SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
(Established under Section 3 of the UGC Act, 1956)
Accredited with 'A' Grade by NAAC
Rajiv Gandhi Road, Jeppiaar Nagar Chennai – 600 119, Tamilnadu, India.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCS4201 OBJECT ORIENTED ANALYSIS AND SYSTEM ENGINEERING

LAB MANUAL

Ex.No	Date	Exercise Name	page no	Date of submission	marks	Staff signature with date
		Study Experiment				
1(a)		Paraphrase Object Oriented Concepts				
1(b)		Paraphrase Unified Modeling Language and , Rational Rose				
2		Implement usecase and interaction diagrams for Student mark analysis system				
3		Present collaboration diagrams for ATM system				
4		Illustrate the Exam Registration system with an appropriate design				
5		Explain the design for E-ticketing System				
6		Generate UML diagrams using CASE tools for Online course reservation system				
7		Review Software personnel management system using CASE tools				
8		Build a design for Credit card processing system				
9		Collaborate all the UML diagrams for e-book management system				
10		Build a detailed design for Recruitment system				
11		Simulate Foreign trading system using tools				
12		Simulate a BPO management system using CASE tools				

EX.NO: 1(a) DATE:	PARAPHRASE OBJECT ORIENTED CONCEPTS

Aim:
To study about the object oriented concepts.

Description:

- Basic OO Concepts

Introduction

- Goal of OO: Reduce complexity of software development by keeping details, and especially changes to details, from spreading throughout the entire program.
- This presentation assumes "Basic Class Design" presentation.
- ▼ Definitions
  - *Client Code - the code that uses the classes under discussion.*
  - ▼ *Coupling - code in one module depends on code in another module*
- ▼ Four key concepts of OOP
  - ▼ *Abstraction - responsibilities (interface) is different from implementation*
    - Distinguish between interface and its behavior vs. the implementation and its details
    - A class provides some services, takes on some responsibilities, that are defined by the public interface. How it works inside doesn't matter.
    - Client should be able to use just the public interface, and not care about the implementation.
  - ▼ *Encapsulation - guarantee responsibilities by protecting implementation from interference*
    - Developer of a class can guarantee behavior of a class only if the internals are protected from outside interference. Specifying private access for the internals puts a wall around the internals, making a clear distinction between which code the class developer is responsible for, and what the client developers are responsible for.
  - ▼ *Inheritance - share interface and/or implementation in related classes*

- Express commonalities between classes of objects by arranging them into an inheritance hierarchy. Allows functionality shared between classes to be written/debugged/maintained in one place, the base class, and then reused in descendant classes (shared implementation).
- More importantly, allows client code to interact with classes in the hierarchy through an interface specified in the base class of the hierarchy (shared interface).

Polymorphism - decouple client from exact class structure

- Allows client code to use classes in an class hierarchy in a way that depends only on the public interface of the base class. Derived classes will implement this interface differently, but in ways compatible with the shared public interface.

General Approach for Designing Classes

- Do the class design work at the level of the public interfaces, not the private implementations.
  - *Think only about what the class responsibilities are and what they do in their public interfaces:*
    - Class X is responsible for …., class Y for ….
    - When an X object needs … it calls the public member function … of the appropriate Y object with … as parameters, which returns …
  - *Try writing pseudo code just for the interactions between class objects through their public interfaces.*
  - *Keep this up until you can't stand it any more, then make implementation choices and write the code.*
- Continue design thinking until you have thought of at least two reasonable ways to solve each design problem.
- *All designs are imperfect - they all involve trade-offs. They are good in some ways, bad inothers.*

- *A good design is good in the most important ways, and bad in the less important ways.*
- *But there might be more than one good design - just different in the specific tradeoffs.*
- *You can't make an intelligent choice if you have only thought of one design - there could be another, better, simpler one.*

EX.NO: 1(b) **DATE:**	**PARAPHRASE UNIFIED MODELING LANGUAGE AND , RATIONAL ROSE**

AIM:

General study of UML

DESCRIPTION:

The heart of object-oriented problem solving is the construction of a model. The model abstracts the essential details of the underlying problem from its usually complicated real world. Several modeling tools are wrapped under the heading of the UML™, which stands for Unified Modeling Language™. The purpose of this course is to present important highlights of the UML.

At the center of the UML are its nine kinds of modeling diagrams, which we describe here.

- Use case diagrams
- Class diagrams
- Object diagrams
- Sequence diagrams
- Collaboration diagrams
- Statechart diagrams
- Activity diagrams
- Component diagrams
- Deployment diagrams

Some of the sections of this course contain links to pages with more detailed information. And every section has short questions. Use them to test your understanding of the section topic.

Why is UML important?

Let's look at this question from the point of view of the construction trade. Architects design buildings. Builders use the designs to create buildings. The more complicated the building, the more critical the communication between architect and builder. Blueprints are the standard graphical language that both architects and builders must learn as part of their trade.

Writing software is not unlike constructing a building. The more complicated the underlying system, the more critical the communication among everyone involved in creating and deploying the software. In the past decade, the UML has emerged as the software blueprint language for analysts, designers, and programmers alike. It is now part of the software trade. The UML gives everyone from business analyst to designer to programmer a common vocabulary to talk about software design.

The UML is applicable to object-oriented problem solving. Anyone interested in learning UML must be familiar with the underlying tenet of object-oriented problem solving -- it all begins with the construction of a model. A model is an abstraction of the underlying problem. The domain is the actual world from which the problem comes.Models consist of objects that interact by sending each other messages. Think of an object as "alive." Objects have things they know (attributes) and things they can do (behaviors or operations). The values of an object's attributes determine its state.

Classes are the "blueprints" for objects. A class wraps attributes (data) and behaviors (methods or functions) into a single distinct entity. Objects are instances of classes.

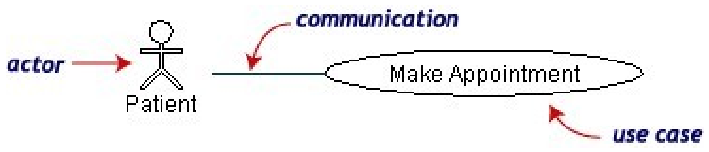
Use case diagrams

Use case diagrams describe what a system does from the standpoint of an external observer. The emphasis is on *what* a system does rather than *how.*

Use case diagrams are closely connected to scenarios. A scenario is an example of what happens when someone interacts with the system. Here is a scenario for a medical clinic.

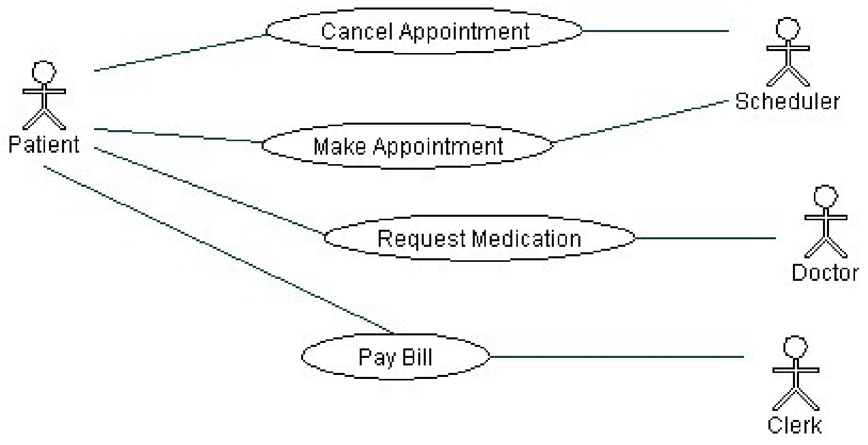
"A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot. "

A use case is a summary of scenarios for a single task or goal. An actor is who or what initiates the events involved in that task. Actors are simply roles that people or objects play. The picture below is a Make Appointment use case for the medical clinic. The actor is a Patient. The connection between actor and use case is a communication association (or communication for short).



Actors are stick figures. Use cases are ovals. Communications are lines that link actors to use cases.

A use case diagram is a collection of actors, use cases, and their communications. We've put Make Appointment as part of a diagram with four actors and four use cases. Notice that a single use case can have multiple actors.



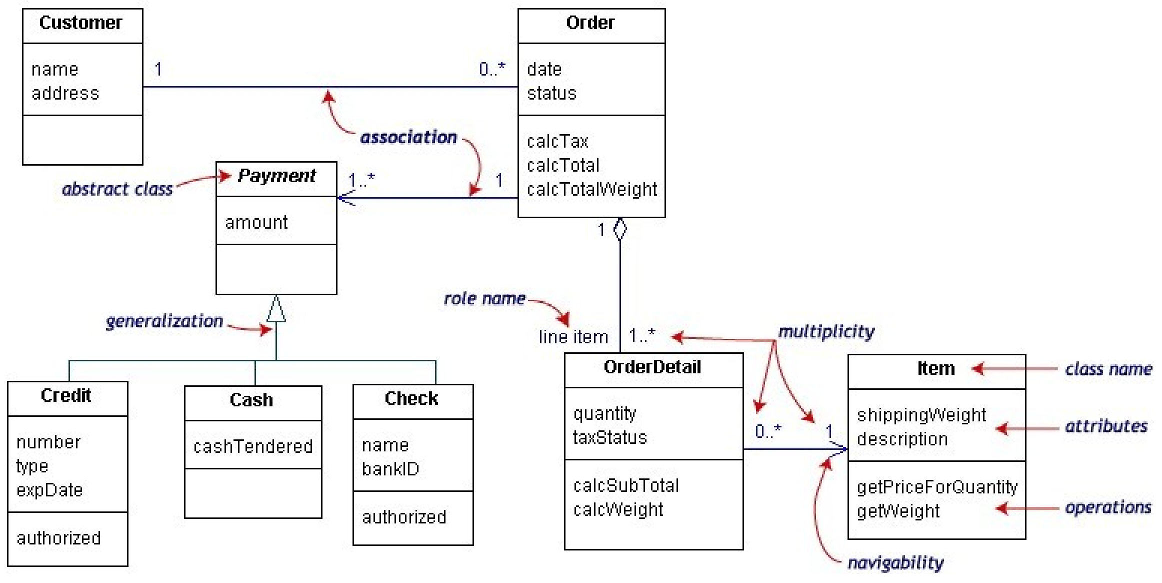
Use case diagrams are helpful in three areas.

- determining features (requirements). New use cases often generate new requirements as the system is analyzed and the design takes shape.
- communicating with clients. Their notational simplicity makes use case diagrams a good way for developers to communicate with clients.
- generating test cases. The collection of scenarios for a use case may suggest a suite of test cases for those scenarios.

Class diagrams

A Class diagram gives an overview of a system by showing its classes and the relationships among them. Class diagrams are static -- they display what interacts but not what happens when they do interact.

The class diagram below models a customer order from a retail catalog. The central class is the Order. Associated with it are the Customer making the purchase and the Payment. A Payment is one of three kinds: Cash, Check, or Credit. The order contains OrderDetails (line items), each with its associated Item.



UML class notation is a rectangle divided into three parts: class name, attributes, and operations. Names of abstract classes, such as *Payment,* are in italics. Relationships between classes are the connecting links.

Our class diagram has three kinds of relationships.

- association -- a relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other in order to perform its work. In a diagram, an association is a link connecting two classes.
- aggregation -- an association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole. In our diagram, Order has a collection of OrderDetails.
- generalization -- an inheritance link indicating one class is a superclass of the other. A generalization has a triangle pointing to the superclass. *Payment* is a superclass of Cash, Check, and Credit.

An association has two ends. An end may have a role name to clarify the nature of the association. For example, an OrderDetail is a line item of each Order.

A navigability arrow on an association shows which direction the association can be traversed or queried. An OrderDetail can be queried about its Item, but not the other way around. The arrow also lets you know who "owns" the association's implementation; in this case, OrderDetail has an Item. Associations with no navigability arrows are bi-directional.

The multiplicity of an association end is the number of possible instances of the class associated with a single instance of the other end. Multiplicities are single numbers or ranges of numbers. In our example, there can be only one Customer for each Order, but a Customer can have any number of Orders.

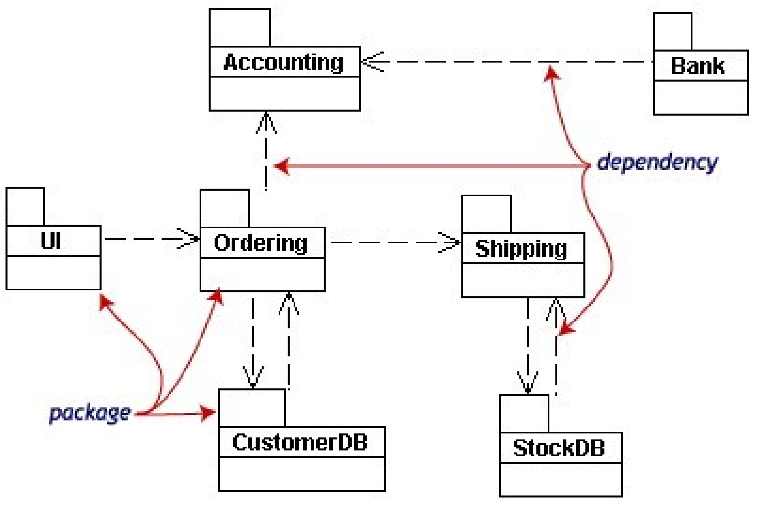
This table gives the most common multiplicities.

Multiplicities	Meaning
0..1	zero or one instance. The notation *n . . m* indicates *n* to *m* instances.
0..* *or* *	no limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance

Every class diagram has classes, associations, and multiplicities. Navigability and roles are optional items placed in a diagram to provide clarity.

Packages and object diagrams

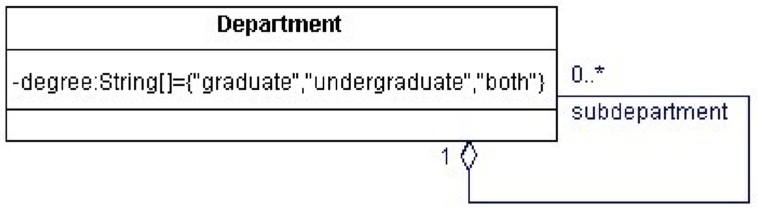
To simplify complex class diagrams, you can group classes into packages. A package is a collection of logically related UML elements. The diagram below is a business model in which the classes are grouped into packages.



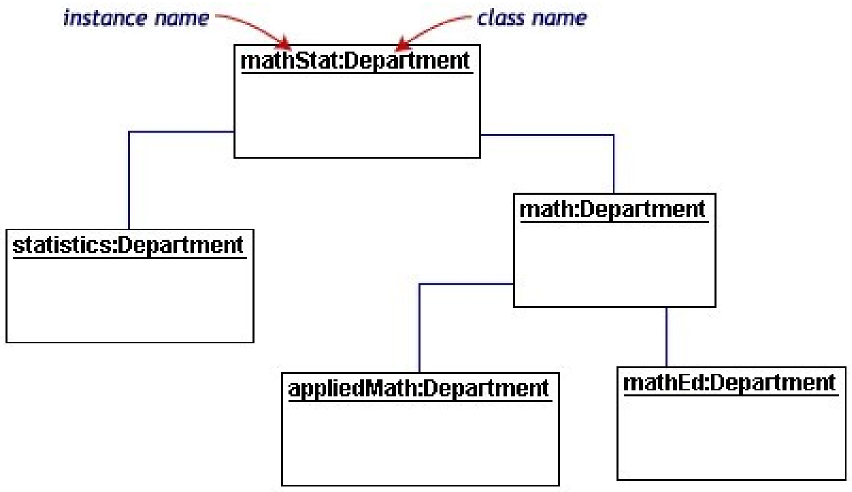
Packages appear as rectangles with small tabs at the top. The package name is on the tab or inside the rectangle. The dotted arrows are dependencies. One package depends on another if changes in the other could possibly force changes in the first.

Object diagrams show instances instead of classes. They are useful for explaining small pieces with complicated relationships, especially recursive relationships.

This small class diagram shows that a university Department can contain lots of other Departments.



The object diagram below instantiates the class diagram, replacing it by a concrete example.



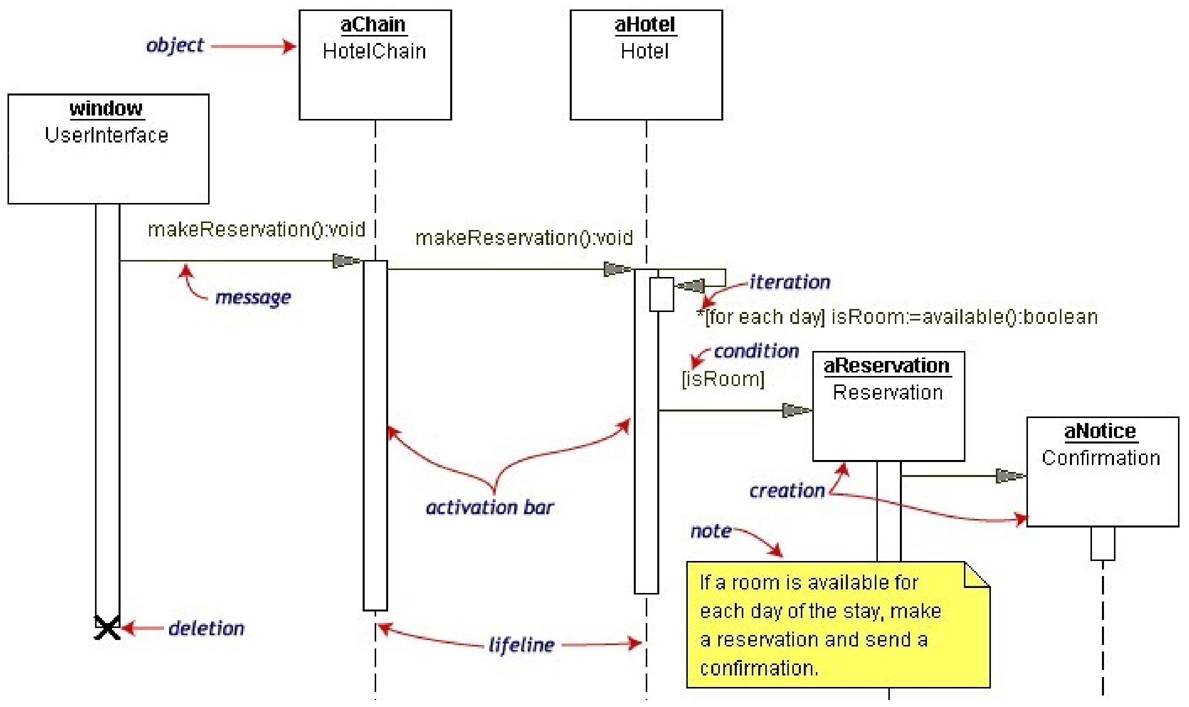
Each rectangle in the object diagram corresponds to a single instance. Instance names are underlined in UML diagrams. Class or instance names may be omitted from object diagrams as long as the diagram meaning is still clear.

Sequence diagrams

Class and object diagrams are static model views. Interaction diagrams are dynamic. They describe how objects collaborate.

A sequence diagram is an interaction diagram that details how operations are carried out -- what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

Below is a sequence diagram for making a hotel reservation. The object initiating the sequence of messages is a Reservation window.



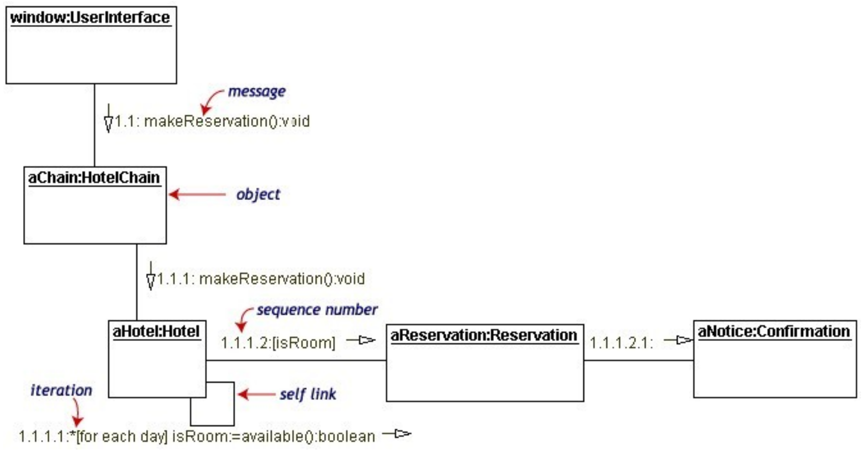
The Reservation window sends a makeReservation() message to a HotelChain. The HotelChain then sends a makeReservation() message to a Hotel. If the Hotel has available rooms, then it makes a Reservation and a Confirmation.

Each vertical dotted line is a lifeline, representing the time that an object exists. Each arrow is a message call. An arrow goes from the sender to the top of the activation bar of the message on the receiver's lifeline. The activation bar represents the duration of execution of the message.

In our diagram, the Hotel issues a self call to determine if a room is available. If so, then the Hotel creates a Reservation and a Confirmation. The asterisk on the self call means iteration (to make sure there is available room for each day of the stay in the hotel). The expression in square brackets, [ ], is a condition.

Collaboration diagrams

Collaboration diagrams are also interaction diagrams. They convey the same information as sequence diagrams, but they focus on object roles instead of the times that messages are sent. In a sequence diagram, object roles are the vertices and messages are the connecting links.



The object-role rectangles are labeled with either class or object names (or both). Class names are preceded by colons (:).

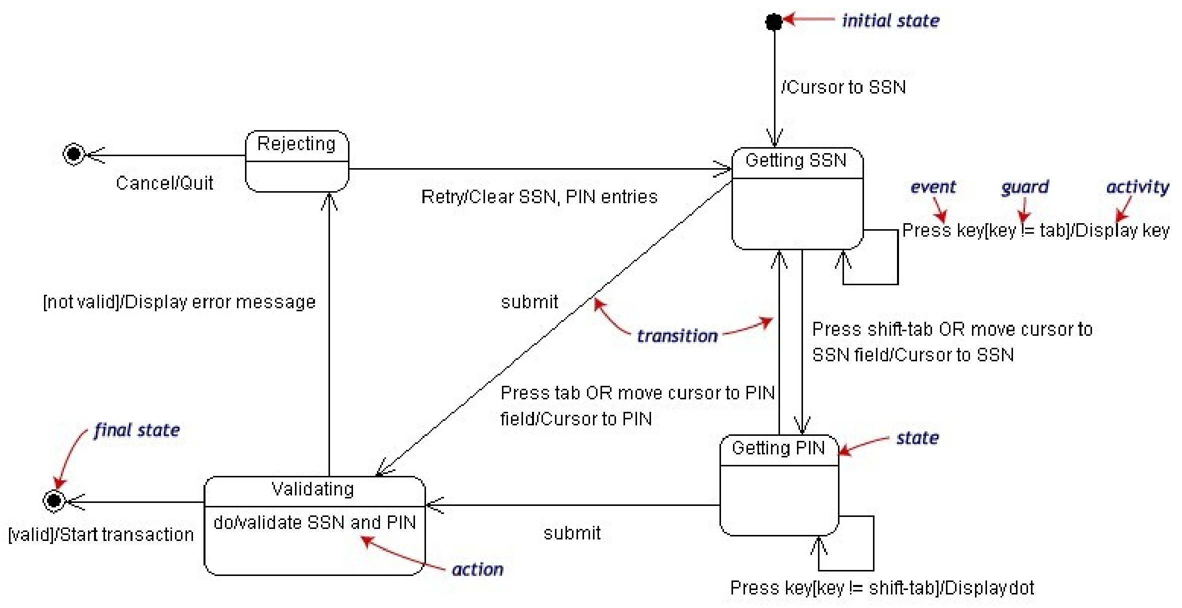
Each message in a collaboration diagram has a sequence number. The top-level message is numbered 1. Messages at the same level (sent during the same call) have the same decimal prefix but suffixes of 1, 2, etc. according to when they occur.

Statechart diagrams

Objects have behaviors and state. The state of an object depends on its current activity or condition. A statechart diagram shows the possible states of the object and the transitions that cause a change in state.

Our example diagram models the login part of an online banking system. Logging in consists of entering a valid social security number and personal id number, then submitting the information for validation.

Logging in can be factored into four non-overlapping states: Getting SSN, Getting PIN, Validating, and Rejecting. From each state comes a complete set of transitions that determine the subsequent state.


initial state
/Cursor to SSN
Rejecting
Getting SSN
Cancel/Quit
Retry/Clear SSN, PIN entries
event
guard
activity
Press key[key != tab]/Display key
[not valid]/Display error message
submit
transition
Press shift-tab OR move cursor to SSN field/Cursor to SSN
Press tab OR move cursor to PIN field/Cursor to PIN
final state
Getting PIN
state
Validating
do/validate SSN and PIN
[valid]/Start transaction
submit
action
Press key[key != shift-tab]/Display dot

States are rounded rectangles. Transitions are arrows from one state to another. Events or conditions that trigger transitions are written beside the arrows. Our diagram has two self-transition, one on Getting SSN and another on Getting PIN.

The initial state (black circle) is a dummy to start the action. Final states are also dummy states that terminate the action.

The action that occurs as a result of an event or condition is expressed in the form /action. While in its Validating state, the object does not wait for an outside event to trigger a transition. Instead, it performs an activity. The result of that activity determines its subsequent state.

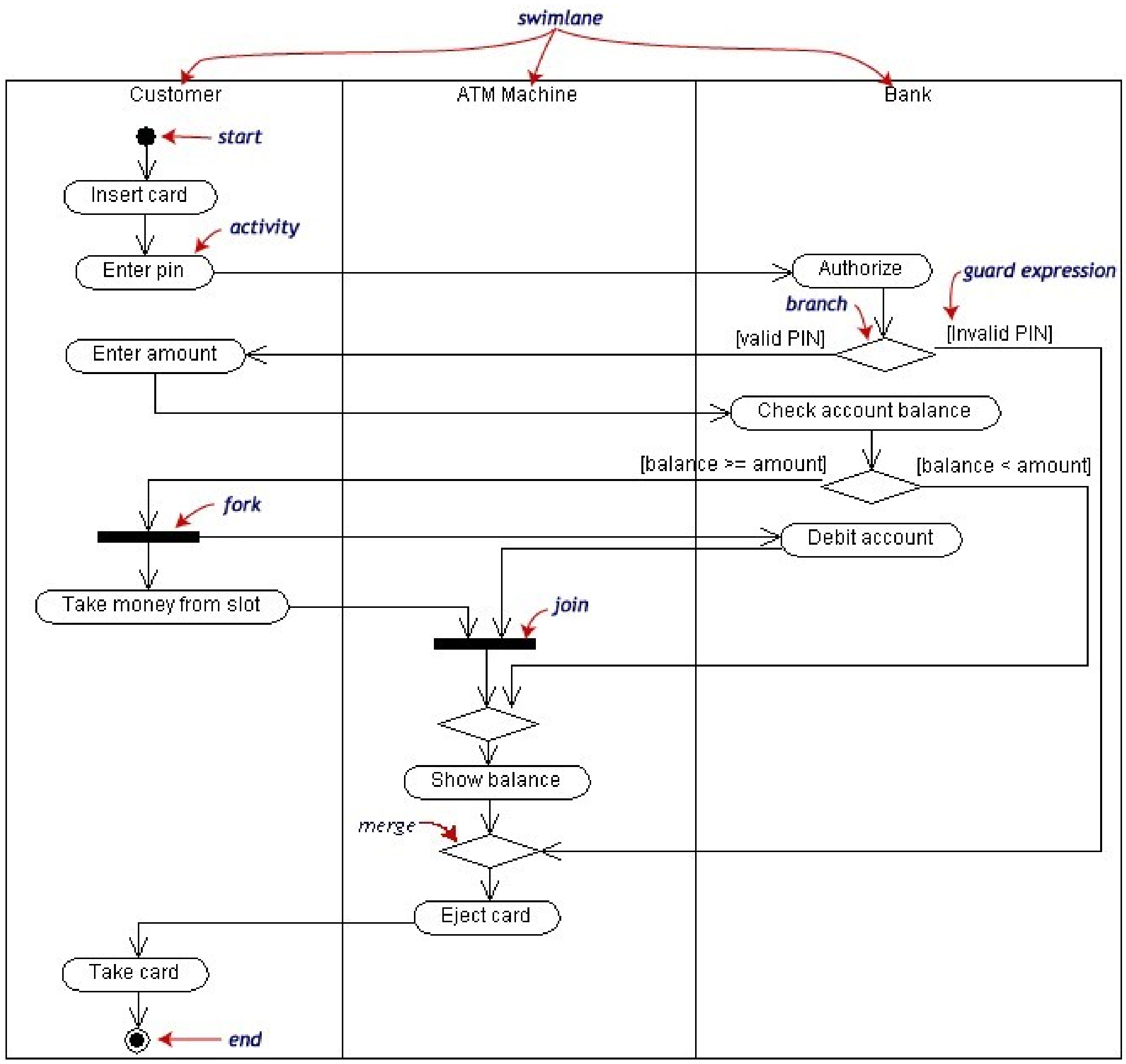
Activity diagrams

An activity diagram is essentially a fancy flowchart. Activity diagrams and statechart diagrams are related. While a statechart diagram focuses attention on an object undergoing a process (or on a process as an object), an activity diagram focuses on the flow of activities involved in a single process. The activity diagram shows the how those activities depend on one another.

For our example, we used the following process.

"Withdraw money from a bank account through an ATM."

The three involved classes (people, etc.) of the activity are Customer, ATM, and Bank. The process begins at the black start circle at the top and ends at the concentric white/black stop circles at the bottom. The activities are rounded rectangles.



Activity diagrams can be divided into object swimlanes that determine which object is responsible for which activity. A single transition comes out of each activity, connecting it to the next activity.

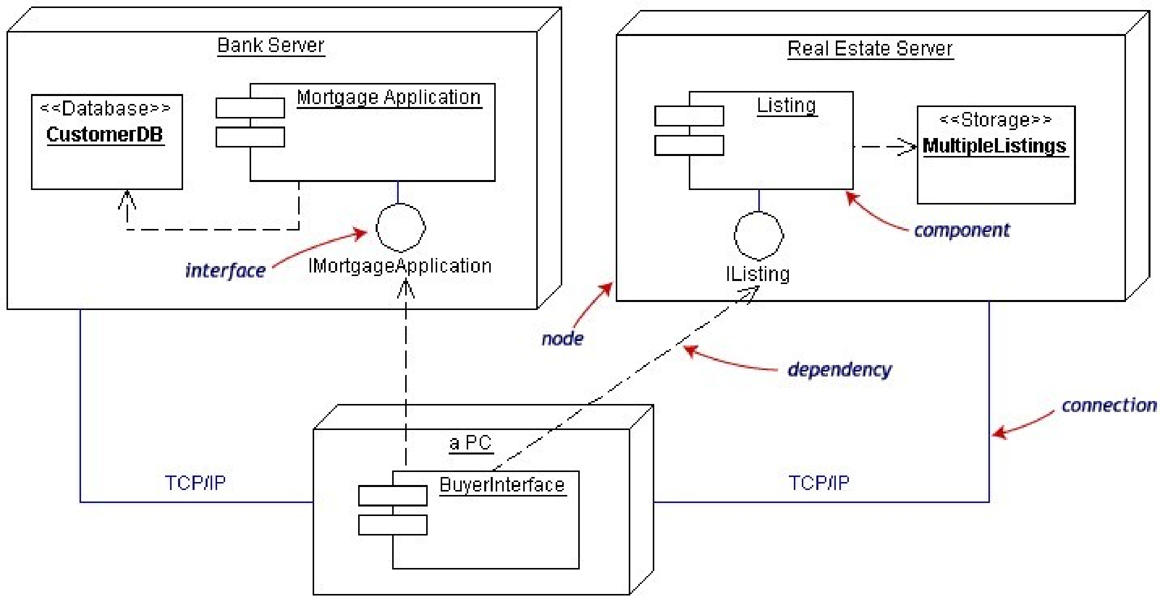
A transition may branch into two or more mutually exclusive transitions. Guard expressions (inside [ ]) label the transitions coming out of a branch. A branch and its subsequent merge marking the end of the branch appear in the diagram as hollow diamonds.

A transition may fork into two or more parallel activities. The fork and the subsequent join of the threads coming out of the fork appear in the diagram as solid bars.

Component and deployment diagrams

A component is a code module. Component diagrams are physical analogs of class diagram. Deployment diagrams show the physical configurations of software and hardware.

The following deployment diagram shows the relationships among software and hardware components involved in real estate transactions.



The physical hardware is made up of nodes. Each component belongs on a node. Components are shown as rectangles with two tabs at the upper left.

EX.NO: 2 **DATE:**	**IMPLEMENT USECASE AND INTERACTION DIAGRAMS FOR STUDENT MARK ANALYSIS SYSTEM**

AIM:

To draw the diagrams[usecase, sequence, collaboration] for the Student mark analysis system.

HARDWARE REQUIREMENTS:

SOFTWARE REQUIREMENTS:

PROJECT DESCRIPTION:

The purpose of this document is to define the requirements of mark analysis system. This system reduces manual work to great extent. The mark analysis is carried out by the system in an efficient manner.

USE CASE DIAGRAM:

IDENTIFY ACTORS:

IDENTIFY CASE:

SEQUENCE DIAGRAM

COLLABORATION DIAGRAM

EX.NO: 3 **DATE:**	**PRESENT THE COLLABORATION DIAGRAMS FOR ATM SYSTEM**

Aim: To draw the diagrams[usecase, activity, sequence, collaboration, class] for the ATM application

USE CASE DIAGRAM:

IDENTIFY ACTORS:

IDENTIFY CASE:

SEQUENCE DIAGRAM

COLLABORATION DIAGRAM

PACKAGE DIAGRAM

EX.NO: 4 **DATE:**	**ILLUSTRATE THE EXAM REGISTRATION SYSTEM WITH AN APPROPRIATE DESIGN**

AIM:

To draw the diagrams [use case, activity, sequence, collaboration, class] for the Exam registration system.

HARDWARE REQUIREMENTS:

SOFTWARE REQUIREMENTS:

PROJECT DESCRIPTION:

This software is designed for the verification of the details of the candidate by the central computer. The details regarding the candidate will be provided to the central computer through the administrator and the computer will verify the details of candidate and provide approval .Then the hall ticket will be issued from the office to the candidate..

USE CASE DIAGRAM:

ACTIVITY DIAGRAM:

CLASS DIAGRAM:

SEQUENCE DIAGRAM:

COLLABORATION DIAGRAM:

EX.NO: 5 **DATE:**	**ORGANIZE THE DESIGN FOR E-TICKETING SYSTEM**

AIM:

To draw the diagrams[use case, activity, sequence, collaboration, class] for the E-tickreting system.

HARDWARE REQUIREMENTS:

SOFTWARE REQUIREMENTS:

PROJECT DESCRIPTION:

This software is designed for supporting the computerized e-ticketing. This is widely used by the passenger for reserving the tickets for their travel. This E-ticketing is organized by the central system. The information is provided from the railway reservation system.

USE CASE DIAGRAM:

ACTIVITY DIAGRAM:

CLASS DIAGRAM:

SEQUENCE DIAGRAM:

COLLABORATION DIAGRAM:

EX.NO: 6	GENERATE UML DIAGRAMS USING CASE TOOLS FOR ONLINE COURSE RESERVATION SYSTEM
DATE:	

AIM:

To draw the diagrams[usecase, activity, sequence, collaboration, class] for the Online course reservation system.

HARDWARE REQUIREMENTS:

- Intel Pentium Processor 3

SOFTWARE REQUIREMENTS:

- Rational rose / VisualBasic

PROJECT DESCRIPTION:

This software is designed for supporting online course reservation system. This system is organized by the central management system . The student first browses and select the desired course of their choice. The university then checks the availability of the seat if it is available the student is enrolled for the course.

USE CASE DIAGRAM:

ACTIVITY DIAGRAM:

CLASS DIAGRAM:

SEQUENCE DIAGRAM:

COLLABORATION DIAGRAM:

EX.NO: 7 DATE:	REVIEW SOFTWARE PERSONNEL MANAGEMENT SYSTEM USING CASE TOOLS

AIM:

To draw the diagrams [usecase, activity, sequence, collaboration, class] for Software personnel management system

HARDWARE REQUIREMENTS:

- Intel Pentium Processor 3

SOFTWARE REQUIREMENTS:

- Rational rose / VisualBasic

PROJECT DESCRIPTION:

This software is designed for the process of knowing the details of a person works in a software company. The details are being stored in the central management system for the crosschecking the person's details.

USE CASE DIAGRAM:

ACTIVITY DIAGRAM:

CLASS DIAGRAM:

SEQUENCE DIAGRAM:

COLLABORATION DIAGRAM:

EX.NO: 8 **DATE:**	**BUILD A DESIGN FOR CREDIT CARD PROCESSING SYSTEM**

AIM:

To draw the diagrams [usecase, activity, sequence, collaboration, class] for Credit Card Processing

HARDWARE REQUIREMENTS:

- Intel Pentium Processor 3

SOFTWARE REQUIREMENTS:

- Rational rose / VisualBasic

PROJECT DESCRIPTION:

This software is designed for supporting the computerized credit card processing System .In this system, the cardholder purchases items and pays bill with the aid of the credit card. The cashier accepts the card and proceeds for transaction using the central system. The bill is verified and the items are delivered to the cardholder.

USE CASE DIAGRAM:

ACTIVITY DIAGRAM:

CLASS DIAGRAM:

SEQUENCE DIAGRAM:

COLLABORATION DIAGRAM:

TEMPLATE CODE

EX.NO: 9 **DATE:**	**Collaborate All The UML Diagrams For E-Book Management System**

AIM:

To draw the diagrams [usecase, activity, sequence, collaboration, class] for E-book management system

HARDWARE REQUIREMENTS:

- Intel Pentium Processor 3

SOFTWARE REQUIREMENTS:

- Rational rose / VisualBasic

PROJECT DESCRIPTION:

This software is designed to manage the books that were read through the internet. This consists of the details of the e-book that were read by the user online. It will be controlled by the central system. This system act as a backup of all details together.

USE CASE DIAGRAM:

ACTIVITY DIAGRAM:

CLASS DIAGRAM:

SEQUENCE DIAGRAM:

COLLABORATION DIAGRAM:

TEMPLATE CODE

EX.NO: 10 **DATE:**	**BUILD A DETAILED DESIGN FOR RECRUITMENT SYSTEM**

AIM:

To draw the diagrams [usecase, activity, sequence, collaboration, class] for Recruitment system

HARDWARE REQUIREMENTS:

SOFTWARE REQUIREMENTS:

PROJECT DESCRIPTION:

This system is designed to recruit the particular job to the person in a company .It was controlled by the central management system to manage the details of the particular candidate that one has to be recruited for a company.

USE CASE DIAGRAM:

ACTIVITY DIAGRAM:

CLASS DIAGRAM:

SEQUENCE DIAGRAM:

COLLABORATION DIAGRAM:

TEMPLATE CODE

Ex-11 **DATE:**	**SIMULATE FOREIGN TRADING SYSTEM USING CASE TOOLS**

AIM:

To draw the diagrams [usecase, activity, sequence, collaboration, class] for Foreign trading system

HARDWARE REQUIREMENTS:

SOFTWARE REQUIREMENTS:

PROJECT DESCRIPTION:

This software is designed to maintain the details about the trading system that exists between the foreign countries. This details are hold by the trading management system.The details to the system are provided by the customer and the supplier

USE CASE DIAGRAM:

ACTIVITY DIAGRAM:

CLASS DIAGRAM:

SEQUENCE DIAGRAM:

COLLABORATION DIAGRAM:

TEMPLATE CODE

NO:12 **DATE:**	**SIMULATE A BPO MANAGEMENT SYSTEM USING CASE TOOLS**

AIM:

To draw the diagrams [usecase, activity, sequence, collaboration, class] for BPO management system

HARDWARE REQUIREMENTS:

- Intel Pentium Processor 3

SOFTWARE REQUIREMENTS:

- Rationalrose

PROJECT DESCRIPTION:

This software is designed to know about the process that were taking place in the BPO office. This system holds the details of the customer who and all approaches to it. It is managed by the centralsystem..

USE CASE DIAGRAM:

ACTIVITY DIAGRAM:

CLASS DIAGRAM:

SEQUENCE DIAGRAM:

COLLABORATION DIAGRAM:

TEMPLATE CODE