JEPPIAAR ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CS6502

OBJECT ORIENTED ANALYSIS AND DESIGN

QUESTION BANK

III YEAR A & B / BATCH: 2015 -2019

VISION OF INSTITUTION

To build Jeppiaar Engineering College as an Institution of Academic Excellence in Technical education and Management education and to become a World Class University.

MISSION OF INSTITUTION

M1	To excel in teaching and learning, research and innovation by promoting the principles of scientific analysis and creative thinking
M2	To participate in the production, development and dissemination of knowledge and interact with national and international communities
M3	To equip students with values, ethics and life skills needed to enrich their lives and enable them to meaningfully contribute to the progress of society
M4	To prepare students for higher studies and lifelong learning, enrich them with the practical and entrepreneurial skills necessary to excel as future professionals and contribute to Nation’s economy

PROGRAM OUTCOMES (POs)

PO1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of computer science
PO2	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of
PO3	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental
PO4	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of

PO5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities
PO6	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant
PO7	Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and
PO8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and
PO11	Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Vision of Department:

To emerge as a globally prominent department, developing ethical computer professionals, innovators and entrepreneurs with academic excellence through quality education and research.

Mission of Department

M1	To create computer professionals with an ability to identify and formulate the engineering problems and also to provide innovative solutions through effective teaching learning process.
M2	To strengthen the core-competence in computer science and engineering and to create an ability to interact effectively with industries.
M3	To produce engineers with good professional skills, ethical values and life skills for the betterment of the society.
M4	To encourage students towards continuous and higher level learning on technological advancements and provide a platform for employment and self-employment.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO 01: To address the real time complex engineering problems using innovative approach with strong core computing skills.

PEO 02: To apply core-analytical knowledge and appropriate techniques and provide solutions to real time challenges of national and global society.

PEO 03: Apply ethical knowledge for professional excellence and leadership for the betterment of the society.

PEO 04: Develop life-long learning skills needed for better employment and entrepreneurship.

PROGRAMME SPECIFIC OUTCOME (PSOs)

PSO1 – An ability to understand the core concepts of computer science and engineering and to enrich problem solving skills to analyze, design and implement software and hardware based systems of varying complexity.

PSO2 - To interpret real-time problems with analytical skills and to arrive at cost effective and optimal solution using advanced tools and techniques.

PSO3 - An understanding of social awareness and professional ethics with practical proficiency in the broad area of programming concepts by lifelong learning to inculcate employment and entrepreneurship skills.

JEPPIAAR ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS6502-OBJECT ORIENTED ANALYSIS AND DESIGN

III YEAR - 2018-2019

V SEM

UNIT I UML DIAGRAMS

Introduction to OOAD – Unified Process - UML diagrams – Use Case – Class Diagrams– Interaction Diagrams – State Diagrams – Activity Diagrams – Package, component and Deployment Diagrams.

UNIT II DESIGN PATTERNS

GRASP: Designing objects with responsibilities – Creator – Information expert – Low Coupling – High Cohesion – Controller - Design Patterns – creational - factory method - structural – Bridge – Adapter - behavioral – Strategy – observer.

UNIT III CASE STUDY

Case study – the Next Gen POS system, Inception -Use case Modeling - Relating Use cases –

include, extend and generalization - Elaboration - Domain Models - Finding conceptual classes and description classes – Associations – Attributes – Domain model refinement – Finding conceptual class Hierarchies - Aggregation and Composition.

UNIT IV APPLYING DESIGN PATTERNS

System sequence diagrams - Relationship between sequence diagrams and use cases Logical

architecture and UML package diagram – Logical architecture refinement - UML class diagrams – UML interaction diagrams - Applying GoF design patterns.

UNIT V CODING AND TESTING

Mapping design to code – Testing: Issues in OO Testing – Class Testing – OO Integration Testing – GUI Testing – OO System Testing.

TEXT BOOK:

1. Craig Larman, "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development", Third Edition, Pearson Education, 2005.

REFERENCES:

1. Simon Bennett, Steve Mc Robb and Ray Farmer, "Object Oriented Systems Analysis and Design Using UML", Fourth Edition, Mc-Graw Hill Education, 2010.

2. Erich Gamma, a n d Richard Helm, Ralph Johnson, John Vlissides, **"**Design patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

3. Martin Fowler, "UML Distilled: A Brief Guide to the Standard Object Modeling Language", Third edition, Addison Wesley, 2003.

4. Paul C. Jorgensen, "Software Testing:- A Craftsman's Approach", Third Edition, Auerbach Publications, Taylor and Francis Group, 2008.

COURSE OUTCOMES :

C303.1	**Understand** the various UML Diagrams
C303.2	**Analyse** various UML design Patterns
C303.3	**Create** inception elaboration and domain models
C303.4	**Apply** UML design Patterns
C303.5	**Create** Code and compare various testing techniques

INDEX

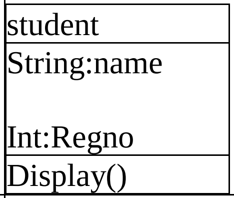
Unit-I

UML Diagrams

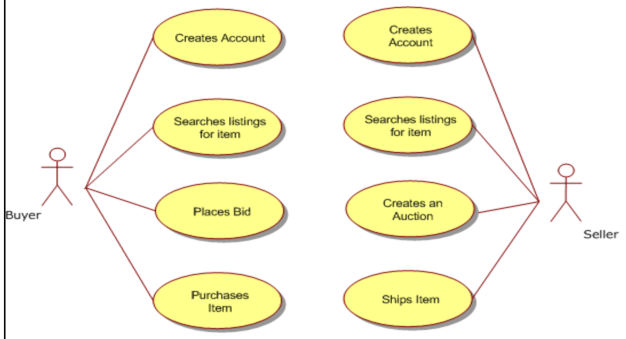
Introduction to OOAD – Unified Process - UML diagrams – Use Case – Class Diagrams– Interaction Diagrams – State Diagrams – Activity Diagrams – Package, component and Deployment Diagram

PART-A

S.NO	QUESTIONS	CO	BLOOM'S LEVEL
1	**What is Object-Oriented Analysis and Design? [APR/MAY 2011,MAY/JUNE 2013, NOV/DEC 2013, MAY/JUNE 2014, APR/MAY 2017]** During **object-oriented analysis**, there is an emphasis on finding and describing the objects or concepts in the problem domain. Ex: Flight information system, some of the concepts include Plane, Flight, and Pilot. During **object-oriented design**, (or simply, object design) there is an emphasis on defining software objects and how they collaborate to fulfill the requirements. The combination of these two concepts shortly known as object oriented analysis and design. Ex: A plane software object may have a **tailNumber attribute** and a **getFlight History method.**	C303.1	BTL1
2	**What is UML?[MAY/JUNE 2012,**	C303.1	BTL1

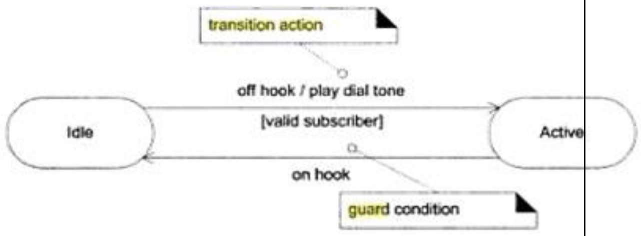
	MAY/JUNE 2013, MAY/JUNE 2014] The Unified Modeling Language is a is a modeling Language used for specifying, constructing,visualizing and documentation of the software system and its components. It includes Graphical tools which helps anlaysis and design in object oriented software Engineering. It is directly connected to programming Languages which help in coding the proposed system. The Unified Modeling Language was developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in the 1990s		
3	**What is Analysis and Design?** Analysis emphasizes an investigation of the problem and requirements, rather than a solution. Design 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.	**C303.1**	**BTL1**
4	**Define Design Class Diagrams [NOV/DEC 2015,MAY/JUNE 2016]** A static view of the class definitions is usefully shown with a design class diagram. This illustrates the attributes and methods of the classes. 	**C303.1**	**BTL1**
5	**What are the three ways and perspectives to Apply UML? [MAY/JUNE 2015, NOV/DEC 2015, NOV/DEC 2016, APR/MAY 2017]**	**C303.1**	**BTL1**

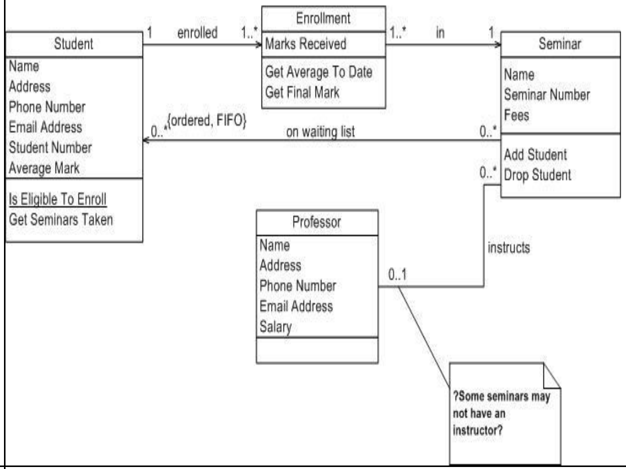
	<u>Three ways of using UML</u> 1. **UML as sketch**: Informal and incomplete diagrams created to explore difficult parts of the problem, or solution space. 2. **UML as blueprint:** Relatively detailed design diagrams used either for reverse engineering or forward engineering. i. If reverse engineering, a UML tool reads the source code and generate UML diagrams. ii. If Forward engineering using UML tool the software developers draw some diagrams and code can be generated automatically. 3. **UML as programming language**: (ie) the code can be generated automatically from the UML diagrams that are designed by the software engineers. Complete executable specification of a software system in UML. **<u>UML can be applied as three perspectives</u>** **1. Conceptual perspective** ➢ Using this perspective the things in the real world situation are described by the UML diagram. **2. Specification perspective** ➢ Using this perspective the UML diagrams describe the software abstractions or components with specifications and interfaces. **3. Implementation perspective** ➢ the diagrams describe software implementations in a particular technology(such as java).		
<u>6</u>	**What is Inception? [<u>APR /MAY 2011</u>]** 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	**C303.1**	**<u>BTL1</u>**

	requirement, creation of a business case, and preparation of the development Environment so that programming can start in the elaboration phase. Inception in one Sentence: Envision the product scope, vision, and business case.		
7	**What are Actors?[NOV/DEC 2011,APR/MAY 2018]** An actor is something with behavior, such as a person (identified by role), computer system, or organization; for example, a cashier.	**C303.1**	**BTL1**
8	**What is a scenario?** A scenario is a specific sequence of actions and interactions between actors and the system; it is also called a use case instance. It is one particular story of using a system, or one path through the use case; for example, the scenario of successfully purchasing items with cash, or the scenario of failing to purchase items because of a credit payment denial.	**C303.1**	**BTL1**
9	**Define Use case. [NOV/JUN 2013, NOV/DEC 2011, APR/MAY 2018]** A use case is a collection of related success and failure scenarios that describe an actor using a system to support a goal. Use cases are text documents, not diagrams, and use-case modeling is primarily an act of writing text, not drawing diagrams.  Creates Account Searches listings for item Places Bid Purchases Item Buyer Creates Account Searches listings for item Creates an Auction Ships Item Seller	**C303.1**	**BTL1**
10	**What are Three Kinds of Actors?** **PRIMARY ACTOR:**	**C303.1**	**BTL1**

	➢ has user goals fulfilled through using services of the SuD(system under design). Example: librarian is the primary actor for the usecase issuing of books. **<u>SUPPORTING ACTOR:</u>** ➢ provides services to the SuD.. Example: Payment validation system is a supporting actor for the online purchase system. **<u>OFF STAGE ACTOR:</u>** ➢ These actors help in behavior of the system. <u>For example</u>: Tax calculation agency is the off stage actor		
<u>11</u>	**What Tests Can Help Find Useful Use Cases? [<u>MAY/JUNE 2016,NOV/DEC 2016]</u>** 1. The Boss Test 2. The EBP Test 3. The Size Test	**C303.1**	**<u>BTL1</u>**
<u>12</u>	**What are Use Case Diagrams?** Use cases are text documents, not diagrams, and use-case modeling is primarily an act of writing text, not drawing diagrams. Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) can perform in collaboration with one or more external users of the system (actors).	**C303.1**	**<u>BTL1</u>**
<u>13</u>	**What is an activity diagram[<u> APR/MAY 2018]</u>** 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. Basic UML activity diagram notation illustrates an	**C303.1**	**<u>BTL1</u>**

	action, partition, fork, join, and object node. In essence, this diagram shows a sequence of actions, some of which may be parallel. Most of the notation is self-explanatory; two subtle points: once an action is finished, there is an automatic outgoing transition the diagram can show both control flow and data flow		
14	**What are interactive diagrams?List out the components involved in interactive diagrams?[NOV /DEC 2012, MAY/JUNE 2013]** The term *interaction diagram* is a generalization of two more specialized UML diagram types; both can be used to express similar message interactions: Collaboration diagrams Sequence diagrams **Components involved in interactive diagrams** i.Lifeline Boxes and Lifelines ii.Singleton objects iii.Messages	**C303.1**	**BTL1**
15	**What is the use of component diagram?[NOV/DEC 2011, MAY/JUNE 2012] [MAY/JUNE 2013]** A component diagram shows how physical components of a system are organized.A component diagram represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A Component defines its behavior in terms of provided and required interfaces.	**C303.1**	**BTL1**
16	**Give the use of UML state diagram? [MAY/JUNE 2014]** 1. Represents the events and states of object and the behavior of object in reaction to an	**C303.1**	**BTL1**

	event 2. Shows the life cycle of the object		
17	**What is meant by State chart Diagrams?** A UML state chart diagram, illustrates the interesting events and states of an object, and the behavior of an object in reaction to an event. Transitions are shown as arrows, labeled with their event. States are shown in rounded rectangles. It is common to include an initial pseudo-state, which automatically transitions to another state when the instance is created.  transition action off hook / play dial tone [valid subscriber] Idle Active on hook guard condition Fig: Transaction action and guard notation	**C303.1**	**BTL1**
18	**Distinguish between method and message in object? [MAY/JUNE 2015, NOV/DEC 2015]** **Methods** are similar to functions, procedures or subroutines in more traditional programming languages. **Messag**e essentially are non-specific function calls. **Method** is the implementation. **Message** is the instruction. In an object oriented system, a **method** is invoked by sending an object a message. An object understans a message when it can match the message to a method that has the same name as the message.	**C303.1**	**BTL1**
19	**What is Analysis?** **Analysis: -** Analysis emphasizes on investigation of the problem and requirements rather than a solution**.** For example if a new	**C303.1**	**BTL1**

	online trading system is desired, how will it be used? what are its functions?		
<u>**20**</u>	**What is Design?** Design emphasizes a conceptual solution that fulfills the requirements, rather than its implementation. For example a description of a database schema and software objects..	**C303.1**	<u>**BTL1**</u>
<u>**21**</u>	**Define class Diagrams <u>(MAY/JUN 2016)</u>** It will show the attributes and methods of the classes. The class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application.  Student Name Address Phone Number Email Address Student Number Average Mark Is Eligible To Enroll Get Seminars Taken 1 enrolled 1..* Enrollment Marks Received Get Average To Date Get Final Mark 1..* in 1 Seminar Name Seminar Number Fees Add Student Drop Student 0..* {ordered, FIFO} on waiting list 0..* 0..* instructs Professor Name Address Phone Number Email Address Salary 0..1 ?Some seminars may not have an instructor?	**C303.1**	**BTL1**
<u>**22**</u>	**Define Software development process** A software development process or life cycle is a structure imposed on the development of a software product. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process. It describes an approach to building, deploying and possibly	**C303.1**	<u>**BTL1**</u>

	maintaining software.		
23	**What is the use of Unified Process ?** The UP has emerged as a popular software development process for building object-oriented systems.The Unified Process is a design framework which guides the tasks, people and products of the design process. It is a framework because it provides the inputs and outputs of each activity. The Unified Software Development Process or Unified Process is a popular iterative and incremental software development process framework. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process(RUP). Other examples are OpenUP and Agile Unified Process.	**C303.1**	**BTL1**
24	**Define Iterative and Incremental Development?** The system grows incrementally over time iteration and thus this approach is also know as iterative and incremental development. Iterative and Incremental development is any combination of both iterative design or iterative method and incremental build model for software development. The combination is of long standing and has been widely suggested for large development efforts.	**C303.1**	**BTL1**
25	**Benefits of Iterative Development?** • Early visible progress • Managed complexity the team is not overwhelmed by analysis paralysis or very long and complex steps • The learning within an iteration can be methodically used to improve the development process itself, Iteration by iteration.	**C303.1**	**BTL1**

26	**Define Development Case?** The choice of UP artifacts for a project may be written up in a short document called the Development Case. It can show the sets of possible interactions between the system and the people who use it.It can also show interactions between computer systems.	**C303.1**	**BTL1**
27	**What is object oriented system development methodology?** Object oriented system development methodology is a way to develop software by building self-contained modules or objects that can be easily replaced,modified and reused.	**C303.1**	**BTL1**
28	**How associations are used in UML?** • Associations are used in UML to represent the relationships between the classes of the system. • The System is operational only because of the presence of these associations. The associations are used to represent many types of relationships like binary relationships, ternary relationships, aggregation, composition, descriptive association by using association classes. • No design is complete without associations.	**C303.1**	**BTL1**
31	**What are the primary goals in the design of UML? [NOV/DEC 2016]** The primary goals in the design of the UML were: • Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models. • Provide extensibility and	**C303.1**	**BTL1**

	specialization mechanisms to extend the core concepts. • Be independent of particular programming languages and development processes. • Provide a formal basis for understanding the modeling language. • Encourage the growth of the OO tools market. • Support higher-level development concepts such as collaborations, frameworks, patterns and components. • Integrate best practices.		

PART-B

S.NO	QUESTIONS	CO	BLOOOM'S LEVEL
1	List various UML diagrams and explain it.[**MAY/JUNE 2014, NOV/DEC 2015,APR/MAY 2017]**	**C303.1**	**BTL1**
2	What do you mean by unified process in OOAD? Explain the phases with suitable diagrams? **[APR/MAY 2011,NOV/DEC 2011,MAY/JUNE 2012, NOV/DEC 2012,MAY/JUNE 2013, NOV/DEC 2013, NOV/DEC 2015, MAY/JUNE 2016,NOV/DEC 2016, APR/MAY 2017, NOV/DEC 2017]**	**C303.1**	**BTL1**
3	What is UML Activity Diagram? Using an example explain the features of basic UML activity diagram notation**. [NOV/DEC 2013, MAY/JUNE 2016, NOV/DEC 2016]**	**C303.1**	**BTL1**
4	List the UML notation for class diagram with example& Explain the concepts of link, association and inheritance? **[MAY/JUNE 2012, MAY/JUNE 2013].**	**C303.1**	**BTL1**
5	Explain about Interaction Diagram Notation (OR) Apply Interative modeling for a payroll system in UML **[APR/MAY 2011, NOV/DEC 2011, NOV/DEC 2013, NOV/DEC 2016, APR/MAY 2018**]	**C303.1**	**BTL2**

6	Explain about Usecase Model for a case study of your choice [**NOV/DEC 2015]**	**C303.1**	**BTL2**
7	Explain a Problem statement for Library Management system .Draw the UML usecase ,Activity,Class,Sequence ,State chart,Package,Component and Deployment diagram [**MAY/JUNE 2016]**	**C303.1**	**BTL2**
8	Draw and discuss an analysis model for banking system [**APR/MAY 2018]**	**C303.1**	**BTL2**
9	Explain the software development life cycle of object oriented approach [**APR/MAY 2018]**	**C303.1**	**BTL2**

UNIT-II

DESIGN PATTERNS

GRASP: Designing objects with responsibilities – Creator – Information expert – Low Coupling – High Cohesion – Controller - Design Patterns – creational - factory method - structural – Bridge – Adapter - behavioral – Strategy – observer

PART- A

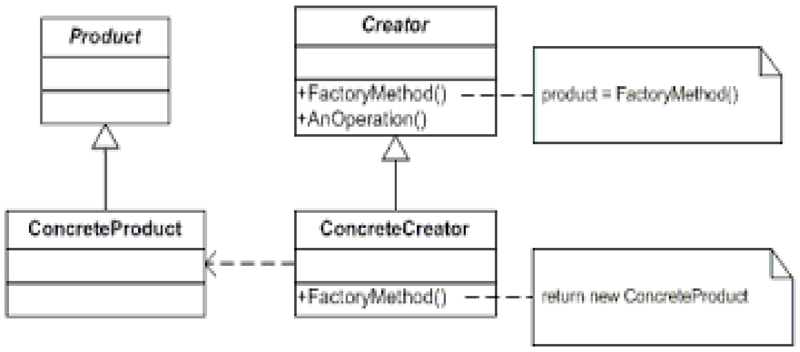
S.NO	QUESTIONS	CO	BLOOM'S LEVEL
1	**Define patterns (or) when to use patterns[NOV/DEC 2011,MAY/JUNE 2012, MAY/JUNE 2013, MAY/JUN 2014, NOV/DEC 2015, NOV/DEC 2016]** • The "patterns" provide a representation of nine basic principles that form a foundation for designing object-oriented systems. • A patternis a named problem/solution pair that can be applied in new context, with advice on how to apply it in novel situations and discussion of its trade-offs. The following sections present the first five GRASP patterns: • Information Expert • Creator • High Cohesion • Low Coupling • Controller	C303.2	**BTL1**
2.	**What is GRASP? How to Apply the GRASP Patterns? [MAY/JUNE 2013]** **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**attern**:** They describe the fundamental principles of object design and responsibility assignment, expressed as patterns. The following sections present the first five GRASP	C303.2	**BTL1**

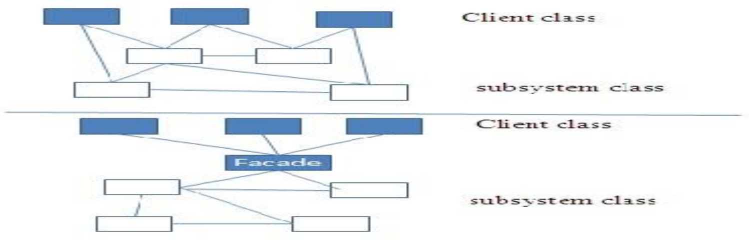
	patterns: Information Expert, Creator.,High Cohesion ,Low Coupling , Controller. **Who is creator?** **<u>Creator</u>** Creation of objects is one of the most common activities in an object oriented system. Which class is responsible for creating objects is a fundamental property of the relationship between objects of particular classes. **Name:** Creator **Problem**: Who should be responsible for creating a new instance of some class ? **Solution** ■ Assign class B the responsibility to create an instance of class A if one or more of the following is true : ■ B aggregates A objects . ■ B contains A objects. ■ B records instances of A objects. ■ B closely uses A objects. ■ B has the initializing data that will be passed to A when it is created ■ B is a creator of A objects.		
3.	**List out some scenarios that illustrate varying degrees of functional cohesion.** -Very low cohesion -low cohesion -High cohesion -Moderate cohesion	C303.2	**BTL1**
4.	**Define Modular Design <u>[MAY/JUNE 2016, APR/MAY 2017]</u>** Modular design, or "modularity in design", is a design approach that subdivides a system into smaller parts called modules or skids, that can be independently created and then used in different systems.	C303.2	**BTL1**

5.	**What are the advantages of Factory objects?** • 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.	C303.2	**BTL1**
6.	**What is meant by Abstract Class Abstract Factory?** A common variation on Abstract Factory is to create an abstract class factory that is accessed using the Singleton pattern, reads from a system property to decide which of its subclass factories to create, and then returns the appropriate subclass instance. This is used, for example, in the Java libraries with the *java.awt.Toolkit* class, which is an abstract class abstract factory for creating families of GUI widgets for different operating system and GUI subsystems.	C303.2	**BTL1**
7.	**What is meant by Fine-Grained Classes?** Consider the creation of the *Credit Card, Drivers License,* and *Check* software objects. Our first impulse might be to record the data they hold simply in their related payment classes, and eliminate such fine-grained classes. However, it is usually a more profitable strategy to use them; they often end up providing useful behavior and being reusable. For example, the *Credit Card* is a natural Expert on telling you its credit company type (Visa, MasterCard, and so on). This behavior will turn out to be necessary for our application.	C303.2	**BTL1**
8.	**How to Choosing the Initial Domain Object?** Choose as an initial domain object a class at or near the root of the containment or aggregation hierarchy of domain objects. This may be a facade controller, such as *Register,* or some other object considered to contain all or most other objects, such as a *Store.*	C303.2	**BTL1**
9.	**How to Connecting the UI Layer to the Domain Layer?** • An initializing routine (for example, a Java *main* method) creates both a UI and a domain object, and passes the domain object to the UI. • A UI object retrieves the domain object from a well-known source, such as a factory object that is responsible for creating domain objects.	C303.2	**BTL1**
10.	**What is Interface and Domain Layer Responsibilities.[MAY/JUN2016]** The UI layer should not have any domain logic responsibilities. It should	C303.2	**BTL1**

	only be responsible for user interface tasks, such as updating widgets. The UI layer should forward requests for all domain-oriented tasks on to the domain layer, which is responsible for handling them.		
11	**Elaorate GRASP methodical approach to learning basic object design?** General Responsibility Assignment Software Patterns (or Principles), abbreviated GRASP, consists of guidelines for assigning responsibility to classes and objects in object-oriented design. GRASP patterns are a learning aid to help one understand essential object design, and apply design reasoning in a methodical, rational, explainable way.	C303.2	**BTL5**
12	**Define GRASP patterns?** .**GRASP patterns:** Creator, Information Expert, Low Coupling, Controller, High Cohesion, Indirection, Polymorphism, Protected Variations ,Pure Fabrication.	C303.2	**BTL1**
13	**Define GRASP responsibilities?** Responsibilities is "a contract or obligation of a classifier" (UML definition). Responsibilities can include behaviour, data storage, object creation and more. They often fall into two categories: **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** • Knowing responsibilities of object include : – Knowing about private encapsulated data – Knowing about related objects – Knowing about things it can derive or calculate	C303.2	**BTL1**
14	**Define cohesion?** Cohesion – the degree to which the information and responsibilities of a class are related to each other. Cohesion refers to the degree to which the elements of a module belong together. Thus, it is a measure of how strongly related each piece of functionality expressed by the source code of a software module is.	C303.2	**BTL1**
15	**Define coupling? (<u>Nov/Dec 2013)</u>**	C303.2	**BTL1**

	Coupling of classes is a measure of how strongly a class is connected to another class. Coupling is the degree to which one class knows about another class. Let us consider two classes class **A** and class **B**. If class **A** knows class**B** through its interface only i.e it interacts with class **B** through its API then class **A** and class **B** are said to be loosely coupled.		
16	**Define low coupling? <u>[May/June 2014]</u>** • Assign a responsibility so that coupling remains low. This is pretty vague, but it means that low number of classes to which a class is coupled. Creator is a more specific case of Low Coupling, related to instantiation. Low Coupling is an evaluative pattern, which dictates how to assign **responsibilities to support:** lower dependency between the classes, change in one class having lower impact on other classes, higher reuse potential	C303.2	**BTL1**
17	**Define high cohesion?** • 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. • Alternatively, low cohesion is a situation in which a given element has too many unrelated responsibilities. Elements with low cohesion often suffer from being hard to comprehend, hard to reuse, hard to maintain and averse to change.	C303.2	**BTL1**
18	**What is creator and its responsibilities?** 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.Assign the responsibility for receiving and handling a system event message to a class that is either: • Representative of the entire subsystem (e.g. a Façade Controller). • Representative of the entire use case scenario.	C303.2	**BTL1**

	 Product
	Creator
	+FactoryMethod()
	+AnOperation()
	product = FactoryMethod()
	ConcreteProduct
	ConcreteCreator
	+FactoryMethod()
	return new ConcreteProduct		

19	**What is facade controller?** The GRASP mentioned that a Controller that represents an entire subsystem might be called a Façade Controller. A facade is an object that provides a simplified interface to a larger body of code, such as a class library.  Client class subsystem class Client class Facade subsystem class	C303.2	**BTL1**
20	**What is polymorphism?** In object-oriented programming, polymorphism is the characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a **BTL1**function, or an object to have more than one form. There are several different kinds of polymorphism. When related behaviours vary by type (class), assign the responsibility polymorphically to the specialization classes. This is basically the purpose of polymorphism, so it is natural for software developers to understand. This is not much of a pattern, and yet another best practice.	C303.2	**BTL1**
21	**What about pure fabrication?** To support high cohesion and low coupling, where no appropriate class is present, invent one even if the class does not represent a problem domain concept .This is a compromise that often has to be made to preserve cohesion and low coupling. This kind of class is called "Service" in Domain-driven design.	C303.2	**BTL1**
22	**Define indirection?**	C303.2	**BTL1**

	To avoid direct coupling between objects, assign an intermediate object as a mediator.Recall that coupling between two classes of different subsystems can introduce maintenance problems. Another possibility is that two classes would be otherwise reusable (in other contexts) except that one has to know of the other.		
23	**What is protected variations?** Assign responsibility to create a stable interface around an unstable or predictably variable subsystem or component. If a component changes frequently, the users of the component will also have to be modified.This is especially time consuming if the component has many users.	C303.2	**BTL1**
24	**What is Responsibility-Driven Design?** **Responsibility-driven design** is a design technique in object-oriented programming. A popular way of thinking about the design of software objects and also larger scale components are in terms of responsibilities, roles and collaborations. Responsibility-driven design is inspired by the client/server model. It focuses on the contract by asking: What actions is this object responsible for? What information does this object share? This is part of a larger approach called responsibility driven design or RDD.	C303.2	**BTL1**
25	**What are the advantages of Factory objects?** • 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.	C303.2	**BTL1**
26	**Define an example for Information Expert pattern or principle?** Information expert is a principle used to determine where to delegate responsibilities. Information expert will lead to placing the responsibility on the class with the most information required to fulfill it. **Name:** Information Expert **Problem:-** What is a general principle of assigning responsibilities to objects?	C303.2	**BTL1**

	Solution:- Assign a responsibility to the information expert- the class that has the information necessary to fulfill the responsibility.**Example**: - Association sale:		
27	**Determine the use of Design patterns? (Nov/Dec 2013) [Nov/Dec 2014]** • **Understandability**: Classes are easier to understand in isolation. • **Maintainability:** Classes aren't affected by changes in other components. • **Reusability**: easier to grab hold of classes.	C303.2	**BTL5**
28	**List the differences between design patterns and frameworks.** • Design patterns are more abstract than frameworks. • Design patterns are smaller architectural elements than frameworks. • Design patterns are less specialized than frameworks.	C303.2	**BTL1**
29	**Distinguish between coupling and cohesion [MAY/JUNE 15, NOV/DEC 2016, APR/MAY 2017]**	C303.2	**BTL4**

Coupling is a measure of how strongly one element is connected to has knowledge of or relies on other elements	Cohesion is a measure o related and focused the re an elelment.
These elements include classes, subsysytems, and so on.	These elements include cla and so on.
A class with high coupling relies on many other classes	A class with low cohes unrelated things or does too

30	**Mention the list of behavourial pattern used during design phase of software development [APR/MAY 2018]** • Chain of Responsibility Pattern • Command Pattern • Interpreter Pattern • Iterator Pattern • Mediator Pattern	C303.2	**BTL4**
31	**List out the types of coupling [APR/MAY 2018]** • Content coupling • Common coupling • Stamp Coupling • Control Coupling • Data Coupling	C303.2	**BTL4**

PART-B

S.NO	QUESTIONS	CO	BLOOMS LEVEL

1	Explain the design principles in object modeling.Explain about GRASP Patterns. **[APR/MAY 2011, NOV/DEC 2011, NOV/DEC 2013, MAY /JUN 2014,MAY/JUNE 2016] NOV/DEC 2016,APR/MAY 2017, APR/MAY 2018**]	**C303.2**	**BTL2**
2	Explain on adapter, singleton, strategy, factory & observer patterns. **[APR/MAY 2011, MAY/JUNE 2013, NOV/DEC 2013, MAY/JUN 2014]**	**C303.2**	**BTL2**
3	**Determine** the concepts of Creator, Low coupling, Controller and High cohesion**,**Information Expert **[MAY/JUNE 2012, MAY/JUNE 2013, NOV/DEC 2015, NOV/DEC 2016, NOV/DEC 2017]**	**C303.2**	**BTL5**
4	**List out** Designing concepts on the Use-Case Realizations with GoF Design Patterns**.[NOV/DEC 2011, MAY/JUNE 2016]]**	**C303.2**	**BTL1**
5	Discuss In detail about Structural Patterns**?**	**C303.2**	**BTL6**
6	Discuss in dateail about Behavioral Patterns**?[NOV/DEC 2015]**	**C303.2**	**BTL6**
7	State role and pattern while developing system design**?** **[NOV/DEC 2015]**	**C303.2**	**BTL1**
8	Differentiate bridge and adaptor**?[NOV/DEC 2015]**	**C303.2**	**BTL1**
9	Explain in detail about the Factory pattern. Mention the Limitations and applications of Factory Pattern [**NOV/DEC 2015, NOV/DEC 2017]**	**C303.2**	**BTL2**
10	Write short notes on adaptor patternand observer pattern Compare different categories of design pattern[**APR/MAY 2018]**	**C303.2**	**BTL2**

UNIT III

Case study – the Next Gen POS system, Inception -Use case Modeling - Relating Usecases include, extend and generalization - Elaboration - Domain Models - Finding conceptual classesand description classes – Associations – Attributes – Domain model refinement – Finding conceptual class Hierarchies - Aggregation and Composition.

PART- A

S.NO	QUESTIONS	CO	BLOOM'S LEVEL
1	**What is the need for modeling? [MAY/JUNE 2014]** The purpose of modeling is to discover,understand and share the understanding. Create models in parallel. For example,start sketching in one whiteboard, Dynamic View UML Interaction diagram,and in another whiteboard,the static view,the UML Class Diagram. This is done during requirement analysis phase. Along with models, other documents such as glossary, business rules and standards also be created. Include UML diagrams optionally. Use case diagrams shows the actors and related functionalities.	C303.3	**BTL1**
2	**What Artifacts May Start in Inception?** Some sample artifacts are Vision and Business Case, Use-Case Model, Supplementary Specification, Glossary, Risk List & Risk Management Plan, Prototypes and proof-of-concepts etc.	C303.3	**BTL1**

3	**Define Requirements and mention its types.** Requirements are capabilities and conditions to which the system and more broadly, the project must conform. 1. Functional 2. Reliability 3. Performance 4. Supportability	C303.3	**BTL1**
4	**What is Elaboration?[MAY/JUNE 2012, MAY/JUNE 2013, MAY/JUN 2014]** **Elaboration is the initial series of iterations during which, on a norn project** **1. The core, risky software architecture is programmed and teste** **2. The majority of requirements are programmed and stabilized.** **3. The major risks are mitigated or retired.** **It is called Executable architecture or Architectural baseline.** ➢ **Elaboration often consists two-four iterations; each iteration recommended to be two-six weeks, unless the team size is massive.**	C303.3	**BTL1**
5	**What is Aggregation and composition? [APR/MAY 2011, MAY/JUNE 2012, MAY/JUNE 2013, NOV/DEC 2013, MAY/JUN 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 (such as a Square) belongs to only one composite instance (such as one Board) at a time, 2) the part must always belong to a composite (no free-floating Fingers), 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.	C303.3	**BTL1**
6	**What is a Domain Model? [NOV /DEC 2013, APR/MAY 2011]** A domain model is a visual representation of conceptual classes or real-situation objects in a domain. The term "Domain Model" means a representation of real-situation conceptual classes, not of software objects. The term does not mean a set of diagrams describing software classes, the domain layer of a software architecture, or software objects with responsibilities	C303.3	**BTL1**

7	**What are the tasks performed in elaboration?[MAY/JUNE 15,NOV/DEC 2015, APR/MAY 2018]** • The core, risky software architecture is programmed and tested • The majority of requirements are discovered and stabilized • The major risks are mitigated or retired	C303.3	**BTL1**
8	**What are the key ideas and best practices that will manifest in elaboration?** • do short time boxed risk-driven iterations • start programming early • adaptively design, implement, and test the core and risky parts of the architecture • test early, often, realistically • adapt based on feedback from tests, users, developers • write most of the use cases and other requirements in detail, through a series of workshops, • once per elaboration iteration	C303.3	**BTL1**
9	**What Artifacts May Start in Elaboration?**	C303.3	**BTL1**

Domain Model	This is a visualization of the domain concep static information model of the domain entiti
Design Model	This is the set of diagrams that describes the includes software class diagrams, object i package diagrams, and so forth.
Software Architecture Document	A learning aid that summarizes the key arc their resolution in the design. It is a summar design ideas and their motivation in the syste
Data Model	This includes the database schemas, and the between object and non-object representatio
Use-Case Storyboards, UI Prototypes	Descriptions of the user interface, paths of models, and so forth.

10	**What are Conceptual Classes? [MAY/JUNE 2016]** The domain model illustrates conceptual classes or vocabulary in the domain. Informally, a conceptual class is an idea, thing, or object. More formally, a conceptual class 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	C303.3	**BTL1**
11	**How to Create a Domain Model?[MAY/JUNE 15, NOV/DEC 2015, NOV/DEC2016]**	C303.3	**BTL1**

Steps to create Domain Model are:

Find the conceptual classes (see a following guideline).
Draw them as classes in a UML class diagram.
Add associations and attributes.

12 **How to Find Conceptual Classes?** C303.3 **BTL1**

Three strategies to find conceptual classes are.

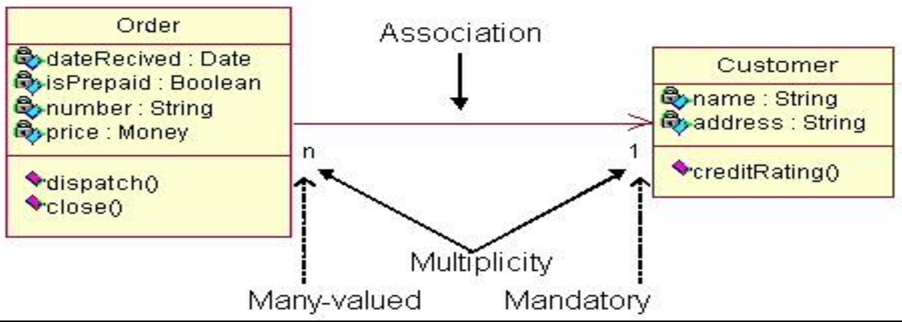
1. Reuse or modify existing models
2. Use a category list
3. Identify noun phrases

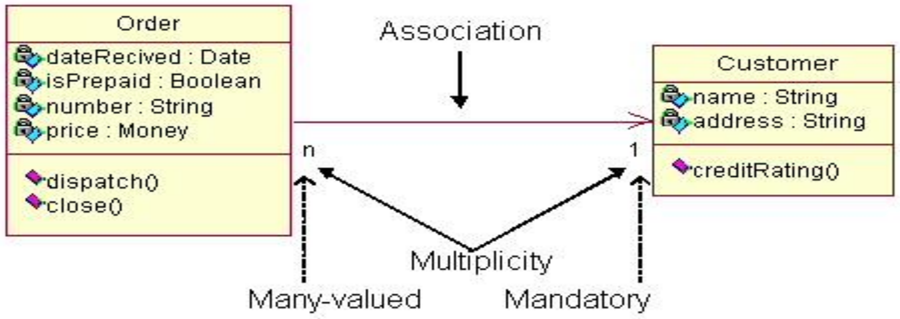
13 **List some Conceptual Class Category.** C303.3 **BTL1**

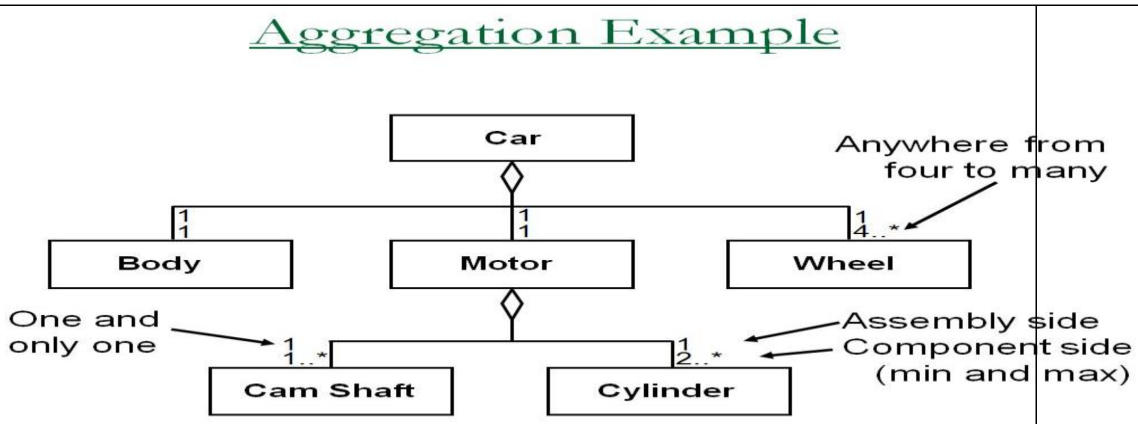
Conceptual Class Category	**Examples**
business transactions	Sale, Payment Reservatio
transaction line items	Sales Line Item
product or service related to a transaction or transaction line item	Item Flight, Seat, Meal
where is the transaction recorded?	Register, Ledger Flight M
roles of people or organizations related to the transaction; actors in the use case	Cashier, Customer, S Player Passenger, Airline
place of transaction; place of service	Store Airport, Plane, Seat

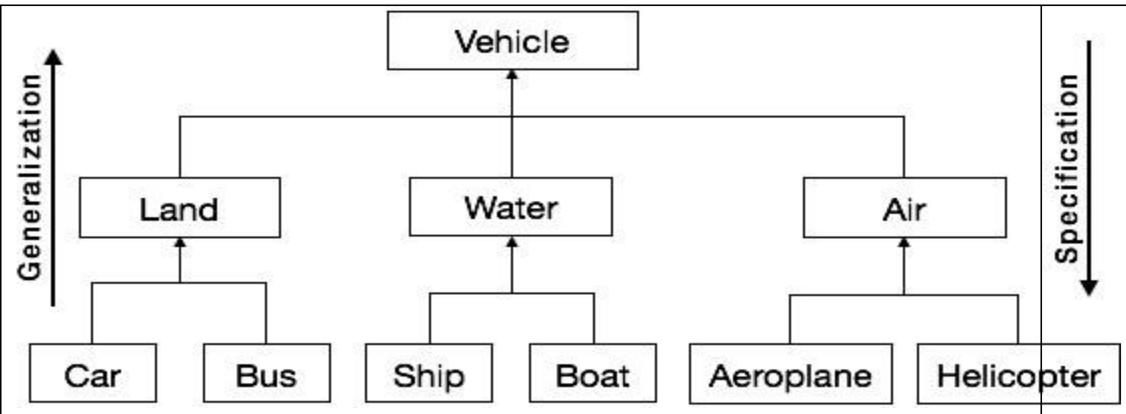
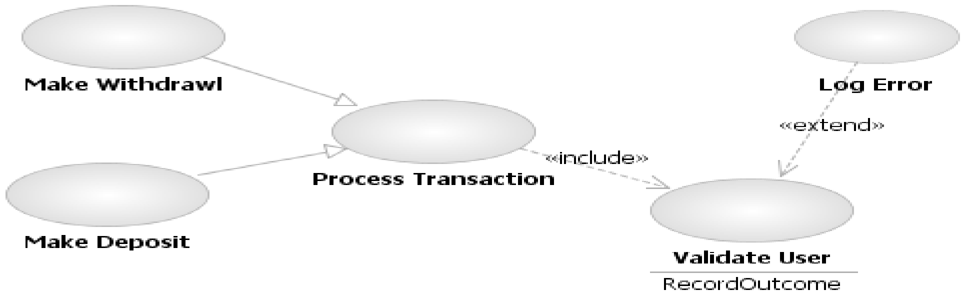
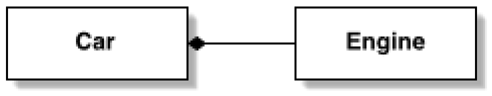
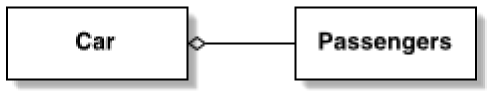
14 **Define Association.** C303.3 **BTL1**

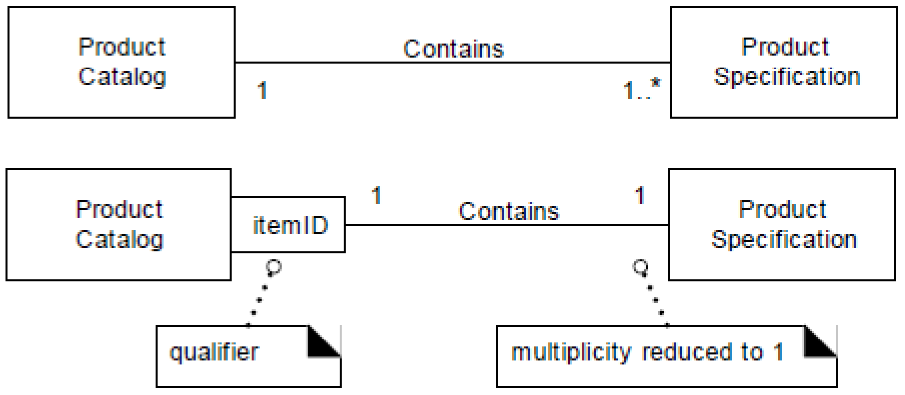
An **association** is a relationship between classes (more precisely, instances of those classes) that indicates some meaningful and interesting connection.



15	**Why Should We Avoid Adding Many Associations?** We need to avoid adding too many associations to a domain model. Digging back into our discrete mathematics studies, you may recall that in a graph with n nodes, there can be associations to other nodes a potentially very large number. A domain model with 20 classes could have 190 associations' lines! Many lines on the diagram will obscure it with "visual noise."	C303.3	**BTL1**
16	**How to Name an Association in UML?** Name an association based on a **Class Name-Verb Phrase-Class Name format** where the verb phrase creates a sequence that is readable and meaningful.  Order dateRecived : Date isPrepaid : Boolean number : String price : Money dispatch() close() Association Customer name : String address : String creditRating() n 1 Multiplicity Many-valued Mandatory	C303.3	**BTL1**
17	**What is an Aggregation? [Nov/Dec 2013,MAY/JUNE 2014]** Aggregation is a vague kind of association in the UML that loosely suggests whole part relationships. Aggregation is a variant of the "has a" or association relationship; aggregation is more specific than association. It is an association that represents a part-whole or part-of relationship. An aggregation may not involve more than two classes.	C303.3	**BTL1**

	 Aggregation Example Car Body Motor Wheel Cam Shaft Cylinder 1 1 1 1 1 4..* Anywhere from four to many One and only one 1 1..* 1 2..* Assembly side Component side (min and max)		
18	**What about attributes in Code?** The recommendation that attributes in the domain model be mainly datatypes does not imply that C# or Java attributes must only be of simple, primitive datatypes. The domain model is a conceptual perspective, not a software one. In the design model, attributes may be of any type.	C303.3	**BTL1**
19	**Define business Modeling** When developing a single application, this includes domain object modeling. When engaged in large scale business analysis or business process reengineering, this include dynamic modeling of the business process across the entire enterprise.	C303.3	**BTL1**
20	**Define inception step.** 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 so that programming can start in the following elaboration phase.	C303.3	**BTL1**
21	**What is generalization relationship?** It is a relationship in which one model element (the child) is based on another model element Generalization relationships are used in class, component, deployment, and use-case diagrams to indicate that the child receives all of the attributes, operations, and relationships that are defined in the parent.	C303.3	**BTL1**

	 Vehicle Land Water Air Car Bus Ship Boat Aeroplane Helicopter Generalization Specification		
22	**What is exclude relationship?** In UML modeling, you can use an extend relationship to specify that one use case (extension) extends the behavior of another use case (base).  Make Withdrawl Make Deposit Process Transaction «include» Log Error «extend» Validate User RecordOutcome	C303.3	**BTL1**
23	**What is composition? (Nov/Dec 2013) (May/June 2014)** Composition ,also known as composite aggregation, is a strong kind of whole-part aggregation and is useful to show in some models. Composition is a special type aggregation where the 'has-a' relationship is more strong. For example an university has departments which cannot exist on their own with the containing 'university' entity  Composition: every car has an engine.  Aggregation: cars may have passengers, they come and go	C303.3	**BTL1**
24	**Distinguish Aggregation and containment.**	C303.3	**BTL4**

	Aggregation is the relationship between the whole and a part. We can add/subtract some properties in the part (slave) side. It won't affect the whole part. Best example is Car, which contains the wheels and some extra parts. Even though the parts are not there we can call it as car. But, in the case of containment the whole part is affected when the part within that got affected. The human body is an apt example for this relationship. When the whole body dies the parts (heart etc) are died.		
25	**List the relationships used in class diagram (Nov/Dec 2014,NOV/Dec 2015,May/June 2014)** Objects (of certain class), with attributes,operations. Links between Objects, Aggregations between Objects. Compsition Generalization	C303.3	**BTL1**
26	**What is qualified association?[MAY/JUN2016]** A qualified association has a qualifier that is used to select an object from a larger set of related objects based upon the qualifier key. It reduces the multiplicity at the target end of the association, usually down from many to one because it implies the selection of usually one instance from a larger set. **EX:** If a ProductCatalog contains many ProductDescriptions and each one can be selected by an itemID.  Product Catalog Contains 1 1..* Product Specification Product Catalog itemID 1 Contains 1 Product Specification qualifier multiplicity reduced to 1 *Qualified association.*	C303.3	**BTL1**
27	**What are the 3 relationships that can be shown in UML diagram? Define them** **1. Association**: how are objects associated? This information will guide	C303.3	**BTL1**

	us in designing classes. **2**. **Super-Sub structure**: How are objects organized into super classes and sub classes? This information provides us the direction of inheritance. **3. Aggregation** and a part of structure: What is the composition of complex classes? This information guides us in defining mechanisms that properly manage object within object.		
28	**What are the advantages of inception?** **1.** Estimation or plans are expected to be reliable. 2. After inception, design architecture can be made easily because all the use cases are written in detail. 3. The life-cycle objectives of the project are stated, so that the needs of every stakeholder are considered. 4. Scope and boundary conditions, acceptance criteria and some requirements are established.	C303.3	**BTL1**
29	**What are the three strategies to find conceptual classes?** There are three strategies. 1. Reuse or modify existing models. 2. Use a category list. 3. Identify noun phrases.	C303.3	**BTL1**
30	**When to model with ‘Description Classes’?** A description class contains information that describes something else. For example, a product description that records the price,picture, and text description of an item.	C303.3	**BTL1**
31	**When are Description Classes useful?** Add a Description Class When: 1. There needs to be a description about an item or service, independent of the current existence of any examples of those items or services. 2. Deleting instances of things they describe results in a loss of information that needs to be maintained. 3. It reduces redundant or duplicated information.	C303.3	**BTL1**
32	**When to define new data type classes?[MAY/JUN 2016]** Encapsulation is a development technique which includes creating new data types (classes) by combining both information (structure)	C303.3	**BTL1**

	and behaviors, and restricting access to implementation details.		
33	**Why call a Domain Model a Visual Dictionary**? [**NOV/DEC 2016]** Because it visualizes and relates words or concepts in the domain. It shows an abstraction of the conceptual classes. The information it illustrates (using a UML notation) could alternatively have been expressed in plain text. But it is easy to understand the terms and especially their relationships in a visual language	C303.3	**BTL1**
34.	**What is the relationship on conceptual superclass to subclass [APR/MAY 2017]**	C303.3	**BTL1**
35	**What is the purpose include and exclude relationship in usecase diagram [APR/MAY 2017]**	C303.3	**BTL1**
36	**List out the components of POS system**	C303.3	**BTL1**

PART- B

S.NO	QUESTIONS	CO	BLOOM'S LEVEL
1	Explain with an example, how use case modeling is used to describe functional requirements. Identify the actors, scenario and use case for the example? **[APR/MAY2011, MAY/JUNE 2012, MAY/JUNE 2013, NOV/DEC 2013, MAY/JUNE 2014, NOV/DEC 2016, APR/MAY 2018]**	**C303.3**	**BTL2**
2	Define Inception. Explain about artifacts of Inception**?**	**C303.3**	**BTL1**
3	Explain about Use-Case Model and its Writing Requirements in Context?	**C303.3**	**BTL2**
4	Discuss the strategies used to identify conceptual classes. Describe the steps to create a domain model used for representing conceptual classes. [**APR/MAY 2011, MAY/JUNE 2012, MAY/JUNE**	**C303.3**	**BTL6**

	2013, NOV/DEC 2013, MAY/JUN 2014, NOV/DEC 2016, APR/MAY 2017,APR/MAY 2018]		
5	Illustrate the concept of Domain model with examples [**MAY/JUNE 2016]**	**C303.3**	**BTL2**
6	Expalin in Deatil about Domain Model Refinement	**C303.3**	**BTL1**
7	Write about Elaboration and discuss the difference between Elaboration and Inception with suitable diagram for university domain **[NOV/DEC 2015, APR/MAY 2017]**	**C303.3**	
8	Construct design for Library Information system which comprises following notations 1.Aggregation 2.Composition 3.Associations[**NOV/DEC 2015 ,NOV/DEC 2016, APR/MAY 2017, NOV/DEC 2017, APR/MAY 2018]**	**C303.3**	
9	What are the guidelines for finding used to partition th classes in the domain model organized into packages .Explain with the suitable examples [**MAY/JUNE 2016]**	**C303.3**	
10	Explain with example on concrete uscase and an abstract use case**[NOV/DEC 2017]**	**C303.3**	
11	Explain with an example generalization and specialization and write a note on abstract class and abstract operation**[NOV/DEC 2017]**	**C303.3**	
12	What is multiciplicity of an association.Expalin with an example of different types of multiplicities **[NOV/DEC 2017]**	**C303.3**	

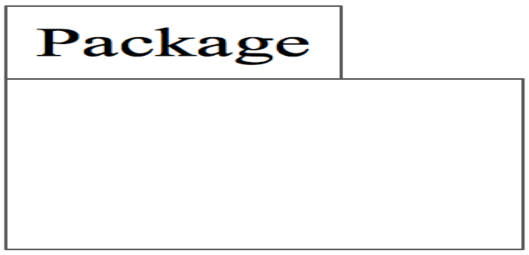
Unit-IV

APPLYING DESIGN PATTERNS

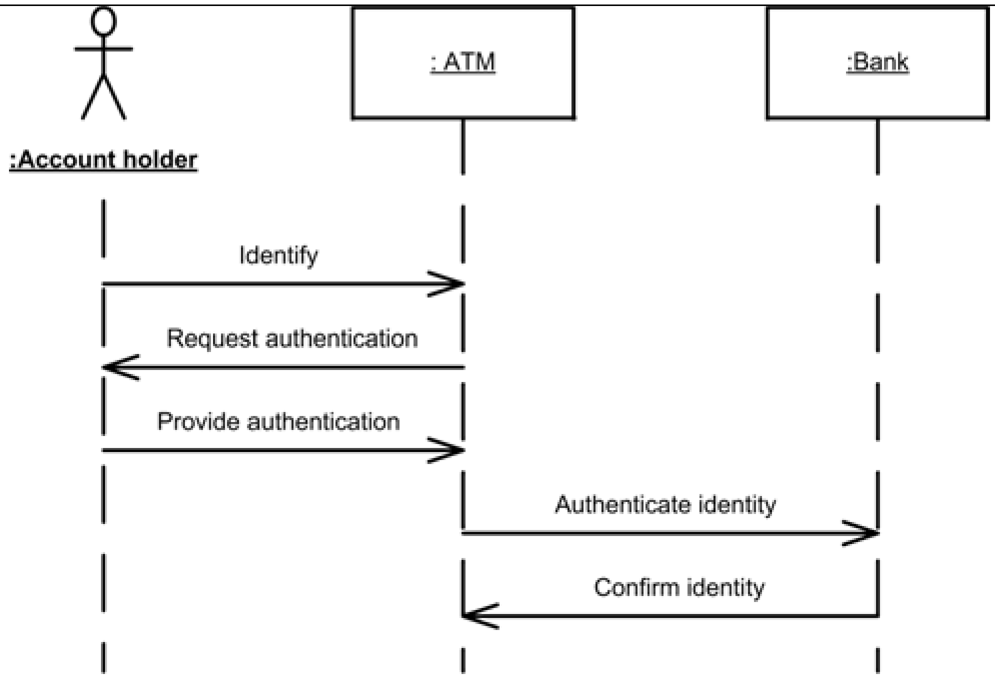
System sequence diagrams - Relationship between sequence diagrams and use cases Logical architecture and UML package diagram – Logical architecture refinement - UML class diagrams – UML interaction diagrams - Applying GoF design patterns.

PART- A

S.NO	QUESTIONS	CO	BLOOM'S LEVEL
1	**Define package and draw the UML notation for package?**[**MAY/JUNE 2012, MAY/JUNE 2013, NOV/DEC 2013, MAY/JUN 2014]** A UML package diagram provides a way to group elements. It can group anything: classes, other packages, use cases. UML package diagrams are often used to illustrate the logical architecture of a system -the layers, the subsystems, packages. The package name is placed on the tab if the package shows the inner members or on the main folder if not. Dependency or coupling is shown by the UML – dependency line – a dashed line with arrow pointing towards depended on package. Fully qualified names are represented in UML for example as java :: util:: date **UML notation for package**	C303.4	**BTL1**

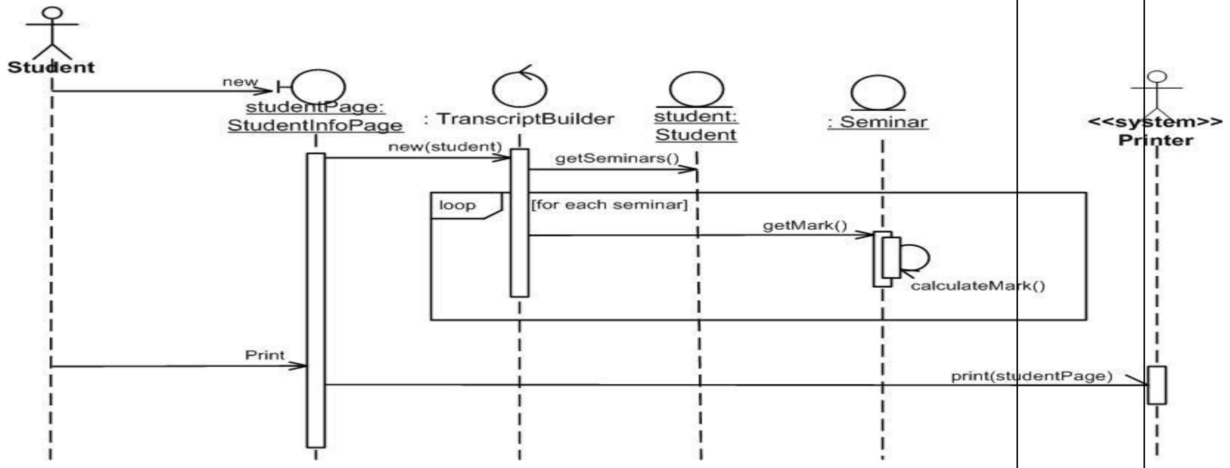
			
2	**. What is the use of system sequence diagram? <u>[APR/MAY 2011, NOV/DEC 2011,</u>** **<u>MAY/JUNE 2014</u>]** A System Sequence Diagram is an artifact that illustrates input and output events related to the system under discussion. A system sequence diagram (SSD) is a picture that shows, for a particular scenario of a use case, the events that external actors generate, their order, and inter-system events. All systems are treated as a black box; the emphasis of the diagram is events that cross the system boundary from actors to systems. A system sequence diagram is a picture that shows for one particular scenario of a usecase, the events that external actors generate, their order, and inter system events. All systems are treated as a black box.	C303.4	**BTL1**
3	**List the relationships used in class diagram. [<u>APR/MAY 2011,</u> <u>NOV/DEC 2013, MAY/JUNE 2014]</u> <u>[MAY/JUNE 2015</u>]** 1. Association 2. Aggregation 3. Composition 4. Dependency	C303.4	**BTL1**
4	**What is meant by System Sequence Diagrams?** A system sequence diagram (SSD) is a picture that shows, for a particular scenario of a use case, the events that external actors generate their order, and inter-system events. All systems are treated as a black box; the emphasis of the diagram is events that cross the system boundary from actors to systems	C303.4	**BTL1**
5	**What is meant by System Behavior? <u>[MAY/JUNE 2015,NOV/DEC 2015</u>]** **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. Other parts include the Use cases, and system contracts .	C303.4	**BTL1**

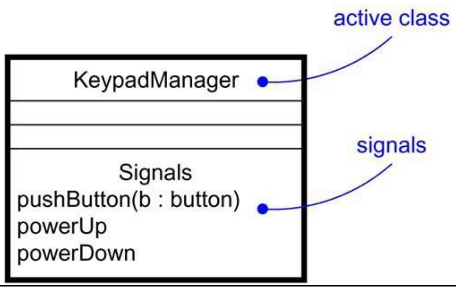
6	**What is meant by Inter-System SSDs?** SSDs can also be used to illustrate collaborations between systems, such as between the Next Gen POS and the external credit payment authorizer. However, this is deferred until a later iteration in the case study, since this iteration does not include remote systems collaboration.	C303.4	**BTL1**
7	**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 boundary, as discussed in the prior chapter 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.	C303.4	**BTL1**
8	**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.	C303.4	**BTL1**
9	**What is meant by link?** **A link** is a connection path between two objects; it indicates some form of navigation And visibility between the objects is possible . More formally, a link is an instance of an association. For example, there is a link or path of navigation from a *Register* to a *Sale*, along which messages may flow, such as the *make 2 Payment* message.	C303.4	**BTL1**
10	**. What is meant by Messages?** Each message between objects is represented with a message expression and small arrow indicating the direction of the message. Many messages may flow along this link. A sequence number is added to show the sequential order of messages in the current thread of control.	C303.4	**BTL1**
11	**Define SSD. Mention its use?** A system sequence diagram (SSD) is a picture that shows, for a particular scenario of use case, the events that external actors generate their order and inter system events. All systems are treated as a black box. The emphasis of the diagram is the events that cross the system boundary from actors to systems.	C303.4	**BTL1**

	 :Account holder : ATM :Bank Identify Request authentication Provide authentication Authenticate identity Confirm identity **Example for a System sequence diagram**		
12	**Define System events?** The system sequence diagrams shows system Events or I/O messages relative to the system. Input system events imply the system has standalone system operations to handle the events, just as an Object Oriented message (a kind of event or signal) is handled by an Object oriented method (a kind of operation).	C303.4	**BTL1**
13	**List out the frame operators in sequence diagram.** The common frame operators: • Alt : alternate fragment for mutual exclusion • Loop : loop fragment while guard is true • Opt : Optional fragment that executes if guard is true • Par: parallel fragments that execute in parallel. • Region: critical region within which only one thread can run.	C303.4	**BTL1**
14	**Define the strength and weakness of sequence and collaboration diagram.** **[APR/MAY 2017]**	C303.4	**BTL1**

Type	Strengths	Weaknesses
Sequence	Clearly shows sequence or time ordering of messages	Forced to extend to the right wh objects. Consumes horizontal space

	Large set of detailed notation option	
	Space economical flexibility to add new objects in two dimensions	More difficult to see sequence of messages Fewer notation options

15	**How to create instance in collaboration diagram. [APR/MAY 2018]** A message 'create' can be used to create an instance in a collaboration diagram. If another name is used, the message may be annotated with a UML stereotype, like <<create>>. The create message may include parameters indicating the passing of initial messages.	C303.4	**BTL1**
16	**What do you mean by synchronous and asynchronous call?** An asynchronous message call does not wait for a response. They are used in multi threaded environments such as .Net and Java so that the new thread executions can be created and initiated. When a task is being executed asynchronously, there is no need to wait for it to finish, before starting with another task. In Synchronous message calls, the task has to be completed before starting another task.  Student new studentPage: StudentInfoPage : TranscriptBuilder student: Student : Seminar <<system>> Printer new(student) getSeminars() loop [for each seminar] getMark() calculateMark() Print print(studentPage) **Example for Asynchronous and synchronous call**	C303.4	**BTL1**
17	**Define classifier.[MAY/JUNE 2016]** A UML classifier is a "model element that describes behavioural and structure features". Classifiers can also be specialized. They are a generalization of many of the elements of the UML, including classes, interfaces, use cases and actors. In class diagram, the two most common classifiers are regular classes and interfaces.	C303.4	**BTL1**

18	**How to show methods in class diagram?** A UML method is the implementation of an operation. If constraints are defined, the method must satisfy them. A method may be illustrated in class diagrams with a UML note symbol stereotype with <<method>>.	C303.4	**BTL1**
19	**Define Active class.** An active object runs on and controls its own thread of execution. Active classes are just Classes which represents an independent flow of control. Active class share the same properties as all the other classes. When an active object is created, the associated flow of control is started. When the object is destroyed the associated flow of control is terminated.  active class KeypadManager signals Signals pushButton(b : button) powerUp powerDown	C303.4	**BTL1**
20	**Justify why class diagram is called static object modelling.** The UML class diagram does not have any dynamic elements and all the representations are static. The classes and the methods in the classes do not change with respect to any external criteria. The characteristics and the methods of a class are standard and constant and cannot be changed. Therefore, the class diagram is called as static object modelling	C303.4	**BTL5**
21	**List the relationships used in class diagram.** The various relationships used in class diagrams are: • Association with multiplicities. • Interface implementation • Inheritance • Dependency • Composition over Aggregation • Qualified Association • Qualified association • Association Class	C303.4	**BTL1**
22	**Define singleton class with an example.** When exactly one instance of a class is allowed, it is called a “singleton” class. In UML diagram, such a class can be marked with a “I” in the upper right corner of the name component.	C303.4	**BTL1**

23	**What is Logical Architecture?** The logical architecture is the large scale organization of the software classes into packages, namespaces, subsystems and layers. It's called the logical architecture because there's no decision about how these elements are deployed across different operating system processes or across physical computers in a network. An architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.	C303.4	**BTL1**
24	**What are the types in layered architecture? List the layers in OO system.** • Strict layered architecture • Relaxed layered architecture • Layers in OO architecture: • User Interface • Application Logic and Domain Objects • Technical Services	C303.4	**BTL1**
25	**List the benefits of using layers?** • Relaxed complexity is encapsulated and decomposable. • Lower layers contain reusable functions • Some layers (primarily the domain and technical services) can be distributed. • Development by team is aided because of the logical segmentation • Some layers can be replaced with new implementations.	C303.4	**BTL1**
26	**What is the Relationship between domain layer and domain model?** The domain layer is part of the software and the domain model is part of the conceptual perspective analysis. By creating a domain layer with inspiration from the domain model, a lower representation gap between the real world domain and the software design is achieved.	C303.4	**BTL1**
27	**Define tiers, layers and partition?** Tier in architecture is a logical layer, not a physical node. The layers of architecture are said to represent the vertical slices, while partitions represent a horizontal division of relatively parallel subsystems of a layer. **Ex: The technical services layer may be divided into partitions such as security and reporting.**	C303.4	**BTL1**
28	**Define model view separation principle?** The model view separation principle states that model objects should not have direct knowledge of view objects, at least as view objects. Ex: a register or sale object should not directly send a message to a GUI window object process Sale Frame, asking it to display something.	C303.4	**BTL1**

29	**Difference between Logical architecture and layers [MAY/JUN 2017]** The ***logical architecture***is the large - scale organization of the software classes into packages (or namespaces), subsystems, and layers. It's called the *logical* architecture because there's no decision about how these elements are deployed across different operating system processes or across physical computers in a network (these latter decisions are part of the ***deployment architecture***). A layer is a very coarse - grained grouping of classes, packages, or subsystems that has cohesive responsibility for a major aspect of the system. Also, layers are organized such that "higher" layers (such as the UI layer) call upon services of "lower" layers, but not normally vice versa.	C303.4	**BTL1**
30	**When to use package diagram and collaboration diagram[APR/MAY 2018]** A **package** in the Unified Modeling Language is used "to group elements, and to provide a namespace for the grouped elements". A **package** may contain other **packages**, thus providing for a hierarchical organization of **packages**. Pretty much all **UML** elements can be grouped into **packages**. A collaboration diagram, also called a **communication** diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the **Unified Modeling Language** (UML).	C303.4	**BTL1**
31	**How to create an instance[APR/MAY 2018]** Object instances can only be created by using a class definition. Even though an object instance is created, it has not been committed to the database.	C303.4	**BTL1**

PART-B

S.NO	**QUESTIONS**	**CO**	**BLOOM'S LEVEL**
1	Illustrate with an example, the relationship between UML Sequence diagram and use cases?**[APR/MAY 2011, MAY/JUNE 2013, NOV/DEC 2013,MAY/JUN 2014, NOV/DEC 2016, APR/MAY 2018]**	C303.4	**BTL2**
2	Explain the logical architecture and UML package diagram**. [MAY/JUN 2014, NOV/DEC 2016]**	**C303.4**	**BTL2**
3	What are concepts involved in logical architecture refinement**?**	**C303.4**	**BTL1**
4	Explain the UML notation for class diagram with example& Explain the concepts of link, association and inheritance**? [MAY/JUNE 2012, MAY/JUNE 2013 , MAY/JUNE 2016]**	**C303.4**	**BTL2**

5	Explain about Interaction Diagram Notation for inventory Management system? (OR). **[APR/MAY 2011, NOV/DEC 2011, NOV/DEC 2013, APR/MAY 2011, MAY/JUNE 2013, NOV/DEC 2013, MAY/JUN 2014, NOV/DEC 2015]**	**C303.4**	**BTL2**
6	How to add new SSD's and contract to the design diagrams **NOV/DEC 2015]**	**C303.4**	**BTL1**
7	What are the concepts involved in domain Refinement **NOV/DEC 2015]**	**C303.4**	**BTL2**
8	What is Model-view-separation principle [**MAY/JUNE 2016, APR/MAY 2017]**	**C303.4**	**BTL2**
9	Explain the UML class,Sequence and Interaction diagram for Library Management System **APR/MAY 2017]**	**C303.4**	**BTL2**
10	Model a class diagram for a Banking system .State the functional requirements you consider[**NOV/DEC 2017]**	**C303.4**	**BTL2**
11	Explain detail about various static and dynamic with UML important diagrams with suitable example **[APR/MAY 2018]**	**C303.4**	**BTL2**

UNIT V

CODING AND TESTING

Mapping design to code – Testing: Issues in OO Testing – Class Testing – OO Integration Testing – GUI Testing – OO System Testing.

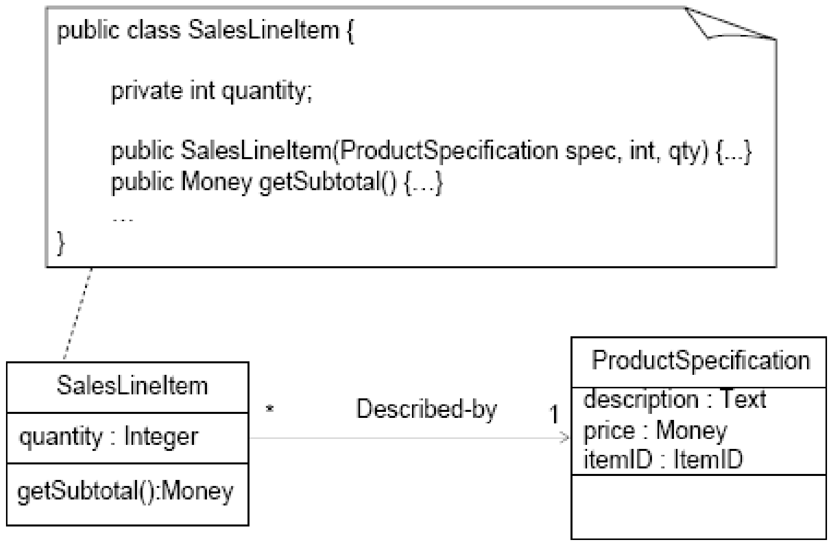
PART- A

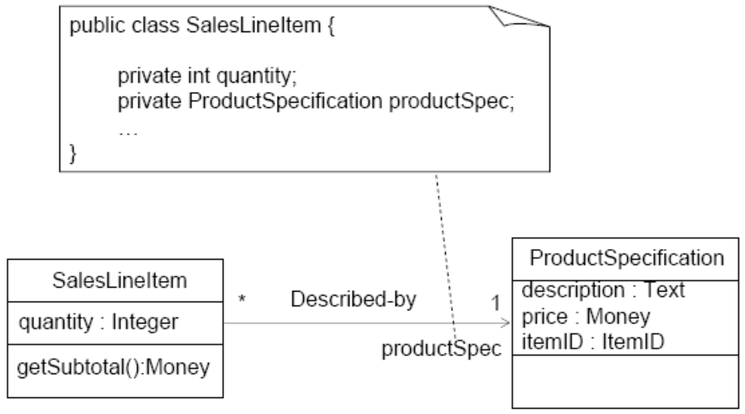
S.NO	**QUESTIONS**	**CO**	**BLOOM'S LEVEL**
1	**What are Steps for Mapping Designs to Code? [MAY/JUNE 2015,MAY/JUNE 2015,MAY/JUNE 2016, MAY/JUNE 2017]** Implementation in an object-oriented programming language requires writing source code for: • Class and interface definitions	C303.5	**BTL1**

	• Method definitions		
2	**When are Contracts useful? [<u>APR/MAY 2011, MAY/JUNE 2014]</u>** • Operation contract describes the behavior in terms of state changes to the objects in the domain model when a system operation gets executed. The domain model is a visual representation of the conceptual classes or the real world objects. • It is used to represent the system behavior. • It uses pre and post condition form to describe the changes in the objects. • The use cases are the main repository of requirements for the project. They may provide most or all of the detail necessary to know what to do in the design, in which case, contracts are not helpful. However, there are situations where the details and complexity of required state changes are awkward to capture in use cases.	C303.5	**BTL1**
3	**What are the issues in OO testing? <u>[MAY/JUNE 2015,NOV/DEC 2015</u>]** • Testing in an OO context must address the basics of testing a base class and the code that uses the base class. Factors that affect this testing are inheritance and dynamic binding. • Therefore, some systems are harder to test (e.g., systems with inheritance of implementations harder than inheritance of interfaces) and OO constructs such as inheritance and dynamic binding may serve to hide faults.	C303.5	**BTL1**
4	**What is OO integration Testing? <u>[MAY/JUNE 2016,</u> MAY/JUNE 2017]** Integration testing is the phase in which the individual Units are combined to form larger and larger aspects of the program, they are tested to determine If the units interact together correctly, for example to check that a Unit is returning the result of a calculation in the correct format. Integration testing requires that the Unit testing phase has been completed successfully. Integration testing will take up much of the whole testing phase and one of its biggest problems is determining exactly how long to spend since this phase could potentially, if you wanted to exhaustively test the program, take a very long time.	C303.5	**BTL1**
5	**What is GUI testing?** GUI testing is the process of ensuring proper functionality of the graphical user interface (GUI) for a given application and making sure it conforms to its written specifications. GUI testing evaluates design elements such as layout, colors,fonts, font sizes, labels, text boxes, text formatting, captions, buttons, lists, icons, links and content. GUI testing processes can be either manual or automatic, and are often performed by third -party companies, rather than developers or end users.	C303.5	**BTL1**

6	**List out the Pros and Cons of Top-down Integration Testing:** **Pro** • Test cases can be defined in terms of the functionality of the system (functional requirements) • No drivers needed **Cons** • Writing stubs is difficult: Stubs must allow all possible conditions to be tested. • Large number of stubs may be required, especially if the lowest level of the system contains many methods. • Some interfaces are not tested separately.	C303.5	**BTL1**
7	**What is Sandwich Testing Strategy:** • Combines top-down strategy with bottom-up strategy • The system is viewed as having three layers • A target layer in the middle • A layer above the target • A layer below the target • Testing converges at the target layer.	C303.5	**BTL1**
8	**What are the Pros and Cons of Sandwich Testing:** • Top and Bottom Layer Tests can be done in parallel • Problem: Does not test the individual subsystems and their interfaces thoroughly before integration • Solution: Modified sandwich testing strategy	C303.5	**BTL1**
9	**What are the Steps in Integration Testing:** • 1. Based on the integration strategy, *select a component* to be tested. Unit test all the classes in the component. • 2. Put selected component together; do any *preliminary fix-up* necessary to make the integration test operational (drivers, stubs) • 3. Test functional requirements*:* Define test cases that exercise all uses cases with the selected component • 4. Test subsystem decomposition*:* Define test cases that exercise all dependencies • 5. Test non-functional requirements: Execute *performance tests*	C303.5	**BTL1**

	• 6. *Keep records* of the test cases and testing activities. • 7. Repeat steps 1 to 7 until the full system is tested. • The primary *goal of integration testing is to identify failures* with the (current) component *configuration*.		
10	**What are the Approaches of GUI Testing?** • Manual Based Testing • Record and Replay • Model Based Testing	C303.5	**BTL1**
11	**What is Functional Testing?** Goal: Test functionality of system • Test cases are designed from the requirements analysis document (better: user manual) and centered around requirements and key functions (use cases) • The system is treated as black box • Unit test cases can be reused, but new test cases have to be developed as well.	C303.5	**BTL1**
12 **13**	**What are the types of Performance Testing?** Compatibility test,Volume testing,Configuration testing,Stress Testing,Compatibility test Volume testing,Configuration testing,,Stress Testing,Timing testing,Security testing Human factors testing,,Quality testing,Recovery testing,Environmental test **What is Acceptance Testing?** • Goal: Demonstrate system is ready for operational use • Choice of tests is made by client • Many tests can be taken from integration testing • Acceptance test is performed by the client, not by the developer. • Alpha test: • Client uses the software at the developer's environment. • Software used in a controlled setting, with the developer always ready to fix bugs. • Beta test:	C303.5	**BTL1**

	• Conducted at client's environment (developer is not present) • Software gets a realistic workout in target environment		
14	**what are difference between alpha test and beta test?** • Alpha test: • Client uses the software at the developer's environment. • Software used in a controlled setting, with the developer always ready to fix bugs. • Beta test: • Conducted at client's environment (developer is not present) • Software gets a realistic workout in target environment	C303.5	**BTL1**
15	**What is Design Class Diagrams:** ● DCDs contain class or interface names, classes, method and simple attributes. ● These are sufficient for basic class definitions. ● Elaborate from associations to add reference attributes	C303.5	**BTL1**
16	**What is Reference Attributes:** An attribute that refers to another complex objects. ● Reference Attributes are suggested by associations and navigability in a class diagram. ● **Example:** A product specification reference on a Sales Line Item. So here we can use product spec as a complex reference attribute to sales line item class.  `public class SalesLineItem { private int quantity; public SalesLineItem(ProductSpecification spec, int, qty) {...} public Money getSubtotal() {…} … }` SalesLineItem quantity : Integer getSubtotal():Money * Described-by 1 ProductSpecification description : Text price : Money itemID : ItemID	C303.5	**BTL1**

17	**What is Role Names:**
	
	● Each end of an association is a role. Reference Attributes are often suggested by role names. (use role names as the names of reference attributes).
	
	 public class SalesLineItem {
	private int quantity;
	private ProductSpecification productSpec;
	…
	}
	SalesLineItem
	quantity : Integer
	getSubtotal():Money
	*
	Described-by
	1
	productSpec
	ProductSpecification
	description : Text
	price : Money
	itemID : ItemID
		C303.5	**BTL1**

18	**Define testing.** Testing is the process of using suitable test cases to evaluate and ensure the quality of a product by removing or sorting out the errors and discrepancies. It is also used to ensure that the product has not regressed (such as, breaking a feature that previously worked).Testing involves various types and levels based on the type of object/product under test. Testing can be described as a process used for revealing defects in software, and for establishing that the software has attained a specified degree of quality with respect to selected attributes.	C303.5	**BTL1**
19	**What is test driven development?** Unit testing code is written before the code to be tested, and the developer writes unit testing code for all production code. Ie, the basic method is to write a test code, then write a little production code, make it pass the test, then write some more test code and so forth.	C303.5	**BTL1**
20	**What is refactoring? [NOV/DEC 2016]** Refactoring is a structured, disciplined method to rewrite or restructure existing code without changing its external behaviour, applying small transformation steps combined with re-executing tests at each step. Refactoring is another extreme programming (XP) step applicable to all iterative methods.	C303.5	**BTL1**
21	**What is the need for testing a code?** The programmers and the testers have to execute the program before it gets to the customer with the specific intent of removing all errors, so that the customer will not experience the frustration associated with a poor-quality product. In order to find the highest possible number of errors, tests must be	C303.5	**BTL1**

22	**What is random class testing?** In random class testing, the classes or the methods of a class can be tested using random test cases in a random sequence. The test process need not follow a procedure or a finite set of test cases for the methods of a class.	C303.5	**BTL1**
23	**What is a test harness?** A test harness is an environment into which a software component can be placed a tested using test cases. If a class under test does not interact with any other classes, th the test harness consist of a main program and the class under test. If the class intera with other classes then the test harness consists of the main program, the class un test and dummy class to replace the other classes.	C303.5	**BTL1**
24	**Compare system testing and integration testing. <u>[MAY/JUNE 2016]</u>** In integration testing, the product is divided into smaller subsystems and tested separately. They may be combined with other subsystems and tested. In system testing the test data and the classes are combined as one and tested as a whole to find out how the entire system works in the test or launch environment.	C303.5	**BTL5**
25	**What are test cases? When we say test case is effective?** The usual approach to detecting defects in a piece of software is for the tester to select a set of input data and then execute the software with the input data under a particular set of conditions. The tester bundles this information into an item called test case. • A greater probability of detecting defects • A more efficient use of organizational resources • A higher probability for test reuse • Closer adherence to testing and project schedules and budgets • The possibility for delivery of a higher-quality software product	C303.5	**BTL1**
26	**What is validation and verification?** Validation is the process of evaluating a software system or component during, or at the end of the development cycle in order to determine whether it satisfies specified requirements. Validation is usually associated with traditional execution based testing, that is exercising the code with test cases.	C303.5	
27	**How is class testing different from conventional testing?** Conventional testing focuses on input-process-output, whereas class testing focuses on each method, then designing sequences of methods to exercise states of a class. In conventional testing methods the test cases are applied and the output is focused on ie, enter the input and wait for the valid and correct output. If there is a false output	C303.5	

	then it signifies the occurrence of an error.In class testing the methods of the classes under test is subjected to various test cases in a suitable test environment.		
28	**Explain about thread based, cluster based, and use based testing in Integration testing.** Thread-based testing – testing of all classes which are required to respond to one system input or event Use-based testing – in this the independent classes are tested first and the dependent classes are tested later. Cluster testing - groups of collaborating classes are tested for interaction errors	C303.5	
29	**Why do conventional top down and bottom up integration testing methods have less meaning in an object oriented context?** Basic object oriented software does not have a hierarchical control structure, hence top down approach and bottom up approach of testing is of less use in testing. Therefore, thread based, use based and cluster based testing methods are incorporated for performing integration testing.	C303.5	
30	**What is the test case design for object oriented software?** White-box testing methods can be applied to testing the code used to implement class operations. In this the code and the methods used in the algorithms can be tested. Black-box testing methods are appropriate for testing OO systems. In black box testing only the interface and the structure of the code can be tested.	C303.5	**BTL1**
31	**When does testing of a product/scenario end?** Practically, testing is a process that never ends. This can be expressed in three ways. • The burden of ensuring quality to a product simply shifts from the developer to the tester and then to the customer. • Testing is done when you run out of time or money. • Use a statistical model: Assume that errors decay logarithmically with testing time Measure the number of errors in a unit period Fit these measurements to a logarithmic curve .	C303.5	**BTL1**
32	**What is Regression Testing? .[NOV/DEC 2016]** **Regression testing** is a type of software **testing** which verifies that software, which was previously developed and **tested**, still performs correctly after it was changed or interfaced with other software. Changes may include software enhancements, patches, configuration changes, etc	C303.5	**BTL1**
33	**What is Refactoring and Testing[APR/MAY 2018]** Refactoring is a structured, disciplined method to rewrite or restructure existing code	C303.5	**BTL1**

	without changing its external behaviour, applying small transformation steps combined with re-executing tests at each step. Refactoring is another extreme programming (XP) step applicable to all iterative methods. Testing is the process of using suitable test cases to evaluate and ensure the quality of a product by removing or sorting out the errors and discrepancies. It is also used to ensure that the product has not regressed (such as, breaking a feature that previously worked).Testing involves various types and levels based on the type of object/product under test. Testing can be described as a process used for revealing defects in software, and for establishing that the software has attained a specified degree of quality with respect to selected attributes		
34	**How to use creating methods from interaction diagram [APR/MAY 2018]** In interaction diagram shows the messages that are sent in response to a method invocation. The sequence of these messages translates to a series of statements in the method definition.	C303.5	

`

PART- B

S.NO	QUESTIONS	CO	BLOOM'S LEVEL
1	Explain in detail the design,artifacts to implementation code in an object oriented language [**MAY/JUNE 2016]**	**C303.5**	
1	Explain the operation of Mapping Designs to Code**[APR/MAY 2011, NOV/DEC 2013 ,NOV/DEC 2105,NOV/DEC2016]**	**C303.5**	**BTL2**
2	List out the issues in object oriented Testing(OO Testing) [**APR/MAY 2017, APR/MAY 2018]**	**C303.5**	**BTL1**
3	Explain about Class Testing?	**C303.5**	**BTL2**
4	What is OO Testing?Explain in detail about the concepts of OO testing in OOAD [**MAY/JUNE 2016]**	**C303.5**	**BTL2**
5	Explain on GUI testing? **[NOV/DEC 2016, APR/MAY 2017, APR/MAY 2018]**	**C303.5**	**BTL2**
6	Discuss in detail about OO Integration testing and System testing?[**NOV/DEC 2016, APR/MAY 2017, NOV/DEC 2017, APR/MAY 2018]**	**C303.5**	**BTL1**
7	Explain in detail about the different types of testing strategies in OOAD	**C303.5**	**BTL2**

	[**MAY/JUNE 2016, APR/MAY 2018]**		
8	How is class testing different from converntional testing .Explain with an example[**NOV/DEC 2017]**	**C303.5**	**BTL2**