Suman Ramesh Tulsiani Charitable Trust's
**Suman Ramesh Tulsiani Technical Campus-
Faculty of Engineering**
Affiliated to Savitribai Phule Pune University, Approved by AICTE,DTE,MSBTE

<u>LABORATORY MANUAL</u>

Laboratory Practice IV

Department of Computer Engineering

BE (2019 Course)

Suman Ramesh Tulsiani Charitable Trust's

Suman Ramesh Tulsiani Technical Campus-Faculty of Engineering

Affiliated to Savitribai Phule Pune University, Approved by AICTE,DTE,MSBTE

LABORATORY MANUAL

AY: 2024-25

Laboratory Practice IV

BE Computer
Engineering Semester –I

Subject Code – 410247

Prepared By:
Prof. Pooja Pawar Kirtikar

(Assistant Professor, Dept of Computer)

SRTCT's

Suman Ramesh Tulsiani Technical CampusFaculty of Engineering

Old Mumbai - Pune Hwy,

KhamshetPune,Maharashtra

410405

DEPARTMENTOF COMPUTER ENGINEERING

Certificate

This is to certify that the LABORATORY MANUAL

Entitled

Laboratory Practice IV

Submitted by

Mayur Narsale (222372)

Is approved by **Prof. Pooja Pawar** for submission. It is certified further that, to the best of-my knowledge, the practical's represents work carried out by my students as the partial fulfillment for B.E. Computer Engineering (Semester I) Laboratory Practice IV work as prescribed by the University of Pune for the academic year 2024-25.

[Prof. Pooja Pawar]	**[Prof. Dr.D.E.Upasani]**	**Prof. (Dr.) J. B.Sankpal**
Subject Coordinator	Head of Department	**Principal, SRTTC-FoE**

CONTENTS

Sr. No.	Problem Statement	Date	Page No.	Faculty Sign
	410244(D) Object Oriented Modeling and Design			
1	Draw state model for telephone line, with various activities.		1-7	
2	Draw basic class diagram to identify and describe key concepts likeclasses, types in your system and their relationships.		7-12	
3	Draw one or more Use Case diagrams for capturing and representing requirements of the system. Use case diagrams must include template showing description and steps of the Use Case for various scenarios		13-18	
4	Draw activity diagrams to display either business flows or like flow charts.		19-21	
5	Draw deployment diagrams to model the runtime architecture of yoursystem.		22-24	
6	Draw component diagrams assuming that you will build your system reusing existing components along with a few new ones		25-28	
7	Mini Project		29	
	410245(D) Software Testing and Quality Assurance			
1	Write TEST Scenario for Gmail Login Page			
2	Write TEST Scenario for Inbox Functionality			
3	Write Test cases in excel sheet for Social Media application or website			
4	Installation of Selenium grid and selenium Web driver			
5	Prepare Software requirement specification for any project or problemstatement			

EXPERIMENT No: 1

Title: Draw state model for telephone line, with various activities.

SW/Tool Required: Star UML

Relevant Theory:

State Chart Diagram Description

The UML state diagrams are directed graphs in which nodes denote states and connectors denote state transitions. In UML, states are represented as rounded rectangles labeled with state names. The transitions, represented as arrows, are labeled with the triggering events followed optionally by the list of executed actions. The initial transition originates from the solid circle and specifies the default state when the system first begins. Every state diagram should have such a transition, which should not be labeled, since it is not triggered by an event. The initial transition can have associated actions.

Purpose of State chart Diagrams

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

State chart diagram describes the flow of control from one state to another state. States are defined as acondition in which an object exists and it changes when some event is triggered. The most important purpose of State chart diagram is to model lifetime of an object from creation to termination.

State chart diagrams are also used for forward and reverse engineering of a system. However, the main purpose isto model the reactive system.

Following are the main purposes of using State chart diagrams −

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

How to Draw a State chart Diagram?

State chart diagram is used to describe the states of different objects in its life cycle. Emphasis is placed on the state changes upon some internal or external events. These states ofobjects are important to analyze and implement them accurately.

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

Before drawing a State chart diagram we should clarify the following points –

- Identify the important objects to be analyzed.
- Identify the states.
- Identify the events.

Where to Use State chart Diagrams?

State chart diagrams are used to model the dynamic aspect of a system like other fourdiagrams discussed in this tutorial. However, it has some distinguishing characteristics for modeling the dynamic nature.

State chart diagram defines the states of a component and these state changes are dynamic in nature. Its specific purpose is to define the state changes triggered by events. Events are internalor external factors influencing the system.

State chart diagrams are used to model the states and also the events operating on the system. When implementing a system, it is very important to clarify different states of an object during its life time and State chart diagrams are used for this purpose. When these states and events are identified, they are used to model it and these models are used during the implementation of thesystem.

If we look into the practical implementation of State chart diagram, then it is mainly used to analyze the object states influenced by events. This analysis is helpful to understand the systembehavior during its execution.

The main usage can be described as −

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

The main purposes of using State chart diagrams:

- To model dynamic aspect of a system.
- To model life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model states of an object Steps to prepare state chart diagram
- Identify important objects to be analyzed.

- Identify the states.
- Identify the events.

State Notation:

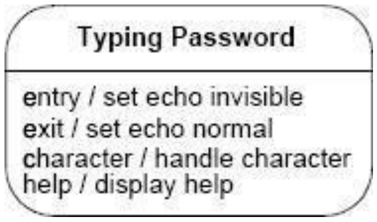


Figure 01- Building Blocks of a State chart Diagram

State

A state is any "distinct" stage that an object (system) passes through in it's lifetime. An object remains in a givenstate for finite time until "something" happens, which makes it to move to another state. All such states can be broadly categorized into following three types:

- **Initial**: The state in which an object remain when created
- **Final**: The state from which an object do not move to any other state [optional]
- **Intermediate**: Any state, which is neither initial, nor final

As shown in figure-02, an initial state is represented by a circle filled with black. An intermediate state is depicted by a rectangle with rounded corners. A final state is represented by a unfilled circle with an inner black-filled circle.

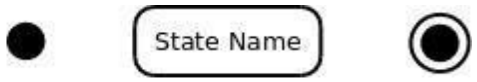


Figure 02- Representation of initial, intermediate, and final states of a state chart diagram

Intermediate states usually have two compartments, separated by a horizontal line, called the namecompartment and internal transitions compartment. They are described below:

- **Name compartment**: Contains the name of the state, which is a short, simple, descriptive string
- **Internal transitions compartment**: Contains a list of internal activities performed as long as the system is in this state

The internal activities are indicated using the following syntax: action-label / action-expression. Action labelscould be any condition indicator. There are, however, four special action labels:

- **Entry**: Indicates activity performed when the system enter this state
- **Exit**: Indicates activity performed when the system exits this state
- **Do**: indicate any activity that is performed while the system remain in this state or until the action expression results in a completed computation
- **Include**: Indicates invocation of a sub-machine

Any other action label identify the event (internal transition) as a result of which the corresponding action is triggered. Internal transition is almost similar to self-transition, except that the former doesn't result in executionof entry and exit actions. That is, system doesn't exit or re-enter that state. Figure-03

shows the typical state in a state chart diagram States could again be either simple or composite (a state congaing other states). Here, however, we will deal only with simple states.

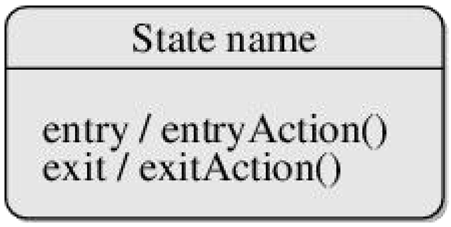


Figure 03. syntax for representing a typical (intermediate) state

Transition

Transition is movement from one state to another state in response to an external stimulus (or any internal event). A transition is represented by a solid arrow from the current state to the next state. It is labeledby: event [guard-condition]/[action-expression], where

- **Event** is the what is causing the concerned transition (mandatory) -- Written in past tense
- **Guard-condition** is (are) precondition(s), which must be true for the transition to happen [optional]
- **Action-expression** indicate action(s) to be performed as a result of the transition [optional]

It may be noted that if a transition is triggered with one or more guard-condition(s), which evaluate to false, the system will continue to stay in the present state. Also, not all transitions do result in a state change. For example,if a queue is full, any further attempt to append will fail until the delete method is invoked at least once. Thus, state of the queue doesn't change in this duration.

Action

As mentioned in , actions represents behavior of the system. While the system is performing any action for thecurrent event, it doesn't accept or process any new event. The order in which different actions are executed, is given below:

1. Exit actions of the present state
2. Actions specified for the transition
3. Entry actions of the next state

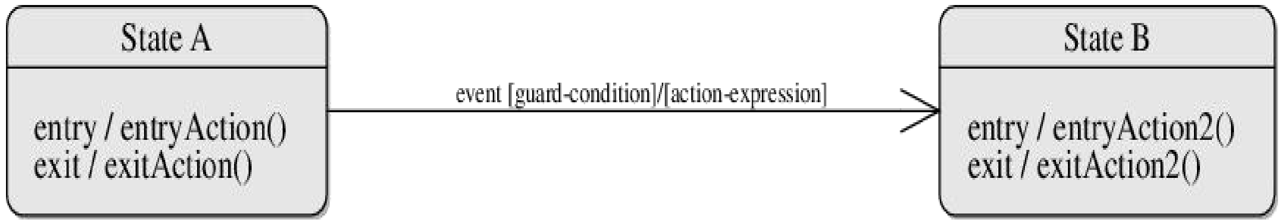
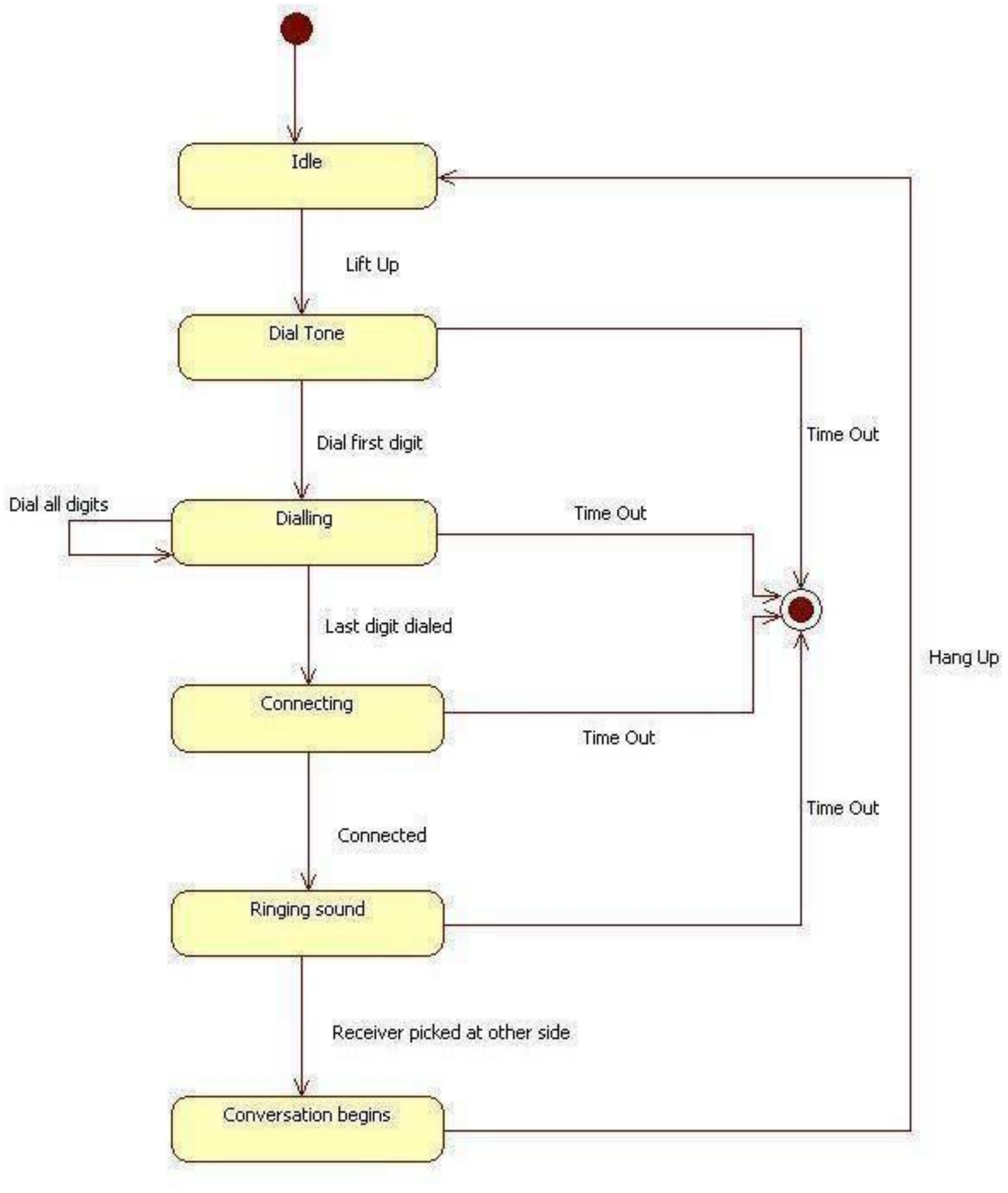


Fig-04: A statechart diagram showing transition from state A to B

State chart diagram for telephone line :



<u>Conclusion:</u>

In this experiment, we studied the state chart diagram for telephone line, with various activities.

EXPERIMENT No: 2

Title:-Draw basic class diagram to identify and describe key concepts like classes, types in your system and their relationships.

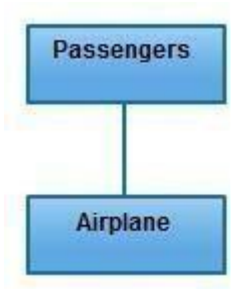
SW/Tool Required: Star UML

Relevant Theory:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

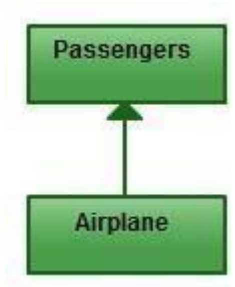
- **Class Diagram Relationships:**
  1. Association
  2. Directed Association
  3. Reflexive Association
  4. Multiplicity
  5. Aggregation
  6. Composition
  7. Inheritance/Generalization
  8. Realization

- **Association**



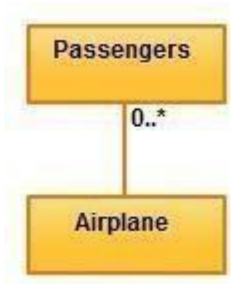
Association is a broad term that encompasses just about any logical connection or relationship between classes. For example, passenger and airline may be linked as above.

- **Directed Association**



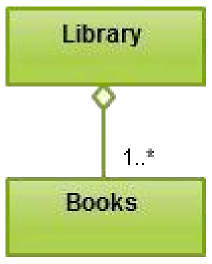
Directed Association refers to a directional relationship represented by a line with an arrowhead. The arrowhead depicts a container-contained directional flow.

- **Multiplicity**



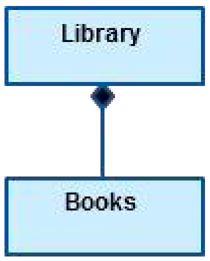
Multiplicity is the active logical association when the cardinality of a class in relation to another is being depicted. For example, one fleet may include multiple airplanes, while one commercial airplane may contain zero to many passengers. The notation 0..* in the diagram means “zero to many”.

- **Aggregation**



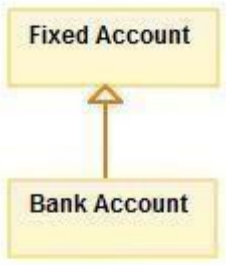
Aggregation refers to the formation of a particular class as a result of one class being aggregated or built as a collection. For example, the class “library” is made up of one or more books, among other materials. In aggregation, the contained classes are not strongly dependent on the lifecycle of the container. In the same example, books will remain so even when the library is dissolved. To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class. To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

- **Composition**



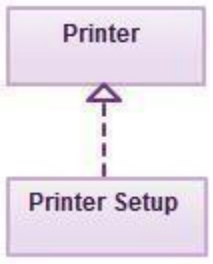
The composition relationship is very similar to the aggregation relationship. with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class. That is, the contained class will be obliterated when the container class is destroyed. For example, a shoulder bag's side pocket will also cease to exist once the shoulder bag is destroyed. To show a composition relationship in a UML diagram, use a directional line connecting the two classes, with a filled diamond shape adjacent to the container class and the directional arrow to the contained class.

- **Inheritance / Generalization**

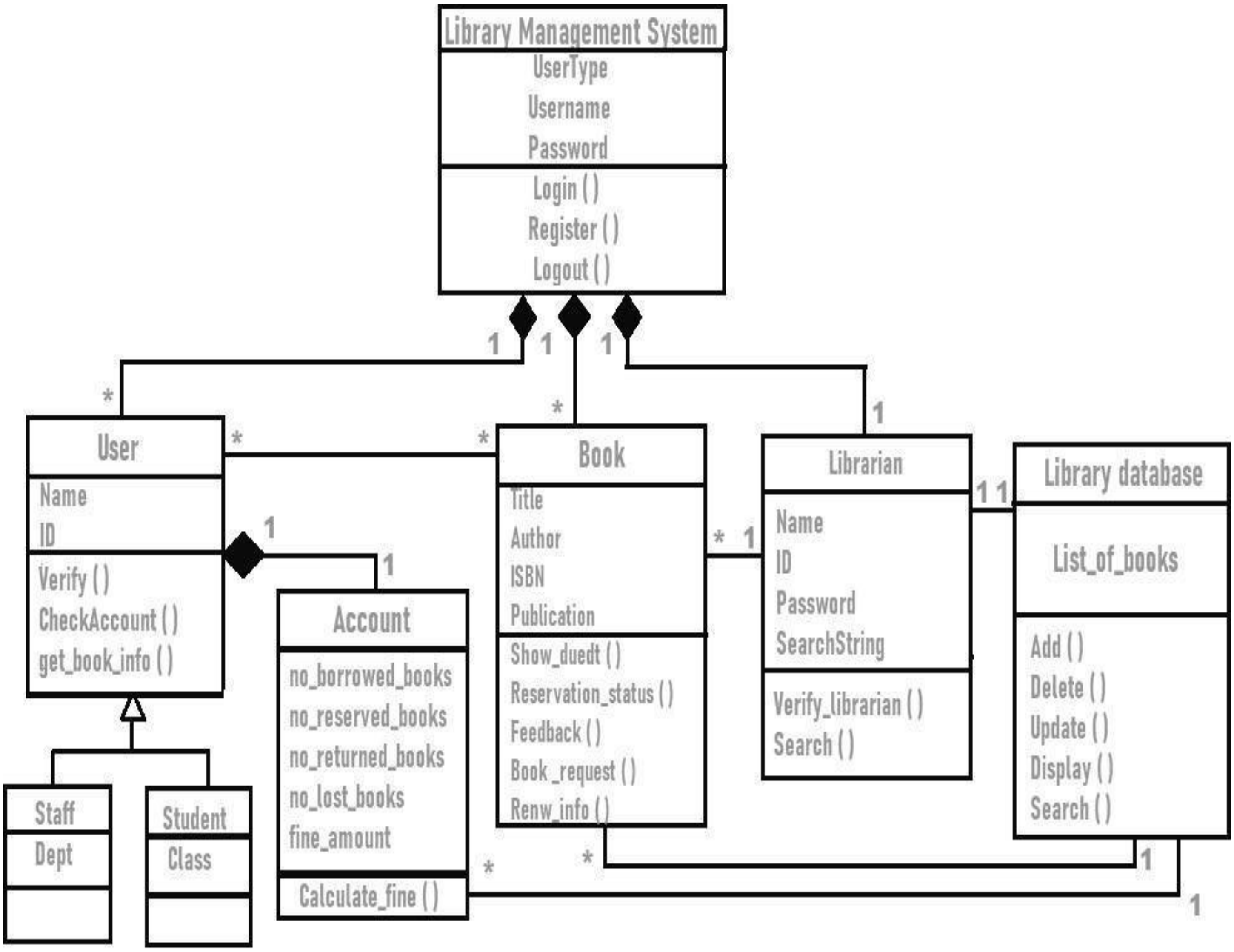


Inheritance refers to a type of relationship wherein one associated class is a child of another by virtue of assuming the same functionalities of the parent class. In other words, the child class is a specific type of the parent class. To show inheritance in a UML diagram, a solid line from the child class to the parent class is drawn using an unfilled arrowhead.

- **Realization**



Realization denotes the implementation of the functionality defined in one class by another class. To show the relationship in UML, a broken line with an unfilled solid arrowhead is drawn from the class that defines the functionality of the class that implements the function. In the example, the printing preferences that are set using the printer setup interface are being implemented by the printer.



CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM

Conclusion:

Thus we have studied how to use class diagrams to identify and describe key concepts like classes, types in system and their relationships.

EXPERIMENT NO:3

Title: Draw one or more Use Case diagrams for capturing and representing requirements of the system. Use case diagrams must include template showing description and steps of the Use Case for various scenarios.

SW/Tool Required: Star UML

Relevant Theory: **Guideline: To Find Use Cases**

Use cases are defined to satisfy the goals of the primary actors. Hence, the basic procedure is:

Step 1: Choose the system boundary. Is it just a software application, the hardware and application as a unit, that plus a person using it, or an entire organization?

Step 2: Identify the primary actors those that have goals fulfilled through using services of the system.The following questions help identify others that may be missed:

Who starts and stops the system?	Who does system administration?
Who does user and security management?	Is "time" an actor because the system does something in response to a time event?
Is there a monitoring process that restarts the systemif it fails?	Who evaluates system activity or performance?
How are software updates handled? Push or pull update?	Who evaluates logs? Are they remotely retrieved?
In addition to human primary actors, are there any	Who gets notified when there are
Who starts and stops the system?	Who does system administration?
External software or robotic systems that call uponservices of the system?	Erors or failures?

Step 3: Identify the goals for each primary actor.

For example, use this table to prepare actor goal list

Actor	Goal
Participant	Play level 1
	Play level 2
	Submit answers
...	...

Step 4: Define Use Cases

In general, define one use case for each user goal. Name the use case similar to the user goal. Start the name of use cases with a verb.

A common exception to one use case per goal is to collapse CRUD (create, retrieve, update, delete) separate goals into one CRUD use case, idiomatically called Manage <X>. For example, the goals "edit user," "delete user," and so forth are all satisfied by the Manage Users use case.

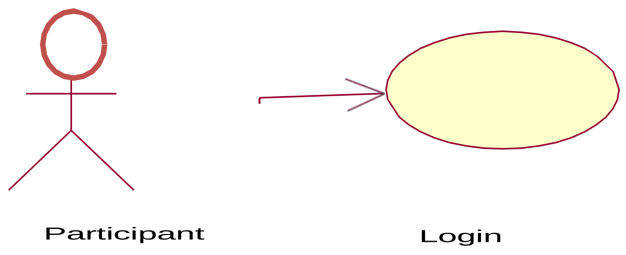
UML Notation:

Actor : An Actor, as mentioned, is a user of the system, and is depicted using a stick figure. The role of the user is written beneath the icon. Actors are not limited to humans. If a system communicates with another application, and expects input or delivers output, then that application can also be considered an actor.

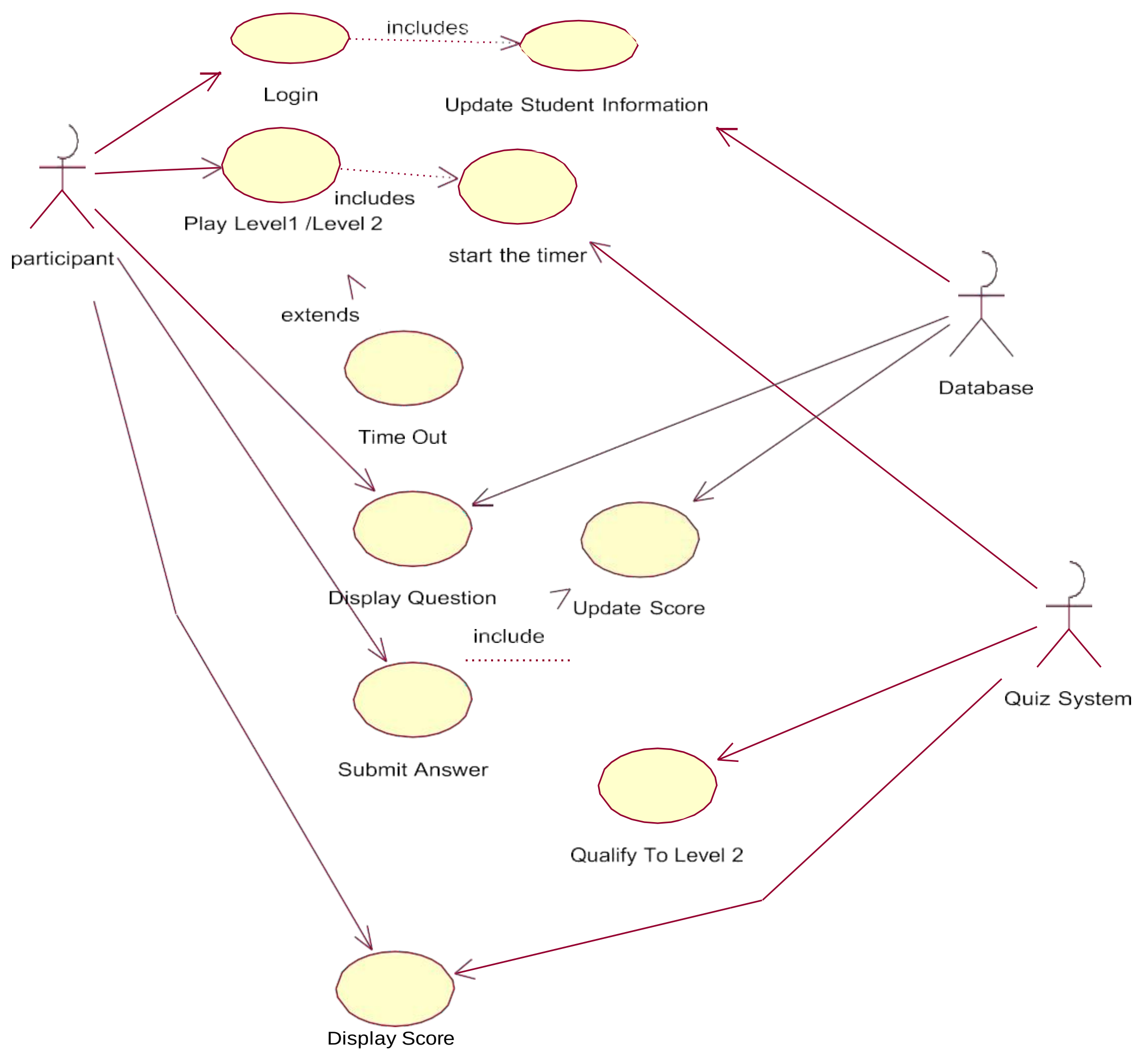
Use Case: A Use Case is functionality provided by the system, typically described as verb+object (e.g. Register Car, Delete User). Use Cases are depictedwith an ellipse. The name of the use case is written within the ellipse.

Association: Associations are used to link Actors with Use Cases, and indicate that an Actor participates in the Use Case in some form. Associations are depicted by a line connecting the Actor and the Use Case.

Actor Association Use case



Use-case Diagram: Quiz System



- **Use cases**

The system will consist of Login screen to authenticate the participants whose information is updated in the database. On starting the quiz, timer is started to maintain the timings. In Level 1 /Level 2 Questions with four options are displayed sequentially .User select the answer and move to the next question. Finally he/she selects the Submit answers which updates the marks and displays the score to the participants. If he is playing Level 1 and qualified for level 2, then next level questions are displayed otherwise Not Qualified Message is displayed.

Functional Requirements

Use Case: Login Brief Description

The use case describes how a Participant logs into the Quiz System

Use Case Section	Comment
Use Case Name	Login
Scope	Quiz System
Level	"user-goal"

Use Case Section	Comment
Primary Actor	Participant
Stakeholders and Interest list	- Participant: logs into the Quiz System . . .
Preconditions	None
Success Guarantee/ Post condition	If the use case was successful, the actor is now logged intothe system. If not, the system State is unchanged.
Main Success Scenario	1. The System requests the actor to enter his/her nameand password 2. The actor enters his/her name and password 3. The System validates the entered name and passwordand logs the actor into the System
Extensions	3a.If the pwd is wrong, user is allowed for 3 attempts
Special Requirements	Timer
Technology and Data Variations List	-
Frequency of Occurrence	Could Be nearly Continuous
Miscellaneous	If the connection is terminated before the form is submitted, the fields are cleared and the Server is returnedto the wait state. The Administrator can make the system not to get updatedby others.

Non-functional requirements

Non-functional requirements are often called qualities of a system. Other terms for non-functional requirements are "constraints", "quality attributes", "quality goals", "quality of service requirements" and "non-behavioral requirements".

Non-functional requirements, can be divided into two main categories:

- Execution qualities, such as security and usability, which are observable at run time.
- Evolution qualities, such as testability, maintainability, extensibility and scalability, which are embodied in the static structure of the software system

Functionality: Multiple users must be able to perform their work concurrently. If the participant has completed 30 Minutes allotted for him/her, he or she should be notified with the message “your time slotis over”.

Usability: The desktop user-interface shall be Windows 95/98/2000/xp compliant.

Reliability: The System should function properly for allotted time slot and produces score card with no more than 10% down time.

Performance

- The System shall support up to 100 simultaneous users against the central database at any given time and up to 100 simultaneous users against the local servers at any one time.
- The System must be able to complete 80% of all transactions within 2 minutes.

Security

- The System should secure so that only registered participants can takepart in Quiz.
- Once the participant had submitted a answer, he/she can’t change the answer later.

Design Constraints:

The system shall provide a window-based desktop interface

Conclusion:

In this experiment, we studied the Use Case diagrams for capturing and representing requirements of the system comprising the use case diagram and use case scenarios.

EXPERIMENT No: 4

Title: Draw activity diagrams to display either business flows or like flow charts.

SW/Tool Required: Star UML

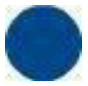
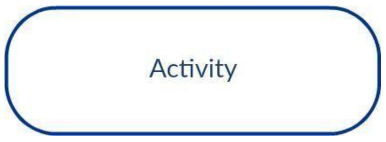
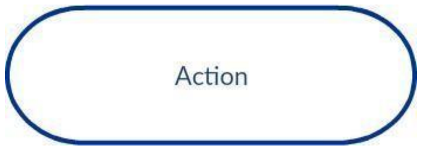
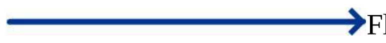
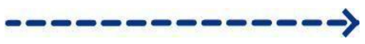
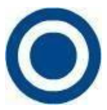
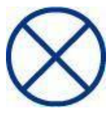
Relevant Theory:

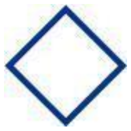
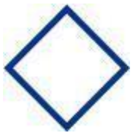
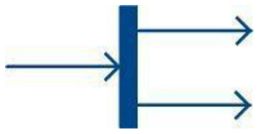
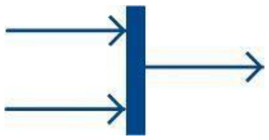
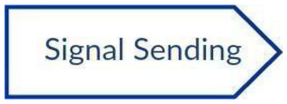
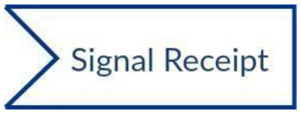
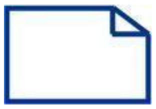
What is an Activity diagram?

A UML activity diagram helps to visualize a certain use case at a more detailed level. It is a behavioral diagram that illustrates the flow of activities through a system.
UML activity diagrams can also be used to depict a flow of events in a business process. They can be used to examine business processes in order to identify their flow and requirements.

Activity Diagram Symbols

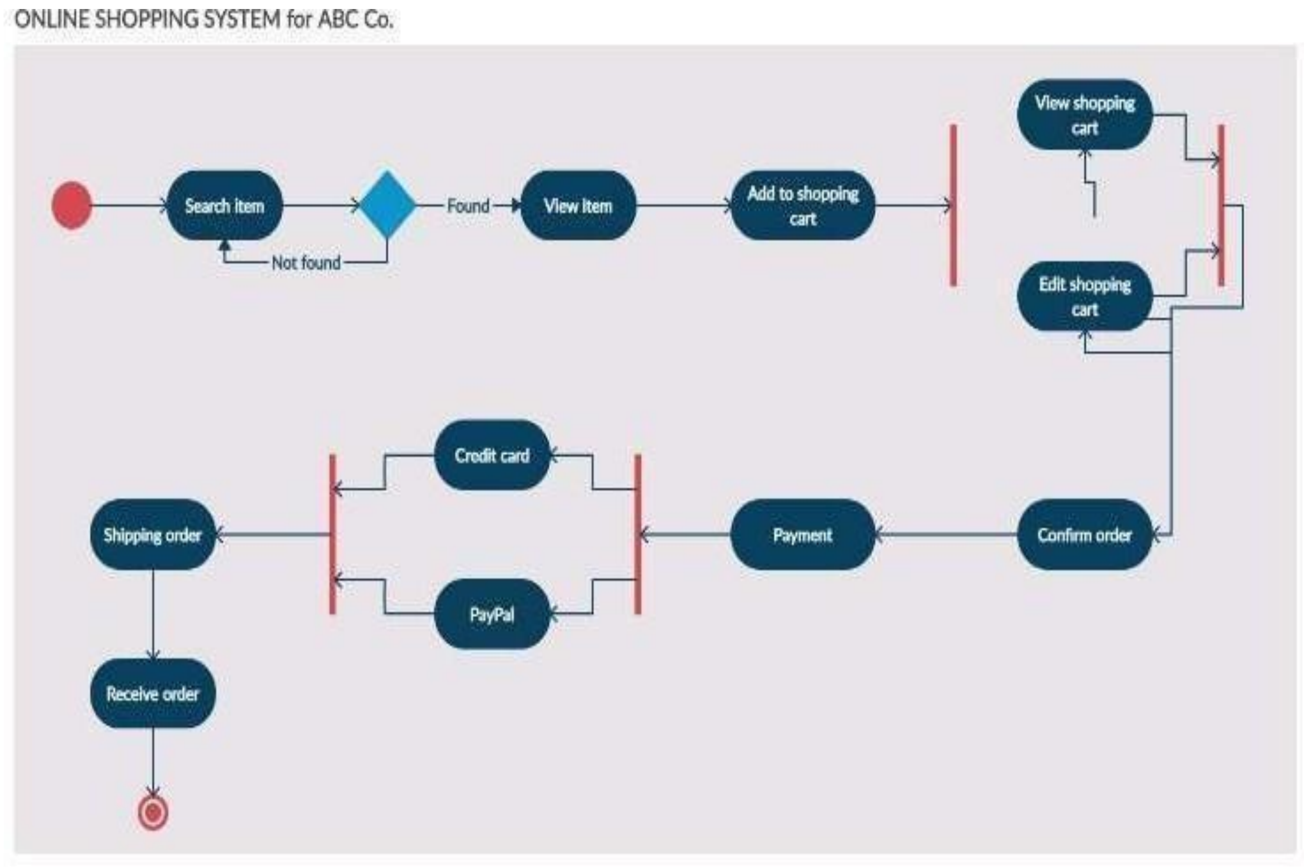
UML has specified a set of symbols and rules for drawing activity diagrams. Following are the commonly used activity diagram symbols with explanations.

Symbol	Name	Use
	Start/ Initial Node	Used to represent the starting point or the initial state of an activity
	Activity / Action State	Used to represent the activities of the process
	Action	Used to represent the executable sub-areas of an activity
 Flow /	Control Edge	Used to represent the flow of control from one action to the other
 Flow /	Object Control Edge	Used to represent the path of objects moving through the activity
	Activity Final Node	Used to mark the end of all control flows within the activity
	Flow Final Node	Used to mark the end of a single control flow

	Decision Node	Used to represent a conditional branch point with one input and multiple outputs
	Merge Node	Used to represent the merging of flows. It has several inputs, but one output.
	Fork	Used to represent a flow that may branch into two or more parallel flows
	Merge	Used to represent two inputs that merge into one output
	Signal Sending	Used to represent the action of sending a signal to an accepting activity
	Signal Receipt	Used to represent that the signal is received
	Note/ Comment	Used to add relevant comments to elements

Activity Diagram Examples

Activity Diagram for Online Shopping System



Conclusion: Thus we have studied how to use activity diagram to find out behavior of any system. System.

EXPERIMENT No: 5

Title: Draw deployment diagrams to model the runtime architecture of your system.

SW/Tool Required: Star UML

Relevant Theory:

What is Deployment Diagram:-

A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them. Deployment diagrams are typically used to visualize the physical hardware and software of a system. Using it you can understand how the system will be physically deployed on the hardware.

Purpose of Deployment Diagrams:-

- They show the structure of the run-time system
- They capture the hardware that will be used to implement the system and the links between differentitems of hardware.
- They model physical hardware elements and the communication paths between them
- They can be used to plan the architecture of a system.
- They are also useful for Document the deployment of software components or nodes

Deployment Diagram at a Glance:-

Deployment diagrams are important for visualizing, specifying, and documenting embedded, client/server, and distributed systems and also for managing executable systems through forward and reverse engineering.

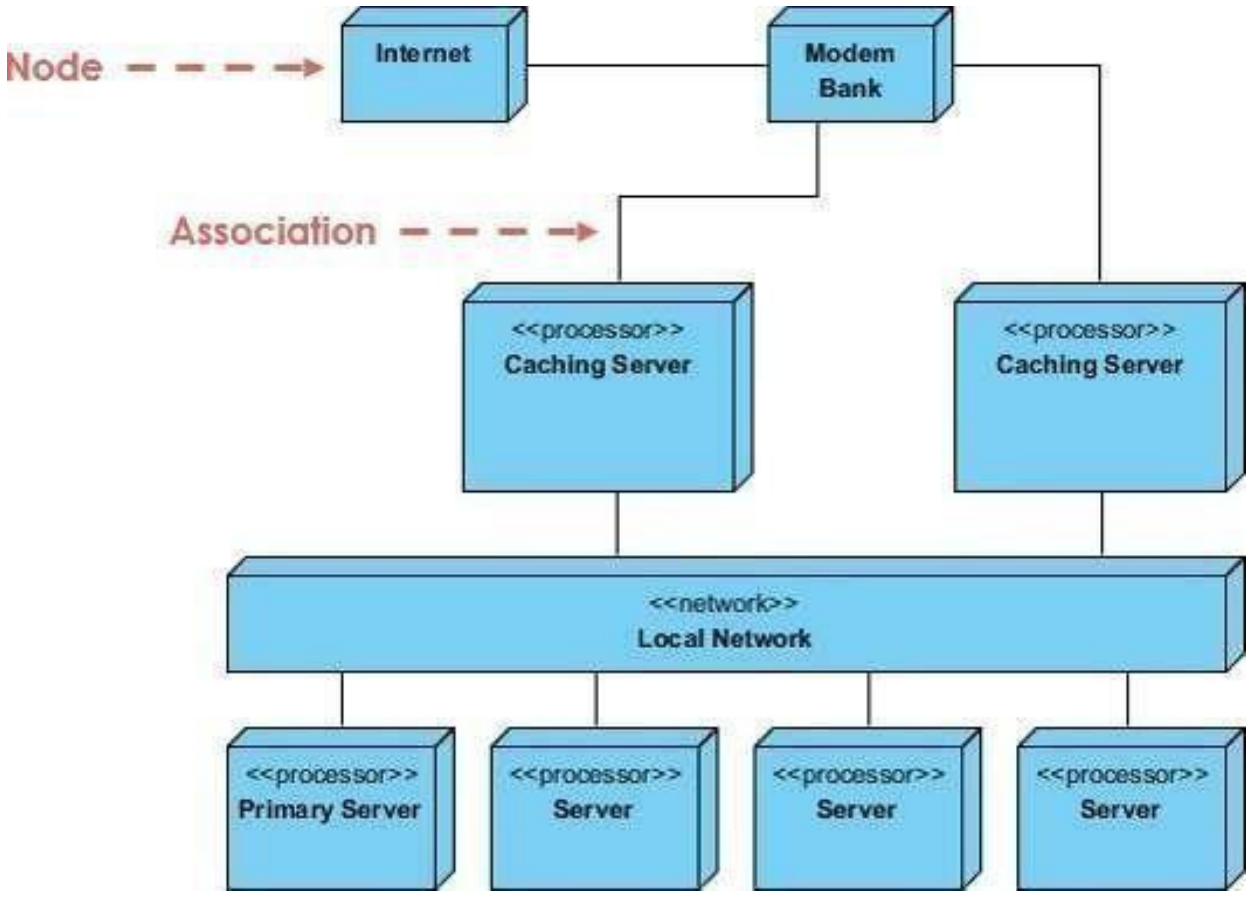
A deployment diagram is just a special kind of class diagram, which focuses on a system's nodes. Graphically, a deployment diagram is a collection of vertices and arcs. Deployment diagrams commonly contain:

Nodes

- 3-D box represents a node, either software or hardware
- HW node can be signified with <<stereotype>>
- Connections between nodes are represented with a line, with optional <<stereotype>>
- Nodes can reside within a node

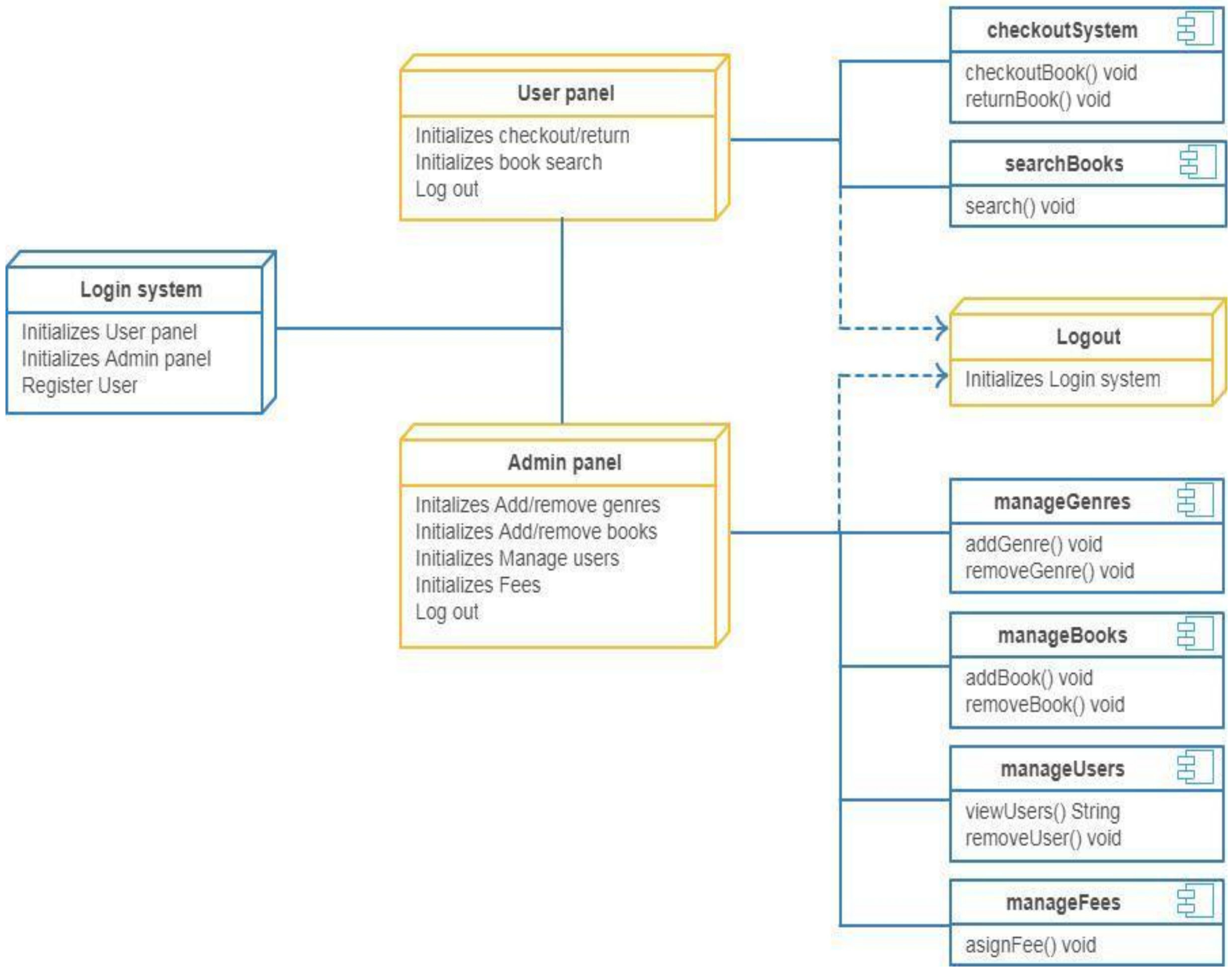
Other Notations

- Dependency
- Association relationships.
- May also contain notes and constraints.


Node
Internet
Modem Bank
Association
<<processor>> Caching Server
<<processor>> Caching Server
<<network>> Local Network
<<processor>> Primary Server
<<processor>> Server
<<processor>> Server
<<processor>> Server

Deployment diagram

Deployment Diagram for Library Management System



<u>Conclusion</u>: Thus we have studied what is deployment diagram and how to use it and visualize the system.

EXPERIMENT No: 6

Title: Draw component diagrams assuming that you will build your system reusing existing components along with a few new ones

SW/Tool Required: Star UML

Theory: The purpose of a component diagram is to show the relationship between different components in a system. For the purpose of UML 2.0, the term "component" refers to a module of classes that represent independent systems or subsystems with the ability to interface with the rest of the system.

There exists a whole development approach that revolves around components: component-based development (CBD). In this approach, component diagrams allow the planner to identify the different components so the whole system does what it's supposed to do.

More commonly, in an OO programming approach, the component diagram allows a senior developer to group classes together based on common purpose so that the developer and others can look at a software development project at a high level.

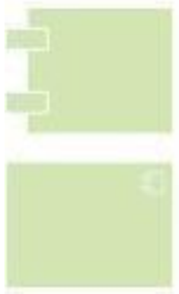
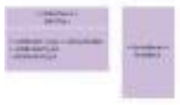
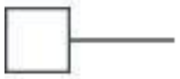
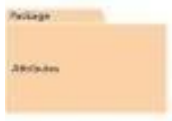
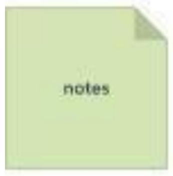
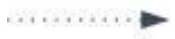
How to use component diagrams

A component diagram in UML gives a bird's-eye view of your software system. Understanding the exact service behavior that each piece of your software provides will make you a better developer. Component diagrams can describe software systems that are implemented in any programming language or style.

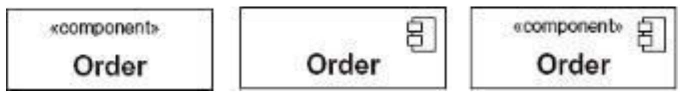
UML is a set of conventions for object-oriented diagrams that has a wide variety of applications. In component diagrams, the Unified Modeling Language dictates that components and packages are wired together with lines representing assembly connectors and delegation connectors.

Component diagram shapes and symbols

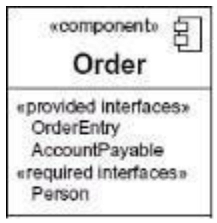
Component diagrams range from simple and high level to detailed and complex. Either way, you'll want to familiarize yourself with the appropriate UML symbols. The following are shape types that you will commonly encounter when reading and building component diagrams:

Symbol	Name	Description
	Component symbol	An entity required to execute a stereotype function. A component provides and consumes behavior through interfaces, as well as through other components. Think of components as a type of class. In UML 1.0, a component is modeled as a rectangular block with two smaller rectangles protruding from the side. In UML 2.0, a component is modeled as a rectangular block with a small image of the old component diagram shape.
	Node symbol	Represents hardware or software objects, which are of a higher level than components.
	Interface symbol	Shows input or materials that a component either receives or provides. Interfaces can be represented with textual notes or symbols, such as the lollipop, socket, and ball-and-socket shapes.
	Port symbol	Specifies a separate interaction point between the component and the environment. Ports are symbolized with a small square.
	Package symbol	Groups together multiple elements of the system and is represented by file folders in Lucidchart. Just as file folders group together multiple sheets, packages can be drawn around several components.
	Note symbol	Allows developers to affix a meta-analysis to the component diagram.
	Dependency symbol	Shows that one part of your system depends on another. Dependencies are represented by dashed lines linking one component (or element) to another.

How to use component shapes and symbols



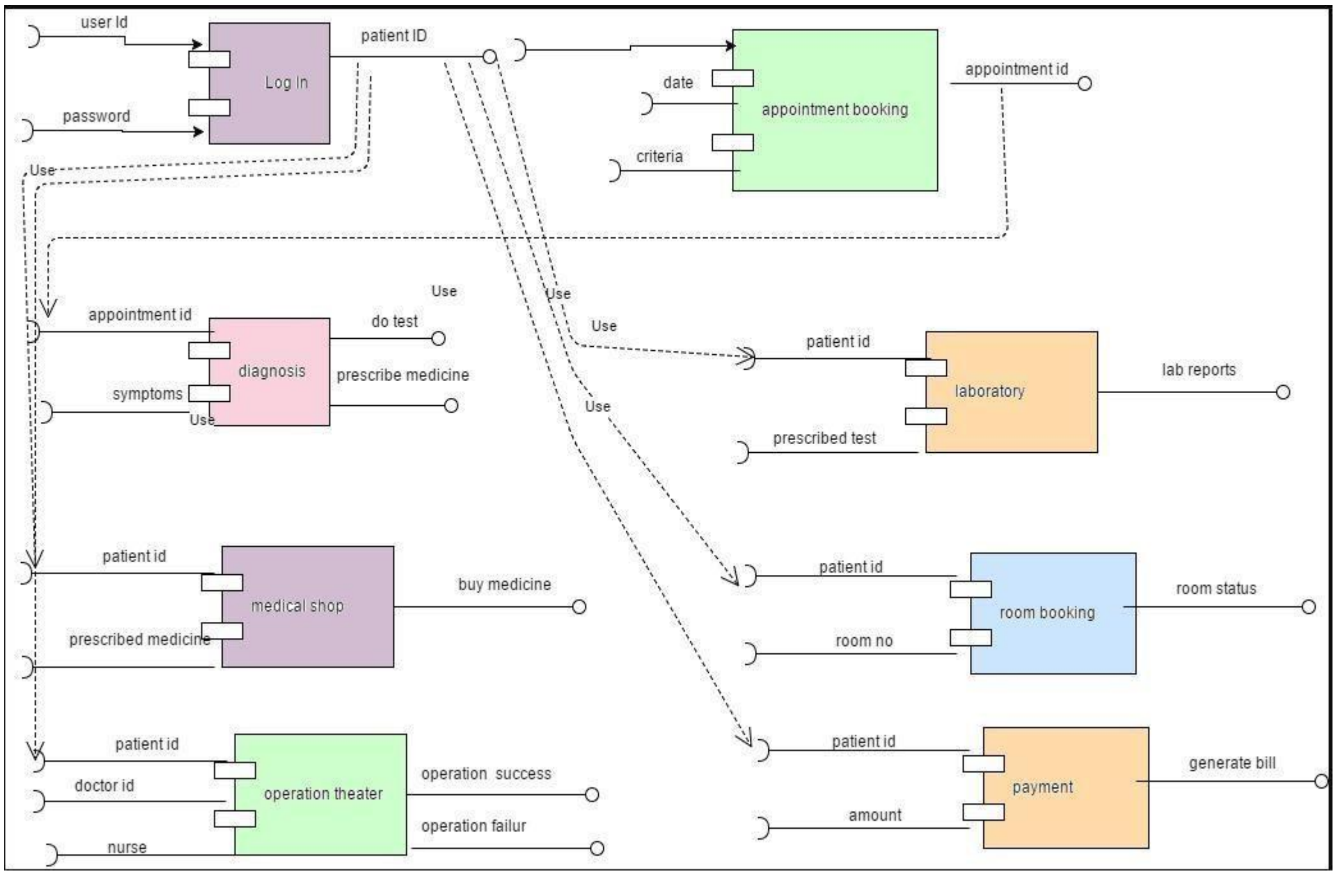
There are three popular ways to create a component's name compartment. You always need to include the component text inside the double angle brackets and/or the component logo. The distinction is important because a rectangle with just a name inside of it is reserved for classifiers (class elements).



As with the class notation, components also have an optional space to list interfaces, similar to the way you add attributes and methods to class notation. Interfaces represent the places where the groups of classes in the component communicate with other system components. An alternative way to represent interfaces is by extending symbols from the component box. Here is a quick rundown of the most commonly used symbols.

Conclusion:- Thus we have studied and drawn component diagrams

Output:-


user Id
password
Log In
patient ID
Use
appointment booking
date
criteria
appointment id
diagnosis
appointment id
symptoms
do test
prescribe medicine
Use
laboratory
patient id
prescribed test
lab reports
medical shop
patient id
prescribed medicine
buy medicine
room booking
room no
room status
operation theater
doctor id
nurse
operation success
operation failur
payment
amount
generate bill

Mini Project

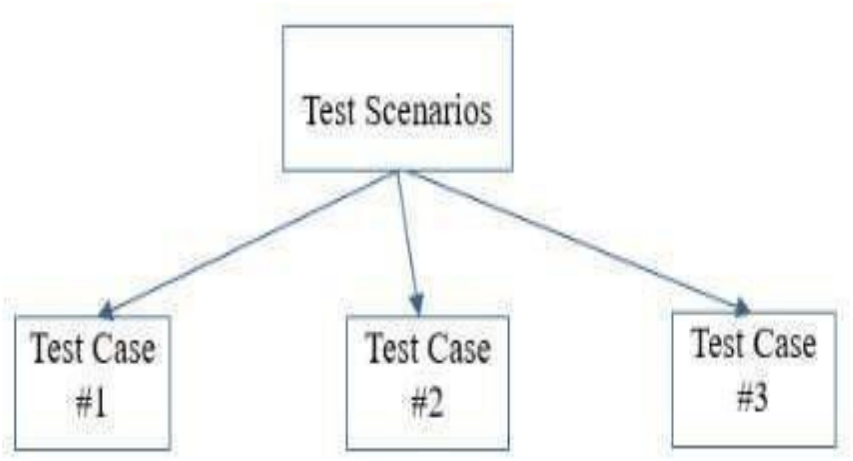
EXPERIMENT NO: 1

Problem Statement:

Write TEST Scenario for Gmail Login Page.

Theory:

Test Scenario is the function that can be tested. It is also called **Test Condition** or **Test Possibility**. A single Test Scenario can cover one or more Test Cases.



Test Scenario** is **'What to be tested'** and **Test Case** is **'How to be tested'.

As a tester, you can put yourself as an end-user and find real situations and possible functionalities of the application being tested.

Why Write Test Scenarios?

Test Scenarios are created for the following reasons:

- To ensure the completion of test coverage.
- Validate that the software is working properly for each Use Case.
- Improve the User Experience.
- Identify the most important end-to-end transactions.
- Study the terminal function and help to build test cases

Who and When Write Test Scenarios?

Typically, Testers are the ones who are responsible for creating Test Scenarios. In case of some complex and critical applications like Banking Software's, Business Analysts or Test Leads provide the Test Scenarios to Testers. But again depends, and vary from organization to organization.

As, Test Scenarios tell us what needs to be tested, therefore always written before test cases.

Steps to writing Test Scenarios:

As a tester, you can follow these five steps to create a test scenario:

1. Go through all the requirement documents available like BRD, SRS, and FSD to understand the functionalities of the application to be tested.
2. For each requirement, find out possible actions and goals of the user.
3. List down all the possible functionalities/scenarios that need to consider for each requirement. (Test Scenarios Template is provided to download)
4. Once all possible test scenarios are listed, create a Traceability Matrix to ensure that all requirements have a corresponding test scenario.
5. Review the Test Scenario document and Traceability Matrix with Test Lead / Business Analyst

Conclusion:

Test Scenario for Gmail Login page written successfully.

EXPERMIENT NO: 2

Problem Statement:

Write TEST Scenario for Inbox Functionality

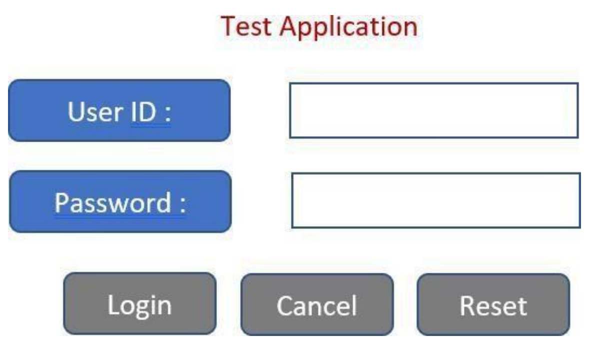
Theory:

Difference between Test Scenario & Test Case

TEST CASE	TEST SCENARIO
Answer “How to Test”	Answer “What to Test”
Test case includes test case name, precondition, test steps, expected result, and the actual result	A test scenario is a high-level documentation which will be associated with multiple test cases
Execute a set of steps to validate the test scenario	Validate functionality of a software application
Hard to maintain due to more in numbers	Easy to maintain due to its high-level design
Derived from Test Scenarios	Derived from Use Cases or requirement documents like BRD, SRS
Low-level actions	High-level actions
Written by Testers	Written by Test Leads, BAs or Testers
Details test cases will require more time and resource	If the test scenario is not detailed, may take time to discuss and understand

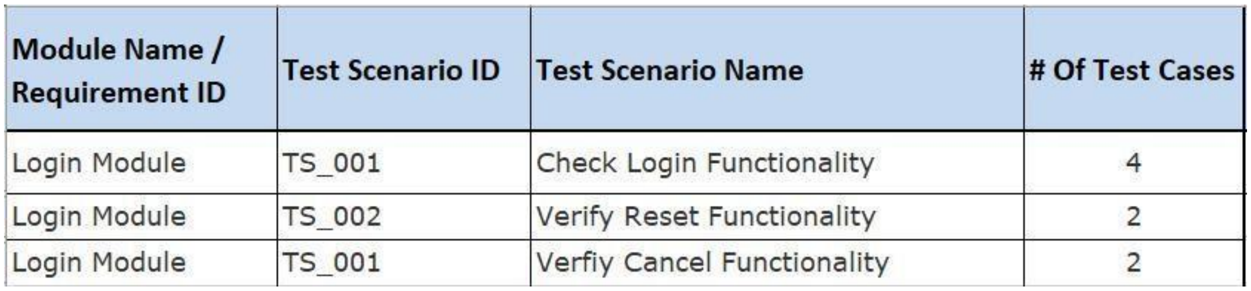
Sample Test Application:

Let’s take an example of a Login page of a Test Application:



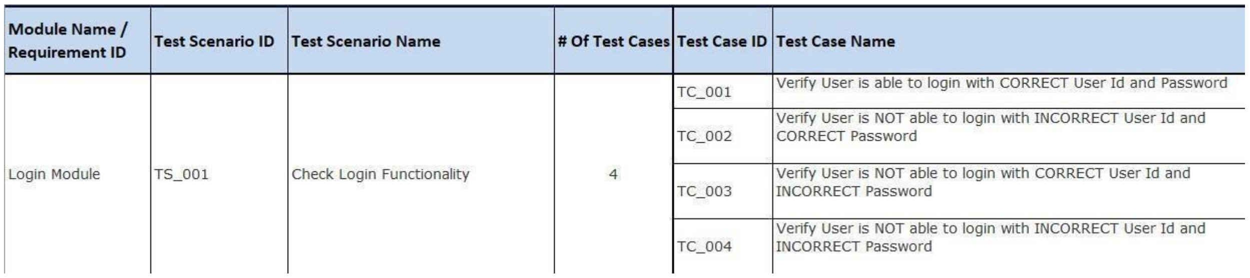
Here we can see **Login**, **Reset** and **Cancel** buttons on the page which will perform the respected functions when clicked.

So below can be the possible Test Scenarios for it:



Module Name / Requirement ID	Test Scenario ID	Test Scenario Name	# Of Test Cases
Login Module	TS_001	Check Login Functionality	4
Login Module	TS_002	Verify Reset Functionality	2
Login Module	TS_001	Verfiy Cancel Functionality	2

Here you can figure out the high-level Test Cases and provide the numbers AND/OR actual Test Case Names.



Module Name / Requirement ID	Test Scenario ID	Test Scenario Name	# Of Test Cases	Test Case ID	Test Case Name
Login Module	TS_001	Check Login Functionality	4	TC_001	Verify User is able to login with CORRECT User Id and Password
				TC_002	Verify User is NOT able to login with INCORRECT User Id and CORRECT Password
				TC_003	Verify User is NOT able to login with CORRECT User Id and INCORRECT Password
				TC_004	Verify User is NOT able to login with INCORRECT User Id and INCORRECT Password

<u>Conclusion:</u>

Test Scenario for Inbox Functionality written successfully.

EXPERIMENT NO 3:

Problem Statement:

Write Test cases in excel sheet for Social Media application or website

Theory:

What is Test Case?

The test case is a set of actions to take to verify if the function of the software application is working as expected.

TCs are designed and executed to find defects in the software application in the early stages. So the defects can be fixed and the quality of software application can be improved. For this reason, preparing the test case as early as possible in the software development process is preferred.

Usually, testers write the test cases (except Unit Tests which are written and executed by developer). In a typical scenario when developers are developing a software application, testers are designing Test Scenarios, Test Cases and creating Test Data.

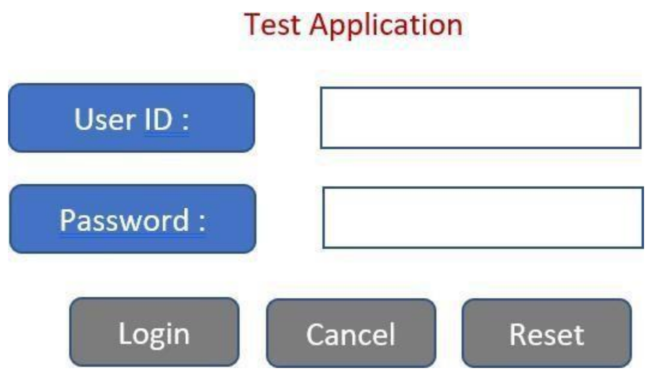
Once Development is completed, testers execute the TCs to test software applications.

Main Components of Test Case:

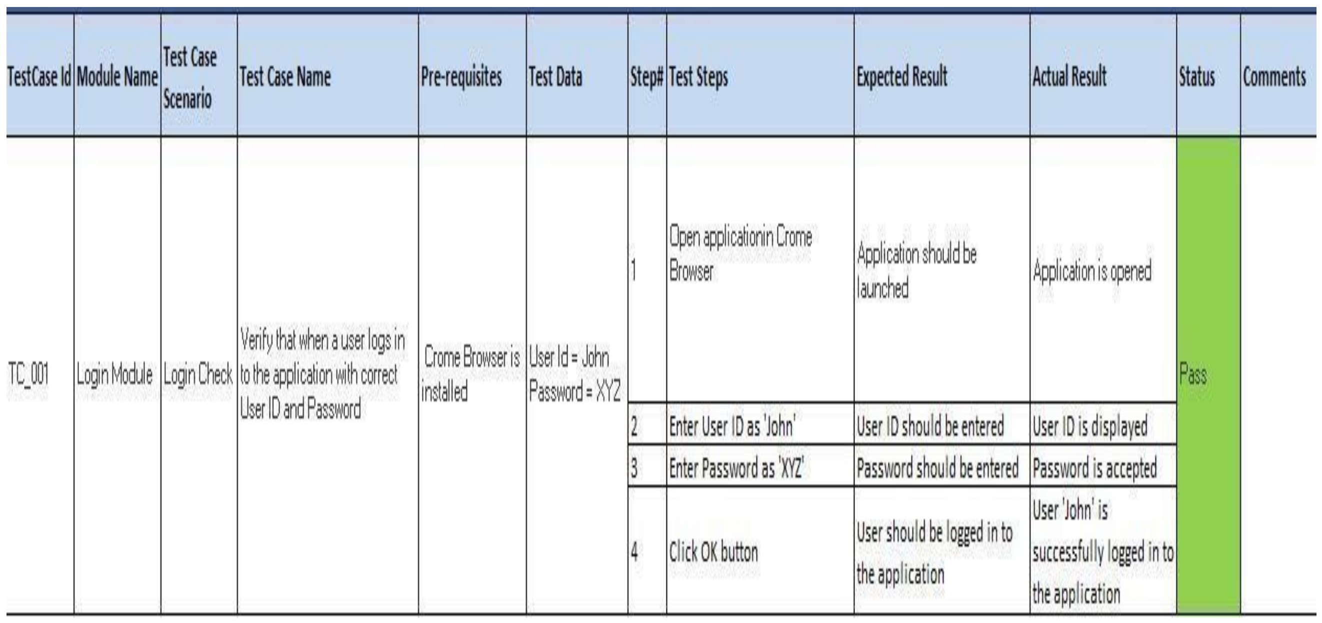
- **Test Case ID:** the unique value to identify the TC or sometimes to determine the number of TCs.
- **Module Name:** Module or Function Name which will be tested by TC.
- **TC Scenario:** A functionality of module to test
- **Test Case Name:** Unique name for TC to know what it verifies.
- **Pre-requisite:** any special setup is needed to be done before running TC
- **Test Data:** the data that needs to be prepared for testing
- **Test Steps:** describe the steps to perform tests
- **Expected Results:** expected results from the steps performed above
- **Actual Result:** usually it will be pass or fail
- **Status:** After execution, if TC is Pass / Failed or On Hold for some reason
- **Comments:** This column is used to note related information when performing test cases.
- **Test Executed By:** Name of the tester who executed the TC

Sample Test Application:

Let’s take a below Test Application



And below is how we can write TCs for Test Scenario of ‘Login Functionality’



TestCase Id	Module Name	Test Case Scenario	Test Case Name	Pre-requisites	Test Data	Step#	Test Steps	Expected Result	Actual Result	Status	Comments
TC_001	Login Module	Login Check	Verify that when a user logs in to the application with correct User ID and Password	Crome Browser is installed	User Id = John Password = XYZ	1	Open applicationin Crome Browser	Application should be launched	Application is opened	Pass	
						2	Enter User ID as 'John'	User ID should be entered	User ID is displayed		
						3	Enter Password as 'XYZ'	Password should be entered	Password is accepted		
						4	Click OK button	User should be logged in to the application	User 'John' is successfully logged in to the application		

Some Sample TCs:

Here are some sample TCs we can think of for Login Functionality:

1. Check the case of only entering the [User ID] field but not the password
2. Check the case of only entering the [Password] field but not the [User ID]
3. Verify login function for correct input [User ID] and [Password]
4. Check the login function in case of incorrect entry 3 times [User ID] and [Password]
5. Check the [User ID]/ [Password] input for more than allowed characters
6. Verify the case of entering special characters into [User ID]
7. Check the input field [User ID] with leading spaces.
8. Check the case of [Password] with leading spaces.
9. Verify the data displayed in the Password textbox must be a star or dot
10. Test if the OK button works as expected
11. Verify if Cancel button works as expected
12. Verify if Reset button works as expected

What is Test Script

In Automated Testing, testers write code/test scripts to automatically execute tests. Testers will use automation tools to build test scenarios and verify that the software works properly. The goal is to complete the test in a shorter time.

So test script is a set of instructions that will be performed on the application using an automation tool to verify if application functionalities work as expected.

Difference between Test Cases & Test Scripts

Test Case is a term used in manual testing and a test script used in an automation Testing.

The only difference between these two concepts is, Test cases are written and executed manually, while Test Scripts are full script/code that QA/testers write to execute using a test automation tool.

Conclusion:

Test Cases for Social Media Application written successfully.

EXPERIMENT NO 4:

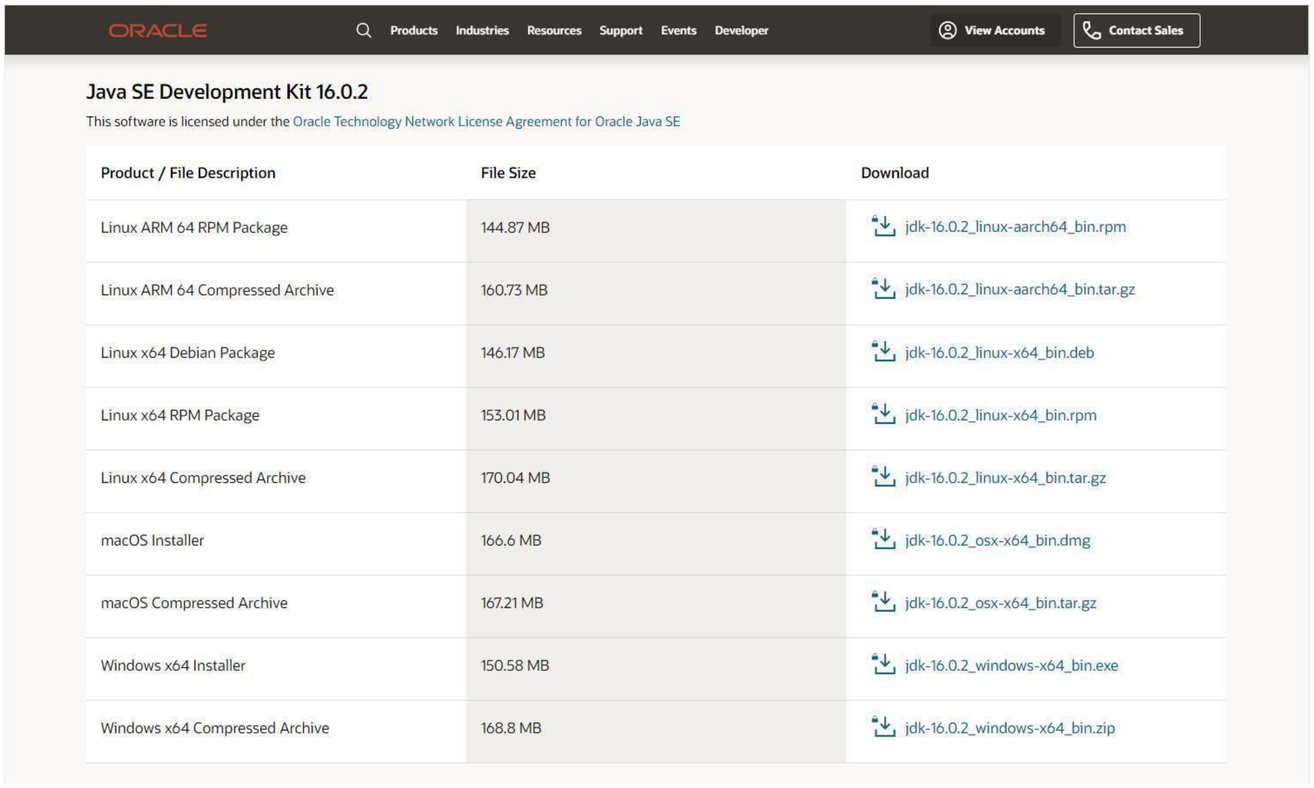
Problem Statement

Installation of Selenium Grid and Selenium Web driver & Java Eclipse (Automation Tools).

Theory

Prerequisites for configuring Selenium in Eclipse

Install java: Download Java SE Development Kit 16.0.2 according to the Windows, Linux, or macOS platform



Java SE Development Kit 16.0.2

This software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE

Product / File Description	File Size	Download
Linux ARM 64 RPM Package	144.87 MB	jdk-16.0.2_linux-aarch64_bin.rpm
Linux ARM 64 Compressed Archive	160.73 MB	jdk-16.0.2_linux-aarch64_bin.tar.gz
Linux x64 Debian Package	146.17 MB	jdk-16.0.2_linux-x64_bin.deb
Linux x64 RPM Package	153.01 MB	jdk-16.0.2_linux-x64_bin.rpm
Linux x64 Compressed Archive	170.04 MB	jdk-16.0.2_linux-x64_bin.tar.gz
macOS Installer	166.6 MB	jdk-16.0.2_osx-x64_bin.dmg
macOS Compressed Archive	167.21 MB	jdk-16.0.2_osx-x64_bin.tar.gz
Windows x64 Installer	150.58 MB	jdk-16.0.2_windows-x64_bin.exe
Windows x64 Compressed Archive	168.8 MB	jdk-16.0.2_windows-x64_bin.zip

Run the JDK Installer by double clicking on the file name in the download location and following the instruction wizard. Alternatively, silently install JDK by entering the following command: **jdk.exe /s**

Install Eclipse IDE

Install Selenium Grid

Download and install Selenium to be set up in Eclipse.

Install Selenium Driver

For Cross Browser Testing, download the relevant Browser Driver – Chrome Driver (for Chrome), Gecko Driver (for Firefox), Safari Driver(for Safari), and Internet Explorer Driver and MS Edge Driver (IE and Edge respectively). Place these Browser Driver files in a directory that is part of the environment PATH. This will allow a command-line call to the programs to execute them irrespective of the working directory.

How to configure Selenium in Eclipse

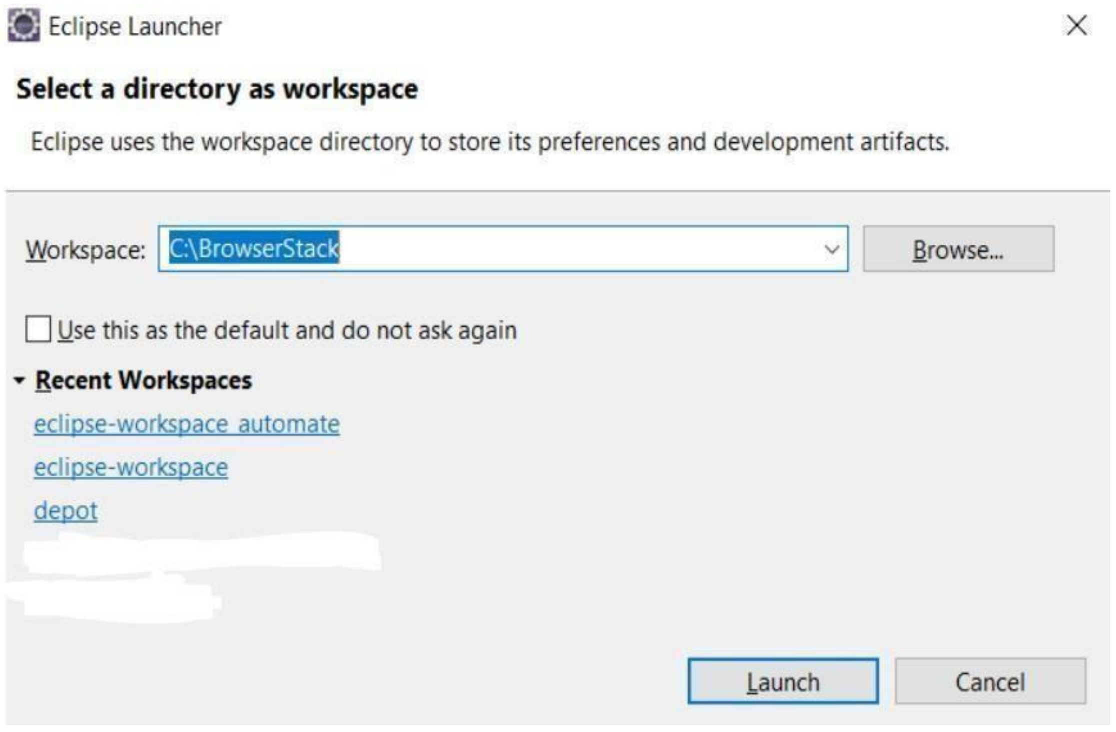
Here are the steps to configure Selenium Web driver with Eclipse:

Step 1: Launch Eclipse

To launch Eclipse double click on the **eclipse.exe** file in the download location.

Step 2: Create Workspace in Eclipse

This workspace named "**C:\BrowserStack**" is like any other folder, which will store all the test scripts. Launch the BrowserStack workspace.



Eclipse Launcher

Select a directory as workspace

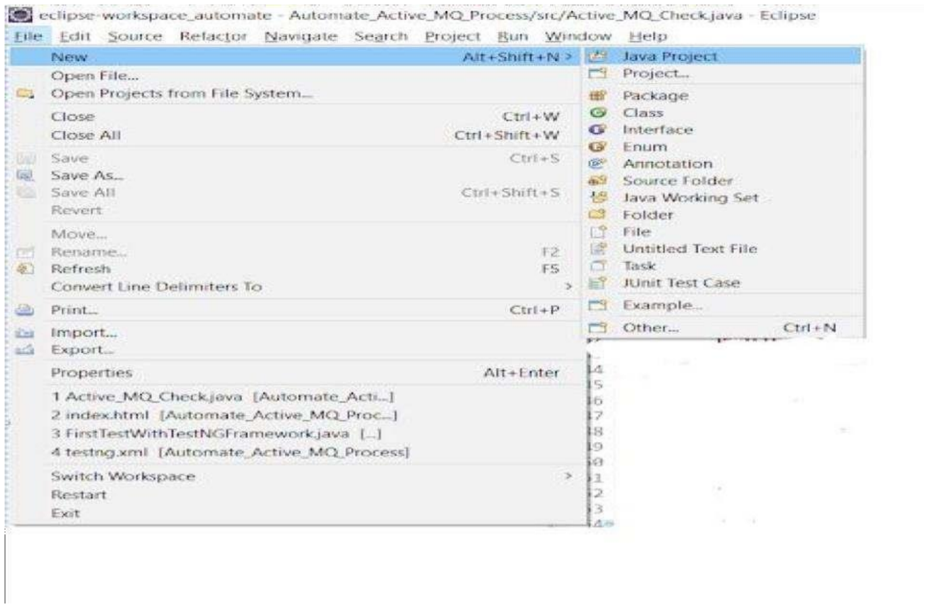
Eclipse uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\BrowserStack

Browse...
Use this as the default and do not ask again
Recent Workspaces
eclipse-workspace automate
eclipse-workspace
depot
Launch
Cancel

Step 3: Create New Java Project in the BrowserStack Workspace

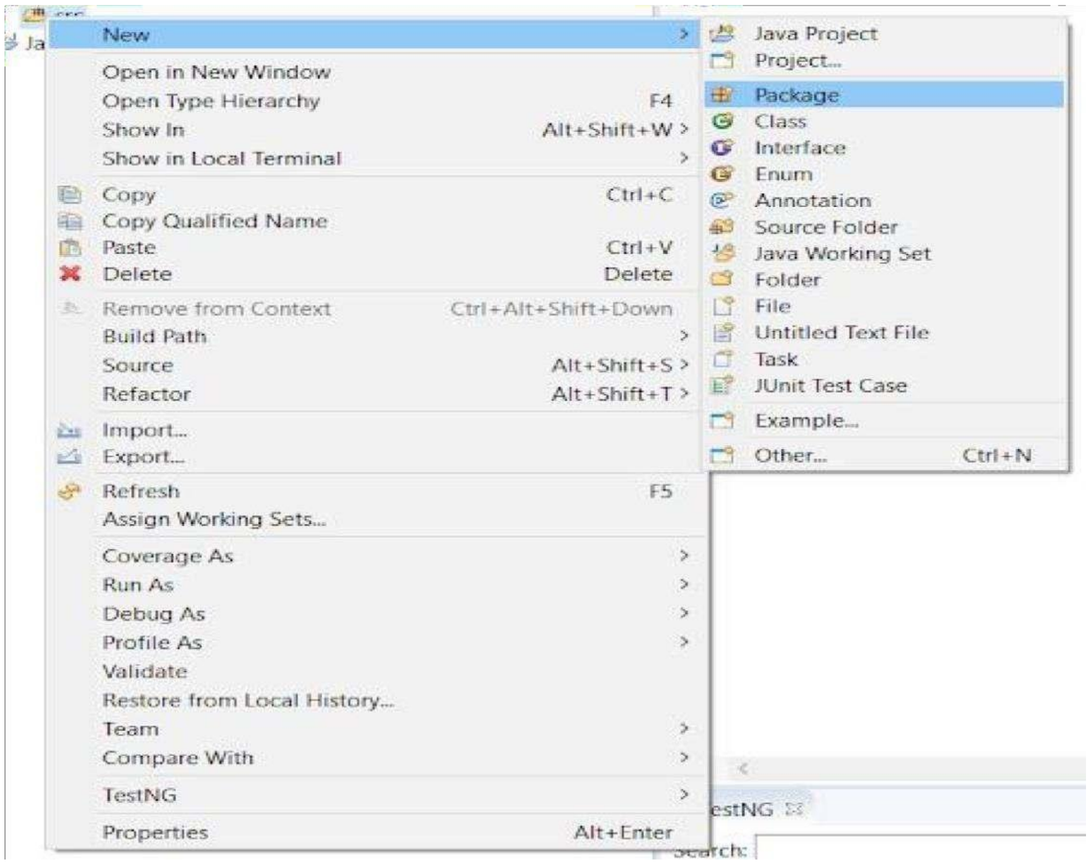
Create a new Java Project by clicking on **File > New > Java Project** and name it.


eclipse-workspace_automate - Automate_Active_MQ_Process/src/Active_MQ_Check.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
New Alt+Shift+N >
Open File...
Open Projects from File System...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To >
Print... Ctrl+P
Import...
Export...
Properties Alt+Enter
1 Active_MQ_Check.java [Automate_Acti...]
2 index.html [Automate_Active_MQ_Proc...]
3 FirstTestWithTestNGFramework.java [...]
4 testng.xml [Automate_Active_MQ_Process]
Switch Workspace >
Restart
Exit
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
Task
JUnit Test Case
Example...
Other... Ctrl+N

Creating a new Java Project

Step 4: Create Package and Class under the Java Project

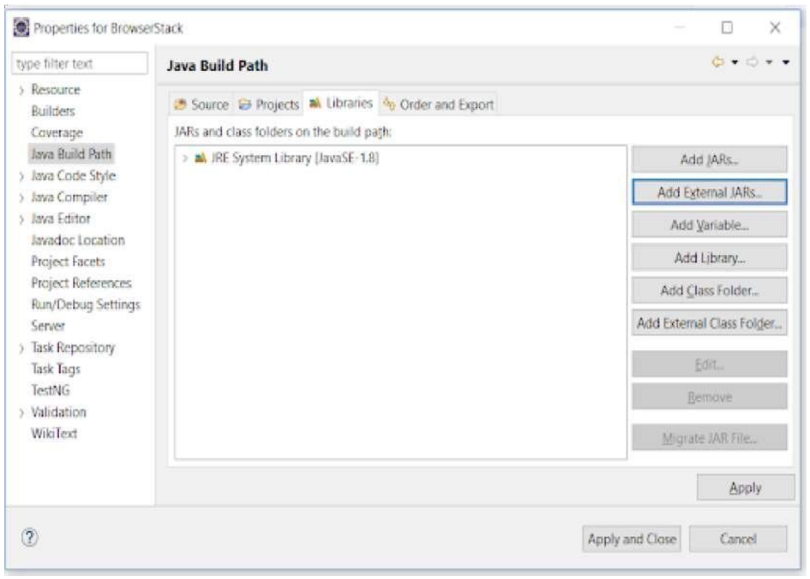
By clicking on the **src folder** (which is the source folder), create a new package and name it (BrowserStack). Then right-click on the package name and create a class.



Creating Class in the BrowserStack Package

Step 5: Add Selenium JARs to the Java Project in Eclipse

To add the Selenium Jars to the BrowserStack Java right click on the BrowserStack Project folder and select the **Properties** option. In the properties window, click on the **Java Build Path** and **Add External JARs**. Browse and add the downloaded Selenium JARs i.e. Client Combined JAR and all the JARs under the Libs folder, then click **Apply and Close**.



Adding Selenium JARs in the BrowserStack Project

This configures Selenium with Eclipse, making it ready to execute the first test script.

<u>Conclusion</u>

Installation of automation tools is done successfully.

EXPERIMENT NO 5:

Problem Statement

Prepare Software requirement specification for any project or problem statement

Theory:

Software Requirement Specification (SRS) Format:

as name suggests, is complete specification and description of requirements of software that needs to be fulfilled for successful development of software system. These requirements can be functional as well as non-functional depending upon type of requirement. The interaction between different customers and contractor is done because its necessary to fully understand needs of customers. Depending upon information gathered after interaction, SRS is developed which describes requirements of software that may include changes and modifications that is needed to be done to increase quality of product and to satisfy customer's demand.

1. **Introduction:**
   - Purpose of this Document – At first, main aim of why this document is necessary and what's purpose of document is explained and described.
   - Scope of this document – In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.
   - Overview – In this, description of product is explained. It's simply summary or overall review of product.
2. **General description :** In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.
3. **Functional Requirements:** In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order.
4. **Interface Requirements :** In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully Described and explained. Examples can be shared memory, data streams, etc.

5. **Performance Requirements:** In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc.
6. **Design Constraints:** In this, constraints which simply means limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc
7. **Non-Functional Attributes:** In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.
8. **Preliminary Schedule and Budget:** In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.
9. **Appendices :** In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained

Sample Software Requirements Specification points:

1) Introduction:
2) Overall description:
    2.1 Product Perspective
    2.2 Product Functions
    2.3 User Classes and Characteristics
    2.4 Operating Environment
    2.5 Design and Implementation Constraints
    2.6 User Documentation
    2.7 Assumptions and Dependencies
3) External Interface Requirements:
    3.1 User Interfaces
    3.2 Hardware Interfaces
    3.3 Software Interfaces
    3.4 Communication Interfaces
4) Functional Requirement Specifications (FRS):
    4.1 System Features
    4.2 Functional Requirements

5) Non Functional Requirements:
   - 5.1 Usability Requirements
   - 5.2 Performance Requirements
   - 5.3 Compatibility Requirements
6) Other Requirements
7) Glossary

Conclusion: Prepared the software requirement specification document successfully.