AMRUTVAHINI COLLEGE OF ENGINEERING, SANGAMNER

Department: Information Technology

Name of Subject	**Laboratory Practice-II (Software Modeling Design)**	**Page**	**01/01**
Number of Practical's Plan		**Rev.: 00, Date:**	

Sr. No.	Description	Rev.	Date	Location of Controlled copy
IT/LP-II-D-01	General instructions to students	00		Notice board, Lab.
IT/LP-II-D-02	Laboratory instructions	00		Notice board Lab.
IT/LP-II-D-03	Do's and Don'ts	00		Notice board, Lab.
IT/LP-II-D-04	Maintenance schedule	00		Notice board, Lab.
IT/LP-II-D-05	Write Problem Statement and Draw Use Case Diagrams For Mini Project	00		Respective Lab
IT/LP-II-D-06	Prepare a use case model, from the given description using UML 2 notations.	00		Respective Lab
IT/LP-II-D-07	Prepare Activity Model from the given description using UML 2 notations.	00		Respective Lab
IT/LP-II-D-08	Prepare a class diagram from the given problem description using UML2.0 and Implement the class diagram with a suitable object oriented language.	00		Respective Lab
IT/LP-II-D-09	Prepare a Design Model from Analysis Model.	00		Respective Lab
IT/LP-II-D-10	Prepare a Sequence diagram from the given problem description using UML2.0 and Implement the Sequence diagram with a suitable object oriented language.	00		Respective Lab
IT/LP-II-D-11	Prepare a state model from the given problem description, draw a state diagram using UML2 notations and Implement the state model with a suitable object oriented language.	00		Respective Lab
IT/LP-II-D-12	Identification and Implementation of GRASP pattern.	00		Respective Lab
IT/LP-II-D-13	Identification and Implementation of GOF pattern.	00		Respective Lab
IT/LP-II-D-14		00		Respective Lab
IT/LP-II-D-15		00		Respective Lab

PREPARED BY APPROVED BY CONTROLLED COPY STAMP MASTER COPY STAMP

AMRUTVAHINI COLLEGE OF ENGINEERING, SANGAMNER

IT/LP-II-D-16		00		Respective Lab
Note: **IT** is the short form of department, **LP-II** indicates the short form for name of subjects				

PREPARED BY APPROVED BY CONTROLLED COPY STAMP MASTER COPY STAMP

IT/LP-II-D-01	**General instructions to students**	**Page**	**01/01**
		Rev.: 00	**Date:**

Students are advised to follow instructions given before they begin work in the laboratory.

1. Before coming to perform the experiment go through the basic theory of the experiment at the home.
2. Keep your working place clean and tidy. A dirty seat leaves a poor impression about the student working there.
3. If necessary the Computers should be cleaned before performing the experiment.
4. Always carry your journal with you in the laboratory. The student without journalis as incomplete as a pen without ink.
5. The records made in the journal must be clear, concise and self-explanatory.
6. All the observations/outputs should be carefully entered in the journal.
7. Do not misplace any instrument like mouse, keyboard, monitor, cpu and chairs.
8. Always put date on which experiment is performed in the laboratory.
9. Journal should be maintained properly in all respect.
10. Avoid unnecessary gossiping, whispering, and / or moving in the laboratory.
11. It is once again emphasized that honest and regular work in laboratory is more important than good results. Those students, who try to change their observation for the sake of getting a accurate result, do not work with the true spirit of a student seeking to cultivate a scientific brain.
12. While implementing programs save your program time to time to avoid losing dueto power failures.
13. Save your program with your storage device before leaving the lab.
14. After completion of the program immediately take printouts.
15. Put your bags towards the notice board.

PREPARED BY APPROVED BY CONTROLLED COPY STAMP MASTER COPY STAMP

AMRUTVAHINI COLLEGE OF ENGINEERING, SANGAMNER

IT/LP-II-D-02	**Laboratory Instructions**	**Page**	**01/01**
		Rev.: 00	**Date:**

- ➢Remove your shoes outside.
- ➢Keep the chairs properly
- ➢Shutdown machine properly while leaving Lab
- ➢Without permission of Lab In Charge do notdisturb the setting of experiments
- ➢Please keep silence.
- ➢Student should maintain their backup with them

PREPARED BY APPROVED BY CONTROLLED COPY STAMP MASTER COPY STAMP

IT/LP-II-D-03	**Do’s and Don’ts**	Page	01/01
		Rev.: 00	Date:

Do’s:

- Switch of Computer properly.
- Remove your shoes outside the Lab.
- After completion of practical arrange the chair properly.
- Keep silence in the Lab
- Make entry in a log book.

Don’ts:

- Don’t delete the back-up data of other student.
- Don’t open the other software apart from your practical.
- Don’t enter in the lab when other practical is going on.
- Don’t enter in the lab without permission.
- Don’t move the hardware from one PC to another PC.

AMRUTVAHINI COLLEGE OF ENGINEERING, SANGAMNER

IT/LP-II-D-04	**Maintenance schedule**	**Page**	**01/01**
		Rev.: 00	**Date:**

Proposed dates for pre-session maintenance	**Proposed dates for post-session maintenance**	**Frequency of pre-session**	**Frequency of post-session.**
One week before the Start of Semester	After the End of Term	Ones	Ones

PREPARED BY APPROVED BY CONTROLLED COPY STAMP MASTER COPY STAMP

IT/LP-II: D-05	Write Problem Statement and Draw Use Case Diagrams For Mini Project	Page	1/5
Experiment No: 1	Semester – VI	Rev.: 00	Date:

Practical No. 1

Title of Practical: Write Problem Statement for System / Project.

Objective of the Assignment: Identify Project of enough complexity, which has at least 4-5 major functionalities. Identify stakeholders, actors and write detail problem statement for your system.

Prerequisite: Basic of Software Engineering, Requirement Gathering, Relationships and Diagram

Theory:

Identifying Requirements

Requirements in this context are the conditions that a proposed solution or application must meet in order to solve the business problem. Identifying the requirements is not an exclusively technical process, and initially involves all the stakeholders, like the representatives of the entity that has commissioned the software project, whomay not necessarily be from a technical background, as well as the software developers, who are necessarily the technical team. Together, they discuss and brainstorm about the problem, and decide what functions the proposed application or system must perform in order to solve it.

Functional vs. Non-Functional Requirements

Functional requirement specifies something that the application or system should do. Often this is definedas a behaviour of the system that takes input and provides output. For example, a traveller fills out a form in an airline's mobile application with his/her name and passport details (input), submits the form, and the

and the application generates a boarding pass with the traveller's details (output).

Non-functional requirements, sometimes also called quality requirements, describe how the system should be, as opposed to what it should do. Non-functional requirements of a system include performance (e.g. response time), maintainability, and scalability, among many others. In the airline application example, the requirement that the application must display the boarding pass after a maximum of five seconds from the time the traveller presses the 'submit' button would be a non-functional requirement.

What is effective Problem Statement?

In project management, the problem statement is part of the project charter and defines what the problem is so that they the project team and stakeholder can focus their attention on solving the problem. It is important to have a good problem statement before starting eliciting requirements for a solution.

IT/LP-II: D-05	Write Problem Statement and Draw Use Case Diagrams For Mini Project	Page	2/5
Experiment No: 1	Semester – VI	Rev.: 00	Date:

Problem Statement: Digital Library Management System:

1. Overall Description

1.1. Product Perspective

WLMS is a replacement for the ordinary library management systems which depend on paper work for recording book and users' information. WLMS will provide an advanced book search mechanism and will make it easy to borrow, insert and index a book in the library.

1.2. Product Functions

1.2.1 Administrators

1. Admin should be able to insert, modify and delete books.
2. Can accept or reject a new user according to the library policy or payment methods.
3. Increase the period for borrowing a book for specific type or group of users.
4. Can get the information (status report) of any member who has borrowed a book.
5. Add and edit book categories and arrange books by categories.
6. Add and edit authors and publisher's information.
7. Can send lateness warnings to people who have exceeded deadline date.
8. Can record books returned by users?

1.2.2 Normal Users (Library Members)

1. The member should be provided with the updated information about the books catalog.
2. Members are given a provision to check their account's information and change it.
3. Members have the ability to search through books by subject, title, authors or any informationrelated to the book.
4. Can extend the period of borrowing books according to the library policy.
5. The customer may suggest a book to be brought to the library book collection.

1.3. Operating Environment

The WLMS is a website and shall operate in all famous browsers, for a model we are taking Microsoft Internet Explorer versions 7.0, 8.0 and 9.0, with Flash Player 9 and JavaScript.

1.4 User Characteristics

Users of the website are members, librarians and the administrators who maintain the website. Members and librarians are assumed to have basic knowledge of computers and Internet browsing. Administrators of the system should have more knowledge of internal modules of the system and are able to rectify small problems that may arise due to disk crashes, power failures and other catastrophes. Friendly user interface, online help and user guide must be sufficient to educate the users on how to use this product without any problems or difficulties.

IT/LP-II: D-05	Write Problem Statement and Draw Use Case Diagrams For Mini Project	Page	3/5
Experiment No: 1	Semester – VI	Rev.: 00	Date:

1.5 Design and Implementation Constraints

1. The information of all users, books and libraries must be stored in a database that is accessibleby the website.

2. MS SQL Server will be used as SQL engine and database.
3. The Online Library System is running 24 hours a day.

   Users may access WLMS from any computer that has Internet browsing capabilities and an Internet connection.
4. Users must have their correct usernames and passwords to enter into their online accounts and do actions.

2. Specific Requirements

2.1 Functional Requirements

Sr. No.	Requirement	Associated User	Description
1	Insert Book	Librarian	This action is done to add new book to library bookcollection.
2	Delete / Modify Book	Librarian	this event is to delete an existing book or modify its information.
3	Validate User Account	Librarian	when a new member sign up then he should wait for acceptance by Administrator according to library policies
4	Delete Member	Librarian	Admin can delete a member due to some specific rules
5	Modify Member Rank	Librarian	Admin can extend the borrowing time or number of book borrowed simultaneity to a user
6	Return Book	Librarian	Admin should confirm the return of books borrowed by users.
7	Register	User	when new user enters WLMS for the first time then he hasto register
8	Issue & Return Book	User	Member can Issue and Return the Book
9	Reset Password	User	when a member forgets his password he can claim it backvia e-mail.
10	Edit Personal Information	User	if some user changes for example his mobile number, he can modify it.
11	Reset Password	User	when a member forgets his password he can claim it backvia e-mail.
12	Search For Book	User	when user or admin wants to search on some book by name, author or subject etc.

IT/LP-II: D-05	Write Problem Statement and Draw Use Case Diagrams For Mini Project	Page	4/5
Experiment No: 1	Semester – VI	Rev.: 00	Date:

2.2 Non-functional Requirements

2.2.1 Error handling

➢ WLMS product shall handle expected and non-expected errors in ways that prevent loss ininformation and long downtime period.

2.2.2 Performance Requirements

➢ The system shall accommodate high number of books and users without any fault.
➢ Responses to view information shall take no longer than 5 seconds to appear on the screen.

2.2.3 Safety Requirements

➢ System use shall not cause any harm to human users.

2.2.4 Security Requirements

➢ System will use secured database
➢ Normal users can just read information but they cannot edit or modify anything except theirpersonal and some other information.
➢ System will have different types of users and every user has access constraints.

2.3 Software and hardware Requirement (Minimum)

2.3.1 Software Requirement:

➢ **Platform:** Windows XP
➢ **Front End:** Java Script
➢ **Backend** My Sql
➢ **Web Browser:** Internet Explorer8

2.3.2 Hardware Requirement:

➢ **Processor:** Pentium4
➢ **RAM:** 1 GB
➢ **Storage:**250 GB

3.Oral Questions

1. What do u mean by Functional and Non Functional Requirement?
2. What is Security Requirement.
3. Why Error Handling is important.
4. Why Consideration of Assumption and dependencies is important.

4.Assignment Question

Study Various template for writing the problem statement and for preparing the Softwarerequirement specifications.

IT/LP-II: D-05	**Write Problem Statement and Draw Use Case Diagrams For Mini Project**	**Page**	**5/5**
Experiment No: 1	**Semester – VI**	**Rev.: 00**	**Date:**

Conclusion: This document will help the software engineer to understand the Purpose, Scope, various functional and nonfunctional requirement of the system as well the software and hardware requirement of the system which can help him to plan accordingly for development of the system.

IT/LP-II: D-06	**Prepare a use case model, from the given description using UML 2 notations.**	**Page**	**1/4**
Experiment No: 2	**Semester – VI**	**Rev.: 00**	**Date:**

Practical No. 2

Title of the Practical: Prepare a use case model, from the given description using UML 2 notations.

Objective of the Assignment: Identify Major Use Cases, Identify actors. Write Use Case specificationfor all major Use Cases. Draw detail Use Case Diagram using UML2.0 notations.

Prerequisite: Basic UML Notations and UML Things, Relationships and Diagrams.

Theory:

1. Use Case Diagram

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system. Use case diagrams are in fact twofold - they are both behavior diagrams, because they describe behavior of the system, and they are also structure diagrams - as a special case of class diagrams where classifiers are restricted to be either actors or use cases related to each other with associations.

1.1 Purpose of use case diagram:

- 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.

1.2 Contents of use case diagram:

- Use Case
- Actors
- Relationships among the use cases and actors.

1.2.1 **Use Cases:** Use cases are nothing but the system functionalities written in an organized manner. So when the requirements of a system are analyzed the functionalities are captured in use cases.

1.2.2 **Actors:** Actors can be defined as something that interacts with the system. The actors can be humanuser, some internal applications or may be some external applications.

1.2.3 **Include Relationship:** An include relationship between use cases means that the base case explicitly incorporates the behavior of another use case at a location specified in the base. The included use case never stands alone, but is only instantiated as part of some larger base that

IT/LP-II: D-06	**Prepare a use case model, from the given description using UML 2 notations.**	Page	2/4
Experiment No: 2	Semester – VI	Rev.: 00	Date:

includes it.

Include relationship between use cases is shown by a dashed arrow with an open arrowhead from theincluding (base) use case to the included (common part) use case. The arrow is labeled with the keyword «include».

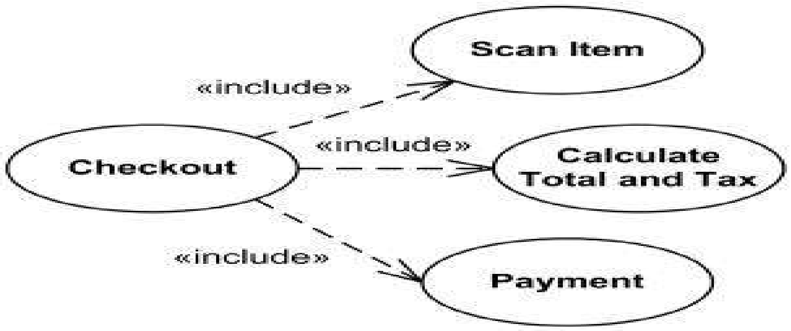


Fig1. Checkout use case includes several use cases - Scan Item, Calculate Total and Tax, and Payment

1.2.4 Extended Relationship: An extend relationship between use cases means that the base use case implicitly incorporates the behaviour of another use case at a location specified indirectly by the extending use case. The base use case may stand alone, but under certain conditions, its behaviour may be extended by the behaviour of another use case. This base use case may be extended only at certain points called as extension points. We use an extend relationship to model the part of use case the user may see as optional behaviour. As the name implies it extends the base use case and adds more functionality to the system.

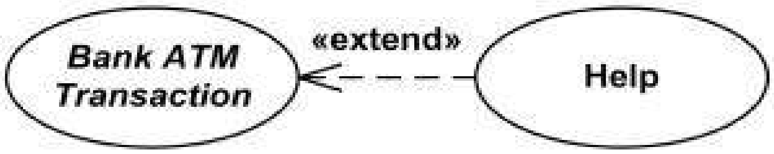


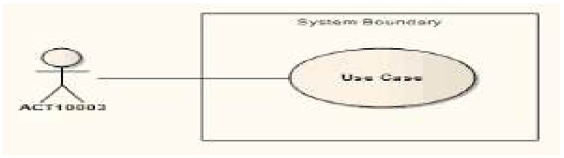
Fig 2 Bank ATM Use case with extended help

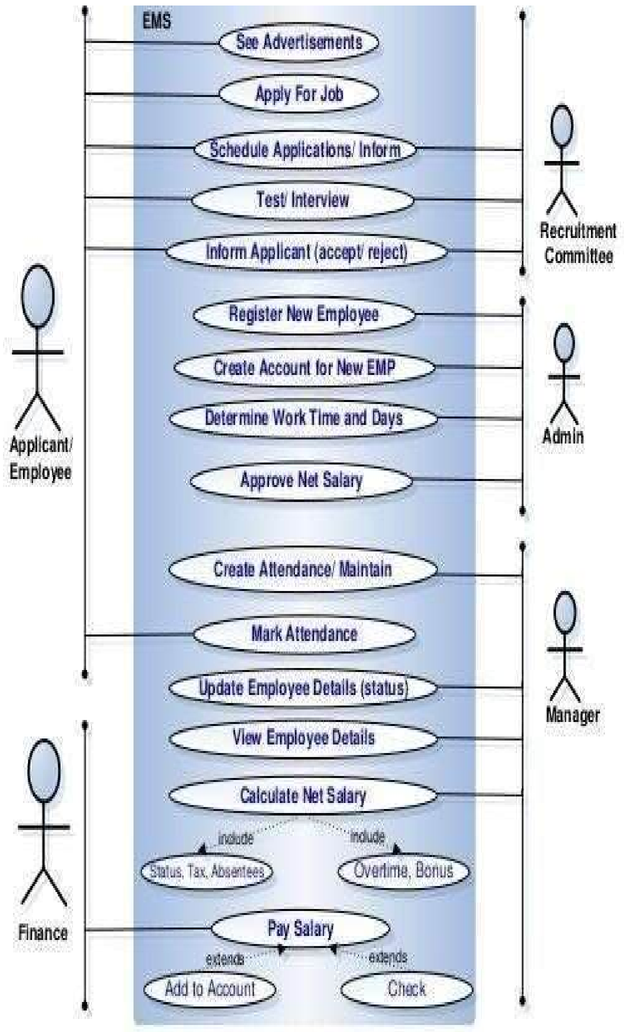
1.2.5 Generalization Relationship: A generalization relationship is also a parent-child relationship between use cases. The child use case in the generalization relationship has the underlying business processmeaning, but is an enhancement of the parent use case. In a use case diagram, generalization is shown as a directed arrow with a triangle arrowhead. The child use case is connected at the base of the arrow. The tipof the arrow is connected to the parent use case.

1.2.6 **System Boundary:** A System Boundary element is a non-UML element used to define conceptual boundaries. You can use System Boundaries to help group logically related

IT/LP-II: D-06	Prepare a use case model, from the given description using UML 2 notations.	Page	3/4
Experiment No: 2	Semester – VI	Rev.: 00	Date:

1.2.7 elements (from a visual perspective, not as part of the UML model).





IT/LP-II: D-06	Prepare a use case model, from the given description using UML 2 notations.	Page	4/4
Experiment No: 2	Semester – VI	Rev.: 00	Date:

4. Oral Questions:

1. What do you mean be use case, actor? Draw their graphical notations.
2. Explain with example include and extend.
3. What are the various components of use case diagram?
4. What is the importance of use case diagram?

5. Assignment Questions:

Prepare use case diagram for library management, railway reservation and online shopping.

Conclusion: We prepared a use case model, which describe us the various user and functionalities associated with the system also it describes the various roles and responsibilities of each user in a system.

IT/LP-II: D-07	Prepare Activity Model from the given description using UML 2 notations.	Page	1/4
Experiment No: 3	Semester – VI	Rev.: 00	Date:

Practical No. 3

Title of the Practical: Prepare Activity Model from the given description using UML 2 notations.

Objective of the Assignment: Identify Activity states and Action states. Draw Activity diagram with Swim lanes using UML2.0 Notations for major Use Cases.

Prerequisite: Basic UML Notations and UML Things, Relationships and Diagrams.

Theory:

1. **Activity Diagram:**
   Activity diagrams are used to model the dynamic aspects of a system. An activity diagram is essentially a flow chart showing flow of central from activity to activity. An Activity diagram shows flow from activity to activity. An activity is a non-atomic execution within a state machine.
2. **Purpose of Activity diagrams:** Activity diagrams are used to typically two ways:
   1. **To model a work flow**: Work flow often lie on fringe of software intensive systems and are used tovisualize, specify, construct and document business processor that involve the system of that is being developed**.**
   2. **To model an operation:** We use flow chart like activity diagrams to model details of a computation.Here modeling of branch, fork and join states are particularly important.
3. **Contents of an activity diagram:** Activity diagram commonly contains:
   1. Activity, states and action states.
   2. Transitions
   3. Objects

3.1 Action states: An action state represents the non-interruptible action of objects. You can draw an action state using a rectangle with rounded corners.

3.2 Activity states: Activity states can be further decomposed; their activity being represented by other activity diagrams. They are not atomic. Hence they may be interrupted and are considered to take some finite time to complete. Action state is a special state of activity state. An activity state is composite and its flow of control is made up of other activity state. An activity state is composite and its flow of control is made up of other activity states and action states. Activity states have additional parts like entry and exit actions and some machine specifications.

3.3 Translations:

1. When the action or activity of state completes, flow of control passes immediately to next action or activity state.
2. This flow is specified by using transitions to show path from one action or activity state to the next action or activity state.

IT/LP-II: D-07	**Prepare Activity Model from the given description using UML 2 notations.**	**Page**	**2/4**
Experiment No: 3	**Semester – VI**	**Rev.: 00**	**Date:**

3. The transition is a simple directed line.
4. The flow of control has to start and end of some place.
5. The start state is specified by a solid ball and the end state is specified by a solid ball in a circle
6. Trigger less transitions are those where control passes immediately once work of source is done.
7. Once the work of a state is done, its exit action is executed. Immediately control is transferred without delay to the next state.
8. Then the entry action is executed and the process continues. Once the work of that state is done this process continues indefinitely.

3.4 Branching:

1. Simple sequential transitions are common. But they are not only kind of path we need to module.
2. As in a flow chart a branch maybe included to show any alternate flow if required.
3. These alt paths are taken on evaluation on Boolean expressions.
4. A branch may have one incoming transition and two or more outgoing ones. On each outgoingone, a Boolean expression is placed, which is evaluated only on entering the branch.
5. The effect of iteration can be achieved by using one action state that sets value of an iterator and abranch that evaluates if the iterator is finished.

3.5 Forking and Joining:

1. Simple branching, sequential transitions are most common paths found in activity diagrams butwhile modeling work flows of business processors that are concurrent, may be encountered.
2. Synchronization bars are used to specify the forking the joining of parallel flows of control.
3. A synch bar is thick horizontal line.
4. Joins and forks should balance, meaning that the number of flows that lead a fork should match the number of flows that enter its corresponding joins.

3.6 Swim Lane: A swim lane (or swim lane diagram) is used in process flow diagrams, or flowcharts, that visually distinguishes job sharing and responsibilities for sub-processes of a business process.

4. Oral Questions:

1. Explain Activity diagram and its components.
2. What do you mean by Forking & Joining in activity diagram?
3. Differentiate Activity State and Action State.
4. What is swim lane in activity diagram?

5. Assignment Question:

IT/LP-II: D-07	**Prepare Activity Model from the given description using UML 2 notations.**	Page	3/4
Experiment No: 3	Semester – VI	Rev.: 00	Date:

Prepare Activity Diagram for Library System, Railway reservation System and Banking System

Conclusion: We prepared activity model to show various activities, to model the dynamic aspects of a system and for showing flow of central from activity to activity.

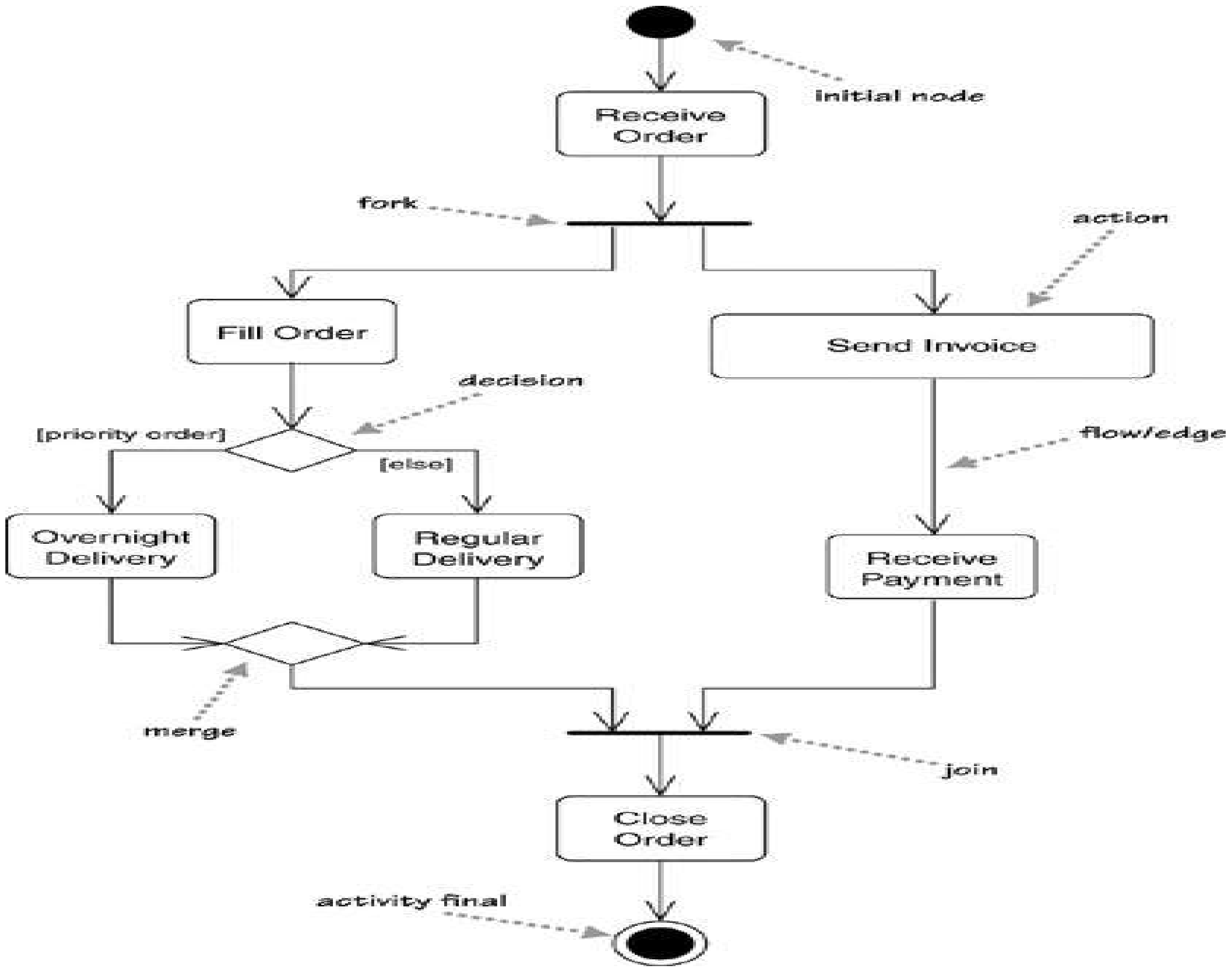


Fig. 1: Activity Diagram

IT/LP-II: D-07	**Prepare Activity Model from the given description using UML 2 notations.**	Page	4/4
Experiment No: 3	Semester – VI	Rev.: 00	Date:

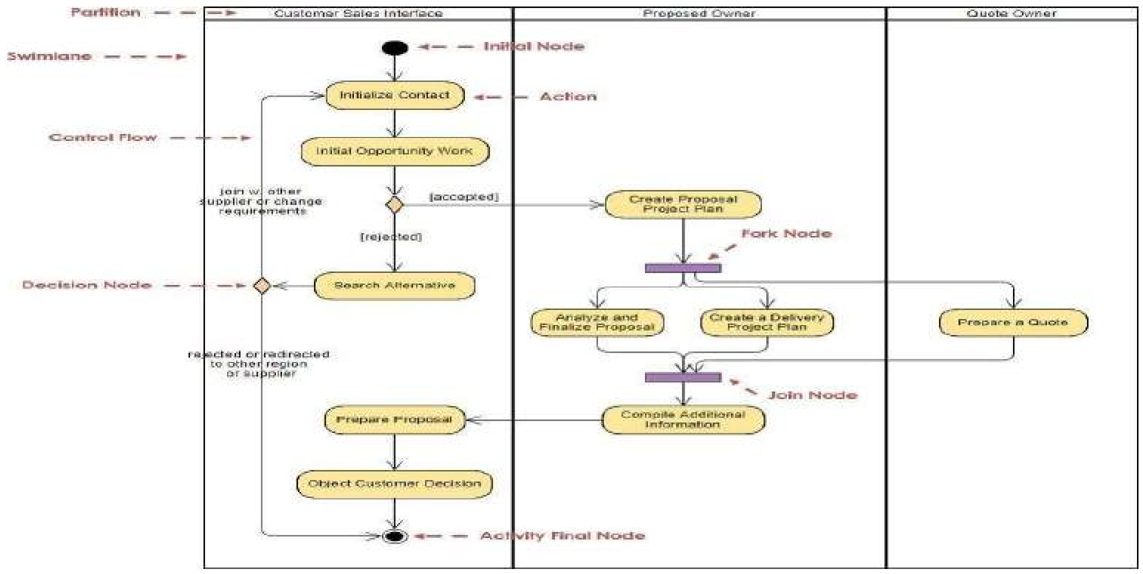


Fig. 2: Activity Diagram with Swim Lane

IT/LP-II: D-08	Prepare Activity Model from the given description using UML 2 notations.	Page	1/7
Experiment No: 4	Semester – VI	Rev.: 00	Date:

Practical No. 5

Title of the Practical: Prepare a class diagram from the given problem description using UML2.0 and Implement the class diagram with a suitable object oriented language.

Objective of the Assignment: Prepare Identify Analysis Classes and assign responsibilities. Prepare Data Dictionary. Draw Analysis class Model using UML2.0 Notations. Implement Analysis class Model-class diagram with a suitable object oriented language.

Prerequisite: Basic object oriented concepts.

Theory:

1. Model:

1.1 Introduction: Model is an abstraction of something for the purpose of understanding, before building it. Software Designers build many kinds of models for various purposes before constructing actual software system. Engineers, artists, and craftsman have built models for thousands of years to try out designs before executing them.

Examples:

1. Architectural models to show customers.
2. Pencil Sketches for Composition of oil painting
3. Blueprints of machine parts
4. Storyboard of advertisements.
5. Outline of books.

1.2 Analysis model:

The purpose of any Analysis activity in the software life-cycle is to create a model of the System ‘s functional requirements that is independent of implementation constraints.

This model:

- Defines what the system does not how it does it.
- Defines logical requirements in more detail than the use case model, rather than a physical solution to the requirements
- Leaves out all technology detail, including system topology

Focuses on the definition of classes and the manner in which they collaborate to effect the customer requirements Traditional analysis methodologies, the two aspects: processes and data are considered separately. For example, data may be modeled by ER diagrams, and behaviors by flow charts.

The primary tasks in analysis model (OOA) are:

- Find the objects
- Organize the objects
- Describe how the objects interact

IT/LP-II: D-08	**Prepare Activity Model from the given description using UML 2 notations.**	Page	2/7
Experiment No: 4	**Semester – VI**	**Rev.: 00**	**Date:**

- Define the behavior of the objects
- Define the internals of the objects

2. Class diagram:

2.1. **Introduction:** Gives an overview of a system by showing its classes and the relationships amongthem.

- The diagram shows attributes and operations of each class
- Good way to describe the overall architecture of system components
- Generally, UML diagrams are not directly mapped with any object oriented programminglanguages but the class diagram is an exception.
- A class is simply represented as a box with the name of the class inside
- Each class is represented by a rectangle subdivided into three compartments
  - Name
  - Attributes
  - Operations
- Modifiers are used to indicate visibility of attributes and operations.
  - _+ ‘is used to denote *Public* visibility (everyone)
  - _# ‘is used to denote *Protected* visibility (friends and derived)
  - _- ‘is used to denote *Private* visibility (no one)
- By default, attributes are hidden and operations are visible.

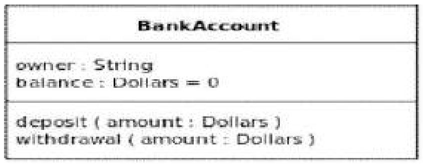


Fig. 1: An example of Class

2.2 **UML Class Notation**

1. Lines or arrows between classes indicate relationships
2. Association
3. Aggregation
4. Composition
5. Generalization(Inheritance)

2.2.1 **Link and Association:**

- When classes are connected together conceptually, that connection is called an association
- Link is a physical connection among objects.
- Association is a description of a group of link with common structure.
- An Association represents a family of links
- Multiplicity
  - The number of objects that participate in the association.
  - Indicates whether or not an association is mandatory.

IT/LP-II: D-08	Prepare Activity Model from the given description using UML 2 notations.	Page	3/7
Experiment No: 4	Semester – VI	Rev.: 00	Date:

Table 1: Different way of multiplicity

Exactly one	1
Zero or more (unlimited)	* (0...*)
One or more	1..*
Zero or one (optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8

- Labeling associations: Each association can be labelled, to make explicit the nature of the association

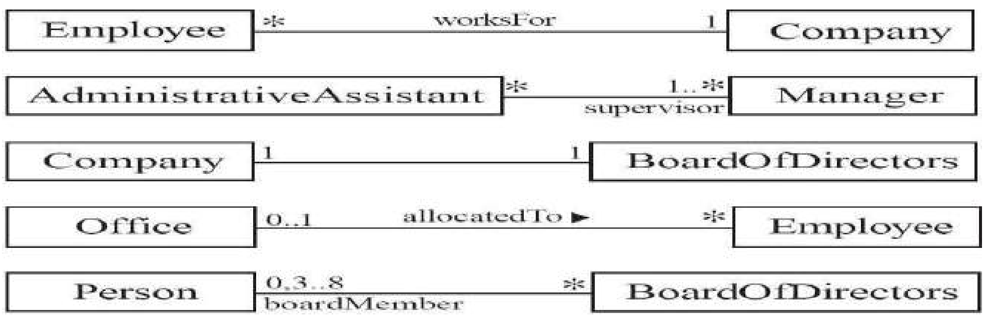


Figure 2: Labelling associations

Directionality in associations
- Associations are by default *bi-directional*
- It is possible to limit the direction of an association by adding an arrow at one end

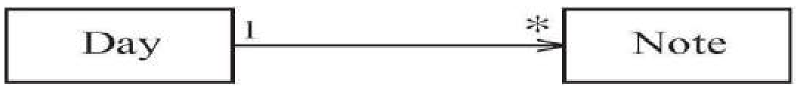


Figure 3: Directionality in associations

2.2.2 Generalization: It is the relationship between a general thing (called super class or parent) and a more specific kind of that thing (called sub class or child). Generalization is called an ―is-a –kind-of‖ relationship.
- The parent class is more general than the child class.

IT/LP-II: D-08	**Prepare Activity Model from the given description using UML 2 notations.**	Page	4/7
Experiment No: 4	Semester – VI	Rev.: 00	Date:

- Object-orientation refers to this as Inheritance. The UML also refers to this as Generalization.
- This type of connection stands for the phrase is a kind of
- Each subclass is said to inherit the features of its super class.
- Generalization expresses a parent/child relationship among related classes.
- Notation: Generalization among classes is rendered as a solid directed line with a large open arrowhead
- Example: Consider a parent class Shape and child classes Rectangle, Circle, The generalization between them is as shown in figure.

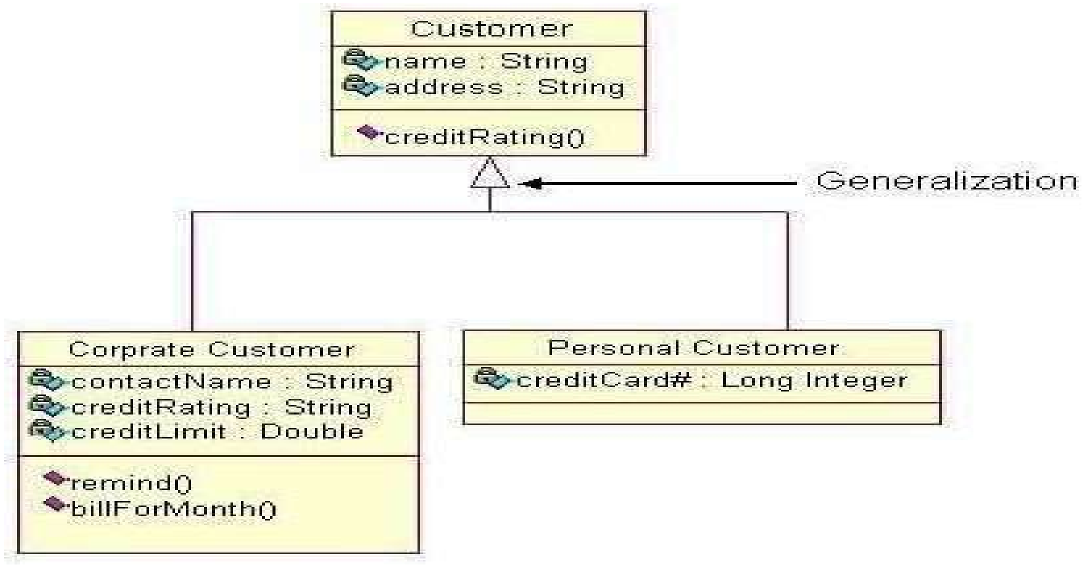


Figure 4: Example of generalization

2.2.3 Aggregation: Two forms of whole relationships:

1. Aggregation: general form
2. Composition: a more restrictive form called composition
   Sometimes a class consists of a number of component classes. This is a special type of relationship called an **aggregation.**

The components and the class they constitute are in a part-whole association It a form of aggregation with two additional constraints.

1. A constituent part can belong to at most one assembly
2. Each component in a composite can belong to just one whole.

Notation: It is specified by adorning a plain association with an open diamond at the whole end.

IT/LP-II: D-08	Prepare Activity Model from the given description using UML 2 notations.	Page	5/7
Experiment No: 4	Semester – VI	Rev.: 00	Date:

Example: Consider the Various Classes Company, Warehouse, Salesman and Store. In these classes Company acts as a whole part. The association between them is shown as follows:

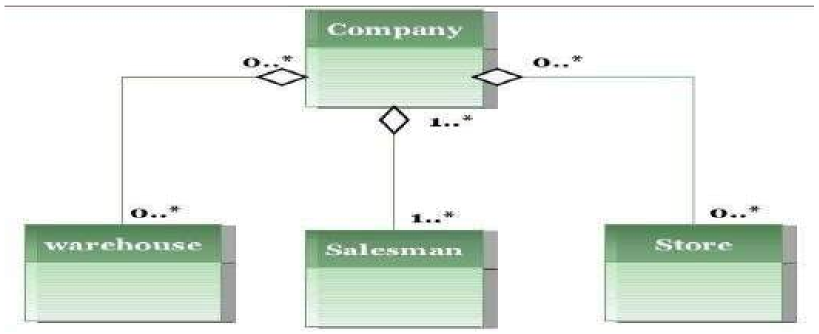


Figure 5: Example of Aggregation

2.2.4 **Composition**: is a form of aggregation, with strong relationship and coincident lifetime as part of the whole. Parts with non-fixed multiplicity may be created after the composite itself, but once created they live and die with it. This means that, in a composite aggregation, an object may be a part of only one composite at a time. Deletion of an assembly object triggers deletion of all constituent objects via composition **Notation**: Solid diamond at the whole end.

Example 1: In windowing system, a Frame belongs to exactly one Window. When you create a Frame in a system, you must attach it to an enclosing Window. Similarly, when you destroy the window, the Window object must in turn destroy its Frame parts.

Example 2: The components of a car are wheel, steering, seatbelt, Windshield make a composite.

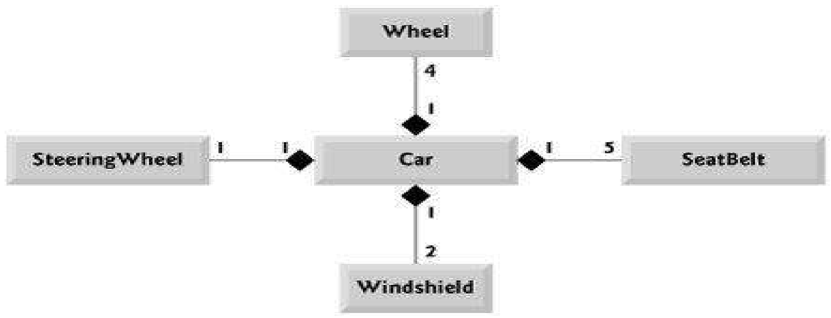


Figure 6: Example of Composition

2.2.5 Dependency: A dependency is a using relationship that states that a change in specification of one thing may affect another thing that uses it, but not necessarily reverse. Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time.

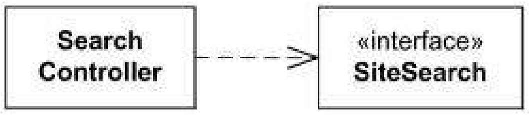
- One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class.

IT/LP-II: D-08	**Prepare Activity Model from the given description using UML 2 notations.**	Page	6/7
Experiment No: 4	Semester – VI	Rev.: 00	Date:

- **Notation**: Graphically, a dependency is rendered as a dashed line, directed to the thing being depended on

- **Example:** The dependency from *Search Controller to SiteSearch* exists in the following diagram

Figure 7: Example of Dependency



3. Object Oriented Language: Implement the class diagram with a suitable object oriented language. **Java** is a general-purpose computer programming language that is concurrent, class-based, object-oriented Java was originally developed by James Gosling at Sun Microsystems (which has since beenacquired by Oracle Corporation) It is intended to let application developers "write once, run anywhere"(WORA),[13] meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) The syntax of Java is largely derived from C++.

The traditional "Hello, world!" program can be written in Java as:

```
classHelloWorldApp
    {
        publicstaticvoidmain(String[] args)
          {
              System.out.println("Hello World!");// Prints the string to the console.    }
    }
```

Source files must be named after the public class they contain, appending the suffix .java, for example, HelloWorldApp.java. It must first be compiled into bytecode, using a Java compiler, producing a file named HelloWorldApp.class. Only then can it be executed, or "launched".

The Java source file may only contain one public class, but it can contain multiple classes with other than public access. When the source file contains multiple classes, make one class "public" and name the source file with that public class name. A class that is not declared public may be stored in any .java file. The compiler will generate a class file for each class defined in the source file. The name of the class file is the name of the class, with .class appended. The keyword public denotes that a method can be called from code in other classes, or that a class may be used by classes outside the class hierarchy. The class hierarchy is related to the name of the directory in which the .java file is located. The keyword static in front of a method indicates a static method, which is associated

IT/LP-II: D-08	**Prepare Activity Model from the given description using UML 2 notations.**	**Page**	**7/7**
Experiment No: 4	**Semester – VI**	**Rev.: 00**	**Date:**

only with the class and not with any specific instance of that class. Only static methods can be invoked without a reference to an object. Static methods cannot access any class members that are not also static. The keyword void indicates that the main method does not return any value to the caller. If a Java program is to exit with an error code, it must call System.exit() explicitly. The method name "main" is not a keyword in the Java language. It is simply the name of the method the Java launcher calls to pass control to the program.

4 Oral Questions

1. What is use of class diagram?

2. Explain with graphical notation and Example a) Association b) Generalization c) Aggregation d) Dependency e) Composition

3. A file system has many files. A file can be ASCII file, binary file or directory file. Files are stored in a disk. Disk contains many tracks. A track is made up of many sectors. Convert this into a class diagram with appropriate relationships, multiplicity.

4. What is a qualified association? Explain with example.

5. Car dealer sells car. A car is owned by an owner. The owner has an address. The owner can be a person, a company or a bank. A car loan may be involved in the purchase of a car. Bank provides a loan.

5. Assignment Questions

Prepare and Implement Class Model for Railway Library Management System, Reservation System & Online Shopping.

Conclusion: In this way we have studied basics of analysis class model, that gives an overview of a system by showing its classes and the relationships among them.

IT/LP-II: D-09	Prepare a Design Model from Analysis Model.	Page	1/4
Experiment No: 5	Semester – VI	Rev.: 00	Date:

Practical No. 5

Title of the Practical: Prepare a Design Model from Analysis Model.

Objective of the Assignment: Study in detail working of system/Project.

1. Identify Design classes/ Evolve Analysis Model. Use advanced relationships.
2. Draw Design class Model using OCL and UML2.0 Notations.
3. Implement the design model with a suitable object-oriented language.

Prerequisite: Basic object oriented concepts.

Theory:

1. Model:

1.1 Introduction: Model is an abstraction of something for the purpose of understanding, before building it. Software Designers build many kinds of models for various purposes before constructing actual software system. Engineers, artists, and craftsman have built models for thousands of years to try outdesigns before executing them.

Examples:

1. Architectural models to show customers.
2. Pencil Sketches for Composition of oil painting
3. Blueprints of machine parts
4. Storyboard of advertisements.
5. Outline of books.

1.2 Design Model:

During design model developer applies implementation constraints to the conceptual model produced in analysis. Such constraints could include the hardware and software platforms, the performance requirements, persistent storage and transaction, usability of the system, and limitations imposed by budgets and time.

Concepts in the analysis model which is technology independent are mapped onto implementing classes and interfaces resulting in a model of the solution domain, i.e., a detailed description of *how* the system is to be built on concrete technologies.

1. Class diagram:

Gives an overview of a system by showing its classes and the relationships among them.

IT/LP-II: D-09	Prepare a Design Model from Analysis Model.	Page	2/4
Experiment No: 5	Semester – VI	Rev.: 00	Date:

- The diagram shows attributes and operations of each classGood way to describe the overall architecture of system components
- Generally UML diagrams are not directly mapped with any object oriented programminglanguages but the class diagram is an exception.
- A class is simply represented as a box with the name of the class inside
- Each class is represented by a rectangle subdivided into three compartments
    - o Name
    - o Attributes o
    Operations
- Modifiers are used to indicate visibility of attributes and operations.
    - o ‗+‘ is used to denote *Public* visibility (everyone)
    - o ‗#‘ is used to denote *Protected* visibility (friends and derived)
    - o ‗-‘ is used to denote *Private* visibility (no one)
- By default, attributes are hidden and operations are visible.

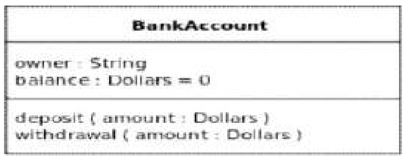
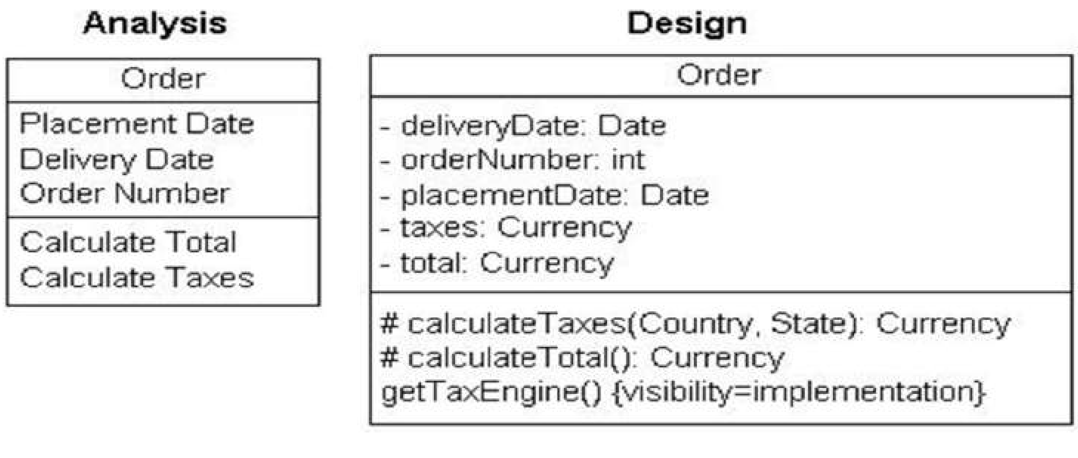


Figure 1: An example of Class

3. Design Model Class Vs Analysis Model Class:

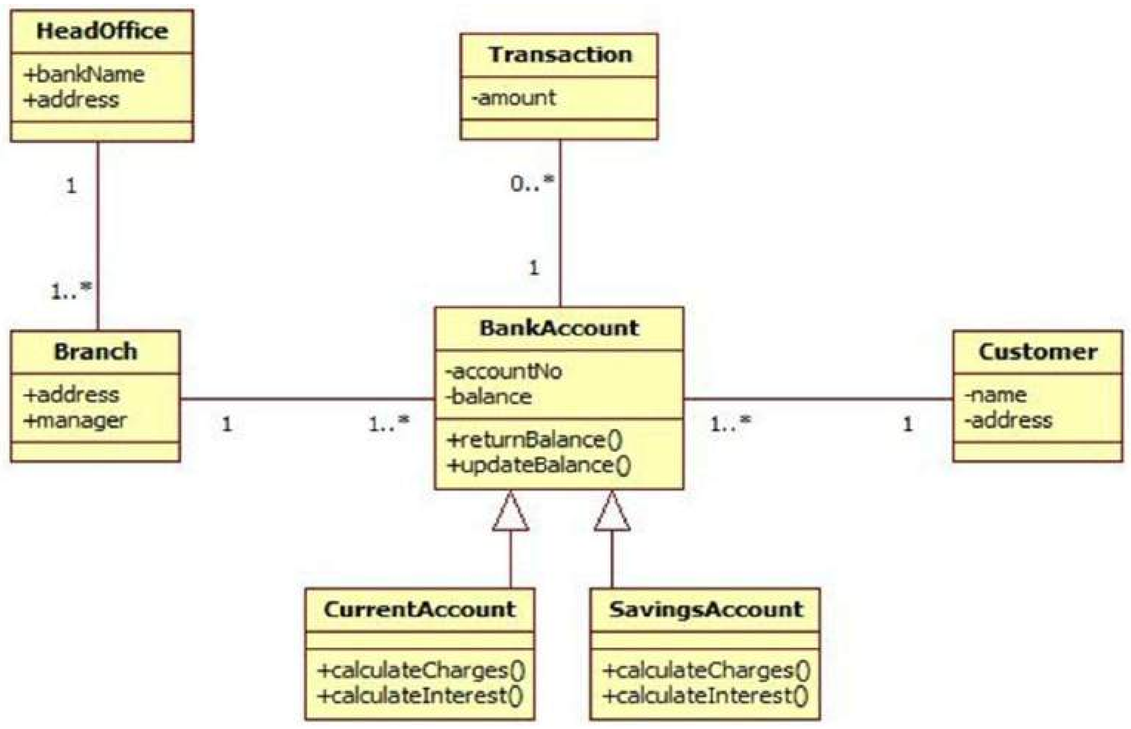
Analysis is all about understanding of requirements, problems within requirements by breaking the whole module into sub modules (Parts). Diagrams can be made to understand and pin point problems in requirements. While Designing is all about how the process in requirement will carry out. Like Unified Modelling Diagrams can help in designing. In short designing is to show how process will be carried out ina sub module. The Design model class will describe the detail about class attributes and class



IT/LP-II: D-09	**Prepare a Design Model from Analysis Model.**	**Page**	**3/4**
Experiment No: 5	**Semester – VI**	**Rev.: 00**	**Date:**

operations, such as access specifier for each variable and function, function attributes, parameters and return types.

Consider the example: A banking system has a variety of classes, which can be quickly outlined with a class diagram. The diagram below shows how items in the system interact, providing a helpful look at tellers, customers, account types, and other factors involved in banking.



IT/LP-II: D-09	Prepare a Design Model from Analysis Model.	Page	4/4
Experiment No: 5	Semester – VI	Rev.: 00	Date:

Implement following design model using java programming language

4 Oral Questions

1. What is use of class diagram?
2. Explain analysis model and design model?
3. What is difference between class diagram and advanced class diagram?
4. What is visibility in class diagram?
5. A musical company has decided to store information about musicians who perform in their albums. Each musician is identified with unique identifier. The instruments are used in songs. Song has a title and an author. Each musician may play several instruments and each instrument is played by several musicians. Each album has exactly one musician who acts as its producer. A musician may produce several albums. Draw a class diagram using advanced notations.

5.Assignment Questions

Prepare and Implement Design Model for Railway Reservation System, Library Management System & HotelManagement System.

Conclusion: In this way we have studied basics of design model and implemented design model usingobject oriented language java.

IT/LP-II: D-10	**Prepare Activity Model from the given description using UML 2 notations.**	**Page**	**1/4**
Experiment No: 6	**Semester – VI**	**Rev.: 00**	**Date:**

Practical No. 6

Title of the Practical: Prepare a Sequence diagram from the given problem description usingUML2.0 and Implement the Sequence diagram with a suitable object oriented language.

Objective of the Assignment: Identify at least 5 major scenarios (sequence flow) for your system. Draw Sequence Diagram for every scenario by using advanced notations using UML2.0 Implement these scenarios by taking reference of design model implementation using suitable object-oriented language.

Prerequisite: Basic object oriented concepts.

Theory:

1. Sequence Diagram

Sequence diagram is the most common kind of interaction diagram, which focuses on the message interchange between a number of lifelines.
Sequence diagram describes an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines.
A sequence diagram is an interaction diagram that emphasizes the time ordering of messages. Graphically, a sequence diagram is a table that shows objects arranged on X-Axis and messages, orderedin increasing time, along Y-axis.

1.1 Contents of Sequence diagram:

1 Objects
2 Messages
3 Interaction Fragments

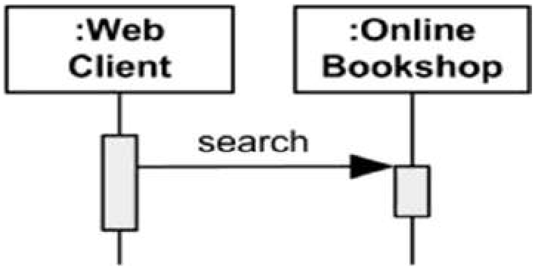
1.1.1 Objects: Objects have some things associated with them like lifeline, focus of control etc.
Lifeline is a named element which represents an individual participant in the interaction.
Focus of Control/Activation box represents the time for which object is active in its lifetime

1.1.2 Message: It is a named element that defines one specific kind of communication between lifelines of an interaction. The message specifies not only the kind of communication, but also the sender and the receiver.
A message reflects either an operation calls and start of execution or a sending and reception of a signal. When a message represents an operation call, the arguments of the message are the arguments of the operation. When a message represents a signal, the arguments of the message are the attributes of the signal.

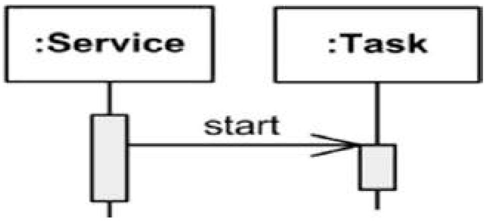
a. **Synchronous call** typically represents operation call - send message and suspend execution while

IT/LP-II: D-10	Prepare Activity Model from the given description using UML 2 notations.	Page	2/4
Experiment No: 6	Semester – VI	Rev.: 00	Date:

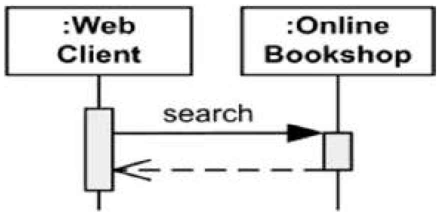
b. waiting for response. Synchronous call messages are shown with filled arrow head.



c. Asynchronous call - send message and proceed immediately without waiting for return value.Asynchronous messages have an open arrow head.



d. Return Message: It is a reply message to an operation call is shown as a dashed line with open arrow head.



e. Create message is sent to a lifeline to create itself. It is shown as a dashed line with open arrowhead (looks the same as reply message), and pointing to the created lifeline's head.

f. Delete message (called **stop** in previous versions of UML) is sent to terminate another lifeline. The lifeline usually ends with a cross in the form of an **X** at the bottom denoting destruction occurrence.

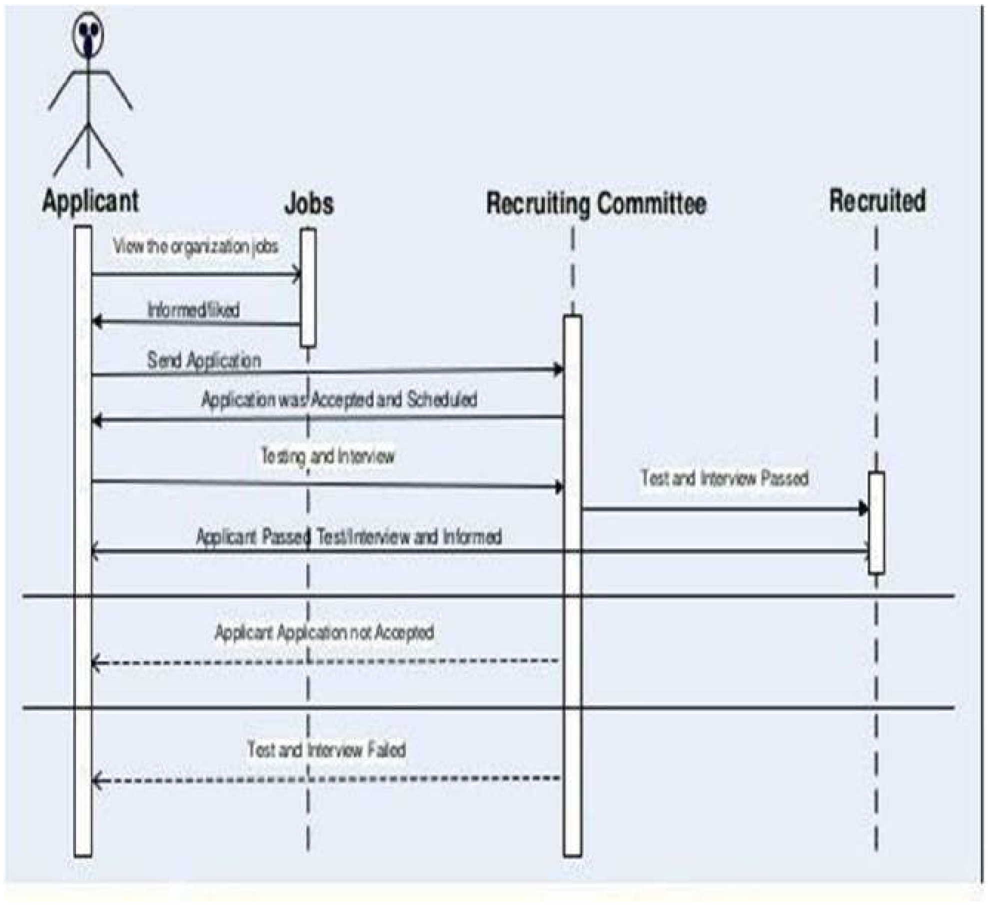
1.2 Interaction Operator: A combined flow is used to group sets of messages together to show conditional flow in sequence diagram.

a. Alteration: Alteration fragment is used to model a sequence diagram in case if alternative solution is present to draw the sequence diagram.

b. Option: The option combination fragment is used to model a sequence that, given a certain condition,

IT/LP-II: D-10	Prepare Activity Model from the given description using UML 2 notations.	Page	3/4
Experiment No: 6	Semester – VI	Rev.: 00	Date:

c. will occur; otherwise does not occur.
d. Parallel: If two activities are going simultaneously, then parallel fragment is used.
d. Loop: When modeling a sequence diagram, there will be times that an object will need to send message to it. To draw an object calling itself, you draw a message as you would normally, but insteadof connecting it to another object, you connect the message back to the object itself.


Applicant
Jobs
Recruiting Committee
Recruited
View the organization jobs
Informed/liked
Send Application
Application was Accepted and Scheduled
Testing and Interview
Test and Interview Passed
Applicant Passed Test/Interview and Informed
Applicant Application not Accepted
Test and Interview Failed

Sequence Diagram for Employee Management System

2. Oral Questions

1. What is difference between Sequence and Communication diagram?
2. Give difference between Synchronous and Asynchronous Message.
3. Explain Interaction fragments like break, seq, strict.
4. What do you mean by focus of control in sequence diagram?

IT/LP-II: D-10	**Prepare Activity Model from the given description using UML 2 notations.**	**Page**	**4/4**
Experiment No: 6	**Semester – VI**	**Rev.: 00**	**Date:**

5. Differentiate Synchronous and Asynchronous Message.

3. Assignment Question

Sequence Diagram for Library Management System, Railway ticket Booking System, Hotel Management System.

Conclusion: For a complex system, we prepared a use case model, sequence model and activity model.

IT/LP-II: D-11	Prepare a state model from the given problem description, draw a state diagram using UML2 notations and Implement the state model with a suitable object oriented language.	Page	1/7
Experiment No: 7	Semester – VI	Rev.: 00	Date:

Practical No. 7

Title of the Practical: Prepare a state model from the given problem description, draw a state diagram using UML2 notations and Implement the state model with a suitable object oriented language.

Objective of the Assignment: Identify States and events for your system.

1. Study state transitions and identify Guard conditions.
2. Draw State chart diagram with advanced UML 2 notations.
3. Implement the state model with a suitable object-oriented language.

Prerequisite: Basic object oriented concepts.

Theory:

According to view of a system there are three models namely,

1. Class Model
2. State Model
3. Interaction Model

1. State Model:

State Model describes those aspects of objects concerned with Time and sequencing of operations, Eventsthat marks changes, States that define the context for events and The organization of events and states.

This model captures control of a system; State diagram express the state model.

2. State Diagram:
State diagrams are used to describe the behaviour of a system. These diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system. Using a state machine, we can model the behaviour of a society of objects that work together or behaviour of an individual object.

2.1 State machine: A state machine is a behavior that specifies the sequences of states an object goesthrough during its life time in response to events, together with its responses to those events.

a. State: A state is a condition of an object in which it performs some activity or waits for an event.

IT/LP-II: D-11	**Prepare a state model from the given problem description, draw a state diagram using UML2 notations and Implement the state model with a suitable object oriented language.**	**Page**	**2/7**
Experiment No: 7	**Semester – VI**	**Rev.: 00**	**Date:**

b. Denoted by a rectangle with rounded corners and compartments (such as a class with rounded corners to denote an Object).

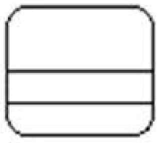


Fig 1 Graphical Notation for State

A state has following things associated with it

Name	A textual string which distinguishes the state from other states; a state may also be anonymous, meaning that it has no name.
Entry/exit actions	Actions executed on entering and exiting the state.
Internal transitions	Transitions that are handled without causing a change in state.
Substates	The nested structure of a state, involving disjoint (sequentially active) or concurrent (concurrently active) sub states.
Deferred events	A list of events that are not handled in that statebut are postponed and queued for handling by the object in another state.

c. Entry and Exit Actions

Entry and exit actions allow the same action to be dispatched every time the state is entered or left, respectively. Entry and exit actions enable this to be done cleanly, without having to explicitly put theactions on every incoming or outgoing transition explicitly. Entry and exit actions may not have arguments or guard conditions. The entry actions at the top-level of a state machine for a model element may have parameters representing the arguments that the machine receives when the element is created.

d. Internal Transitions

IT/LP-II: D-11	Prepare a state model from the given problem description, draw a state diagram using UML2 notations and Implement the state model with a suitable object oriented language.	Page	3/7
Experiment No: 7	Semester – VI	Rev.: 00	Date:

e.
Internal transitions allow events to be handled within the state without leaving the state, thereby avoiding triggering entry or exit actions. Internal transitions may have events with parameters and guard conditions, and essentially represent interrupt-handlers.

f. Deferred Events
Deferred events are those whose handling is postponed until a state in which the event is not deferred becomes active. When this state becomes active, the event occurrence is triggered and may cause transitions as if it had just occurred. The implementation of deferred events requires the presence of an internal queue of events. If an event occurs but is listed as deferred, it is queued. Events are taken off this queue as soon as the object enters a state that does not defer these events.

g. Composite state- A state that consists of sub states (concurrent sub states or sequential sub states) is called as composite states. A composite state may contain either concurrent (orthogonal) or sequential (disjoint) substrates. A composite state is either a simple composite state (with just one region) or an orthogonal state (with more than one region)

h. Region: A region is an orthogonal part of either a composite state or a state machine. It contains statesand transitions.

i. Sequential sub state: A sequential substate is a substate in which an object can reside to the exclusion of all other sub states at that same level within the given composite state. Given two or more sequential sub states at the same level, an object can be in only one of them. It is also called as disjoint state.

IT/LP-II: D-11	Prepare a state model from the given problem description, draw a state diagram using UML2 notations and Implement the state model with a suitable object oriented language.	Page	4/7
Experiment No: 7	Semester – VI	Rev.: 00	Date:

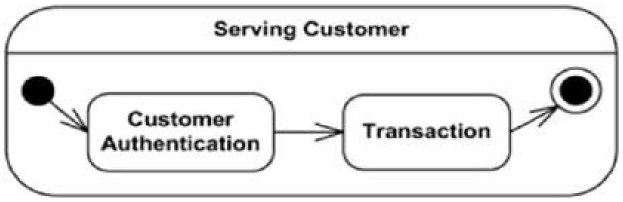


Fig2: sequential Sub states

j. Concurrent sub state: A concurrent substate is a substate in which an object can reside simultaneously with other sub states at that same level within the given composite state. Given two or more composite sub states at the same level, an object may be in one or more of them. In this situation, two or more sets of sub states can represent parallel flows of control. When the object enters a composite state with concurrent sub states, the object also enters the initial states of each set of sub states. Just like on an activity diagram, you can resynchronize these parallel flows. On a state diagram, however, you show this resynchronization by using a final state for each parallel set of concurrent sub states.

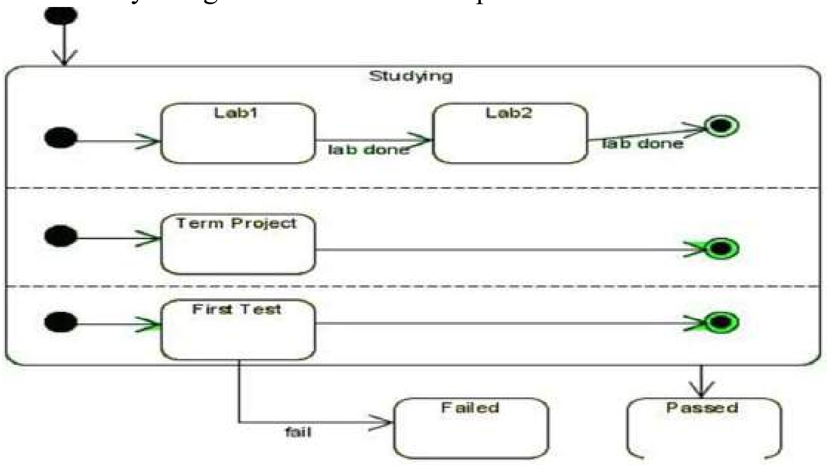


Fig3 Concurrent sub states separated by region

k. Final State: The end of the state diagram is shown by a bull's eye symbol, also called a final state. Afinal state is another example of a pseudo state because it does not have any variable or action described.

l. Transition

IT/LP-II: D-11	Prepare a state model from the given problem description, draw a state diagram using UML2 notations and Implement the state model with a suitable object oriented language.	Page	5/7
Experiment No: 7	Semester – VI	Rev.: 00	Date:

m. A transition is a directed relationship between a source vertex and a target vertex. It may be part of a compound transition, which takes the state machine from one state configuration to another, representingthe complete response of the state machine to an occurrence of an event of a particular type

A transition has five parts:

1. Source state: The state affected by the transition; if an object is in the source state an outgoing transition may fire when the object receives the trigger event of the transition and if the guard condition, if any, is satisfied.
2. Event trigger: The event whose reception by the object in the source state makes the transition eligible to fire.
3. Guard condition: A Boolean expression that is evaluated when the transition is triggered by thereception of the event trigger.
4. Action: An executable atomic computation that may directly act on the object that owns the statemachine.
5. Target state: The state that is active after the completion of the transition.

n. Event Triggers

In the context of the state machine, an event is an occurrence of a stimulus that can trigger a state transition. Events may include signal events, call events, the passing of time, or a change in state. A signalor call may have parameters whose values are available to the transition, including expressions for the guard conditions and action. It is also possible to have a trigger less transition, represented by a transition with no event trigger. These transitions, also called completion transitions, is triggered implicitly when its source state has completed its activity.

o. Guard Conditions

A guard condition is evaluated after the trigger event for the transition occurs. It is possible to have multiple transitions from the same source state and with the same event trigger, as long as the guard conditions don't overlap. A guard condition is evaluated just once for the transition at the time the event occurs. The boolean expression may reference the state of the object.

p. Actions

An action is an executable atomic computation, meaning that it cannot be interrupted by an event and therefore runs to completion. This is in contrast to an activity, which may be interrupted by other events. Actions may include operation calls (to the owner of the state machine as well as other visible objects), the creation or destruction of another object, or the sending of a signal to another object. In the case of sendinga signal, the signal name is prefixed with the keyword 'send'.

IT/LP-II: D-11	Prepare a state model from the given problem description, draw a state diagram using UML2 notations and Implement the state model with a suitable object oriented language.	Page	6/7
Experiment No: 7	Semester – VI	Rev.: 00	Date:

3. UML State Machine Diagram Example for Bank ATM

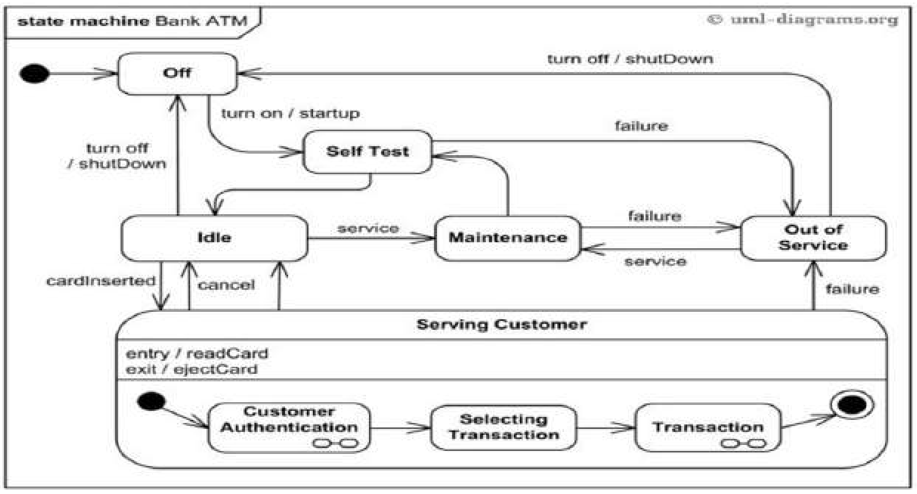


Fig 4 state Machine diagram for ATM

This is an example of UML behavioural state machine diagram showing Bank Automated Teller Machine (ATM) top level state machine.

ATM is initially turned off. After the power is turned on, ATM performs startup action and enters Self Test state. If the test fails, ATM goes into Out of Service state, otherwise there is trigger less transition tothe Idle state. In this state ATM waits for customer interaction.

The ATM state changes from Idle to Serving Customer when the customer inserts banking or credit card in the ATM's card reader. On entering the Serving Customer state, the entry action read Card is performed. Note, that transition from Serving Customer state back to the Idle state could be triggered by cancel eventas the customer could cancel transaction at any time.

Serving Customer state is a composite state with sequential sub states Customer Authentication, Selecting Transaction and Transaction. Customer Authentication and Transaction are composite states by themselves which is shown with hidden decomposition indicator icon. Serving Customer state has trigger

IT/LP-II: D-11	Prepare a state model from the given problem description, draw a state diagram using UML2 notations and Implement the state model with a suitable object oriented language.	Page	7/7
Experiment No: 7	Semester – VI	Rev.: 00	Date:

less transition back to the Idle state after transaction is finished. The state also has exit action eject Card which releases customer's card on leaving the state, no matter what caused the transition out of the state.

4. Oral Questions

1. State Diagram is which kind of diagram?
2. Define State, State machine, Transition and Event.
3. Give difference between Activity diagram and State Diagram.
4. What do you mean by Tireless Transition?

5. Assignment Question

Prepare and Implement State Diagram for Washing Machine, ATM Machine and Coffee wandering machine

Conclusion:

So we can conclude that a state diagram*,* sometimes known as a state machine *diagram*, is a type of behavioural diagram in the Unified Modelling Language (UML) that shows transitions between various objects.

IT/LP-II: D-12	Identification and Implementation of GRASP pattern.	Page	1/5
Experiment No: 8	Semester – VI	Rev.: 00	Date:

Practical No. 8

Title of the Practical: Identification and Implementation of GRASP pattern

Objective of the Assignment: Apply any two GRASP pattern to refine the Design Model for a given problem description Using effective UML 2 diagrams and implement them with a suitable object oriented language.

Prerequisite: Basic object oriented concepts.

Theory:

1. Introduction:

GRASP (General Responsibility Assignment Software Patterns) is an acronym created by Craig Larman to encompass nine object oriented design principles related to creating responsibilities for classes. These principles can also be viewed as design patterns and offer benefits similar to the classic –Gang of Four‖ patterns. GRASP is an attempt to document what expert designers probably know intuitively.
General Responsibility Assignment Software Patterns (or Principles), is a collection of generalobjected- oriented design patterns related to assigning defining objects.

2. Responsibility-Driven Design:

GRASP patterns are used to assign responsibility to objects. As such, their use results in a Responsibility-Driven Design (RDD) for Object Orientation (OO) – Contrast to (the more traditional) Data-Driven Design. With this point of view, assigning responsibilities to objects is a large part of basic object design

3. Why GRASP?

Traditionally in Object-Oriented Programming (OOP), object design is glossed over

- E.g., think of nouns and convert to objects; think of verbs and convert to methods
- Or even: After requirements analysis and creation of a domain model, just create objects and theirmethods to fulfill requirements.

UML is just a language—it expresses an OO design but for the most part does not provide guidance. Per Larman, GRASP helps one ―understand essential object design and apply reasoning in a methodical, rational, explainable way.

4. GRASP patterns:

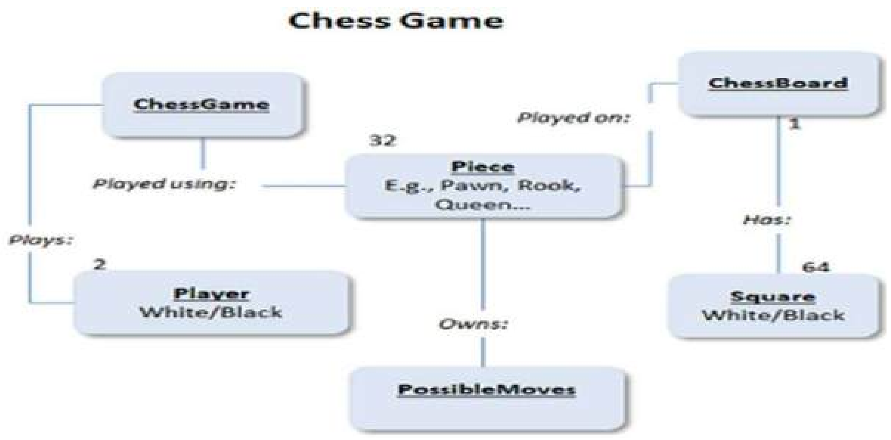
There are nine GRASP patterns, likely some already recognizable and some not:

1. Controller
2. Creator
3. High Cohesion
4. Indirection
5. Information Expert (or just Expert)

IT/LP-II: D-12	**Identification and Implementation of GRASP pattern.**	**Page**	**2/5**
Experiment No: 8	**Semester – VI**	**Rev.: 00**	**Date:**

6. Low Coupling
7. Polymorphism
8. Protected Variations
9. Pure Fabrication

We'll be talking about a chess game and the various responsibilities and relationships between the objects andclasses within the game.

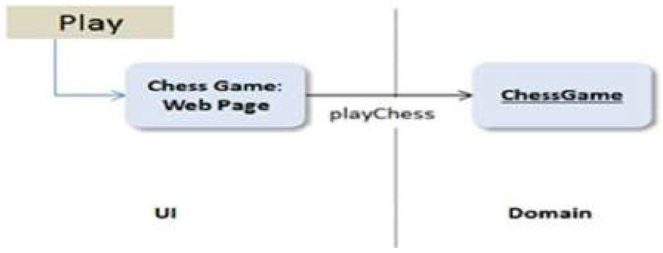


1. Controller

The controller pattern assigns the responsibility of dealing with system events to a non-UI class that represents the overall system or a use case scenario. A controller object is a non-user interface object responsible for receiving or handling a system event. A use case controller should be used to deal with all system events of a use case, and may be used for more than one use case.

It is defined as the first object beyond the UI layer that receives and coordinates ("controls") a system operation. The controller should delegate the work that needs to be done to other objects; it coordinates or controls the activity. It should not do much work itself. The GRASP Controller can be thought of as being a part of the application/service layer in an object-oriented system with common layers in an information system logical architecture.

IT/LP-II: D-12	Identification and Implementation of GRASP pattern.	Page	3/5
Experiment No: 8	Semester – VI	Rev.: 00	Date:



2. **Creator**

Creation of objects is one of the most common activities in an object-oriented system. Which class isresponsible for creating objects is a fundamental property of the relationship between objects of particular classes. In general, a class B should be responsible for creating instances of class A if one, or preferablymore, of the following apply:

- Instances of B contain or compositely aggregate instances of AInstances of B record instances of A
- Instances of B closely use instances of A
- Instances of B have the initializing information for instances of A and pass it on creation.

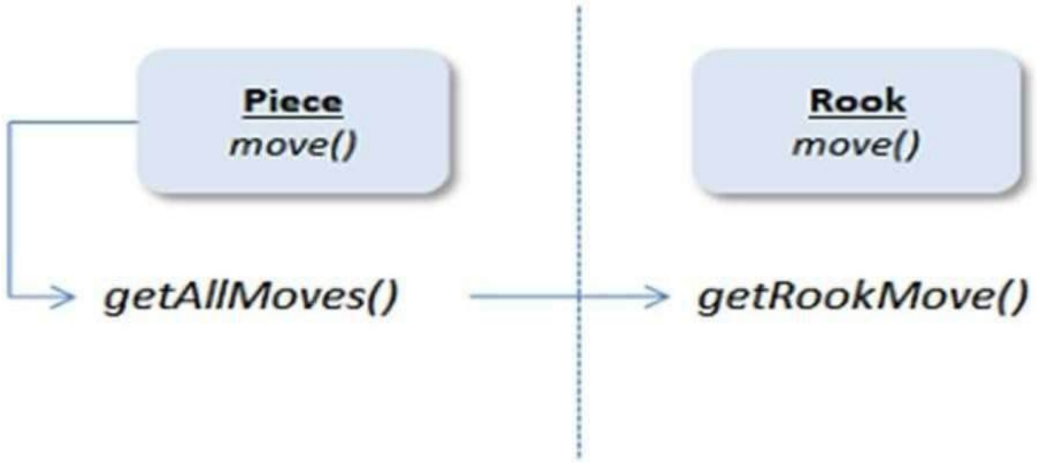
3. **High cohesion**

High cohesion is an evaluative pattern that attempts to keep objects appropriately focused, manageable and understandable. High cohesion is generally used in support of low coupling. High cohesion means that the responsibilities of a given element are strongly related and highly focused. Breaking programs into classes and subsystems is an example of activities that increase the cohesive properties of a system. Alternatively, low cohesion is a situation in which a given element has too many unrelated responsibilities. Elements with low cohesion often suffer from being hard to comprehend, hard to reuse, hard to maintain and averse to change.

4. **Indirection**

The indirection pattern supports low coupling (and reuse potential) between two elements by assigning the responsibility of mediation between them to an intermediate object. An example of this is the introduction of a controller component for mediation between data (model) and its representation (view) in the model-view-controller pattern.

IT/LP-II: D-12	Identification and Implementation of GRASP pattern.	Page	4/5
Experiment No: 8	Semester – VI	Rev.: 00	Date:



5. **Information expert**

Information expert (also expert or the expert principle) is a principle used to determine where to delegateresponsibilities. These responsibilities include methods, computed fields, and so on.

Using the principle of information expert, a general approach to assigning responsibilities is to look at a given responsibility, determine the information needed to fulfil it, and then determine where that information is stored.

Information expert will lead to placing the responsibility on the class with the most information required to fulfil.

6. **Low coupling**

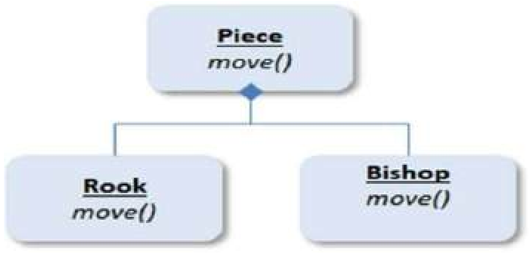
Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. Low coupling is an evaluative pattern that dictates how to assign responsibilities to support

- lower dependency between the classes,
- change in one class having lower impact on other classes,
  higher reuse potential.

7. **Polymorphism**

According to polymorphism principle, responsibility of defining the variation of behaviors based on type is assigned to the type for which this variation happens. This is achieved using polymorphic operations. The user of the type should use polymorphic operations instead of explicit branching based on type.

IT/LP-II: D-12	Identification and Implementation of GRASP pattern.	Page	5/5
Experiment No: 8	Semester – VI	Rev.: 00	Date:



8. Protected variations

The protected variations pattern protects elements from the variations on other elements (objects, systems, subsystems) by wrapping the focus of instability with an interface and using polymorphism to create various implementations of this interface.

9.Pure fabrication

A pure fabrication is a class that does not represent a concept in the problem domain, specially made up to achieve low coupling, high cohesion, and the reuse potential thereof derived. This kind of class is called a "service" in domain-driven design.

4 Oral Questions

1. What is GRASP?
2. Explain Responsibility- Driven Design?
3. What is difference between GRASP pattern and GOF pattern?
4. What are various GRASP patterns?
5. Explain Creator and Controller pattern of GRASP.

5.Assignment Questions

Apply any two GRASP pattern to refine the Design Model for a given problem description Using effective UML 2 diagrams

Conclusion: GRASP provides a map of considerations to provide strong guidance for an OO designer. But at the same time, GRASP still leaves a lot of room to the designer and creating a good design is still an art. Taking a look at GRASP and really Applying UML and Patterns is a good bet for OO designers who know the basics of OOP but are still inexperienced

IT/LP-II: D-13	**Identification and Implementation of GOF pattern.**	Page	1/7
Experiment No: 9	Semester – VI	Rev.: 00	Date:

Practical No. 9

Title of the Practical: Identification and Implementation of GOF pattern.

Objective of the Assignment: Apply any two GOF pattern to refine Design Model for a given problem description Using effective UML 2 diagrams and implement them with a suitable object oriented language.

Prerequisite: Basic object oriented concepts.

Theory:

1. Design pattern

Architect to create a design using design pattern. It represents an idea not particular implementation. A design pattern is a description of communicating objects and classes that are customized to solve a general design problem in a particular context. A design pattern captures design expertise –patterns are not created from thin air, but abstracted from existing design examples. Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

1.2 Uses of Design Patterns

Helping new designers to have a more flexible and reusable design Improving the documentation and maintenance of existing system by furnishing an explicit specification of class and object interactions and their intent. Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem. Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

1.3 Gang of Four (GoF) patterns:

The gang of four (GoF) are:

1.3.1 Erich Gamma
1.3.2 Richard Helm
1.3.3 Ralph Johnson
1.3.4 John Vlissides

Four researchers combined forces that pattern-based design This group was known as the gang of four (GoF) GOF presents each pattern in a structured format.

IT/LP-II: D-13	**Identification and Implementation of GOF pattern.**	Page	2/7
Experiment No: 9	Semester – VI	Rev.: 00	Date:

1.4 Three main categories/types of design pattern

1 Creational pattern
2 Structural pattern
3 Behavioral pattern

1.4.1 Creational pattern

This design patterns is all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get patterns solves this problem by somehow controlling this object creation.

Following are six creational patterns:

1. Abstract Factory - Creates an instance of several families of classes
2. Builder-Separates object construction from its representation
3. Factory Method -Creates an instance of several derived classes
4. Prototype -A fully initialized instance to be copied or cloned
5. Singleton- A class of which only a single instance can exist.

1.4.2 Structural Pattern

Structural Patterns describe how objects and classes can be combined to form larger structures. The difference between class patterns and object patterns is that class patterns describe abstraction with the help of inheritance and how it can be used to provide more useful program interface. Object patterns, on other hand, describe how objects can be associated and composed to form larger, more complex structures.

Following are seven structural patterns:

1. **Adapter** - Convert the interface of a class into another interface client expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

2. **Bridge** - Decouple an abstraction from its implementation so that the two can vary independently.

3. **Composite** - Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

4. **Decorator** - Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub classing for extending functionality.

5. **Façade** - Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

IT/LP-II: D-13	Identification and Implementation of GOF pattern.	Page	3/7
Experiment No: 9	Semester – VI	Rev.: 00	Date:

6. Flyweight - Use sharing to support large numbers of fine-grained objects efficiently.

7. Proxy - Provide a surrogate or placeholder for another object to control access to it.

1.4.3 Behavioural pattern

Behavioural patterns are those which are concerned with interactions between the objects. The interactions between the objects should be such that they are talking to each other and still are loosely coupled. The loose coupling is the key to n-tier architectures. In this, the implementation and the client should be loosely coupled in order to avoid hard-coding and dependencies.

The behavioural patterns are:

1. **Chain of Responsibility-** Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.
2. **Command-** Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.
3. **Interpreter -** Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.
4. **Iterator -** Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
5. **Mediator-** Define an object that encapsulates how a set of objects interact. Mediator promotesloose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.
6. **Memento-** Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.
7. **Observer -** Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
8. **State-** Allow an object to alter its behavior when it‘s internal state changes. The object will appear to change its class.
9. **Strategy -**Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Template Method - Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Visitor- Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

1.5 Each pattern is divided into these sections according to the presented template.

Intent -short description of the pattern & its purpose. That is the goals and objectives it wants to reach withingiven contexts and force.

IT/LP-II: D-13	**Identification and Implementation of GOF pattern.**	Page	4/7
Experiment No: 9	**Semester – VI**	**Rev.: 00**	**Date:**

Also Known As-Any aliases this pattern is known by

Motivation -A scenario that illustrates a design problem and how the class and object structures in the patternsolve the problem.

Applicability - The situations in which the problem and its solution seem to recur, and for which the solutionis desirable. It can be thought of as the initial configuration of a system before the pattern is applied to it.

Structure - A graphical representation of the classes in the pattern using notation based on the ObjectModelling technique.

Participants -participating classes and/or objects & their responsibilities

Collaborations -how participants cooperate to carry out their responsibilities

Consequences - of applying the pattern, both good and bad, and other problems and patterns that may arisefrom the new context. It describes the post conditions and side-effects of the pattern.

Implementation - What pitfalls, hints, or techniques should you be aware of when implementing the patterns?What are language-specific issues?

Sample Code- sample implementations in C++, Java, C#, Smalltalk, C, etc.

Known Uses - examples drawn from existing systems

Related patterns -discussion of other patterns that relate to this one: What design patterns are closely relatedto this one? What are important differences? With which other patterns should this one be used?

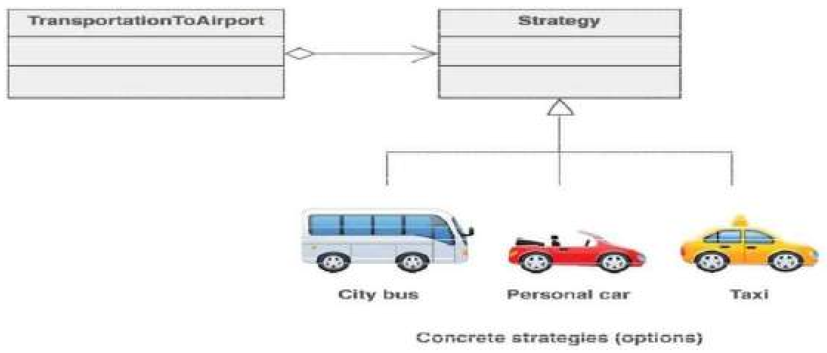
1.6 Implement Strategy Pattern

The strategy pattern (also known as the policy pattern). Strategy pattern **is one of the** behavioural design patterns**.** In Strategy pattern, a class behaviour or its algorithm can be changed at run time. Strategy pattern is used when we have multiple algorithms for a specific task and client decides the actual implementation to be used at runtime. In Strategy pattern, we create objects which represent various strategies and a context object whose behaviour varies as per its strategy object. The strategy object changes the executing algorithm of the context object for instance, a class that performs validation on incoming data may use a strategy pattern to select a validation algorithm based on the type of data, the source of the data, user choice, or other discriminating factors.

Strategy pattern is useful when we have multiple algorithms for specific task and we want our application tobe flexible to choose any of the algorithms at runtime for specific task.

IT/LP-II: D-13	Identification and Implementation of GOF pattern.	Page	5/7
Experiment No: 9	Semester – VI	Rev.: 00	Date:

Example: A Strategy defines a set of algorithms that can be used interchangeably. Modes of transportationto an airport are an example of a Strategy. Several options exist such as driving one's own car, taking a taxi, an airport shuttle, a city bus, or a limousine service. For some airports, subways and helicopters are also available as a mode of transportation to the airport. Any of these modes of transportation will get a traveller to the airport, and they can be used interchangeably. The traveller must chose the Strategy based on trade-offs between cost, convenience, and time.



1.6.1 : Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

1.6.2 Motivation:
Many algorithms exist for breaking a stream of text into lines.

Hard-wiring all such algorithms into the classes that require them isn‘t desirable for several Reasons. Clients that need line breaking get more complex if they include the line breaking code. That makes clients bigger and harder to maintain, especially if they support multiple line breaking algorithms. Different algorithms will be appropriate at different times. We don‘t want to support multiple line breaking algorithms if we don't use them all. It's difficult to add new algorithms and vary existing ones when line breaking is an integral part of a client. We

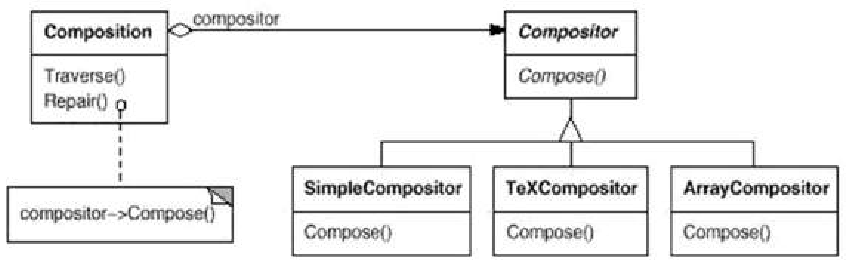


Figure 1: Example of Strategy

IT/LP-II: D-13	**Identification and Implementation of GOF pattern.**	Page	6/7
Experiment No: 9	Semester – VI	Rev.: 00	Date:

can avoid these problems by defining classes that encapsulate different line breaking algorithms. An algorithm that's encapsulated in this way is called a strategy.

Suppose a Composition class is responsible for maintaining and updating the line breaks of text displayed in a text viewer. Line breaking strategies aren't the class Composition. Instead, they are implemented separately by subclasses of the abstract Compositor class.

Compositor subclasses implement different strategies:

Simple Compositor implements a simple strategy that determines line breaks one at a time.

TeXCompositor implements the TeX algorithm for finding line breaks. This strategy tries to optimize line breaks globally, that is, one paragraph at a time. Array Compositor implements a strategy that selects breaks so that each row has a fixed number of items. It's useful for breaking a collection of icons into rows,

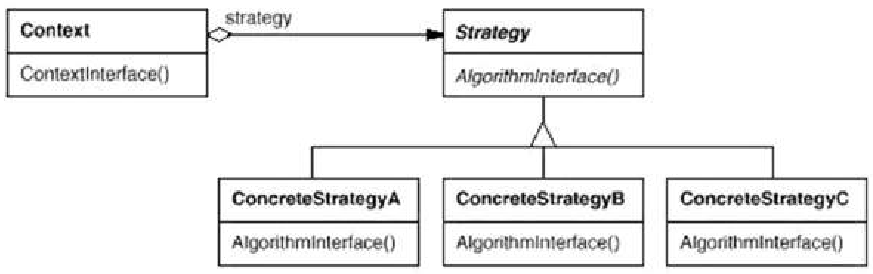
For example: A Composition maintains a reference to a Compositor object. Whenever a Composition reformats its text, it forwards this responsibility to its Compositor object. The client of Composition specifies which Compositor should be used by installing the Compositor it desires into the Composition.

1.6.3 Applicability:

Use the Strategy pattern when many related classes differ only in their behaviour. Strategies provide a way to configure a class with one of many behaviours.

You need different variants of an algorithm. For example, you might define algorithms reflecting different space/time trade-offs. Strategies can be used when these variants are implemented as a class hierarchy of algorithms. An algorithm uses data that clients shouldn't know about. Use the Strategy pattern to avoid exposing complex, algorithm-specific data structures. A class defines many behaviours, and these appear as multiple conditional statements in its operations. Instead of many conditionals, move related conditional branches into their own Strategy class.

1.6.4 Structure



IT/LP-II: D-13	Identification and Implementation of GOF pattern.	Page	7/7
Experiment No: 9	Semester – VI	Rev.: 00	Date:

Figure 1.2: UML structure of Strategy pattern

1.6.5 Participants

Strategy (Compositor) declares an interface common to all supported algorithms. Context uses this interface to call the algorithm defined by a Concrete Strategy. Concrete Strategy Implements the algorithm using the Strategy interface. Context (Composition): Is configured with a Concrete Strategy object. Maintains a reference to a Strategy object. May define an interface that lets Strategy access its data.

1.6.5 Collaborations

Strategy and Context interact to implement the chosen algorithm A context may pass all data required by the algorithm to the strategy when the algorithm is called. Alternatively, the context can pass itself as an argumentto Strategy operations. That lets the strategically back on the context as required. A context forwards requests from its clients to its strategy. Clients usually create and pass a Concrete Strategy object to the context; thereafter, clients interact with the context exclusively. There is often a family of Concrete Strategy classes fora client to choose from.

1.6.7 Consequences: The Strategy pattern has the following benefits and drawbacks:

1. ***Families of related algorithms -***Hierarchies of Strategy classes define a family of algorithms or behaviors for contexts to reuse. Inheritance can help factor out common functionality of the algorithms.
2. ***An alternative to sub Classing-***Inheritance offers another way to support a variety of algorithms or behaviors. You can subclass a Context class directly to give it different behaviors. But this hard-wire the behavior into Context. It mixes the algorithm implementation with Context's, making Context harder to understand, maintain, and extend. And you can't vary the algorithm dynamically. You wind up with many related classes whose only difference is the algorithm or behavior they employ. Encapsulating the algorithm in separate Strategy classes lets you vary the algorithm independently of its context, making it easier to switch, understand, and extend.
3. ***Strategies eliminate conditional statements-***The Strategy pattern offers an alternative to conditional statements for selecting desired behavior. When different behaviors are lumped into one class, it's hard to avoid using conditional statements to select the right behaviour. Encapsulating the behaviour in separate Strategy classes eliminates these conditional statements.
4. ***A choice of implementations: Strategies*** can provide different implementations of the *same* behavior.
   The client can choose among strategies with different time and space trade-offs.
5. ***Clients must be aware of different Strategies:*** The pattern has a potential drawback in that a

IT/LP-II: D-13	**Identification and Implementation of GOF pattern.**	Page	8/7
Experiment No: 9	Semester – VI	Rev.: 00	Date:

6. client must understand how Strategies differ before it can select the appropriate one. Clients might be exposed to implementation issues. Therefore, you should use the Strategy pattern only when the variation in behavior irrelevant to clients.
7. ***Communication overhead between Strategy and Context***: The Strategy interface is shared by all Concrete Strategy classes whether the algorithms they implement are trivial or complex. Hence it's likely that some Concrete Strategies won't use all the information passed to them through this interface; simple Concrete Strategies may use none of it! That means there will be times when the context creates and initializes parameters that never get used. If this is an issue, then you'll need tighter coupling between Strategy and Context.
8. ***Increased number of objects:*** Strategies increase the number of objects inane application. Sometimes you can reduce this overhead by implementing strategies as stateless objects that contexts can share. Any residual state is maintained by the context, which passes it in each request to the Strategy object. Shared strategies should not maintain state across invocations.

1.6.8 Implementation: For our example, we will try to implement a simple Shopping Cart where we have twopayment strategies – using Credit Card or using PayPal.

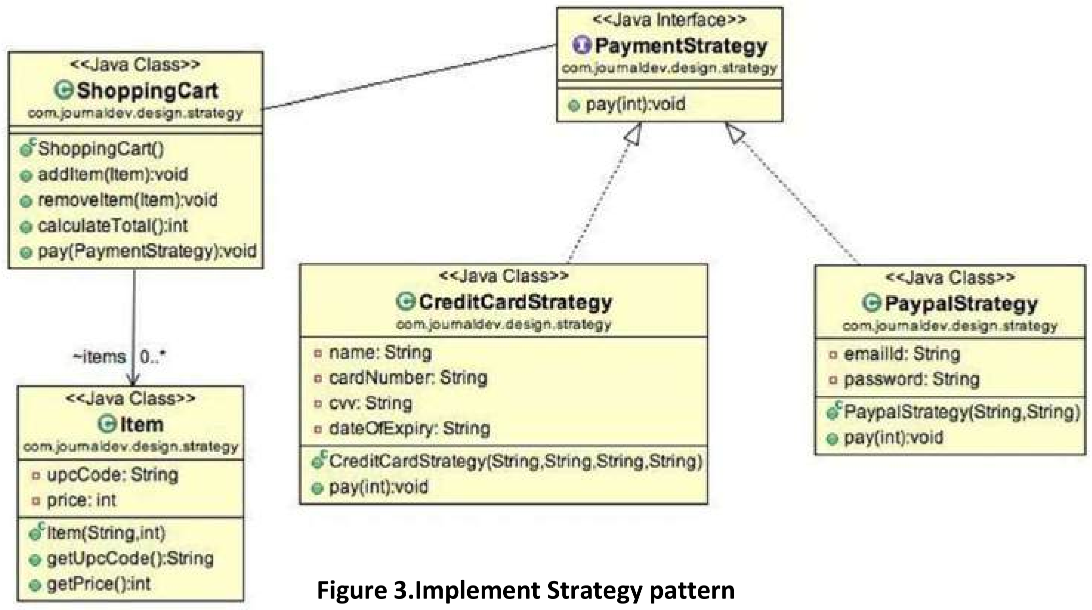


Figure 3.Implement Strategy pattern

IT/LP-II: D-13	**Identification and Implementation of GOF pattern.**	Page	9/7
Experiment No: 9	Semester – VI	Rev.: 00	Date:

1.6.9 Sample Code: First of all we will create the interface for our strategy, in our case to pay the amountpassed as argument.

```
package
com.journaldev.design.strategy;
public interface PaymentStrategy {

   public void pay(int amount);

}
```

Now we will have to create concrete implementations of algorithms for payment using credit/debit card orthrough paypal.

```
package com.journaldev.design.strategy;

public class CreditCardStrategy implements
   PaymentStrategy {private String name;

   private String
   cardNumber;private
   String cvv;

   private String dateOfExpiry;

   public CreditCardStrategy(String nm, String ccNum, String cvv, String
      expiryDate){this.name=nm;

      this.cardNumber=ccNum;
      this.cvv=cvv;
      this.dateOfExpiry=expiryDate;

   }
   @Override
   public void pay(int amount) {
      System.out.println(amount +" paid with credit/debit card");
   }
}
```

IT/LP-II: D-13	**Identification and Implementation of GOF pattern.**	Page	10/7
Experiment No: 9	**Semester – VI**	**Rev.: 00**	**Date:**

```
package com.journaldev.design.strategy;

public class PaypalStrategy implements
    PaymentStrategy {private String emailId;

    private String password;

    public PaypalStrategy(String email, String
        pwd){this.emailId=email;

        this.password=pwd;

    }

    @Override

    public void pay(int amount) {
        System.out.println(amount +" paid using Paypal.");

    } }
```

Now our algorithms are ready and we can implement Shopping Cart and payment method will require inputas Payment strategy.

```
package
com.journaldev.design.strategy;
public class Item {

    private String upcCode;

    private int price;

    public Item(String upc, int
        cost){this.upcCode=upc;
        this.price=cost;

    }

    public String
        getUpcCode() {return
        upcCode;

    }
```

IT/LP-II: D-13	**Identification and Implementation of GOF pattern.**	Page	11/7
Experiment No: 9	**Semester – VI**	**Rev.: 00**	**Date:**

```
   public intgetPrice()
       {return price;

   }

}
package
com.journaldev.design.strategy;
import java.text.DecimalFormat;

import
java.util.ArrayList;
import java.util.List;

public class ShoppingCart {

   //List of items
   List<Item> items;
   public
   ShoppingCart(){

       this.items=new ArrayList<Item>();
   }
   public void addItem(Item
       item){this.items.add(item);

   }
   public void removeItem(Item
       item){
       this.items.remove(item);

   }
   public
       intcalculateTotal(){
       int sum = 0;
```

IT/LP-II: D-13	**Identification and Implementation of GOF pattern.**	Page	12/7
Experiment No: 9	Semester – VI	Rev.: 00	Date:

```
    for(Item item : items){
      sum +=
      item.getPrice();

    }

    return sum;

  }
  public void pay(PaymentStrategypaymentMethod){

    int amount = calculateTotal();
    paymentMethod.pay(amount);

  }}
```

Notice that payment method of shopping cart requires payment algorithm as argument and doesn‘t store itanywhere as instance variable.

Let‘s test our setup with a simple program.

```
  public static void main(String[] args) {
    ShoppingCart cart = new
    ShoppingCart();Item item1 = new
    Item("1234",10);

    Item item2 = new
     Item("5678",40);
     cart.addItem(item1);
     cart.addItem(item2);

     //pay by paypal

     cart.pay(new PaypalStrategy("myemail@example.com", "mypwd"));

     //pay by credit card

     cart.pay(new CreditCardStrategy("Pankaj Kumar", "1234567890123456", "786",

"12/15")); ) }
```

Output of above program is:

IT/LP-II: D-13	Identification and Implementation of GOF pattern.	Page	13/7
Experiment No: 9	Semester – VI	Rev.: 00	Date:

1 50 paid using Paypal.
2 50 paid with credit/debit card

1.6.10 Known Uses: In the RTL System for compiler code optimization strategies define different register allocation schemes (Register Allocator) and instruction set scheduling policies (RISC scheduler, CISC scheduler). This provides flexibility in targeting the optimizer for different machine architectures.

1.6.11 Related Patterns: Flyweight: Strategy objects often make good flyweights.

1.7 Oral Questions

1. What is design pattern?
2. Use of pattern?

What is observer pattern?4.What GoF pattern?

1.8 Assignment Question

1.Explain Strategy pattern with example?

Conclusion: In this way we have studied basics of design pattern, types of pattern and basic of strategypattern, and implemented this pattern using object oriented language java