SECTION 1 OBJECT ORIENTED ANALYSIS AND DESIGN LAB

Structure **Page Nos.**

1.0 INTRODUCTION

The Object Oriented System tools provide a deep insight into the actual working methodology of the software industry for Object Oriented Analysis and Design. It enables us to learn the various tools employed in the software development life cycle, using object oriented methodologies, which makes the process easy to understand and implement.

We begin with the requirement gathering and management process. Once the requirement framework is complete we begin our analysis and design process. It provides a direct mapping of our requirements to our design model and implementation model.

For any team to work in harmony it is very essential that a common language is used for communication. This is true for software development team, hence, we use the **Unified Modeling Language (UML)** for communication between the team members. UML is the language consisting of notations and diagrams for depicting any element of the development process. This has become the industry standard and is widely used.

There are many software tools that support analysis, design, and development of object oriented software. These tools provide us the environment where the entire process of development can be specified to its minutest detail. The tools also contain the facility to design our front-end processes as well as our database structure.

1.1 OBJECTIVES

After going through this section, you should be able to:

- analyse and design their project;
- identify the analysis elements of the project and define the association between them, and
- create an analysis model of the project.

1.2 A BRIEF INTRODUCTION TO OBJECT ORIENTED PROGRAMMING AND UNIFIED MODELING LANGUAGE

Let us first answer the question what is Object Oriented Programming. Let us describe it in the context of modeling.

What is Object Oriented programming?

Object Oriented Programming (OOP) is a programming language model organised around "objects" rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data. The programming challenge was how to write the logic, not how to define the data. Object Oriented programming takes the view that what we really care about are the objects we want to manipulate rather than the logic required to manipulate them. Examples of objects range from human beings (described by name, address, and so forth) to buildings and floors (whose properties can be described and managed) down to the little widgets on your computer desktop (such as buttons and scroll bars).

The first step in OOP is to identify all the objects you want to manipulate and how they relate to each other, an exercise often known as data modeling. Once you have identified an object, you generalise it as a class of objects (think of Plato's concept of the "ideal" chair that stands for all chairs) and define the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. A real instance of a class is called (no surprise here) an "object" or, in some environments, an "instance of a class." The object or class instance is what you run in the computer. Its methods provide computer instructions and the class object characteristics provide relevant data. You communicate with objects and they communicate with each other with well-defined interfaces called *messages*.

The concepts and rules used in object oriented programming provide the following important benefits:

- The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics. Called *inheritance*, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.
- Since a class defines only the data it needs to be concerned with, when an instance of that class (an object) is run, the code will not be able to accidentally access other program data. This characteristic of *data hiding* provides greater system security and avoids unintended data corruption.
- The definition of a class is reusable not only by the program for which it is initially created but also by other object oriented programs (and, for this reason, can be more easily distributed for use in networks).
- The concept of data classes allows a programmer to create any new data types called *user defined data types* that are not already defined in the language itself.

One of the first object oriented computer languages was Smalltalk. C++ and Java are the most popular object-oriented languages today. The Java programming language is designed especially for use in distributed applications on corporate networks and the Internet.

Ok, so we have revised the concepts of Object Oriented Programming, but how is an object model represented?
The most commonly used representation for object models is *called Unified Modeling Language (UML).*

What is UML?

The **OMG** defines UML as:

"The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components."

The Unified Modeling Language has quickly become the *de facto* standard for building Object Oriented software. The important point to note here is that UML is a '*language'* for specifying and not a method or procedure. The UML is used to define a software system; to detail the artifacts in the system, to document and construct it is the language that the blueprint is written in. The UML may be used in a variety of ways to support a *software development methodology* (such as the Rational Unified Process) but in itself it does not specify that methodology or process.

UML defines the notation and semantics for the following domains:

- The User Interaction or Use Case Model ─ describes the boundary and interaction between the system and users. Corresponds in some respects to a requirements model.
- The Interaction or Communication Model ─ describes how objects in the system will interact with each other to get work done.
- The State or Dynamic Model ─ State charts describe the states or conditions that classes assume over time. Activity graphs describe the workflows the system will implement.
- The Logical or Class Model ─ describes the classes and objects that will make up the system.
- The Physical Component Model ─ describes the software (and sometimes hardware components) that make up the system.
- The Physical Deployment Model ─ describes the physical architecture and the deployment of components on that hardware architecture.

The UML also defines extension mechanisms for extending the UML to meet specialised needs (for example Business Process Modeling extensions).

What are the UML supported object oriented analysis and design tools?

Object Oriented Unified Modeling Language (UML) software design tools are intended for visual modeling and component construction of enterprise-level software applications. In much the same way a theatrical director blocks out a play, a software designer uses such tools to visually create (model) the framework for an application by blocking out classes with actors (stick figures), use case elements (ovals), objects (rectangles) and messages/relationships (arrows) in a sequence diagram using drag-and-drop symbols. These tools also document the diagram as it is being constructed and then generate code in the designer's choice of C++, Visual Basic, Java, Oracle etc. Some such tools are available to many organisations.

Two important features of such tools are that they have the ability to provide iterative development and reverse engineering. These tools allow designers to take advantage of iterative development (or evolutionary development) because the new application can be created in stages with the output of one iteration becoming the input to the next. (This is in contrast to waterfall development where the whole project is completed from start to finish before a user gets to try it out). Then, as the developer begins to understand how the components interact and makes modifications in the design, by going back and updating the rest of the model to ensure the code remains consistent. The reverse engineering makes sure that the model and documentations are made available even for the code where it did not exist.

Latest software development environments help organisations create business value by improving their software development capability. These tools integrate the software engineering best practices, tools, and services. With these, organisations thrive in an on-demand world by being more responsive, resilient, and focused. Thus, these standards-based, cross-platform solutions help software development teams create and extend business applications, embedded systems and software products.

1.3 UML TERMINOLOGY AND OBJECT ORIENTED ANALYSIS AND DESIGN (OOAD)

In this section let us relate common terminology used in UML for Object Oriented Analysis and Design:

Predefined Models: Most of the OOAD tools contain a set of predefined model elements that are needed to model a certain kind of system. The purpose of a specific model can be to define the architecture of systems of a certain kind or to provide a set of reusable components. Such basic models are used as templates when creating a new model. (Please refer to *Figure 1*).

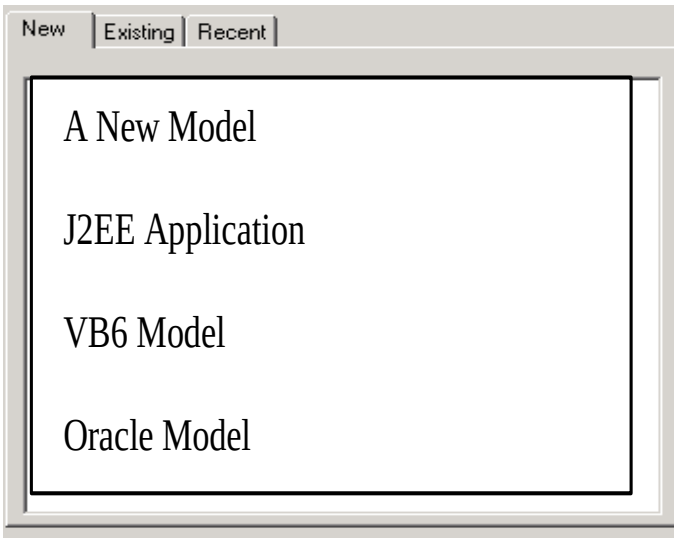


Figure 1: Some sample models

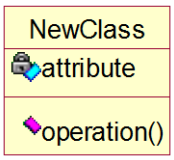
Use Case: A use case is a sequence of actions a system performs that yields an observable result of value to a particular actor.

NewUseCase

Actor: An actor represents many things that interact with the system.

Actor

A class icon is drawn as a 3 part box, with the class name in the top part, a list of attributes (with optional types and values) in the middle part, and a list of operations (with optional argument lists and return types) in the bottom part.

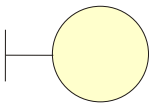


Stereotype: Representing one model element in the form of another model element.

Classes are stereotyped depending on the following categories:

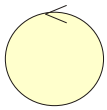
Boundary Class: A boundary class represents an interface between the system and some entity outside the system: a person or another system. Its role is to mediate the exchange of information with the outside world, and to insulate the system from changes in its surroundings. They are of the following types:

- **User Interface Classes**: Intermediate communication with human users of the system.
- **System Interface Classes**: Intermediate communication with other system.
- **Device Interface Classes**: Intermediate communication with external devices.



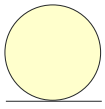
Boundary Class

Control Class: A control class is a class used to model control behaviour specific to one or a few use cases. Control objects (instances of control classes) often control other objects, so their behavior is of the coordinating type. Control classes encapsulate use-case specific behaviour.



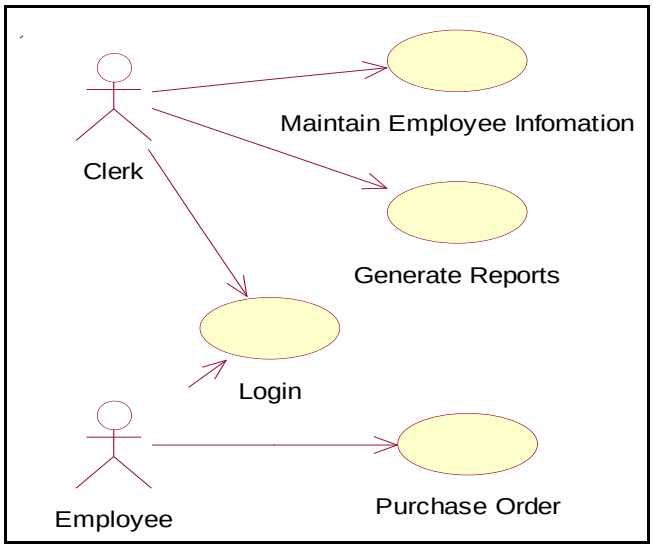
Control Class

Entity Class: An entity class is a class used to model information and associated behavior that must be stored. Entity objects (instances of entity classes) are used to hold and update information about some phenomenon, such as an event, a person, or some real-life object. They are usually persistent, having attributes and relationships needed for a long period, sometimes for the life of the system.

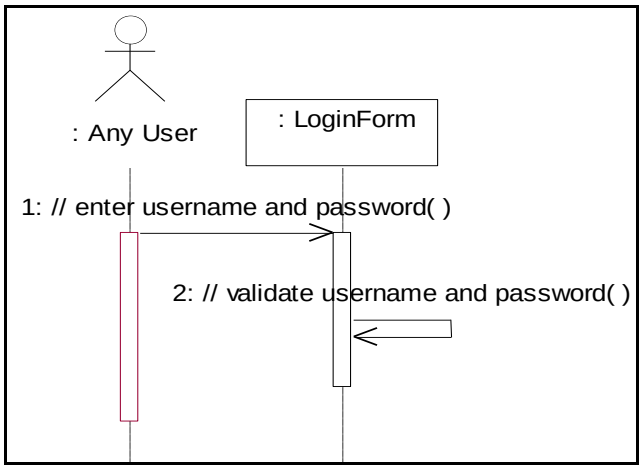


Entity Class

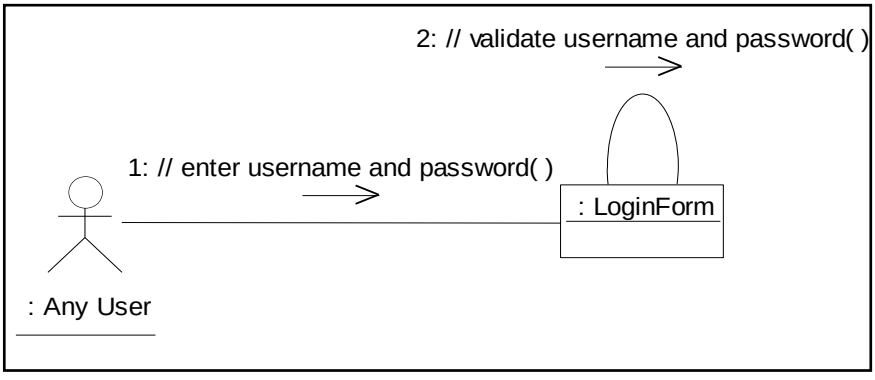
Use Case Diagram: Depicts interaction between Actors and Use Cases.



Sequence Diagram: A sequence diagram is a graphical view of a scenario that shows object interaction in a time-based sequence of what happens first, what happens next. Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.



Collaboration Diagram: A collaboration diagram is an interaction diagram that shows the order of messages that implement an operation or a transaction. Collaboration diagrams show objects, their links, and their messages. They can also contain simple class instances.

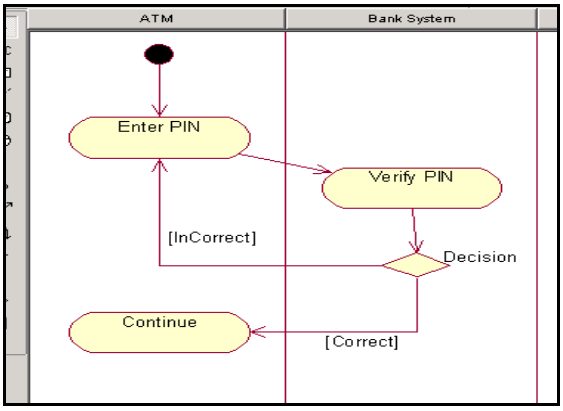


Sequence diagrams are closely related to collaboration diagrams and both are alternate representations of an interaction.

Main difference between sequence and collaboration diagrams: sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other.

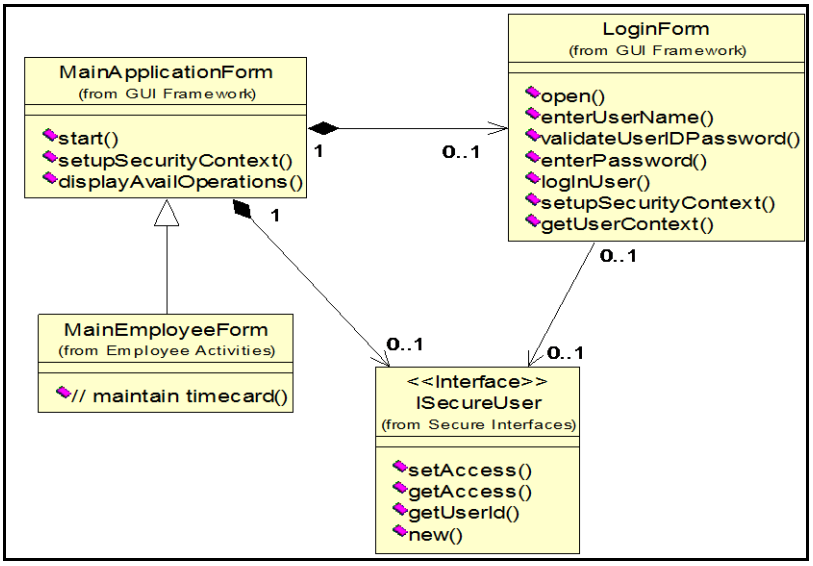
State Chart Diagram: State chart diagrams model the dynamic behaviour of individual classes or any other kind of object. They show the sequences of states that an object goes through, the events that cause a transition from one state to another and the actions that result from a state change.

Activity Diagram: An activity diagram is basically a special case of a state machine in which most of the states are activities and most of the transitions are implicitly triggered by completion of the actions in the source activities.



Class and Object Diagram: A class diagram is a picture for describing generic descriptions of possible systems. Class diagrams contain classes and object diagrams contain objects, but it is possible to mix classes and objects when dealing with various kinds of metadata, so the separation is not rigid.

Class diagrams are more prevalent than object diagrams. Class diagrams contain icons representing classes, interfaces, and their relationships. You can create one or more class diagrams to depict the classes at the top level of the current model; such class diagrams are themselves contained by the top level of the current model. You can also create one or more class diagrams to depict classes contained by each package in your model; such class diagrams are themselves contained by the package enclosing the classes they depict; the icons representing logical packages and classes in class diagrams.



1.4 A SAMPLE PROBLEM

Let us show the process of OOAD through software tools.

1.4.1 Problem Statement for Railway Reservation System

Software has to be developed for automating the manual railway reservation system. The system should be distributed in nature. It should be designed to provide functionalities as explained below:

1. **Reserve Seat:** A passenger should be able to reserve seats in the train. A reservation form is filled by the passenger and given to the clerk, who then checks for the availability of seats for the specified date of journey. If seats are available then the entries are made in the system regarding the train name, train number, date of journey, boarding station, destination, person name, sex and total fare. Passenger is asked to pay the required fare and the tickets are printed. If the seats are not available then the passenger is informed.

2. **Cancel Reservation:** A passenger wishing to cancel a reservation is required to fill a form. The passenger then submits the form and the ticket to the clerk. The clerk then deletes the entries in the system and changes the reservation status of that train. The clerk crosses the ticket by hand to mark as cancelled.

3. **Update Train Information:** Only the administrator enters any changes related to the train information like change in the train name, train number, train route etc. in the system.

4. **Report Generation:** Provision for generation of different reports should be given in the system. The system should be able to generate reservation chart, monthly train report etc.

5. **Login:** For security reasons all the users of the system are given a user id and a password. Only if the id and password are correct is the user allowed entry to the system and select from the options available in the system.

6. **View Reservation Status:** All the users should be able to see the reservation status of the train online. The user needs to enter the train number and the pin number printed on his ticket so that the system can display his current reservation status like confirmed, RAC or Wait-listed.

7. **View Train Schedule:** Provision should be given to see information related to the train schedules for the entire train network. The user should be able to see the train name, train number, boarding and destination stations, duration of journey etc.

1.4.2 Analysis and Design Methodology

The sequences of steps for the process are as follows:

Step 1: Choose a pre-designed model if it exists.

Choose the framework according to the development platform.

Step 2: Make Use of Case Diagram.

The general flow of any analysis and design process is to begin with the identification of the functionalities of the system and the actors associated with the system. We depict the overview of the system to be developed by a Use Case diagram.

After analysing the above problem we have identified the following Use Case and Actors of the system.

Actors associated with the system are:

1. Admin
2. Reservation Clerk
3. Passenger
4. Printer

Use Cases for the system are:

1. Reserve Seat
2. Cancel Reservation
3. Update Train Information
4. Report Generation
5. Login
6. View Reservation Status
7. View Train Schedule

A sample Use Case Diagram is shown in *Figure 2:*

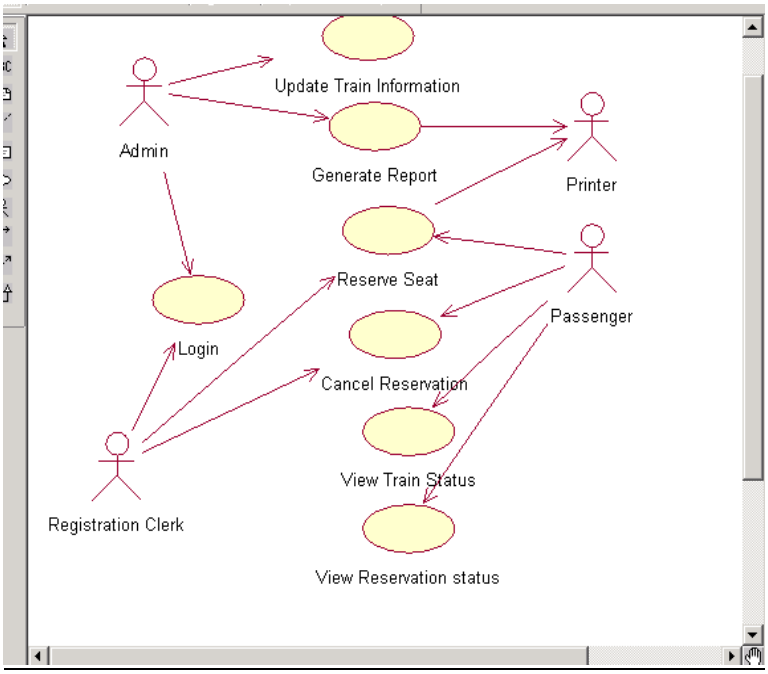


Figure 2: Use case diagram for railway reservation system

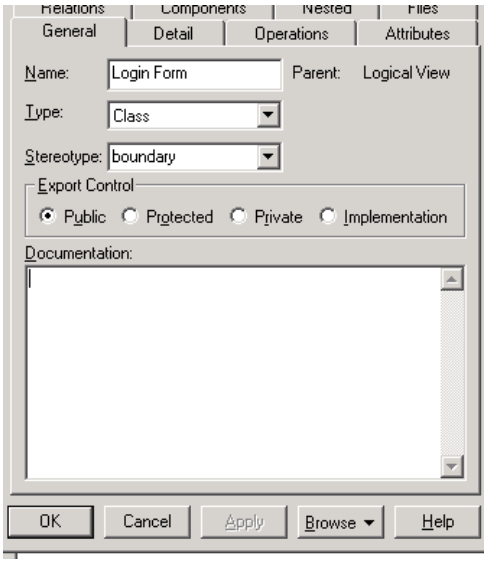
Step 3: Define Classes and associate them with each Use Case.

We will consider the **Login Use Case** and describe its complete flow.

For realisation of this Use Case we need a user interface for entering the user id and password. This needs to be validated with the database so we need an entity class and for coordinating the working between the front-end and back-end we need a controller class. We need to assign a name for each of these classes.

Type of Class		Name of Class
Boundary Class	:	Login Form
Entity Class	:	Login_Information (Name of table in the database)
Control Class	:	Login_Ctrl

Now create the new classes and give a name to the class. A sample login class is shown below:



Step 4: Depict the interaction between the classes with a Sequence and Collaboration Diagram.

So, now create the new sequence diagram, give a name to it. Most of the tools allow use of drag and drop the classes on the sequence diagram. Depict the flow of login with the help of this diagram. Now this diagram can be linked to construct a Collaboration Diagram.

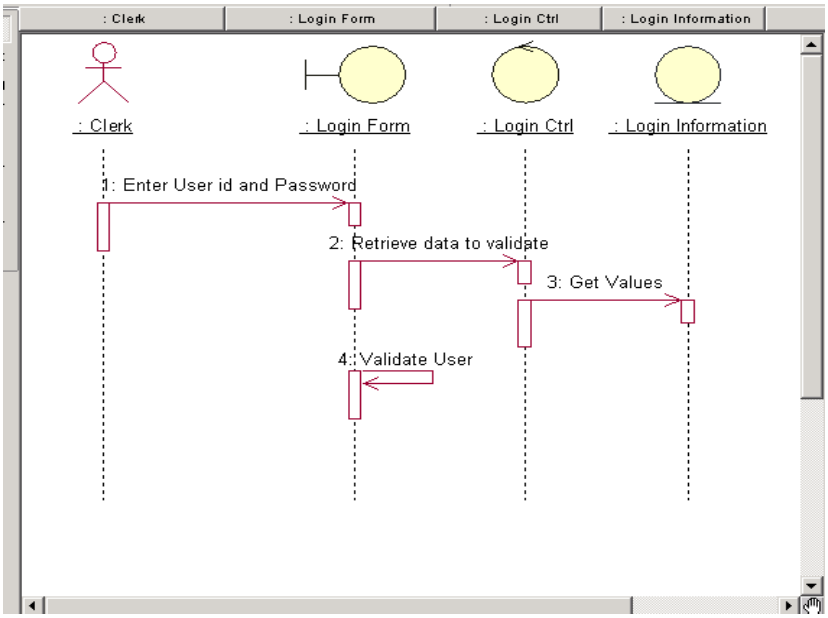


Figure 3: Sequence diagram

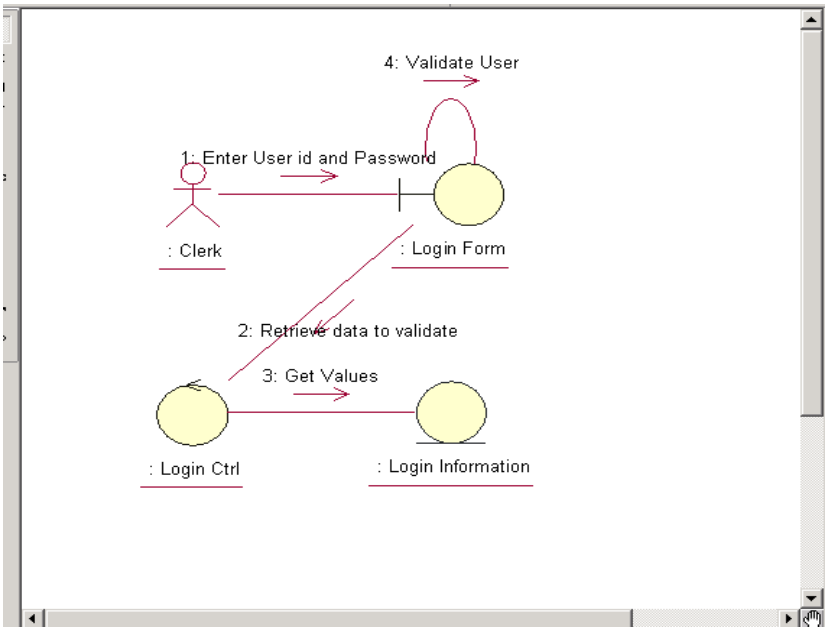


Figure 4: Collaboration diagram

Step 5: Draw Activity Diagram and State Chart Diagram if required. These diagrams should be names and linked to other related diagrams.

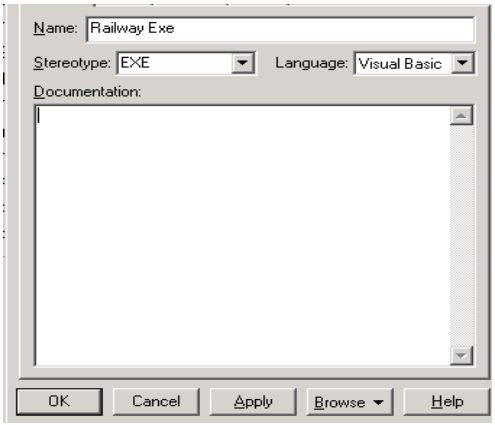
Step 6: Draw Class Diagram for all the Entity classes depicting relationships between the classes. Attributes are defined for each class. You must give proper names to these diagrams. Mark each class as persistent. They are the database tables where data is to be stored.

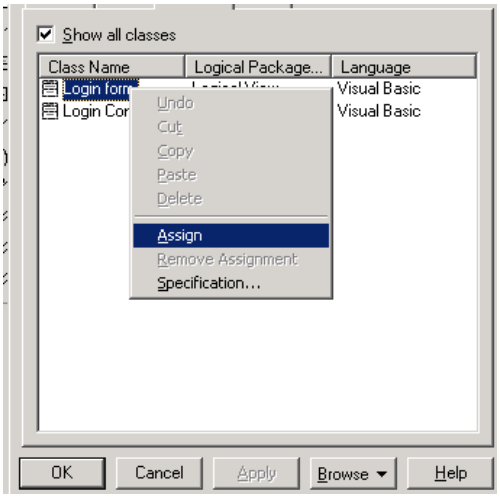
Step 7: Define the components of the system, for example, exes, dlls, activeX controls etc.

Suppose, you wish to develop the software as an executable, so all the boundary and control classes are assigned to the exe.

Create an executable software component and give it a name. This component must be assigned to class.

Then, draw a component diagram to show the relations between the components of the system. You must give a name to the diagram.





Step 8: Now, draw a deployment diagram for depicting the network layout of the system.

Create the Deployment diagram and give it a name.

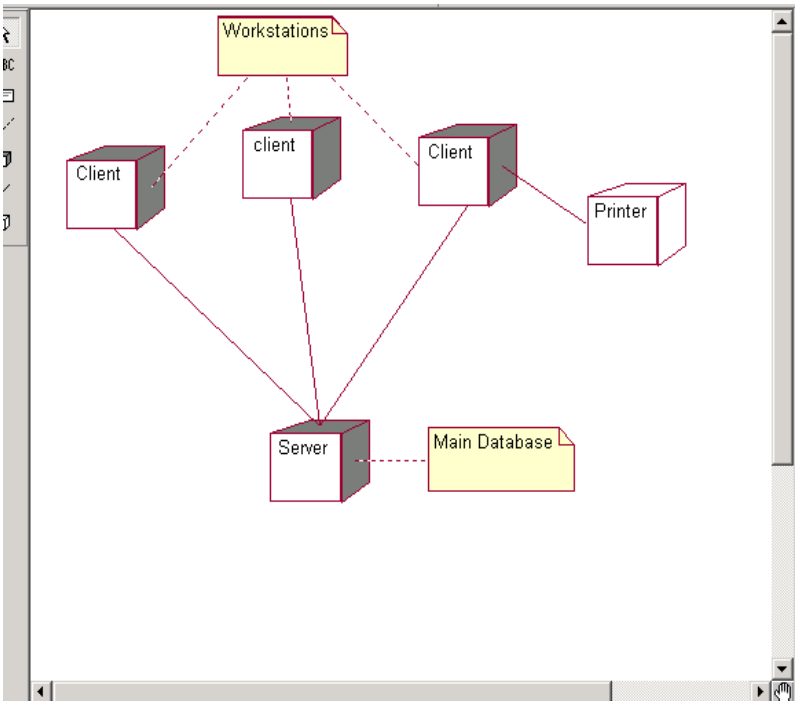


Figure 5: Deployment diagram

Step 9: Now, perform the task of forward engineering to generate pseudo code.

1.5 SESSION DETAILS

The following are some of the proposed problems that may be performed during the practical on this course. Please note that for these problems you may make suitable assumptions, if any. However state the assumptions categorically. It is also advisable to document the logic/ reasons of your analysis and design.

Sessions 1, 2 and 3:

1) Draw the Use Cases and define all the classes for Employee Management System.
2) Draw the Use Cases and define all the classes for Automatic Teller Machine.
3) Draw the Use Cases and define all the classes for Library Management System.
4) Draw the Use Cases and define all the classes for Order Processing Application.
5) Draw the Use Cases and define all the classes for Student Registration Process.

Sessions 4 and 5:

6) Draw the Sequence and Collaboration diagrams for Automatic Teller Machine.
7) Draw the Sequence and Collaboration diagrams for Employee Management System.
8) Draw the Sequence and Collaboration diagrams for Library Management System.
9) Draw the Sequence and Collaboration diagrams for Order Processing Application.
10) Draw the Sequence and Collaboration diagrams for Student Registration Process.

Sessions 6, 7 and 8:

11) Draw the state Transition Diagrams and Class Diagrams for Automatic Teller Machine.
12) Draw the state Transition Diagrams and Class Diagrams for Employee Management System.
13) Draw the state Transition Diagrams and Class Diagrams for Library Management System.
14) Draw the state Transition Diagrams and Class Diagrams for Order Processing Application.
15) Draw the state Transition Diagrams and Class Diagrams for Student Registration Process.

Sessions 9 and 10:

16) Draw the Component Deployment Model for Automatic Teller Machine.
17) Draw the Component Deployment Model for Employee Management System.
18) Draw the Component Deployment Model for Library Management System.
19) Draw the Component Deployment Model for Order Processing Application.
20) Draw the Component Deployment Model for Student Registration Process.

1.6 SUMMARY

This practical section covers aspects related to Object Oriented Analysis and Design. In this section, we have presented a very brief account of Object Oriented Programming, UML and OOAD. The process of development of an Object Oriented application in step by step fashion was also discussed. Finally, we have presented a session-wise list of the problems that you should try to solve during your practical sessions. You must implement all the diagrams in the tool that is provided at your study centre.