NCS 652 Software Engineering Lab

For any given case/ problem statement do the following;

1. Prepare a SRS document in line with the IEEE recommended standards.
2. Draw the use case diagram and specify the role of each of the actors. Also state the precondition, post condition and function of each use case.
3. Draw the activity diagram.
4. Identify the classes. Classify them as weak and strong classes and draw the class diagram.
5. Draw the sequence diagram for any two scenarios.
6. Draw the collaboration diagram.
7. Draw the state chart diagram.
8. Draw the component diagram.
9. Perform forward engineering in java.(Model to code conversion)
10. Perform reverse engineering in java.(Code to Model conversion)
11. Draw the deployment diagram.

- Allotment of project to each students group. (not specified in UPTU syllabus)

Course Objective

1. Understand the activities associated with typical software development processes and to solve problem in a team environment.

Course Outcomes

1. Students understand basic software engineering approaches for software development.
2. Students familiar with procedures for eliciting and documenting problem requirements
3. Students understand the role of software architecture in designing solutions

S.No.	List of Experiments	CO
1	Prepare a SRS document in line with the IEEE recommended standards.	Co-2
2	Draw the use case diagram and specify the role of each of the actors. Also state the precondition, post condition and function of each use case.	Co -1
3	Draw the activity diagram.	Co-3
4	Identify the classes. Classify them as weak and strong classes and draw the class diagram.	Co-3
5	Draw the sequence diagram for any two scenarios.	Co-3
6	Draw the collaboration diagram.	Co-3
7	Draw the state chart diagram.	Co-3
8	Draw the component diagram.	Co-3
9	Perform forward engineering in java.(Model to code conversion)	Co-1
10	Perform reverse engineering in java.(Code to Model conversion)	Co-1
11	Draw the deployment diagram.	Co -1, Co-3

Mapping of Course Objective and Program Outcomes:

Course Outcomes	Program Outcomes											
	A.	B.	C.	D.	E.	F.	G.	H.	I.	J.	K.	L.
1	2	2										
2		2	2						1			
3			2		1							

CONTENTS

Application Tools

- StarUML

Schedule For Different Documents Submission

The objective of the Lab work is to make the student conversant with the StarUML tool for modeling of Business Applications. During the semester, we will allocate project to each group which will be completely solved in the lab classes with the assistance of the Instructor. Students are requested to implement the project case study during the lab classes.

Simultaneously, students are expected to solve an additional case study given in the lab manual and submit its solution as per the following schedule.

Date of Submission	Deliverables
	SRS Document
	Document of Use Case Diagram
	Activity Diagram Document
	Class Diagram Document
	Sequence Diagram Document
	Collaboration Diagram Document
	State Chart Diagram Document
	Component Diagram Document
	Deployment Diagram Document

Guidelines for the students:

1. The students will work individually under the guidance of the faculty.
2. They should have at least one meeting in a week with the faculty for the assessment of their work progress.
3. The students should meet the faculty with the formal documents and proper presentations prepared of their work till date.
4. Every student should always be prepared for a viva-voice or presentation of his or her work.
5. Regularity and sincerity will be taken into consideration while evaluating the students.

Lab Assignments

EXERCISE 1:

Prepare a SRS document in line with the IEEE recommended standards.

EXERCISE 2:

Draw the use case diagram and specify the role of each of the actors. Also state the precondition, post condition and function of each use case.

Objective:

To understand the problem statement and analyze it.

To understand the meaning of use cases and actors

EXERCISE 3:

Draw the activity diagram for the problem statement.

EXERCISE 4:

Identify the classes. Classify them as weak and strong classes and draw the class diagram

Objective:

Identify the entity classes.

To understand the relationships between these classes

Identify dependency, hierarchy and associativity between these classes.

EXERCISE 5:

Draw sequence diagram for each Use Case identified in the problem statement.

Objective:

To understand how different objects are interacting with each other.

To understand which event occurs first, and what happens next.

To understand the operations of the classes.

EXERCISE 6:

Draw the collaboration diagram for the problem statement.

Objective:

To understand the difference b/w sequence diagram and collaboration diagram

To learn what notations and symbols are used in a collaboration diagram.

EXERCISE 7:

Draw the State Chart Diagram for the problem statement.

EXERCISE 8:

Draw the Component Diagram for the problem statement.

EXERCISE 9:

Perform forward engineering in java. (Model to code conversion)

EXERCISE 10:

Perform reverse engineering in java. (Code to Model conversion)

EXERCISE 11:

Draw the Deployment Diagram for the problem statement.

Lab Exercise No.1

Develop requirements specification for a given problem.

Description: A software requirements specification describes the essential behaviour of a software product from a user's point of view.

The purpose of the SRS is to:

a. **Establish the basis for agreement between the customers and the suppliers on what the software product is to do.** The complete description of the functions to be performed by the software specified in the SRS will assist the potential user to determine if the software specified meets their needs or how the software must be modified to meet their needs
b. **Provide a basis for developing the software design.** The SRS is the most important document of reference in developing a design
c. **Reduce the development effort.** The preparation of the SRS forces the various concerned groups in the customer's organisation to thoroughly consider all of the requirements before design work begins. A complete and correct SRS reduces effort waisted on redesign, recoding and retesting. Careful review of the requirements in the SRS can reveal omissions, misunderstandings and inconsistencies early in the development cycle when these problems are easier to correct
d. **Provide a basis for estimating costs and schedules.** The description of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates
e. **Provide a baseline for** validation and verification. Organisations can develop their test documentation much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured
f. **Facilitate transfer.** The SRS makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organisation and suppliers find it easier to transfer it to new customers
g. **Serve as a basis for enhancement.** Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued product evaluation.

IEEE Standard SRS Template

1. Introduction
- 1.1. Purpose
- 1.2. Scope
- 1.3. Definitions, acronyms & abbreviations
- 1.4. References
- 1.5. Overview

2. Overall description
- 2.1. Product perspective
  - 2.1.1. System interfaces
  - 2.1.2. User interfaces
  - 2.1.3. Hardware interfaces
  - 2.1.4. Software interfaces
  - 2.1.5. Communications interfaces
  - 2.1.6. Memory constraints
  - 2.1.7. Operations
  - 2.1.8. Site adaptation requirements
- 2.2. Product functions
- 2.3. User characteristics
- 2.4. Constraints
- 2.5. Assumptions and dependencies
- 2.6. Apportioning of requirements

3. Specific Requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communication interfaces
- 3.2 Specific requirements
  - 3.2.1 Sequence diagrams
  - 3.2.2 Classes for classification of specific requirements
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
  - 3.5.1 Reliability
  - 3.5.2 Availability
  - 3.5.3 Security
  - 3.5.4 Maintainability
- 3.6 Other requirements

4. Supporting information
- 4.1 Table of contents and index
- 4.2 Appendixes

Note: History of versions of this document with author/contributor info may be included before the main sections of the document.

Lab Exercise No.2

Develop UML Use case model for a problem

Description: Use case diagrams are usually referred to as behaviour 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.

Importance of Use Case Diagrams
As mentioned before use case diagram are used to gather a usage requirement of a system. Depending on your requirement you can use that data in different ways. Below are few ways to use them.

- **To identify functions and how roles interact with them** – The primary purpose of use case diagrams.
- **For a high level view of the system** – Especially useful when presenting to managers or stakeholders. You can highlight the roles that interact with the system and the functionality provided by the system without going deep into inner workings of the system.
- **To identify internal and external factors** – This might sound simple but in large complex projects a system can be identified as an external role in another use case.

Use Case Diagram objects

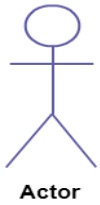
Use case diagrams consist of 4 objects.

- Actor
- Use case
- System
- Package

The objects are further explained below.

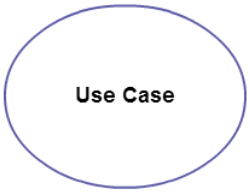
Actor

Actor in a use case diagram is **any entity that performs a role** in one given system. This could be a person, organization or an external system and usually drawn like skeleton shown below.



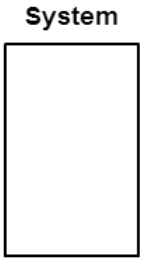
Use Case

A use case **represents a function or an action within the system**. Its drawn as an oval and named with the function.



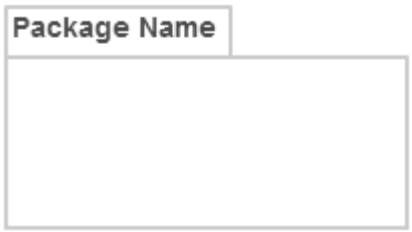
System

System is used to **define the scope of the use case** and drawn as a rectangle. This an optional element but useful when your visualizing large systems. For example you can create all the use cases and then use the system object to define the scope covered by your project. Or you can even use it to show the different areas covered in different releases.



Package

Package is another optional element that is extremely useful in complex diagrams. Similar to **class diagrams**, packages are **used to group together use cases**. They are drawn like the image shown below.

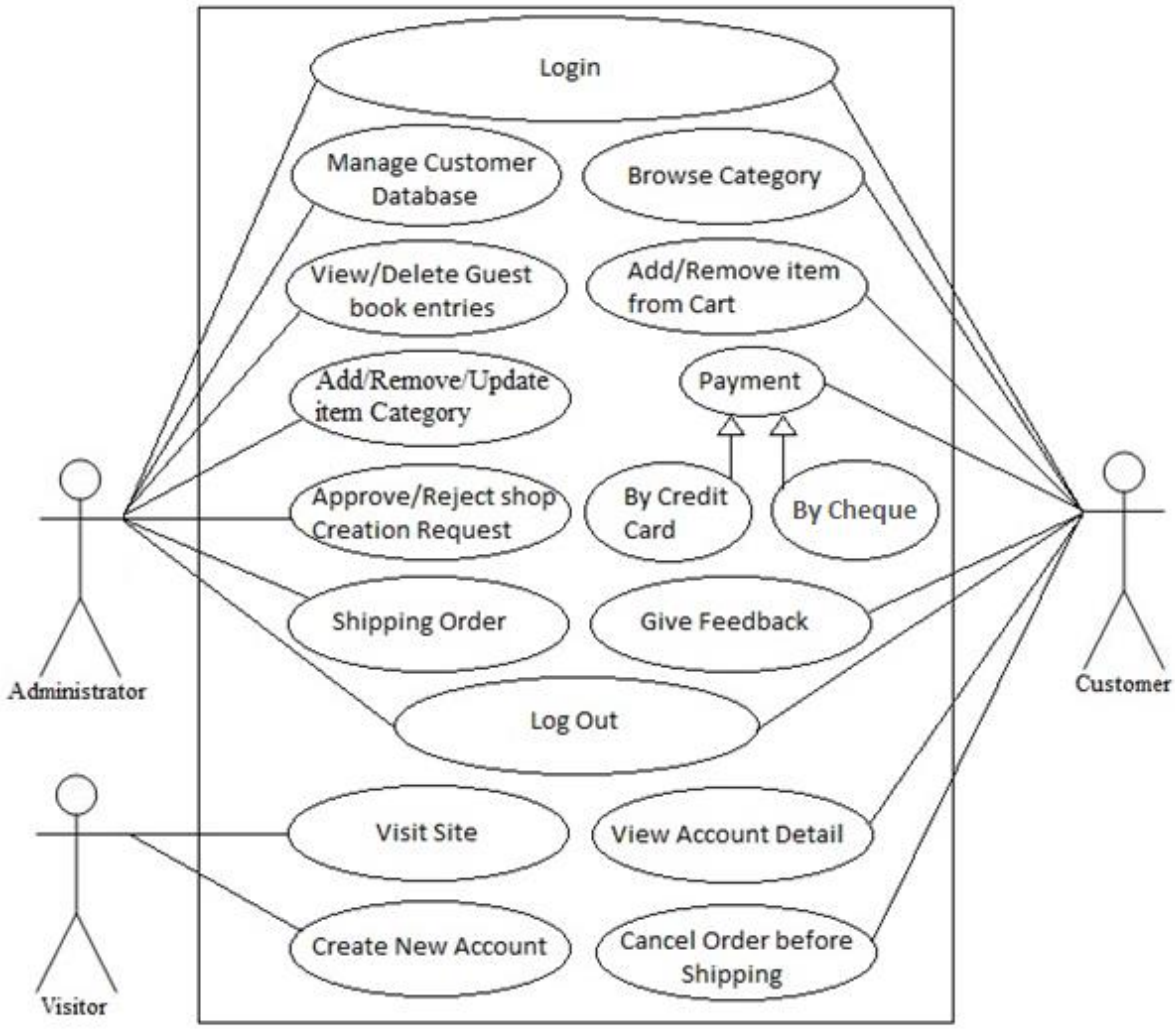


Relationships in Use Case Diagrams

There are five types of relationships in a use case diagram. They are

- Association between an actor and a use case
- Generalization of an actor
- Extend relationship between two use cases
- Include relationship between two use cases
- Generalization of a use case

Example: UML Use Case Model for Online Shopping System



Use case diagram for Customer, Visitor and Administrator.

Lab Exercise No.3

Develop activity diagram for the problem

Description: An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart and a data flow diagram. Activity diagrams are often used in business process modeling. They can also describe the steps in a use case diagram. Activities modeled can be sequential and concurrent. In both cases an activity diagram will have a beginning and an end.

Basic Activity Diagram Notations and Symbols

Initial State or Start Point

A small filled circle followed by an arrow represents the initial action state or the start point for any activity diagram. For activity diagram using swimlanes, make sure the start point is placed in the top left corner of the first column.

Activity or Action State

An action state represents the non-interruptible action of objects. You can draw an action state in StarUML using a rectangle with rounded corners.

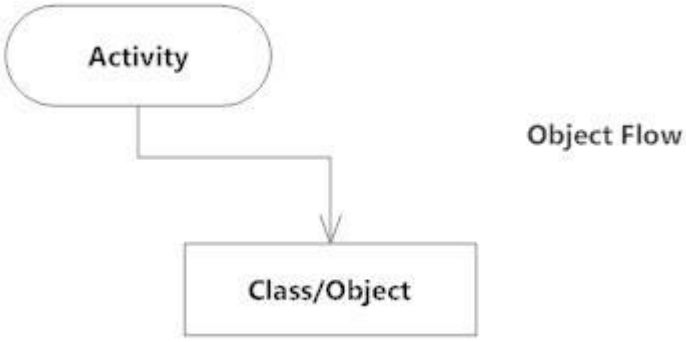
Action Flow

Action flows, also called edges and paths, illustrate the transitions from one action state to another. They are usually drawn with an arrowed line.

Object Flow

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the

object. An object flow arrow from an object to an action indicates that the action state uses the object.



Decisions and Branching

A diamond represents a decision with alternate paths. When an activity requires a decision prior to moving on to the next activity, add a diamond between the two activities. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."

Guards

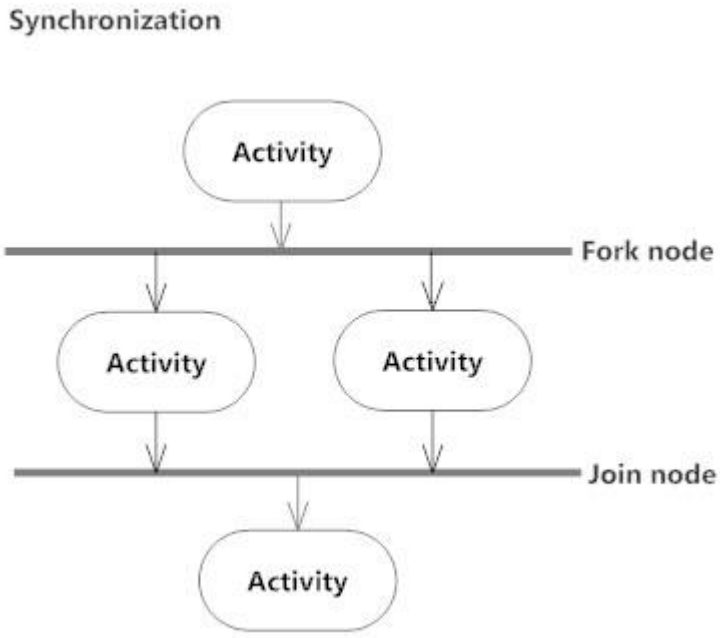
In UML, guards are a statement written next to a decision diamond that must be true before moving next to the next activity. These are not essential, but are useful when a specific answer, such as "Yes, three labels are printed," is needed before moving forward.

Synchronization

A fork node is used to split a single incoming flow into multiple concurrent flows. It is represented as a straight, slightly thicker line in an activity diagram.

A join node joins multiple concurrent flows back into a single outgoing flow.

A fork and join mode used together are often referred to as synchronization.

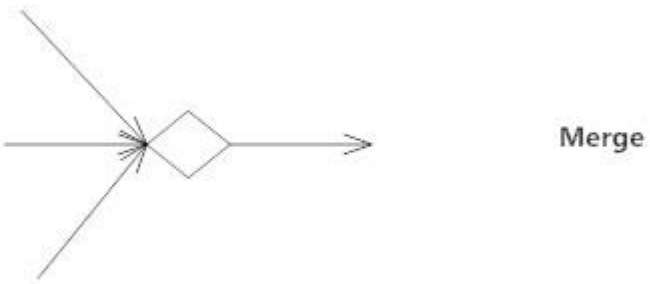


Time Event

This refers to an event that stops the flow for a time; an hourglass depicts it.

Merge Event

A merge event brings together multiple flows that are not concurrent.



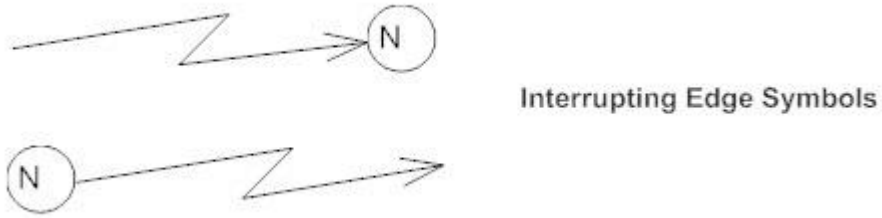
Sent and Received Signals

Signals represent how activities can be modified from outside the system. They usually appear in pairs of sent and received signals, because the state can't change

until a response is received, much like synchronous messages in asequence diagram. For example, an authorization of payment is needed before an order can be completed.

Interrupting Edge

An event, such as a cancellation, that interrupts the flow denoted with a lightning bolt.

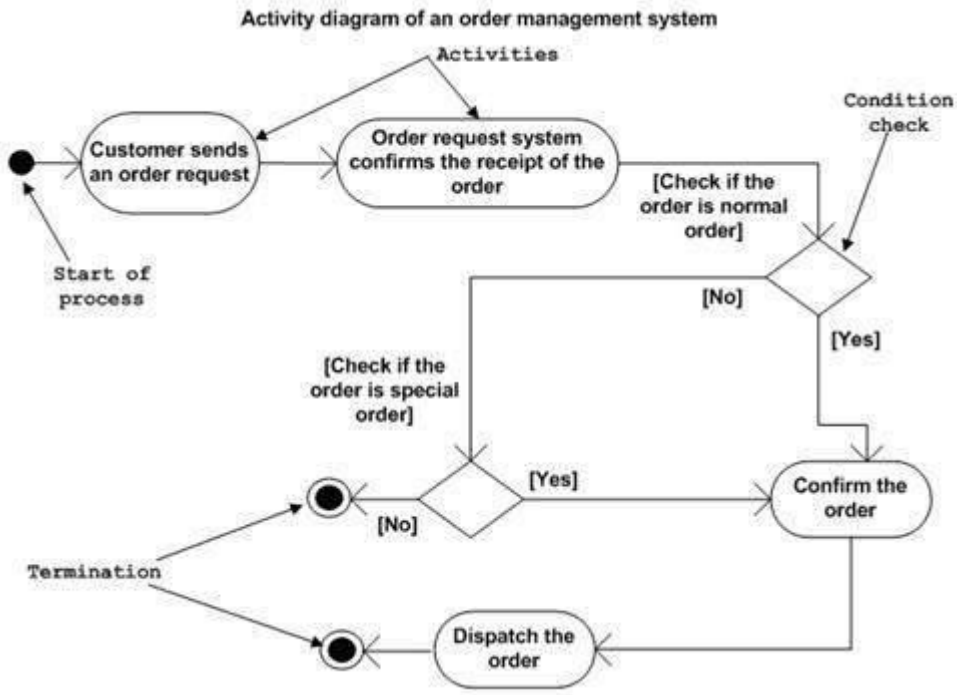


The following is an example of an activity diagram for order management system. In the diagram four activities are identified which are associated with conditions. One important point should be clearly understood that an activity diagram cannot be exactly matched with the code. The activity diagram is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

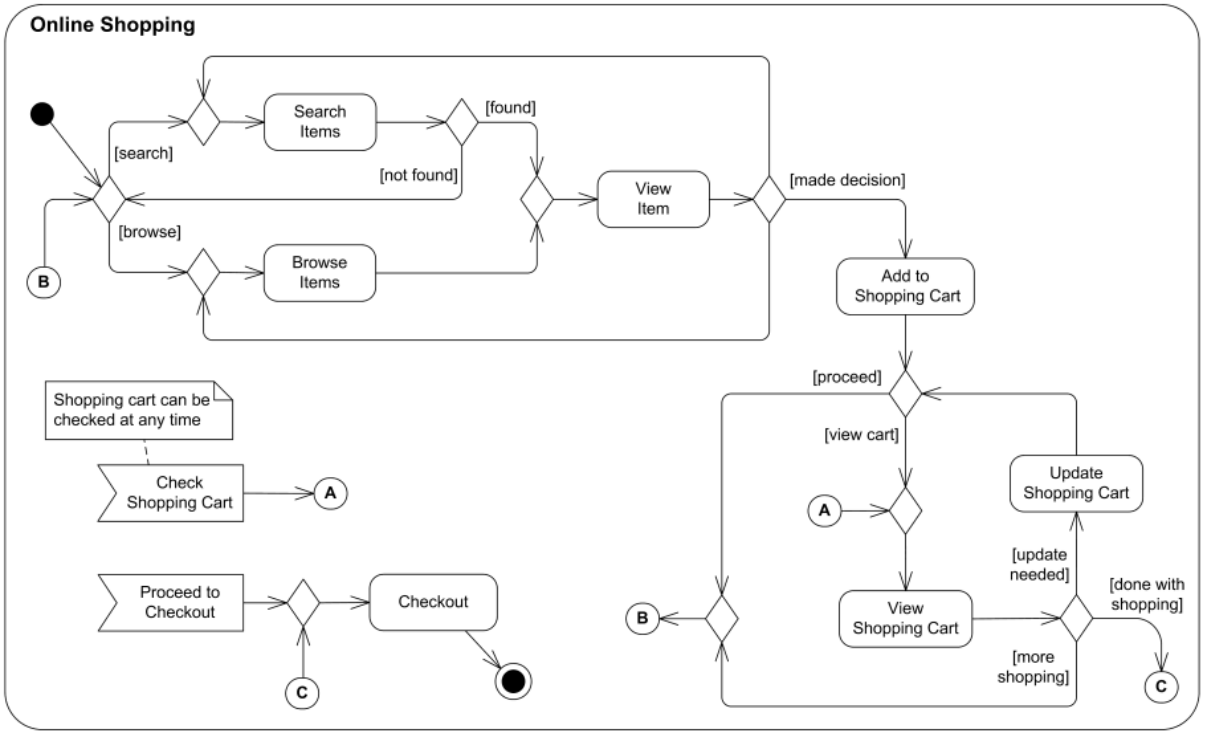
The following diagram is drawn with the four main activities:

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

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



Activity Diagram for Online Shopping System:



Lab Exercise No.4

Develop class diagram for the problem

Description: Class diagrams model the static structure of a system. They show relationships between classes, objects, attributes, and operations.

Basic Class Diagram Symbols and Notations

Classes

Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.

Illustrate classes with rectangles divided into compartments. Place the name of the class in the first partition (centered, bolded, and capitalized), list the attributes in the second partition (left-aligned, not bolded, and lowercase), and write operations into the third.

Class

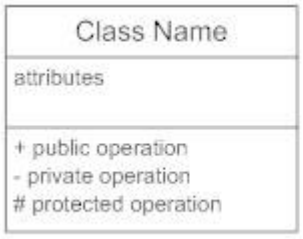
Active Classes

Active classes initiate and control the flow of activity, while passive classes store data and serve other classes. Illustrate active classes with a thicker border.

Active class

Visibility

Use visibility markers to signify who can access the information contained within a class. Private visibility, denoted with a - sign, hides information from anything outside the class partition. Public visibility, denoted with a + sign, allows all other classes to view the marked information. Protected visibility, denoted with a # sign, allows child classes to access information they inherited from a parent class.

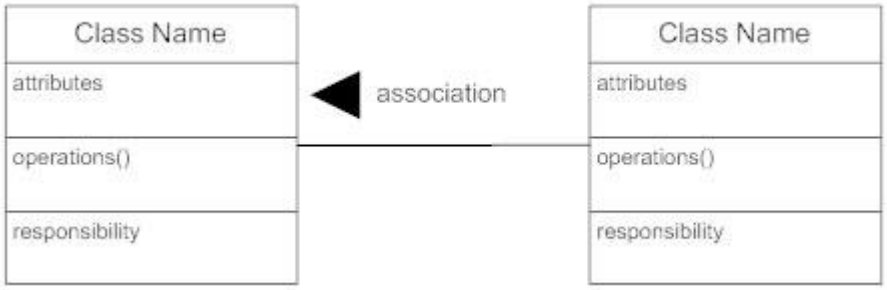


Visibility

Marker	Visibility
+	public
-	private
#	protected
~	package

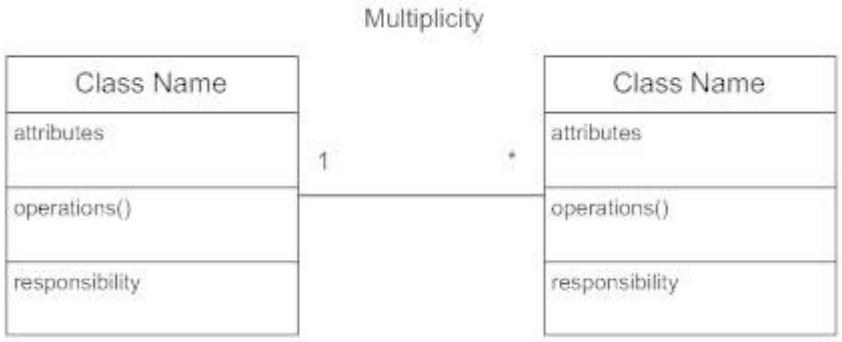
Associations

Associations represent static relationships between classes. Place association names above, on, or below the association line. Use a filled arrow to indicate the direction of the relationship. Place roles near the end of an association. Roles represent the way the two classes see each other.



Multiplicity (Cardinality)

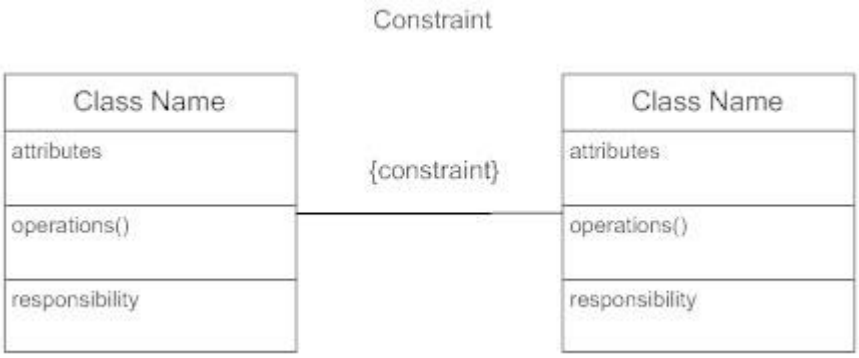
Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class. For example, one company will have one or more employees, but each employee works for just one company.



Indicator	Meaning
0..1	Zero or one
1	One only
0..*	0 or more
1..* *	1 or more
n	Only n (where n > 1)
0..n	Zero to n (where n >1)
1..n	One to n (where n > 1)

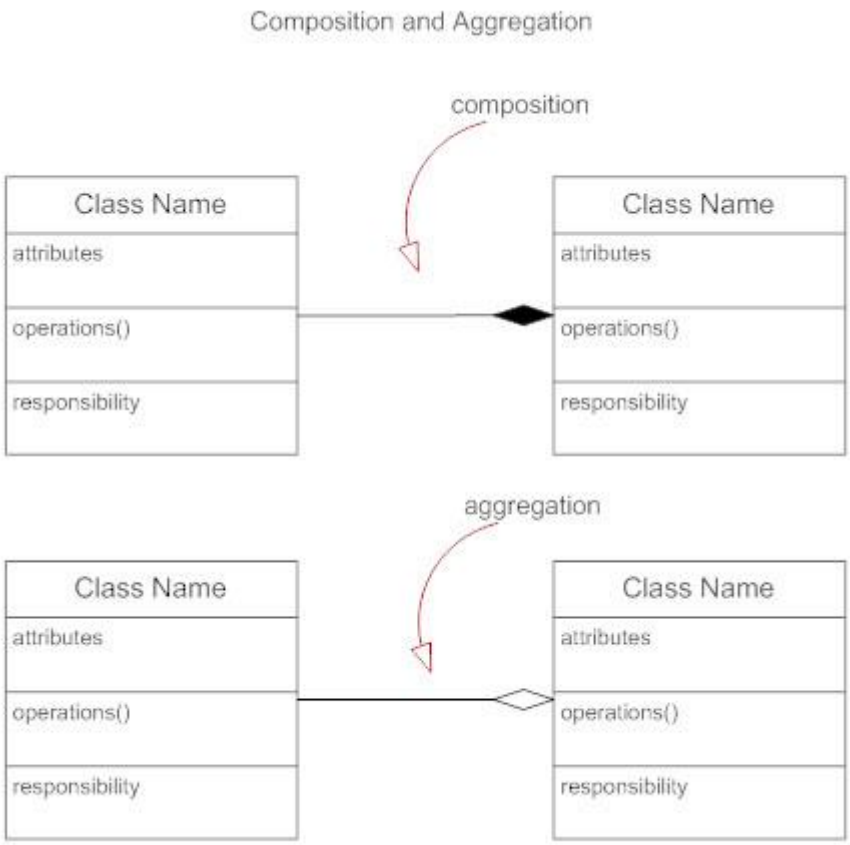
Constraint

Place constraints inside curly braces {}.



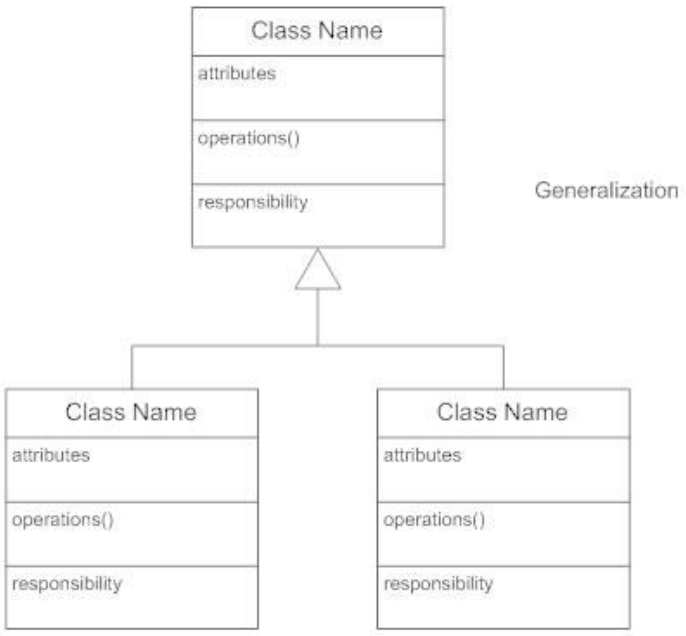
Composition and Aggregation

Composition is a special type of aggregation that denotes a strong ownership between Class A, the whole, and Class B, its part. Illustrate composition with a filled diamond. Use a hollow diamond to represent a simple aggregation relationship, in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other. The diamond ends in both composition and aggregation relationships point toward the "whole" class (i.e., the aggregation).

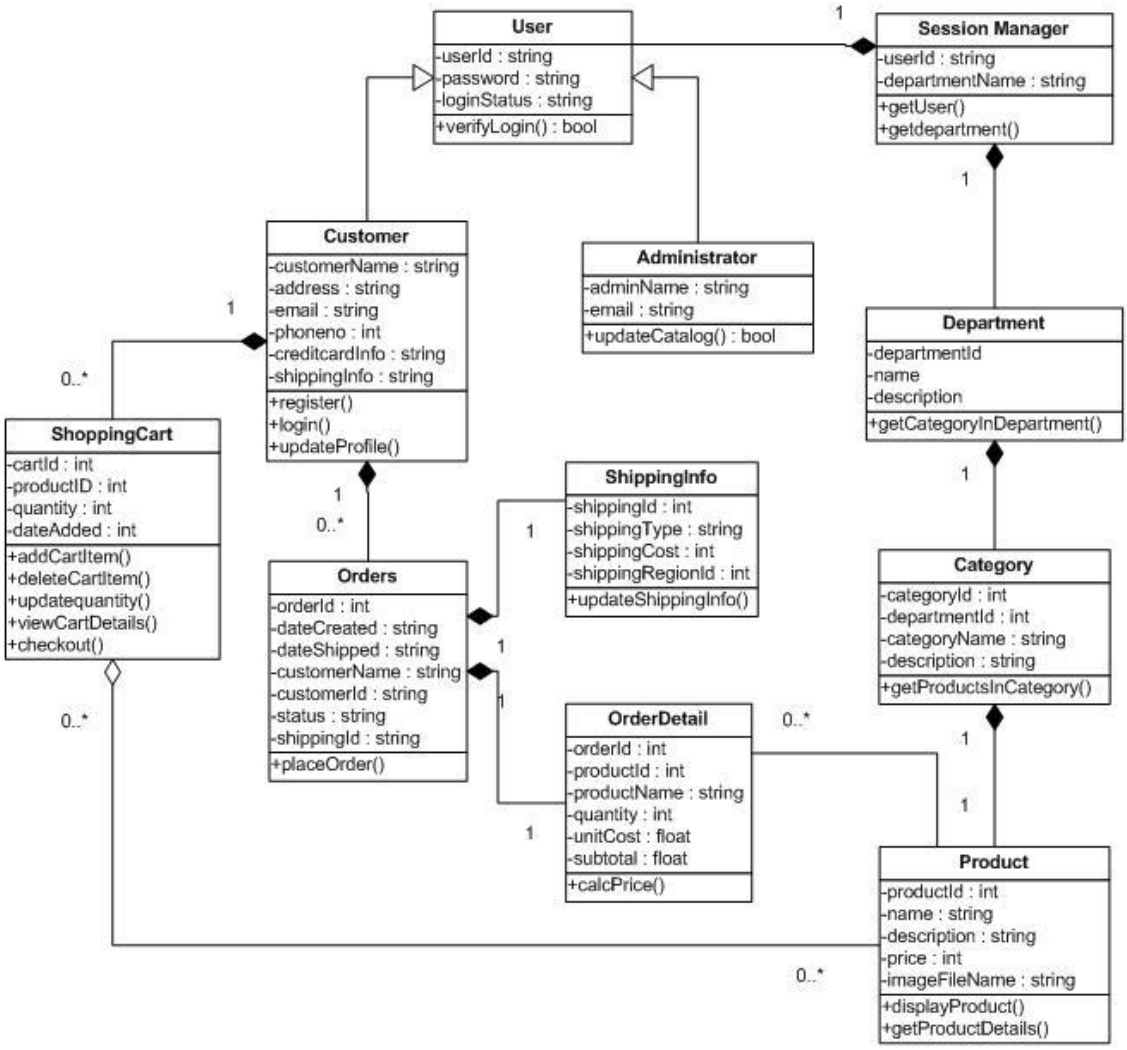


Generalization

Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another. For example, Honda is a type of car. So the class Honda would have a generalization relationship with the class car.



Example: Class Diagram for Online Shopping System



Lab Exercise No.5

Draw the sequence diagram for any two scenarios.

Description- To show the dynamic behaviour of the system interaction diagram is used. These diagrams describe some type of interactions among the different elements in the model.

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

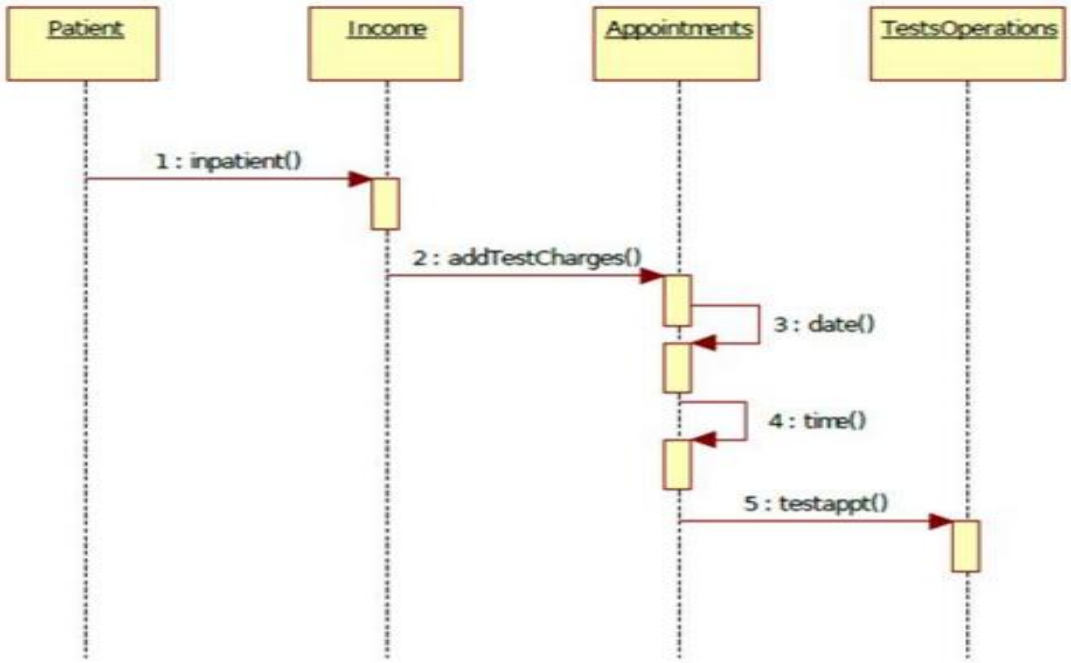
Sequence diagram emphasizes on time sequence of messages and *collaboration diagram* emphasizes on the structural organization of the objects that send and receive messages.

Sequence diagrams in UML shows how object interact with each other and the order those interactions occur. It's important to note that they show the interactions for a particular scenario. The processes are represented vertically and interactions are show as arrows.

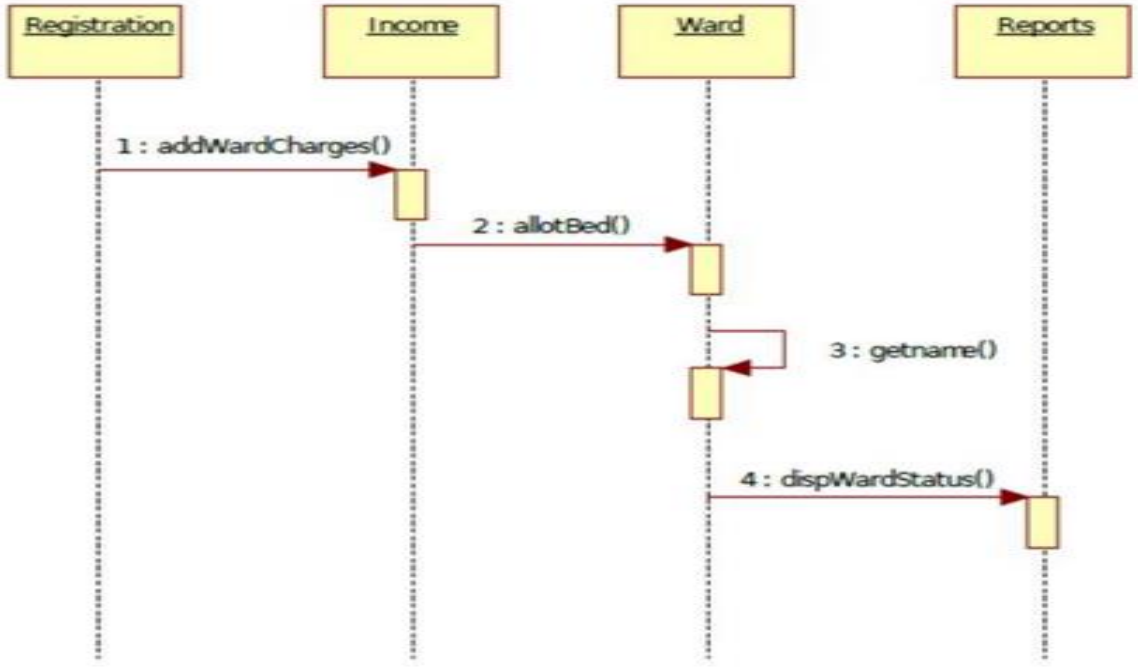
The following things are to be identified clearly before drawing the interaction diagram:

- Objects taking part in the interaction.
- Message flows among the objects.
- The sequence in which the messages are flowing.
- Object organization.

Sequence Diagram:- Test Appointments



Sequence Diagram:- Bed Allotment



Lab Exercise No.6

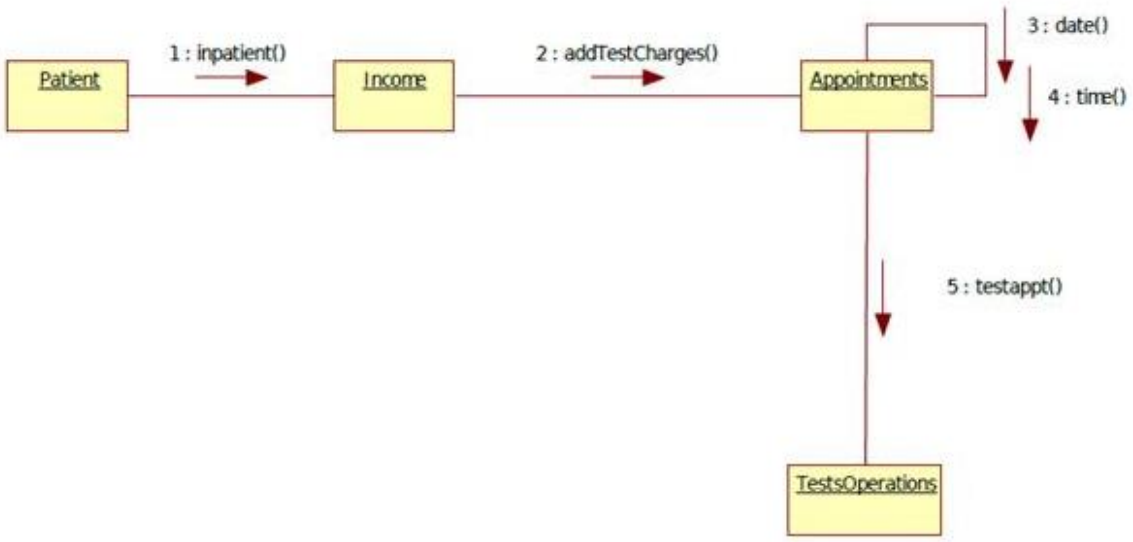
Draw the collaboration diagram

Description: The second interaction diagram is collaboration diagram. In collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another.

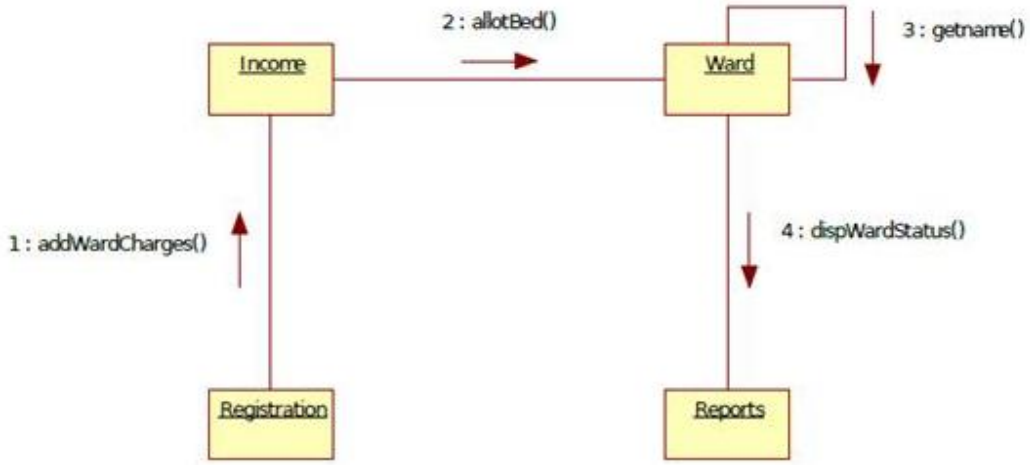
The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization where as the collaboration diagram shows the object organization.

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

Collaboration Diagram:- Test Appointments



Collaboration Diagram:- Bed Allotments



Lab Exercise No.7

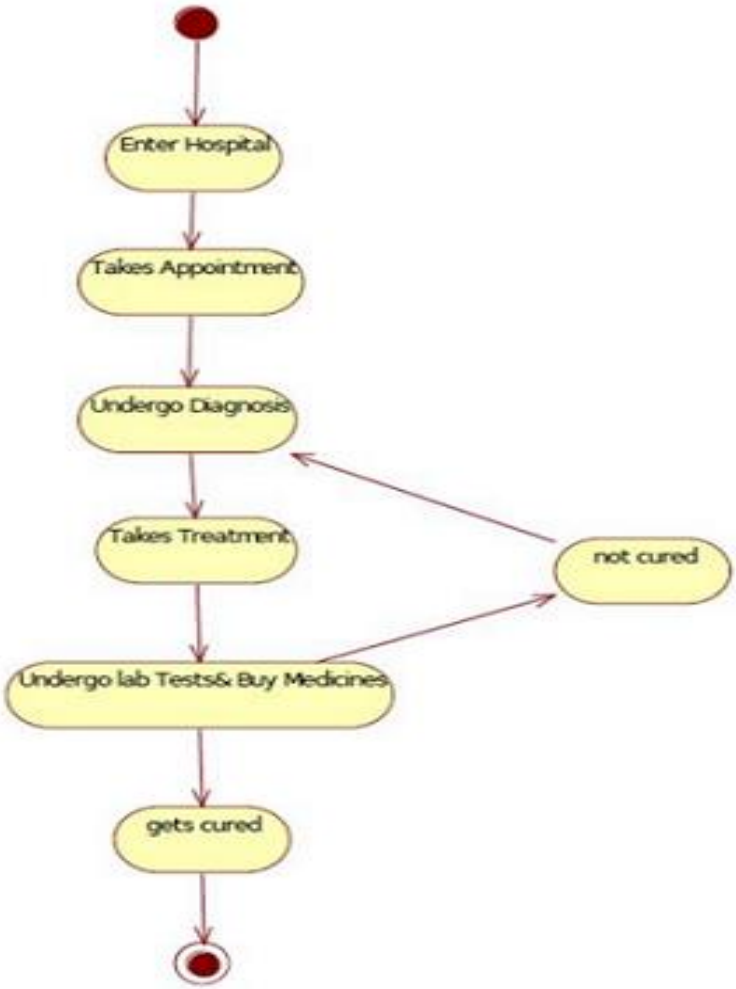
Draw the state chart diagram

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

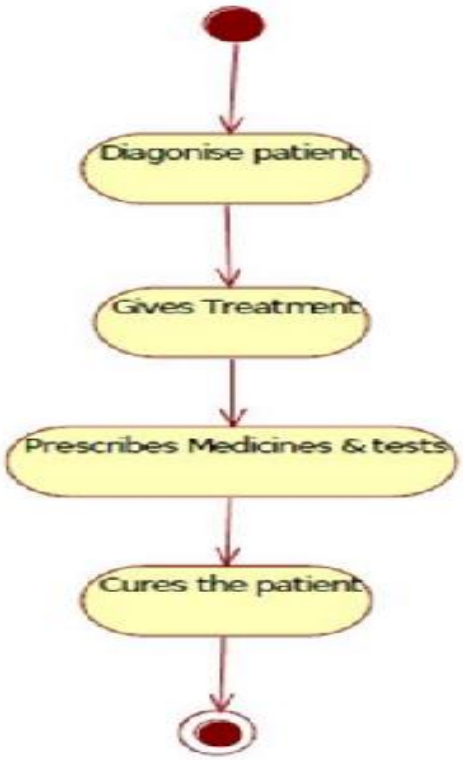
Following are the main purposes of using Statechart diagrams:

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

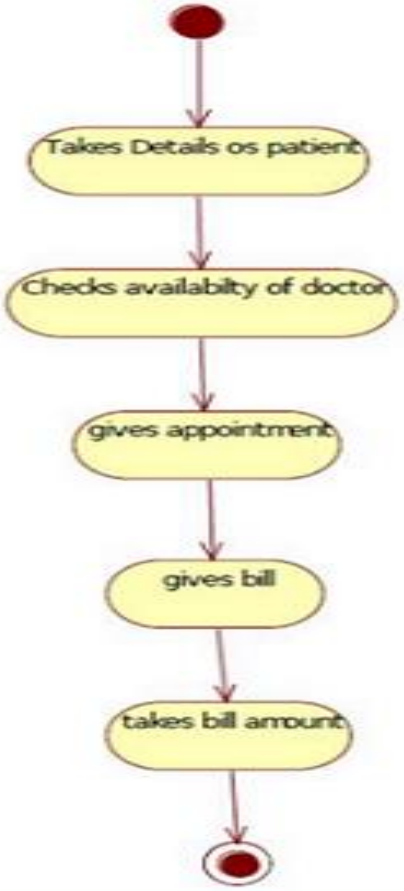
State Chart Diagram :- Patient



State Chart Diagram :- Doctor



State Chart Diagram:- Receptionist



Lab Exercise No.8

Draw the component diagram

Description: Component diagram describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.

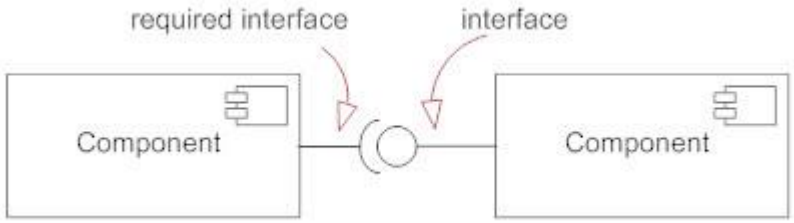
Basic Component Diagram Symbols and Notations

Component

A component is a logical unit block of the system, a slightly higher abstraction than classes. It is represented as a rectangle with a smaller rectangle in the upper right corner with tabs or the word <component> written above the name of the component to help distinguish it from a class.

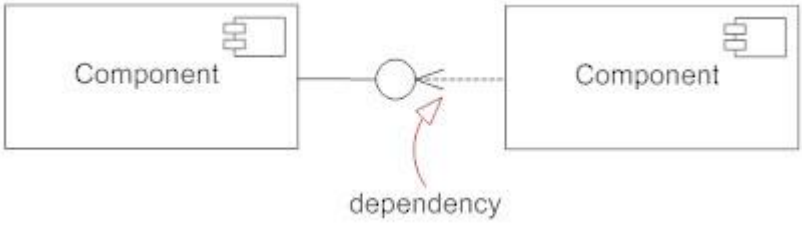
Interface

An interface (small circle or semi-circle on a stick) describes a group of operations used (required) or created (provided) by components. A full circle represents an interface created or provided by the component. A semi-circle represents a required interface, like a person's input.



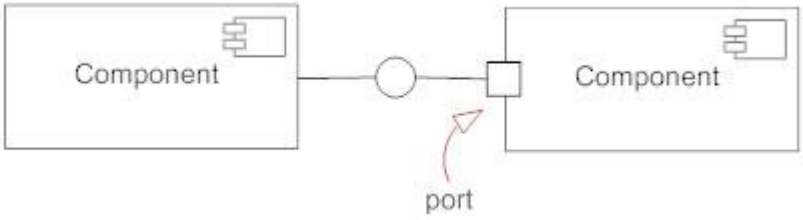
Dependencies

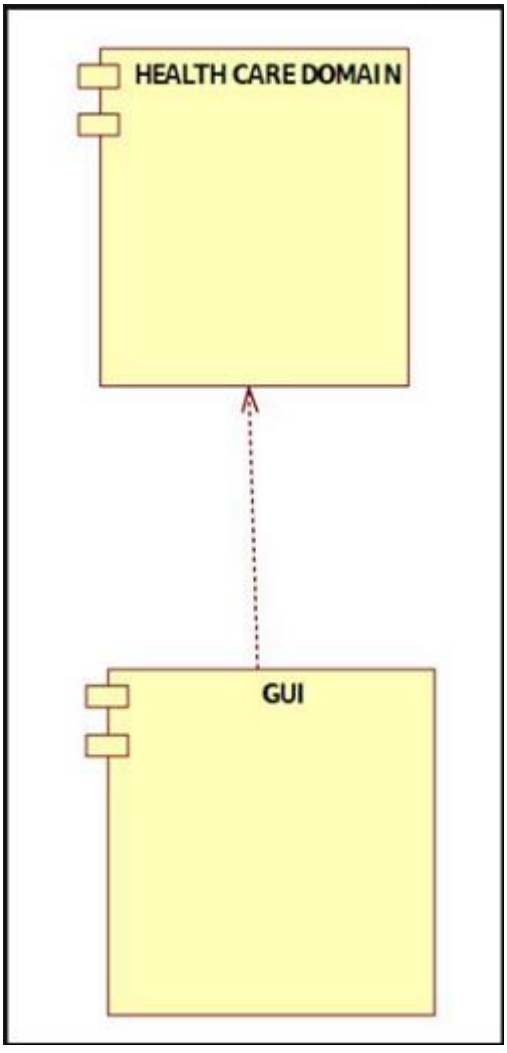
Draw dependencies among components using dashed arrows.



Port

Ports are represented using a square along the edge of the system or a component. A port is often used to help expose required and provided interfaces of a component.





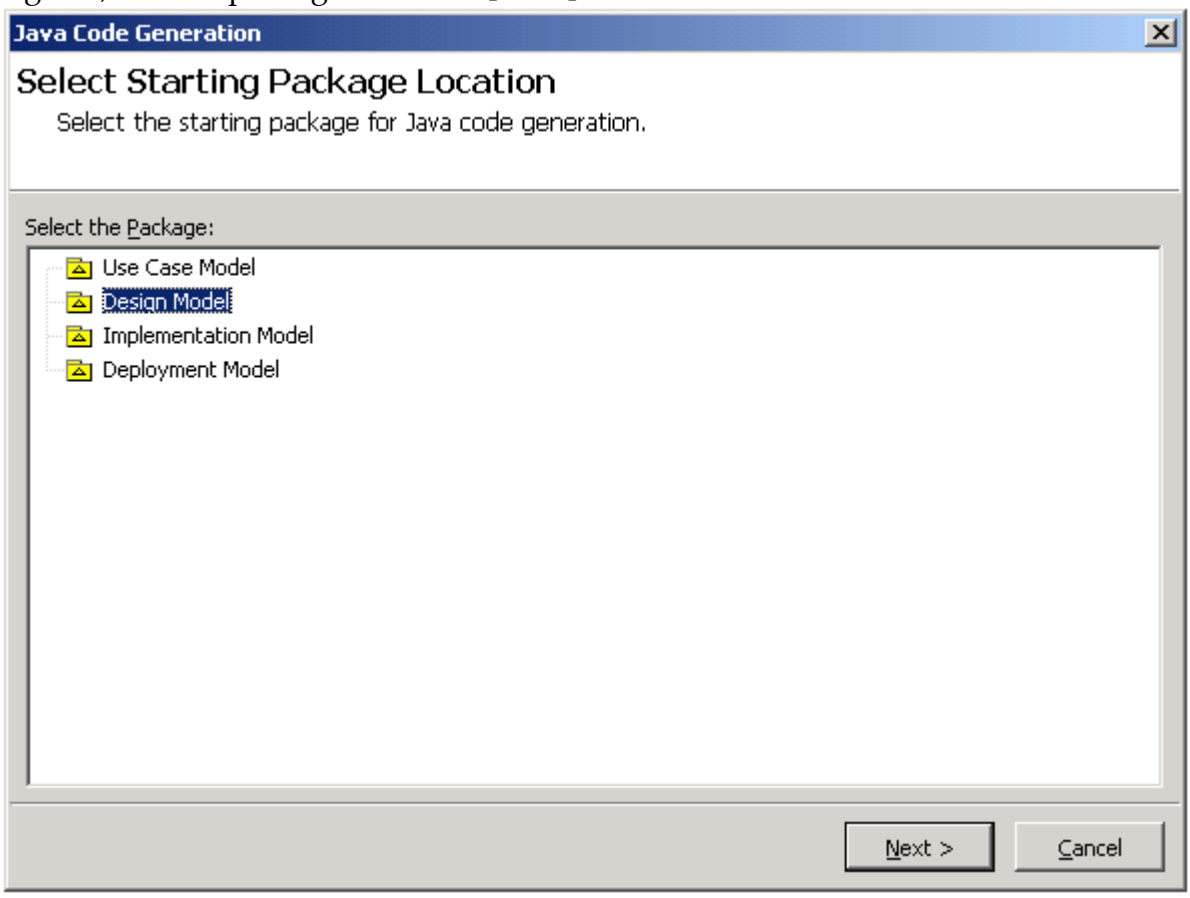
Lab Exercise No.9

Perform forward engineering in java. (Model to code conversion)

Description: This is where you have a diagram, or set of diagrams (perhaps put together by a business analyst) and you turn them into some executable code.

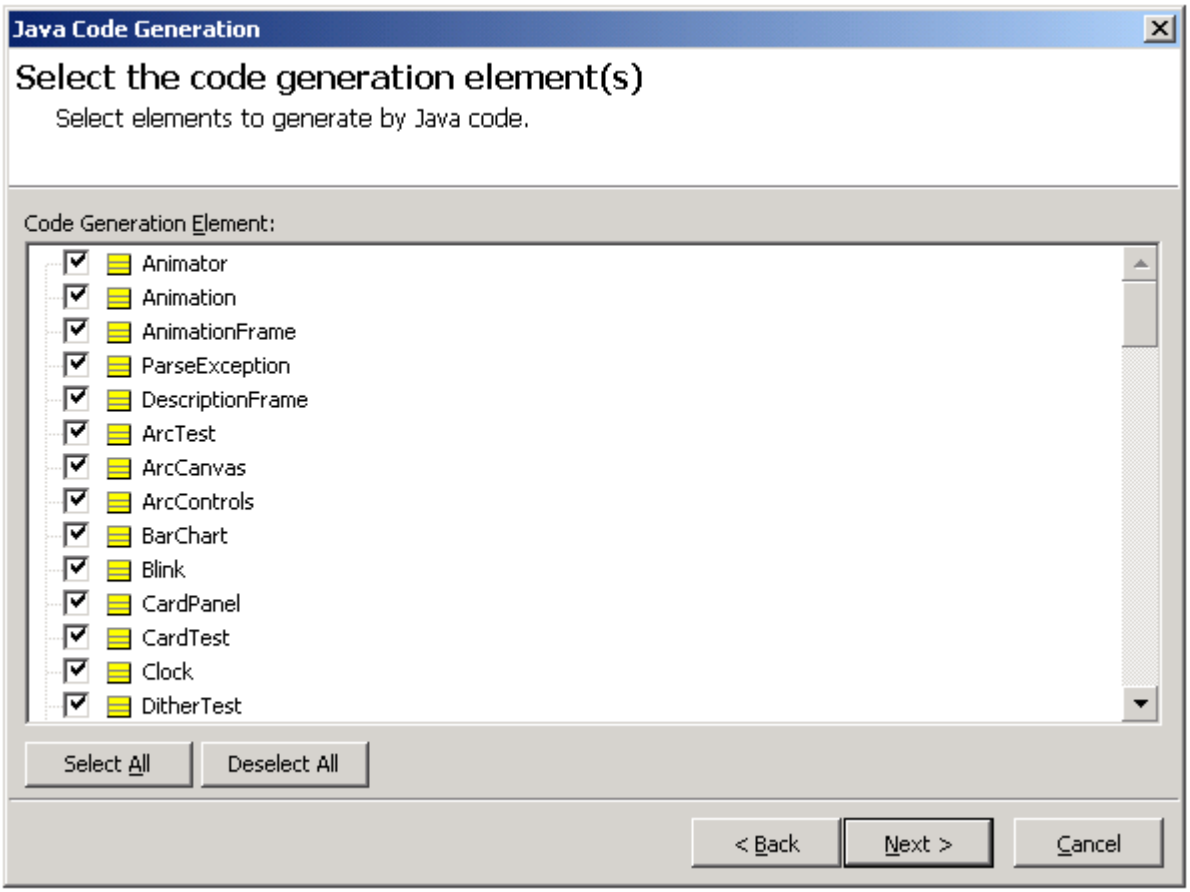
Procedure for Code Generation:

1. In StarUML(tm), select the **[Tools] -> [Java] -> [Generate Code...]** menu.
2. At the **[Select Starting Package Location]** page in the **[Java Code Generation]** dialog box, select a package and click **[Next]**.

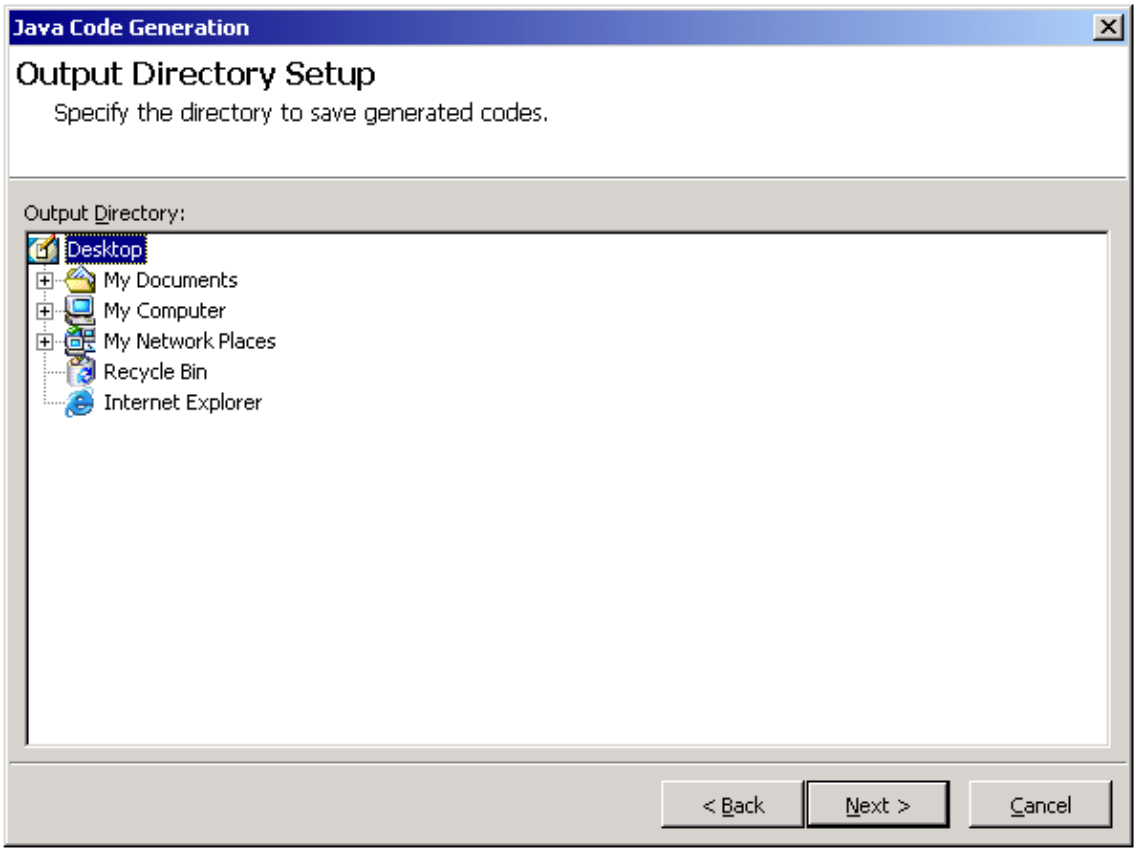


3. At the **[Select the code generation element(s)]** page, select the elements and click

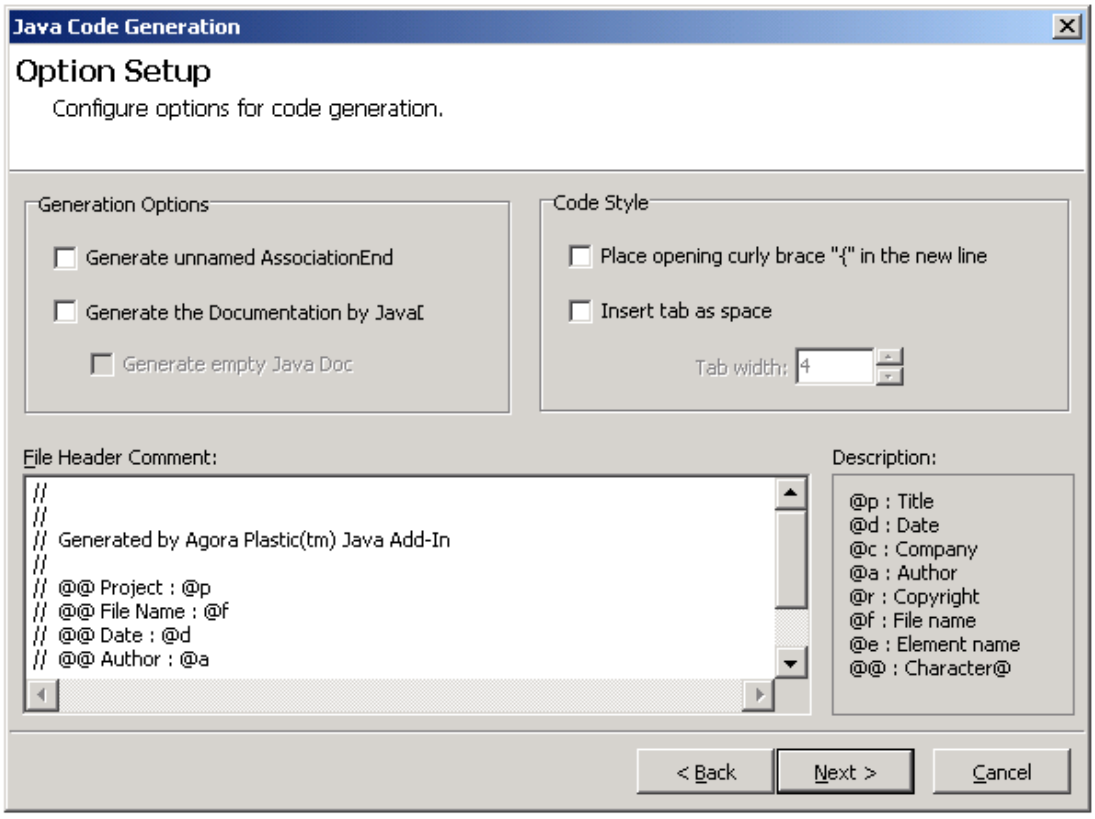
[Next].



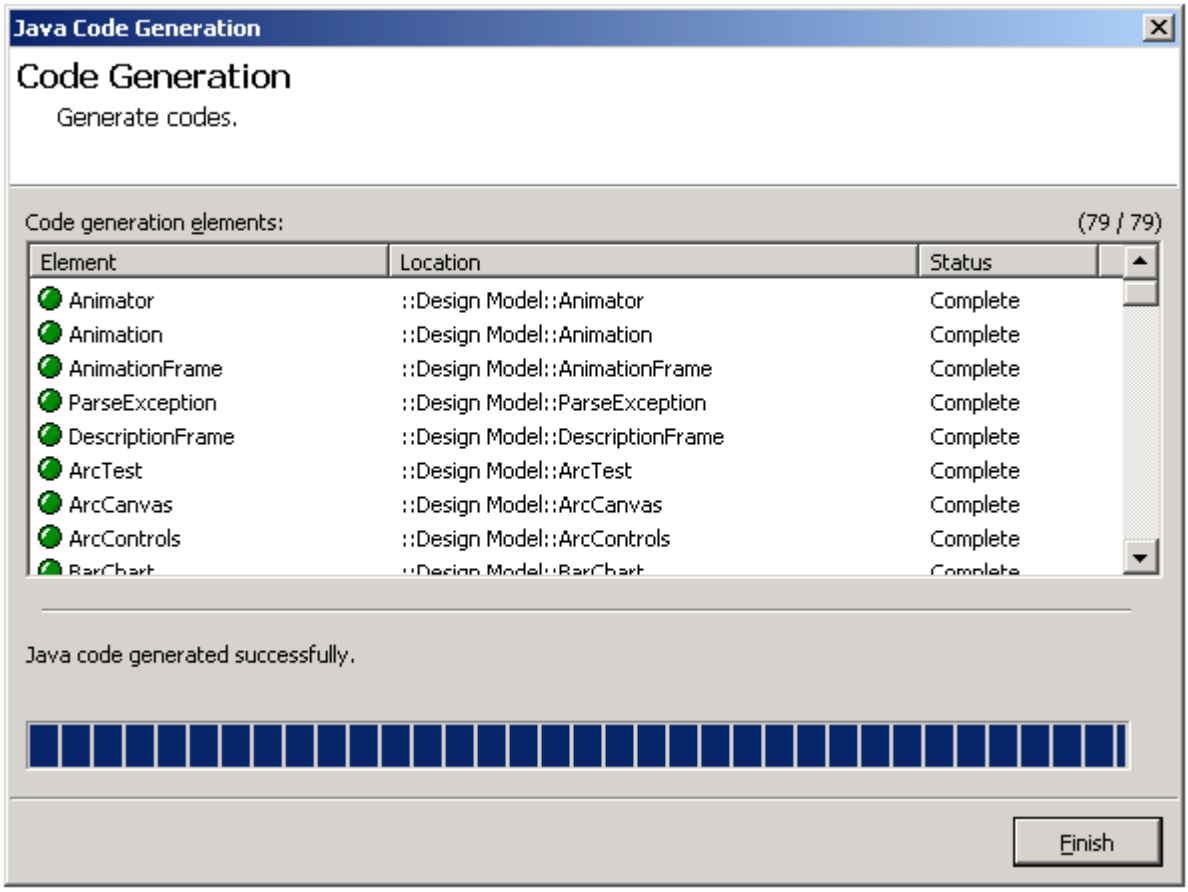
4. At the **[Output Directory Setup]** page, select a directory to save the output sources and click **[Next]**.



5. At the **[Option Setup]** page, select options and click **[Next]**. Reverse engineering will start now.



6 The **[Code Generation]** page will show the code generation progress status and
. return code generation failure or success results.



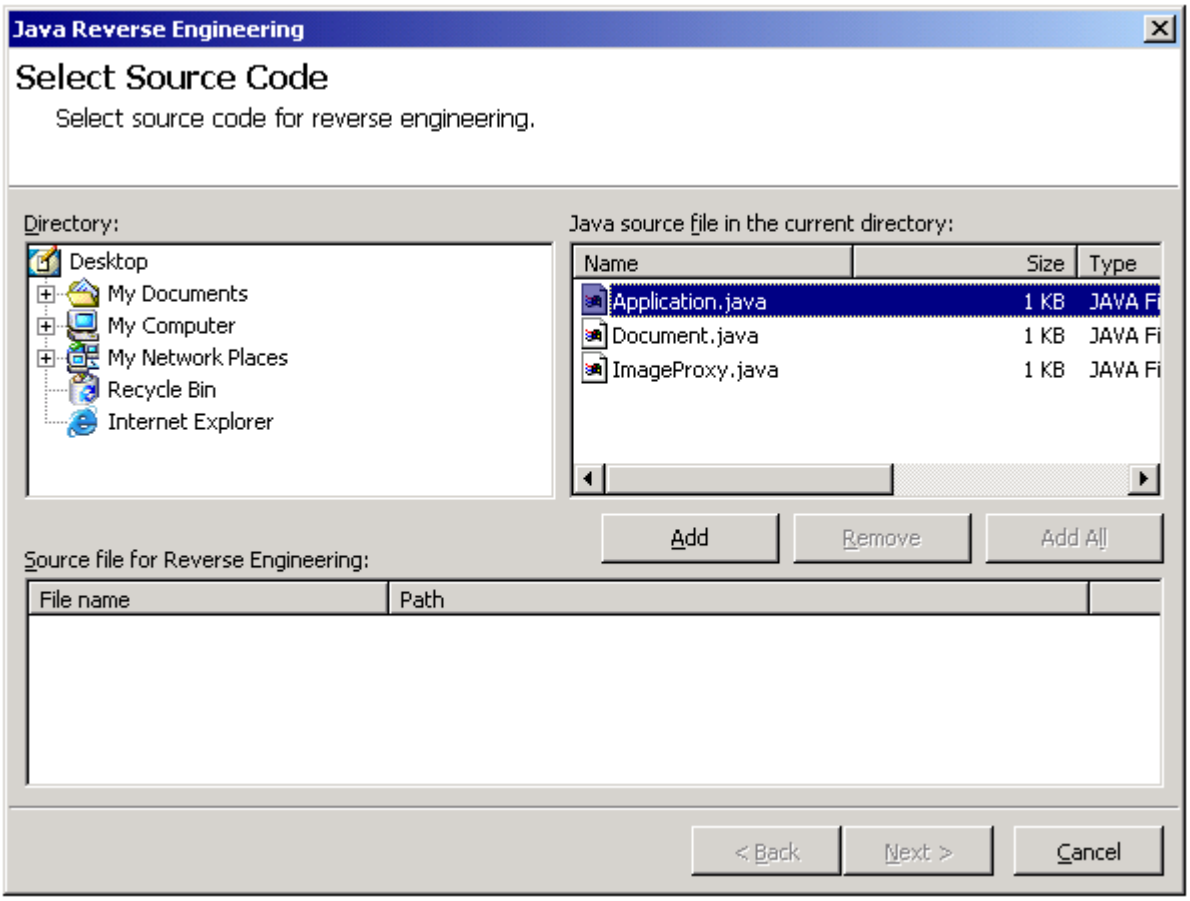
Lab Exercise No.10

Perform reverse engineering in java. (Code to Model conversion)

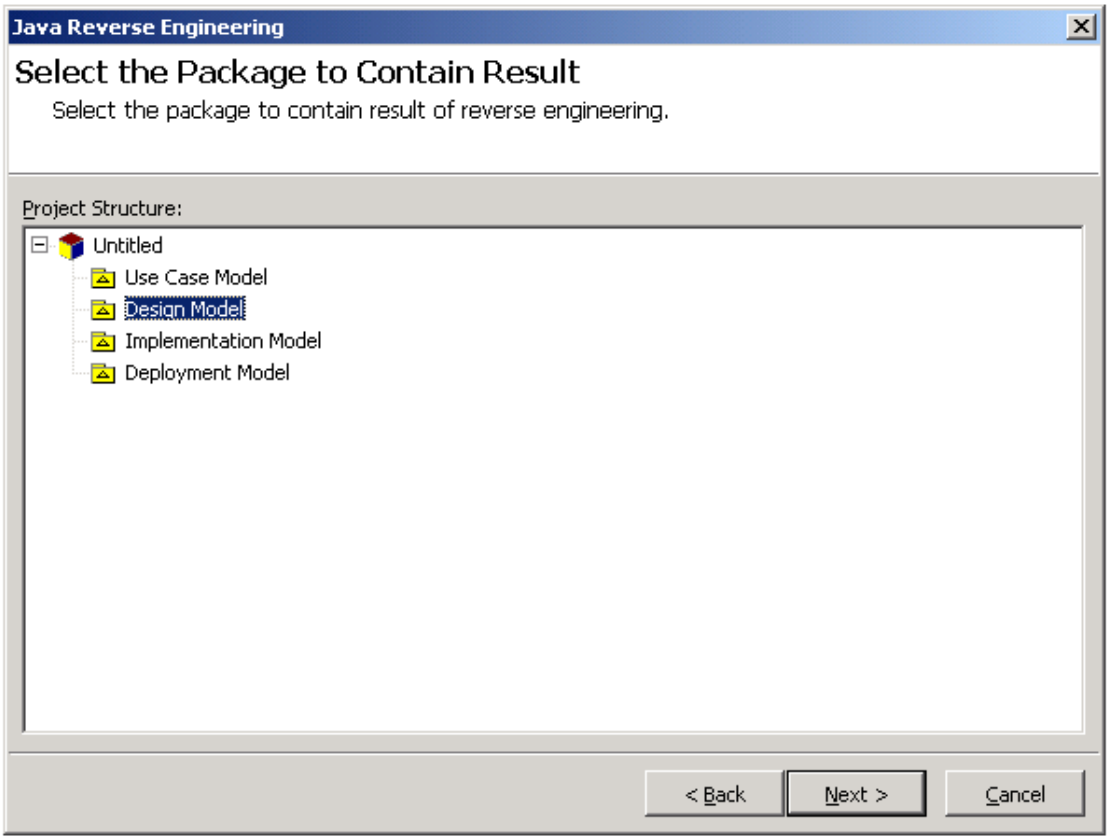
Description: Java Reverse Engineering is available in the extra plugins of Papyrus. It allows Java files or packages to be reverse-engineered into a Papyrus class diagram.

Procedure for Reverse Engineering:

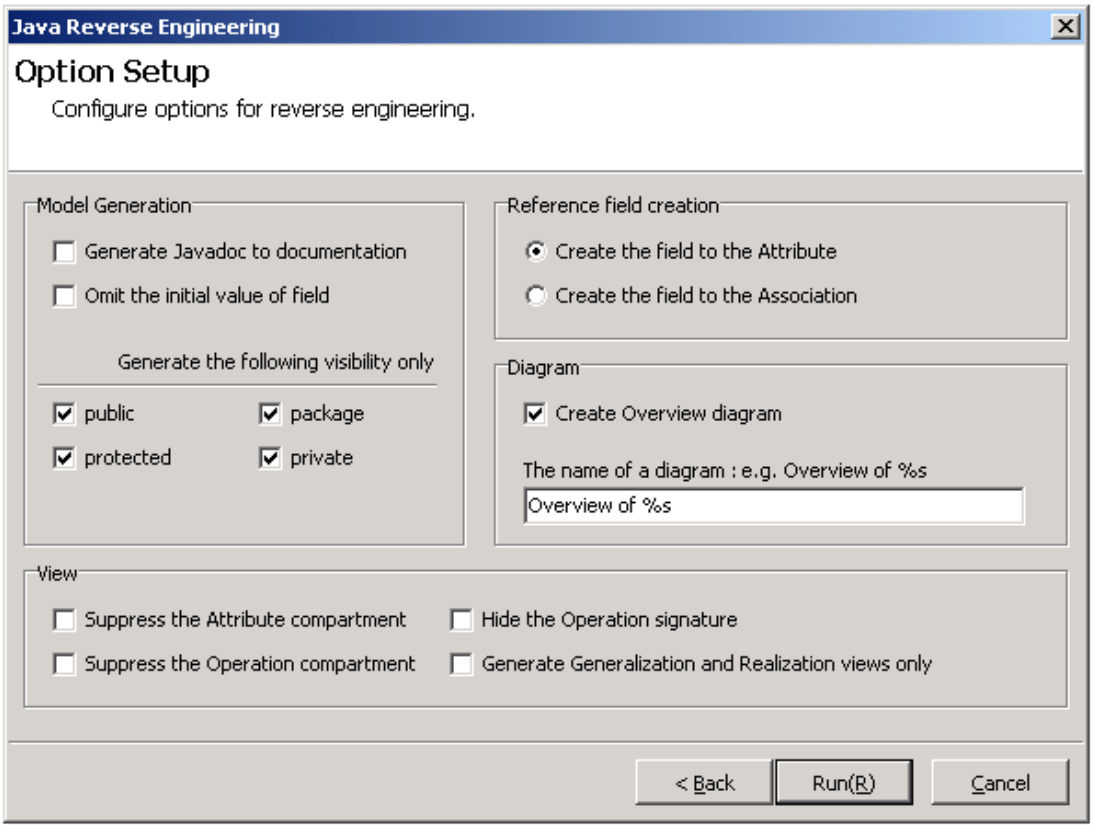
1. In StarUML(tm), select the **[Tools] -> [C#] -> [Reverse Engineer...]** menu.
2. At the **[Select Source Code]** page in the **[Java Reverse Engineering]** dialog box, select a source and click **[Add]**. Click **[Next]** once you have completed adding the target sources for reverse engineering.



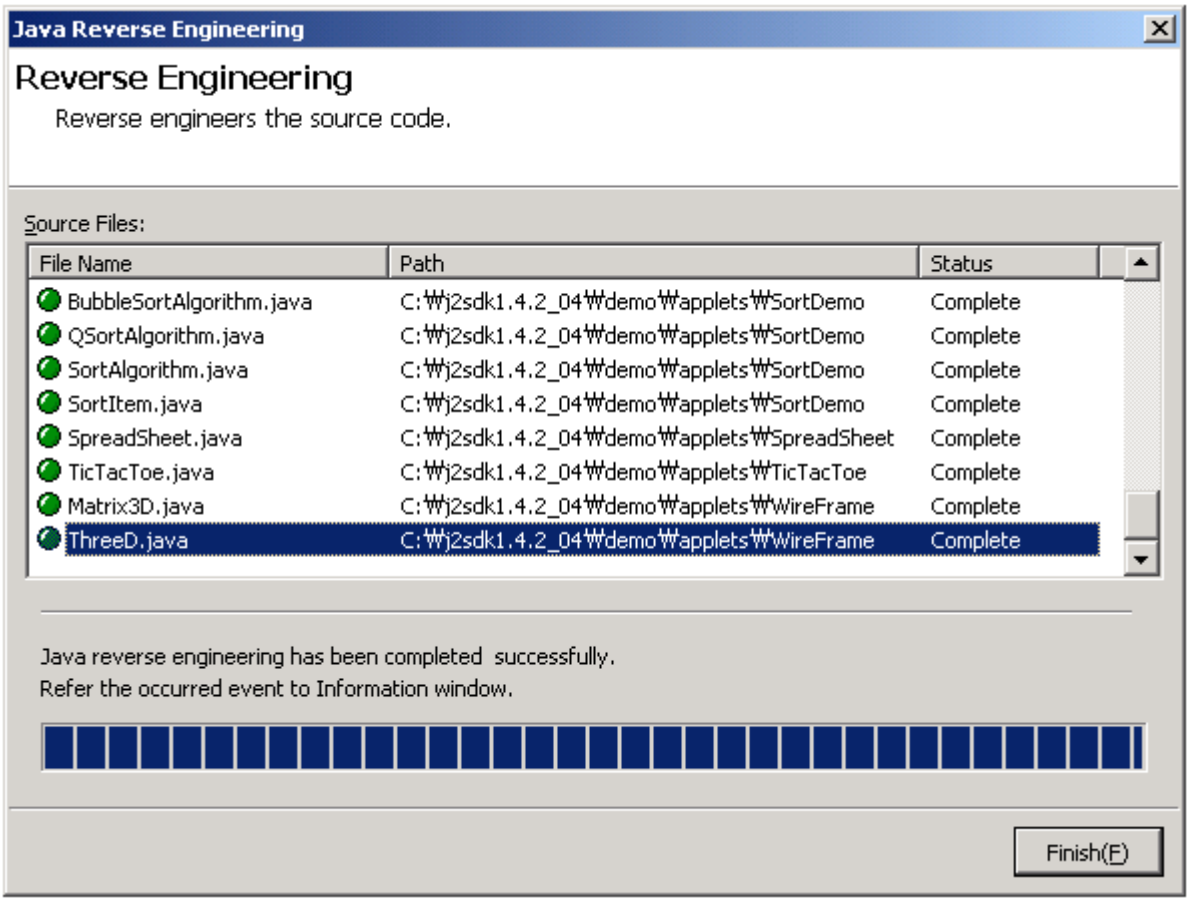
3. At the **[Select the Package to Contain Result]** page, select a package to contain the output results from the package tree and click **[Next]**.



4. At the **[Option Setup]** page, select the reverse engineering options and click **[Run]**. Reverse engineering will start now.



5. The **[Reverse Engineering]** page will show the reverse engineering progress status and return reverse engineering failure or success results.



Lab Exercise No.11

Draw the Deployment Diagram for the problem statement.

Purpose:

The name *Deployment* itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components where software components are deployed. Component diagrams and deployment diagrams are closely related.

Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.
UML is mainly designed to focus on software artifacts of a system. But these two diagrams are special diagrams used to focus on software components and hardware components.

So most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on hardware topology of a system. Deployment diagrams are used by the system engineers.
The purpose of deployment diagrams can be described as:

- Visualize hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe runtime processing nodes.

How to draw Deployment Diagram?

Deployment diagram represents the deployment view of a system. It is related to the component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardwares used to deploy the application.

Deployment diagrams are useful for system engineers. An efficient deployment diagram is very important because it controls the following parameters

- Performance
- Scalability
- Maintainability
- Portability

So before drawing a deployment diagram the following artifacts should be identified:

- Nodes

- Relationships among nodes

The following deployment diagram is a sample to give an idea of the deployment view of order management system. Here we have shown nodes as:

- Monitor
- Modem
- Caching server
- Server

The application is assumed to be a web based application which is deployed in a clustered environment using server 1, server 2 and server 3. The user is connecting to the application using internet. The control is flowing from the caching server to the clustered environment.
So the following deployment diagram has been drawn considering all the points mentioned above: