CMR INSTITUTE OF TECHNOLOGY

USN	1	C								

Internal Assessment Test 1 – September 2019- Answer Key

Sub:	Object Oriented Modeling and Design Patterns							Code:	17MCA51
Date:	21-09-19	Duration:	90 mins	Max Marks:	50	Sem:	V A & B	Branch:	MCA

Total Marks: 50

		Marks	OBE CO	OBE RBT
	PART-1			
1.	**What is pattern? Write the categories of pattern in detail.**	**10**	CO5	L2

Pattern is a kind of an expert behavior. Experts works on a particular problem every time will not get a distinct solution, Recall a similar problem, and reuse the essence of its solution to solve it.

Each pattern is a 3 – part rule:
A relation between a certain context
A problem
A solution

Each pattern is a relationship a context, a system of forces which occurs repeatedly in that context.

Each pattern is an instruction which helps for re-usability, a relevant context and to resolve systems of forces.

Each pattern is both a process and thing.

Pattern categories :
Architectural
Design Patterns
Idioms

Architectural Patterns: – Expresses a fundamental structural organizational schema for software system. It provides a set of pre-defined subsystem specify their responsibilities and includes rules and guidelines for organizing the relationship between them.

Design Pattern: A design pattern provides schema for refining the subsystems/components of a software system or relationship between them.

Idioms: An idiom is a low level pattern specific to a programming language. Idioms deals with the implementation of particular design pattern

(OR)

2	**Describe the pattern description template in detail.**	**10**	CO5	L2

Name: The name and a short summary of the patterns.
Also Known as: Other names for the patterns
Example: A real-world example demonstrating the existence of the problem.
Context: a situation in which the pattern may apply.
Problem: The problem the pattern addresses, including a discussion

Solution: The fundamental solutions principles underlying the pattern.
Structure: A detailed specification of the structural aspects of the pattern.

PART-II **10** CO5 L3

3. **Explain the Client Dispatcher Server design pattern in detail**

Provides transparent inter-process communication for software systems in which the distribution of components is not known at compile time i.e may vary dynamically at run-time. This pattern allows peer to migrate to other locations at run-time by unregistering and re-registering with the dispatcher.

This pattern provides an intermediate layer between clients and servers called as dispatcher. The pattern provides location transparency by means of a name service. The pattern hides the details of the establishment of the communication connection between clients and servers.

Dispatcher implements a name service to allow clients to refer the servers by names instead of physical location. Dispatcher is responsible for establishing a communication channel between a client and a server.

Each server is uniquely identified by name and is connected to client by dispatcher. Clients rely on dispatcher to locate a particular server and connect. The roles of the server and client can change dynamically.

Structure

1> Before sending a request to a server, the client gets information from dispatcher for communication channel.

2> Servers provide set of operations to clients.

3> Server registers itself OR is registered with the dispatcher by its name and address.

4> The server component can be located on same computer as a client or may be reachable via a network.

5> The dispatcher establishes a communication link to server using available communication mechanism and returns a communication handle to the client.

6> Dispatcher implements registering of servers.

(OR)

4. **Briefly write about the structure and dynamics of Forwarder Receiver design Pattern** **10** CO5 L3

Peer pattern

1) Continuously monitor network events and resources.
2) Listen for incoming messages from remote agents.
3) Each agent may connect to any other agent to exchange information and requests.
4) The network management infrastructure connects the network

administrator's console with all other agents.

5) Administrators may send requests to network agents or retrieve messages from them by using available network administration tools.

Forwarder pattern

1) Sends messages across process boundaries
2) Provides a general interface that is an abstraction of a particular IPC mechanism.
3) Includes functionality for marshaling and delivery of messages.
4) Contains a mapping from names o physical addresses.
5) Determines the physical location of the recipient by using its name-to-addresses mapping.
6) In the transmitted message, he forwarder specifies its own peer, so that remote peer is able to send a response to the message originator.

Receiver pattern

1) Wait for incoming messages on behalf of their agent process.
2) As soon as the message arrives, they convert the received data stream into a general message format and forward the message to their agent process.

Class:	*Collaborators*
peer	*Forwarder*
Responsibility	*Receiver*
Provides application services.	
Communicates with other peers.	

PART-III **10** CO1 L3

5. **What are the various stages of Object Oriented Methodology in software development?**

The major phases of software development using object–oriented methodology are object-oriented analysis, object-oriented design, and object-oriented implementation.

Object–Oriented Analysis

In this stage, the problem is formulated, user requirements are identified, and then a model is built based upon real–world objects. The analysis produces models on how the desired system should function and how it must be developed. The models do not include any implementation details so that it can be understood and

examined by any non–technical application expert.

Object–Oriented Design

Object-oriented design includes two main stages, namely, system design and object design.

System Design

In this stage, the complete architecture of the desired system is designed. The system is conceived as a set of interacting subsystems that in turn is composed of a hierarchy of interacting objects, grouped into classes. System design is done according to both the system analysis model and the proposed system architecture. Here, the emphasis is on the objects comprising the system rather than the processes in the system.

Object Design

In this phase, a design model is developed based on both the models developed in the system analysis phase and the architecture designed in the system design phase. All the classes required are identified. The designer decides whether −

- new classes are to be created from scratch,
- any existing classes can be used in their original form, or
- new classes should be inherited from the existing classes.

The associations between the identified classes are established and the hierarchies of classes are identified. Besides, the developer designs the internal details of the classes and their associations, i.e., the data structure for each attribute and the algorithms for the operations.

Object–Oriented Implementation and Testing

In this stage, the design model developed in the object design is translated into code in an appropriate programming language or software tool. The databases are created and the specific hardware requirements are ascertained. Once the code is in shape, it is tested using specialized techniques to identify and remove the errors in the code.

6 a **Describe the object oriented themes.** **5** CO1 L2

- Abstraction

  Abstraction consists of focusing on the essential, inherent aspects of an entity and ignoring its accidental aspects.

- Classes and instances

  Objects are instances of classes. A class is a definitive Description of a group of objects with similar properties and behaviors. Classes are abstract, objects are concrete. Objects are aware of their class identity.

- Encapsulation

  Encapsulation (also information hiding) consists of separating the external aspects of an object, which are accessible to other objects, from the internal details of the object, which are hidden from other objects.

  Objects have an outside (how they are seen or interact) and an inside (what they are.)

- Polymorphism

  Polymorphism is the kindred to the incorporation of data and behavior. From the outside, the same operation Solve or Draw applies to various objects (solvable or, respectively, drawable). The object user is not burdened with the implementation - the inside - of the object. The object knows how to do the job.

- Inheritance

  Inheritance is a mechanism for sharing similarities among classes while preserving their differences.

b **Write short notes on the various modeling techniques.** **5** CO1 L2

The different types of modeling techniques are:
i) **Class Model**: It describes the structure of objects in a system – their identity, their relationships to other objects, their attributes and their operations. The goal of constructing the class model is to capture those concepts from the real world that are important to an application. Class diagram express the class model.
ii) **State Model**: It describes those aspects of objects concerned with time and the sequencing of operations – events that mark changes, state that define the context for events, and the organization of events and states. State diagram expresses the state model.
iii) **Interaction Model**: It describes interactions between objects – How individual objects collaborate to achieve the behavior of the system as a whole. Use case, sequence diagram and activity diagram documents the interaction model.

PART-IV **10** CO1 L2

7 **Describe the following terms with suitable example**

Ordering
Multiplicity
Generalization
Bags & Sequence

- Ordinary a binary association has at most one link for a pair of objects
- However we can permit multiple links for a pair of objects by annotating an association end with {bag} or {sequence}
- A **bag** is a collection of elements with duplicates allowed.
- A **sequence** is an ordered collection of elements with duplicates allowed

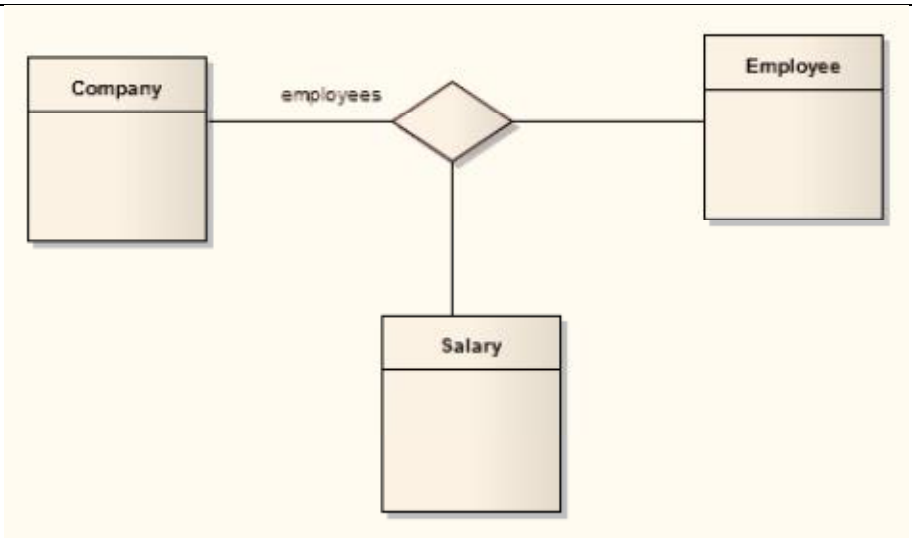
- It specifies the number of instances of one class that may relate to a single instance of the associated class.
- UML diagrams explicitly list multiplicity at the end of association lines.
- Intervals are used to express multiplicity:
  - 1 (exactly one)
  - 0..1 (zero or one)
  - 1..* (one or more)
  - 0..* (zero or more)
  - 3..5 (three to five inclusive)

- Generalization is the relationship between a class (**superclass**) and one or more **variations** of the class (**subclasses**).
- Generalization organizes classes by their **similarities** and their **differences, structuring** the descriptions of objects.
- A superclass holds **common** attributes, attributes and associations.
- The subclasses **adds specific** attributes, operations, and associations. They i**nherit** the features of their superclass.
- Often **Generalization** is called a "**IS A"** relationship
- **Simple generalization** organizes classes into a **hierarchy**.
- A subclass may **override** a superclass **feature (**attribute default values, operation) by **redefining a feature with the same name**.
- Never override the signature of methods.

(OR)

8 **Explain the links and association with example** **10** CO1 L2

Link is an instance of an association. It is a tuple with one value for the each end of the association, where each value is an instance of the type of the end.
Association is reference based relationship between two classes. Here a class A holds a class level reference to class B. Association can be represented by a line between these classes with an arrow indicating the navigation direction. In case arrow is on the both sides, association has bidirectional navigation.



PART-V

			10 CO5 L4

9 **Explain the structure and dynamics of publisher – subscriber design pattern**

Push-model	Pull model
Publisher sends all changed data when it notifies the subscriber.	Publisher only sends the minima information when sending a change notification.
The subscribers have no choice about if and hen they want to retrieve the data.	Subscribers are responsible for retrieving the data they need.
Very rigid dynamic behavior. Model is not suitable for complex data changes.	Offers more flexibility by more messages between publisher and subscribers.
Even pushing a package description is a overhead.	Data changes are found. The process of finding data changes i organized as a decision tree.
This model is best suited when subscribers need the published information most of the time.	The model is used when only the individual subscribers can decid if and when they need a specific piece of inf rmati n.

(OR)

10 **Demonstrate Whole-Part design pattern in detail** **10** CO5 L4

The Whole-Part design pattern helps with the aggregation of components that together form a semantic unit. to the Parts is not An aggregate component, the

Whole, encapsulates it constituent components, the Parts, organizes their collaboration, and provides a common interface to its functionality. Direct access possible.

- Sometimes called Composite
- Helps with the aggregation of components (parts) that together form a semantic unit (whole).
- Direct access to the Parts is not possible
- Compose objects into tree structures to represent part-whole hierarchies.

- Whole-Part lets clients treat individual objects and compositions of object uniformly

- Supported in OO Programming Language

- Whole-Part is part of why OOP design the way it is

- Remember to read the text book

Example

- Graphics applications, like drawing editors, let users build complex diagrams out of simple components. For example, a graph might contain lines, rectangles, texts, pictures

- The user can group components to form larger components, which in turn can be grouped to form still larger components

- A simple implementation could define classes for graphical primitives such as Text and Lines plus other classes that act as containers for these primitives

- Computer-Aided Design (CAD)

- Java Beans

- DevPack

- GL, OpenGL ?

Context

- Implementing aggregate objects

Problem

- A complex object should either be decomposed into smaller objects or composed of existing objects, to support reusability, changeability and the recombination of the constituent objects in other types of aggregate
- Clients should see the aggregate object as an atomic object that does not allow any direct access to its constituent parts
- Code that uses these classes must treat primitive (Text, Lines, etc.) and container (Graphic) objects differently
- Having to distinguish these objects makes the application more complex.

Solution

- Use a component that encapsulates smaller objects, and prevents clients from accessing these constituent parts directly
- Use an interface as the only means of access to the functionality of the encapsulated objects (appear as a semantic unit)
- An assembly-parts relationship differentiates between a product and its parts or subassemblies
- A container-contents relationship
- A collection-members relationship helps to group similar objects - such as an organization and its members

Structure

- Two types of participant
- Two CRC cards

Class: Whole

Responsibility:

- Aggregates several smaller objects
- Provides services built on top of part objects
- Acts as a wrapper around its constituent parts

Collaborators: Part

Class: Part

Responsibility:

- Pepresents a particular object and its services

Collaborators: -