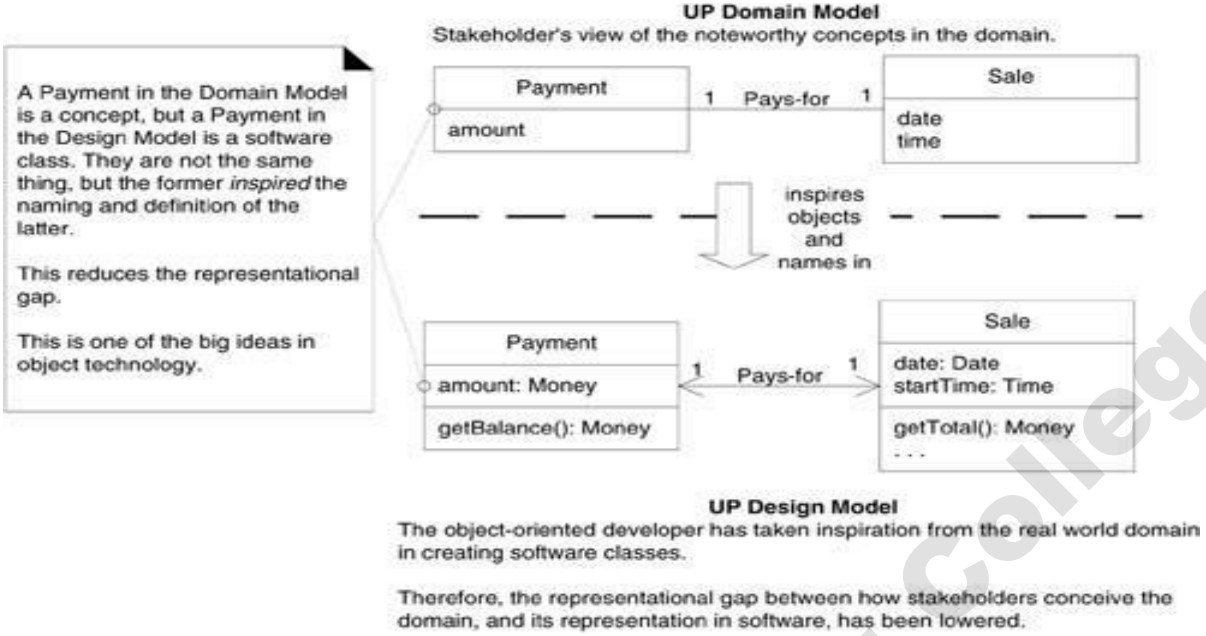

8. **How** to Create a Domain Model**? (Nov/Dec 2015, Nov/Dec 2016)**
   Bounded by the current iteration requirements under design:
   1. Find the conceptual classes (see a following guideline).
   2. Draw them as classes in a UML class diagram.
   3. Add associations and attributes.

9. **List** the methods to find Conceptual Classes**.**
   Three Strategies to Find Conceptual Classes:
   1. Reuse or modify existing models. This is the first, best, and usually easiest approach. There are published, well-crafted domain models and data models for many common domains, such as inventory, finance, health, and so forth.
   2. Use a category list.
   3. Identify noun phrases.

10. **Write the Guidelines to create a domain model. (Nov/Dec 2015)**
    - Agile Modeling Sketching a Class Diagram
    - Agile Modeling Maintain the Model in a Tool?
    - Report Objects - Include 'Receipt' in the Model?
    - Use Domain Terms
    - How to Model the Unreal World?
    - A Common Mistake with Attributes vs. Classes

11. **Why is it necessary to create a domain model?**

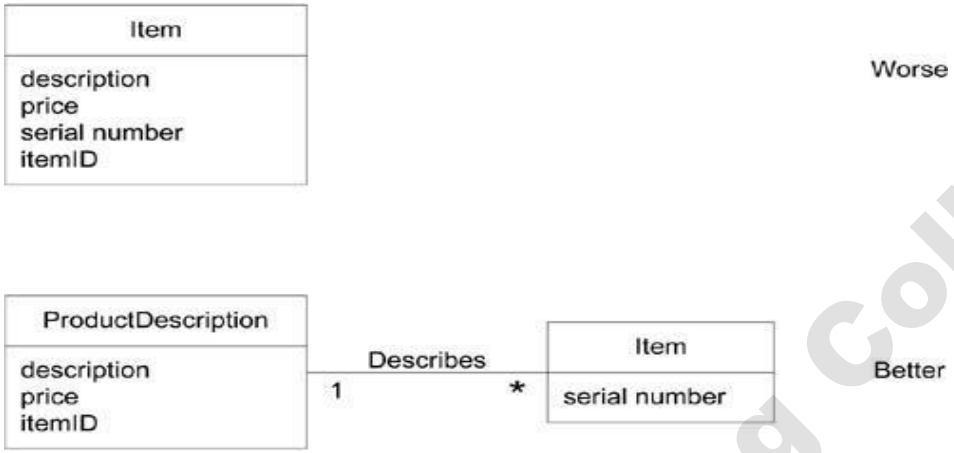
**Lower Representational Gap with OO Modeling:**. This supports a low representational gap between our mental and software models.



12. **What is** Description class**?**

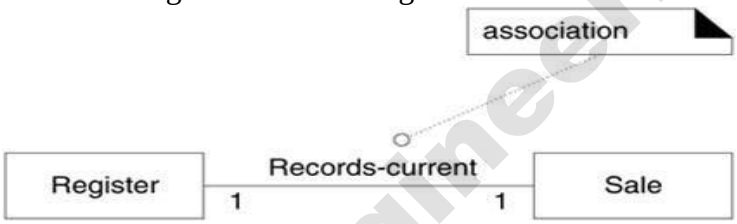
Add a description class (for example, ProductDescription) when:

- There needs to be a description about an item or service, independent of the current existence of any examples of those items or services.
- Deleting instances of things they describe (for example, Item) results in a loss of information that needs to be maintained, but was incorrectly associated with the deleted thing.
- It reduces redundant or duplicated information.




13. **Define** Association**. (Remember)**

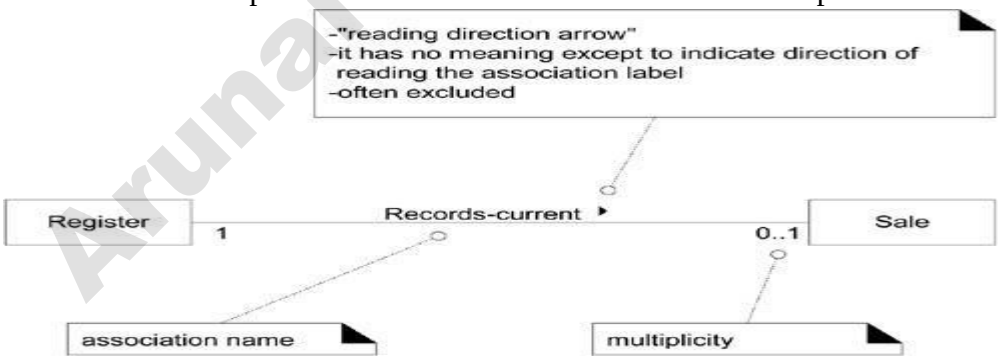
An **association** is a relationship between classes (more precisely, instances of those classes) that indicates some meaningful and interesting connection




In the UML, associations are defined as "the semantic relationship between two or more classifiers that involve connections among their instances."

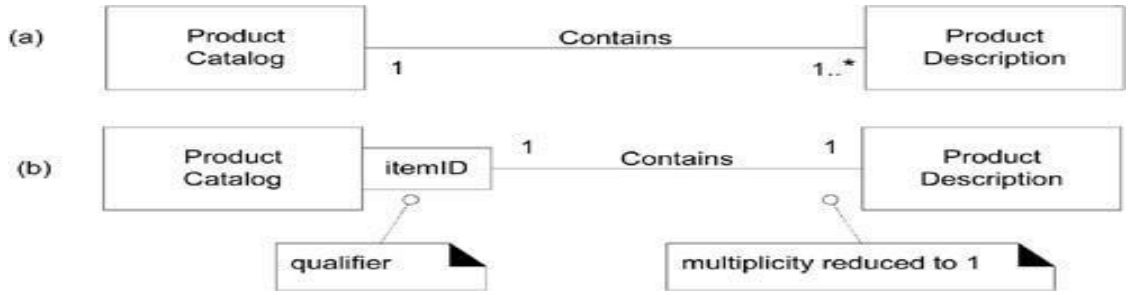
14. **Draw** the UML representation of Association

An association is represented as a line between classes with a capitalized association name.




15. **What** is qualified association**?**

A qualifier may be used in an association; it distinguishes the set of objects at the far end of the association based on the qualifier value. An association with a qualifier is a **qualified association**.




**16. What** is Reflexive Associations**?**

A concept may have an association to itself; this is known as a reflexive association


**17. Give** the hint to identify the attributes of a class **( Nov/Dec 2011)**

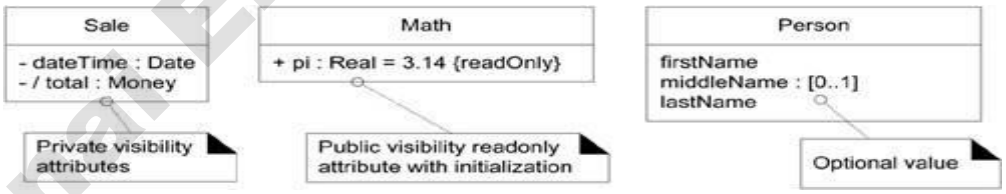
- Focus on Data Type Attributes in the Domain Model-Attributes in a domain model should preferably be data types
- Relate conceptual classes with an association, not with an attribute.
- No Attributes Representing Foreign Keys
- Modeling Quantities and Units

**18. Write** the attribute representation in UML

Attributes are shown in the second compartment of the class box. Their type and other information may optionally be shown.The full syntax for an attribute in the UML is:

**Visibility name : type multiplicity = default {property-string}**

**Attribute notation in UML.**

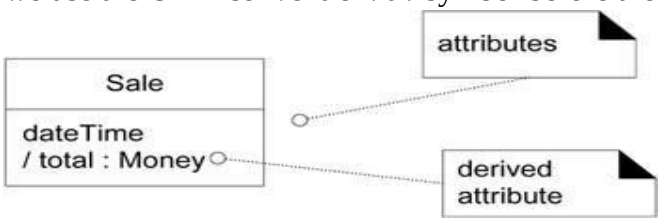



{readOnly} is probably the most common property string for attributes.

Multiplicity can be used to indicate the optional presence of a value, or the number of objects that can fill a (collection) attribute.

**19. What** is derived attributes**? (Remember)**

**Derived Attributes:** When we want to communicate that 1) this is a noteworthy attribute, but 2) it is derivable, we use the UML convention: a / symbol before the attribute name.



**20. When** to define New Data Type Classes**?**

Guidelines for modeling data types:

Represent what may initially be considered a number or string as a new data type class in the domain model if:

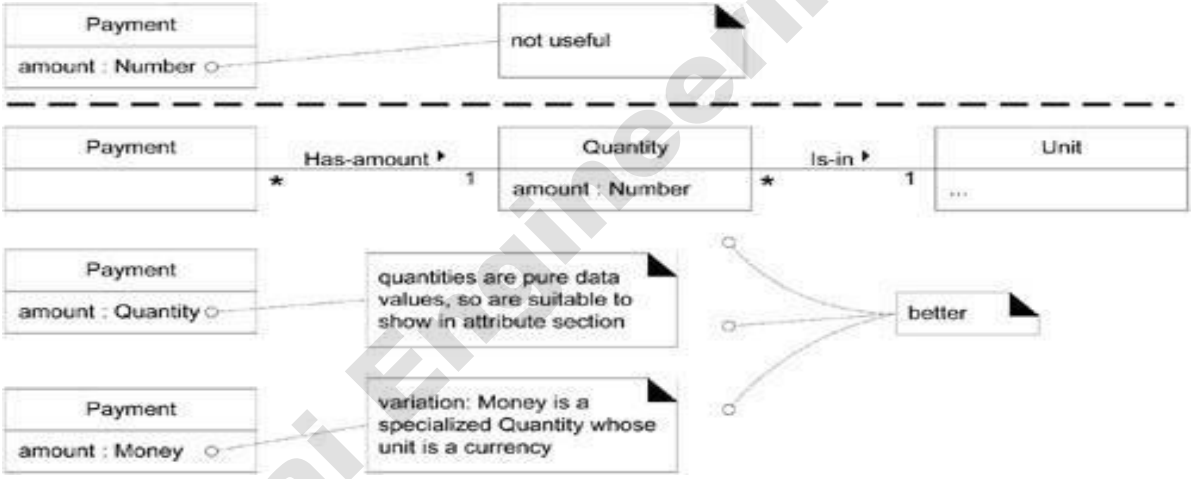
- It is composed of separate sections.
  - phone number, name of person
- There are operations associated with it, such as parsing or validation.
  - social security number
- It has other attributes.
  - promotional price could have a start (effective) date and end date
- It is a quantity with a unit.
  - payment amount has a unit of currency
- It is an abstraction of one or more types with some of these qualities.

In the NextGen POS system an itemID attribute is needed; it is probably an attribute of an Item or ProductDescription. Casually, it seems like just a number or perhaps a string. For example, itemID:Integer or itemID:String.

**21. How** to model quantities and units**? (Remember)**

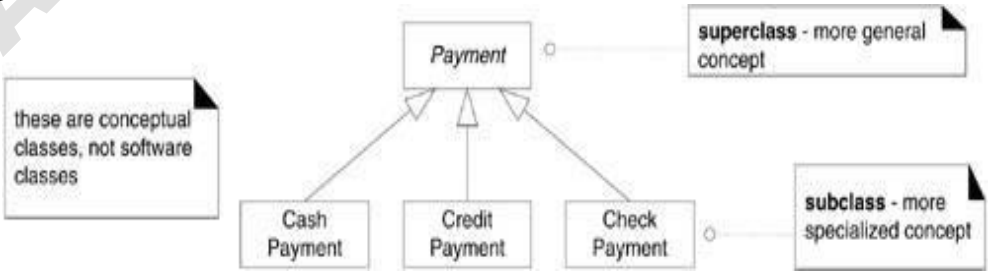
Modeling Quantities and Units

Most numeric quantities should not be represented as plain numbers. Consider price or weight. These are quantities with associated units, and it is common to require knowledge of the unit to support conversions.




**22. Define** Generalization**. (NOV/DEC 2018)**

Generalization is the activity of identifying commonality among concepts and defining super class (general concept) and subclass (specialized concept) relationships. Identifying a super class and subclasses is of value in a domain model because their presence allows us to understand concepts in more general, refined and abstract terms.



23. **Define** 100% rule, IS a Rule, Correct Conceptual Subclass**.**

**100% Rule:** 100% of the conceptual super class's definition should be applicable to the subclass. The subclass must conform to 100% of the super class's:

Attributes
associations

**Is-a Rule:** All the members of a subclass set must be members of their superclass set. In natural language, this can usually be informally tested by forming the statement: Subclass is a Super class.

**Correct Conceptual Subclass:**

A potential subclass should conform to the:

- 100% Rule (definition conformance)
- Is-a Rule (set membership conformance)

24. **When** to create conceptual subclass **(NOV/DEC 2017)**

Create a conceptual subclass of a super class when:

1. The subclass has additional attributes of interest.
2. The subclass has additional associations of interest.
3. The subclass concept is operated on, handled, reacted to, or manipulated differently than the super class or other subclasses, in ways that are of interest.
4. The subclass concept represents an animate thing (for example, animal, robot) that behaves differently than the superclass or other subclasses, in ways that are of interest.

25. **When** to Define a Conceptual Super class**?**

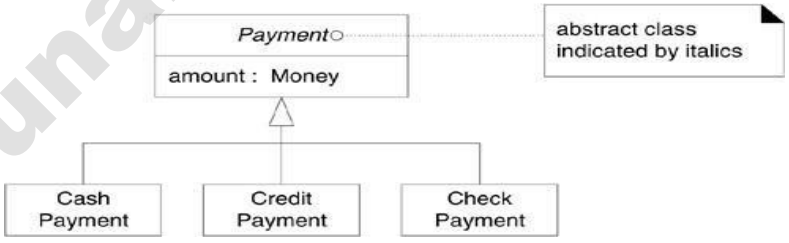
**Motivations to generalize and define a super class: Guideline** Create a super class in a generalization relationship to subclasses when:

- The potential conceptual subclasses represent variations of a similar concept.
- The subclasses will conform to the 100% and Is-a rules.
- All subclasses have the same attribute that can be factored out and expressed in the superclass.
- All subclasses have the same association that can be factored out and related to the superclass.

26. **Give** the Abstract Class Notation in the UML**.**

To review, the UML provides a notation to indicate abstract classes the class name is italicized

**Abstract class notation.**




27. **Why** Call a Domain Model a "Visual Dictionary"**?(Nov/Dec 2016)**

The information it illustrates could alternatively have been expressed in plain text. But it's easy to understand the terms and especially their relationships in a visual language, since our brains are good at understanding visual elements and line connections. Therefore, the domain model is a visual dictionary of the noteworthy abstractions, domain vocabulary, and information content of the domain.

**28. What** is the use of system sequence diagram**?(April/May 2011, May/June 2012, May/June 2014,Apr/May 2015) (Remember)**

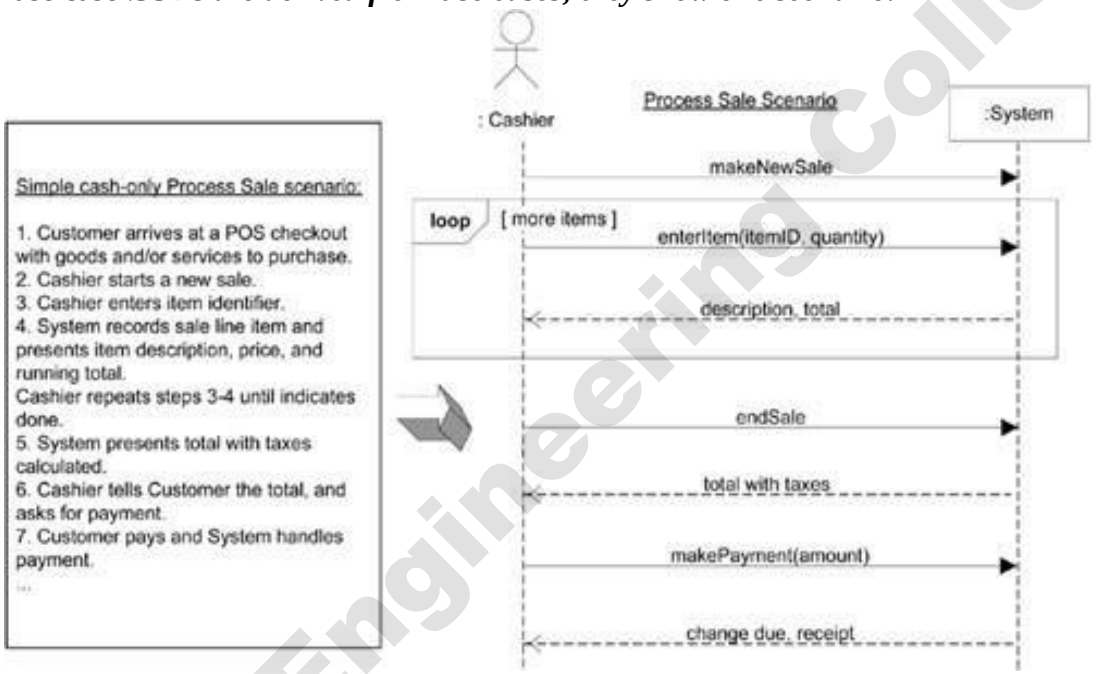
A software system reacts to three things:

1) External events from actors (humans or computers),
2) Timer events,
3) Faults or exceptions (which are often from external sources).

Therefore, it is useful to know what, precisely, are the external input events the system events. They are an important part of analyzing system behavior. System behavior is a description of what a system does, without explaining how it does it. One part of that description is a system sequence diagram.

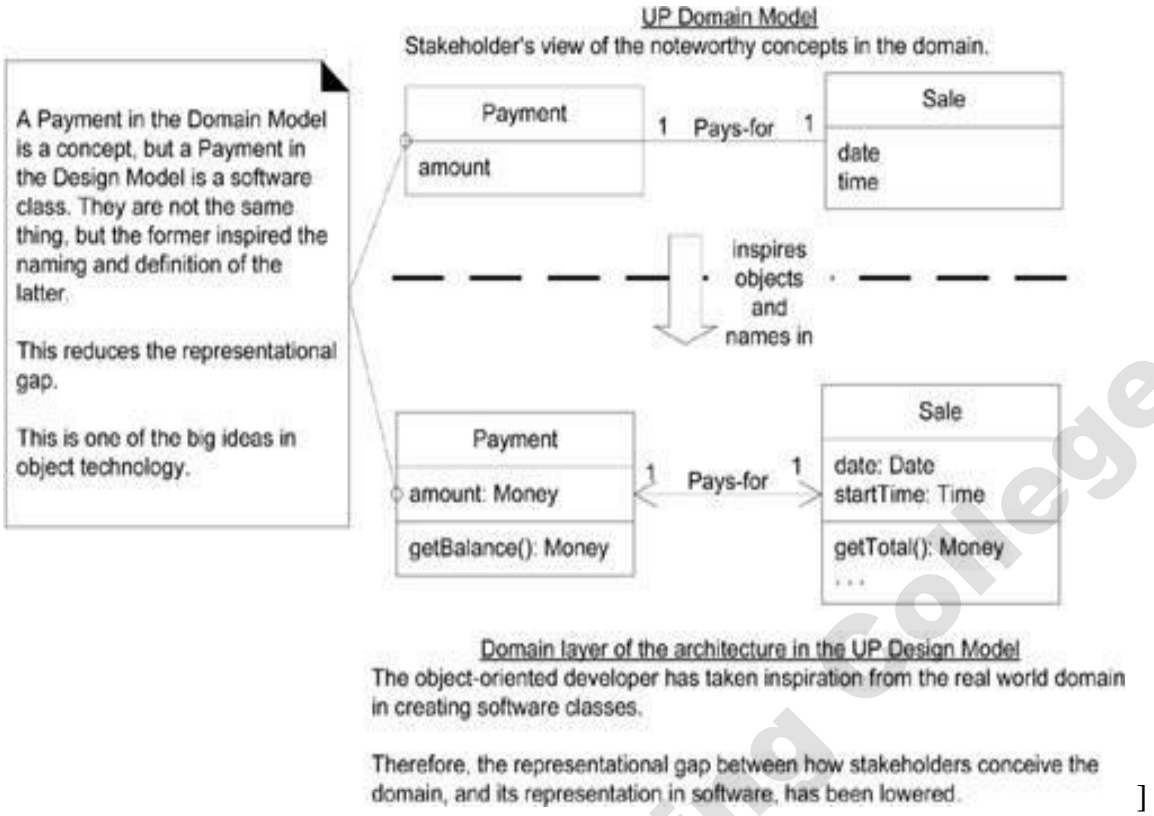
**29. What** is the relationship between SSD and Use case**?**

An SSD shows system events for one scenario of a use case, therefore it is generated from inspection of a use case .***SSDs are derived from use cases; they show one scenario.***




**30. What** is the Relationship Between the Domain Layer and Domain Model?

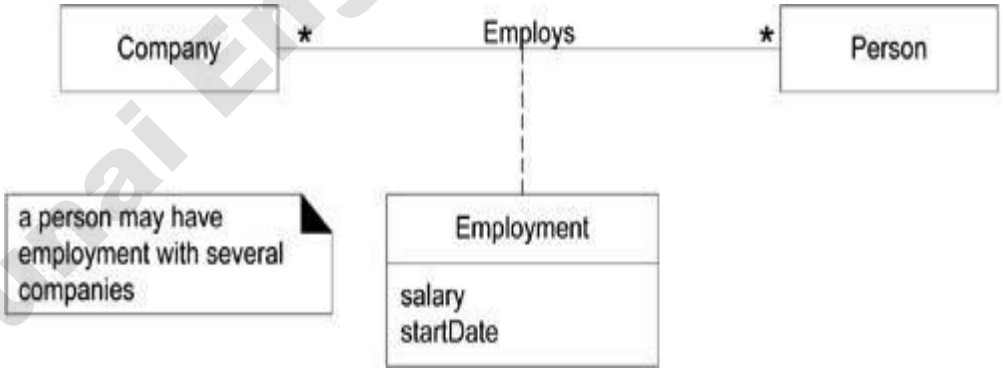
By creating a domain layer with inspiration from the domain model, we achieve a lower representational gap, between the real-world domain, and our software design



]

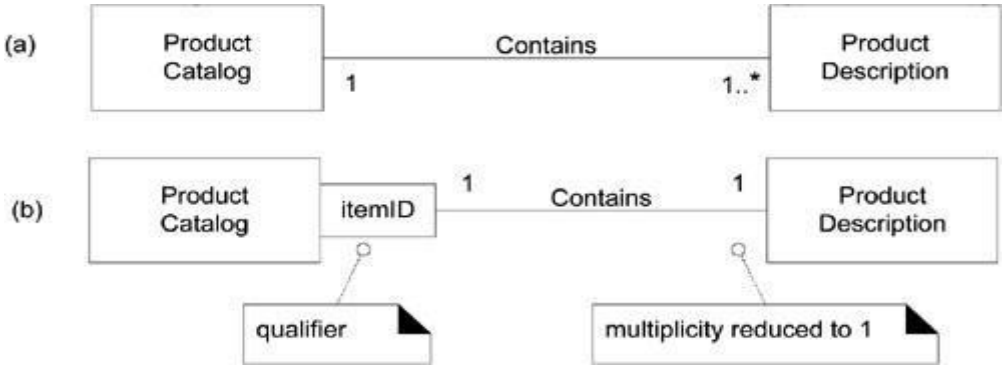
**31. What** is an association class**? (Nov/Dec 2019)**

An association class allows you treat an association itself as a class, and model it with attributes, operations, and other features. For example, if a Company employs many Persons, modeled with an Employs association, you can model the association itself as the Employment class, with attributes such as start Date.




**32. What** is qualified association**?**

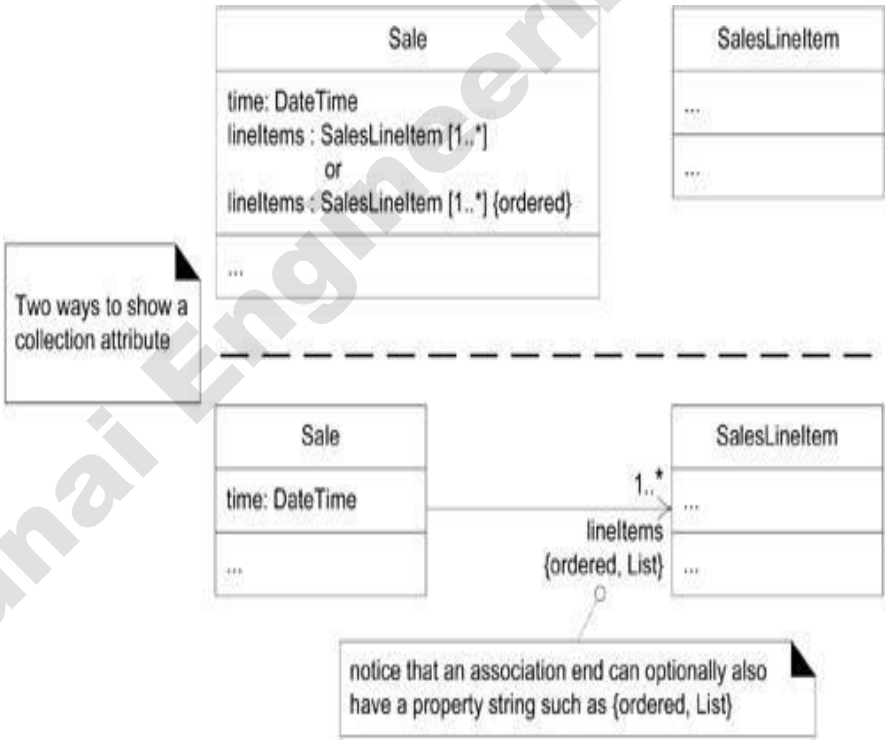
A qualified association has a qualifier that is used to select an object (or objects) from a larger set of related objects, based upon the qualifier key.




**33. How** to Show Collection Attributes with Attribute Text and Association Lines**?**

Suppose that a Sale software object holds a List (an interface for a kind of collection) of many SalesLineItem objects. For example, in Java:

```
public class Sale
{
private List<SalesLineItem> lineItems = new ArrayList<SalesLineItem>();
// …
}
```




**34. How** to Naming System Events and Operations? (**Nov/Dec 2016)**

- System events (and their associated system operations) should be expressed at the level of intent rather than in terms of the physical input medium or interface widget level.
- It also improves clarity to start the name of a system event with a verb Thus "enter item" is better than "scan" (that is, laser scan) because it captures the intent of the operation while remaining abstract and noncommittal with respect to design choices about what interface is used to capture the system event.

**35. Define** System Events and the System Boundary**. (Nov/Dec 2016)**

To identify system events, it is necessary to be clear on the choice of system on use cases. For the purposes of software development, the system boundary is usually chosen to be the software system itself; in this context, a system event is an external event that directly stimulates the software.

**36.Define** Object. **(NOV/DEC 2018/Nov/Dec 2019)**

- Objects are the things you think about first in designing a program and they are also the units of code that are eventually derived from the process
- Each object is an instance of a particular class or subclass with the class's own methodsor procedures and data variables.
- Object determines the behavior of the class. When you send a message to an object, you are asking the object to invoke or execute one of its methods.

  Ex: Birds - Class, Crow - object

**36. What** is meant by Attributes? (Remember) **(NOV/DEC 2018)**

An **attribute** is a logical data value of an object. Include attributes that the requirements suggest or imply a need to remember information. For example, a receipt (which reports the information of a sale) in the Process Sale use case normally includes Therefore,

Sale needs a dateTime attribute.
Store needs a name and address.
Cashier needs an ID.

**37. What** is the difference between a class and an object? **(NOV/DEC 2018)**

| No. | Object | Class |
|---|---|---|
| 1) | Object is an instance of a class. | Class is a blueprint from which objects are created. |
| 2) | Object is a real world entity such as pen, laptop, mouse, chair etc. | Class is a group of similar objects. |
| 3) | Object is a physical entity. | Class is a logical entity. |
| 5) | Object is created many times as per requirement. | Class is declared once. |
| 6) | Object allocates memory when it is created. | Class doesn't allocated memory when it is created. |

**38.** Define Multiplicity of an Association **(Nov/Dec 2019).**

**Association** end **multiplicity** defines the number of entity type instances that can be at one end of an **association**. An **association** end **multiplicity** can have one of the following values: ... many (*): Indicates that zero, one, or more entity type instances exist at the **association** end

**39.** Outline the advantages of modeling state machine diagram.(**Understand)(Nov/Dec 2019)**

**State diagrams** are the ideal way to **model** object life cycles.

**State diagrams** enable you to describe the behavior of objects during their entire life span. In addition, the different **states** and **state** changes as well as events causing transitions can be described

**PART-B**

1. For the Hospital Management system draw the following UML diagrams **(Nov/Dec 2012)**
   Conceptual Class Diagram (overall system)
2. Explain the following with an example (**Nov/Dec 2011**)
   Conceptual class diagram
3. Explain with an example, how use case modeling is used to describe functional requirements. Identify the actors, scenario and use cases for the example**? (Library Management system)(April/May 2011, May/June 2012, May/June 2013, Nov/Dec 2013, May/june 2014,Nov/Dec 2016) (Understand)**
4. Describe the strategies used to identify conceptual classes. Describe the steps to create a domain model used for representing conceptual classes. **?(April/May 2011, may/june 2012, may/June 2013, Nov/Dec 2013, May/june 2014,Nov/Dec 2016) (Understand)**
5. Write briefly about elaboration and discuss the difference between Elaboration and inception with neat diagram (or) examples/for University Domain**. ( May/June 2014,Nov/Dec 2015,Apr/May 2017) (Remember)**
6. Explain the method of identifying the classes using the common class approach with an example **(Nov/Dec 2012) (Understand)**
7. Explain Domain Model Refinement with an example **(Apr/May 2015)? (Understand)**
8. Explain the different kinds of relationships between classes (generalization., dependency, realization, association, composition, aggregation)
9. Construct Design for Library Information System which comprises and Following notations **(Nov/Dec 2015, Nov/Dec 2016).**
   i) Aggregations
   ii) Compositions
   iii) Associations
10. Differentiate (or) Explain Aggregations and compositions with suitable example ( **Apr/May 17)(Nov/Dec 2019)**
11. Discuss about aggregation and composition. (7) **(APR /MAY 2018)**
12. Describe the strategies used to identify the conceptual classes. Describe the steps to create a domain model used for representing the conceptual classes. (13) **(APR /MAY 2018)**
13. Draw and discuss an analysis model for Banking system. (8) **(APR /MAY 2018)**
14. With an example, Explain the need for Activity Diagram. (13) (**NOV /DEC 2018)**

15. Draw class diagram,activity diagram and component diagram for designing this system.(13) (Apply) **(APR /MAY 2018)**
16. Explain in detail about the interaction diagrams and also notations.(13) (Understand) (APR /MAY **2018)**
17. Illustrate with an example,the relationship between sequence diagram and use cases.(13) **(Understand) (APR /MAY 2018)**
18. Explain details about various static and dynamic UML important diagram with suitable example.(13) **(Understand) (APR /MAY 2018)**
19. Draw and explain the class diagram for a banking application.(13) **(Remember) (NOV /DEC 2018)**
20. What is Domain Model Refinement? Explain with suitable examples.(13) **(Understand) (NOV /DEC 2018)**
21. How will you find conceptual class hierarchies? Give example. **(13) (Remember) (NOV /DEC 2018)**
22. Discuss the relationship between sequence diagram and class diagram. **(13) (Understand) (NOV /DEC 2018)**
23. Elaborate Generalization and specialization with an example(6)**(Understand)(Nov/Dec 2019)**
24. Outline the steps in modeling sequence diagram with an example(13)**(Remember)(Nov/Dec 2019)**
25. Develop a Usecase model for the activities involved in ordering food in a restaurant from the point when the customer enters the restaurant to the point when he leaves the restaurant**(15)(Create)(Nov/Dec 2019)**
26. Model the Class diagram for "Library management System ".State the functional requirements you are considering**.(15)(Nov/Dec 2019)**

## UNIT – III
## DYNAMIC AND IMPLEMENTATION UML DIAGRAMS
## PART A

1. **What is** UML Activity Diagram? Mention the elements of Activity Diagram **(Apr 2018)**

A UML activity diagram shows sequential and parallel activities in a process. They are useful for modeling business processes, workflows, data flows, and complex algorithms.
Elements:
Start, End, activity, object, fork, join, decision, merge, time signal rake,
sub activity, accept signal.

2. **How** to Apply Activity Diagrams**?**

A UML activity diagram offers rich notation to show a sequence of activities, including parallel activities.

- Business Process Modeling
- Data Flow Modeling
- Concurrent Programming and Parallel Algorithm Modeling

3. **Define** swim lane **(Nov/Dec 2011, Nov/Dec 2012)**

Swim lanes divide activity diagrams into sections. Each swim lane is separated from adjacent swim lanes by vertical, solid lines on both sides.

4. **When** to use UML Collaboration diagram **(Apr 2018)**

- To model collaborations between objects or roles that deliver the functionalities of use cases and operations
- To model mechanisms within the architectural design of the system
- To capture interactions that show the messages passing between objects and roles within the collaboration
- To model alternative scenarios within use cases or operations that involve the collaboration of different objects and interactions
- To support the identification of objects (hence classes) that participate in use cases

5. **What** is the use of Interaction Diagram? **(May/June 2013)**

The UML includes interaction diagrams to illustrate how objects interact via messages. They are used for dynamic object modeling. There are two common types: sequence and communication interaction diagrams.

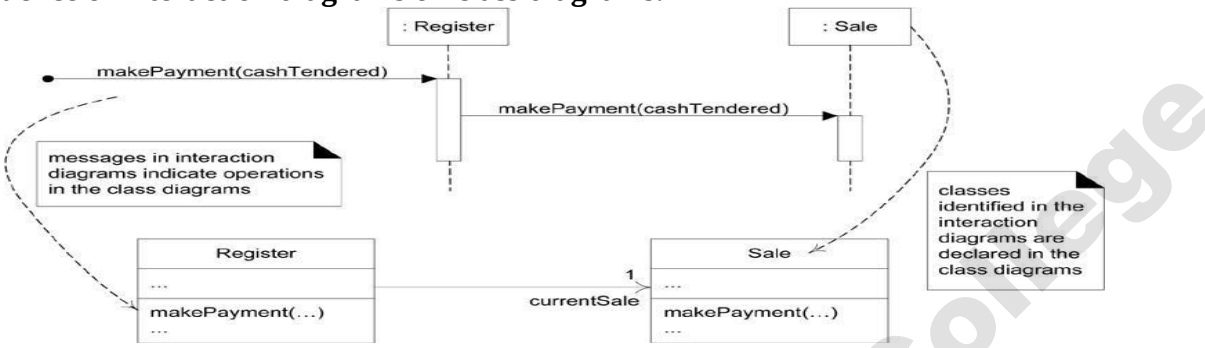
6. **What** are interactive diagrams? List out the components involved in interactive diagram (**Nov/Dec 2012).**

The UML includes interaction diagrams to illustrate how objects interact via messages. They are used for dynamic object modeling. There are two common types: sequence and communication interaction diagrams.

7. **What** is the Relationship Between Interaction and Class Diagrams?

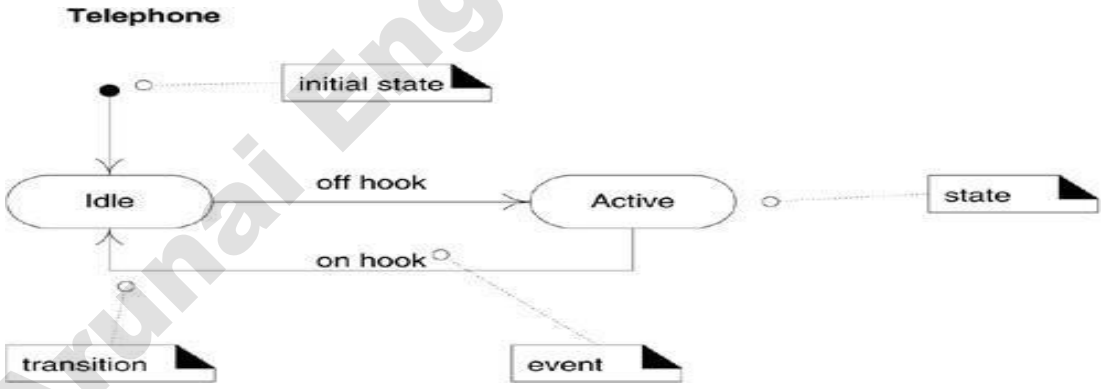
When we draw interaction diagrams, a set of classes and their methods emerge from the creative design process of dynamic object modeling. For example, if we started with the make Payment sequence diagram, we see that a Register and Sale class definition in a class diagram can be obviously derived.

**The influence of interaction diagrams on class diagrams.**



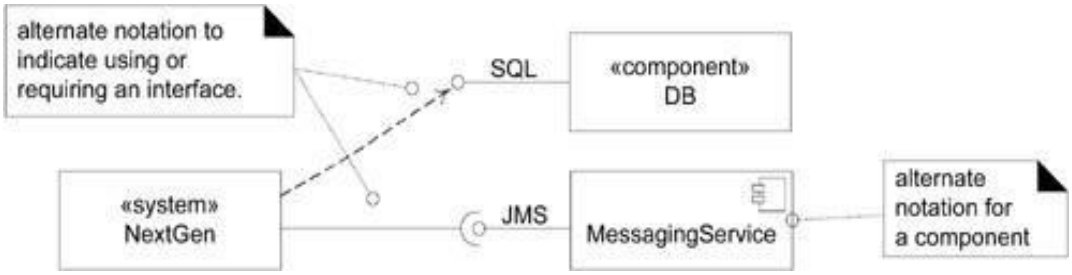

8. **Give** the meaning of event, state, transition**? (April/May 2011, may/June 2012, May/June 2013, Nov/Dec 2013)**

- An **event** is a significant or noteworthy occurrence. For example: A telephone receiver is taken off the hook.
- A **state** is the condition of an object at a moment in time, the time between events. For example: A telephone is in the state of being "idle" after the receiver is placed on the hook and until it is taken off the hook.
- A **transition** is a relationship between two states that indicates that when an event occurs, the object moves from the prior state to the subsequent state. For example: When the event "off hook" occurs, transition the telephone from the "idle" to "active" state.



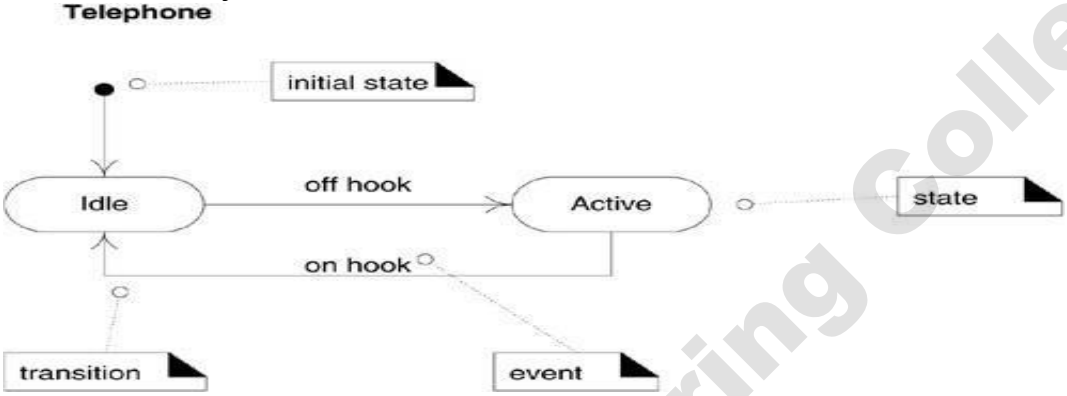

9. **What** is the use of component diagram **(May/June 2012, May/June 2013) / Define** Component with an example **(Nov/Dec 2011, Nov/Dec 2012)**

The Component Diagram helps to model the physical aspect of an Object-Oriented software system. It illustrates the architectures of the software components and the dependencies between them. Those software components including run-time components, executable components are also the source code components.




**10. Define** State Diagram**. (Nov/Dec 2012)**

A UML state machine diagram, illustrates the interesting events and states of an object, and the behavior of an object in reaction to an event.




**11. What** is state –Independent Object and state –dependent Object

If an object always responds the same way to an event, then it is considered state-independent (or modeless) with respect to that event.
If for all events of interest, an object always reacts the same way, it is a state-independent object.
By contrast, state-dependent objects react differently to events depending on their state or mode.

**12. How** to apply state Machine diagram**? / Give** the use of UML State Diagram (**May/june 2014)**

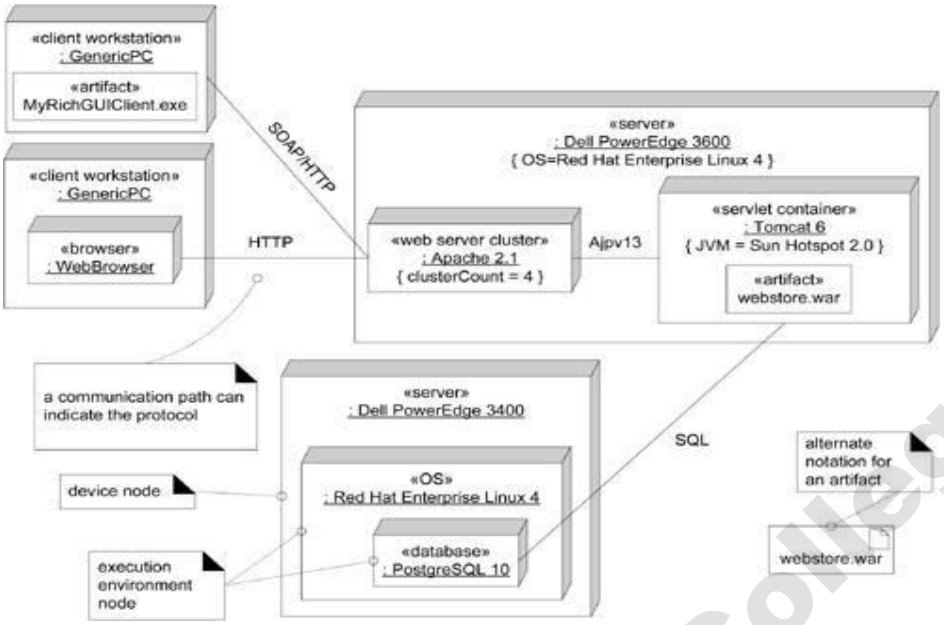
State machines are applied in two ways:

1. To model the behavior of a complex reactive object in response to events.
2. To model legal sequences of operations, protocol or language specifications.

This approach may be considered a specialization of #1, if the "object" is a language, protocol, or process. A formal grammar for a context-free language is a kind of state machine.

**13. What** is deployment diagram & draw its representation**? (Remember)(Nov/Dec 2019)**

A deployment diagram shows the assignment of concrete software artifacts (such as executable files) to computational nodes (something with processing services). It shows the deployment of software elements to the physical architecture and the communication (usually on a network) between physical elements.




**14. What** is Execution Environment Node?

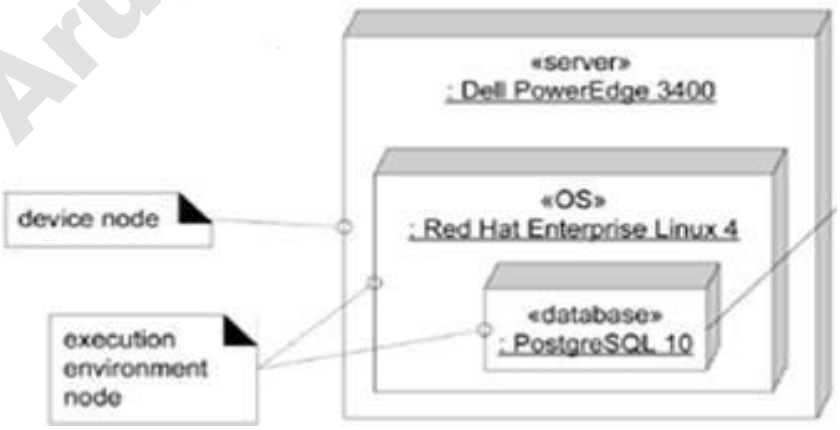
**Execution environment node** (EEN): This is a software computing resource that runs within an outer node (such as a computer) and which itself provides a service to host and execute other executable software elements. For example:

a. An operating system (OS) is software that hosts and executes programs
b. A virtual machine (VM, such as the Java or .NET VM) hosts and executes programs
c. A database engine (such as postgresql) receives SQL program requests and executes them, and hosts/executes internal stored procedures (written in Java or a proprietary language)
d. A Web browser hosts and executes javascript, Java applets, Flash, and other executable technologies
e. A workflow engine
f. A servlet container or EJB container

**Node:** A node may contain and show an artifact a concrete physical element, usually a file. This includes executables such as JARs, assemblies, .exe files, and scripts. It also includes data files such as XML, HTML, and so forth

**15. How** do you represent a node in a Deployment diagram? What kind of information can appear in a node**? (Nov/Dec 2013)**

There are 2 types of nodes: 1) Node 2) EEN

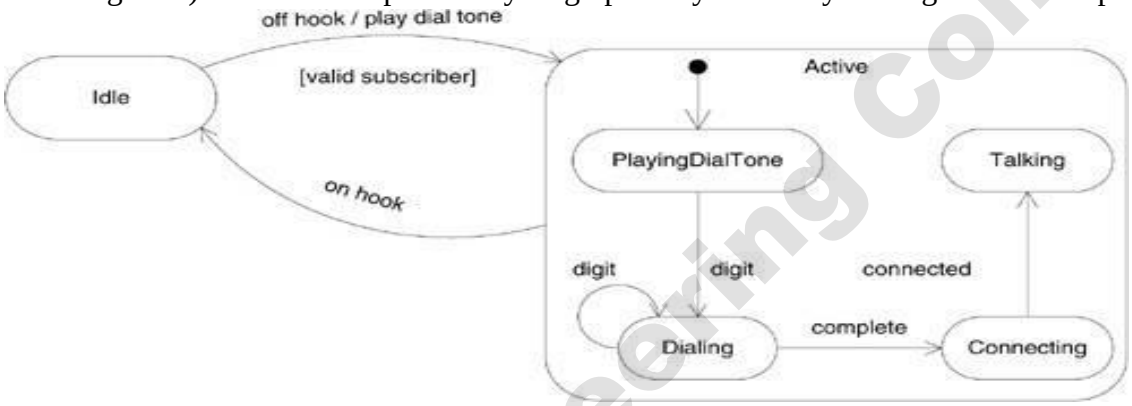


**Kind of information can appear in a node :**

a) An artifact a concrete physical element, usually a file
b) An operating system (os) is software that hosts and executes programs
c) A virtual machine (vm, such as the java or .net vm) hosts and executes programs
d) A database engine (such as postgresql) receives sql program requests and executes them, and hosts/executes internal stored procedures (written in java or a proprietary language)
e) A web browser hosts and executes javascript, java applets, flash, and other executable technologies
f) A workflow engine
g) A servlet container or EJB container

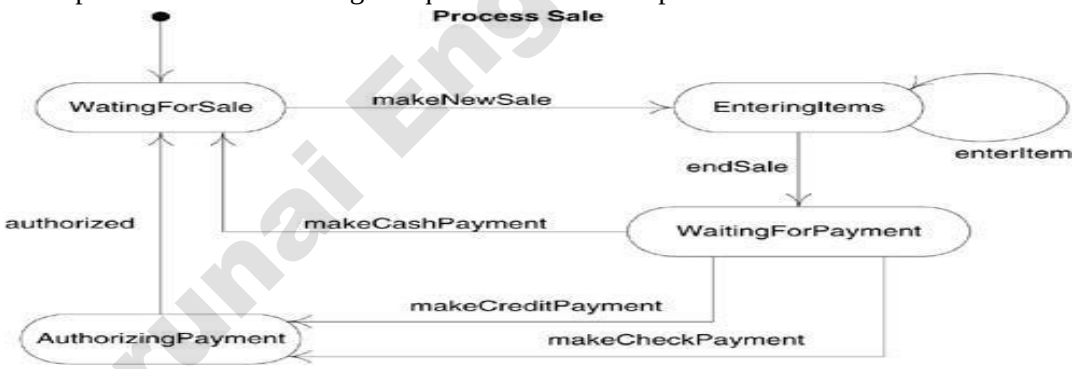
16. **What** is Nested States in state transition diagram**?**

A state allows nesting to contain substates; a substate inherits the transitions of its superstate (the enclosing state). See the example. It may be graphically shown by nesting them in a superstate box.




17. **Draw** state transition diagram for next gen POS system

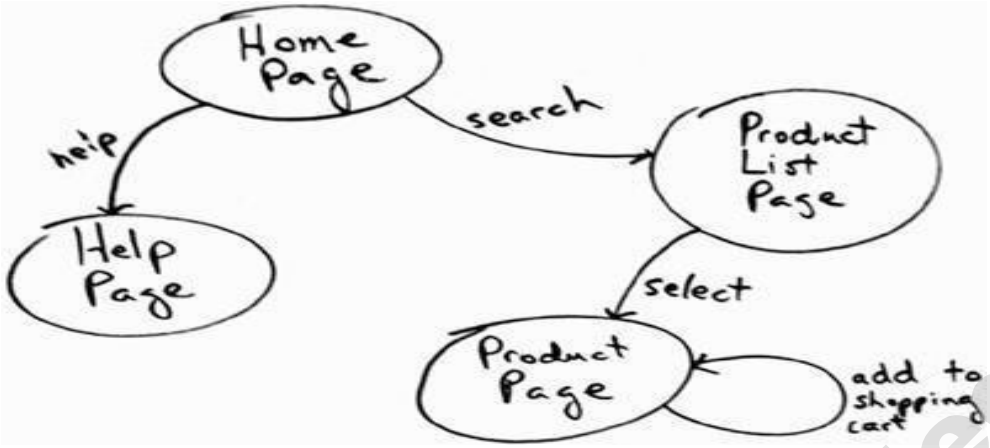
A sample state machine for legal sequence of use case operations.




18. **What** are the primary goals in the design of UML**?(Nov/Dec 2016)**

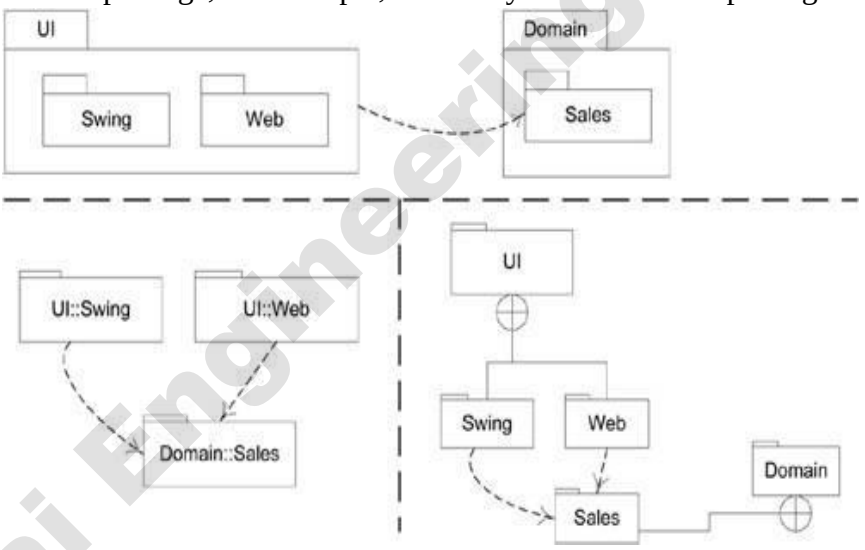
- Provide users a ready – to use expressive visual modeling language so they can develop and exchange meaningful models.
- Provide extensibility and specialization mechanism to extend the coreconcepts.
- Be independent of particular programming language and development process.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of the OO tools market.
- Support higher – level development concepts.
- Integrate best practices and methodologies.

19. **Draw** the state machine for Web page navigation modeling**.**




20. **Define** package and draw the UML notation for package **(May/June 2012, May/June 2013)**

- Logical Architecture is as part of the Design Model and also be summarized as a view in the Software Architecture Document. It defines the packages within which software classes are defined.
- A UML package can group anything: classes, other packages, use cases, and so on. A layer can be modeled as a UML package; for example, the UI layer modeled as a package named UI.




21. **What** do you mean by sequence diagram**?** Mention its use**.(Nov/Dec 2011,Apr/May 2015)**

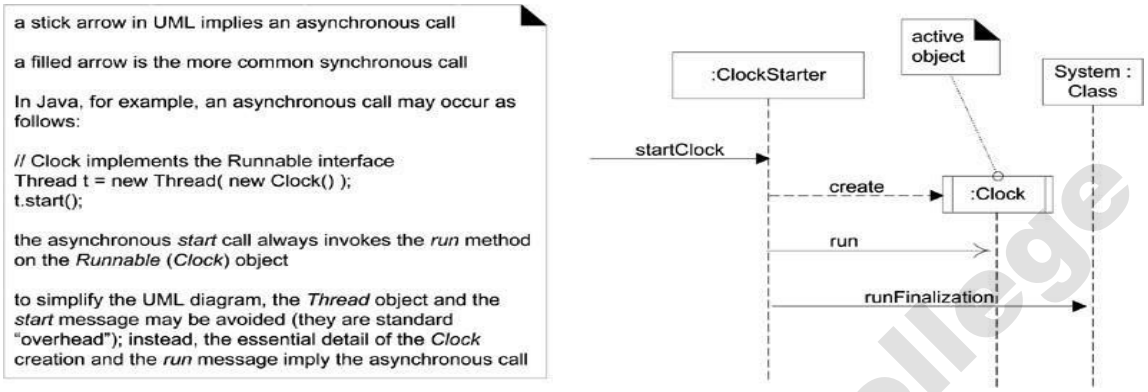
The UML includes interaction diagrams to illustrate how objects interact via messages. They are used for dynamic object modeling. There are two common types: sequence and communication interaction
diagrams.

**Uses of Sequence diagram:**

- Clearly shows sequence or time ordering of messages
- Large set of detailed notation options

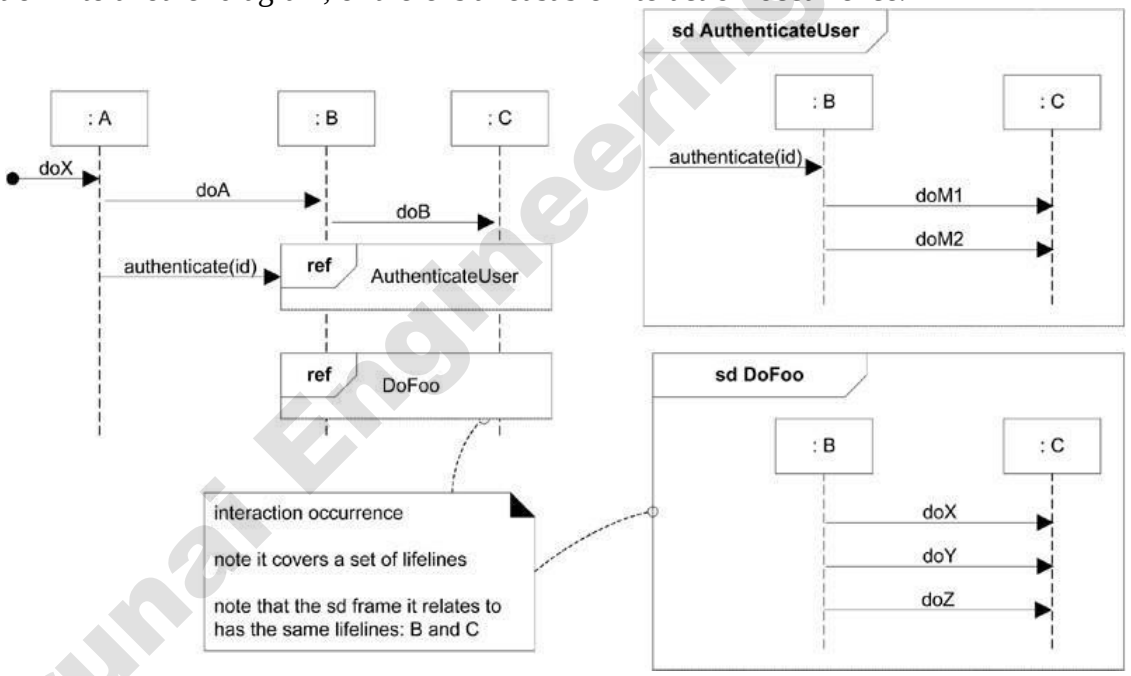
**22. What** is difference between synchronous and asynchronous message**?**

An asynchronous message call does not wait for a response; it doesn't block. They are used in multi-threaded environments such as .NET and Java so that new threads of execution can be created and initiated.




**23. How** to relate interaction diagram**?**

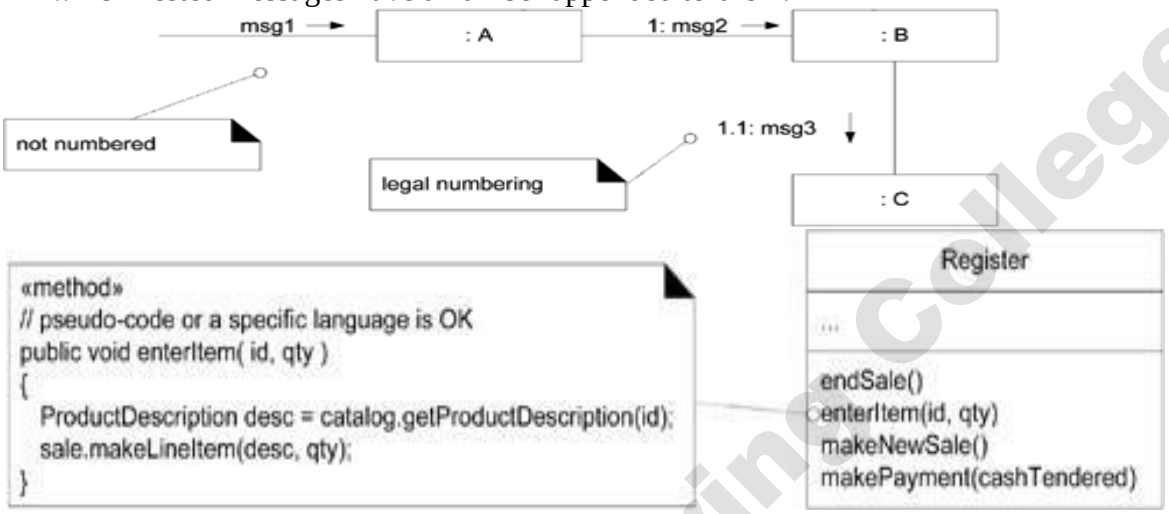
An interaction occurrence (also called an interaction use) is a reference to an interaction within another interaction. It is useful, for example, when you want to simplify a diagram and factor out a portion into another diagram, or there is a reusable interaction occurrence.



**24. What** do you mean by sequence number in UML? Where and for what it is used**? (Nov /Dec 2011, Nov/Dec 2012)**

The order of messages is illustrated with sequence numbers.

1. The first message is not numbered. Thus, msg1 is unnumbered.
2. The order and nesting of subsequent messages is shown with a legal numbering scheme in which nested messages have a number appended to them.



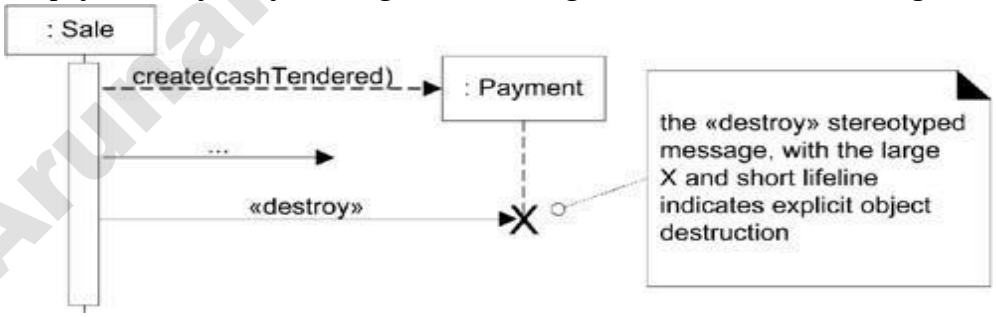

**27. What** are UML Properties and Property Strings**?**

In the UML, a property is "a named value denoting a characteristic of anelement

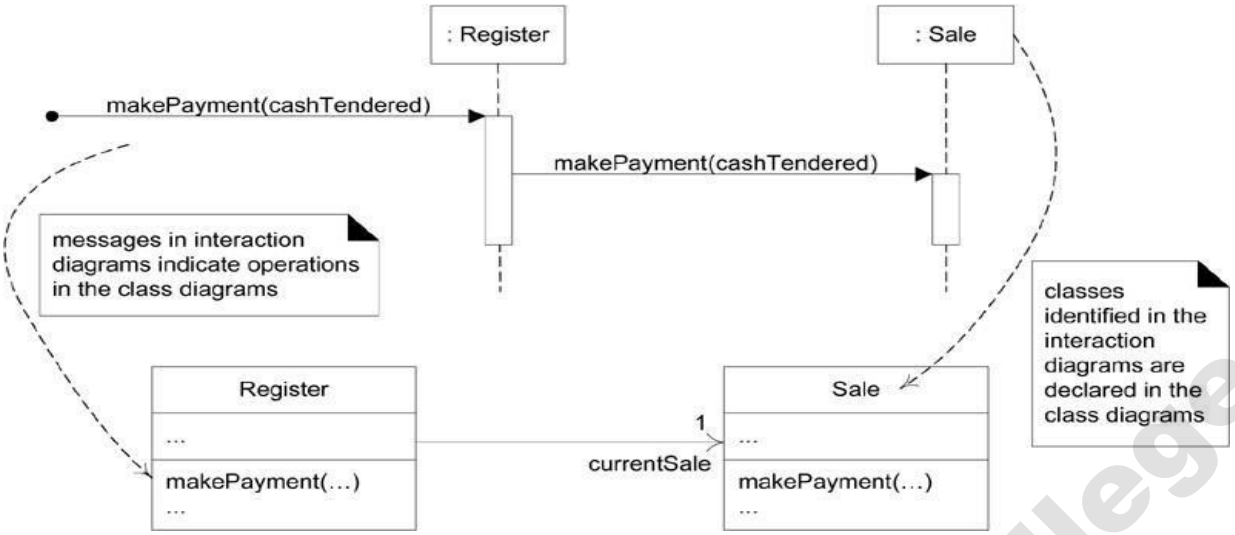
Properties of elements may be presented in many ways, but a textual approach is to use the UML property string {name1=value1, name2=value2} format, such as {abstract, visibility=public}. Some properties are shown without a value, such as {abstract}; this usually implies a boolean property, shorthand for {abstract=true}. Note that {abstract} is both an example of a constraint and a property string

**28. How** to Create Instance **? (Apr 2018)**

Instance creation is represented using create msg to newly created object. In the example sale object creates payment object by sending create message with cash tendered as argument.




**29.How** to use the creating methods from Interaction diagrams? **(Understand) (Apr 2018)** Message in interaction diagrams gives the operation name in the class diagrams. In the example, make Payment message to register class indicates the presence of make Payment () in register class.




**PART-B**

1. For the Hospital Management system **draw** the Activity Diagram (billing) (Nov/Dec 2012)
2. **Explain** about activity diagram with an example? (**April/May 2011, May/June 2012, may/June 2013, Nov/Dec 2013, Nov/Dec 2016, Nov/Dec 2019)**
3. **Discuss** about UML deployment and component diagram. Draw deployment diagram for a banking application.? **(April/May 2011, May/June 2014, Nov/Dec 2019)**
4. **Explain** the state chart diagram with a suitable example. Also define its components and use **(Nov /Dec 2011, Nov/Dec 2013, May/June 2014)**
5. **Consider** the Hospital Management System application with the following requirements **(Nov/Dec 2011, Nov/Dec 2012)**
   (i) System should handle the in-patient, out-patient information through receptionist.
   (ii) Doctors are allowed to view the patient history and give their prescription.
   (iii) There should be a information system to provide the required information.
   Give the state chart, component and deployment diagrams.
6. **Explain** the following with an example (**Nov/Dec 2011**)
   i) Activity diagram
7. **Draw** and **explain** the activity diagram for the online purchase system (8) **(Apr/May 2015)**
8. **What** is the purpose of Deployment diagram? **Explain** the basic elements of deployment diagram through an example (16) **(Apr/May 2015)**
9. **Apply** Interactive modeling for Payroll system in UML **(Nov/Dec 2016)**
10. **Design** the activity diagram for the following scenario. Booking a Ticket on the Indian Railways e-Ticket System.
11. **Design** and **explain** the activity diagram for the online purchase system

**UNIT-IV**
**DESIGN PATTERNS**

**PART- A**

**1. What** is design pattern**? (April/May 2011, May/June 2014, Nov/Dec 2016, NOV/DEC 2017) / When** to use patterns **(May/June 2012, May/June 2013, Nov/Dec 2015, Nov/Dec 2019) / Write** a note on Patterns **(Nov/Dec 2011, Apr/May 2015, and Nov/Dec 2016) / State** the use of Design Pattern **(Nov/Dec 2013)**

A pattern is a named description of a problem and solution that can be applied to new contexts; ideally, a pattern advises us on how to apply its solution in varying circumstances and considers the forces and trade-offs. Many patterns, given a specific category of problem, guide the assignment of responsibilities to objects. To solve the problems that arise during design phase we use pattern.

For example

| | |
|---|---|
| Pattern Name: | Information Expert |
| Problem: | What is a basic principle by which to assign responsibilities to objects? |
| Solution: | Assign a responsibility to the class that has the information needed to fulfill it. |

**2. Define** coupling**? (April/May 2011, Nov/Dec 2013, Apr 2018)**

Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. An element with low (or weak) coupling is not dependent on too many other elements;

**Types of coupling**

1. Low coupling or weak coupling
2. High coupling or strong coupling

**Low coupling:** An element if does not depend on too many other elements like classes, subsystems and systems it is having low coupling.

**High coupling:** A class with high coupling relies on many other classes.

**3. What** is Grasp **(May/June 2013)**

Grasp stands for "General Responsibility Assignment Software Patterns"

- It is a Learning Aid for OO Design with Responsibilities. This approach to understanding and using design principles is based on patterns of assigning responsibilities.
- We can apply the GRASP principles while drawing UML interaction diagrams, and also while coding where we deciding on responsibly assignments.
- **GRASP** defines **nine basic OO design principles or basic building blocks** in design. They are

1. Information Expert
2. Creator
3. Controller
4. Low Coupling
5. High Cohesion
6. Polymorphism
7. Pure Fabrication
8. Indirection
9. Protected Variations.

**4. What is Responsibility Driven Design?**

RDD is a general metaphor for thinking about objects oriented design. The responsibilities include Doing responsibility -> creation action, initializing, controlling and coordinating activities.

**5. What are the two responsibilities?**

The responsibilities are of the following two types: doing and knowing.

Doing responsibilities of an object include:

- Doing something itself, such as creating an object or doing a calculation
- Initiating action in other objects
- Controlling and coordinating activities in other objects

Knowing responsibilities of an object include:

- Knowing about private encapsulated data
- Knowing about related objects
- Knowing about things it can derive or calculate

**6. List** the Grasp Pattern**.**

- Information Expert
- Creator
- High Cohesion
- Low Coupling
- Controller

**7. Mention** the list of behavioral Patterns used during design phase of software development **(Apr 2018)**

Interpreter, Template Method, Command, Iterator, Mediator, Memento, Flyweight, Observer, State, Strategy, Visitor.

**8. Define** Object with suitable example **(Nov/Dec 2012)**

Object is instance of class. In the NextGen POS application, who should be responsible for creating a SalesLineItem instance. Here "Sale "takes the responsibility of creating 'SalesLineItem' instance. Since sale contains many 'SalesLineItem' objects.

**9. Define Creator.**

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.

Who creates? (Note that Factory is a common alternate solution.)

Assign class B the responsibility to create an instance of class A if one of these is true:

1. B contains A
2. B aggregates A
3. B has the initializing data for A
4. B records A
5. B closely uses A

**10. Define** Controller**.**

The Controller pattern assigns the responsibility of dealing with system events to a non-UI class that represents the overall system or a use case scenario. A Controller object is a non-user interface object responsible for receiving or handling a system event.

11. **What** first object beyond the UI layer receives and coordinates ("controls") a system operation?

Assign the responsibility to an object representing one of these choices:

1. Represents the overall "system," a "root object," a device that the software is running within, or a major subsystem (these are all variations of a façade controller).
2. Represents a use case scenario within which the system operation occurs ( a use-case or session controller)

12. **Define** Low Coupling**. ( May/june 2014)**

Low Coupling is an evaluative pattern, which dictates how to assign responsibilities to support:
- Low dependency between classes;
- Low impact in a class of changes in other classes;
- High reuse potential;

Problem: How to reduce the impact of change?

Solution: Assign responsibilities so that (unnecessary) coupling remains low. Use this principle to evaluate alternatives.

13. **Define** High Cohesion**. (Nov/Dec 2012)**

High Cohesion is an evaluative pattern that attempts to keep objects appropriately focused, manageable and understandable. High cohesion is generally used in support of Low Coupling. High cohesion means that the responsibilities of a given element are strongly related and highly focused. Breaking programs into classes and subsystems is an example of activities that increase the cohesive properties of a system.

Problem : How to keep objects focused, understandable, and manageable, and as a side-effect, support low Coupling?

Solution: Assign responsibilities so that cohesion remains high. Use this to evaluate alternatives.

14. **Distinguish** between coupling and cohesion **(Nov /Dec 2011,Nov/Dec 2015,Nov/Dec 2016,Apr/May 2017,NOV/DEC 2017,Nov/Dec 2019)**

**Cohesion:** The responsibilities of a given element are strongly related and highly focused.

Breaking programs into classes and subsystems is an example of activities that increase the cohesive properties of a system. A good design principle follows High cohesion and low coupling **Coupling:** Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. An element with low (or weak) coupling is not dependent on too many other elements;

**Types of coupling**

1. Low coupling or weak coupling
2. High coupling or strong coupling

15. **Define** Information Expert**.**

Information Expert is a principle used to determine where to delegate responsibilities. These responsibilities include methods, computed fields and so on. Using the principle of Information Expert, a general approach to assigning responsibilities is to look at a given responsibility, determine the information needed to fulfill it, and then determine where that information is stored. Information Expert will lead to placing the responsibility on the class with the most information required to fulfill it.

A general principle of object design and responsibility assignment:

Assign a responsibility to the information expert – the class that has the information necessary to fulfill the responsibility.

16. **What** is adapter pattern**?**

| | |
|---|---|
| Name: | Adapter |
| Problem: | How to resolve incompatible interfaces, or provide a stable interface to similar components with different interfaces? |
| Solution: (advice) | Convert the original interface of a component into another interface, through an intermediate adapter object. |

17. **List** the Advantages of factory

Factory objects have several advantages:

- Separate the responsibility of complex creation into cohesive helper objects.
- Hide potentially complex creation logic.
- Allow introduction of performance-enhancing memory management strategies, such as object caching or recycling.

**18. What** is factory pattern**?**

| | |
|---|---|
| Name: | Factory |
| Problem: | Who should be responsible for creating objects when there are special considerations, such as complex creation logic, a desire to separate the creation responsibilities for better cohesion, and so forth? |
| Solution: (advice) | Create a Pure Fabrication object called a Factory that handles the creation. |

**19. What** is Observer pattern**?**

| | |
|---|---|
| Name: | Observer (Publish-Subscribe) |
| Problem: | Different kinds of subscriber objects are interested in the state changes or events of a publisher object, and want to react in their own unique way when the publisher generates an event. Moreover, the publisher wants to maintain low coupling to the subscribers. What to do? |
| Solution: (advice) | Define a "subscriber" or "listener" interface. Subscribers implement this interface. The publisher can dynamically register subscribers who are interested in an event and notify them when an event occurs. |

**20. What** is façade Controller?

The Facade controller representing the overall system, device, or a subsystem. facade controller representing the overall system, device, or a subsystem.

**Façade controllers are used**

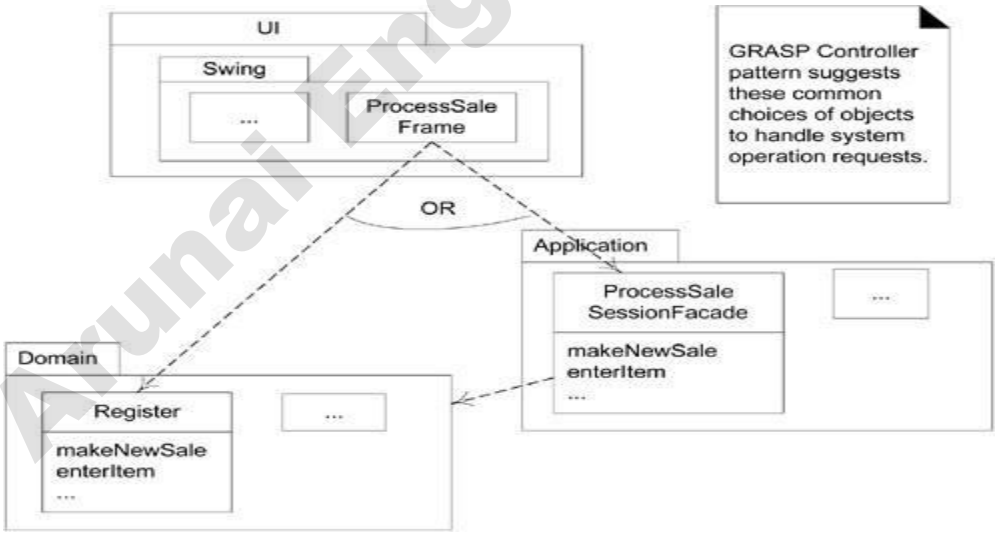
1) Facade controllers are suitable when there are not "too many" system events,
2) When the user interface (UI) cannot redirect system event messages to alternating controllers, such as in a message-processing system.

**21. What** is use case Controller?

A use case controller is a good choice when there are many system events across different processes; it factors their handling into manageable separate classes and also provides a basis for knowing and reasoning about the state of the current scenario in progress.

**22. What** is controller**?**

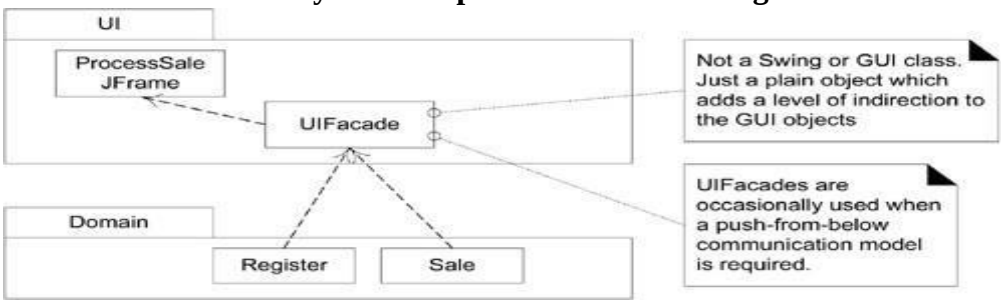
A controller is the first object beyond the UI layer that is responsible for receiving or handling a system operation message. Here ProcessSaleFrame is the controller.



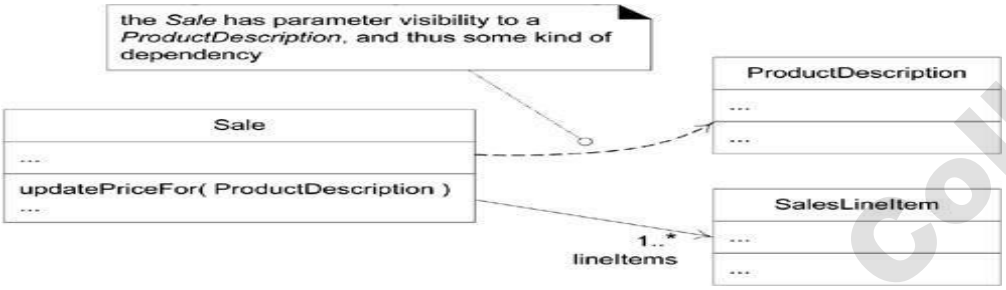

**23. What** is "Upward" Communication?

It is sufficient to send messages to domain objects, querying for information which they then display in widgets a polling or pull-from-above model of display updates.

**A UI layer UI Facade is occasionally used for push-from-below designs.**




**24. How** will you represent dependency relationship**?**




**25. Define** Singleton Classes

| | |
|---|---|
| Name: | Singleton |
| Problem: | Exactly one instance of a class is allowedit is a "singleton." Objects need a global and single point of access. |
| Solution: (advice) | Define a static method of the class that returns the singleton. |

A class can be marked with a '1' in the upper right corner of the name compartment. it means that there is only one instance of a class instantiated never two.

26. **List** the GOF pattern
Adapter pattern
Singleton Pattern
Factory pattern
Observer Pattern

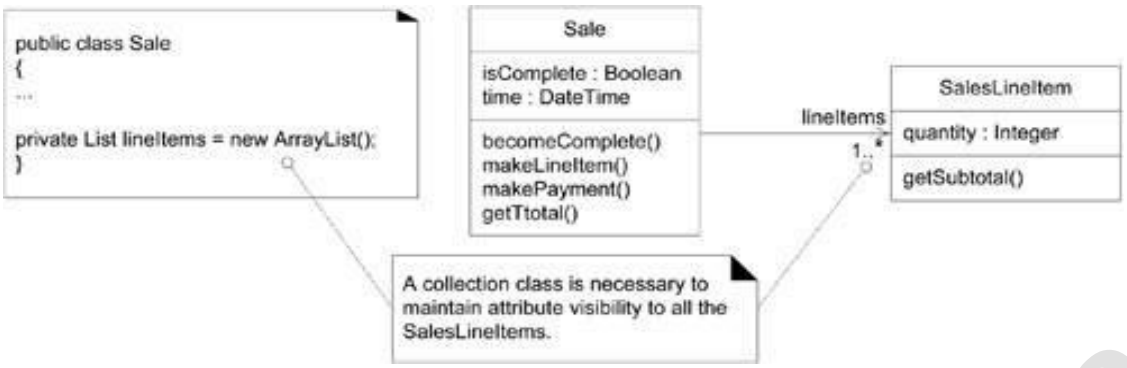
**27. What** is Facade Pattern?

A facade is an object that provides a simplified interface to a larger body of code, such as a class library. A facade can:

- make a software library easier to use, understand and test, since the facade has convenient methods for common tasks;
- make code that uses the library more readable, for the same reason;
- reduce dependencies of outside code on the inner workings of a library, since most code uses the facade, thus allowing more flexibility in developing the system;
- Wrap a poorly-designed collection of APIs with a single well-designed API (as per task needs).

**28. How** will you represent Collection classes in Code?

In OO programming languages One-to-many relationships are usually implemented with the introduction of a collection object, such as a List or Map, or even a simple array.




## Part-B

1. **Explain** about GRASP Patterns**? (April/May 2011, May/June 2014, Apr/May 2017, nov/Dec 2019)**
2. **Write** short notes on adapter, singleton, factory and observer patterns? (**April/May 2011, Apr/May 2015**)
3. **What** is GRASP? Explain the design patterns and the principles used in it **(Nov/Dec 2011, Nov/Dec 2012, Nov/Dec 2016, NOV/DEC 2017)**
4. i) **Describe** the concept of creator (7) (**May/June 2012**, **May/June 2013, Apr/May 2015**)
   ii) **Explain** about low coupling, controller, high cohesion (3x3) **(Apr/May 2015)**
5. **Write** short notes on adapter, singleton, factory and observer pattern (**May/June 2012, May/June 2013, Nov/Dec 2013, May/June 2014)**
6. i) **Compare** Cohesion and Coupling with Suitable example (8) **(Nov/Dec 2015)**
   ii) **State** Roles and Patterns while Developing systems (8) **(Nov/Dec 2015)**
   i) **Differentiate** Bridge and Adapter. (8) **(Nov/Dec 2015)**
   ii) **How** will you Design the Behavioral Pattern. (8) **(Nov/Dec 2015)**
7. **Explain** Creator and Controller Design pattern with example (Nov/Dec 2016)
8. **Explain** about Factory pattern and mention the limitations and applications of Factory pattern (**Apr/May 2017, NOV/DEC 2017)**.
9. **What** is design pattern? Explain the GoF design patterns **(Nov/Dec 2011, Nov/Dec 2012, NOV/DEC 2017, Nov/Dec 2019)**
10. **Explain** in detail about GRASP pattern and also explain in designing objects with responsibilities. (13) **(APR /MAY 2018)**
11. **write** short notes on adapter pattern and observer pattern. (7) **(APR /MAY 2018)**
12. **compare** between different categories of design patterns. (6) **(APR/MAY 2018)**
13. **Describe** the features of Low Coupling and High Coupling with suitable. (13) **(NOV /DEC 2018)**
14. **How** will you generate source code from design using UML? Illustrate. (13) **(NOV /DEC 2018)**
15. **Write** about implementation model (mapping design to code)?(**April/May2011,Nov/Dec 2015,Nov/Dec 2016)**

## UNIT V
## TESTING

### PART –A

1. **What** are method, methodology and process**?**
   A method is an implementation of an object's behavior. A model is an abstract of a system constructed to understand the system prior to building or modifying it.
   Methodology is going to be a set of methods, models and rules for developing systems based on any set of standards. The process is defined as any operation being performed.

2. **Write** the difference between a method and a process.
   Method is going to be implanted version of an objects behavior whereas the process is any operation being performed. Methods concentrate on the data or the object invoking it and modify their behavior. Process is any operation that needs to be carried out in the system.

3. **What** are the phases of OMT (Object modeling technique)?
   The different phases of OMT are:
   a) Analysis: This results in the object and dynamic and functional models. The object model describes the structure of objects in a system and is represented by means of an object diagram. The dynamic model is going to be a detailed state transition diagram. The diagram is going to be a set of states receiving events so as to make transitions. The functional model is a representation of flow of data between different processes in a business. The process is any function being performed, data flow shows the direction of data element movement, data store is location of the data storage; an external entity is a source or destination of data element.
   b) System Design: The results are a structure of the basic architecture of the system along with high-level strategy decisions.
   c) Object Design: The phase produces a detailed design document of all the models.
   d) Implementation: This phase produces a comprehensive code for the problem.

4. **Name** five Booch diagrams**.**
   Five Booch diagrams are Class diagrams, Object diagrams, State transition diagrams, Module diagrams, and Process diagrams.

5. **Briefly** list the Booch system development process**.**
   It helps us design the system using the object paradigm. It covers the analysis and design phases of a system. It includes a macro and a micro development process.

6. **List** the activities of Macro development process
   Macro development process is concerned with the technical management of the system. It includes
   a) Conceptualization where the core requirements of the system are outlined
   b) Analysis and the development model which focuses on the class diagrams,
   c) Design or creation of the computer architecture to establish relationships between the' classes
   d) Evolution or implementation to produce a code and
   e) Maintenance to add new requirements and to eliminate the bugs.

7. **List the activities of Micro development process**
   Each macro development process has its own micro development process which aims at
   a) Identifying class and objects
   b) Identifying class and object semantics.
   c) Identifying class and object relationships

d) Identifying class and object interfaces and implementation

8. **What** is a use case? What are some of the ways that use cases can be described?
   Use Case is a scenario depicting a user system interaction. It begins with the user of the system issuing a sequence of interrelated events. Use cases are described as:
   a) Non formal text with no clear flow of events.
   b) Text, easy to read but with a clear flow of events.
   c) Formal style using pseudo code.

9. **What** is the strength of Jacobson et. Al. Methodology?
   The strength of the Jacobson et. Al. methodology is that it entire life cycle and stress trace ability between the different phases, both forward and backward. This enables the reuse of analysis and design work, reducing development time significantly.

10. **What** do you mean by difference between patterns and frameworks**?**
    A pattern is instructive information that captures the essential structure and insight of a successfully family of proven solutions to a recurring problem that arises within certain context and system of forces. Pattern solves a problem, is a proven concept, describes relationships, and has significant human component.
    A framework is a way of presenting a generic solution to a problem that can be applied to all levels in a development. It represents a set of classes that make up a reusable design for a specific class of software. It partitions the design into abstract classes and also defines relationships between them. They emphasize design reuse over code reuse.

11. **What** are anti-patterns?
    A pattern represents a "best practice," whereas an antipattern represents "worst practice" or a "lesson learned."

12. **What** is pattern mining?
    A pattern should help its users comprehend existing systems, customize systems to fit user needs, and construct new system. The process of looking for patterns to document is called pattern mining.

- **Outline** the processes involved in UA to software development?
- Use-case driven development
- Object-oriented analysis
- Object-oriented design
- Incremental development and prototyping
- Continuous testing

13. **How** are models represented and organized?

**Static model:** A static model can be viewed as a snapshot of a system's parameters at rest or at a specific point in time. Static models are needed to represent the structural or static aspect of a system. For example, a customer could have more than one account or an order could be aggregated from one or more line items.

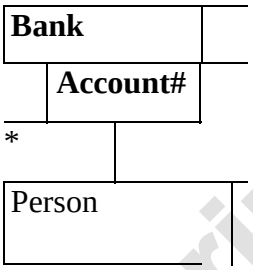
**Dynamic model:** A dynamic model contrast to a static model, can be viewed as a collection of procedures or behaviours that, taken together, reflect the behaviour of a system over time.
For example, an order interacts with inventory to determine product
availability. Dynamic relationships show how the business objects interact to perform tasks.

14. **Illustrate** the graphical diagrams defined by UML**.**

1 Class diagram (static)
2 Use-case diagram
3 Behavior diagram (dynamic)
Interaction diagram
Sequence diagram
Collaboration diagram
State chart diagram
4 Implementation diagram
Component diagram
Deployment diagram

**15. What** is a Qualifier?

A qualifier is an association attribute. For example, a person object may be associated to a Bank object. An attribute of this association is the account#. The account# is the qualifier of this association.




A qualifier is shown as a small rectangle attached to the end of an association path, between the final path segment and the symbol of the class to which it connects. The qualifier rectangle is part of the association path, not part of the class. The qualifier rectangle usually is smaller than the attached class rectangle.

**16. What** is meant by Multiplicity?

Multiplicity specifies the range of allowable associated classes. It is given for roles within associations, parts within compositions, repetitions, and other purpose. A multiplicity specification is as a text string comprising a period-separated sequence of integer intervals, where an interval represents a range of integers in this format: " lower bound.. upper bound". The terms lower bound and upper bound are integer values. The star character (*) may be used for the upper bound, denoting an unlimited upper bound. If a single integer value is specified, then the integer range
contains the single values.

Eg:0..1
0..*
1. . 3,7. . 10,15,19..*

**17. What** is Association class.?

An association class is an association that also as class properties. The name in the class symbol and the name string attached to the association path are the same. If an association class has attributes but no operations or other associations, then the name may be displayed on the association path and omitted from the association class to emphasize its “association nature". If it has operations and attributes then the name may be omitted from the path and placed in the class rectangle to emphasize its "class nature".

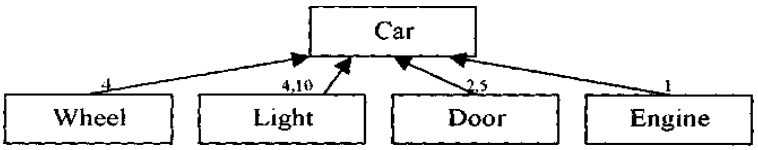
**18. What** are the contents of UML behaviour diagrams**?**

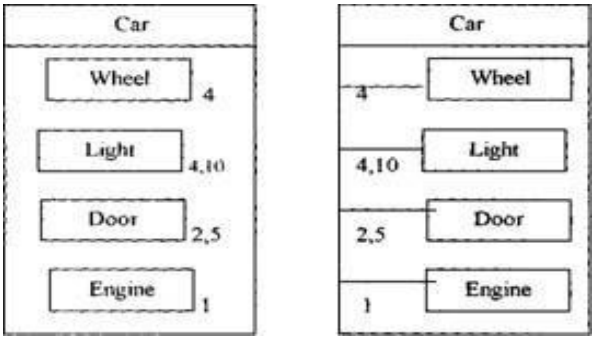
- Interaction diagrams: a) Sequence and b) collaboration diagrams
- State chart diagrams

- Activity diagrams

**19. What** is an Association rule?

An association may have an association name. This name may have an optional black triangle in it, the point of the triangle indication the direction in which to read the name. The end of an association where it connects to a class is called the association role.






**20. What** is a meta-model?

It is a model of modeling elements. This assures consistency among diagrams. This has made possible for a team to explore ways to make the modeling language much simpler by in a sense unifying the elements of the unified modeling language.

**21. What** do you mean by usability Testing**?**

It is the effectiveness, efficiency and satisfaction with which a specified set of users can achieve a specified set of tasks in particular environments. The ISO definition requires

- Defining tasks. What are the tasks?
- Defining users. Who are the users?

**22. Infer the guidelines for developing usability testing.**

- The usability testing should include all of the s/w components.
- Usability testing need not be very expensive or elaborate, such as including trained specialists working in a sound proof lab with one-way mirrors and sophisticated recording equipment.
- Similarly, all tests need not involve many subjects.
- Consider the user's experience as part of your s/w usability. You can study 80-90% of most design problems with as few as 3 or 4 users if you target only a single skill level of users, such as novices or intermediate level users.
- Apply usability testing early and often.

**23. What** are the designing measurable goals assisted for usability?

Usability can be assisted by designing measurable goals, such as

- 95% users should be able to find how to with draw money from the ATM machine with out error and with no formal training.
- 70% of all users should experience the new function as "a clear improvement over the previous one.
- 90% of consumers should be able to operate the VCR within 30 minutes.

**24. What** are the objectives of the user satisfaction test?

- As a communication vehicle between designers, as well as between users and designers.
- To detect and evaluate changes during the design process.
- To provide periodic indications of divergence of opinion about the current design.
- To enable pin pointing specific areas of dissatisfaction for remedy.
- To provide a clear understanding of just how the completed design is to be evaluated.

**25. Why** quality assurance is needed?

It is because computers are information for doing what you tell to do, not necessarily what you want them to do. To close this gap, the code must be free from errors, that because unexpected results a process called "Debugging".

**26. What** are the kinds of errors you might encounter when you run your program?

1. Language Errors: It results from incorrectly constructed code, such as an incorrectly typed keyword or some necessary punctuation omitted. These are easiest types of errors.
2. Run time errors: They occur and are detected as the program is running, when a statement attempts an operation that is impossible to carry out.
3. Logic errors: When codes do not perform the way you intended. The code might be syntactically valid and run without performing any invalid operations and yet produce incorrect results.

**27. Define** Black box testing

The concept of black box testing is used to represent a system whose inside workings are not available for inspection. In a black box testing, the test item is treated as "black" since its logic- is unknown; all that is known is what goes in and what conies out, or the input and output.

In a black box testing, you try various inputs and examine the resulting output; you call earn what the box does but nothing about how this conversion in implemented. Black box testing works very nicely in testing objects in an object-oriented environment. The black box testing works very nicely in testing objects in an object-oriented environment.

**28. What** is white box testing?

White box testing: This assumes that the specific logic is important and must be tested guarantee the system's proper functioning of the white box is in error-based testing, when you already have tested all objects of an application and all external or public methods of an object that you believe to be greater importance.

**29. Explain** top down testing.

Top Down Testing: It assumes that the main logic or object interactions and systems messages of the application need testing than an individual objects methods or supporting logic. A top down strategy can detect the serious flaws early in the implementation. Testing the user interface using a top down approach means testing interface navigation. Thus serves two purposes, according to Conger.

**30. What** is Bottom-up Testing?

This starts with the details of the system and proceeds to higher levels by a progressive aggregation of details until they collectively fit the requirements for the system. In this approach, you start with the methods and classes that call or rely on no methods and classes that use only the bottom level ones already tested.

**31. What** are the types of path testing?

a) Statement Testing Coverage:

The main idea of statement testing coverage is to test every statement in the objects method by executing it at least once. Murray states, "Testing less than this for new software is unconscionable and should be criminalized".

b) Branch Testing Coverage:

The main idea behind branch testing is to perform enough tests to ensure that every In branch testing coverage is to perform enough tests to ensure that every branch alternative has been executed at least once under some test.

**32. Summarize** the impact of an object orientation on testing.

- Some types of errors could become less plausible .
- Some types of errors could become more plausible
- Some new types of errors might appear

**33. List** the objective of testing**.**

- Testing is the process of executing a program with the intent of finding errors.
- A good test cases is the one that has s high probability of detecting an as-yet undiscovered error
- A successful test case is the one that detects as -yet undiscovered error.

**34. List** the guidelines for developing quality assurance test cases.

Freedman and Thomas have developed guidelines that have been adopted for the UA:

- Describe which feature or service your test attempts to cover.
- If the test case is based on a use case, it is good idea to refer to the use-case name.
- Specify what you are testing and which particular feature.
- test the normal use of the object methods.
- test the abnormal but reasonable use of the objects methods.
- test the boundary conditions.
- Test objects interactions and the messages sent among them.
- Attempting to reach agreement on answers generally will raise other what-if questions.
- The internal quality of the software, such as its reusability and extensibility, should be assessed as well.

**35. What** do you understand from test plan?

A test plan is developed to detect and identify potential problems before delivering the software to its users. The test plan need not be very large; in fact, devoting too much time to the plan can be counterproductive.

**36. What** are the steps needed to create a test plan?

1. Objectives of the test: create the objectives and describes how to achieve them.
2. Development of a text case: develop test case, both input and expected output.
3. Test analysis: This step involves the examination of the test output and the documentations of the test results.

**38. What** is Regression testing?

All passed tests should be repeated with the revised program, called "Regression". This can discover errors introduced during the debugging process. When sufficient testing is believed to have been conducted, this fact should be reported, and testing to this specific product is complete.

**39. Define** Error based testing

Error based testing techniques search a given class's method for particular clues of interests, and then describe how these clues should be tested..

**40. What** is Iterative Development and Continuous Testing?

The UA encourages the integration of testing plans from day 1 of the project. Usage scenarios or Use Cases can become test scenarios; therefore, use cases will drive the usability testing. It must be iterated and reiterate until, satisfied with the system.

**41. Classify** the types of 3 Layered Approach to Software Development**.**

- Business layer,
- User interface layer and
- The access layer

**42. Define** software quality Assurance **(NOV/DEC 2019)**

*Software Quality Assurance* (SQA) is simply a way to assure quality in the *software*. Software Quality Assurance is a process which works parallel to development of a software. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue. Software Quality Assurance is a kind of an Umbrella activity that is applied throughout the software process.

**43. What** is unit testing? **(NOV/DEC 2019)**

Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent module are tested to determine if there are any issue by the developer himself. It is correlated with functional correctness of the independent modules.

## PART B

1. **Discuss** the analysis and the methodology by Booch, Rambaugh compared to booch briefly. In which aspect Booch analysis is successful. (13)
2. **Define** patterns. How do analysis patterns differ from design patterns? Discuss generative, non-generative patterns and anti-patterns and list the guidelines on capturing patterns? (13)
3. **Elaborate** are the major differences between a framework and a pattern? (13)
4. **Analyze** the unified methods for the development of the software? (13)
5. **Discuss** the various design methods and explain in detail the layered method to software development. (13)
6. **Discuss** the activities involved in Micro and Macro Development process. (8)
7. **Describe** the use of Sequence and collaboration diagrams with the help of example programs.
8. **Explain** component diagram and deployment diagram with examples (6)
9. **Elaborate** OMT functional model (7)
10. **What** are the primary goals of UML and explain the various class diagrams? (10)
11. **What** are the various testing strategies and discuss the impact of object orientation on testing? (7)
9. **Discuss** the issues of software complexity and discuss on test tool? (7)
10. **Write** short notes on the following: (Analyze)
    - i) Guideline for developing a user satisfaction test.
    - ii) White box testing
    - iii) Black box testing
    - iv) iv)Debugging
11. **Discuss** inheritance testing with an example. (7)

12. **What** is a test plan? Describe the contents and characteristics of a test plan. (8)
13. **Describe** the different testing strategies. How to develop test plans guided by Thomas? (7)
14. **Discuss** the guidelines for developing quality assurance test cases described by Freedman and Thomas adapted for the UA. What are the steps involved to make the testing successful? (13)
15. i)**Explain** standards for testing any particularly Quality Assurance (QA)?
    ii) **Explain** the guidelines adapted for developing QA test cases. (7)
16. . **Give** a brief note on issue in Object Oriented Testing. (13) (**NOV /DEC 2018)**
17. **Outline** the object oriented testing strategies. **(NOV /DEC 2019)**
18. **What** is a test case? Describe in detail the test case design for OO software. **(NOV /DEC 2019)**