
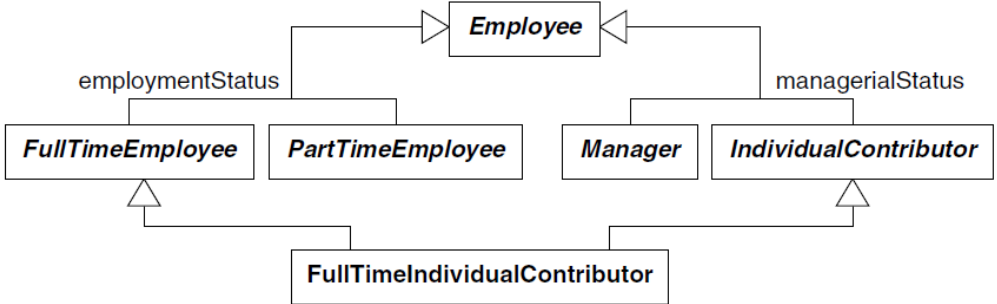


CMR INSTITUTE OF TECHNOLOGY		USN								
Internal Assessment Test - I										
Sub:	Object Oriented Modeling And Design						Code:	18MCA43		
Date:	07-04-2021	Duration:	90 mins	Max Marks:	50	Sem:	IV	Branch:	MCA	
Answer ONE FULL QUESTION from each part										
								Marks	OBE	
									CO	RBT
Part - I										
1	<p>What is Generalization and multiple inheritance? Explain different kinds of multiple inheritances in advance class modeling concept, with example.</p> <p>Generalization is the process of extracting shared characteristics from two or more classes, and combining them into a generalized super class. Shared characteristics can be attributes, associations, or methods.</p> <p>Multiple inheritance Permits a class to have more than one superclass and to inherit features from all parents. Then you can mix information from two or more sources.</p> <p>Kinds of Multiple Inheritance The most common form of multiple inheritance is from sets of disjoint classes. Each subclass inherits from one class in each set.</p>  <p style="text-align: center;">Figure 4.15 Multiple inheritance from disjoint classes. This is the most common form of multiple inheritance.</p> <p>Multiple classification An instance of a class is inherently an instance of all ancestors of the class. For example, an instructor could be both faculty and student. But what about a Harvard Professor taking classes at MIT? There is no class to describe the combination. This is an example of multiple classification, in which one instance happens to participate in two overlapping classes.</p>						10	CO1	L2	

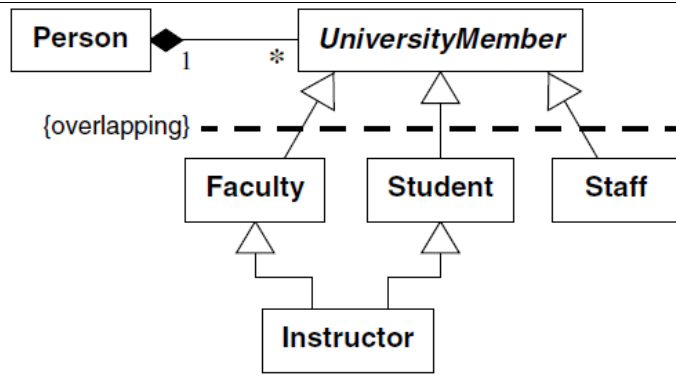


Figure 4.17 Workaround for multiple classification. OO languages do not handle this well, so you must use a workaround.

(OR)

2 Explain in detail about following topics with UML notation in advanced class modelling
 a.) Enumeration b.) Multiplicity (attributes) c.) Scope d.) visibility

10

CO1

L2

- a. **Enumeration:** An **enumeration** is a data type that has a finite set of values. Figure 4.1 illustrates. Figure.pen-Type is an enumeration that includes solid, dashed, and dotted.
- TwoDimensional.fillType is an enumeration that includes solid, grey, none, horizontal lines, and vertical lines.

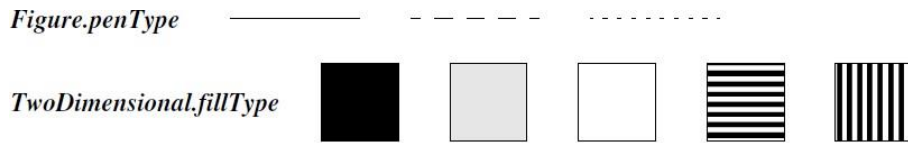


Figure 4.1 Examples of enumerations. Enumerations often occur and are important to users. Implementations must enforce the finite set of values.

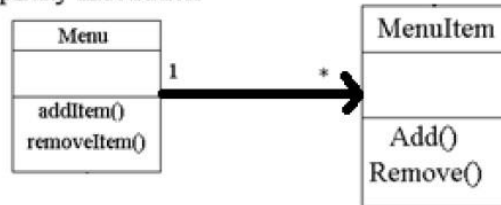
In the UML an enumeration is a data type. You can declare an enumeration by listing the keyword *enumeration* in guillemets (*<<>*) above the enumeration name in the top section of a box. The second section lists the enumeration values.

- (a) The UML representation of an association is a line connecting the two associated classes. At each end of the line there is optional notation. For example, we can indicate, using an arrowhead that the pointy end is visible from the arrow tail. We can indicate ownership by the placement of a ball, the role the elements of that end play by supplying a name for the role, and the
- (b) *multiplicity* of instances of that entity (the range of number of objects that participate in the association from the perspective of the other end).

0	No instances (rare)
0..1	No instances, or one instance
1	Exactly one instance
0..*	Zero or more instances
*	Zero or more instances
1..*	One or more instances

- c) **Scope:** The *scope* indicates if a feature applies to an object or a class. An underline distinguishes features with class scope (static) from those with object scope. Our convention is to list attributes and operations with class scope at the top of the attribute and operation boxes, respectively.

Multiplicity Illustration



It is acceptable to use an attribute with class scope to hold the *extent* of a class (the set of objects for a class)—this is common with OO databases. Otherwise, you should avoid

attributes with class scope because they can lead to an inferior model. It is better to model groups explicitly and assign attributes to them.

Figure 4.4 shows a simple model of phone mail. Each message has an owner mailbox, date recorded, time recorded, priority, message contents, and a flag indicating if it has been received. A message may have a mailbox as the source or it may be from an external call. Each mailbox has a phone number, password, and recorded greeting. For the *PhoneMessage* class we can store the maximum duration for a message and the maximum days a message will be retained. For the *PhoneMailbox* class we can store the maximum number of messages that can be stored.

In contrast to attributes, it is acceptable to define operations of class scope. The most common use of class-scoped operations is to create new instances of a class. Sometimes it is convenient to define class-scoped operations to provide summary data. You should be careful with the use of class-scoped operations for distributed applications.

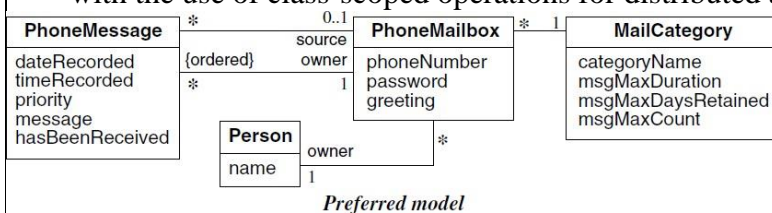


Figure 4.4 Attribute scope. Instead of assigning attributes to classes, model groups explicitly.

- d) **Visibility:** *Visibility* refers to the ability of a method to reference a feature from another class and has the possible values of *public*, *protected*, *private*, and *package*. Any method can freely access *public* features. Only methods of the containing class and its descendants via inheritance can access *protected* features. (Protected features also have package accessibility in Java.) Only methods of the containing class can access *private* features. Methods of classes defined in

the same package as the target class can access *package* features.

The UML denotes visibility with a prefix. The character “+” precedes public features. The character “#” precedes protected features. The

	character “-” precedes private features. And the character “~” precedes package features.			
Part – II				
3	<p>Describe the stages of Object oriented methodology, used in software development</p> <ul style="list-style-type: none"> ■ System conception. Software development begins with business analysts or users conceiving an application and formulating tentative requirements. ■ Analysis. The analyst must work with the requestor to understand the problem, because problem statements are rarely complete or correct. The analysis model is a concise, precise abstraction of <i>what</i> the desired system must do, not <i>how</i> it will be done. The analysis model should not contain implementation decisions. For example, a <i>Window</i> class in a workstation windowing system would be described in terms of its visible attributes and operations. <p>The analysis model has two parts: the domain model, a description of the real-world objects reflected within the system; and the application model, a description of the parts of the application system itself that are visible to the user. For example, domain objects for a stockbroker application might include stock, bond, trade, and commission. Application objects might control the execution of trades and present the results. Application</p> <p>experts who are not programmers can understand and criticize a good model.</p> <ul style="list-style-type: none"> ■ System design. The development team devise a high-level strategy—the system architecture—for solving the application problem. They also establish policies that will serve as a default for the subsequent, more detailed portions of design. The system designer must decide what performance characteristics to optimize, choose a strategy of attacking the problem, and make tentative resource allocations. For example, the system designer might decide that changes to the workstation screen must be fast and smooth, <p>even when windows are moved or erased, and choose an appropriate communications protocol and memory buffering strategy.</p> <ul style="list-style-type: none"> ■ Class design. The class designer adds details to the analysis model in accordance with the system design strategy. The class designer elaborates both domain and application objects using the same OO concepts and notation, although they exist on different conceptual planes. The focus of class design is the data structures and algorithms needed to implement each class. For example, the class designer now determines data structures and algorithms for each of the operations of the <i>Window</i> class. ■ Implementation. Implementers translate the classes and relationships developed during class design into a particular programming language, database, or hardware. Programming should be straightforward, because all of the hard decisions should have already 	10	CO1	L1

	<p>been made. During implementation, it is important to follow good software engineering practice so that traceability to the design is apparent and so that the system remains flexible and extensible. For example, implementers would code the <i>Window</i> class in a programming language, using calls to the underlying graphics system on the workstation. OO concepts apply throughout the system development life cycle, from analysis through design to implementation.</p>			
--	--	--	--	--

(OR)

4	<p>Explain in detail about class modelling with suitable example</p> <p>A class model captures the static structure of a system by characterizing the objects in the system, the relationships between the objects and the attributes and operations for each class of objects.</p> <p>Class model provides a graphical representation of a system and are used for communicating with customers.</p> <h2>Class and Object Concept</h2> <p>Classes</p> <ul style="list-style-type: none"> ▶ Class is a group of objects having same attributes and operations, relationships and semantics. ▶ The classes appear as common nouns or noun phrases. ▶ Objects in a class share a common semantic purpose. For Example both the dog and cat have the properties like tail and legs and they belong to same class Animal. ▶ Grouping the objects into corresponding classes make the design Abstract. <p>Objects</p> <ul style="list-style-type: none"> ▶ The main purpose of class model is to describe objects. ▶ Object is an instance of a class. ▶ The objects can be conceptual entities, real world entities, or important things from implementation point of view. ▶ The objects are normally nouns. The choice of objects is done by judgements. ▶ For Example: If a student is a class then Anuja, pooja and Kajal are the objects of the class students. Each student has its own name, roll no, and address. <h2>Class Diagrams</h2> <ul style="list-style-type: none"> ▶ The class Model is represented by two types of diagrams:- <p>1) Class Diagrams</p> <ul style="list-style-type: none"> ▶ Class diagrams provide a graphic notation for modeling classes and their relationships thereby describing possible objects. ▶ Useful for abstract modeling and designing actual programs. ▶ They are concise ,easy to understand <p>2) Object Diagrams</p> <ul style="list-style-type: none"> ▶ Object diagrams shows individual objects and their relationship . ▶ A class diagram corresponds to an infinite set of object diagrams. 	10	CO1	L2
---	--	----	-----	----

Continue....

#Notation used for class & object diagram:

→For class diagram:-



→For object diagram:-



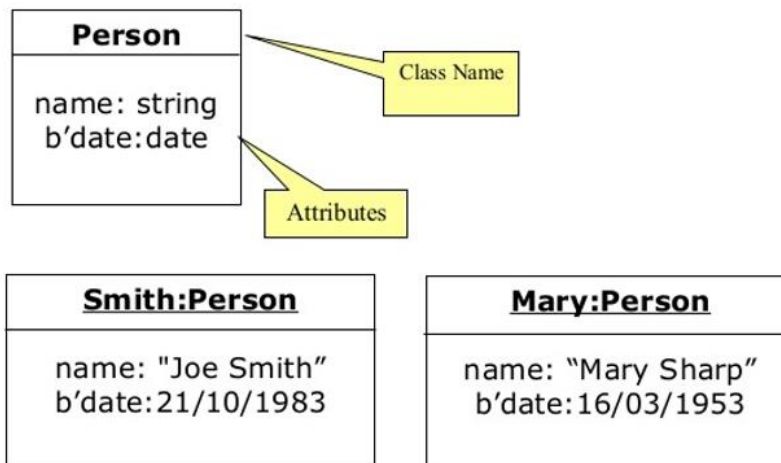
Values and Attributes

- ▶ A value is a piece of data.
- ▶ An attribute is a named property of a class that describes a value held by each object of the class.
- ▶ E.g. Name, birthdate and weight are attributes of Person class.
 - Color, model year and weight are att. Of Car class.
- ▶ Each attribute name is unique within a class.
- ▶ So Person and Car class have attribute called weight.

Values and Attributes continue....

Example:-

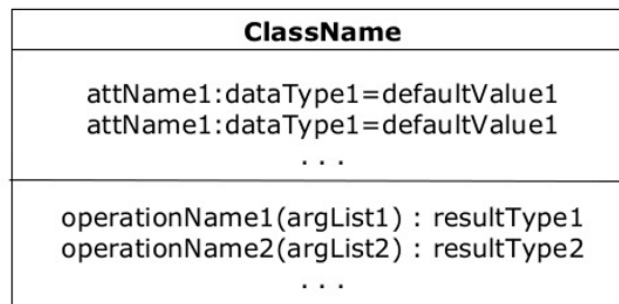
→List att. In the second compartment of the class box.



Operations and Methods

- ▶ Objects have procedures or functions which are called as **operations**.
 - ▶ All the objects in the same class share the common set of operations.
 - ▶ For Example- The class Shape can have various objects such as rectangle, triangle or square having **common operations** such as move, draw, print.
 - ▶ **A method** is the implementation of an operation for a class.
 - ▶ When an operation has methods on several classes it is important that the methods all have the same Signature.
 - ▶ Signature means the number and types of arguments and type of return value.
 - ▶ e.g. print should not have fileName as an argument for one method filePointer for another.
- The attribute and operation compartments are optional.
- A missing attribute / operation compartment means that attributes / operations are unspecified.
- An empty compartment means attributes/operations are specified and that are none.

UML 16

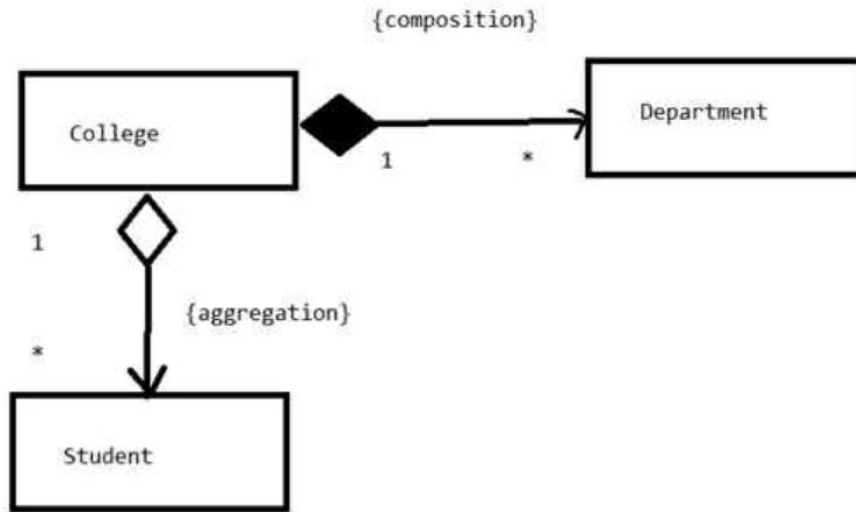


PART - III

5	<p>What is abstract class? Explain the difference between aggregations versus composition with suitable example and UML representation</p> <p>Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods</p> <p>Aggregation is a stronger form of association. An association is a link connecting two classes. In UML, a link is placed between the “whole” and the “parts” classes with a diamond head attached to the “whole” class to indicate that this association is an aggregation</p> <p>Composition is really a strong form of aggregation. Composition has only one owner. Composition cannot exist independent of their owner. Composition lives or dies with their owner. It is represented using a filled diamond head.</p> <p>The main differentiator between aggregation and composition is the lifecycle dependence between whole and part. In aggregation, the part may have an independent lifecycle, it can exist independently. When the whole is destroyed the part may continue to exist. Composition is a stronger form of aggregation. The lifecycle of the part is strongly dependent on the lifecycle of the whole. When the whole is destroyed, the part is destroyed too.</p>	10	CO1	L3
---	--	----	-----	----

Aggregation Example A car has many parts. A part can be removed from one car and installed into a different car. If we consider a salvage business, before a car is destroyed, they remove all saleable parts. Those parts will continue to exist after the car is destroyed.

Composition Example For example, a building has rooms. A room can exist only as part of a building. The room cannot be removed from one building and attached to a different one. When the building ceases to exist so do all rooms that are part of it.



(OR)

6

What do you mean by abstraction? Discuss different types of modeling techniques used for object oriented modeling and design

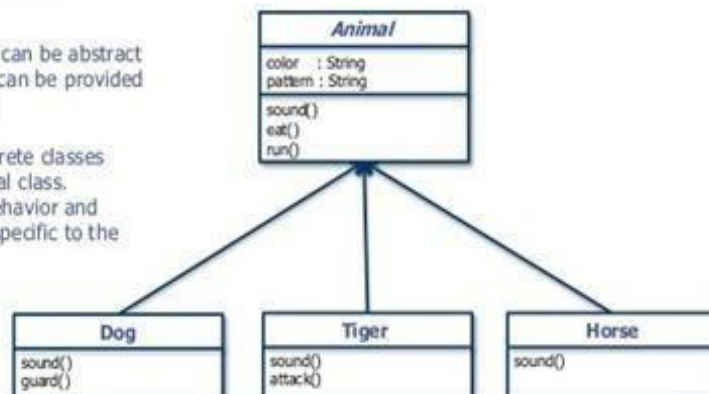
In object-oriented programming, abstraction is one of three central principles (along with encapsulation and inheritance). Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency. In the same way that abstraction sometimes works in art, the object that remains is a representation of the original, with unwanted detail omitted. The resulting object itself can be referred to as an abstraction, meaning *a named entity made up of selected attributes and behavior specific to a particular usage of the originating entity*. Abstraction is related to both encapsulation and data hiding.

Example

Here we have the abstract class *Animal*. We have defined **color** and **pattern** properties and **sound**, **eat** and **run** behaviors which are present in all animals.

The behaviors/ methods can be abstract and the implementation can be provided in the inheriting classes.

Now we have three concrete classes inheriting from the Animal class. Overriding the **sound** behavior and adding some behaviors specific to the entities.

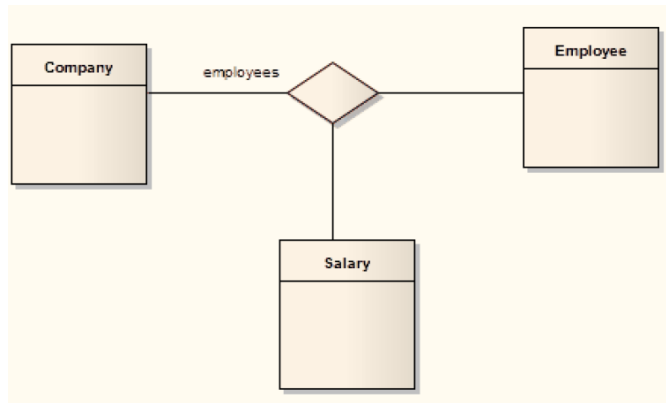


10

CO1

L2

	<p>The different types of modeling techniques are:</p> <p>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.</p> <p>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.</p> <p>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.</p>			
Part – IV				
7	<p>Describe in brief about any 5 of the following a.)Links b.)Association c.)Association class d.)Qualified Association e.) n-ary Association f.)Multiplicity g.)Bags & sequence</p> <p>Links: In object modeling links provides a relationship between the objects. These objects or instance may be same or different in data structure and behavior. Therefore a link is a physical or conceptual connection between instances (or objects). For example: Ram works for HCL company. In this example “works for” is the link between “Ram” and “HCL company”. Links are relationship among the objects (instance)</p> <p>Associations: The object modeling describes as a group of links with common structure and common semantics. All the links among the object are the forms of association among the same classes. The association is the relationship among classes.</p> <p>Association class: is an association that is also a class. Like the links of an association, the instances of an association class derive identity from instances of the constituent classes. Like a class, an association class can have attributes and operations and participate in associations.</p> <p>Qualified associations: A qualifier lets you make a more precise traversal. The syntax is to enclose the qualifier value in brackets. Alternatively, you can ignore the qualifier and traverse a qualified association as if it were a simple association.</p> <p>n-Ary Association : element is used to model complex relationships between three or more elements, typically in a Class diagram. It is not a commonly-employed device, but can be used to good effect where there is a dependant relationship between several elements. It is generally used with the Associate connector, but the relationships can include other types of connector.</p>	10	CO1	L2



(a) **Bags and Sequence:** A **bag** is a collection of elements with duplicates allowed. A **sequence** is an ordered collection of elements with duplicates allowed. In Figure 3.16 an itinerary is a sequence of airports and the same airport can be visited more than once. Like the *{ordered}* indication, *{bag}* and *{sequence}* are permitted only for binary associations.

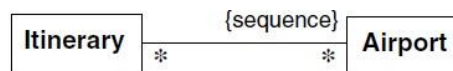


Figure 3.16 An example of a sequence. An itinerary may visit multiple airports, so you should use *{sequence}* and not *{ordered}*.

Note that the *{ordered}* and the *{sequence}* annotations are the same, except that the first disallows duplicates and the other allows them. A sequence association is an ordered bag,

while an ordered association is an ordered set.

(OR)

8 (a)	<p>What is a pattern? Describe pattern categories</p> <p>A Pattern in software architecture describes a particular recurring design problem that arises in specific design context, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the way in which they collaborate</p> <p>Categories</p> <ul style="list-style-type: none"> • Architectural patterns • Design patterns • Idioms <p>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. Architectural patterns are templates for concrete software architectures. They specify the systemwide structural properties of an application, and have an impact on the architecture of its subsystems. The selection of an</p>	06	CO5	L2
-------	--	----	-----	----

	<p>architectural pattern is therefore a fundamental design decision when developing a software system.</p> <p>E,g The Model-View-Controller pattern</p> <p>Design pattern provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes a commonly-recurring structure of communicating components that solves a general design problem within a particular context. Design patterns are medium-scale patterns. They are smaller in scale than architectural patterns, but tend to be independent of a particular programming language or programming paradigm. The application of a design pattern has no effect on the fundamental structure of a software system, but may have a strong influence on the architecture of a subsystem. Idioms deal with the implementation of particular design issues.</p> <p>An idiom is a low-level pattern specific to a programming language. An idiom describes how to implement particular aspects of components or the relationships between them using the features of the given language. Idioms represent the lowest-level patterns. They address aspects of both design and implementation. Most idioms are language-specific—they capture existing programming experience</p>			
(b)	<p>What are design patterns? Describe its categories</p> <p>A design pattern provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes a commonly-recurring structure of communicating components that solves a general design problem within a particular context."</p> <p>A design pattern is a mid-level abstraction. Its choice does not affect the fundamental structure of the software system, but it does affect the structure of a subsystem. Like the architectural pattern, the design pattern tends to be independent of the implementation language to be used.</p> <p>Creational Patterns These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator. This gives program more flexibility in deciding which objects need to be created for a given use case.</p> <p>Structural Patterns These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.</p> <p>Behavioral Patterns These design patterns are specifically concerned with communication between objects.</p>	04	CO5	L2
Part – V				
9	<p>Describe pattern template What are the contents of pattern description template?</p> <p>A Pattern in software architecture describes a particular recurring design problem that arises in specific design context, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its</p>	10	CO5	L1

constituent components, their responsibilities and relationships, and the way in which they collaborate

Pattern Description template

- Name** Meaningful name and short summary
- Example** Demonstrate existence of the problem & need for the pattern.
- Context** Situation in which the pattern may apply
- Problem** Problem addressed & forces associated
- Solution** Solution principle underlying the pattern
- Structure** Specification of the structural aspect
- Dynamics** Run-time behaviour
- Implementation** Guideline for implementation
- Variants** Description of variants
- Known Uses** Examples of the use of the pattern
- Consequences** Benefits and potential liabilities
- See Also** Reference to patterns that solve similar problems

(OR)

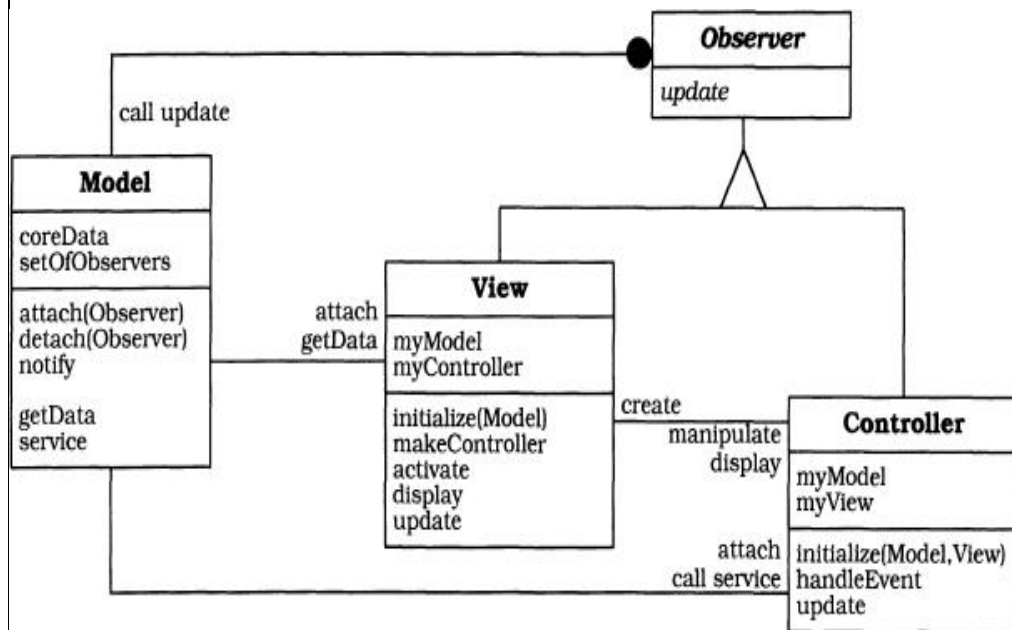
<p>10</p>	<p>Discuss the concept of architectural pattern with Model View Controller pattern</p> <p>An architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them. Architectural patterns are templates for concrete software architectures. They specify the systemwide structural properties of an application, and have an impact on the architecture of its subsystems. The selection of an architectural pattern is therefore a fundamental design decision when developing a software system.</p> <p>E,g The Model-View-Controller pattern</p> <p><u>Model-View-Controller pattern (MVC)</u></p> <p>It divides an interactive application into three components. The model contains the core functionality and data. Views display information to the user. Controllers handle user input. Views and controllers together comprise the user interface. A change-propagation mechanism ensures consistency between the user interface and the model.</p> <p>EXAMPLE</p> <p>Consider a simple information system for political elections with proportional representation.</p> <p>CONTEXT Interactive applications with a flexible human-computer interface.</p> <p>PROBLEM User interfaces are especially prone to change requests.</p> <ul style="list-style-type: none"> ▪ The display and behavior of the application must reflect data manipulations immediately. ▪ Changes to the user interface should be easy ▪ Porting the user interface should not affect code in the core of the application <p>SOLUTION MVC divides an interactive application into the three areas: <i>processing</i>, <i>output</i>, and <i>input</i>.</p> <p>STRUCTURE The change-propagation mechanism maintains a registry of the dependent components within the model.</p>	<p>10</p>	<p>CO5</p>	<p>L2</p>
-----------	--	-----------	------------	-----------

<p>Class Model</p>	<p>Collaborators</p> <ul style="list-style-type: none"> • View • Controller
<p>Responsibility</p> <ul style="list-style-type: none"> • Provides functional core of the application. • Registers dependent views and controllers. • Notifies dependent components about data changes. 	

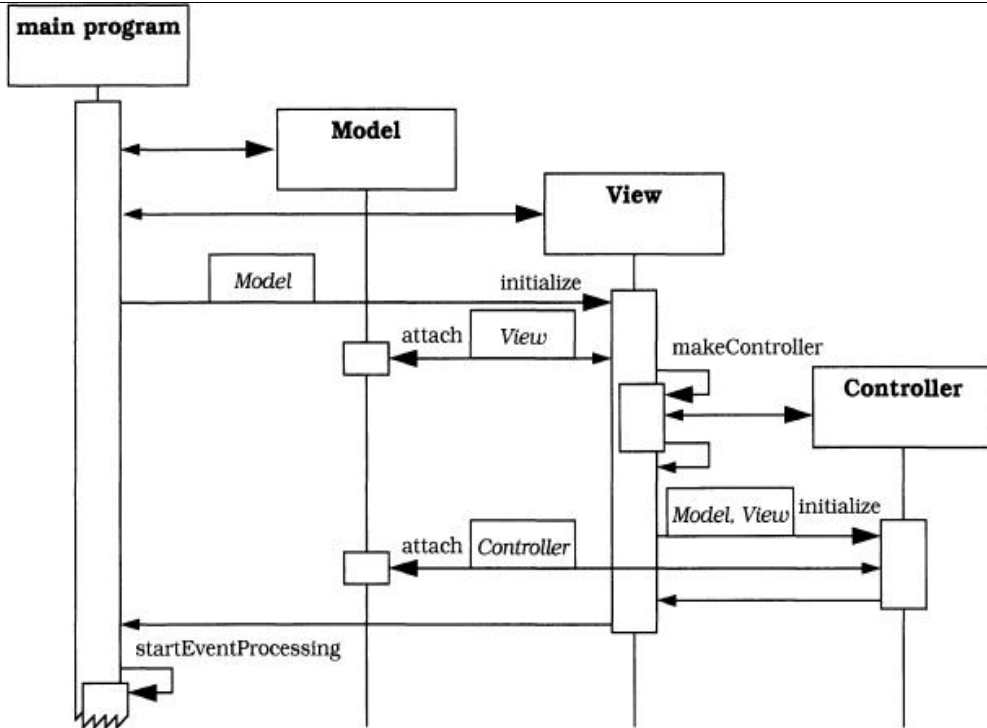
<p>Class View</p>	<p>Collaborators</p> <ul style="list-style-type: none"> • Controller • Model 	<p>Class Controller</p>	<p>Collaborators</p> <ul style="list-style-type: none"> • View • Model
<p>Responsibility</p> <ul style="list-style-type: none"> • Creates and initializes its associated controller. • Displays information to the user. • Implements the update procedure. • Retrieves data from the model. 		<p>Responsibility</p> <ul style="list-style-type: none"> • Accepts user input as events. • Translates events to service requests for the model or display requests for the view. • Implements the update procedure, if required. 	

- ❑ An object oriented implementation of MVC would define a separate class for each component

Scenario 1



Scenario 2



IMPLEMENTATION

1. *Separate human-computer interaction from core functionality*
2. *Implement the change-propagation mechanism.*
3. *Design and implement the views.*
4. *Design and implement the controllers.*
5. *Design and implement the view-controller relationship.*
6. *Implement the set-up of MVC.*

VARIANTS

This variant relaxes the separation of view and controller. You can combine the responsibilities of the view and the controller from MVC in a single component by sacrificing exchangeability of controllers. This kind of structure is often called a Document-View architecture.

KNOWN USES: SMALLTALK and MFC

CONSEQUENCES

Benefits:

- Multiple views of the same model.*
- Synchronized views.*
- 'Pluggable' views and controllers.*
- Exchangeability of 'look and feel'.*
- Framework potential*

Liabilities

- *Increased complexity.*
- *Potential for excessive number of updates.*
- *Intimate connection between view and controller.*
- *Close coupling of views and controllers to a model.*
- *Inefficiency of data access in view.*
- *Inevitability of change to view and controller when porting.*
- *Difficulty of using MVC with modern user-interface tools.*

SEE ALSO

The Presentation-Abstraction –Control pattern.