**OBJECT ORIENTED CONCEPTS**

**WITH UML**

**LAB MANUAL**

**PARUL UNIVERSITY**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**INFORMATION TECHNOLOGY**

**WAGHODIA,VADODARA**

**PREFACE**

It gives us immense pleasure to present the first edition of *Object Oriented Concepts with UML* for the B.Tech. 2nd year students for PARUL UNIVERSITY.

The Object Oriented Concepts with UML theory and laboratory courses at **PARUL UNIVERSITY, WAGHODIA, VADODARA** are designed in such a way that students develop the basic understanding of the subject in the theory classes and then try their hands on the experiments to realize the various physical phenomena learnt during the theoretical sessions. The main objective of the Object Oriented Concepts with UML laboratory course is: **Learning Object Oriented Concepts with UML through Experimentations**. All the practicals are designed to illustrate various phenomena of Object Oriented Concepts withUML and also to expose the students to various notations of UML using Visio Pro and their uses.

The objective of this ***Object Oriented Concepts with UML Practical Book*** is to provide a comprehensive source for all the practicals included in the Object Oriented Concepts with UML laboratory course. It explains all the aspects related to every experiment such as: basic underlying analysis and design of real world problems, How to draw the basic UML diagrams, how to use the notations of UML, how to analyse and draw the class diagrams, object diagram, state diagram, use case diagram , sequence diagram , activity diagram and collaboration diagram for the particular problem..

We acknowledge the authors and publishers of all the books which we have consulted while developing this Practical book. Hopefully this Object Oriented Concepts with UML *Practical Book* will serve the purpose for which it has been developed.

**Instructions to students**

1. The main objective of the OBJECT ORIENTED CONCEPTS WITHUML laboratory is: ***Learning OBJECT ORIENTED CONCEPTS WITH UML through the Experimentation***. ***All the experiments are designed to illustrate various phenomena in different areas of subject and also to expose the students to various instruments and their uses***.
2. Be prompt in arriving to the laboratory and always come well prepared for the experiment.
3. Be careful while working on the equipment’s operated with high voltage power supply.
4. Work quietly and carefully. Give equal opportunity to all your fellow students to work on the instruments.
5. Every student should have his/her individual copy of the notes/book.
6. Every student have to prepare the file specifically reserved for the subject
7. Every student has to necessarily bring his/her *file* when he/she comes to the Practical to perform the experiment.
8. After completion of the experiment, get them verified by the instructor.
9. Do not forget to get the information of your next allotment (the experiment which is to be performed by you in the next laboratory session) before leaving the laboratory from the Technical Assistant.

The grades ***OBJECT ORIENTED CONCEPTS WITH UML*** practical course work will be awarded based on your performance in the laboratory, regularity, recording of experiments , lab quiz, regular viva-voce and end-term examination.



CERTIFICATE

This is to certify that Mr./Ms.…………………………………………………… with enrolment no. …………………………has successfully completed his/her laboratory experiments

In Object Oriented Concepts with UML laboratory during the academic year…………

Date: Signature of lab teacher:

Signature of HOD:

**INDEX**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SR. NO.** | **TITLE** | **Number of hours** | **Page**  **No.** | **Performance date** | **Assessment date** | **Marks out 0f 10** | **Sign** |
| 1 | Implementation of class Diagram. | 2 |  |  |  |  |  |
| 2 | Implementation of object Diagram. | 2 |  |  |  |  |  |
| 3 | Implementation of state diagram | 2 |  |  |  |  |  |
| 4 | Implementation of use case diagram | 2 |  |  |  |  |  |
| 5 | Implementation of Sequence diagram | 2 |  |  |  |  |  |
| 6 | Implementation of activity Diagram | 2 |  |  |  |  |  |
| 7 | Implementation of collaboration diagram | 2 |  |  |  |  |  |
| 8 | Study of domain class model & domain state model. | 2 |  |  |  |  |  |
| 9 | Study on Rational Tool. | 2 |  |  |  |  |  |
| 10 | Study of MVC Design Pattern. | 2 |  |  |  |  |  |

**Practical No: 1**

**Aim:** Implementation of Class Diagram

**Overview:** 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.

The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a *structural diagram*.

So the purpose of the class diagram can be summarized as:

* Analysis and design of the static view of an application.
* Describe responsibilities of a system.
* Base for component and deployment diagrams.
* Forward and reverse engineering.

## How to draw Class Diagram?

The following points should be remembered while drawing a class diagram:

* The name of the class diagram should be meaningful to describe the aspect of the system.
* Each element and their relationships should be identified in advance.
* Responsibility (attributes and methods) of each class should be clearly identified.
* For each class minimum number of properties should be specified. Because unnecessary properties will make the diagram complicated.
* Use notes when ever required to describe some aspect of the diagram. Because at the end of the drawing it should be understandable to the developer/coder
* Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

Example of an *Order System* of an application. So it describes a particular aspect of the entire application.

* First of all *Order* and *Customer* are identified as the two elements of the system and they have a *one to many* relationship because a customer can have multiple orders.
* We would keep *Order* class is an abstract class and it has two concrete classes (inheritance relationship) *Specialorder* and *NormalOrder*.
* The two inherited classes have all the properties as the *Order* class. In addition, they have additional functions like *dispatch ()* and *receive ()*.

So the following class diagram has been drawn considering all the points

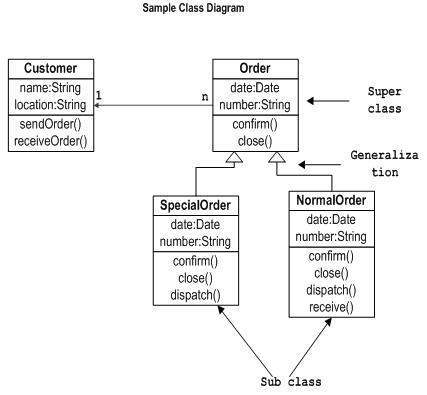


Figure: Class Diagram

**Practical No: 2**

## Aim: Implementation of Object Diagram

## Overview: Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.

Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams. Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.

So the purpose of the object diagram can be summarized as:

* Forward and reverse engineering.
* Object relationships of a system
* Static view of an interaction.
* Understand object behaviour and their relationship from practical perspective

## How to draw Object Diagram?

* First, analyse the system and decide which instances are having important data and association.
* Second, consider only those instances which will cover the functionality.
* Third, make some optimization as the numbers of instances are unlimited.

It has the following objects

* Customer
* Order
* Specialorder
* NormalOrder

Example:

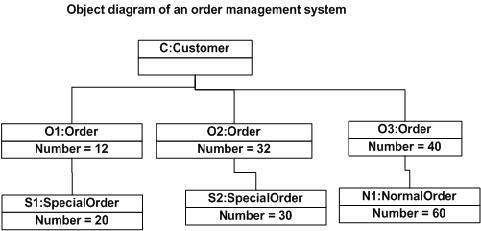
* Now the customer object (C) is associated with three order objects (O1, O2 and O3). These order objects are associated with special order and normal order objects (S1, S2 and N1). The customer is having the following three orders with different numbers (12, 32 and 40) for the particular time considered.
* Now the customer can increase number of orders in future and in that scenario the object diagram will reflect that. If order, special order and normal order objects are observed then you will find that they are having some values.
* For orders the values are 12, 32, and 40 which implies that the objects are having these values for the particular moment (here the particular time when the purchase is made is considered as the moment) when the instance is captured.
* The same is for special order and normal order objects which are having number of orders as 20, 30 and 60. If a different time of purchase is considered then these values will change accordingly.
* So the following object diagram has been drawn considering all the points mentioned above:
* 

Figure: Object Diagram

**Practical No: 3**

**Aim:** Implementation of State diagram

The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system.

A State chart diagram describes a state machine.

## Purpose:

State chart diagram is one of the five UML diagrams used to model dynamic nature of a system. They define different states of an object during its lifetime. And these states are changed by events. So Statechart diagrams are useful to model reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. So the most important purpose of Statechart diagram is to model life time of an object from creation to termination.

Statechart diagrams are also used for forward and reverse engineering of a system. But the main purpose is to model reactive system.

Following are the main purposes of using Statechart diagrams:

* To model dynamic aspect of a system.
* To model life time of a reactive system.
* To describe different states of an object during its life time.
* Define a state machine to model states of an object.

## How to draw State chart Diagram?

State chart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

Before drawing a State chart diagram we must have clarified the following points:

* Identify important objects to be analysed.
* Identify the states.
* Identify the events.

The following is an example of a Statechart diagram where the state of *Order*object is analysed.

The first state is an idle state from where the process starts. The next states are arrived for events like *send request*, *confirm request*, and *dispatch order*. These events are responsible for state changes of order object.

During the life cycle of an object (here order object) it goes through the following states and there may be some abnormal exists also. This abnormal exit may occur due to some problem in the system. When the entire life cycle is complete it is considered as the complete transaction as mentioned below.

The initial and final state of an object is also shown below.

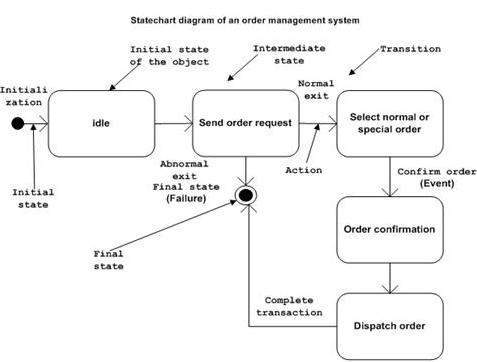


Figure: State diagram

So, the main usages can be described as:

* To model object states of a system.
* To model reactive system. Reactive system consists of reactive objects.
* To identify events responsible for state changes.
* Forward and reverse engineering.

**Practical No: 4**

**Aim:** Implementation of Use-Case Diagram

## Overview: To model a system the most important aspect is to capture the dynamic behaviour. To clarify a bit in details, dynamic behaviour means the behaviour of the system when it is running /operating.

## Purpose: The purpose of use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Because other four diagrams (activity, sequence, collaboration and Statechart) are also having the same purpose. So we will look into some specific purpose which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analysed to gather its functionalities use cases are prepared and actors are identified.

Now when the initial task is complete use case diagrams are modelled to present the outside view.

So, in brief, the purposes of use case diagrams can be as follows:

* Used to gather requirements of a system.
* Used to get an outside view of a system.
* Identify external and internal factors influencing the system.
* Show the interacting among the requirements are actors.

**How to draw Use Case Diagram?**

Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analysed the functionalities are captured in use cases.

So, we can say that use cases are nothing but the system functionalities written in an organized manner. Now the second things which are relevant to the use cases are the actors. Actors can be defined as something that interacts with the system.

The actors can be human user, some internal applications or may be some external applications. So, in a brief when we are planning to draw an use case diagram we should have the following items identified.

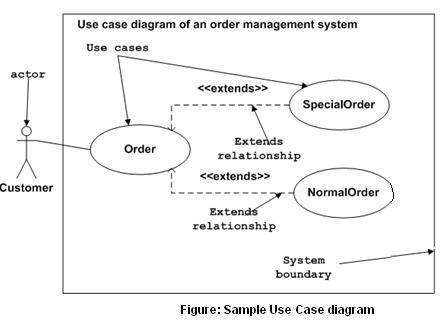
* Functionalities to be represented as a use case
* Actors
* Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. So, after identifying the above items we must follow the following guidelines to draw an efficient use case diagram.

* The name of a use case is very important. So, the name should be chosen in such a way so that it can identify the functionalities performed.
* Give a suitable name for actors.
* Show relationships and dependencies clearly in the diagram.
* Do not try to include all types of relationships. Because the main purpose of the diagram is to identify requirements.
* Use note whenever required to clarify some important points.

The following is a sample use case diagram representing the order management system. So, if we look into the diagram then we will find three use cases (Order, Specialorder and NormalOrder) and one actor which is customer.

The Specialorder and NormalOrder use cases are extended from Order use case. So, they have extended relationship. Another important point is to identify the system boundary which is shown in the picture. The actor Customer lies outside the system as it is an external user of the system.



## Where to Use Case Diagrams?

So, the following are the places where use case diagrams are used:

* Requirement analysis and high level design.
* Model the context of a system.
* Reverse engineering.
* Forward engineering.

**Practical No: 5**

# **Aim: Implementation of Sequence Diagram Sequence Diagram**

**Overview:** Sequence diagrams describe interactions among classes in terms of an exchange of messages over time. They're also called event diagrams. A sequence diagram is a good way to visualize and validate various runtime scenarios.

## How to Use Sequence Diagrams

* Model and document how your system will behave in various scenarios
* Validate the logic of complex operations and functions

## Basic Sequence Diagram Symbols and Notations

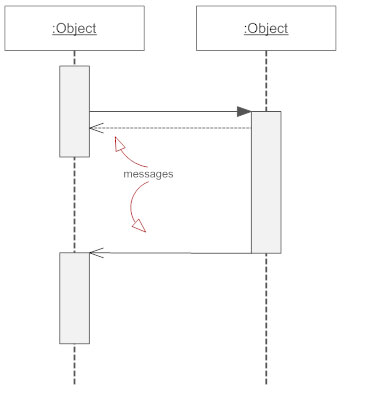
**Class Roles or Participants**  
Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

Object symbol - Sequence diagram

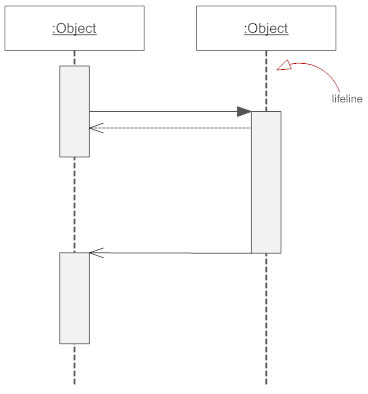
**Activation or Execution Occurrence**  
Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process, or waiting for a reply message, use a thin grey rectangle placed vertically on its lifeline.



**Messages**  
Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks. For message types, see below.



**Lifelines**  
Lifelines are vertical dashed lines that indicate the object's presence over time.



**Destroying Objects**  
Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X. This object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

**Loops**  
A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].

## Types of Messages in Sequence Diagrams

**Synchronous Message**  
A synchronous message requires a response before the interaction can continue. It's usually drawn using a line with a solid arrowhead pointing from one object to another.

Synchronous message - Sequence diagram

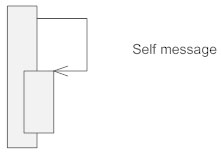
**Asynchronous Message**  
Asynchronous messages don't need a reply for interaction to continue. Like synchronous messages, they are drawn with an arrow connecting two lifelines; however, the arrowhead is usually open and there's no return message depicted.

Simple messageAsyncrhonous message

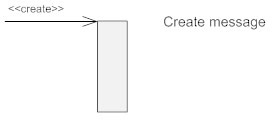
**Reply or Return Message**  
A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

Reply messages

**Self Message**  
A message an object sends to itself, usually shown as a U shaped arrow pointing back to itself.



**Create Message**  
This is a message that creates a new object. Similar to a return message, it's depicted with a dashed line and an open arrowhead that points to the rectangle representing the object created.



**Delete Message**  
This is a message that destroys an object. It can be shown by an arrow with an x at the end.

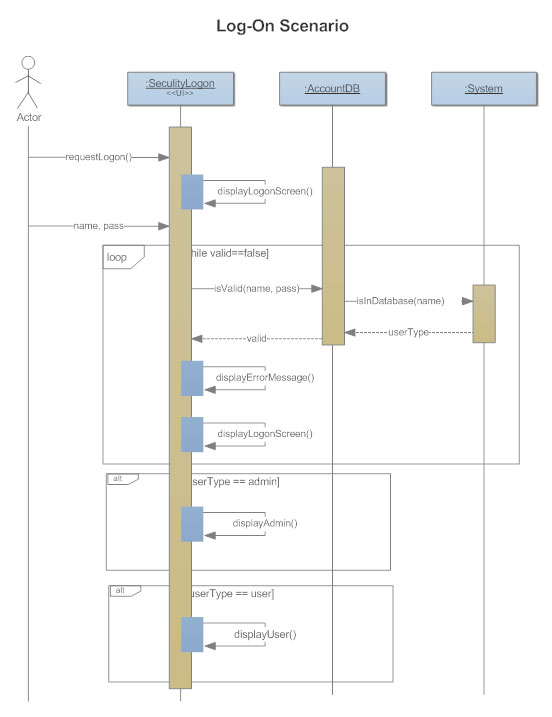
Delete message

**Found Message**  
A message sent from an unknown recipient, shown by an arrow from an endpoint to a lifeline.

Found message

**Lost Message**  
A message sent to an unknown recipient. It's shown by an arrow going from a lifeline to an endpoint, a filled circle or an x.

Lost message

****

**Figure: Sequence Diagram**

**Practical No: 6**

**Aim:** Implementation of Activity Diagram

## Overview: Activity diagram is another important diagram in UML to describe dynamic aspects of the system.

Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system.

o the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow control by using different elements like fork, join etc.

**Purpose:** The basic purposes of activity diagrams are similar to other four diagrams. It captures the dynamic behaviour of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part.

It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

So the purposes can be described as:

* Draw the activity flow of a system.
* Describe the sequence from one activity to another.
* Describe the parallel, branched and concurrent flow of the system.

**How to draw Activity Diagram?**

Activity diagrams are mainly used as a flow chart consists of activities performed by the system. But activity diagram is not exactly a flow chart as they have some additional capabilities. These additional capabilities include branching, parallel flow, swim lane etc.

Before drawing an activity diagram, we must have a clear understanding about the elements used in activity diagram. The main element of an activity diagram is the activity itself. An activity is a function performed by the system. After identifying the activities, we need to understand how they are associated with constraints and conditions.

So before drawing an activity diagram we should identify the following elements:

* Activities
* Association
* Conditions
* Constraints

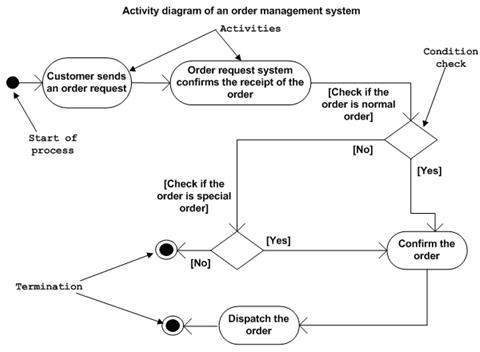
Once the above-mentioned parameters are identified we need to make a mental layout of the entire flow. This mental layout is then transformed into an activity diagram.

The following is an example of an activity diagram for order management system. In the diagram four activities are identified which are associated with conditions. One important point should be clearly understood that an activity diagram cannot be exactly matched with the code. The activity diagram is made to understand the flow of activities and mainly used by the business users.

The following diagram is drawn with the four main activities:

* Send order by the customer
* Receipt of the order
* Confirm order
* Dispatch order

After receiving the order request condition checks are performed to check if it is normal or special order. After the type of order is identified dispatch activity is performed and that is marked as the termination of the process.



## Figure: Activity Diagram

## Where to use Activity Diagrams?

Following are the main usages of activity diagram:

* Modeling work flow by using activities.
* Modeling business requirements.
* High level understanding of the system's functionalities.
* Investigate business requirements at a later stage.

**Practical No: 7**

**Aim:** Implementation of Collaboration Diagram

**Overview:** This interactive behaviour is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram. The basic purposes of both the diagrams are similar.

Collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

## Purpose: The purposes of interaction diagrams are to visualize the interactive behaviour of the system. Now visualizing interaction is a difficult task. So, the solution is to use different types of models to capture the different aspects of the interaction.

So, the purposes of interaction diagram can be describes as:

* To capture dynamic behaviour of a system.
* To describe the message flow in the system.
* To describe structural organization of the objects.
* To describe interaction among objects.

**Collaboration Diagram**

The Collaboration diagram is having four objects (Customer, Order, SpecialOrder and NormalOrder).

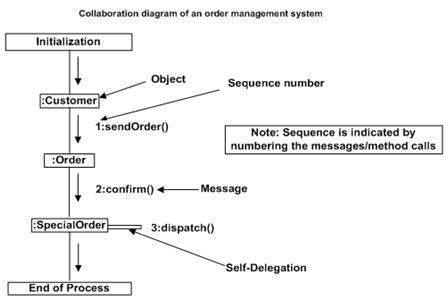
The following diagram has shown the message sequence for *SpecialOrder* object and the same can be used in case of *NormalOrder* object.

The first call is *sendOrder ()* which is a method of *Order* object. The next call is*confirm ()* which is a method of *SpecialOrder* object and the last call is *Dispatch ()* which is a method of *SpecialOrder* object.

It shows the object organization as shown below. Here in collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.

The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization whereas the collaboration diagram shows the object organization.

Now to choose between these two diagrams the main emphasis is given on the type of requirement. If the time sequence is important then sequence diagram is used and if organization is required, then collaboration diagram is used.



**Figure: Collaboration Diagram**

**Practical No: 8**

**Aim:** Study of Domain Class Model & Domain State Model.

Domains and Domain Models

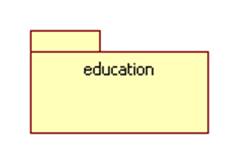
A domain is a collection of related concepts, relationships, and workflows.

Examples of domains include:

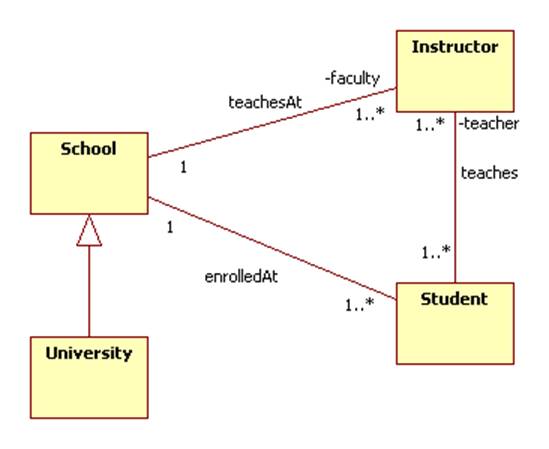
Science, engineering, medicine, business, government, military

A domain model is a package containing class and activity diagrams.

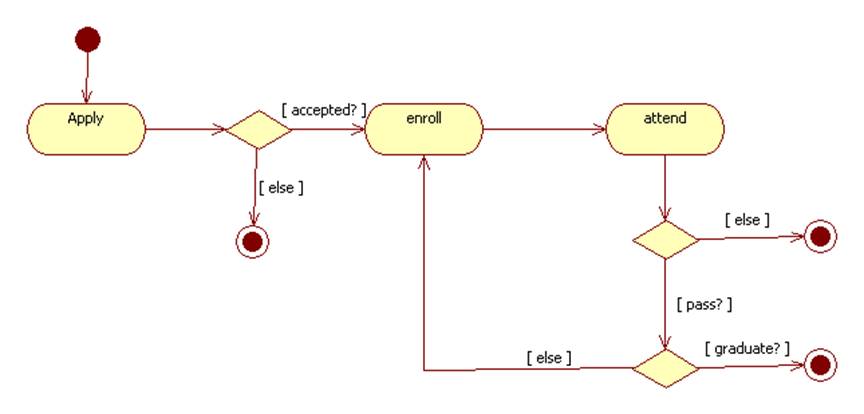
For example, the education domain can be modelled by a package:



This package contains the following class diagram which models the concepts of school, university, instructor, and student as well as the relationships teaches-at, teaches, and enrolled-at:



It also contains the following activity diagram that represents the student workflow:

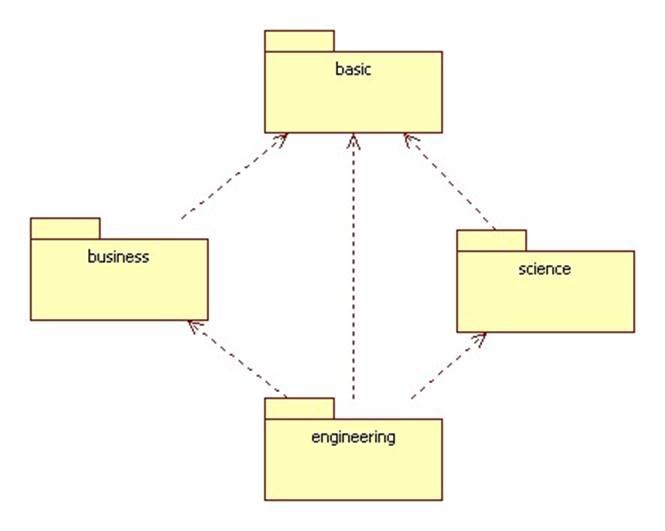


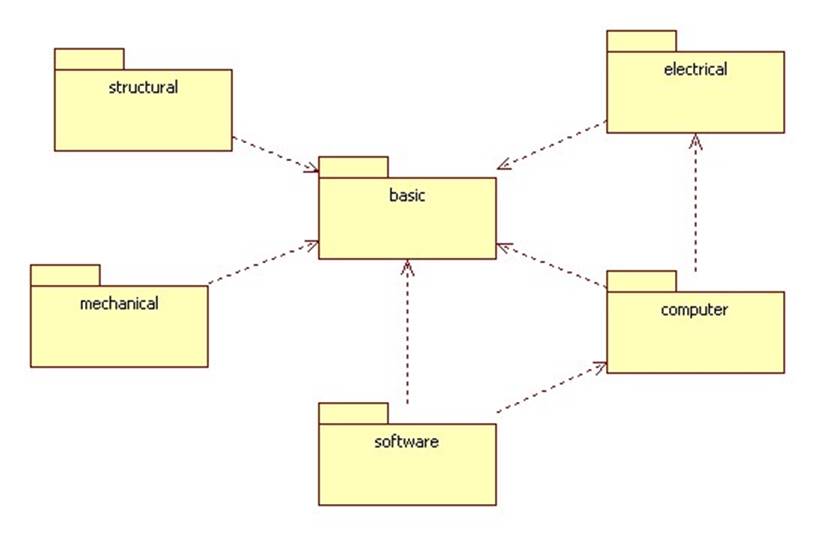
### **Dependencies between Domains**

Domains can import elements from other domains. This can be shown by a package diagram. A package diagram is a class diagram that shows packages and the dependencies between them.

A dependency between two packages is represented by a dashed arrow. More specifically, dashed arrow from package A to package B indicates that package A uses some of the items contained in package B. We say that A imports from B or that B exports to A.

Here are a few examples:





**Practical No: 9**

**Aim:** Study of Rational Tool.

Rational Rose is an [object-oriented](https://searchmicroservices.techtarget.com/definition/object-oriented-programming-OOP) Unified Modeling Language ([UML](https://searchsoftwarequality.techtarget.com/definition/Unified-Modeling-Language)) software design tool intended for visual modeling and [component](https://whatis.techtarget.com/definition/component) construction of enterprise-level software applications. In much the same way a theatrical director blocks out a play, a software designer uses Rational Rose to visually create (model) the framework for an application by blocking out classes with actors (stick figures), [use case](https://searchsoftwarequality.techtarget.com/definition/use-case) elements (ovals), objects (rectangles) and messages/relationships (arrows) in a sequence diagram using drag-and-drop symbols. Rational Rose documents the diagram as it is being constructed and then generates code in the designer's choice of [C++](https://searchsqlserver.techtarget.com/definition/C), [Visual Basic](https://searchwindevelopment.techtarget.com/definition/Visual-Basic), [Java](https://www.theserverside.com/definition/Java), Oracle8, [Corba](https://searchsqlserver.techtarget.com/definition/CORBA) or Data Definition Language.

**Practical No: 10**

**Aim:** Study of MVC Design Pattern.

**Model–View–Controller** (usually known as MVC) is an [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern) commonly used for developing [user interfaces](https://en.wikipedia.org/wiki/User_interface) that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for [code reuse](https://en.wikipedia.org/wiki/Code_reuse) and parallel development.

Traditionally used for desktop [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface) (GUIs), this architecture has become popular for designing [web applications](https://en.wikipedia.org/wiki/Web_application). Popular programming languages like [JavaScript](https://en.wikipedia.org/wiki/JavaScript), [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)), [PHP](https://en.wikipedia.org/wiki/PHP), [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), and [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) have MVC frameworks that are used in web application development straight [out of the box](https://en.wikipedia.org/wiki/Out_of_the_box_(feature)).

**Model**

The central component of the pattern.

It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.

**View**

Any representation of information such as a chart, diagram or table.

Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

**Controller**

Accepts input and converts it to commands for the model or view.

