

Techno India NJR Institute of Technology



Lab Manual Software Engineering

Payal Patel
Department of CSE

Theory Session

THEORY												
SN	Category	Course		Contact hrs/week			Marks				Cr	
		Code	Title	L	T	P	Exm Hrs	IA	ETE	Total		
1	BSC	3CS2-01	Advanced Engineering Mathematics	3	0	0	3	30	120	150	3	
2	HSMC	3CS1-02/ 3CS1-03	Technical Communication/ Managerial Economics and Financial Accounting	2	0	0	2	20	80	100	2	
3	ESC	3CS3-04	Digital Electronics	3	0	0	3	30	120	150	3	
4	PCC	3CS4-05	Data Structures and Algorithms	3	0	0	3	30	120	150	3	
5		3CS4-06	Object Oriented Programming	3	0	0	3	30	120	150	3	
6		3CS4-07	Software Engineering	3	0	0	3	30	120	150	3	
Sub Total				17	0	0		170	680	850	17	

Practical Session

PRACTICAL & SESSIONAL												
7	PCC	3CS4-21	Data Structures and Algorithms Lab	0	0	3		45	30	75	1.5	
8		3CS4-22	Object Oriented Programming Lab	0	0	3		45	30	75	1.5	
9		3CS4-23	Software Engineering Lab	0	0	3		45	30	75	1.5	
10		3CS4-24	Digital Electronics Lab	0	0	3		45	30	75	1.5	
11	PSIT	3CS7-30	Industrial Training	0	0	1		0	0	50	1	
12	SODE CA	3CS8-00	Social Outreach, Discipline & Extra Curricular Activities							25	0.5	
Sub- Total				0	0	13		180	120	375	7.5	
TOTAL OF III SEMESTER				17	0	13		350	800	1225	24.5	

Note: L: Lecture; T: Tutorial; Cr: Credits; ETE: End Term Exam; IA: Internal Assessment

Syllabus



RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

II Year-III Semester: B.Tech. Computer Science and Engineering

3CS4-23: Software Engineering Lab

Credit-1.5
OL+OT+3P

Max. Marks : 100 (IA:60, ETE:40)

SN	CONTENTS
1	Development of requirements specification, function oriented design using SA/SD, object-oriented design using UML, test case design, implementation using Java and testing. Use of appropriate CASE tools and other tools such as configuration management tools, program analysis tools in the software life cycle.
2	Develop Software Requirements Specification (SRS) for a given problem in IEEE template.
3	Develop DFD model (level-0, level-1 DFD and Data dictionary) of the project.
4	Develop structured design for the DFD model developed.
5	Developed all Structure UML diagram of the given project.
6	Develop Behavior UML diagram of the given project.
7	Manage file, using ProjectLibre project management software tool.

Objectives:

- To understand the software development methodologies for software development.
- To gain the knowledge about open-source tools for computer aided software engineering i.e., CASE
- To develop an efficient software using case tools

List of Experiments

1. Develop requirements specification for a given problem.
2. Develop DFD model (level-0, level-1 and Data dictionary) of the project.
3. Develop Structured design for the DFD model developed.
4. Develop UML Use case model for a
problem.
5. Develop sequence diagram.
6. Develop Class diagrams
7. Use testing tool such as Junit.
9. Using one project management tool -Libra

Lab Experiment No.1

Develop requirements specification for a given

problem Objective:

To find the requirement specification (both functional and nonfunctional) of a given Problem.

Procedure:

Step 1:

Introduction:

Purpose

Identify the product whose software requirements are specified in this document. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem. Describe the different types of user that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers. Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.

Project Scope

Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here. An SRS that specifies the next release of an evolving product should contain its own scope statement as a subset of the long-term strategic product vision.

Step 2:

Overall Description

Product Perspective

Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.

Product Features

Summarize the major features the product contains or the significant functions that it performs or lets the user perform. Only a high level summary is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or a class diagram, is often effective.

User Classes and Characteristics

Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the favored user classes from those who are less important to satisfy.

Operating Environment

Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.

Design and Implementation Constraints

Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer's organization will be responsible for maintaining the delivered software).

Step 3:

System Features

This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.

System Feature 1

Don't really say "System Feature 1." State the feature name in just a few words.

1 Description and Priority

Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).

2 Stimulus/Response Sequences

List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.

3 Functional Requirements

Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary.

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>

REQ-1:

REQ-2:

Step 4:

External Interface Requirements

User Interfaces

Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.

Hardware Interfaces

Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.

Software Interfaces

Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.

Communications Interfaces

Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.

Nonfunctional Requirements

Performance Requirements

If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.

Safety Requirements

Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.

Security Requirements

Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.

Software Quality Attributes

Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.

Other Requirements

Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.

Questions

1. Document the SRS of College automation system.
2. Document the SRS of Banking Management System.
3. Why we need SRS in any Project.
4. Which part of SRS is more important?
5. What is the difference between functional and nonfunctional requirement.

Experiment No. 2

AIM OF THE EXPERIMENT:

Develop DFD model (level-0, level-1 DFD and Data dictionary) of the project.

OVERALL DESCRIPTION:

Data analysis attempts to answer four specific questions:

- What processes make up a system?
- What data are used in each process?
- What data are stored?
- What data enter and leave the system?

Data drive business activities and can trigger events (e.g. new sales order data) or be processed to provide information about the activity. Data flow analysis, as the name suggests, follows the flow of data through business processes and determines how organisation objectives are accomplished. In the course of handling transactions and completing tasks, data are input, processed, stored, retrieved, used, changed and output. Data flow analysis studies the use of data in each activity and documents the findings in data flow diagrams, graphically showing the relation between processes and data.

Physical and Logical DFDs

There are two types of data flow diagrams, namely *physical data flow diagrams* and *logical data flow diagrams* and it is important to distinguish clearly between the two:

Physical Data Flow Diagrams

An implementation-dependent view of the current system, showing what tasks are carried out and how they are performed. Physical characteristics can include:

Names of people

Form and document names or numbers

Master and transaction files

Equipment and devices used

Logical Data Flow Diagrams

An implementation-*independent* view of the a system, focusing on the flow of data between processes without regard for the specific devices, storage locations or people in the system. The physical characteristics listed above for physical data flow diagrams will not be specified.

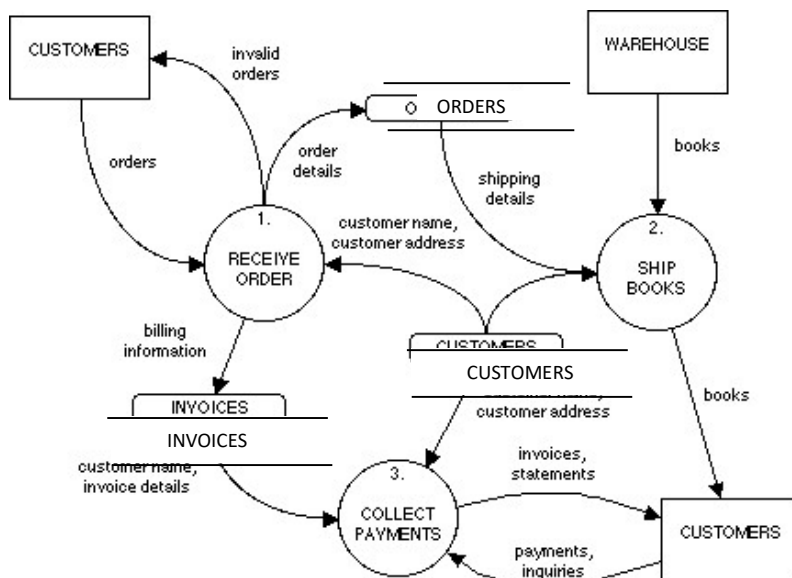


Fig. A typical DFD

Data Flow Diagram (DFD)

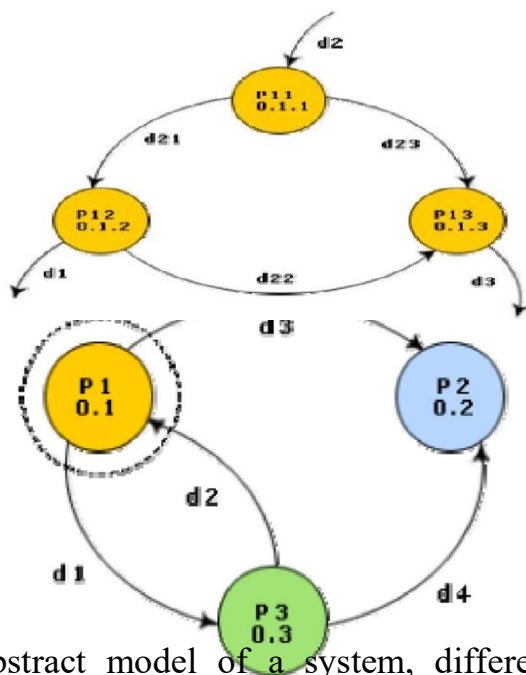
The DFD (also known as a bubble chart) is a hierarchical graphical model of a system that shows the different processing activities or functions that the system performs and the data interchange among these functions. Each function is considered as a processing station (or process) that consumes some input data and produces some output data. The system is represented in terms of the input data to the system, various processing carried out on these data, and the output data generated by the system. A

DFD model uses a very limited number of primitive symbols [as shown in figure_] represent the functions performed by a system and the data flow among these functions.

Symbols used for designing DFDs

Here, two examples of data flow that describe input and validation of data are considered. In Figure, the two processes are directly connected by a data flow. This means that the 'validate-number' process can start only after the 'read-number' process had supplied data to it. However, in Figure, the two processes are connected through a data store. Hence, the operations of the two bubbles are independent. The first one is termed 'synchronous' and the second one 'asynchronous'.

Importance of DFDs in a good software design



The main reason why the DFD technique is so popular is probably because of the fact that DFD is a very simple formalism – it is simple to understand and use. Starting with a set of high-level functions that a system performs, a DFD model hierarchically represents various sub-functions. In fact, any hierarchical model is simple to understand. Human mind is such that it can easily understand any hierarchical model of a system – because in a hierarchical model, starting with a very simple and

abstract model of a system, different details of the system are slowly introduced through different hierarchies. The data flow diagramming technique also follows a very simple set of intuitive concepts and rules. DFD is an elegant modeling technique that turns out to be useful not only to represent the results of structured analysis of a software problem, but also for several other applications such as showing the flow of documents or items in an organization.

Data dictionary

A data dictionary lists all data items appearing in the DFD model of a system. The data items listed include all data flows and the contents of all data stores appearing on the DFDs in the DFD model of a system. A data dictionary lists the purpose of all data items and the definition of all composite data items in terms of their component data items. For example, a data dictionary entry may represent that the data **grossPay** consists of the components **regularPay** and **overtimePay**.

Balancing a DFD

The data that flow into or out of a bubble must match the data flow at the next level of DFD. This is known as balancing a DFD. The concept of balancing a DFD has been illustrated in fig. 5.3. In the level 1 of the DFD, data items d1 and d3 flow out of the bubble 0.1 and the data item d2 flows into the bubble

0.1. In the next level, bubble 0.1 is decomposed. The decomposition is balanced, as d1 and d3 flow out of the level 2 diagram and d2 flows in.

Questions

1. What are the symbols used in a DFD.
2. What is an external entity?
3. What is a context free diagram?
4. What is Data-dictionary?
5. Why balancing of DFD is required.

Lab Experiment No.4

Develop Structured design for the DFD model developed.

A DFD model of a system graphically depicts the transformation of the data input to the system to the final result through a hierarchy of levels. A DFD starts with the most abstract definition of the system (lowest level) and at each higher level

DFD, more details are successively introduced. To develop a higher-level DFD model, processes are decomposed input data to these functions and the data output by these functions and represent them appropriately in the diagram.

If a system has more than 7 high-level functional requirements, then some of the related requirements have to be combined and represented in the form of a bubble in the level 1 DFD. Such a bubble can be split in the lower DFD levels. If a system has less than three high-level functional requirements, then some of them need to be split into their sub-functions so that we have roughly about 5 to 7 bubbles on the diagram.

Decomposition:-

Each bubble in the DFD represents a function performed by the system. The bubbles are decomposed into sub-functions at the successive levels of the DFD.

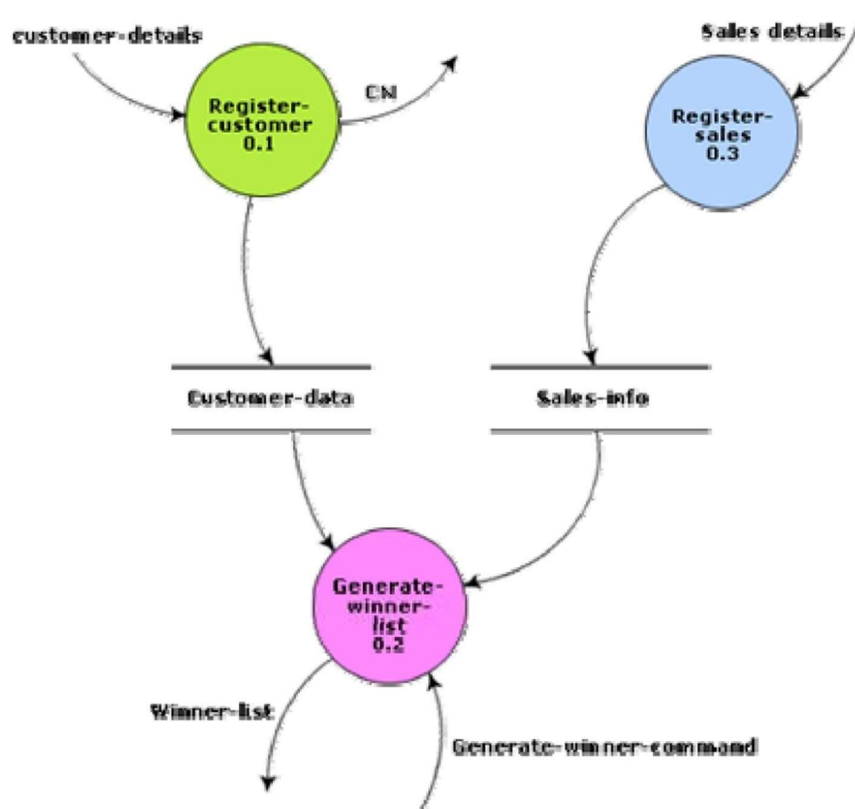
Decomposition of a bubble is also known as factoring or exploding a bubble. Each bubble at any level of DFD is usually decomposed to anything between 3 to 7 bubbles. Too few bubbles at any level make that level superfluous. For example, if a bubble is decomposed to just one bubble or two bubbles, then this decomposition becomes redundant. Also, too many bubbles, i.e. more than 7 bubbles at any level of a DFD makes the DFD model hard to understand. Decomposition of a bubble should be carried on until a level is reached at which the function of the bubble can be described using a simple algorithm.

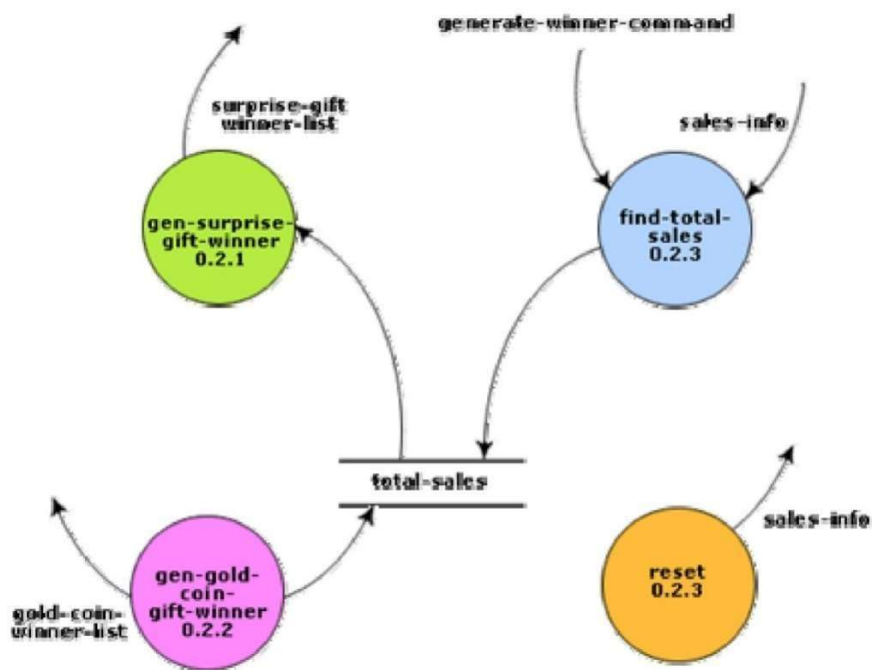
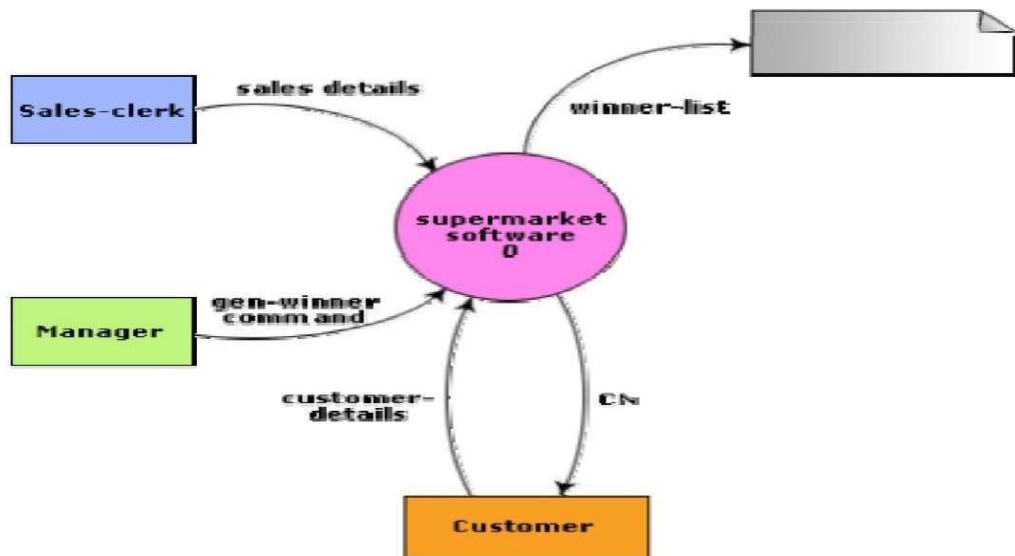
Numbering of Bubbles:-

It is necessary to number the different bubbles occurring in the DFD. These numbers help in uniquely identifying any bubble in the DFD by its bubble number. The bubble at the context level is usually assigned the number 0 to indicate that it is the 0 level DFD. Bubbles at level 1 are numbered, 0.1, 0.2, 0.3, etc, etc. When a bubble numbered x is decomposed, its children bubble are numbered x.1, x.2, x.3, etc. In this numbering scheme, by looking at the number of a bubble we can unambiguously determine its level, its ancestors, and its successors.

Example:-

A supermarket needs to develop the following software to encourage regular customers. For this, the customer needs to supply his/her residence address, telephone number, and the driving license number. Each customer who registers for this scheme is assigned a unique customer number (CN) by the computer. A customer can present his CN to the check out staff when he makes any purchase. In this case, the value of his purchase is credited against his CN. At the end of each year, the supermarket intends to award surprise gifts to 10 customers who make the highest total purchase over the year. Also, it intends to award a 22 caret gold coin to every customer whose purchase exceeded Rs.10,000. The entries against the CN are the reset on the day of every year after the prize winners' lists are generated.





Questions

1. Draw the DFD of College Automation System.
2. How we balance a DFD.
3. Draw the DFD of Banking Management System.
4. How we choose the level of DFD.
5. What is the need of DFD in a project

Experiment No.4

Develop UML Use case model for a

problemObjective :

To understand the users view of a project using Use case Diagram

Software Required :-

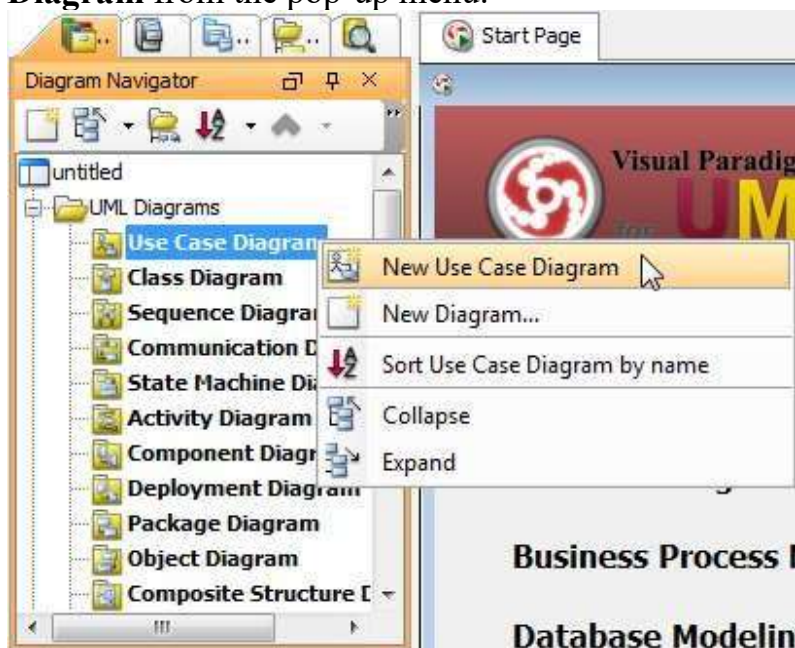
Visual Paradigm for UML 8.2

Procedure :-

You can draw use case diagrams in VP-UML as well as to document the event flowsof use cases using the flow-of-events editor of UML 8.2 .The steps are as follows.

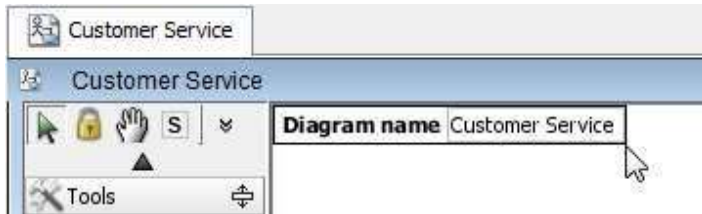
Step 1:

Right click **Use Case Diagram** on **Diagram Navigator** and select **New Use Case Diagram** from the pop-up menu.



Step 2:-

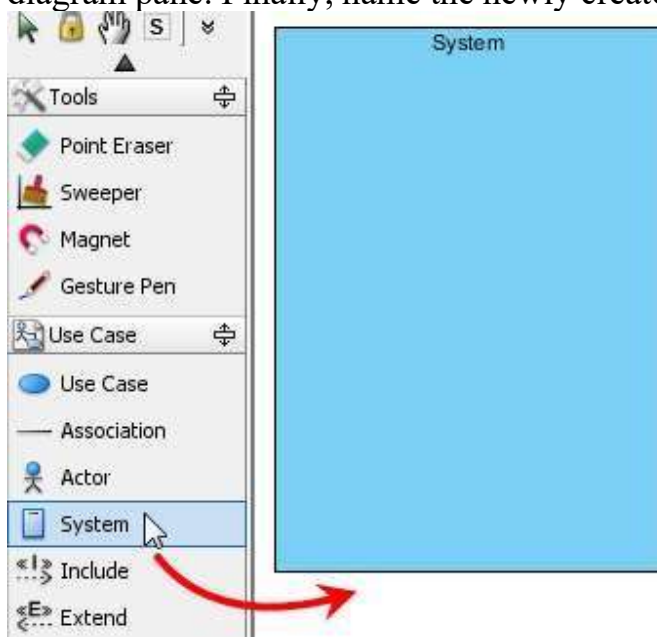
Enter name for the newly created use case diagram in the text field of pop-up box on the top left corner.



Step 3:

Drawing a system

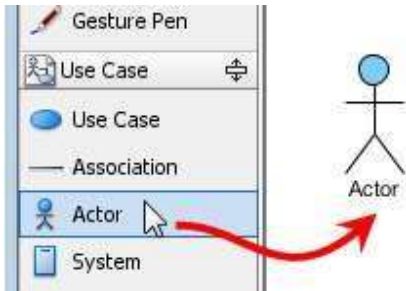
To create a system, select **System** on the diagram toolbar and then click it on the diagram pane. Finally, name the newly created system when it is created.



Step 4:

Drawing an actor

To draw an actor, select **Actor** on the diagram toolbar and then click it on the diagram pane. Finally, name the newly created actor when it is created.

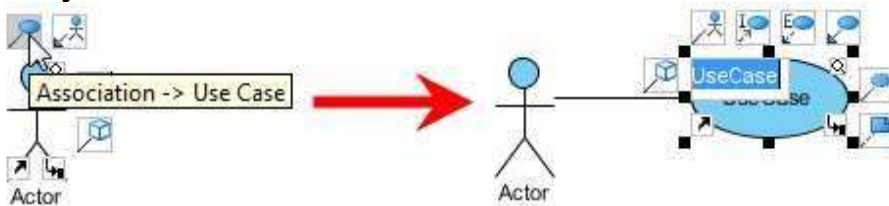


Step 5 :-

Drawing a use case

Besides creating a use case through diagram toolbar, you can also create it through resource icon.

Move the mouse over a shape and press a resource icon that can create use case. Drag it and then release the mouse button until it reaches to your preferred place. The source shape and the newly created use case are connected. Finally, name the newly created use case.



Step 6:-

Create a use case through resource icon

Line wrapping use case name

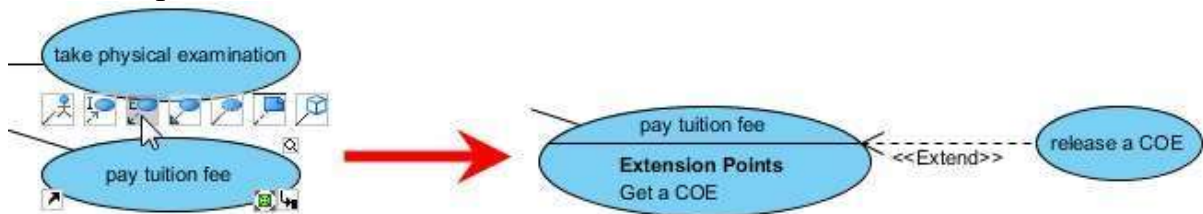
If a use case is too wide, for a better outlook, you may resize it by dragging the filled selectors. As a result, the name of use case will be line-wrapped automatically.



Step 7:

Resize a use case

To create an extend relationship, move the mouse over a use case and press its resource icon **Extend -> Use Case**. Drag it to your preferred place and then release the mouse button. The use case with extension points and a newly created use case are connected. After you name the newly created use case, a pop-up dialog box will ask whether you want the extension point to follow the name of use case. Click **Yes** if you want it to do so; click **NO** if you want to enter another name for extension point.

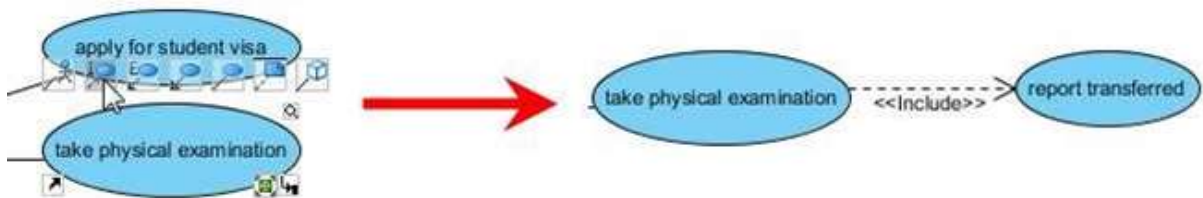


Step 8:

Create an extend relationship

Drawing <<Include>> relationship

To create an include relationship, mouse over a use case and press its resource icon **Include -> Use Case**. Drag it to your preferred place and then release the mouse button. A new use case together with an include relationship is created. Finally, name the newly created use case.



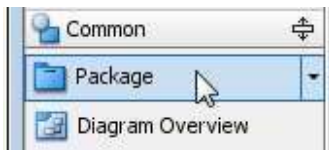
Step 9:

Include relationship
iscreated

Structuring use cases with package

You can organize use cases with package when there are many of them on the diagram.

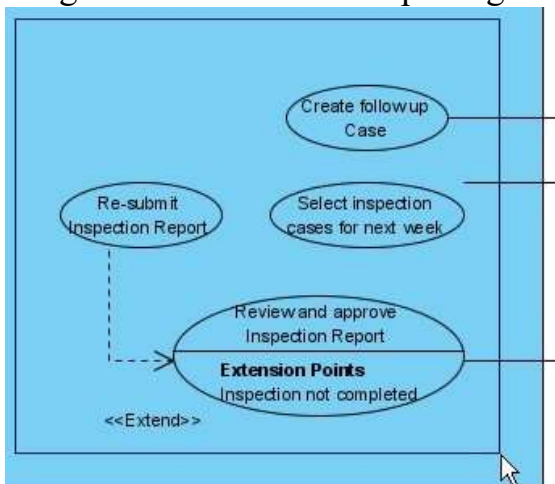
Select **Package** on the diagram toolbar (under **Common** category).



Step 10:

Create a package

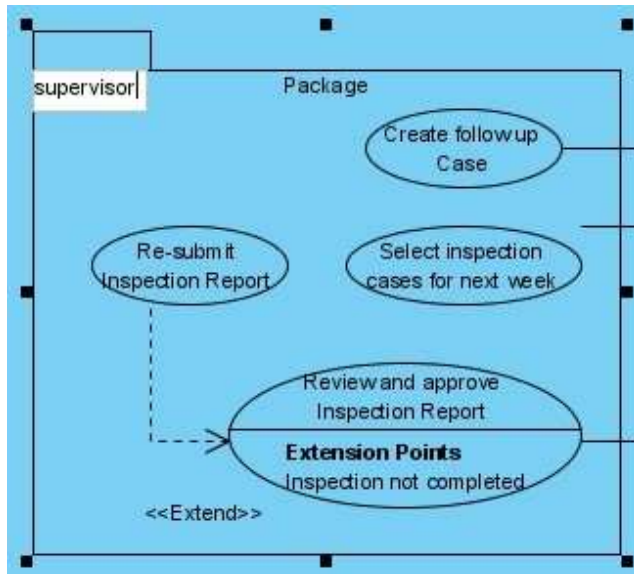
Drag the mouse to create a package surrounding those use cases.



Step 11:

Surround use cases with package

Finally, name the package.



Step 12

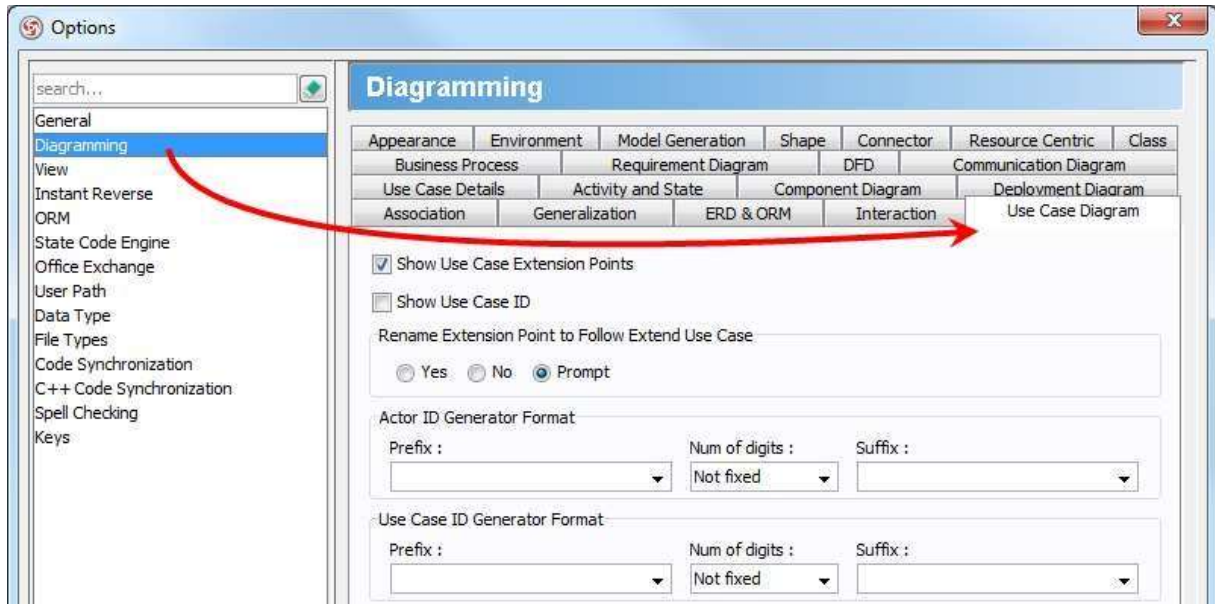
Name the package

Assigning IDs to actors/Use cases

You may assign IDs to actors and use cases. By default, IDs are assigned with the order of object creation, starting from one onwards. However, you can define the format or even enter an ID manually.

Defining the format of ID

To define the format of ID, select **Tools > Options** from the main menu to unfold the **Options** dialog box. Select **Diagramming** from the list on the left hand side and select the **Use Case Diagram** tab on the right hand side. You can adjust the format of IDs under **Use Case Diagram** tab. The format of ID consists of prefix, number of digits and suffix.



Step 13:

Use Case Diagram tab

The description of options for ID generator format is shown below.

Option

Description

Prefix The prefix you enter in **Prefix** text field will be inserted before the number.

Num of digits The number of digits for the number. For example, when digit is 3, ID "1" will become "001".

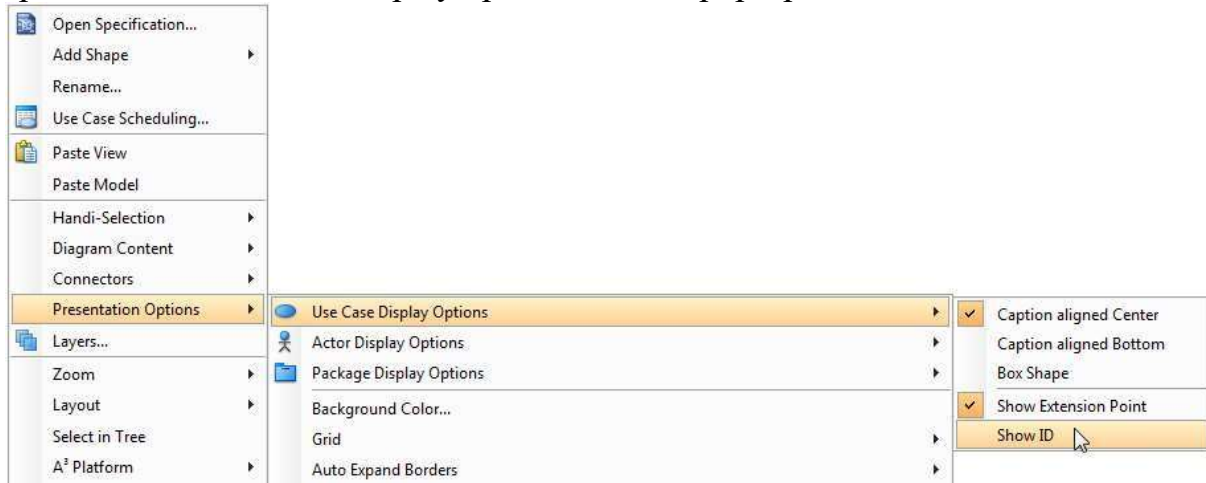
Suffix The suffix you enter in **Suffix** text field will be inserted behind the number.

Options for formatting ID

Showing ID on diagram

By default, ID is just a text property. It usually doesn't appear on diagram. However, you can make it shown within a use case.

Right click on the diagram background, select **Presentation Options** and the specific model element display option from the pop-up menu.



Step 14 :

Show ID on diagram

As a result, the use case is displayed with ID.



A use case with
ID displayed

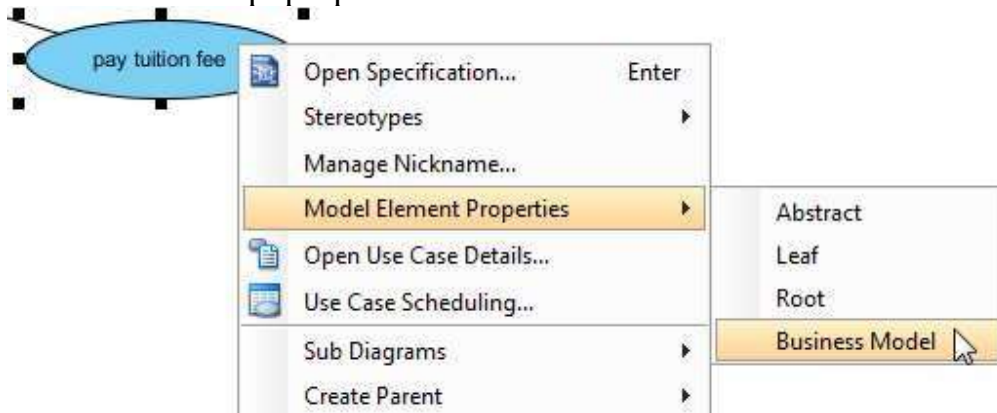
NOT E: The feature of showing ID does only support for use case,
but not for actor.

ID assignment

There are several ways that you can assign an ID to a model element, including:

- Through the specification dialog box (Right click on the selected model element and select **Open Specification...** from the pop-up menu)
 - Through the **Property Pane**
- Drawing business use case

1. Right click on a use case and select **Model Element Properties > Business Model** from the pop-up menu.



Step 15:

1. Click **Business Model**
2. After selected, an extra slash will be shown on the left edge of the use case.



Business model

And Finally The Use case Diagram is ready.

Questions

1. What is the importance of UML.
2. What are the UML foundations.
3. What is Use case.
4. Who are the actors in a UML.

5. What is boundary in a USE
CASE.

Lab Experiment No.5

Develop sequence

diagram Objective :

To understand the interactions between objects that are represented as lifelines in a sequential order of a project using Sequence Diagram.

Software Required :-

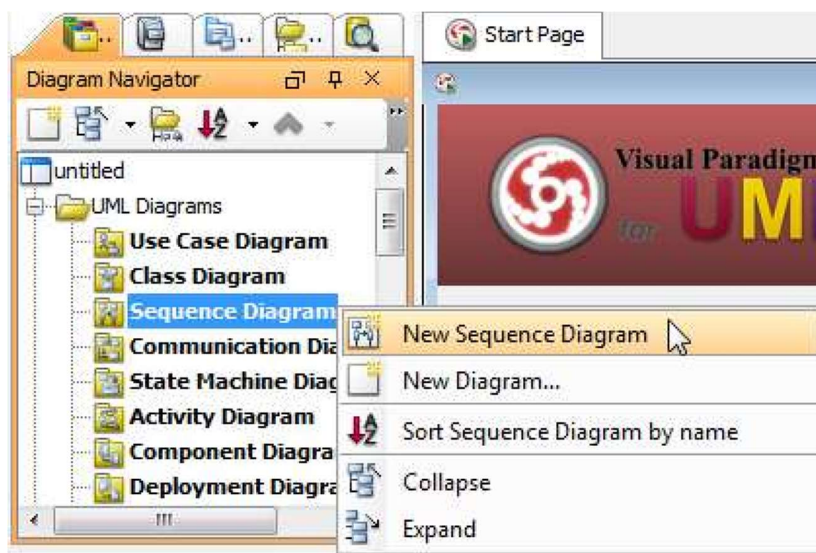
Visual Paradigm for UML 8.2

Procedure :-

A sequence diagram is used primarily to show the interactions between objects that are represented as lifelines in a sequential order.

Step 1:-

Right click **Sequence diagram** on **Diagram Navigator** and select **New Sequence Diagram** from the pop-up menu to create a sequence diagram.

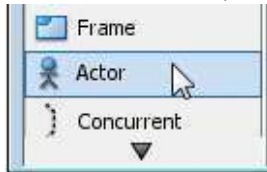


Step 2:-

Enter name for the newly created sequence diagram in the text field of pop-up box on the top left corner.

Creating actor

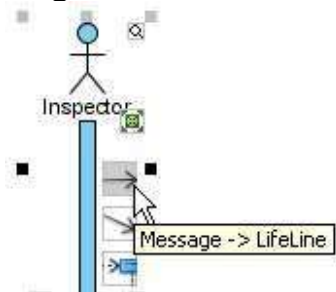
To create actor, click **Actor** on the diagram toolbar and then click on the diagram.



Creating lifeline

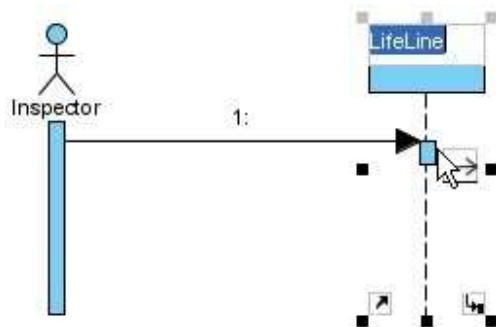
To create lifeline, you can click **LifeLine** on the diagram toolbar and then click on the diagram.

Alternatively, a much quicker and more efficient way is to use the resource-centric interface. Click on the **Message -> LifeLine** resource beside an actor/lifeline and drag.



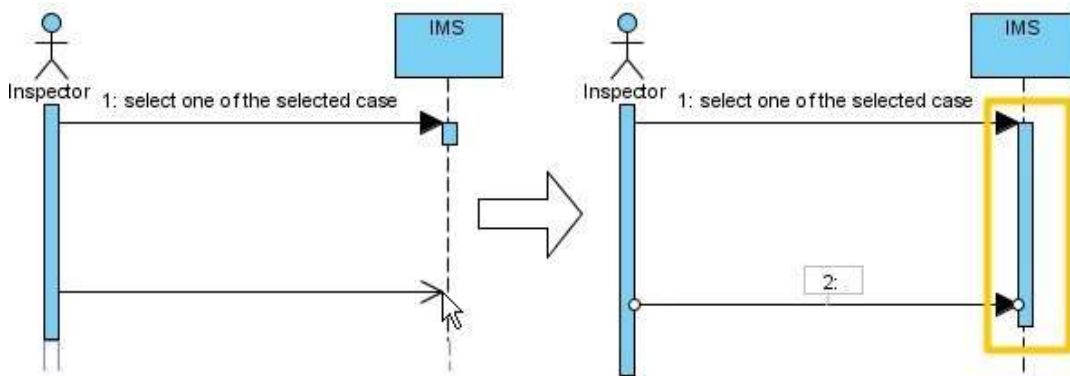
Step 3:-

Move the mouse to empty space of the diagram and then release the mouse button. A new lifeline will be created and connected to the actor/lifeline with a message.



Auto extending activation

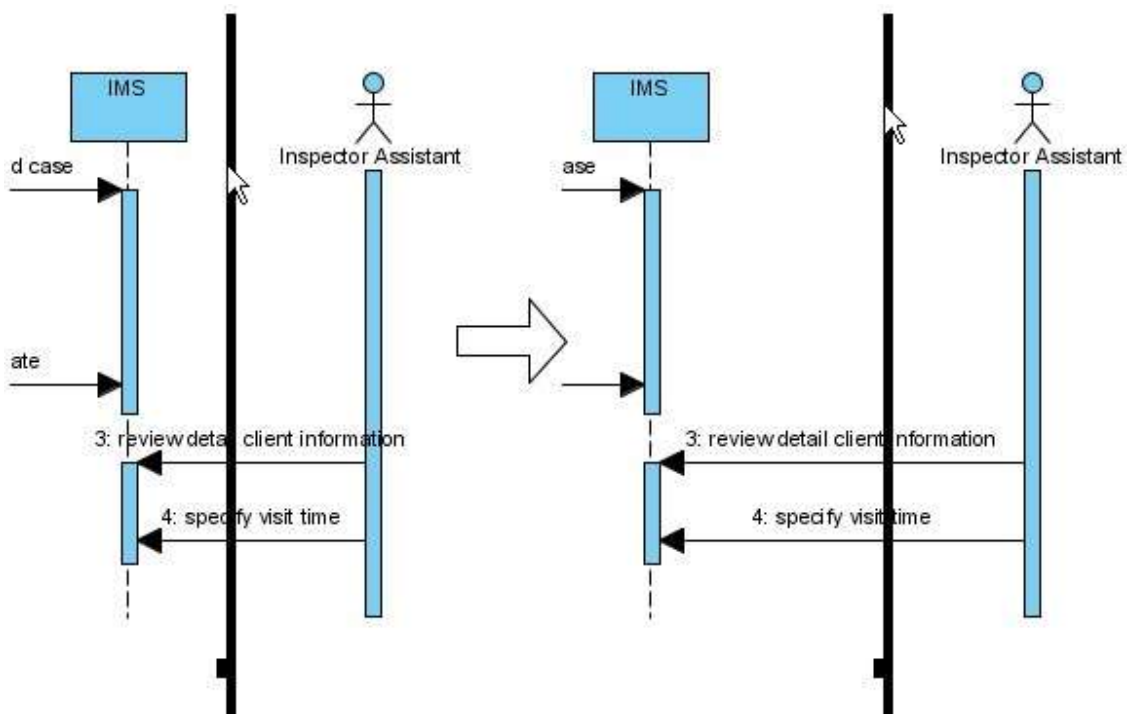
When create message between lifelines/actors, activation will be automatically extended.



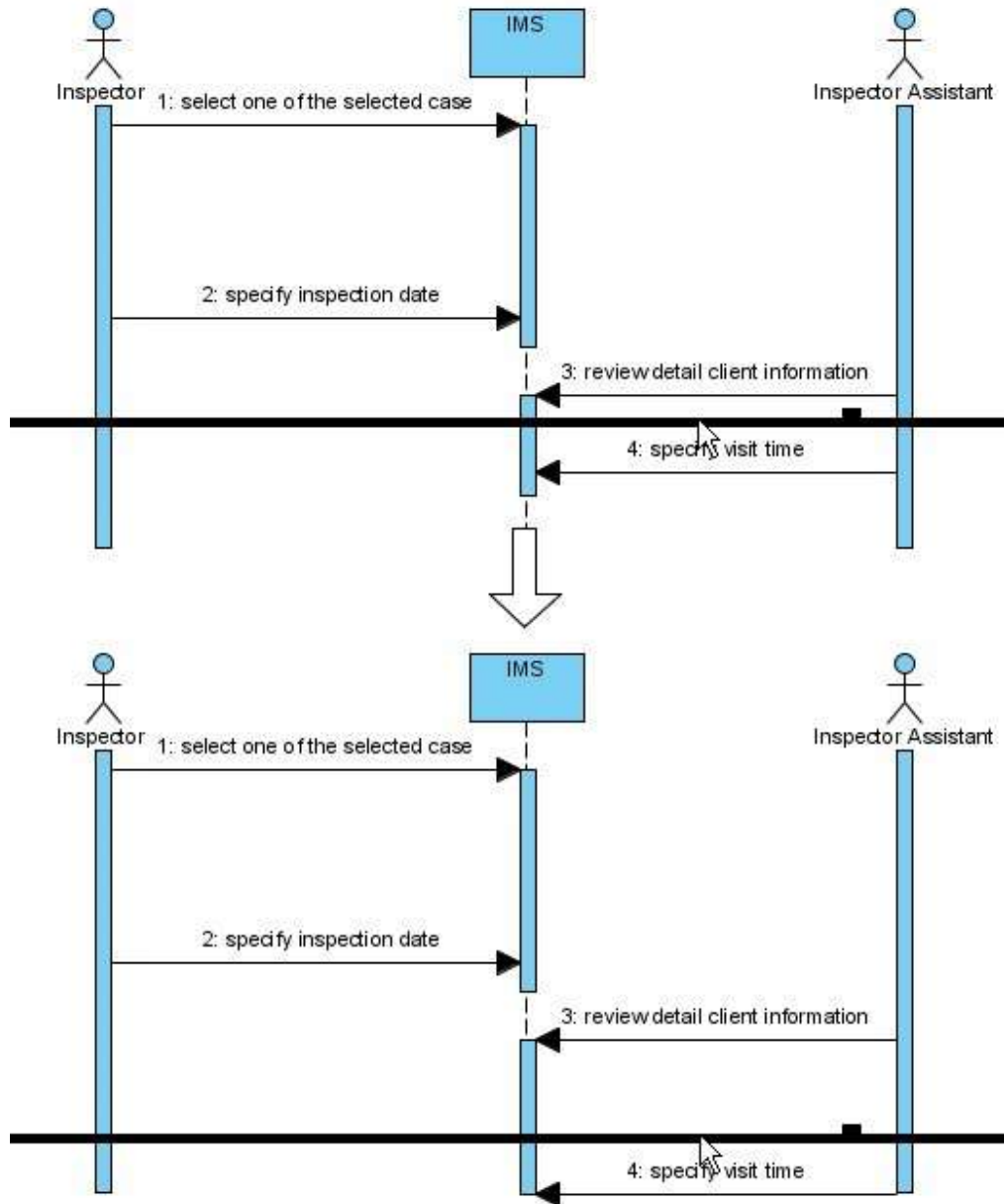
Step 4:-

Using sweeper and magnet to manage sequence diagram

Sweeper helps you to move shapes aside to make room for new shapes or connectors. To use sweeper, click **Sweeper** on the diagram toolbar (under the **Tools** category).



The picture below shows the message *specify visit time* is being swept downwards, thus new room is made for new messages.



Step 5:-

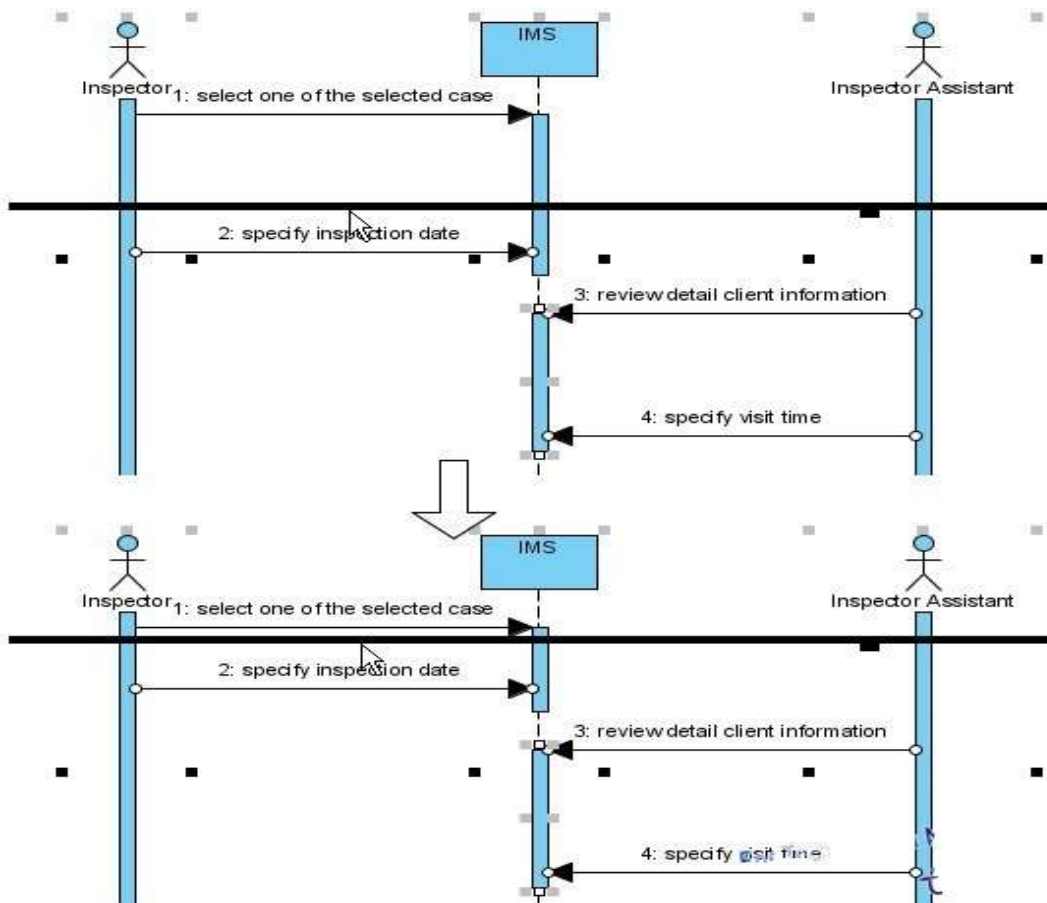
You can also use magnet to pull shapes together. To use magnet, click **Magnet** on the diagram toolbar (under the **Tools** category).



Magnet

Click on empty space of the diagram and drag towards top, right, bottom or left. Shapes affected will be pulled to the direction you dragged.

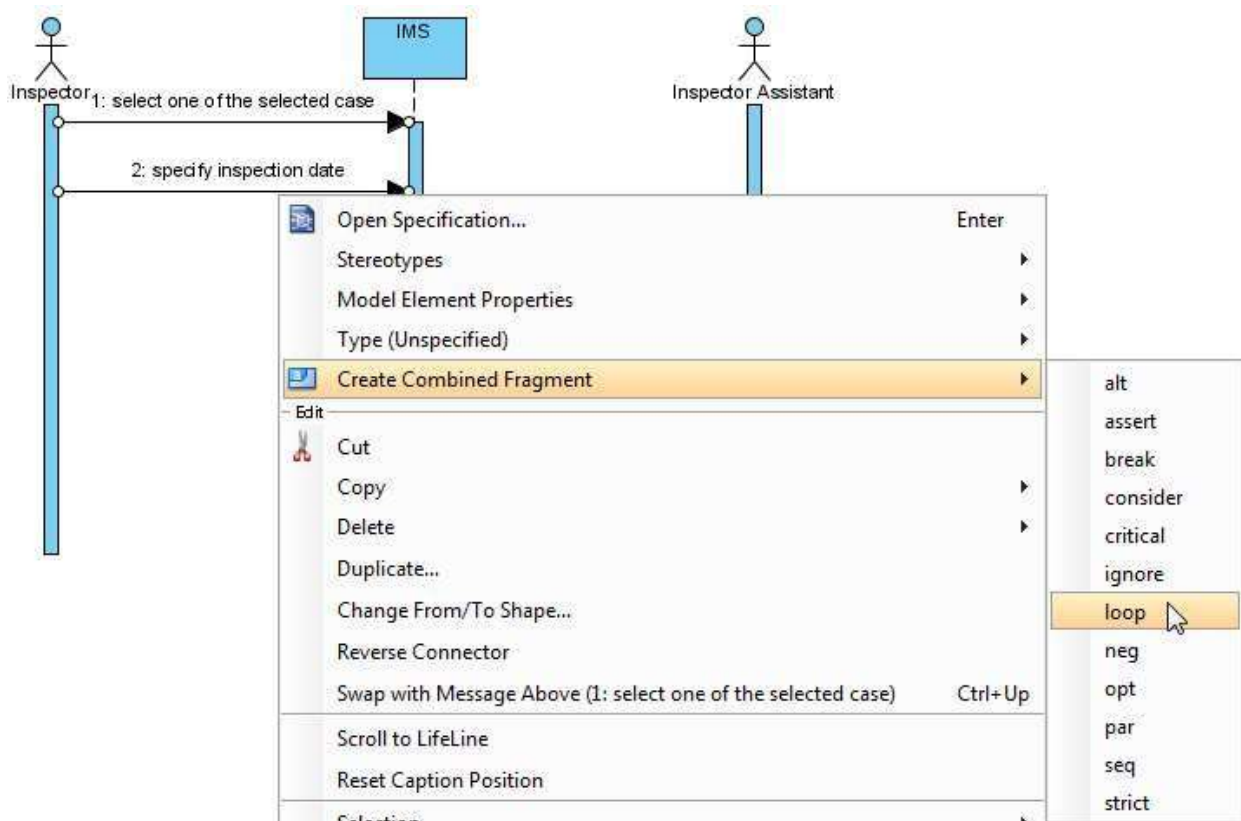
The picture below shows when drag the magnet upwards, shapes below dragged position are pulled upwards.



Step 6:-

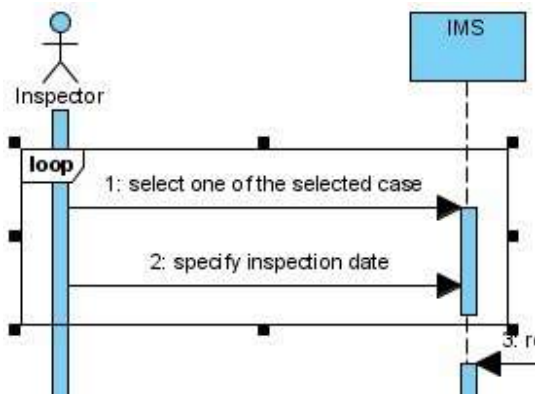
Creating combined fragment for messages

To create combined fragment to cover messages, select the messages, right-click on the selection and select **Create Combined Fragment**, and then select a combined fragment type (e.g. loop) from the popup menu.



Step 7:-

A combined fragment of selected type will be created to cover the messages.

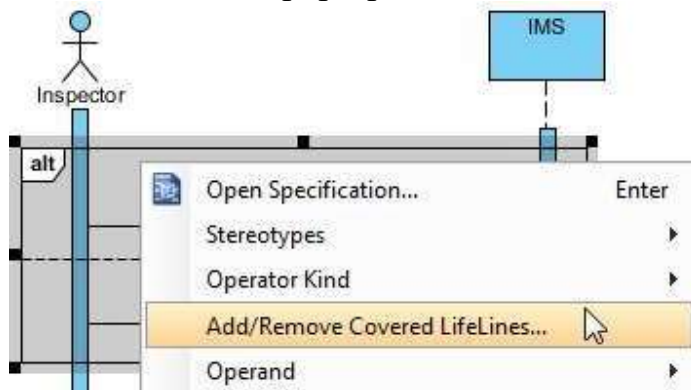


Step 8:-

Adding/removing covered lifelines

After you've created a combined fragment on the messages, you can add or remove the covered lifelines.

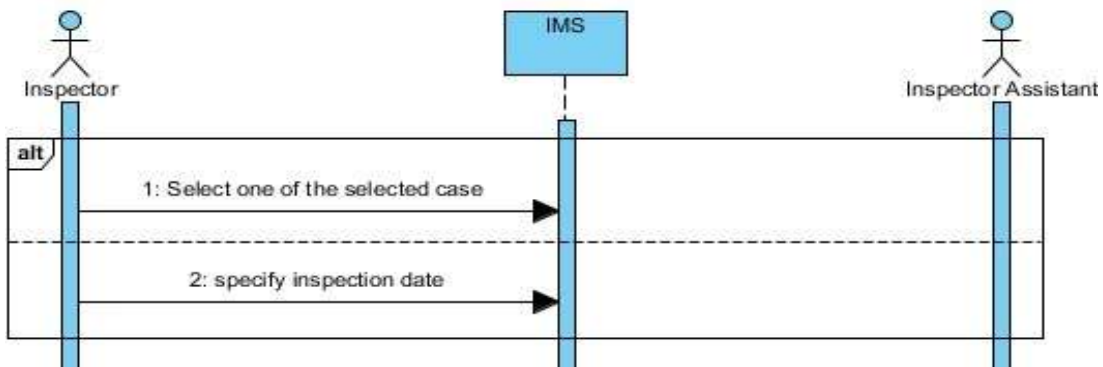
1. Move the mouse over the combined fragment and select **Add/Remove Covered Lifeline...** from the pop-up menu.



2. In the **Add/Remove Covered Lifelines** dialog box, check the lifeline(s) you want to cover or uncheck the lifeline(s) you don't want to cover. Click **OK** button.



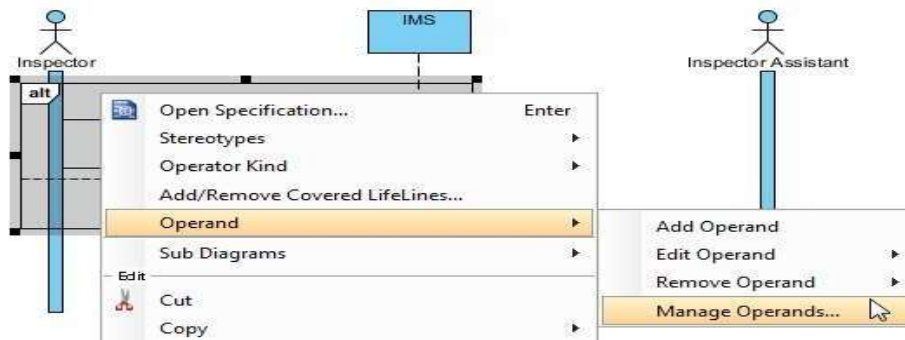
3. As a result, the area of covered lifelines is extended or narrowed down according to your selection.



Managing Operands

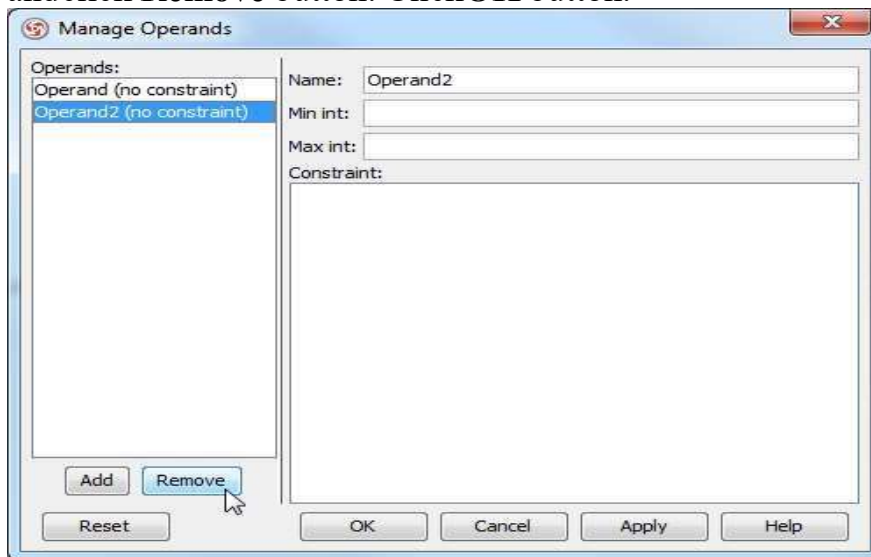
After you've created a combined fragment on the messages, you can also add or remove operand(s).

1. Move the mouse over the combined fragment and select **Operand > Manage Operands...** from the pop-up menu.



Step 9:-

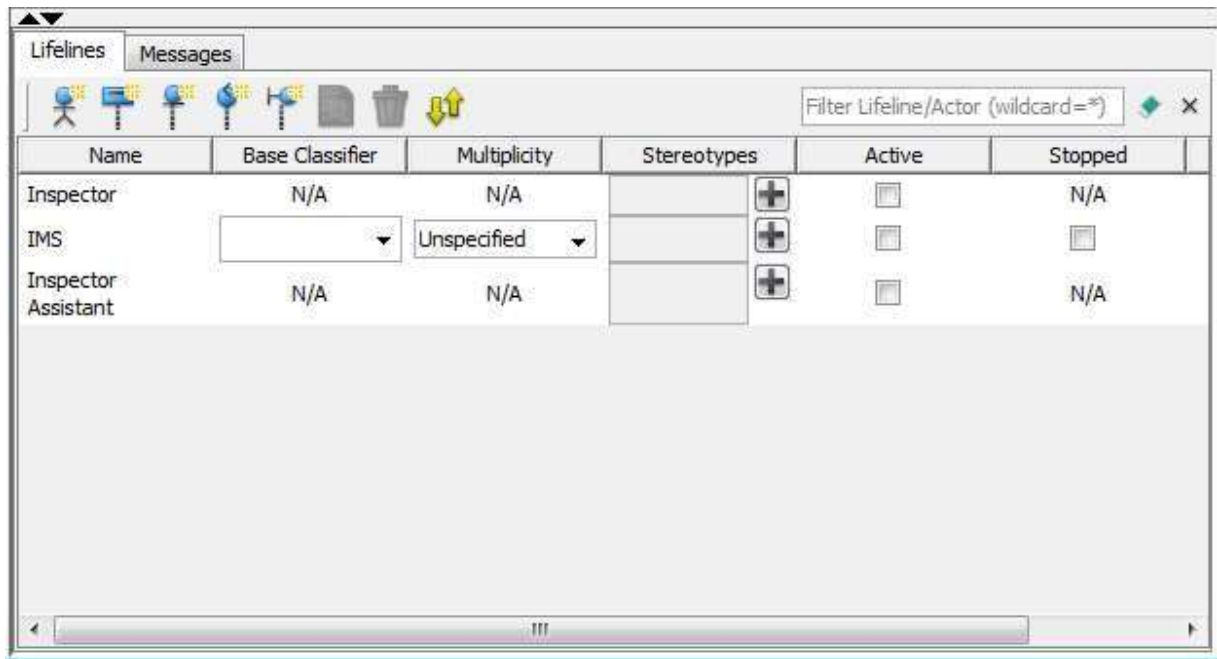
1. To remove an operand, select the target operand from **Operands** and click **Remove** button. Click **OK** button.



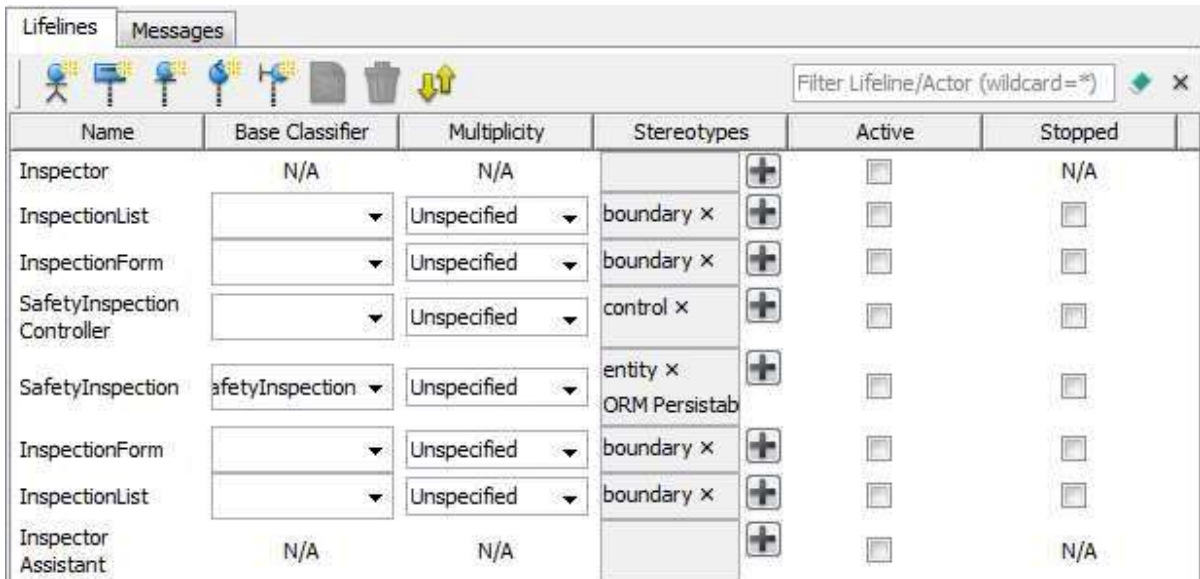
2. Otherwise, click **Add** button to add a new operand and then name it. Click **OK** button.

Developing sequence diagram with quick editor or keyboard shortcuts









In sequence diagram, an editor appears at the bottom of diagram by default, which enables you to construct sequence diagram with the buttons there. The shortcut keys assigned to the buttons provide a way to construct diagram through keyboard. Besides constructing diagram, you can also access diagram elements listing in the editor.



There are two panes, **Lifelines** and **Messages**. The **Lifelines** pane enables you to create different kinds of actors and lifelines.



B utt On	Shortcut	Description

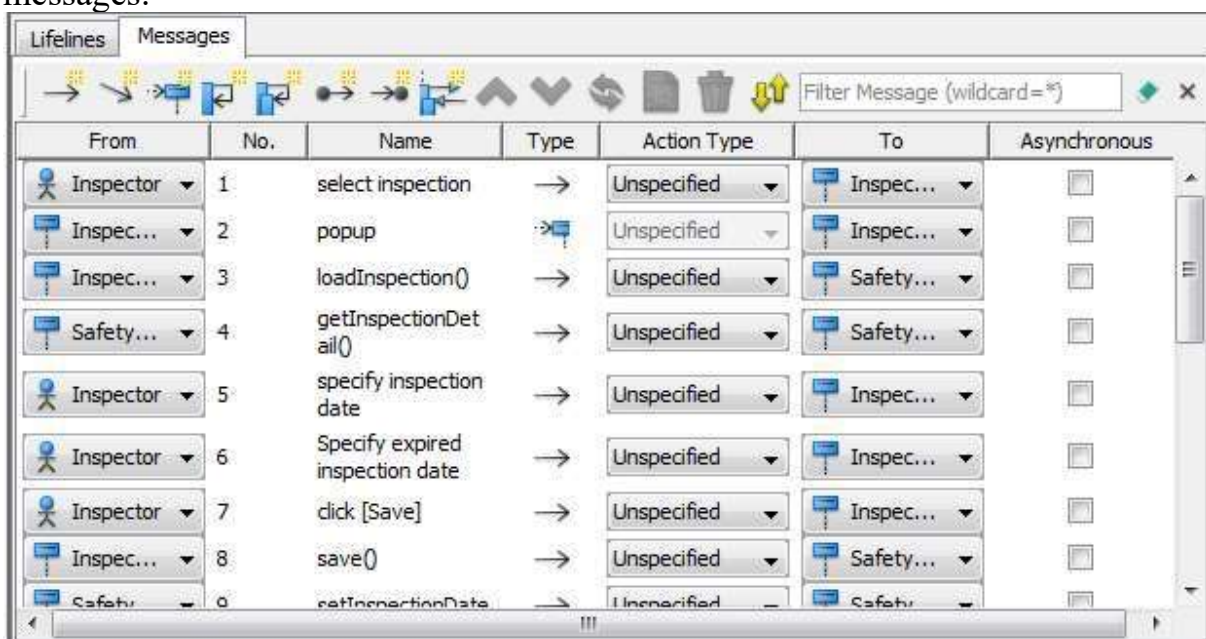
	Alt-Shift-A	To create an actor
	Alt-Shift-L	To create a general lifeline
	Alt-Shift-E	To create an <<entity>> lifeline
	Alt-Shift-C	To create a <<control>> lifeline
	Alt-Shift-B	To create a <<boundary>> lifeline
	Alt-Shift-O	To open the specification of the element chosen in quick editor
	Ctrl-Del	To delete the element chosen in quick editor
	Ctrl-L	To link with the diagram, which cause the diagram element to be selected when selecting an element in editor, and vice versa

Step 10:-

Buttons in
Lifelinespane

Editing messages

The **Messages** pane enables you to connect lifelines with various kinds of messages.



Messages pane in quick editor

Button		
Shortcut	Description	
	Alt-Shift-M	To create a message that connects actors/lifelines
	Alt-Shift-D	To create a duration message that connects actor diagram
	Alt-Shift-C	To create a create message that connects actor diagram
	Alt-Shift-S	To create a self message on an actor/lifeline in dia
	Alt-Shift-R	To create a recursive message on an actor/lifeline
	Alt-Shift-F	To create a found message that connects to an act
	Alt-Shift-L	To create a lost message from an actor/lifeline
	Alt-Shift-E	To create a reentrant message that connects actor diagram
	Ctrl-Shift-Up	To swap the chosen message with the one above
	Ctrl-Shift-Down	To swap the chosen message with the one below
	Ctrl-R	To revert the direction of chosen message
	Alt-Shift-O	To open the specification of the message chosen i
	Ctrl-Del	To delete the message chosen in quick editor
	Ctrl-L	To link with the diagram, which cause the m selected when selecting a message in editor, and v
Buttons in Messages pane		

Expanding and collapsing the editor

To hide the editor, click on the down arrow button that appears at the bar on top of the quick editor. To expand, click on the up arrow button.



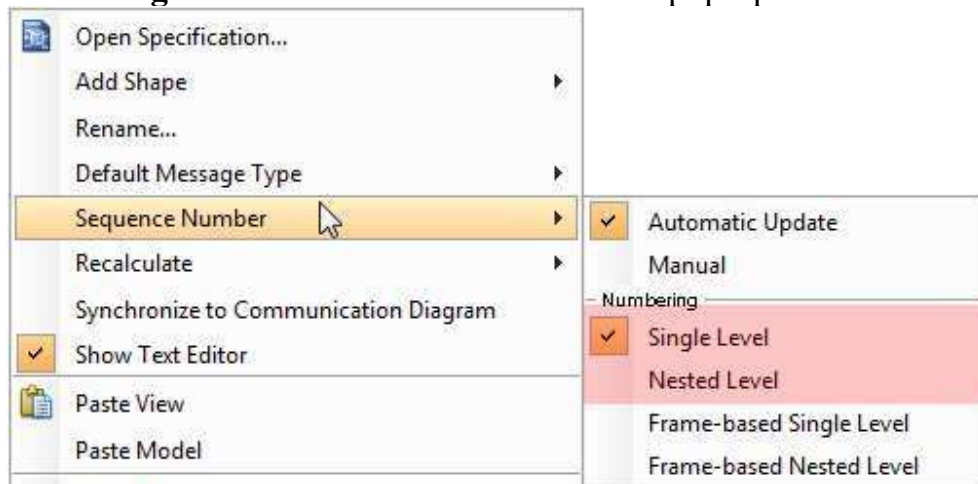
Collapse the quick editor

Setting different ways of numbering sequence messages

You are able to set the way of numbering sequence messages either on diagram base or frame base.

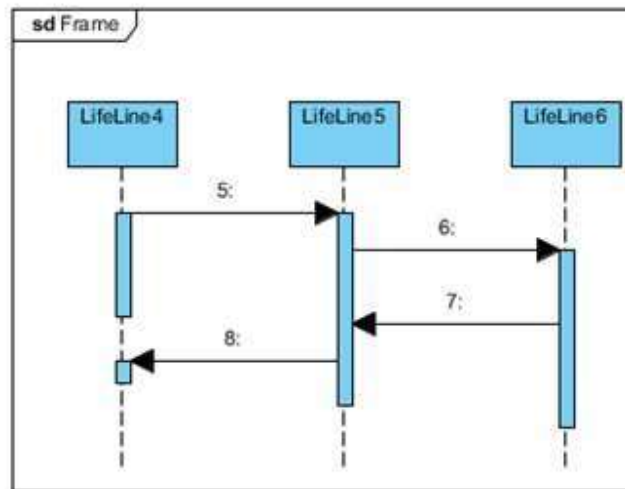
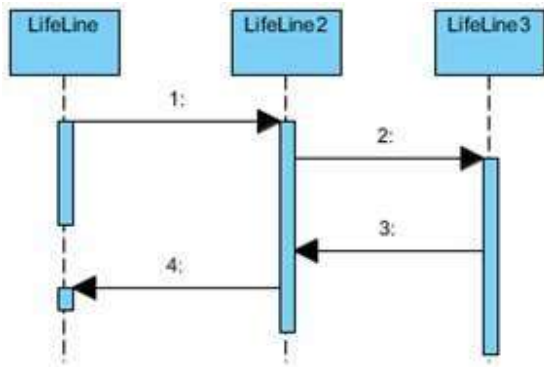
Diagram-based sequence message

Right click on the diagram's background, select **Sequence Number** and then either **Single Level** or **Nested Level** from the pop-up menu.

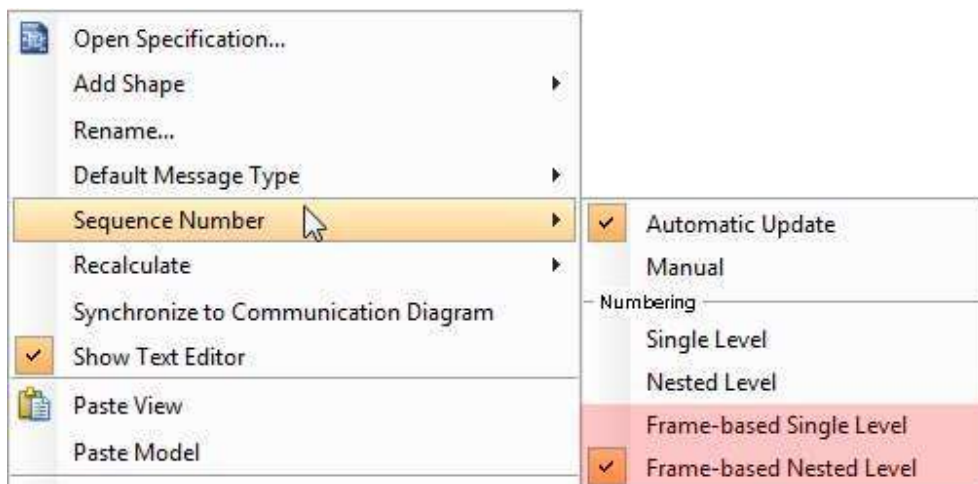


Step 11:-

If you choose **Single Level**, all sequence messages will be ordered with integers on diagram base. On the other hand, if you choose **Nested Level**, all sequence messages will be ordered with decimal place on diagram base.

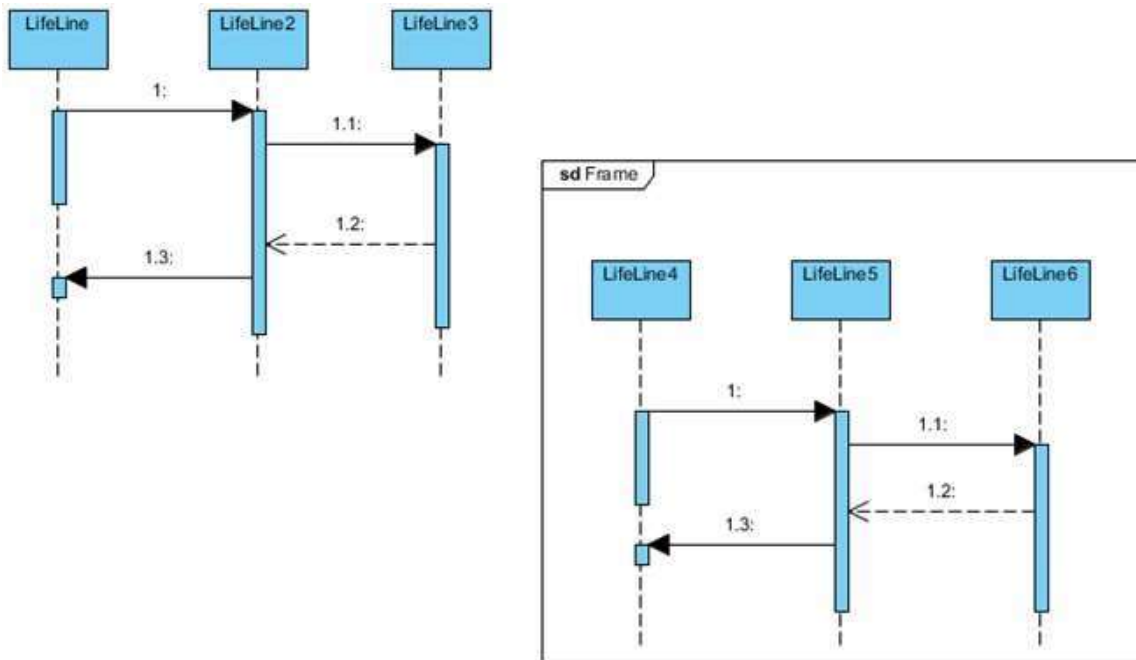


Right click on the diagram's background, select **Sequence Number** and then either **Frame-based Single Level** or **Frame-based Nested Level** from the pop-up menu.



When you set the way of numbering sequence messages on frame base, the sequence messages in frame will restart numbering sequence message since they

are independent and ignore the way of numbering sequence message outside the frame.



QUESTIONS

1. Draw the Sequence diagram of College Automation System.
2. What is the need of sequence diagram in a project?
3. What is the difference between nested level and single level sequence?
4. Draw the Sequence diagram of Banking Management system.

Experiment No. 6:

Develop Class diagram

Objective:-

To show diagrammatically the objects required and the relationships between them while developing a software product.

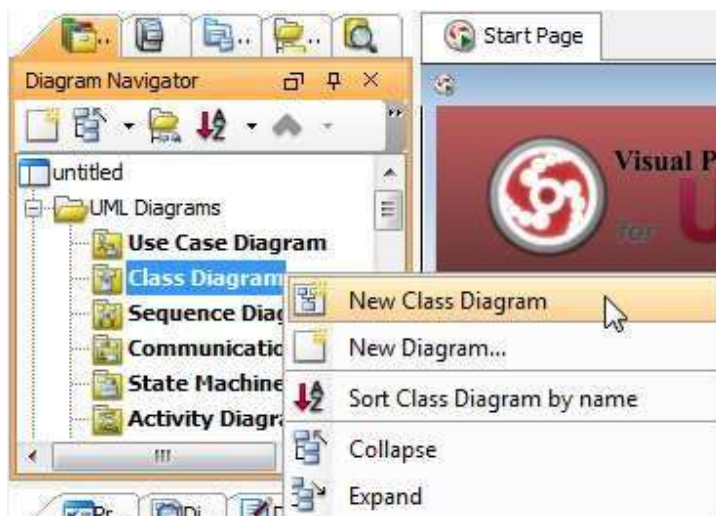
Software Required :-

Visual Paradigm for UML 8.2

Procedure :-

Step 1:-

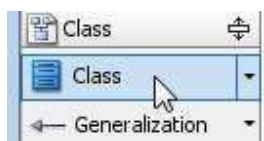
Right click **Class Diagram** on **Diagram Navigator** and select **New ClassDiagram** from the pop-up menu to create a class diagram.



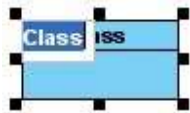
Step 2:-

Creating class

To create class, click **Class** on the diagram toolbar and then click on the diagram.

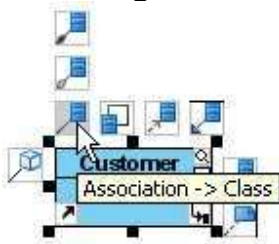


A class will be created.

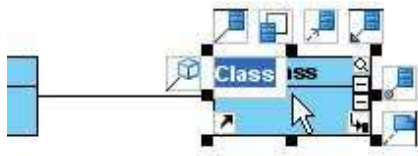


Creating association

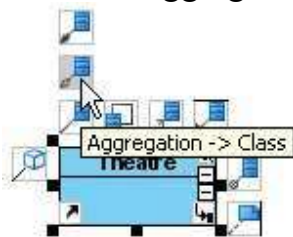
To create association from class, click the **Association -> Class** resource beside it and drag.



Drag to empty space of the diagram to create a new class, or drag to an existing class to connect to it. Release the mouse button to create the association.

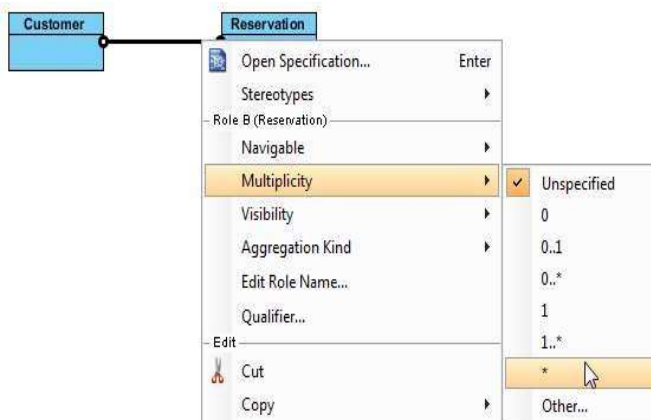


To create aggregation, use the **Aggregation -> Class** resource instead.

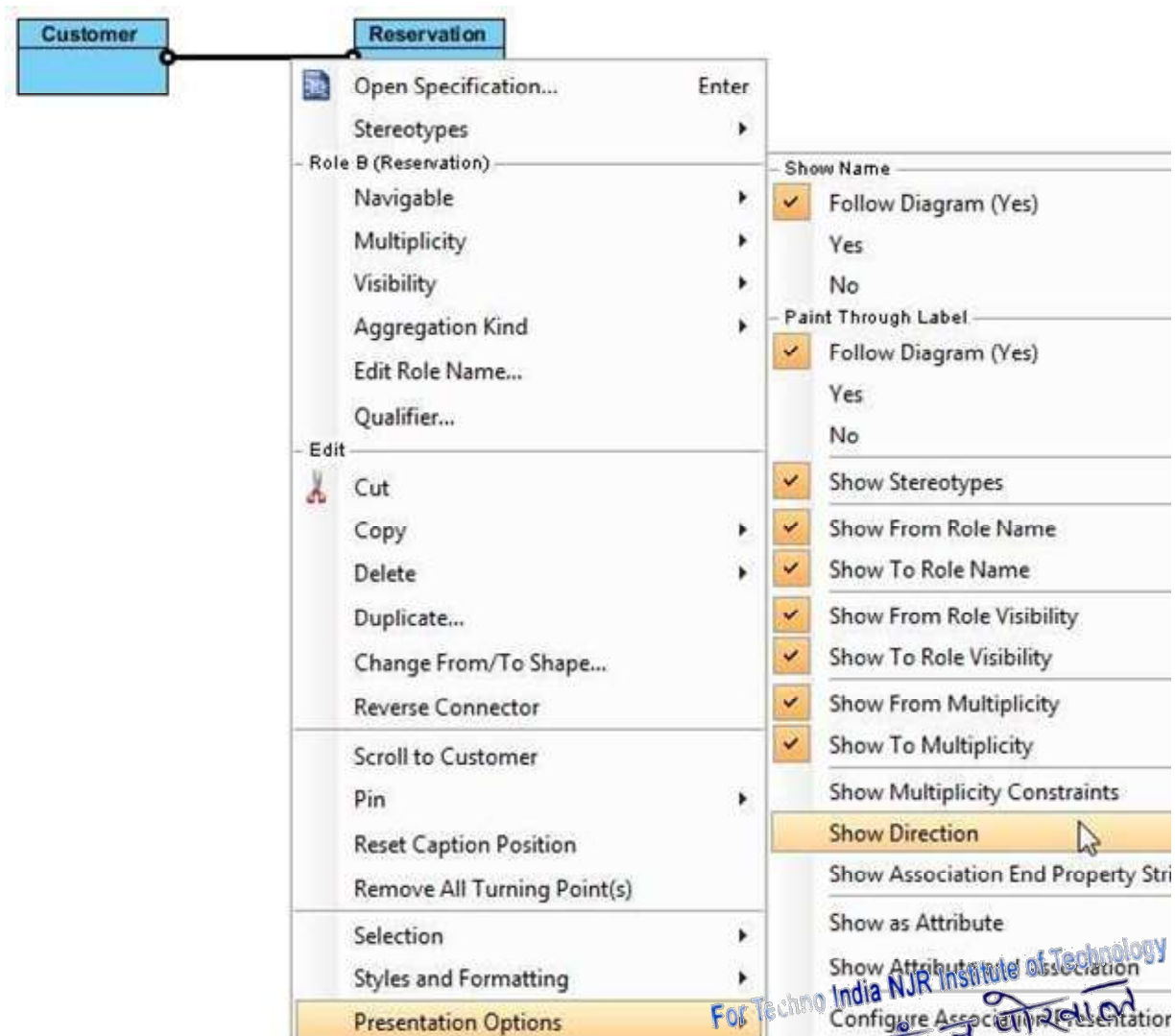


Step 3:-

To edit multiplicity of an association end, right-click near the association end, select **Multiplicity** from the popup menu and then select a multiplicity.

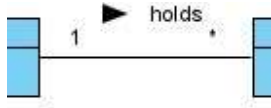


To show the direction of an association, right click on it and select **Presentation Options > Show Direction** from the pop-up menu.



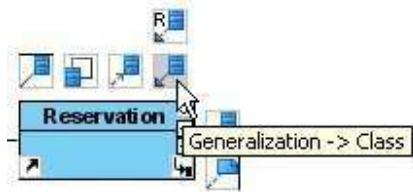
Step 4:-

The direction arrow is shown beside the association.

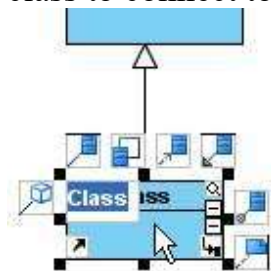


Creating generalization

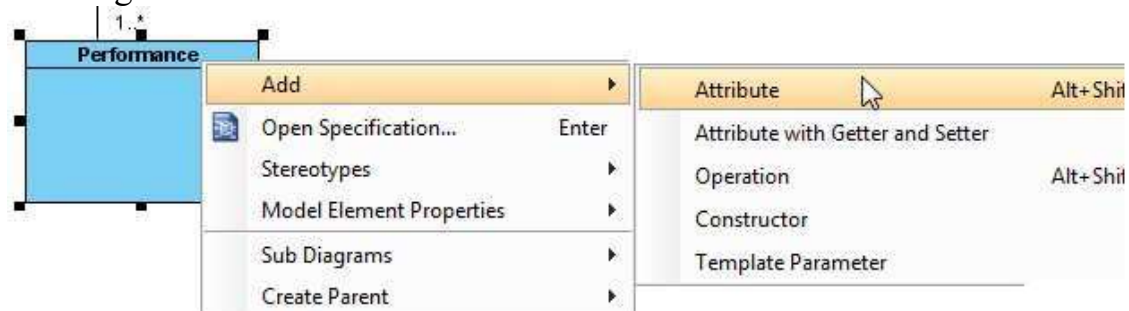
To create generalization from class, click the **Generalization -> Class** resource beside it and drag.



Drag to empty space of the diagram to create a new class, or drag to an existing class to connect to it. Release the mouse button to create the generalization.



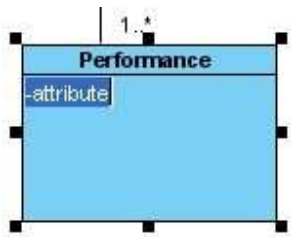
Creating attribute



An attribute is created.

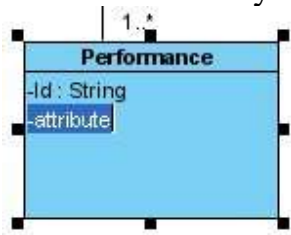
To create attribute, right click the class and select **Add > Attribute** from the pop-up

menu.



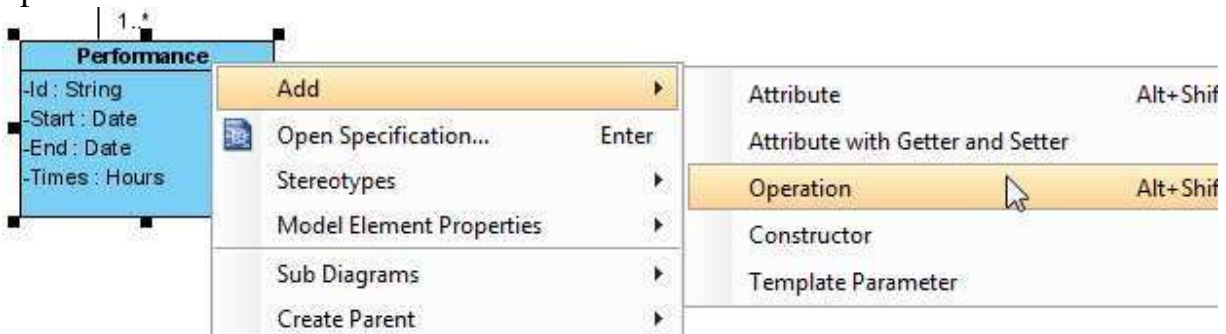
Creating attribute with enter key

After creating an attribute, press the Enter key, another attribute will be created. This method lets you create multiple attributes quickly and easily.

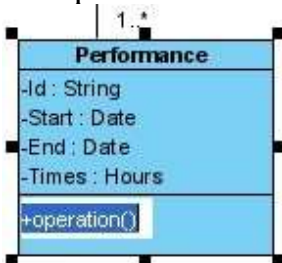


Creating operation

To create operation, right click the class and select **Add > Operation** from the pop-up menu.



An operation is created.



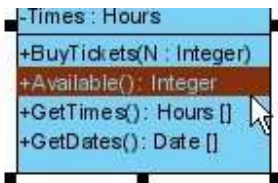
Similar to creating attribute, you can press the Enter key to create multiple operations continuously.

Drag-and-Drop reordering, copying and moving of class members

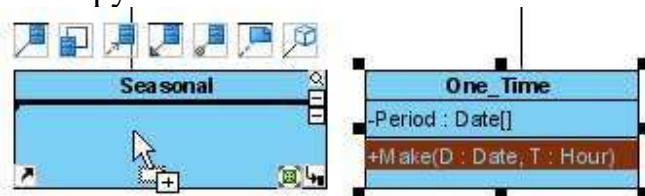
To reorder a class member, select it and drag within the compartment, you will see a thick black line appears indicating where the class member will be placed.



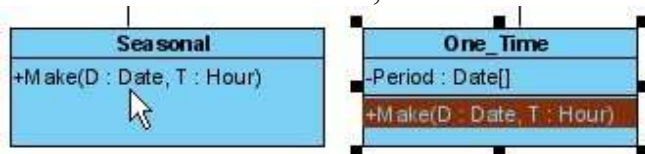
Release the mouse button, the class member will be reordered.



To copy a class member, select it and drag to the target class while keep pressing the Ctrl key, you will see a thick black line appears indicating where the class member will be placed. A plus sign is shown beside the mouse cursor indicating this is a copy action.



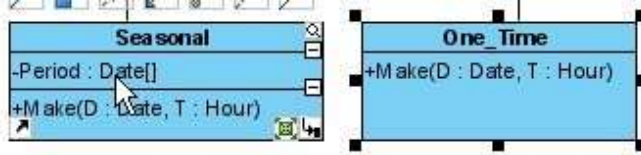
Release the mouse button, the class member will be copied.



To move a class member, select it and drag to the target class, you will see a thick black line appears indicating where the class member will be placed. Unlike copy, do not press the Ctrl key when drag, the mouse cursor without the plus sign indicates this is a move action.

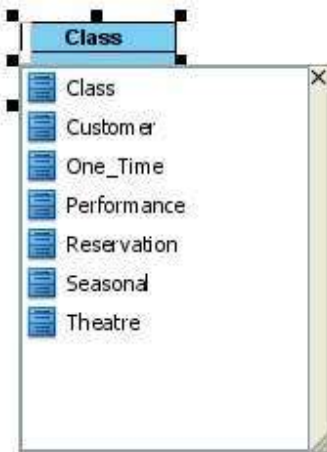


Release the mouse button, the class member will be moved.

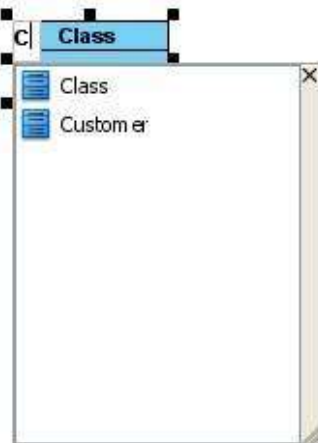


Model name completion for class

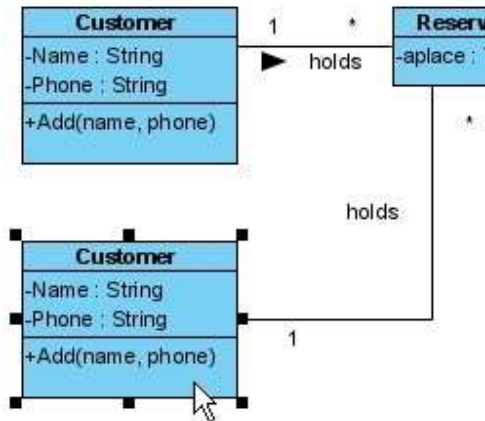
The model name completion feature enables quick creation of multiple views for the same class model. When create or rename class, the list of classes is shown.



Type text to filter classes in the list.

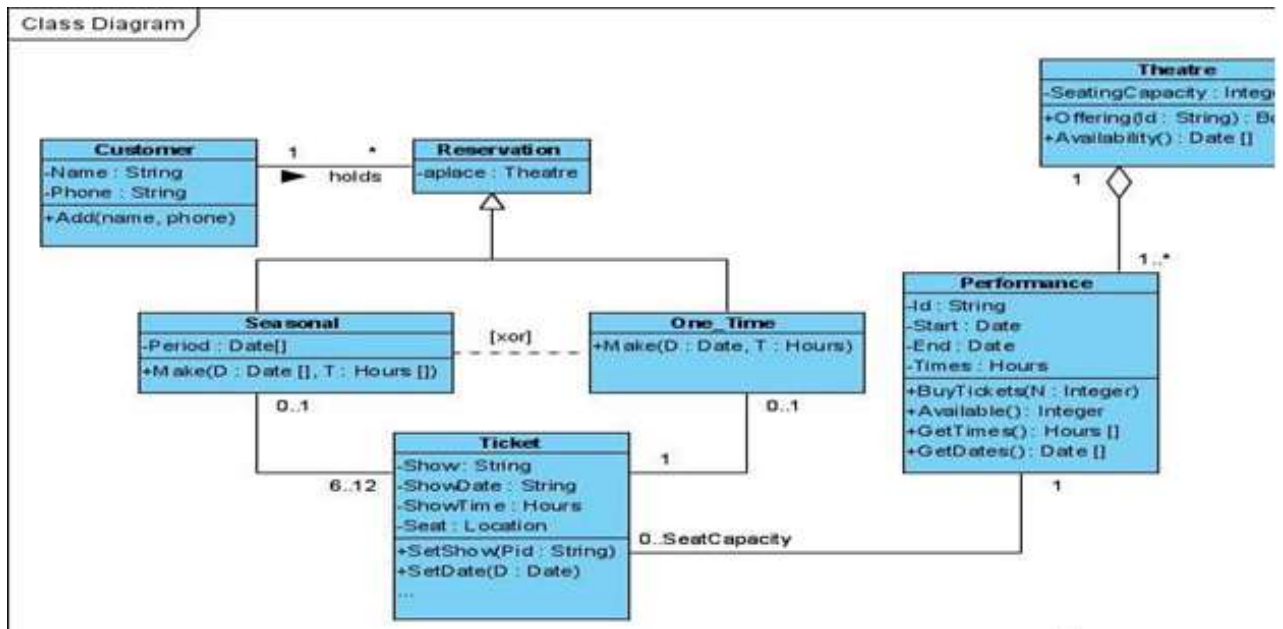


Press up or down key to select class in the list, press Enter to confirm. Upon selecting an existing class, all class members and relationships are shown immediately.



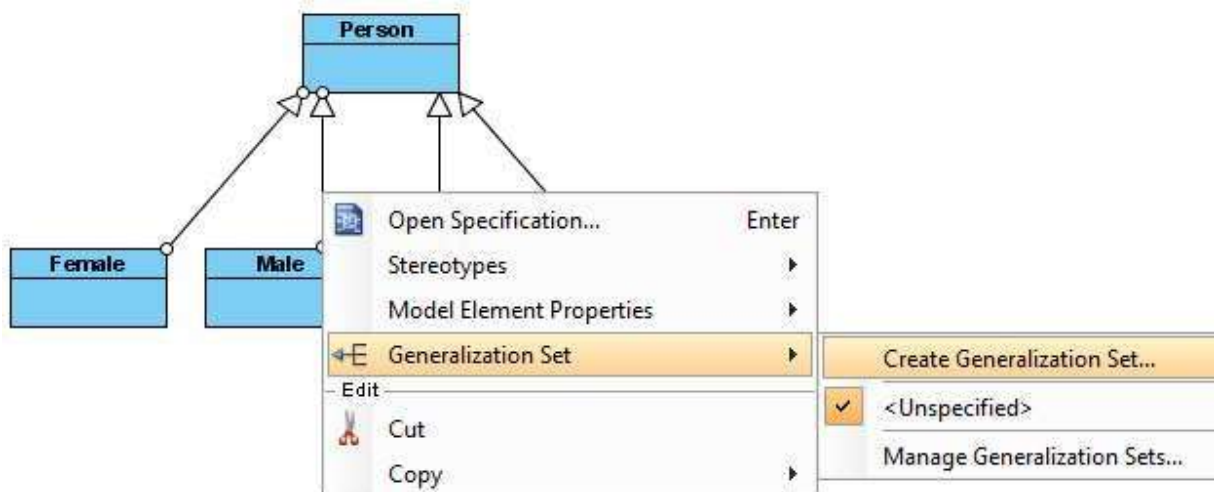
Step 5:-

Continue to complete the diagram.



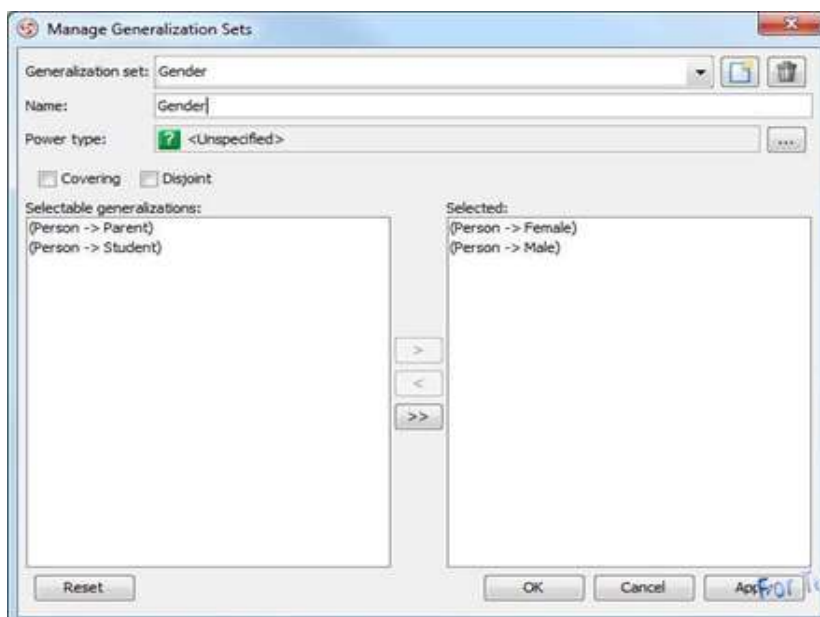
Generalization set

A generalization set defines a particular set of generalization relationships that describe the way in which a general classifier (or superclass) may be divided using specific subtypes. To define a generalization set, select the generalizations to include, right click and select Generalization set > Create Generalization Set... from the popup menu.

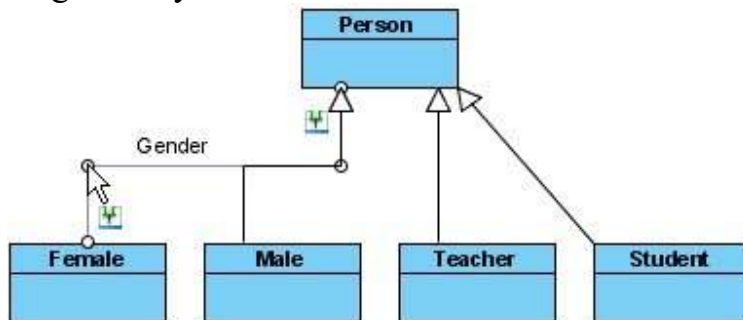


Step 6:-

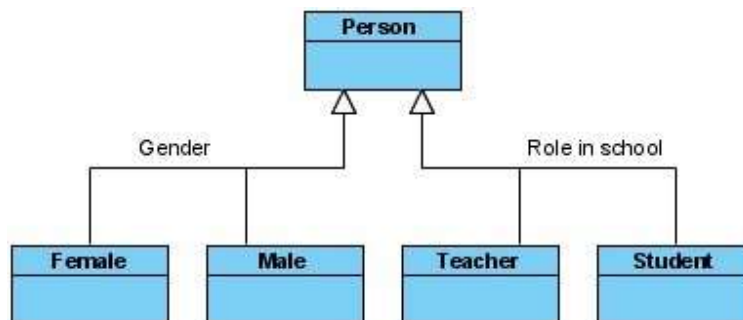
Name the set in the Manage Generalization Sets dialog box, and confirm by pressing OK.



The selected generalizations are grouped. Adjust the connector to make the diagram tidy.



Repeat the steps for other generalizations.



Questions :

1. Define Class.
2. What is the difference between Class diagram and UML.
3. What is dependency.
4. What is composition.
5. Define Recursive Association.

Lab Experiment No.7

Use testing tool such as Junit.

Testing is the process of checking the functionality of the application whether it is working as per requirements and to ensure that at developer level, unit testing comes into picture. Unit testing is the testing of single entity (class or method). Unit testing is very essential to every software company to give a quality product to their customers.

Unit testing can be done in two ways

Manual testing	Automated testing
<p>Executing the test cases manually without any tool support is known as manual testing.</p> <ul style="list-style-type: none">• Time consuming and tedious: Since test cases are executed by human resources so it is very slow and tedious.• Huge investment in human resources: As test cases need to be executed manually so more testers are required in manual testing.• Less reliable: Manual testing is less reliable as tests may not be performed with precision each time because of human errors.• Non-programmable: No programming can be done to write sophisticated tests which fetch hidden information.	<p>Taking tool support and executing the test cases by using automation tool is known as automation testing.</p> <ul style="list-style-type: none">• Fast Automation runs test cases significantly faster than human resources.• Less investment in human resources: Test cases are executed by using automation tool so less tester are required in automation testing.• More reliable: Automation tests perform precisely same operation each time they are run.• Programmable: Testers can program sophisticated tests to bring out hidden information.

What is JUnit ?

JUnit is a unit testing framework for the Java Programming Language. It is important in the test driven development, and is one of a family of unit testing frameworks collectively known as xUnit.

JUnit promotes the idea of "first testing then coding", which emphasis on setting up the test data for a piece of code which can be tested first and then can be implemented . This approach is like "test a little, code a little, test a little, code a little..." which increases programmer productivity and stability of program code that reduces programmer stress and the time spent on debugging.

Features

- JUnit is an open source framework which is used for writing & running tests.
- Provides Annotation to identify the test methods.
- Provides Assertions for testing expected results.
- Provides Test runners for running tests.
- JUnit tests allow you to write code faster which increasing quality
- JUnit is elegantly simple. It is less complex & takes less time.
- JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
- JUnit tests can be organized into test suites containing test cases and even other test suites.
- Junit shows test progress in a bar that is green if test is going fine and it turns red when a test fails

What is a Unit Test Case ?

A Unit Test Case is a part of code which ensures that the another part of code (method) works as expected. To achieve those desired results quickly, test framework is required .JUnit is perfect unit test framework for java programming language.

A formal written unit test case is characterized by a known input and by an expected output, which is worked out before the test is executed. The known input should test a precondition and the expected output should test a postcondition.

There must be at least two unit test cases for each requirement: one positive test and one negative test. If a requirement has sub-requirements, each sub-requirement must have at least two test cases as positive and negative.

Online

You really do not need to set up your own environment to start learning Java &JUnit programming language. Reason is very simple, we already have setup Java Programming environment online, so that you can compile and execute all the available examples online at the same time when you are doing your theory work. This gives you confidence in what you are reading and to check the result with different options. Feel free to modify any example and execute it online.

Try following example using **Try it** option available at the top right corner of the below sample code box:

```
public class MyFirstJavaProgram {  
  
    public static void main(String []args)  
    {System.out.println("Hello World");  
    }  
}
```

For most of the examples given in this tutorial, you will find **Try it** option, so just make use of it and enjoy your learning.

Local Environment Setup

JUnit is a framework for Java, so the very first requirement is to have JDK installed in your machine.

System Requirement

- JDK** 1.5 or above.
- Memory** no minimum requirement.
- Disk Space** no minimum requirement.
- Operating System** no minimum requirement.

Step 1 - verify Java installation in your machine

Now open console and execute the following **java** command.

OS	Task	Command
Windows	Open Command Console	c:\> java -version
Linux	Open Command Terminal	\$ java -version
Mac	Open Terminal	machine:~ joseph\$ java -version

Let's verify the output for all the operating systems:

OS	Output
Windows	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Linux	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Mac	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM)64-Bit Server VM (build 17.0-b17, mixed mode, sharing)

Step 2: Set JAVA environment

Set the **JAVA_HOME** environment variable to point to the base directory location where Java is installed on your machine. For example

Windows	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.6.0_21
Linux	export JAVA_HOME =/usr/local/java-current
Mac	export JAVA_HOME =/Library/Java/Home

Append Java compiler location to System Path.

OS	Output
Windows	Append the string ;C:\Program Files\Java\jdk1.6.0_21\bin to the end of the system variable, Path.
Linux	export PATH =\$PATH:\$ JAVA_HOME /bin/
Mac	not required

Verify Java Installation using **java -version** command explained above.

Step 3: Download Junit archive

Download latest version of JUnit jar file from <http://www.junit.org>. At the time of writing this tutorial, I downloaded *Junit-4.10.jar* and copied it into C:\>JUnit folder.

OS	Archive name
Windows	junit4.10.jar
Linux	junit4.10.jar
Mac	junit4.10.jar

Step 4: Set JUnit environment

Set the **JUNIT_HOME** environment variable to point to the base directory location where JUNIT jar is stored on your machine. Assuming, we've stored junit4.10.jar in JUNIT folder on various Operating Systems as follows.

OS	Output
Windows	Set the environment variable JUNIT_HOME to C:\JUNIT
Linux	export JUNIT_HOME=/usr/local/JUNIT
Mac	export JUNIT_HOME=/Library/JUNIT

Step 5: Set CLASSPATH variable

Set the **CLASSPATH** environment variable to point to the JUNIT jar location. Assuming, we've stored junit4.10.jar in JUNIT folder on various Operating Systems as follows.

OS	Output
Windows	Set the environment variable CLASSPATH to %CLASSPATH%;%JUNIT_HOME%\junit4.10.jar;.
Linux	export CLASSPATH=\$CLASSPATH:\$JUNIT_HOME/junit4.10.jar;.
Mac	export CLASSPATH=\$CLASSPATH:\$JUNIT_HOME/junit4.10.jar;.

Step 6: Test JUnit Setup

Create a java class file name Test unit in **C:\ > JUNIT_WORKSPACE**

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;
```

```
public class TestJUnit {  
    @Test  
    public void testAdd() {
```

```
String str= "JUnit is working fine";
assertEquals("JUnit is working
fine",str);
}
}
```

Create a java class file name TestRunner in **C:\ > JUNIT_WORKSPACE** to executeTest case(s)

```
importorg.junit.runner.JUnitCore;
importorg.junit.runner.Result;
importorg.junit.runner.notification.Failure;

public class TestRunner {
public static void main(String[] args) {
    Result result =
JUnitCore.runClasses(TestJunit.class);for (Failure
failure : result.getFailures()) {
System.out.println(failure.toString());
    }
System.out.println(result.wasSuccessful());
}
}
```

Step 7: Verify the Result

Compile the classes using **javac** compiler as follows

```
C:\JUNIT_WORKSPACE>javac TestJunit.java
TestRunner.java
```

Now run the Test Runner to see the result

```
C:\JUNIT_WORKSPACE>java Test Runner
```

Verify the output.

Experiment No-8

Using configuration management tool-libra

Installation and Use

The Libra features can be installed from the p2 repository of the Indigo Simultaneous Release (since Indigo M6). As a prerequisite you may install Eclipse IDE for Java EE Developers.

The update site contains:

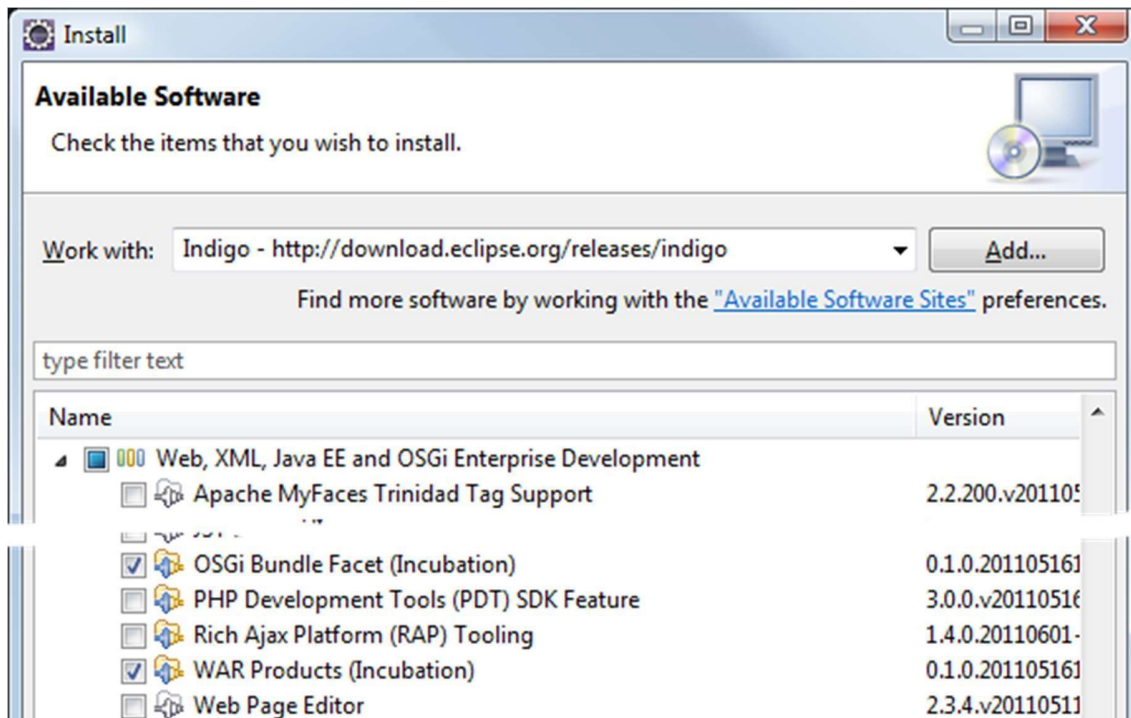
- **OSGi Bundle Facet** feature that introduces:
 1. A new facet **OSGi Bundle** for Dynamic Web, JPA and Utility projects.
 2. Wizard for converting WTP standard projects to OSGi Enterprise bundle projects:
 - Dynamic Web projects to Web Application Bundle projects
 - JPA projects to Persistent Bundle projects
 - Utility projects and simple Java projects to OSGi Bundle projects

Both options modify project's **MANIFEST.MF** in order to become a valid OSGi bundle.

The facet may be enabled during the project creation or after that from the **Properties** page of the project. The wizard is available from project's context menu **Configure > Convert to OSGi Bundle Projects...**

Note that you may need to adjust your target platform accordingly.

- **WAR Products** feature which provides WAR deployment for Equinox based applications



Create new Web Application Bundle

1. Call the New Dynamic Web Project wizard: *New > Project... > Web > DynamicWeb Project*
2. Enter the necessary project information like *Project name*, *Target runtime*, etc.
3. Add the *OSGi Bundle* facet in the Configuration:
 1. Click on the *Modify...* button in the *Configuration* group.
 2. Choose the *OSGi Bundle* facet in the *Project Facets* dialog and click OK.
4. Click *Finish* to create the Web Application Bundle project.

Create new OSGi Bundle

1. Call the New Faceted Project wizard: *New > Project... > General > FacetedProject*
2. Enter the necessary project information like *Project name*.
3. Click the *Next* button.
4. Select the *OSGi Bundle* and *Java* facets.

5. Click *Finish* to create the OSGi Bundle project.

Obtaining Sources

You can find the sources available in [Git repository](#)

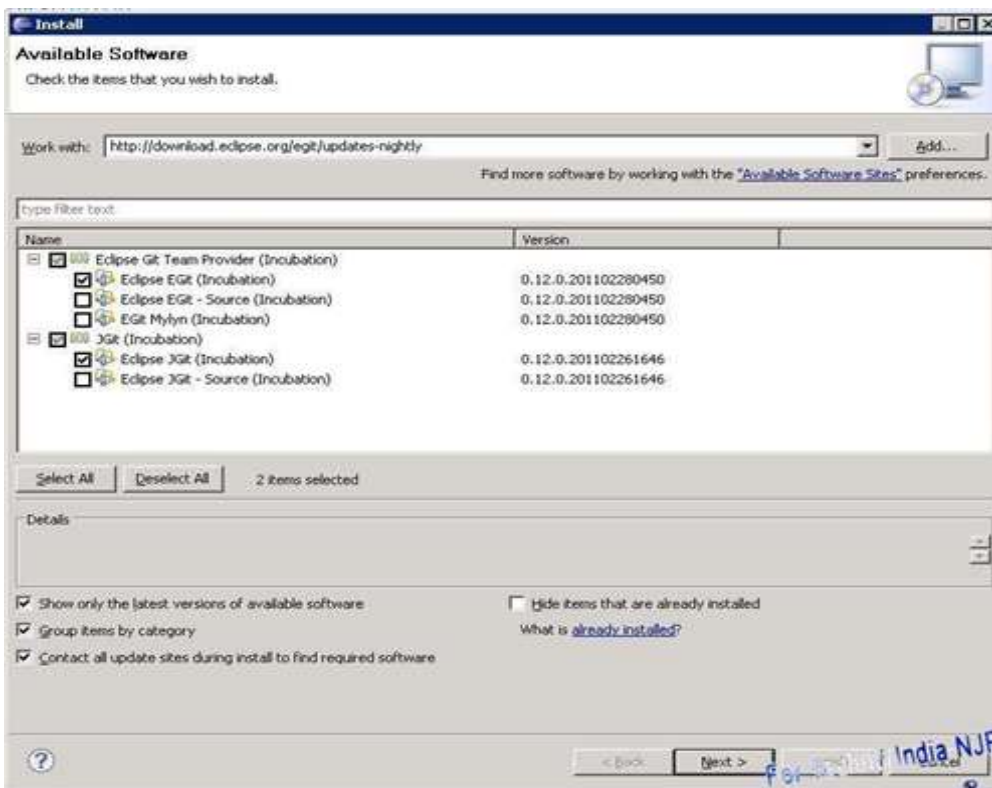
In order to synchronize them locally, you may use the [EGit](#) step-by-step procedure.

The [EGit/User Guide](#) provides detailed instruction how to work with EGit.

Updating/Installing EGit

- Start your **Eclipse IDE** and navigate to **Help->Install New Software->Add...**
- Enter the software update site [\[1\]](#)
- Select the **Eclipse EGit (Incubation)** and **Eclipse JGit (Incubation)** and choose **Next>** to finish the installation.

During the installation you will be asked to accept the License Agreements.



Identifying Yourself

To identify yourself, follow these instructions [these instructions](#)

Setting up the Home Directory on Windows

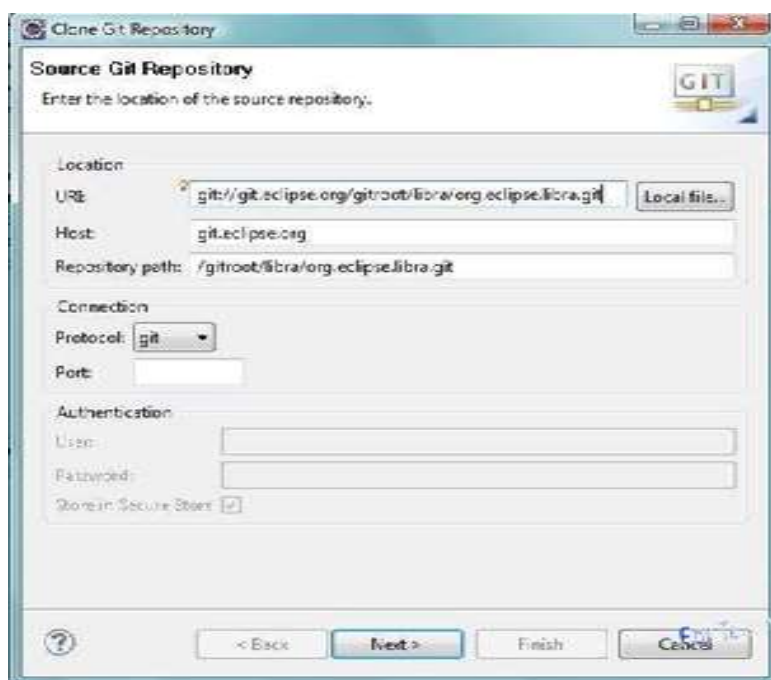
To set up the **HOME** directory, go through [these steps](#)

Configuring SSH in Eclipse

To configure ssh, proceed [as follows](#).

Clone Git Repository

- Open the **Git Repository Exploring** perspective and in the **Git Repositories** view choose the **Clone a Git Repository** toolbar button
- In the **URI** field of the opened **Clone Git Repository** wizard enter the URI of the libra git repository: `[2]` and choose **Next**



- Select the added repository and from its context menu choose **Import Projects...**
- Expand the repository tree to **Remote Tracking** level, select the remote branch **origin/master** and from its context menu choose **Create Branch...** to create a new local branch

Updating Sources

To keep the sources up to date you have to pull the new changes from the upstream branch.

Build Infrastructure

The build is based on Maven (at least 3.0.0) and Tycho, executed on the Hudson server, hosted at Eclipse Foundation.

There are two Hudson jobs available for Libra:

- libra - for building the **master** git branch.
- libra-indigo - for building the **indigo** git branch.

Maven Build Sequence

Complete build sequence for a clean build (assuming \$M2_HOME/bin is on the path and local Maven repository at ~/.m2/repository):

```
[~/org.eclipse.libra/development/org.eclipse.libra.releng] $ mvn clean install
```

Note that you may need to configure your proxy settings

Proposing and Committing a Patch

The patch file contains a description of changes of a set of resources which can be automatically applied to another eclipse workspace or git repository. If you want to propose or commit a patch you need to know that the Eclipse update hook will examine the Committer's entries of an incoming push. All the committer's entries have to be made by the committer performing the push, otherwise the push will fail. Furthermore, your committer ID, or the committer e-mail address registered

with your committer account at the Eclipse Foundation must be present in the Committer Email record. For more information on that restriction see: [this](#) page.

Proposing a Patch

If you want commit a change on a local feature or bugfix branch and then to export this change into a patch file, follow the steps below:

1. Open a bug
 - Specify the bug component (General; OSGi Facet or WAR)
 - Complete the fields about the product version, bug severity, type of your hardware, operating systems and write a summary and description of the patch
2. Open the **History view** of your **Eclipse IDE** and choose **Create Patch...** (*The patch file will contain the difference between the commit and its parent in the history view. Note that the filter of the history view applies also for patchcreation.*)
3. Start the **Patch Wizard**, select the location of the patch and choose **Next** (*The name of the patch file is created from the first line of the commit message.*)
4. Change the patch format, if necessary.
5. Attach the patch created to the bug
6. Submit the bug

Committing a Patch

To commit a patch, proceed as follows:

1. Apply the proposed patch using the **Patch Wizard**
2. Test the patch
3. Commit the patch
4. Setup push configuration with the following push URL:

```
ssh://committer_id@git.eclipse.org/gitroot/libra/org.eclipse.libra.git
```
5. Push the patch and see its change number in order to be able to inspect it

Reproducing a Build to a Certain Change List

Fetch to the specific change and build with Maven as described above.

Viva voce questions:

1. What are the important categories of software?
2. What is the main difference between a computer program and computer software?
3. What is software re-engineering?
4. Describe the software development process
5. What are SDLC models available?
6. What is software engineering?
7. What are various phases of SDLC?
8. What is data flow diagram ?
9. Who is software project manager?
10. What is verification and validation?
11. What is mean by level-0 Data flow diagram?
12. What is cohesion?
13. What is UML diagram?
14. What is SRS?

MCQ Questions:

1. Software is defined as _____
- a) set of programs, documentation & configuration of data
 - b) set of programs
 - c) documentation and configuration of data
 - d) None of the mentioned

Answer: a

2. What is Software Engineering?
- a) Designing a software
 - b) Testing a software

- c) Application of engineering principles to the design a software
- d) None of the above

Answer: c

3. Who is the father of Software Engineering?

- a) Margaret Hamilton
- b) Watts S. Humphrey
- c) Alan Turing
- d) Boris Beizer

Answer: b

4. What are the features of Software Code?

- a) Simplicity
- b) Accessibility
- c) Modularity
- d) All of the above

Answer: c

5. _____ is a software development activity that is not a part of software processes.

- a) Validation
- b) Specification
- c) Development
- d) Dependence

Answer: d

6. Define Agile scrum methodology.

- a) project management that emphasizes incremental progress
- b) project management that emphasizes decremental progress
- c) project management that emphasizes neutral progress
- d) project management that emphasizes no progress

Answer: a

7. CASE stands for

- a) Computer-Aided Software Engineering
- b) Control Aided Science and Engineering
- c) Cost Aided System Experiments
- d) None of the mentioned

Answer: a

8. _____ is defined as the process of generating analysis and designing documents?

- a) Re-engineering
- b) Reverse engineering
- c) Software re-engineering
- d) Science and engineering

Answer: b

9. The activity that distributes estimated effort across the planned project duration by allocating the effort to specific software developing tasks is _____

- a) Project scheduling
- b) Detailed schedule
- c) Macroscopic schedule
- d) None of the mentioned

Answer: a

10. What is a Functional Requirement?

- a) specifies the tasks the program must complete
- b) specifies the tasks the program should not complete
- c) specifies the tasks the program must not work
- d) All of the mentioned

Answer: a

11. Why do bugs and failures occur in software?

- a) Because of Developers
- b) Because of companies
- c) Because of both companies and Developers
- d) None of the mentioned

Answer: c

12. Attributes of good software is _____

- a) Development
- b) Maintainability & functionality
- c) Functionality
- d) Maintainability

Answer: b

13. The Cleanroom philosophy was proposed by _____

- a) Linger
- b) Mills
- c) Dyer
- d) All of the Mentioned

Answer: d

14. What does SDLC stands for?

- a) System Design Life Cycle
- b) Software Design Life Cycle
- c) Software Development Life Cycle
- d) System Development Life cycle

Answer: c

15. Who proposed the spiral model?

- a) Barry Boehm
- b) Pressman
- c) Royce
- d) IBM

Answer: a

16. _____ is not among the eight principles followed by the Software Code of Ethics and Professional Practice.

- a) PRODUCT
- b) ENVIRONMENT
- c) PUBLIC
- d) PROFESSION

Answer: b

17. Which of the following are CASE tools?

- a) Central Repository
- b) Integrated Case Tools
- c) Upper Case Tools
- d) All of the mentioned

Answer: d

18. _____ suits the Manifesto for Agile Software Development.

- a) Customer collaboration
- b) Individuals and interactions
- c) Working software
- d) All of the mentioned

Answer: d

19. Software patch is defined as _____

- a) Daily or routine Fix
- b) Required or Critical Fix
- c) Emergency Fix
- d) None of the mentioned

Answer: c

20. _____ software development team has no permanent leader.

- a) Controlled Centralized (CC)
- b) Controlled decentralized (CD)
- c) Democratic decentralized (DD)
- d) None of the mentioned

Answer: c

----- Good Luck -----