INTRODUCTION

Course Title	Software Design and Architecture		
Course Code	SL2002		
Credit Hours	4(3+1)		
Semester	Spring-2022		
Lab. Instructor	Engr. Khuram Shahzad	**Course Instructor**	Dr. M. Taimoor Khan

List of Experiments

Lab#	Title of Experiment and outcomes	CLO Mapping of Each Experiment
01	**Introduction to UML and Introduction of Visual Paradigm:** After this lab students will be able to understand: • What is UML • Installation of Visual Paradigm • How to create a new design model from scratch	**CLO-4**
02	**Working with Use Case Models (Simple)** After this lab student will be able to: • Familiarize with the concepts underlining Use Case Model for Requirement Engineering • Model Use Case Model and its application • Familiarize with the concepts of Extend and Include relationship among use cases	**CLO-4**

03	**Working with Domain Models** After this lab student will be able to: • Familiarize with the concepts underlining Domain Model for Requirement Analysis • How to model Domain diagram and its application	**CLO-4**
04	**Working with Sequence Models** After this lab student will be able to: • Familiarize with the concepts underlining Sequence Model • How to model Sequence diagram and its application	**CLO-4**
Lab Quiz No. 1 (at the start of 4th week)		**CLO-4**
05	**Working with Class Models** After this lab student will be able to: • Familiarize with the concepts underlining Class Model • How to model Class diagram and its application	**CLO-4**
06	**Working with Activity Models** After this lab student will be able to: • Familiarize with the concepts underlining Activity Model • How to model Activity diagram and its application **Working with State Machine Models** After this lab student will be able to: • Familiarize with the concepts underlining State Machine Model • How to model State Machine diagram and its application	**CLO-4**
Lab Quiz No 2 (at the start of 6th Week)		**CLO-4**

07	**Working with Deployment Models** After this lab student will be able to: • Familiarize with the concepts underlining Deployment Model • How to model Deployment diagram and its application **Working with Component Models** After this lab student will be able to: • Familiarize with the concepts underlining Component Model • How to model Component diagram and its application	**CLO-4**
08	**Working with Communication Models** After this lab student will be able to: Familiarize with the concepts underlining Communication Model How to model Communication diagram and its application	**CLO-4**
Lab Quiz No. 3 (at the start of 8th week)		**CLO-4**
09	**Working with Package Models** After this lab student will be able to: • Familiarize with the concepts underlining Package Model • How to model Package diagram and its application	**CLO-4**
	Working with Data Flow Models After this lab student will be able to: • Familiarize with the concepts underlining Data Flow Model • How to model Data Flow diagram and its application **Working with ER Models** After this lab student will be able to: • Familiarize with the concepts underlining ER Model • How to model ER diagram and its application	**CLO-4**

10	**Working with Object diagram** After this lab student will be able to: • Familiarize with the concepts underlining Object diagram • How to model Object diagram and its application **Working with Composite Structure Diagram** After this lab student will be able to: • Familiarize with the concepts underlining Composite Structure Diagram • How to model Composite Structure Diagram and its application	**CLO-4**
11	**Working with Timing Diagram** After this lab student will be able to: • Familiarize with the concepts Timing Diagram • How to model Timing Diagram and its application **Working with Interaction overview Diagram** After this lab student will be able to: • Familiarize with the concepts Interaction overview Diagram • How to model Interaction overview Diagram and its application	**CLO-4**
L ab Quiz No.4 (at the start of 12th week)		**CLO-4**
12	**Working with Proxy Design Patterns** After this lab student will be able to: • Familiarize with the concepts underlining Proxy Design Patterns • How to model Proxy Design Patterns and its application	**CLO-4**

13	**Working with Prototype Design Patterns** After this lab student will be able to: • Familiarize with the concepts underlining Prototype Design Patterns • How to model Prototype Design Patterns and its application	**CLO-4**
L ab Quiz No. 5 (At the start of 14th week)		**CLO-4**
14	**Working with Observer Design Patterns** After this lab student will be able to: • Familiarize with the concepts underlining Observer Design Patterns How to model Observer Design Patterns and its application	**CLO-4**
15	**Working with Chain Design Patterns** After this lab student will be able to: • Familiarize with the concepts underlining Chain Design Patterns How to model Chain Design Patterns and its application	**CLO-4**
16	**Viva of Project / Assignment**	**CLO-4**

Lab Engineer **Course Instructor**

Table of Contents

LAB-01

OBJECTIVE

After this lab students will be able to understand

- The basics of UML
- Introduction of Visual Paradigm
- Installation of Visual Paradigm

PRE-LAB READING ASSIGNMENT

NONE

LAB RELATED CONTENT

Introduction to UML and Introduction of Visual Paradigm

- What is UML
- Installation of Visual Paradigm

1. What is UML

The **Unified Modeling Language** (**UML**) is a general-purpose, developmental, modeling language in the field of software engineering, which is intended to provide a standard way to visualize the design of a system.

1.1. Design

UML offers a way to visualize a system's architectural blueprints in a diagram, including elements such as:

- Any activities (jobs);
- Individual components of the system;
- And how they can interact with other software components;
- How the system will run;
- How entities interact with others (components and interfaces); External user interface.

Although originally intended for object-oriented design documentation, UML has been extended to a larger set of design documentation (as listed above), and been found useful in many contexts.

1.2. Diagrams

UML has many types of diagrams, which are divided into two categories. Some types represent structural information, and the rest represent general types of behavior, including a few that represent different aspects of interactions. These diagrams can be categorized hierarchically as shown in the figure 1.

1.3. Where can UML be used?

1. Flow/control of processes/activities using Activity Diagram
2. Requirements Capture using Use Case Diagram
3. Requirements Analysis using Use Case Details and Class Diagram
4. Initial Design using Sequence Diagrams and second version of Class Diagram
5. System Architecture using Deployment Diagram
6. Design using Design Patterns
7. Detailed design using Collaboration Diagram
8. Refine all models through iterations
9. Implement the models by translating into code
10. Deploy software within operational environment

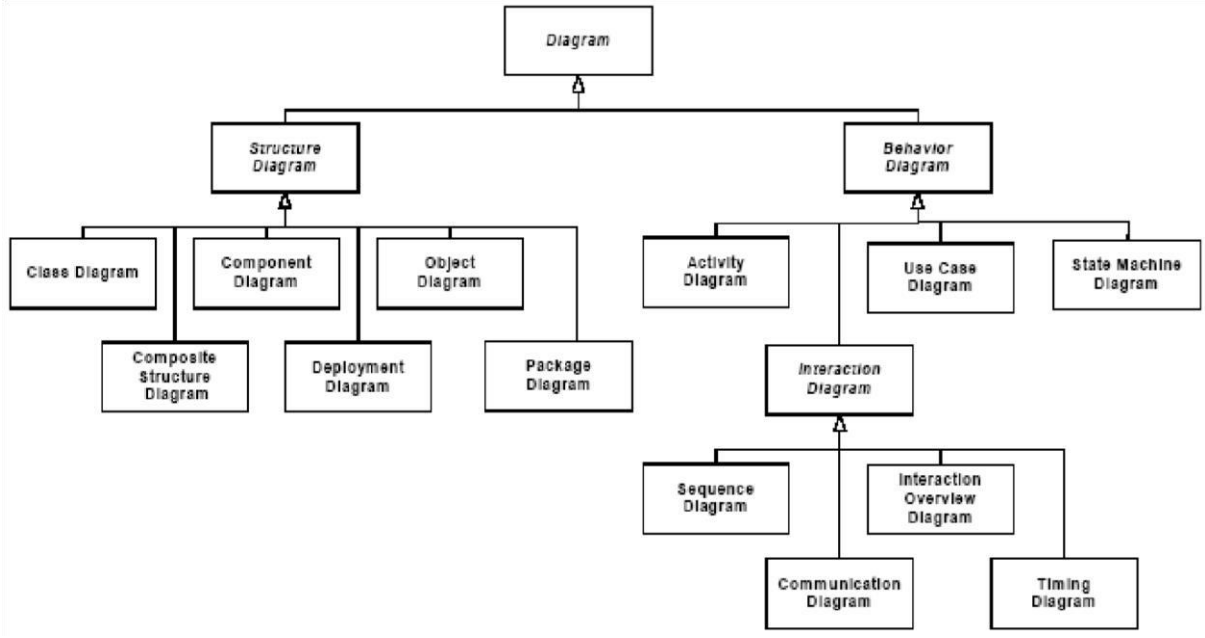


Figure 1: UML Diagrams Hierarchy

2. Installation of Visual Paradigm

Having downloaded the installer of Visual Paradigm, execute it, run through the installation to install Visual Paradigm. If you are using the InstallFree version, you just need to unzip it and run Visual Paradigm directly. In this lab, we will go through the installation of Visual Paradigm both with installer (.exe) and InstallFree (.zip).

Step-1

Execute the downloaded Visual Paradigm installer file. The setup wizard appears as below.

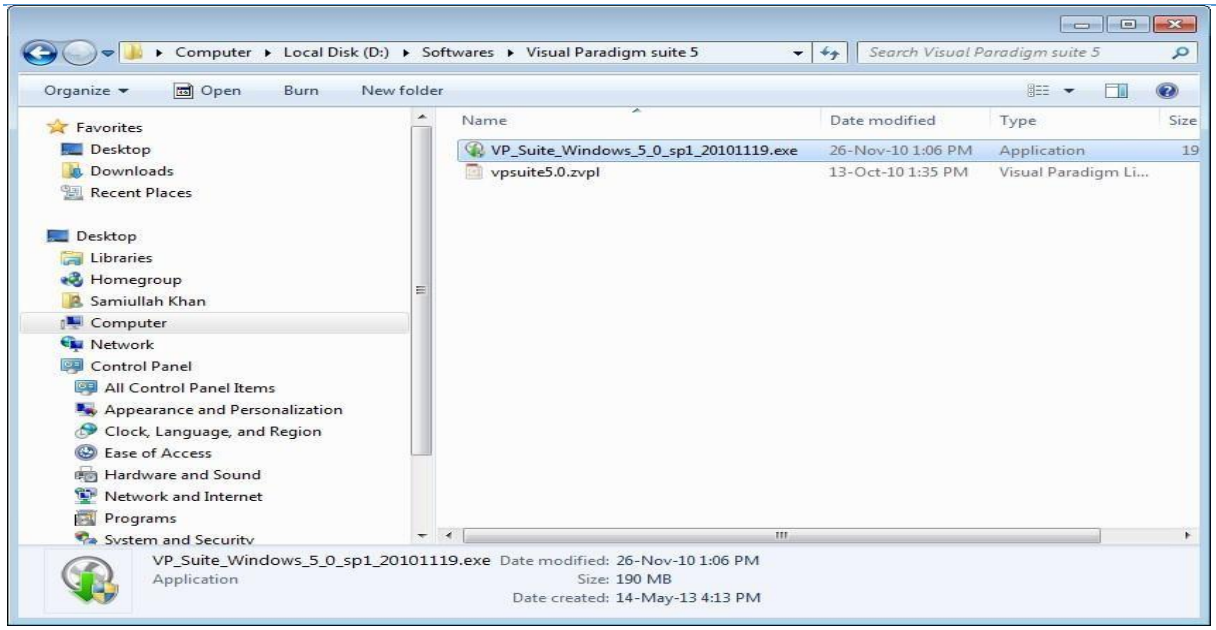


Figure 2: Visual Paradigm installer file

Step-2

On double clicking ".exe" file, installation wizard start like shown below in figure 3.

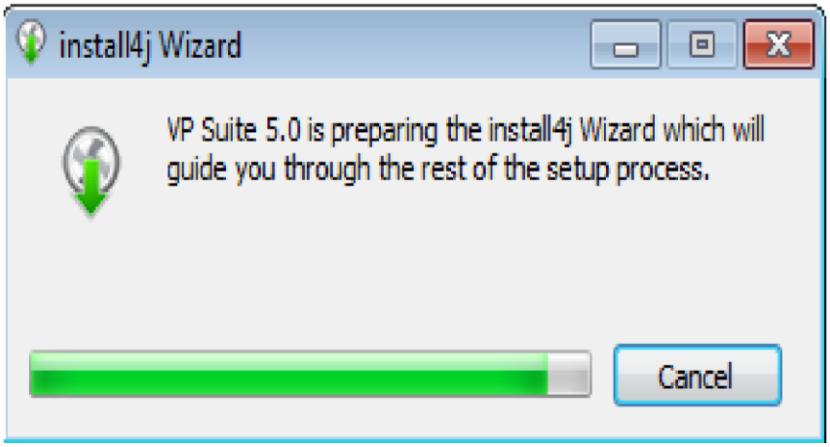


Figure 3: Visual Paradigm installation wizard

Step-3

Click on the "Next" button.

Figure 4:

Visual Paradigm Welcome Screen

Step-4

Accept the License Agreement shown in Figure

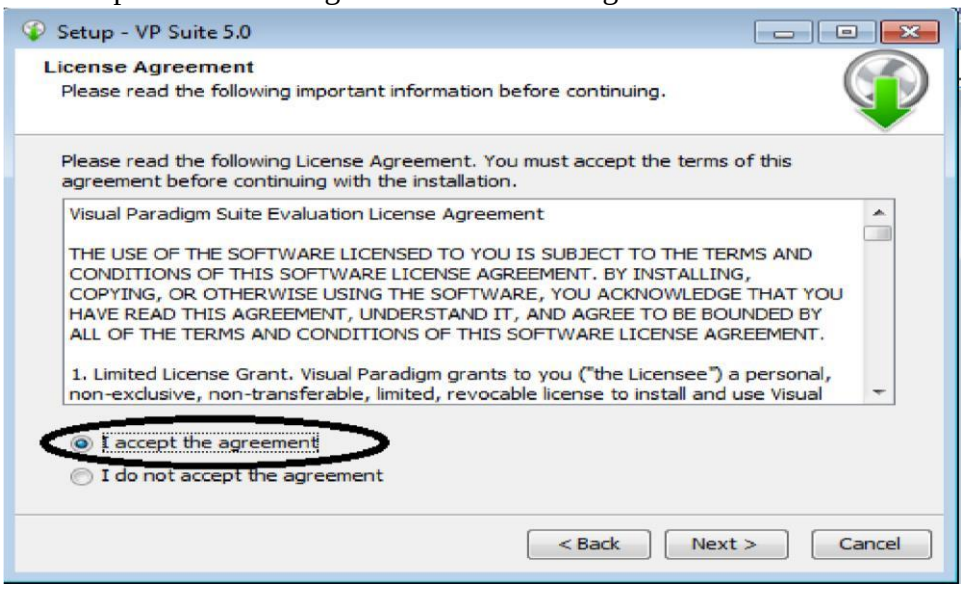


Figure 5: Accept the License Agreement

Step-5

Select Destination directory.

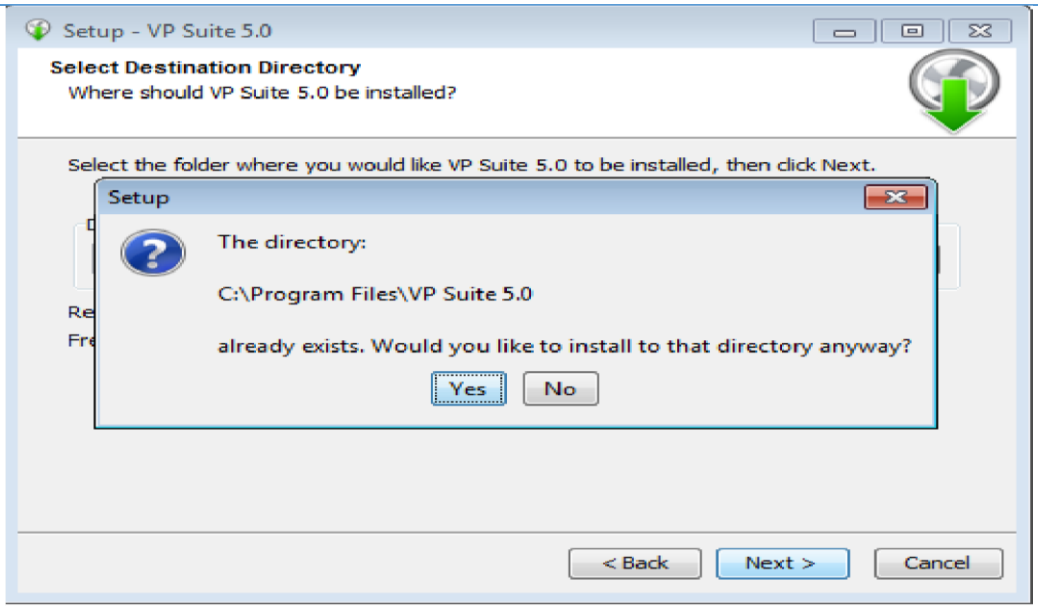


Figure 6: Selecting Destination directory

Step-6

Select file association.

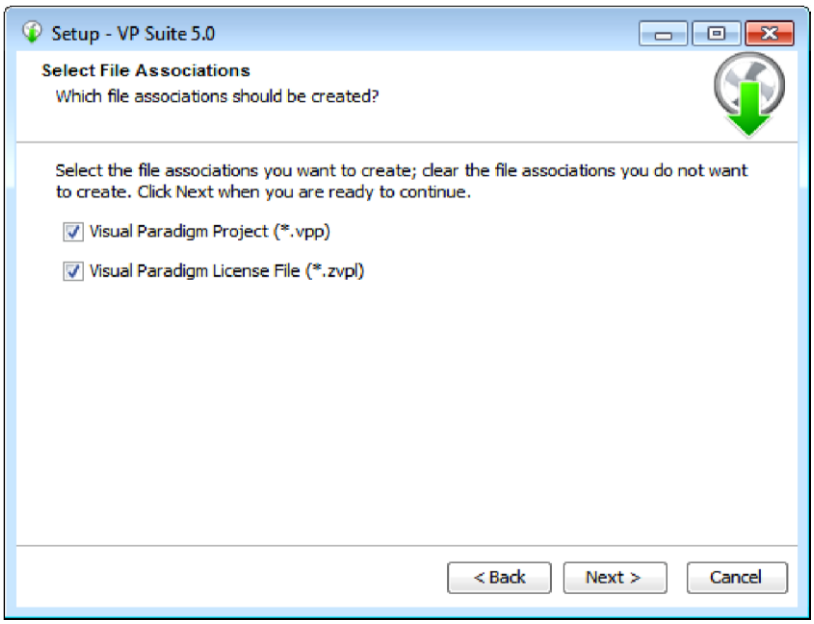
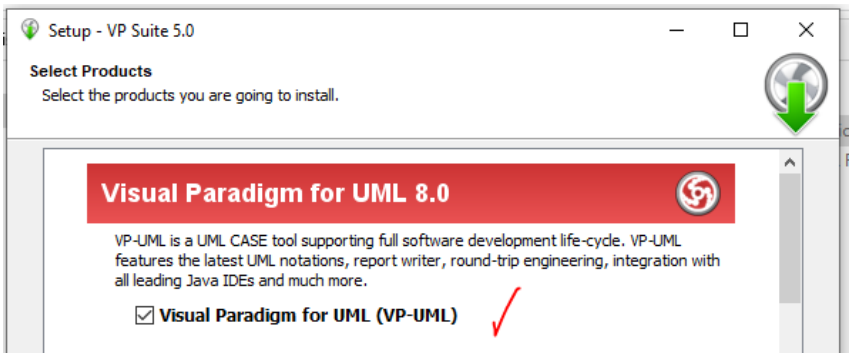
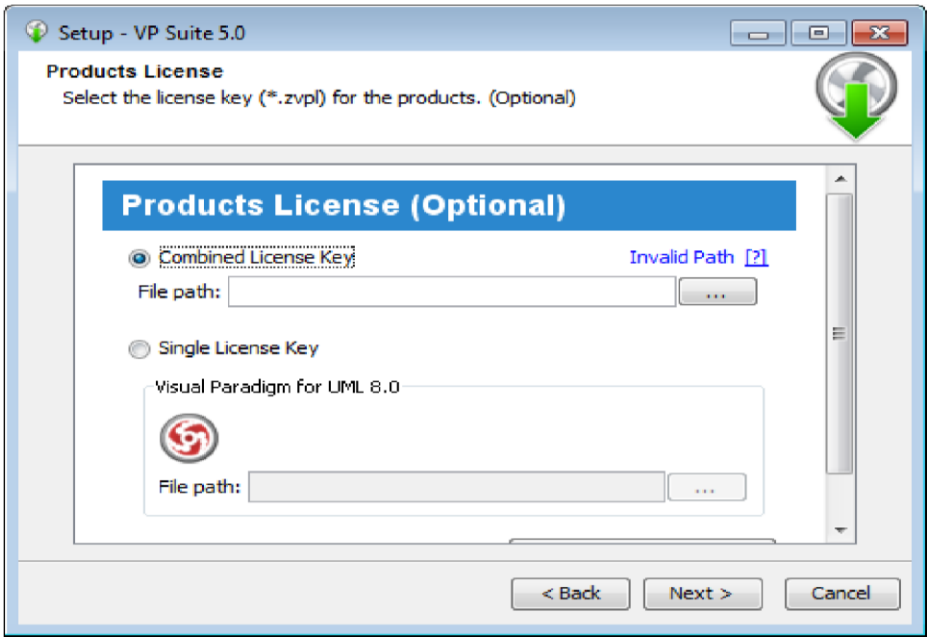


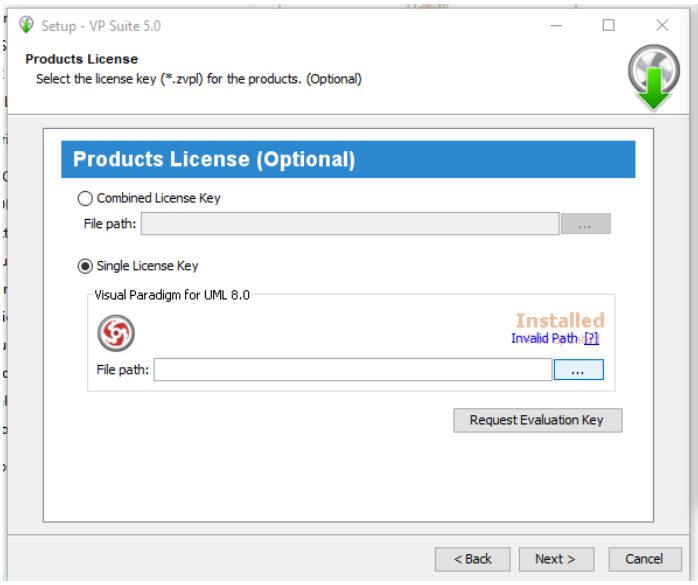
Figure 7: Selection of file association

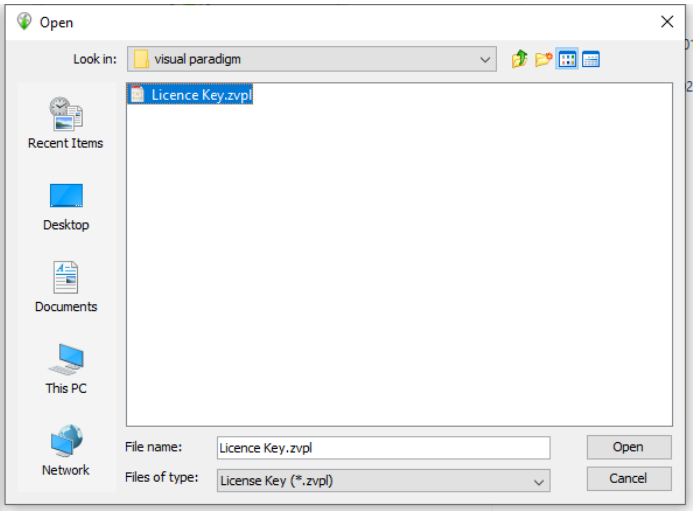


Step-7

Select the license key.







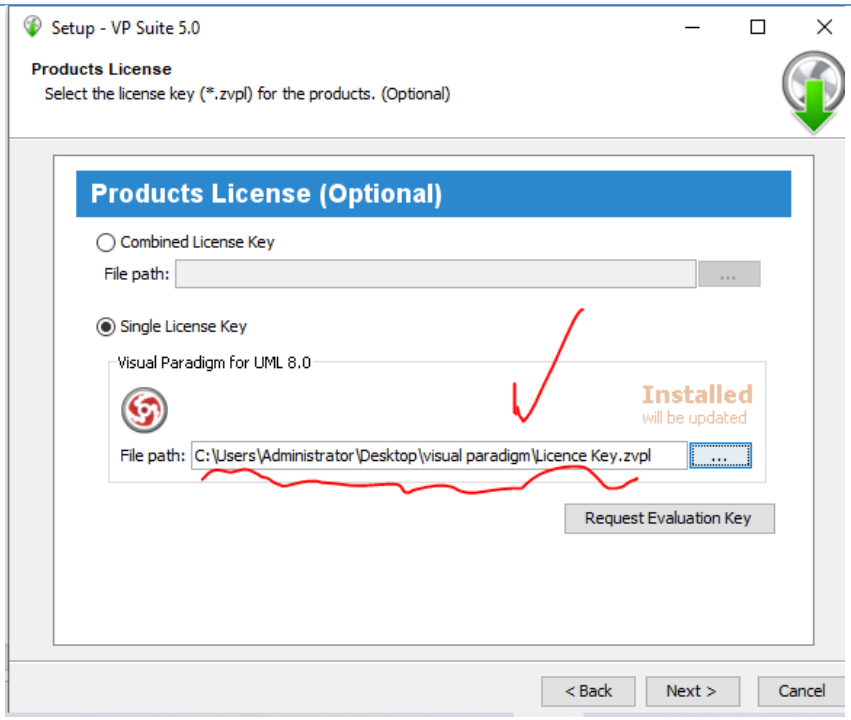


Figure 8: Selection of the license key

Step-8

Setup Installation in progress

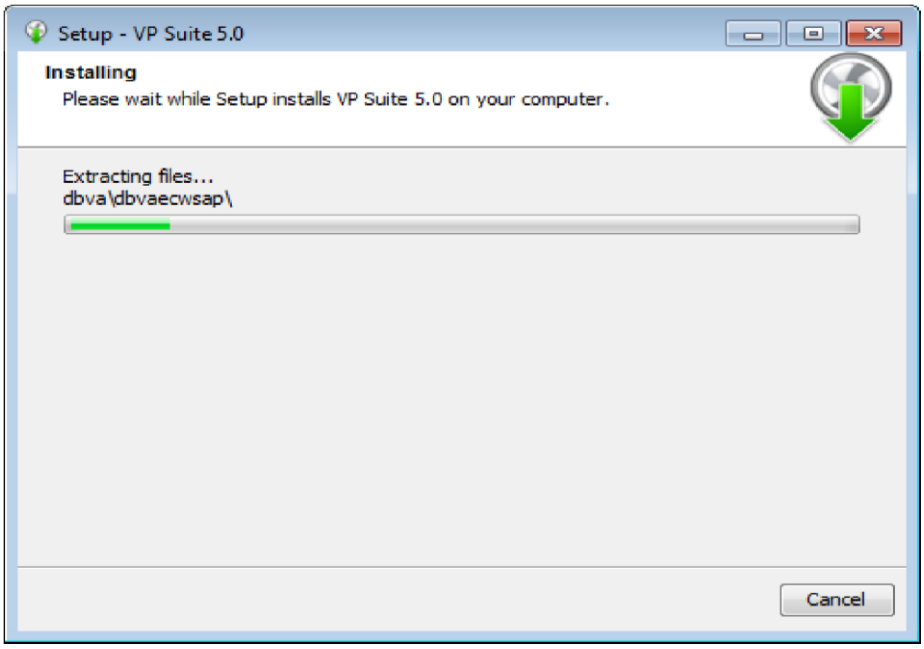


Figure 9: Setup Installation in progress **Step-9**

Exit from setup

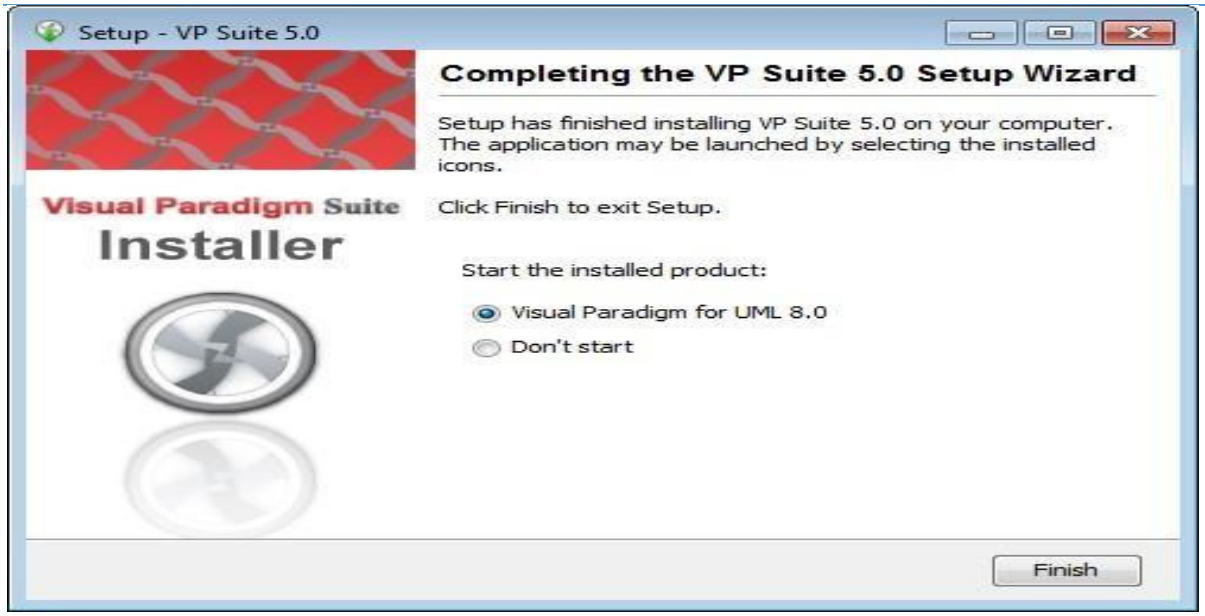


Figure 10: Exit from Setup

SOFTWARE TOOL

Visual paradigm

LAB-02

OBJECTIVE

After this lab student will be able to:

- Understand the concepts underlining Use Case Model for Requirement Engineering
- Model Use Case Model and its application

PRE-LAB READING ASSIGNMENT

NONE

LAB RELATED CONTENT

Use case diagram is a kind of UML diagram that enables you to model system functions (i.e. goals) as well as the actors that interact with those functions. You can draw use case diagrams in Visual Paradigm as well as to document the use case scenario of use cases using the flow-of-events editor.

SOFTWARE TOOL

Visual paradigm

EXPERIMENTS

Draw a use case diagram using your tool and Follow below rules.

Use Cases

A use case is a written description of how users will perform tasks on your website. It outlines, from a user's point of view, a system's behavior as it responds to a request. Each use case is represented as a sequence of simple steps, beginning with a user's goal and ending when that goal is fulfilled.

Benefits of Use Cases

Use cases add value because they help explain how the system should behave and in the process, they also help brainstorm what could go wrong. They provide a list of goals and this list can be used to establish the cost and complexity of the system. Project teams can then negotiate which functions become requirements and are built.

What Use Cases Include	What Use Cases Do NOT Include
• Who is using the System • What the user want to do • The user's goal • The steps the user takes to accomplish a particular task • How the website should respond to an action	• Implementation-specific language • Details about the user interfaces or screens.

Elements of a Use Case

Depending on how in depth and complex you want or need to get, use cases describe a combination of the following elements:

- **Actor** – anyone or anything that performs a behavior (who is using the system)
- **Stakeholder** – someone or something with vested interests in the behavior of the system under discussion (SUD)
- **Primary Actor** – stakeholder who initiates an interaction with the system to achieve a goal
- **Preconditions** – what must be true or happen before and after the use case runs.
- **Triggers** – this is the event that causes the use case to be initiated.
- **Main success scenarios** [Basic Flow] – use case in which nothing goes wrong.
- **Alternative paths [**Alternative Flow] – these paths are a variation on the main theme. These exceptions are what happen when things go wrong at the system level.

How To Write a Use Case

Write the steps in a use case in an easy-to-understand narrative. Kenworthy (1997) outlines the following steps:

1. Identify who is going to be using the System.
2. Pick one of those users.
3. Define what that user wants to do on the site. Each thing the use does on the site becomes a use case.
4. For each use case, decide on the normal course of events when that user is using the site.
5. Describe the basic course in the description for the use case. Describe it in terms of what the user does and what the system does in response that the user should be aware of.
6. When the basic course is described, consider alternate courses of events and add those to "extend" the use case.
7. Look for commonalities among the use cases. Extract these and note them as common course use cases.
8. Repeat the steps 2 through 7 for all other users.

vidu

PROCEDURE

Perform the steps below to create a UML use case diagram in Visual Paradigm.

1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Use Case Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

1. Drawing a system

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

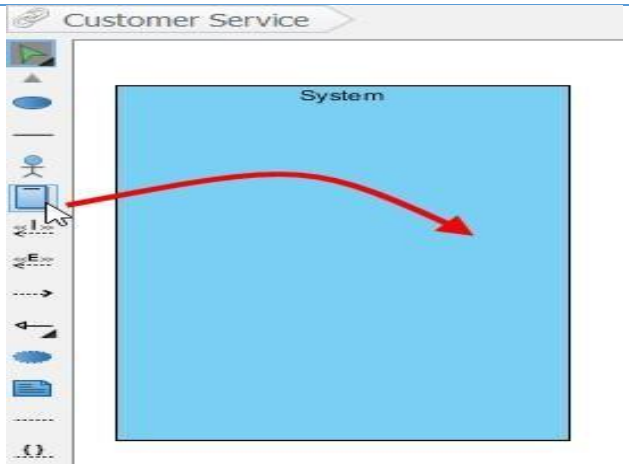


Figure 11: Create a system

Drawing an actor

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

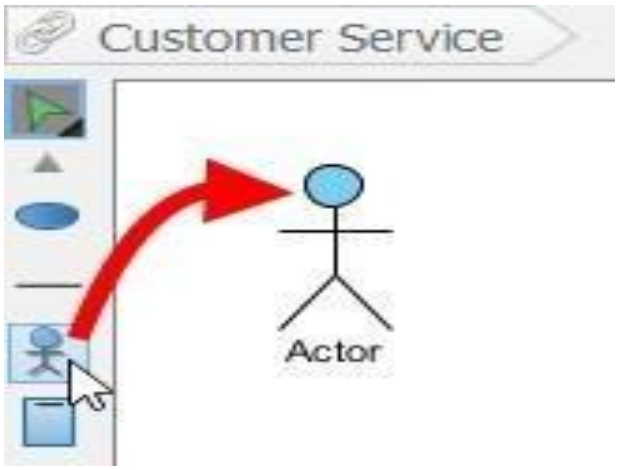


Figure 12: Drawing an actor

Drawing a use case

Besides creating a use case through diagram toolbar, you can also create it through Resource Catalog:

1. Move the mouse over a source shape (e.g. an actor).
2. Release the mouse button until it reaches your preferred place.
3. Select **Association -> Use Case** from Resource Catalog.

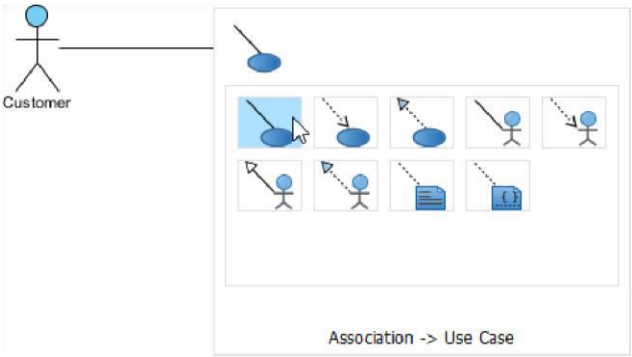


Figure 13: Drawing a Use Case

The source shape and the newly created use case are connected. Finally, name the newly created use case.

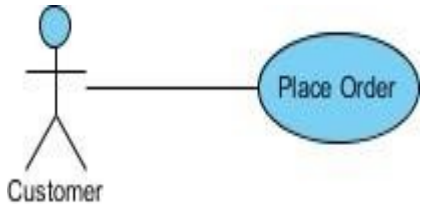


Figure 14: Association of Actor with Use Case

STUDENT TASK-01

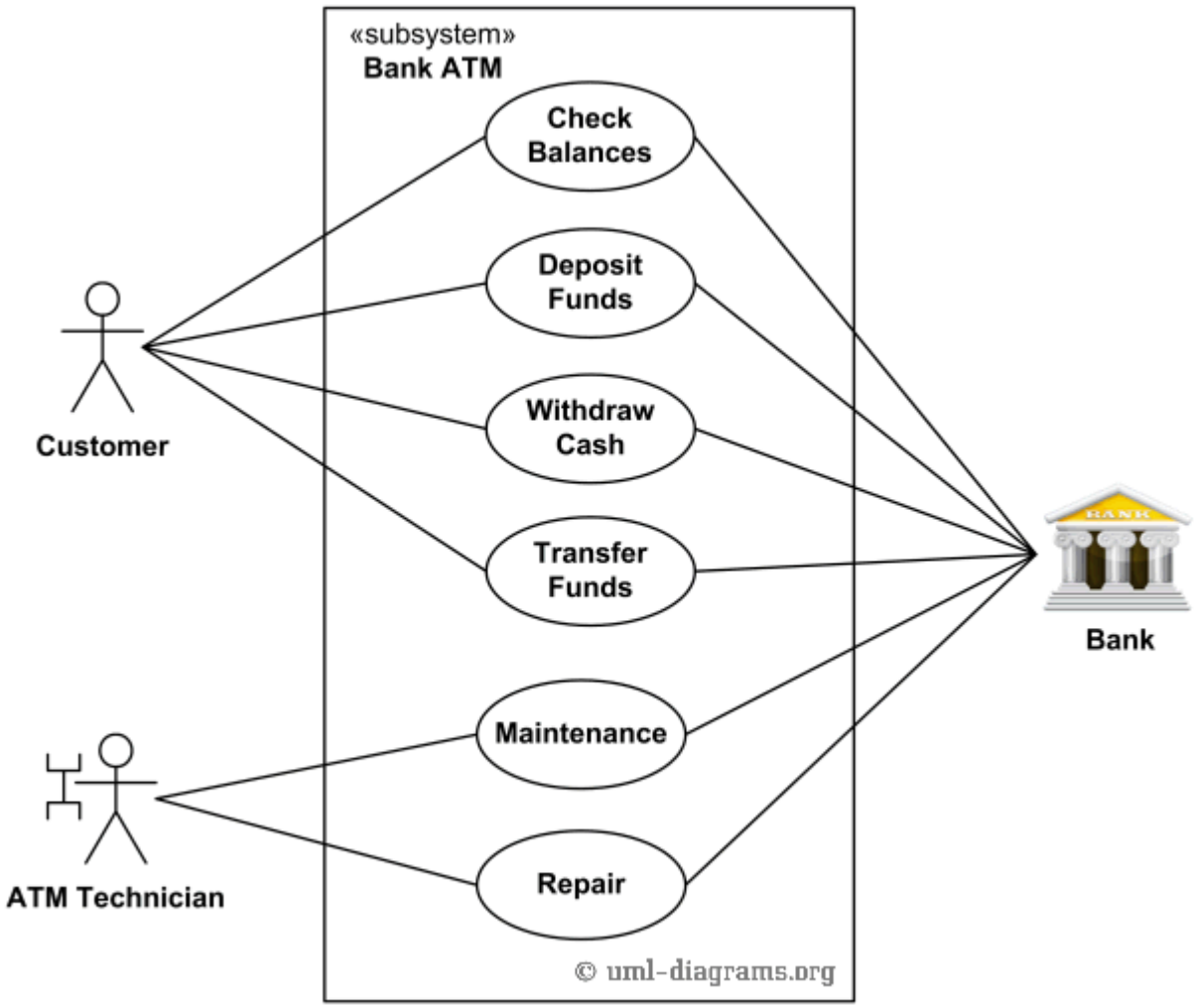
Using your knowledge of how an ATM is used, develop a use case diagram that could serve as a basis for understanding the requirements for an ATM system.

Bank ATM

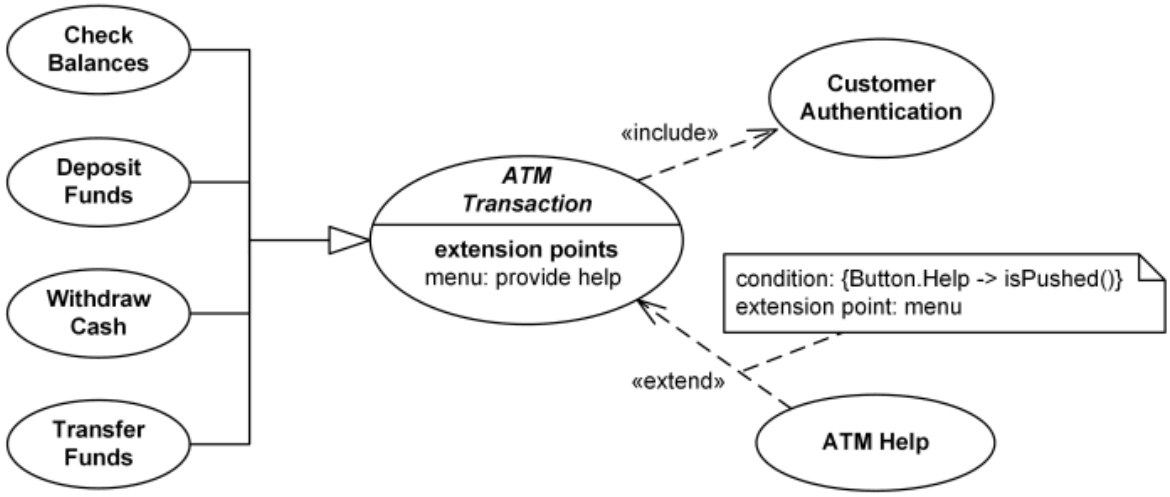
UML Use Case Diagram Examples

An automated teller machine (**ATM**) or the automatic banking machine (**ABM**) is a banking subsystem (subject) that provides bank customers with access to financial transactions in a public space without the need for a cashier, clerk, or bank teller.

Customer (actor) uses bank ATM to *Check Balances* of his/her bank accounts, *Deposit Funds*, *Withdraw Cash* and/or *Transfer Funds* (use cases). *ATM Technician* provides *Maintenance* and *Repairs*. All these use cases also involve *Bank* actor whether it is related to customer transactions or to the ATM servicing.

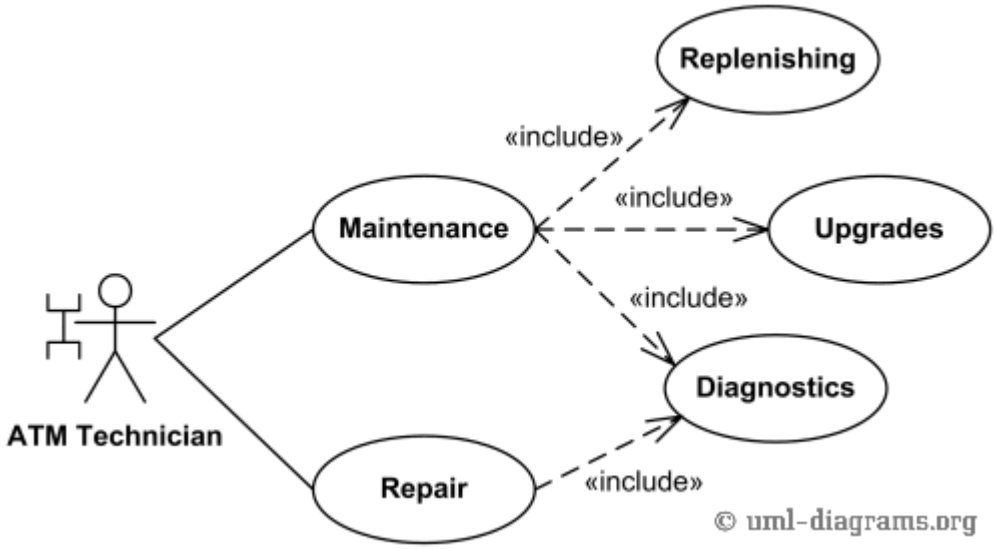


An example of use case diagram for Bank ATM subsystem - top level use cases.
On most bank ATMs, the customer is authenticated by inserting a plastic ATM card and entering a personal identification number (PIN). *Customer Authentication* use case is required for every ATM transaction so we show it as include relationship. Including this use case as well as transaction generalizations make the *ATM Transaction* an abstract use case.



Bank ATM Transactions and Customer Authentication Use Cases Example.

Customer may need some help from the ATM. *ATM Transaction* use case is extended via extension point called *menu* by the *ATM Help* use case whenever *ATM Transaction* is at the location specified by the *menu* and the bank customer requests help, e.g. by selecting Help menu item. **Replenishing** (to make full or complete again)



Bank ATM Maintenance, Repair, Diagnostics Use Cases Example.
ATM Technician maintains or repairs Bank ATM. *Maintenance* use case includes *Replenishing* ATM with cash, ink or printer paper, *Upgrades* of hardware, firmware or software, and remote or on-site *Diagnostics*. *Diagnostics* is also included in (shared with) *Repair* use case.

STUDENT TASK-02

For a hospital management system proposes a set of use cases that illustrates the interactions between a doctor, who sees patients and prescribes medicine and treatments.

LAB-03

OBJECTIVE

After this lab students will be able to get;

- Familiarize with the concepts underlining Domain Model for Requirement Analysis
- Model Domain diagram and its application

PRE-LAB READING ASSIGNMENT

Concepts of domain model

LAB RELATED CONTENT

What is Domain Model?

A partial domain model is drawn with UML class diagram notation. In the Unified Modeling Language (UML), a class diagram is used to represent the domain model.

Applying the UML class diagram notation for a domain model yields a conceptual perspective model (describing things in situation of real world). It usually pays off during design, when it supports better understanding and communication.

Is a Domain Model a Picture of Software Business Objects?

A Domain Model is a visualization of things in a real-situation domain of interest, not of software objects such as Java or C# classes, or software objects with responsibilities. Therefore, the following elements are not suitable in a domain model:

- Software artifacts (i.e. window, database), unless the domain being modeled is of software concepts, such as a model of graphical user interfaces.
- Responsibilities or methods **What are Conceptual Classes?**

The domain model illustrates conceptual classes or vocabulary in the domain. Informally, a conceptual class is an idea, thing, or object. Formally, a conceptual class may be considered in terms of its symbol, intension, and extension.

- Symbol words or images representing a conceptual class.
- Intension the definition of a conceptual class.
- Extension the set of examples to which the conceptual class applies.

SOFTWARE TOOL

Visual paradigm

EXPERIMENTS

EXAMPLE EXPERIMENT-01

Draw a domain diagram of point of sale system.

PROCEDURE

Perform the steps below to create a UML activity diagram in Visual Paradigm.

1. Select **Diagram > New** from the application toolbar.

2. In the **New Diagram** window, select **Class Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

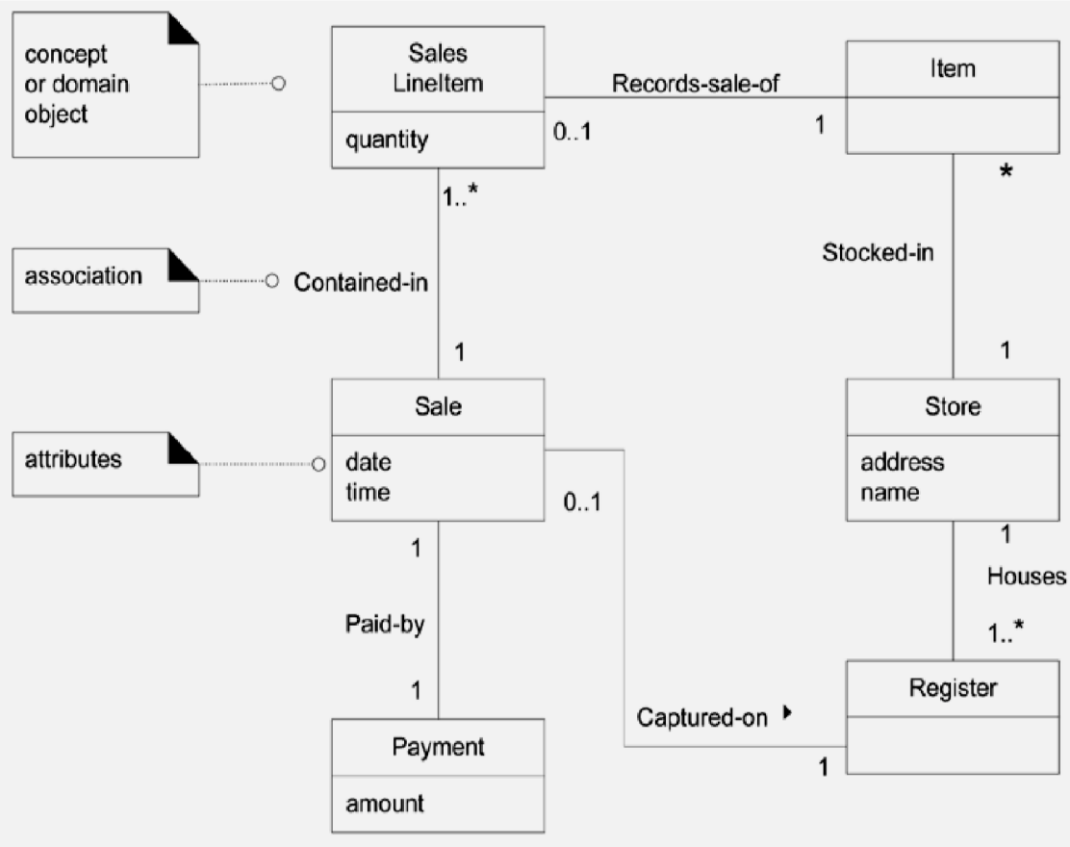


Figure 18: Point of Sale (POS) System

STUDENT TASK-01

Using your knowledge of how an ATM is used, develop a domain model that could serve as a basis for understanding the conceptual classes for an ATM system.

STUDENT TASK-02

Look carefully at how messages and mailboxes are represented in the e-mail system that you use. Model the conceptual classes (domain model) that might be used in the system implementation to represent a mailbox and an e-mail message.

LAB-04

OBJECTIVE

After this lab students will be able to:

- Understand the concepts underlining Sequence Model
- Model Sequence diagram and its application

PRE-LAB READING ASSIGNMENT

Basics of sequence diagram

LAB RELATED CONTENT

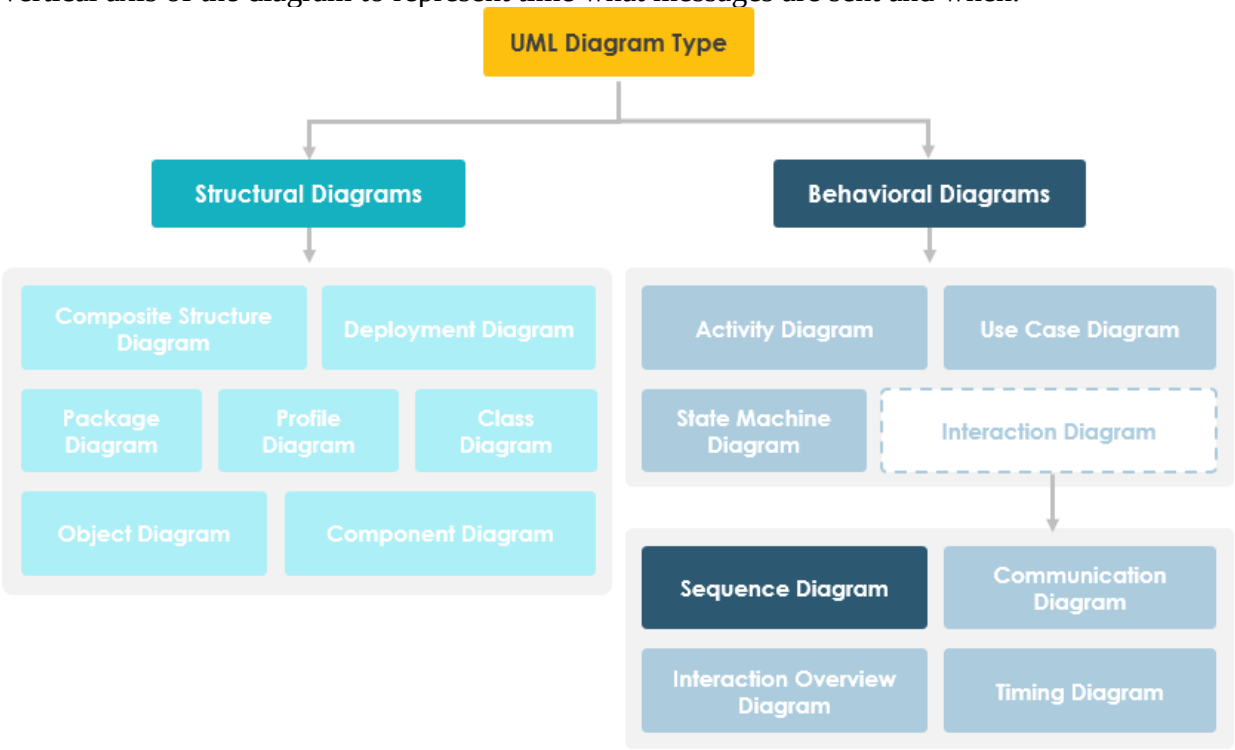
A sequence diagram is a kind of UML diagram that is used primarily to show the interactions between objects that are represented as lifelines in a sequential order

SOFTWARE TOOL

Visual paradigm

EXPERIMENTS

UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.



To understand what a sequence diagram is, it's important to know the role of the Unified Modeling Language, better known as UML. UML is a modeling toolkit that guides the creation and notation of many types of diagrams, including behavior diagrams, interaction diagrams, and structure diagrams.

A sequence diagram is a type of interaction diagram because **it describes how—and in what order—a group of objects works together.** These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

Benefits of sequence diagrams

Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:

- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario.

Use cases for sequence diagrams

The following scenarios are ideal for using a sequence diagram:

- **Usage scenario:** A usage scenario is a diagram of how your system could potentially be used. It's a great way to make sure that you have worked through the logic of every usage scenario for the system.
- **Method logic:** Just as you might use a UML sequence diagram to explore the logic of a use case, you can use it to explore the logic of any function, procedure, or complex process.
- **Service logic:** If you consider a service to be a high-level method used by different clients, a sequence diagram is an ideal way to map that out.

Sequence Diagrams at a Glance

Sequence Diagrams show elements as they interact over time and they are organized according to object (horizontally) and time (vertically):

Object Dimension

- The horizontal axis shows the elements that are involved in the interaction
- Conventionally, the objects involved in the operation are listed from left to right according to when they take part in the message sequence. However, the elements on the horizontal axis may appear in any order

Time Dimension

- The vertical axis represents time proceedings (or progressing) down the page.

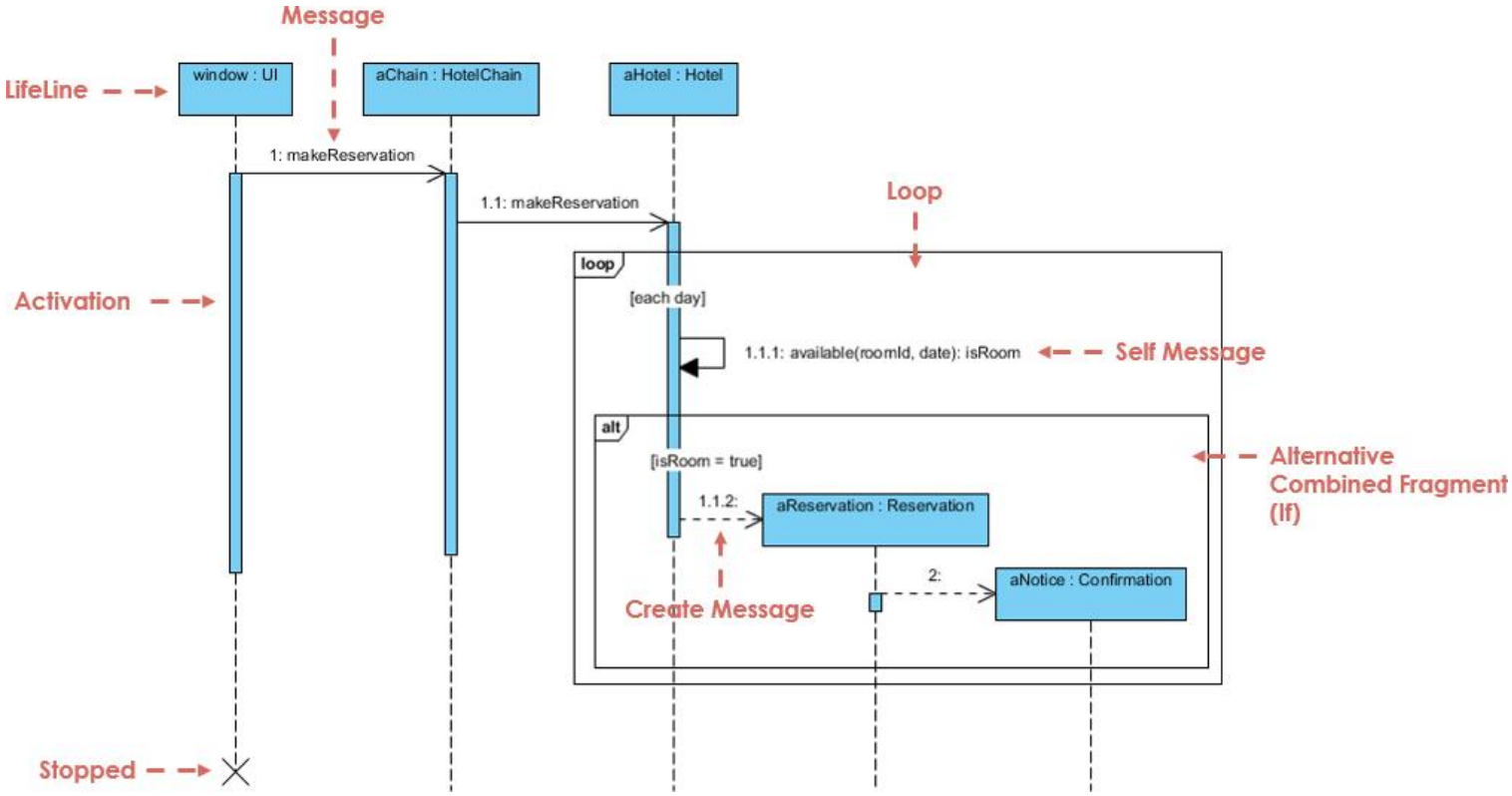
Note that:

Time in a sequence diagram is all a about ordering, not duration. The vertical space in an interaction diagram is not relevant for the duration of the interaction.

Sequence Diagram Example: Hotel System

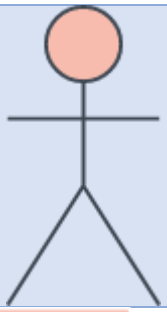
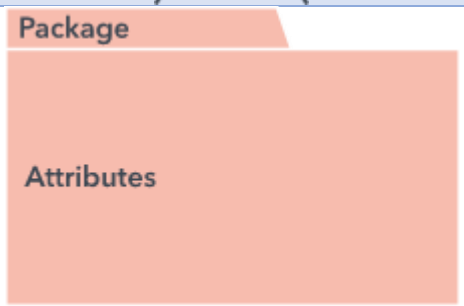
Sequence Diagram is an interaction diagram that details how operations are carried out -- what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when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

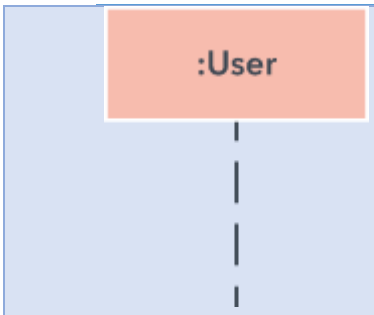
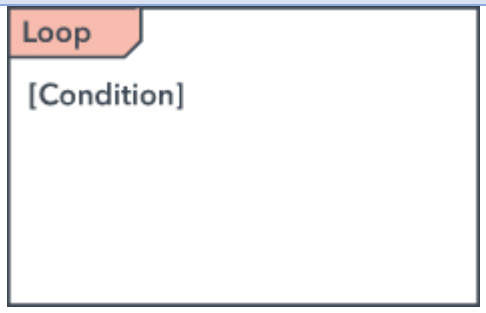
Below is a sequence diagram for making a hotel reservation. The object initiating the sequence of messages is a Reservation window.



Basic symbols and components

To understand what a sequence diagram is, you should be familiar with its symbols and components. Sequence diagrams are made up of the following icons and elements:

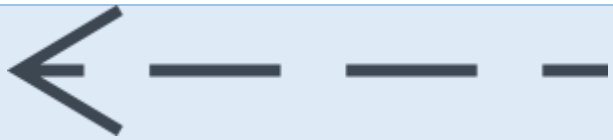
Symbol	Name	Description
	Object symbol	Represents a class or object in UML. The object symbol demonstrates how an object will behave in the context of the system. Class attributes should not be listed in this shape. The boxes across the top of the diagram represent classifiers or their instances Typically these show objects, classes, or actors (usually depicted as rectangles, although they can also be symbols)
	**Activation box**	Represents the time needed for an object to complete a task. The longer the task will take, the longer the activation box becomes.
	**Actor symbol**	Shows entities that interact with or are external to the system.
	**Package symbol**	Used in UML 2.0 notation to contain interactive elements of the diagram. Also known as a frame, this rectangular shape has a small inner rectangle for labeling the diagram.

	**Lifeline symbol**	Represents the passage of time as it extends downward. This dashed vertical line shows the sequential events that occur to an object during the charted process. Lifelines may begin with a labeled rectangle shape or an actor symbol.
	**Option loop symbol**	Used to model if/then scenarios, i.e., a circumstance that will only occur under certain conditions.
	**Alternative symbol**	Symbolizes a choice (that is usually mutually exclusive) between two or more message sequences. To represent alternatives, use the labeled rectangle shape with a dashed line inside.

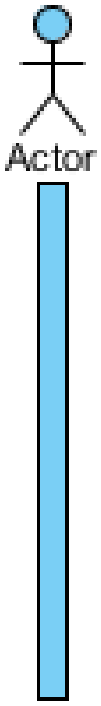
Common message symbols

Use the following arrows and message symbols to show how information is transmitted between objects. These symbols may reflect the start and execution of an operation or the sending and reception of a signal.

Symbol	Name	Description

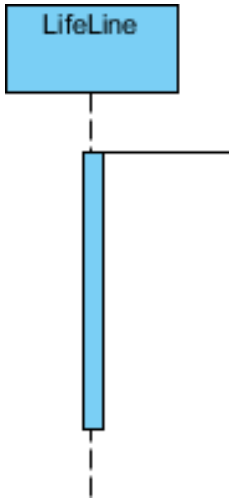
	**Synchronous message symbol**	Represented by a solid line with a solid arrowhead. This symbol is used when a sender must wait for a response to a message before it continues. The diagram should show both the call and the reply.
	**Asynchronous message symbol**	Represented by a solid line with a lined arrowhead. Asynchronous messages don't require a response before the sender continues. Only the call should be included in the diagram.
	**Asynchronous return message symbol**	Represented by a dashed line with a lined arrowhead.
	**Asynchronous create message symbol**	Represented by a dashed line with a lined arrowhead. This message creates a new object.
	**Reply message symbol**	Represented by a dashed line with a lined arrowhead, these messages are replies to calls.
	**Delete message symbol**	Represented by a solid line with a solid arrowhead, followed by an X. This message destroys an object.

Sequence Diagram Notation

Notation Description	Visual Representation
Actor • a type of role played by an entity that interacts with the subject (e.g., by exchanging signals and data) • external to the subject (i.e., in the sense that an instance of an actor is not a part of the instance of its corresponding subject). • represent roles played by human users, external hardware, or other subjects. Note that: • An actor does not necessarily represent a specific physical entity but merely a particular role of some entity • A person may play the role of several different actors and, conversely, a given actor may be played by multiple different person.	
Lifeline • A lifeline represents an individual participant in the Interaction.	

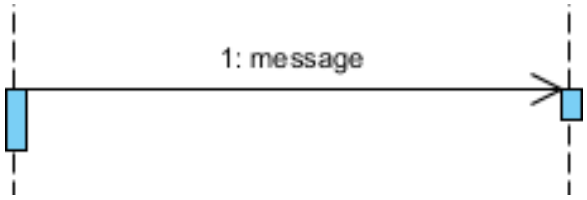
Activations

- A thin rectangle on a lifeline) represents the period during which an element is performing an operation.
- The top and the bottom of the of the rectangle are aligned with the initiation and the completion time respectively



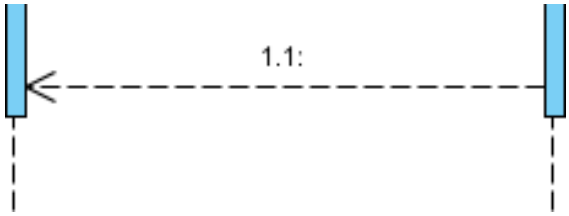
Call Message

- A message defines a particular communication between Lifelines of an Interaction.
- Call message is a kind of message that represents an invocation of operation of target lifeline.



Return Message

- A message defines a particular communication between Lifelines of an Interaction.
- Return message is a kind of message that represents the pass of information back to the caller of a corresponded former message.

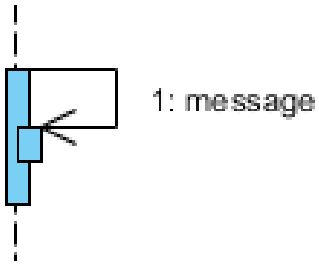


Self Message

- A message defines a particular communication between Lifelines of an Interaction.
- Self message is a kind of message that represents the invocation of message of the same lifeline.

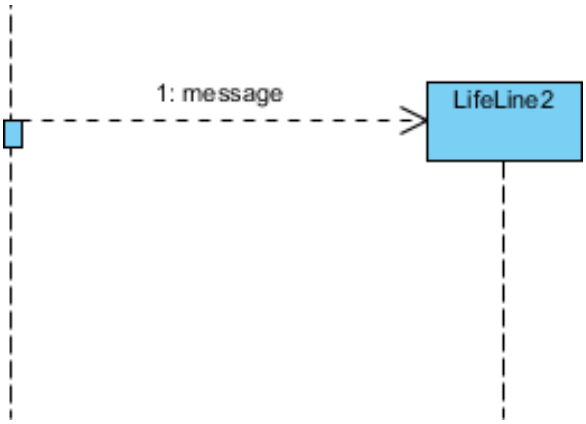
Recursive Message

- A message defines a particular communication between Lifelines of an Interaction.
- Recursive message is a kind of message that represents the invocation of message of the same lifeline. It's target points to an activation on top of the activation where the message was invoked from.



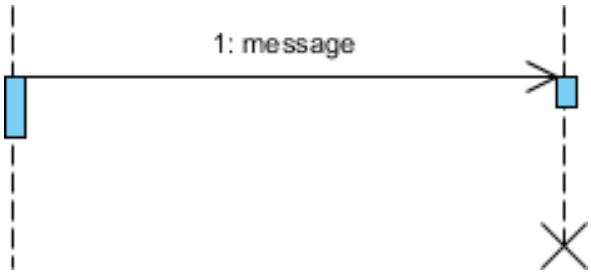
Create Message

- A message defines a particular communication between Lifelines of an Interaction.
- Create message is a kind of message that represents the instantiation of (target) lifeline.



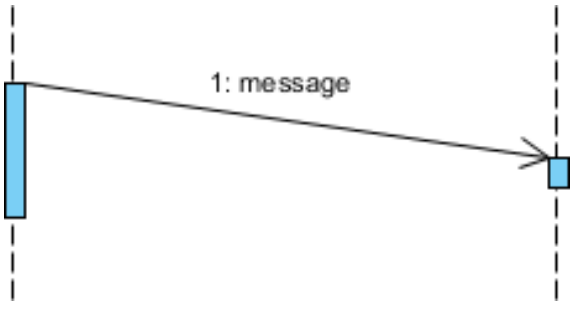
Destroy Message

- A message defines a particular communication between Lifelines of an Interaction.
- Destroy message is a kind of message that represents the request of destroying the lifecycle of target lifeline.

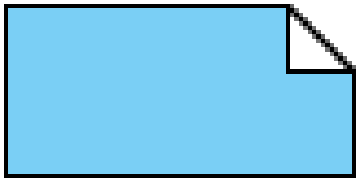


Duration Message

- A message defines a particular communication between Lifelines of an Interaction.
- Duration message shows the distance between two-time instants for a message invocation.

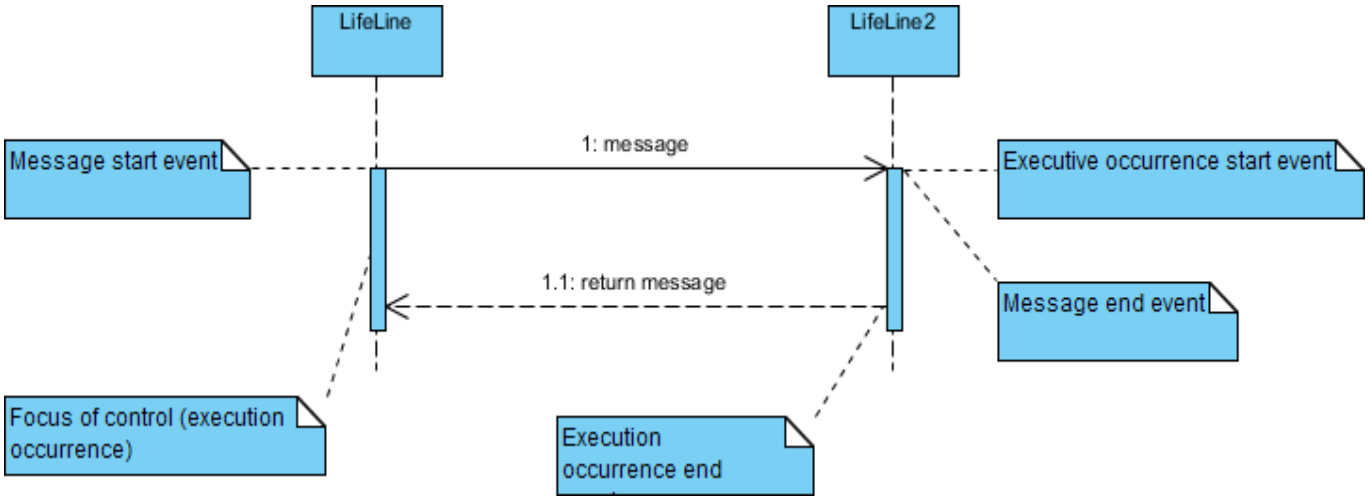


Note
A note (comment) gives the ability to attach various remarks to elements. A comment carries no semantic force, but may contain information that is useful to a modeler.



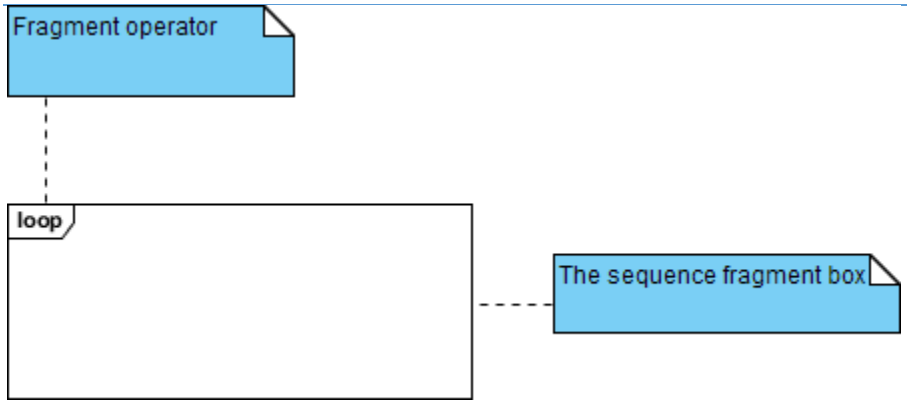
Message and Focus of Control

- An Event is any point in an interaction where something occurs.
- Focus of control: also called execution occurrence, an execution occurrence
- It shows as tall, thin rectangle on a lifeline)
- It represents the period during which an element is performing an operation. The top and the bottom of the rectangle are aligned with the initiation and the completion time respectively.



Sequence Fragments

- **UML 2.0** introduces sequence (or interaction) fragments. Sequence fragments make it easier to create and maintain accurate sequence diagrams
- A sequence fragment is represented as a box, called a combined fragment, which encloses a portion of the interactions within a sequence diagram
- The fragment operator (in the top left cornet) indicates the type of fragment
- Fragment types: ref, assert, loop, break, alt, opt, neg

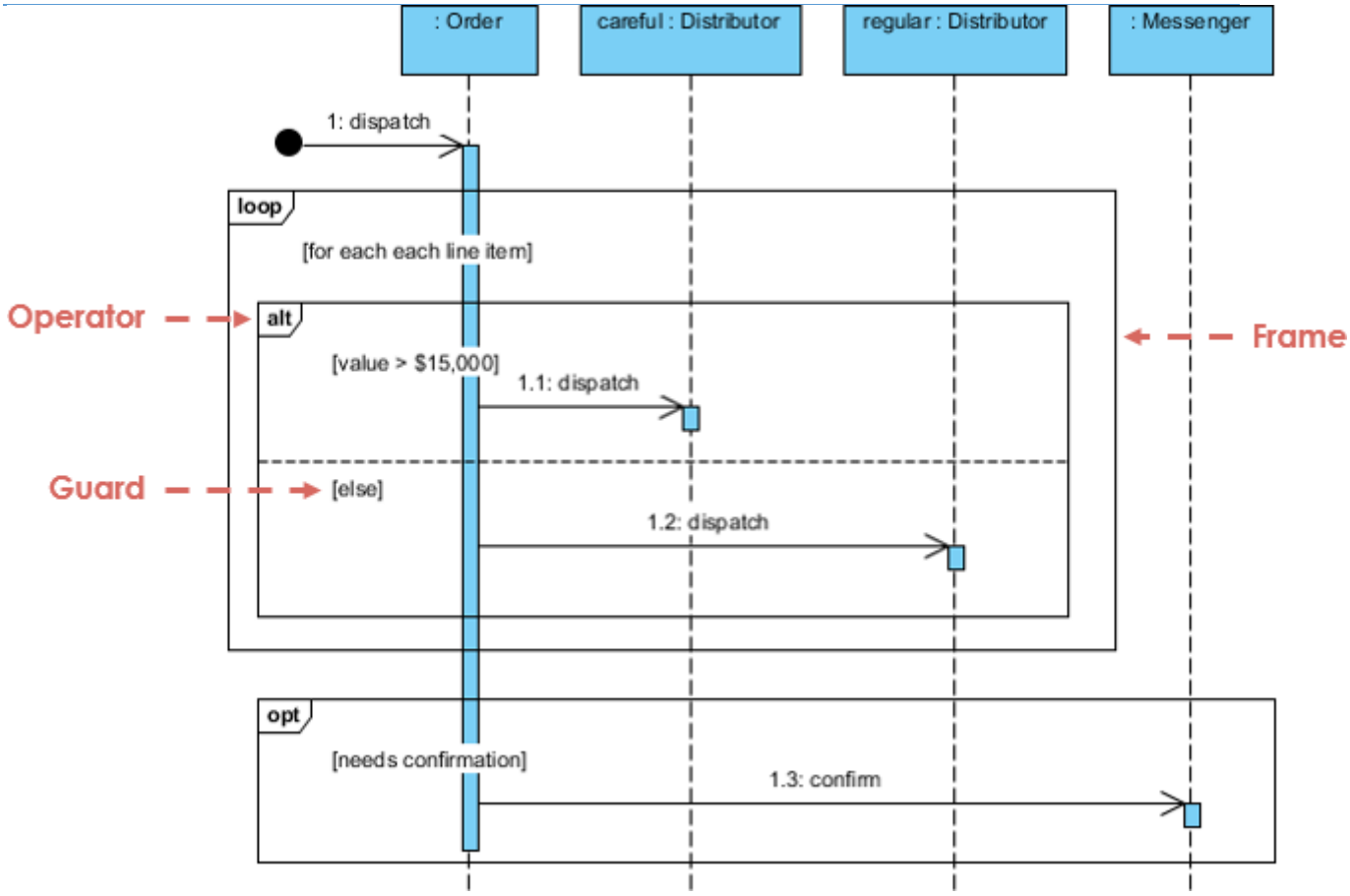


Operator	Fragment Type
alt	Alternative multiple fragments: only the one whose condition is true will execute.
opt	Optional: the fragment executes only if the supplied condition is true. Equivalent to an alt only with one trace.
par	Parallel: each fragment is run in parallel.
loop	Loop: the fragment may execute multiple times, and the guard indicates the basis of iteration.
region	Critical region: the fragment can have only one thread executing it at once.
neg	Negative: the fragment shows an invalid interaction.
ref	Reference: refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
sd	Sequence diagram: used to surround an entire sequence diagram.

Note That:

- It is possible to combine frames in order to capture, e.g., loops or branches.
- **Combined fragment** keywords: alt, opt, break, par, seq, strict, neg, critical, ignore, consider, assert and loop.
- Constraints are usually used to show timing constraints on messages. They can apply to the timing of one message or intervals between messages.

Combined Fragment Example



Sequence Diagram for Modeling Use Case Scenarios

User requirements are captured as use cases that are refined into scenarios. A use case is a collection of interactions between external actors and a system. In UML, a use case is:

"the specification of a sequence of actions, including variants, that a system (or entity) can perform, interacting with actors of the system."

A scenario is one path or flow through a use case that describes a sequence of events that occurs during one particular execution of a system which is often represented by a sequence diagram

Sequence Diagram - Model before Code

Sequence diagrams can be somewhat close to the code level, so why not just code up that algorithm rather than drawing it as a sequence diagram?

- A good sequence diagram is still a bit above the level of the real code
- Sequence diagrams are language neutral
- Non-coders can do sequence diagrams
- Easier to do

Search Book : Use Case

- Main scenario -

The Customer specifies an author on the Search Page and then presses the Search button.

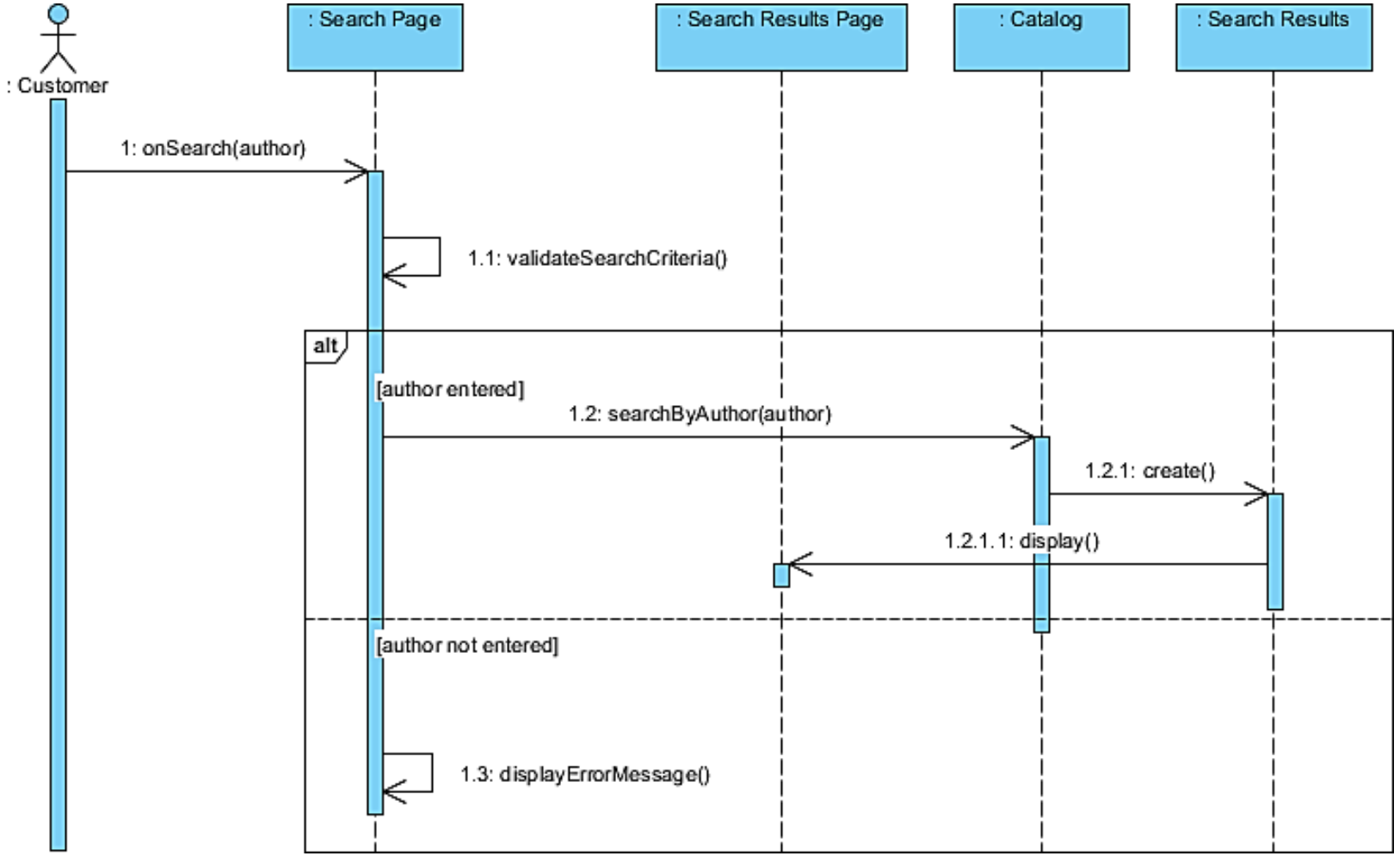
The system validates the Customer's search criteria.

If author is entered, the System searches the Catalog for books associated with the specified author.

When the search is complete, the system displays the search results on the Search Results page.

- Alternate path -

If the Customer did not enter the name of an author before pressing the Search button, the System displays an error message



PROCEDURE

Creating sequence diagram

Perform the steps below to create a UML sequence diagram Visual Paradigm uml diagram tools.

1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Sequence Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

Creating actor

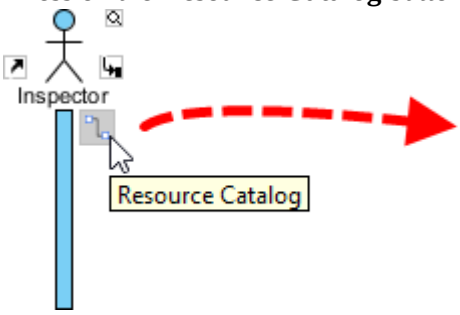
To create actor, click **Actor** on the diagram toolbar and then click on the diagram.

Create actor

Creating lifeline

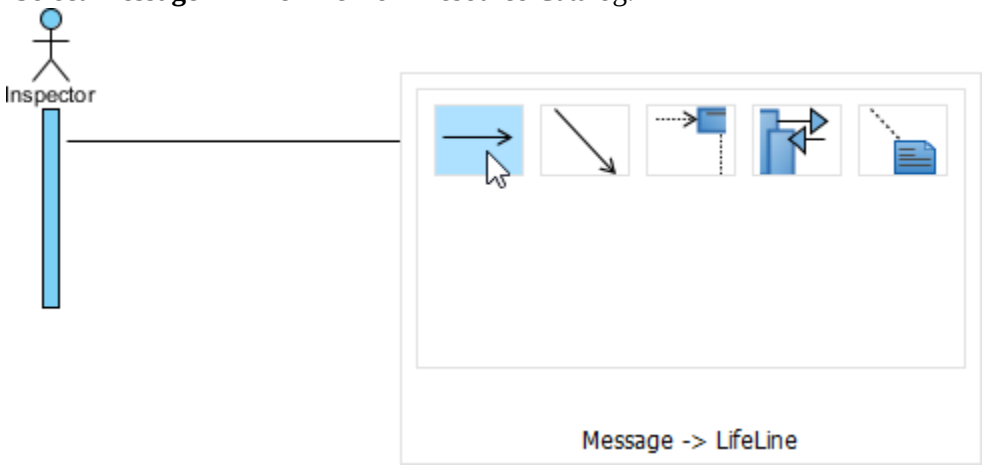
To create lifeline, you can click **LifeLine** on the diagram toolbar and then click on the diagram. Alternatively, a much quicker and more efficient way is to use Resource Catalog:

1. Move your mouse pointer over the source lifeline.
2. Press on the **Resource Catalog** button and drag it out.



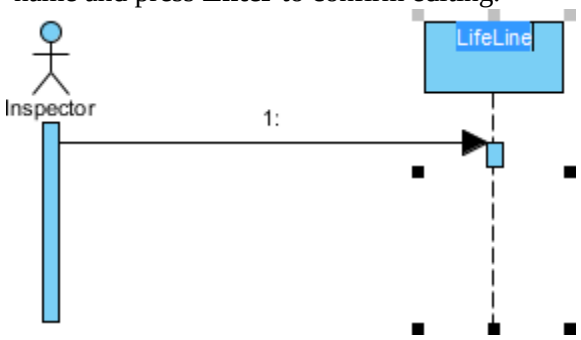
Using Resource Catalog

3. Release the mouse button at the place where you want the lifeline to be created.
4. Select **Message** -> **LifeLine** from Resource Catalog.



To create a lifeline

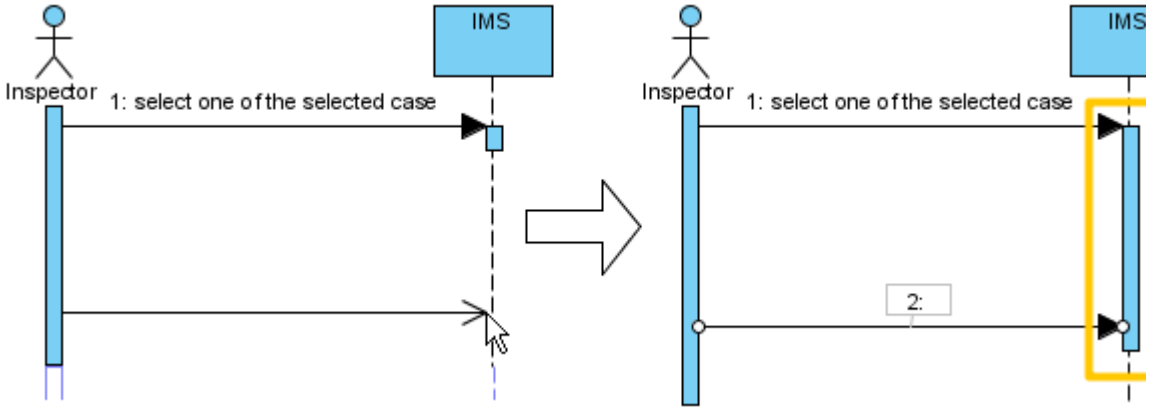
5. A new lifeline will be created and connected to the actor/lifeline with a message. Enter its name and press **Enter** to confirm editing.



Lifeline created

Auto extending activation

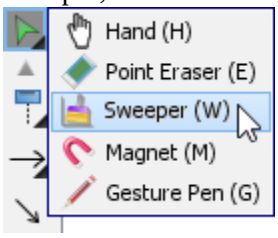
When create message between lifelines/actors, activation will be automatically extended.



Auto extending activation

Using sweeper and magnet to manage sequence diagram

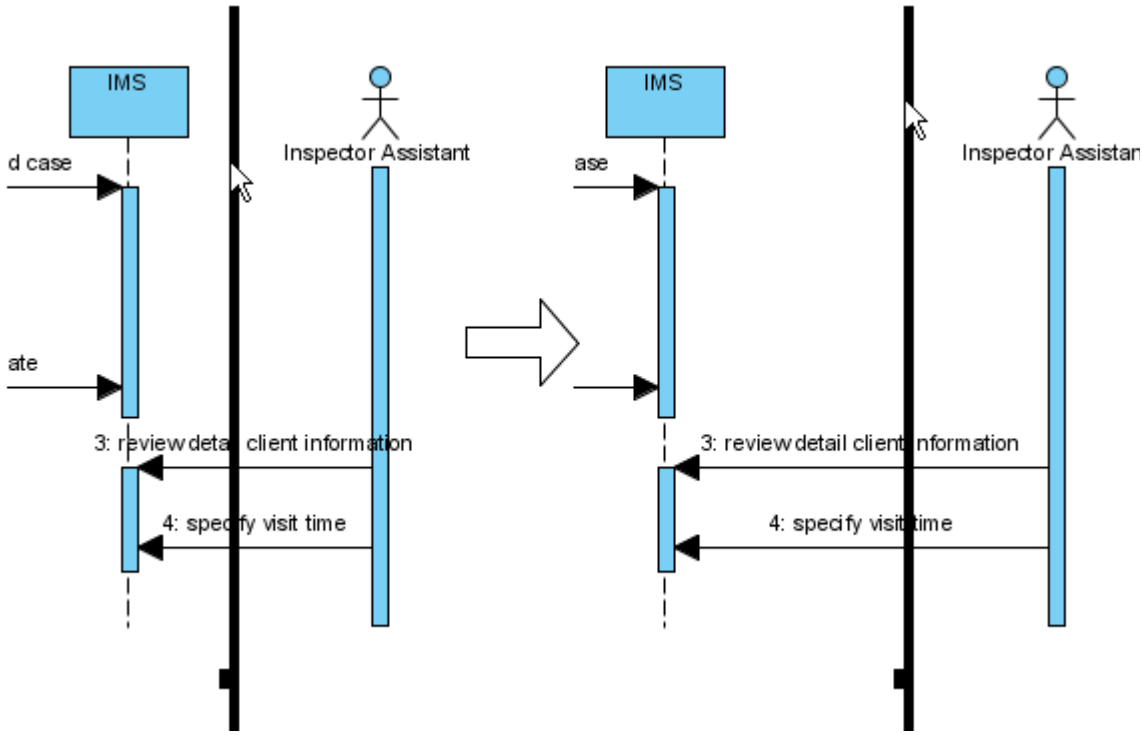
Sweeper helps you to move shapes aside to make room for new shapes or connectors. To use sweeper, click the **Selector** on the toolbar, then select **Sweeper**.



sweeper

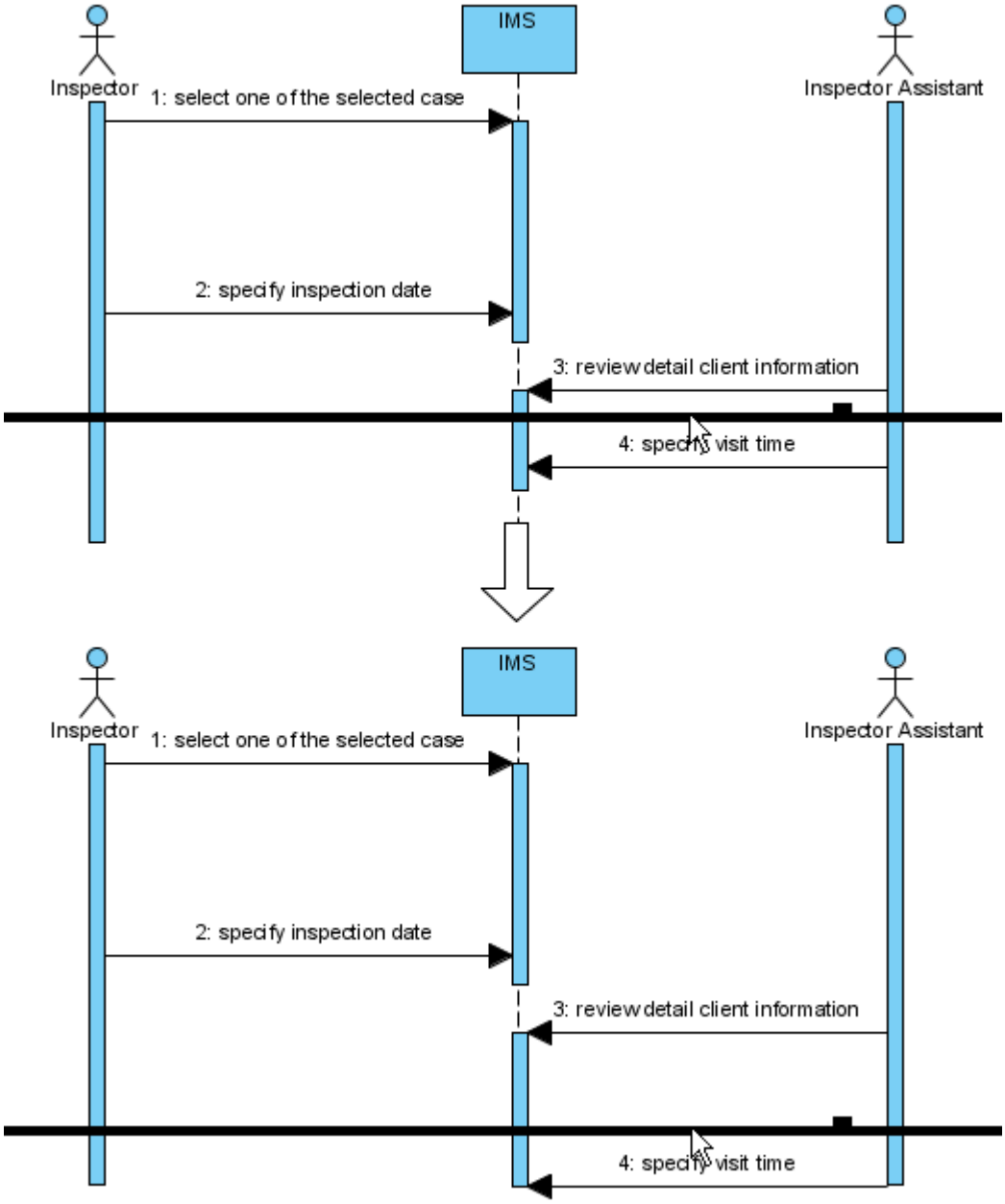
Click on empty space of the diagram and drag towards top, right, bottom or left. Shapes affected will be swept to the direction you dragged.

The picture below shows the actor *Inspector Assistant* is being swept towards right, thus new room is made for new lifelines.



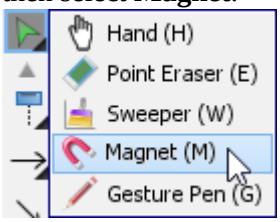
Sweep towards right

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



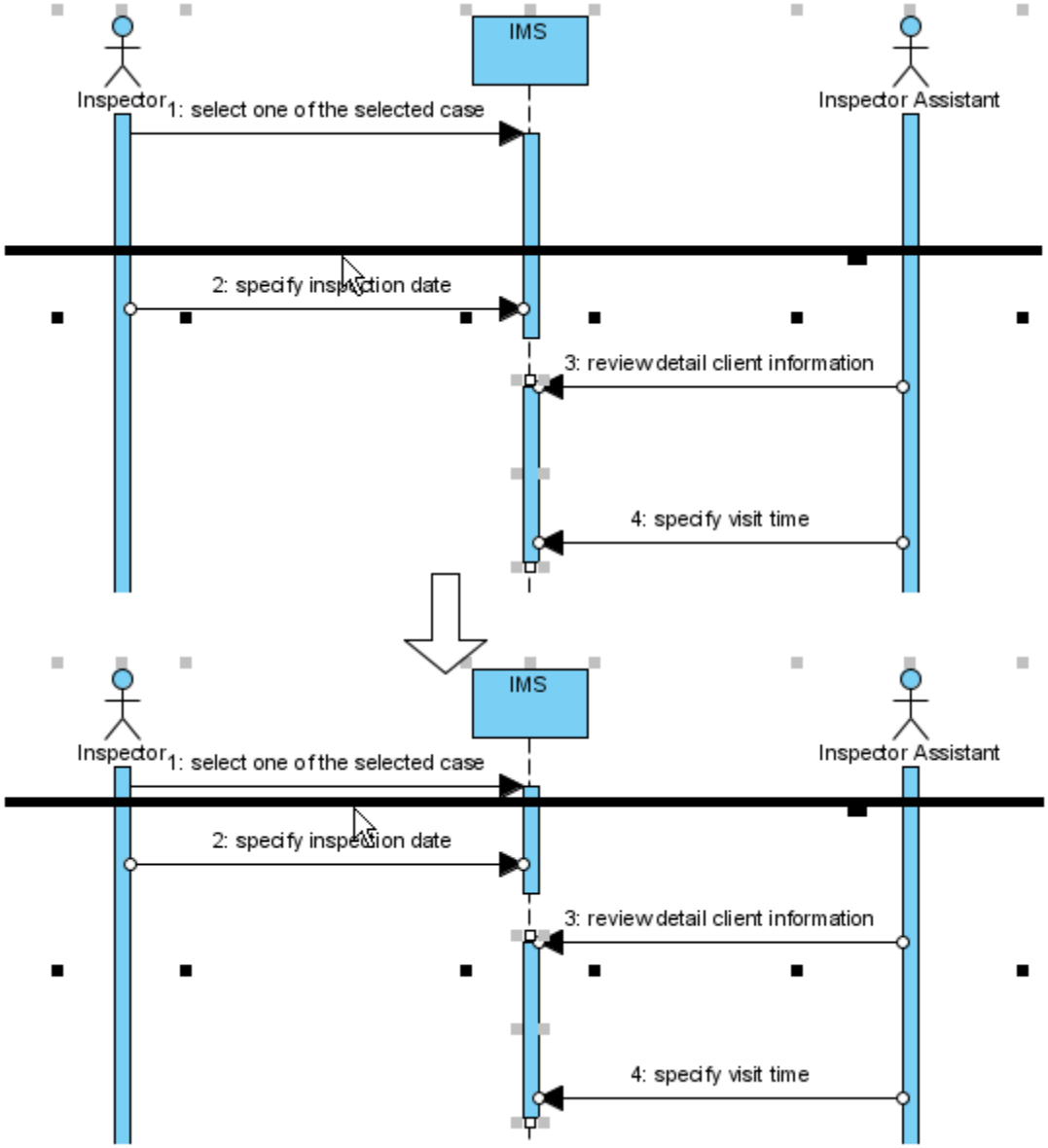
Sweep downwards

You can also use magnet to pull shapes together. To use magnet, click the **Selector** on the toolbar, then select **Magnet**.



Magnet

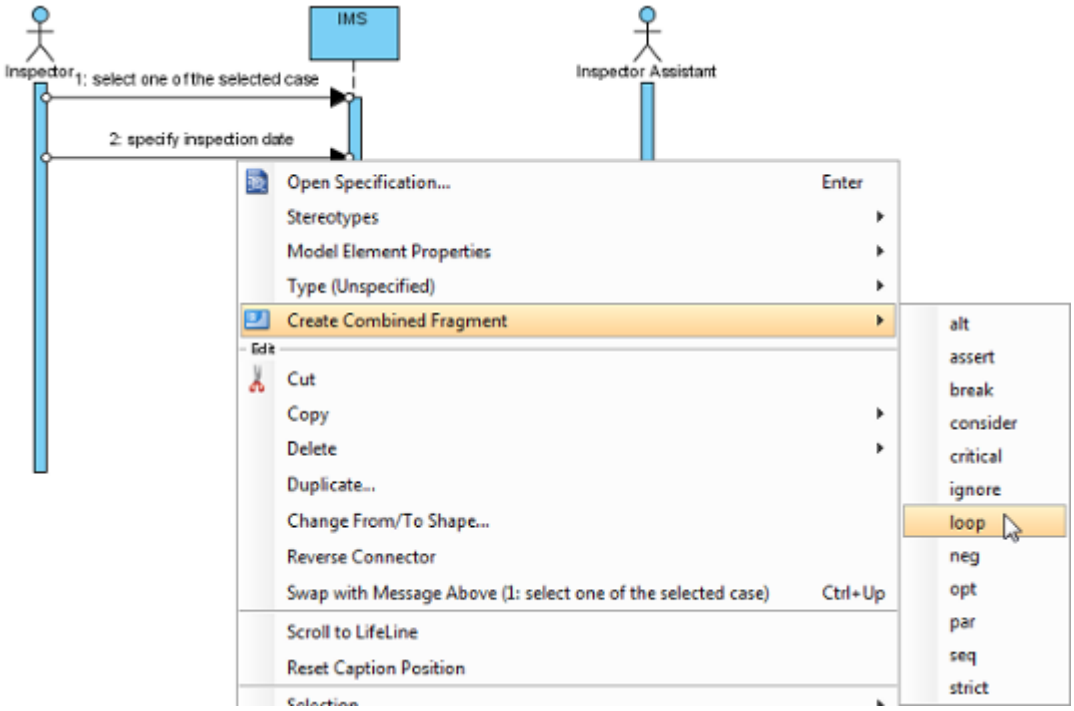
Click on empty space of the diagram and drag towards top, right, bottom or left. Shapes affected will be pulled to the direction you dragged.
The picture below shows when drag the magnet upwards, shapes below dragged position are pulled upwards.



Pull shapes upwards using magnet

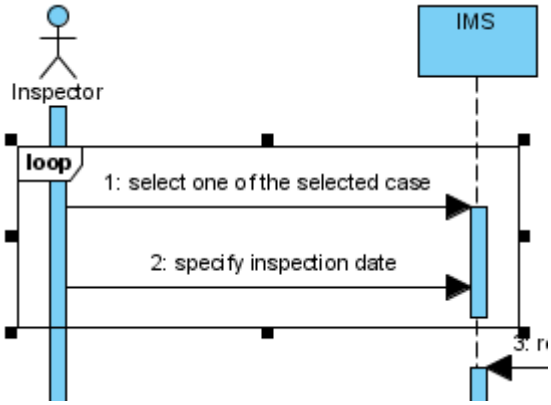
Creating combined fragment for messages

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.



Create combined fragment for messages

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

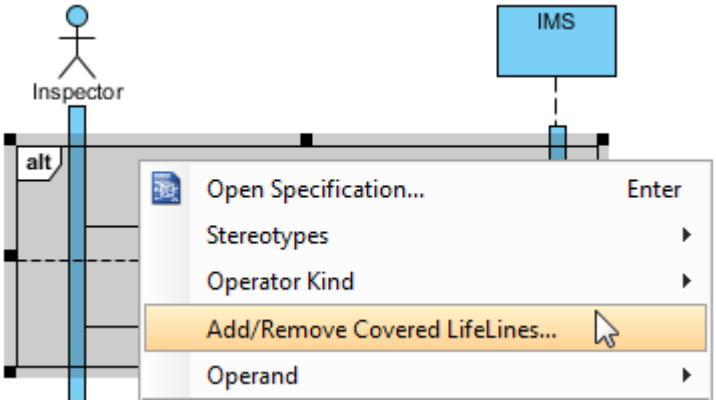


Combined fragment created

Adding/removing covered lifelines

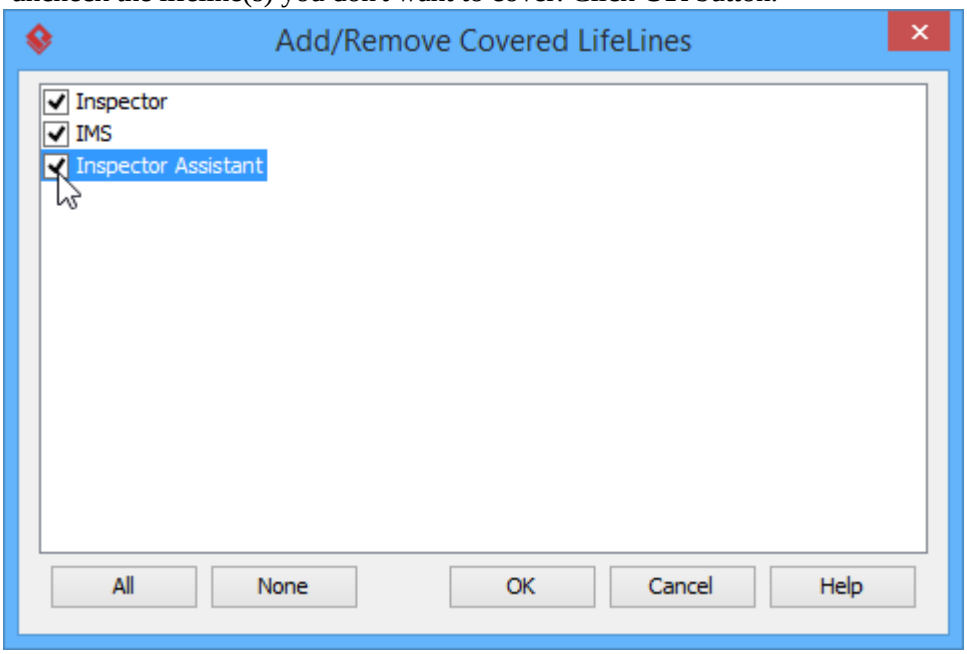
After you've created a combined fragment on the messages, you can add or remove the covered lifelines.

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



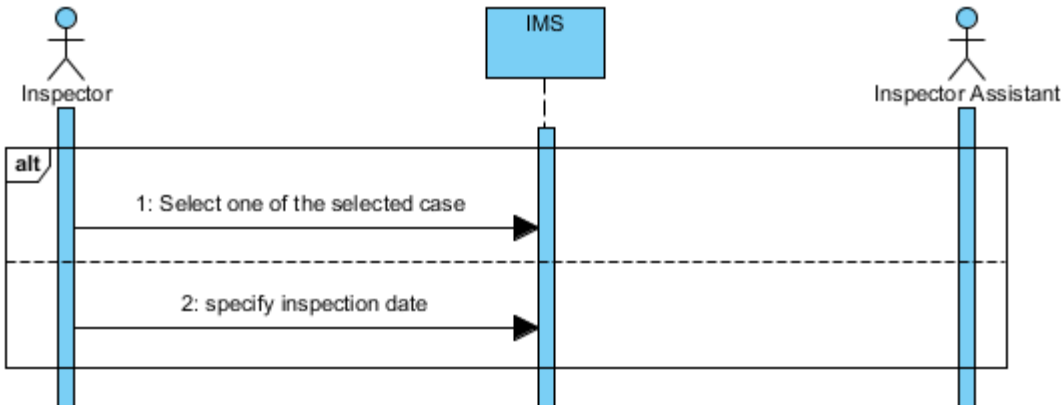
Add/Remove covered lifelines

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



*Check **Inspector Assistant***

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

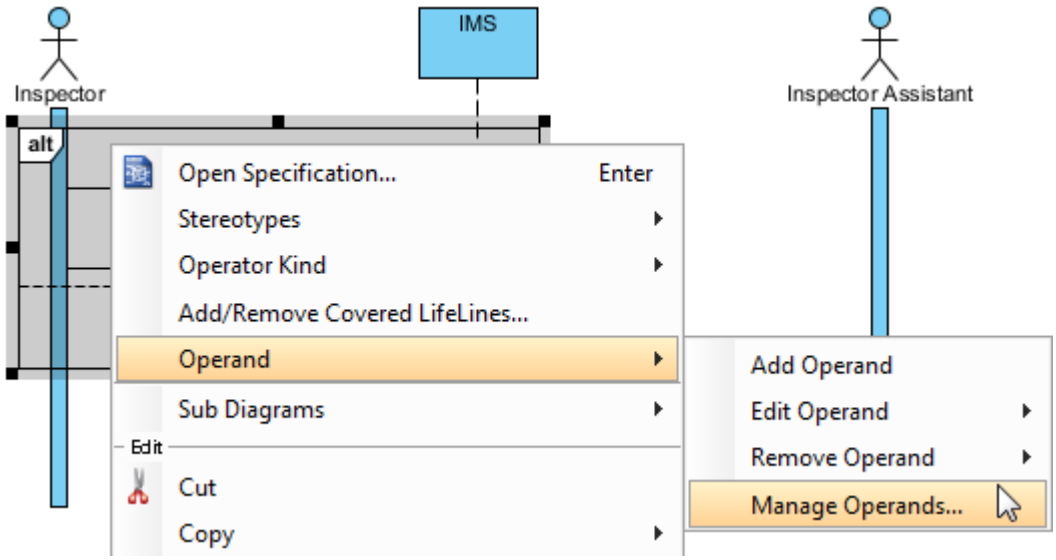


The area of covered lifelines is extended

Managing Operands

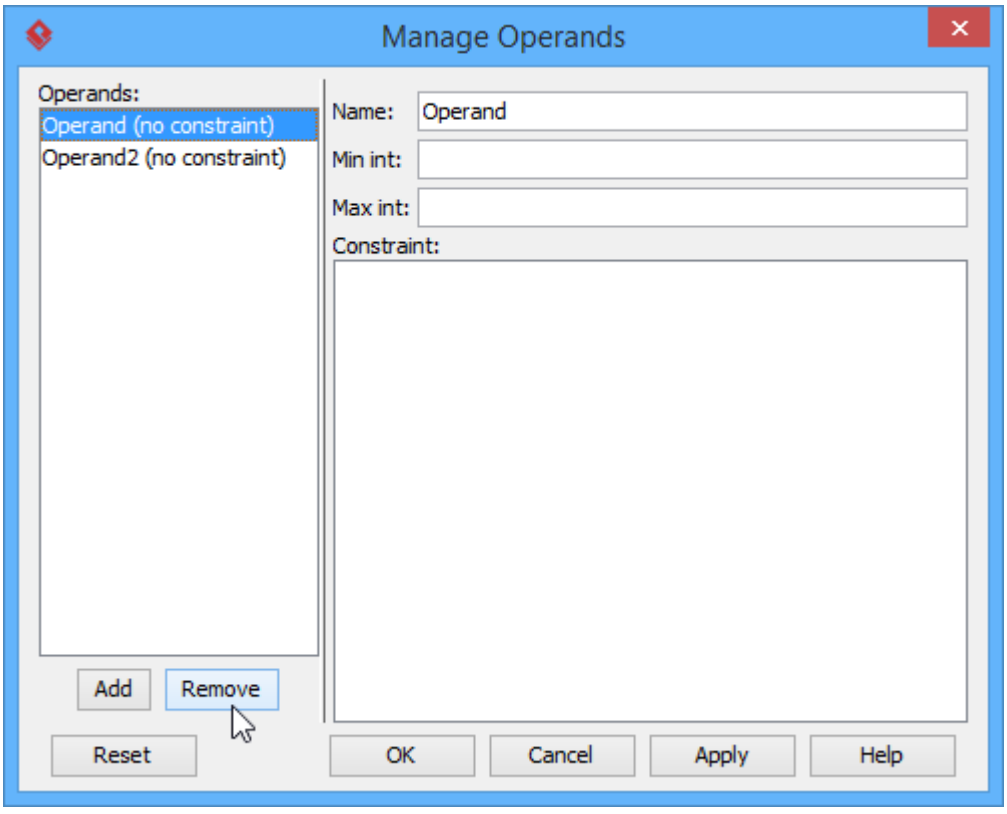
After you've created a combined fragment on the messages, you can also add or remove operand(s).

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



Manage operands

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

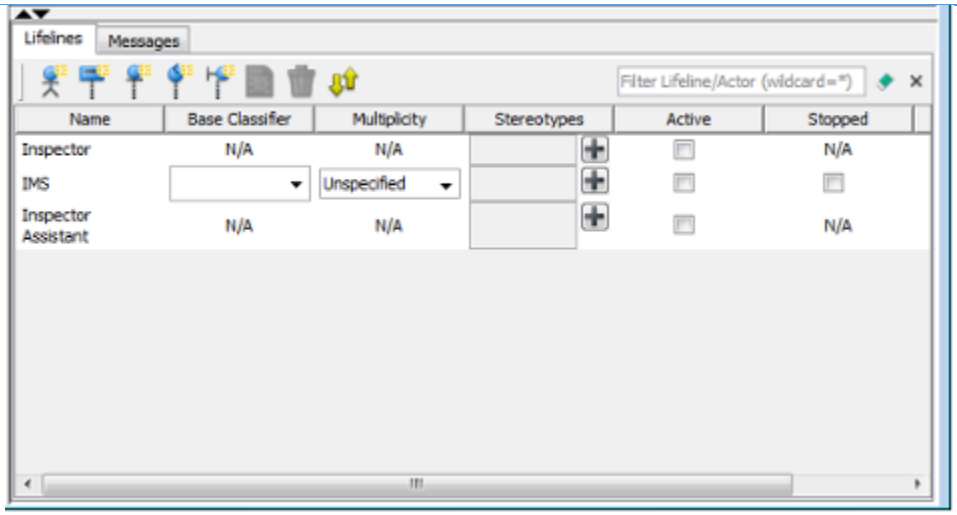


Remove Operand

3. 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.



The quick editor

Editing lifelines

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

Lifelines pane in quick editor

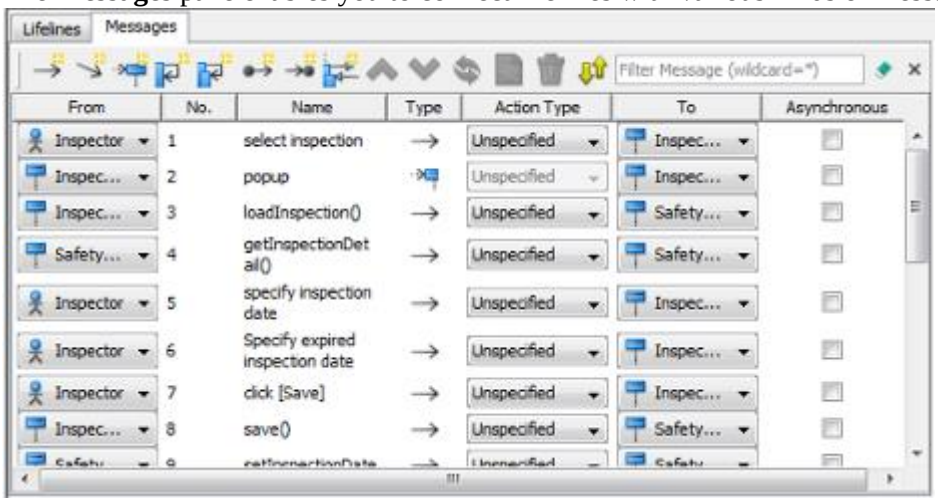
Button	Shortcut	Description
	Alt-Shift-A	To create an actor
	Alt-Shift-L	To create a general lifeline
	Alt-Shift-E	To create an <<entity>> lifeline
	Alt-Shift-C	To create a <<control>> lifeline
	Alt-Shift-B	To create a <<boundary>> lifeline
	Alt-Shift-O	To open the specification of the element chosen 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

Buttons in Lifelines pane

Editing messages

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



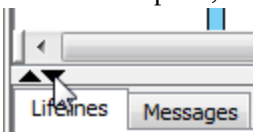
Messages pane in quick editor

Button	Shortcut	Description
	Alt-Shift-M	To create a message that connects actors/lifelines in diagram
	Alt-Shift-D	To create a duration message that connects actors/lifelines in diagram
	Alt-Shift-C	To create a create message that connects actors/lifelines in diagram
	Alt-Shift-S	To create a self message on an actor/lifeline in diagram
	Alt-Shift-R	To create a recursive message on an actor/lifeline in diagram
	Alt-Shift-F	To create a found message that connects to an actor/lifeline
	Alt-Shift-L	To create a lost message from an actor/lifeline
	Alt-Shift-E	To create a reentrant message that connects actors/lifelines in 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 in quick editor
	Ctrl-Del	To delete the message chosen in quick editor
	Ctrl-L	To link with the diagram, which cause the message to be selected when selecting a message in editor, and vice versa

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.

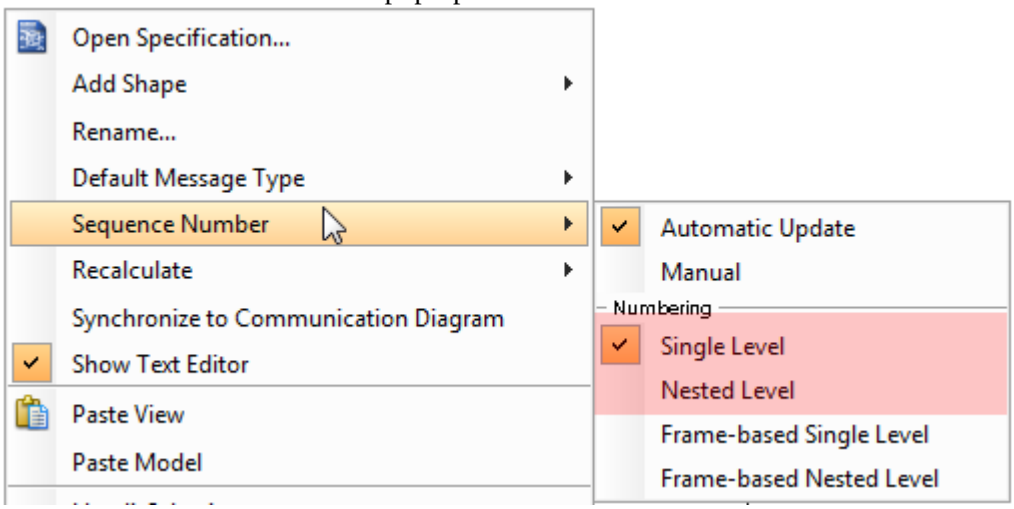
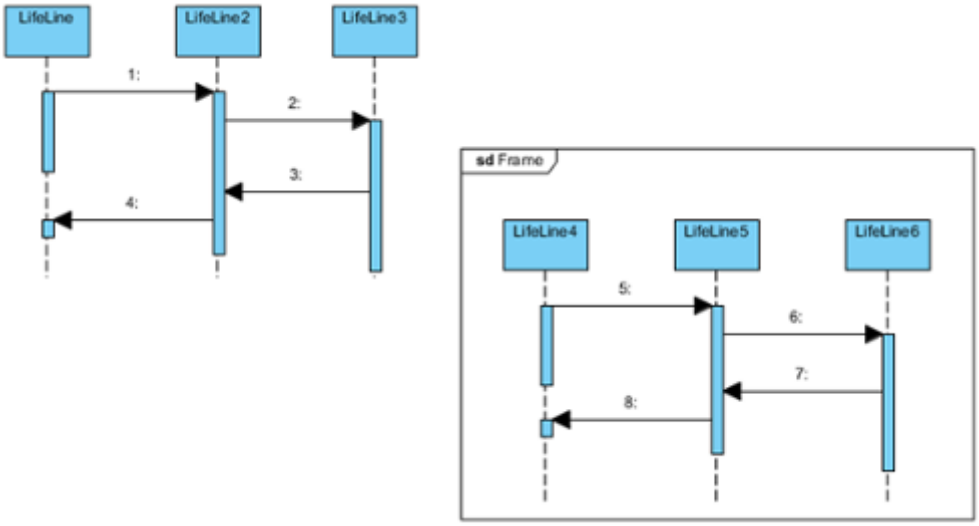


Diagram-based pop-up menu

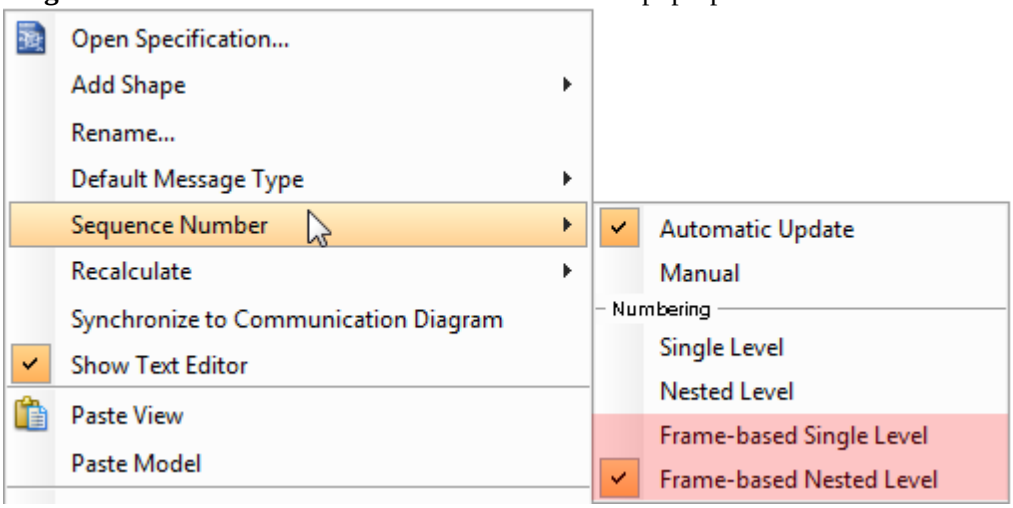
If you choose **Single Level**, all sequence messages will be ordered with integers on diagram base. On the other hand, if you choose **Nested Level**, all sequence messages will be ordered with decimal place on diagram base.



Single level

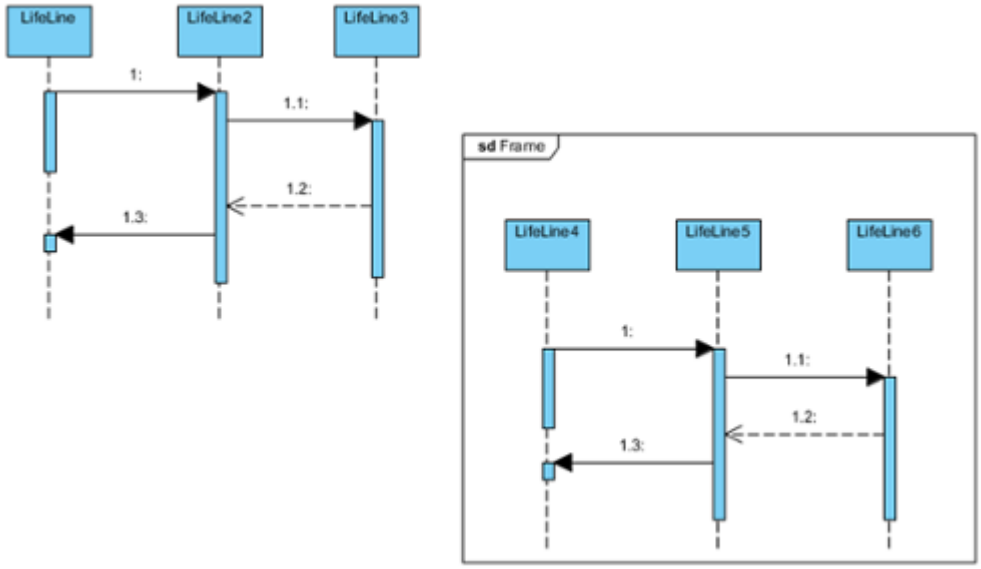
Frame-based sequence message

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.

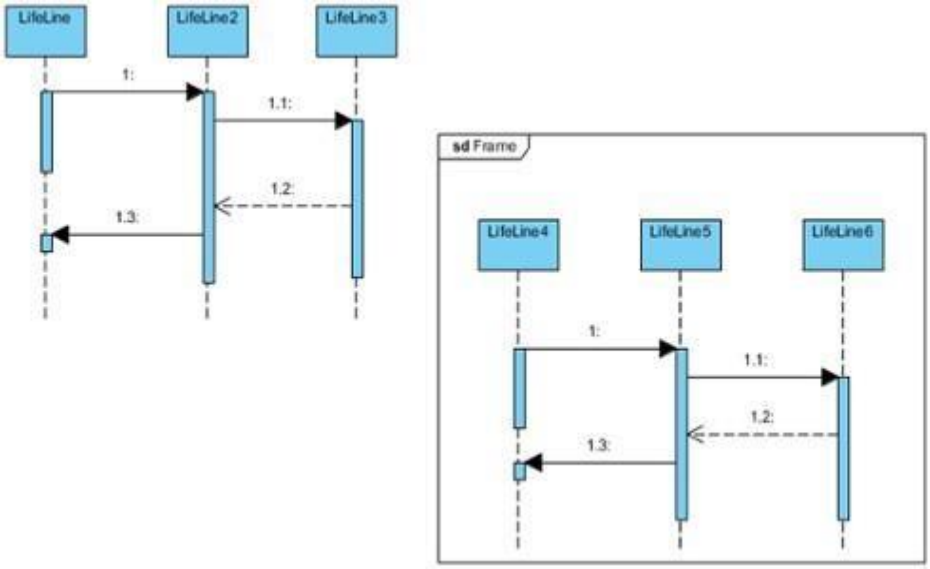


Frame-based pop-up menu

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.



Frame-based nested level

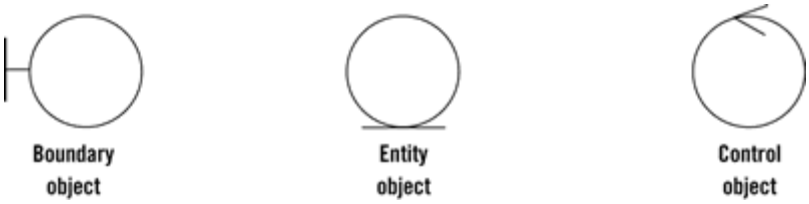


Robustness diagrams are written after use cases and before class diagrams. They help to identify the roles of use case steps. You can use them to **ensure your use cases are sufficiently robust** to represent usage requirements for the system you're building.

They involve:

1. Actors
2. Use Cases
3. ***Entities***
4. ***Boundaries***
5. ***Controls***

Whereas the Model-View-Controller pattern is used for user interfaces, the Entity-Control-Boundary Pattern (ECB) is used for systems. The following aspects of ECB can be likened to an abstract version of MVC, if that's helpful:



Entities *(model)*
Objects representing system data, often from the domain model.

Boundaries *(view/service collaborator)*
Objects that interface with system actors (e.g. a **user** or **external service**). Windows, screens and menus are examples of boundaries that interface with users.

Controls *(controller)*
Objects that mediate between boundaries and entities. These serve as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions. It is important to understand that you may decide to implement controllers within your design as something other than objects – many controllers are simple enough to be implemented as a method of an entity or boundary class for example.

Four rules apply to their communication:

1. Actors can only talk to boundary objects.
2. Boundary objects can only talk to controllers and actors.
3. Entity objects can only talk to controllers.
4. Controllers can talk to boundary objects and entity objects, and to other controllers, but not to actors

Communication allowed:

	Entity	Boundary	Control
Entity	X		X
Boundary			X
Control	X	X	X

STUDENT TASK-01

Develop a sequence diagram showing the interactions involved when a student registers for a course in a university. Courses may have limited enrollment, so the registration process must include checks that places are available. Assume that the student accesses an electronic course catalog to find out about available courses.

STUDENT TASK-02

Draw a sequence diagram that models the data processing involved when a customer withdraws cash from the ATM.

LAB-05

OBJECTIVE

After this lab students will be able to:

- Know the concepts underlining Class Model
- Model Class diagram and its application

PRE-LAB READING ASSIGNMENT

The concepts of class diagram

LAB RELATED CONTENT

A class diagram is a kind of UML diagram that shows the objects that are required and the relationships between them. Since it provides detailed information about the properties and interfaces of the classes, it can be considered as the main model and regard the other diagrams as supplementary models.

SOFTWARE TOOL

Visual paradigm

EXPERIMENTS

EXAMPLE EXPERIMENT-01

Draw a class diagram for online reservation system.

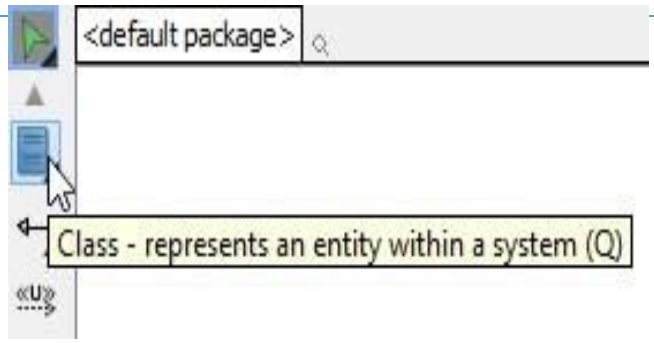
PROCEDURE

Perform the steps below to create a UML class diagram in Visual Paradigm.

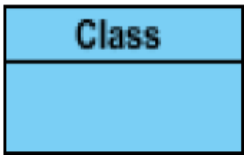
1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Class Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

Creating class

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



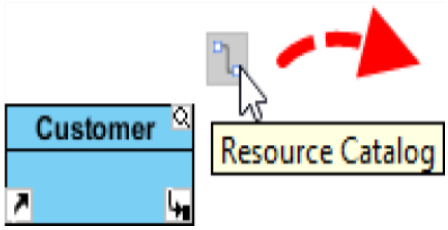
A class will be created.



Creating association

To create an associated class in a class diagram:

1. Move your mouse pointer over the source shape.
2. Press on the **Resource Catalog** button and drag it out.



3. Release the mouse button at the place where you want the class to be created. If you want to connect to an existing class, drop at that class. Otherwise, drop an empty space (either at the diagram background or container shape like package).
4. If you are connecting to an existing class, select Association from Resource Catalog. If you are creating a new class, select Association -> Class from Resource Catalog. If you want to create an aggregation or composition, select Aggregation -> Class or Composition -> Class instead.

5. If you are creating a new class, you should see the class now and it is connected to the source shape. Enter its name and press Enter to confirm editing.

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

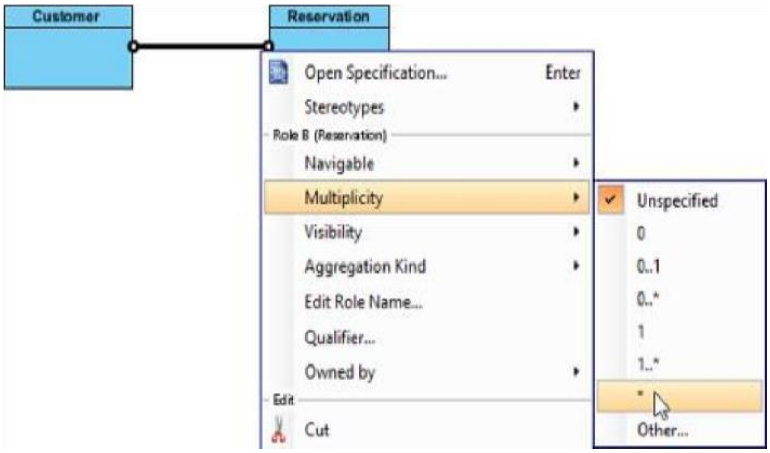
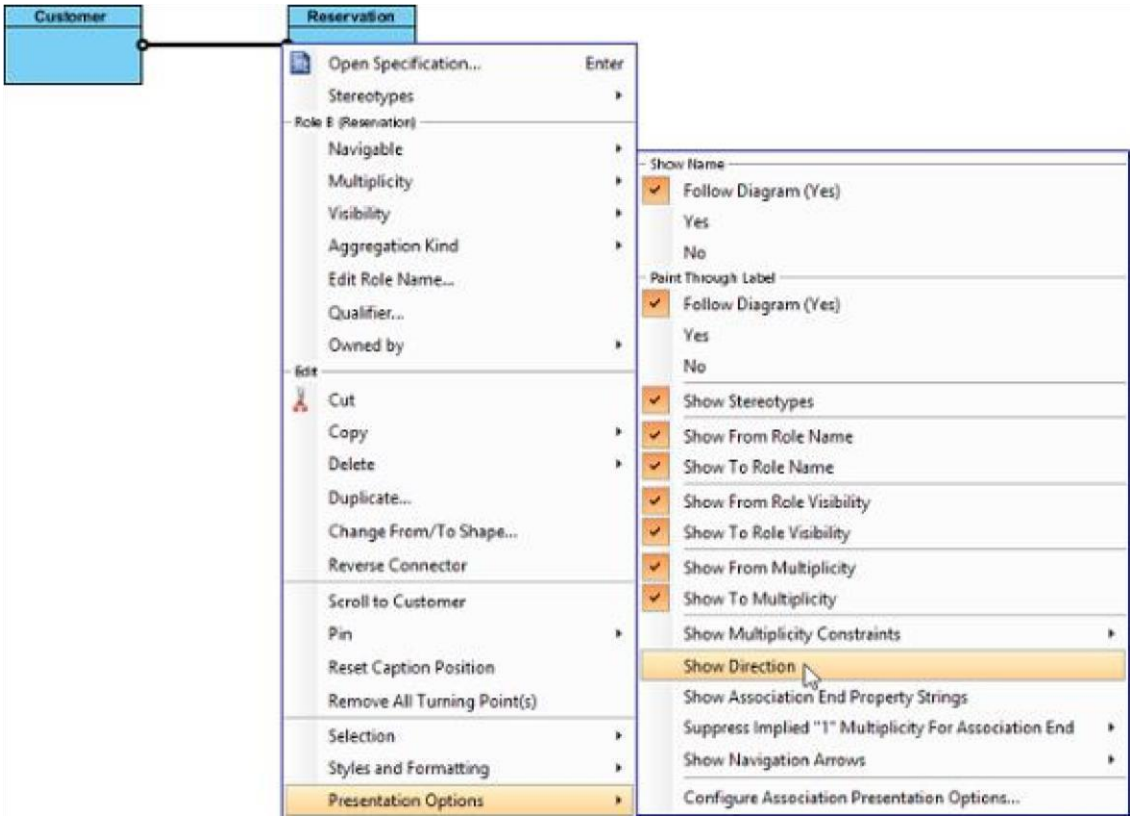
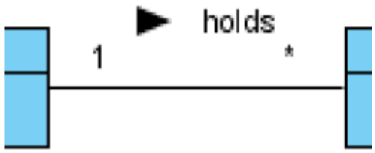


Figure 20: Edit multiplicity

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



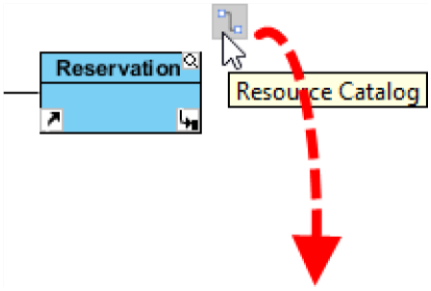
The direction arrow is shown beside the association.



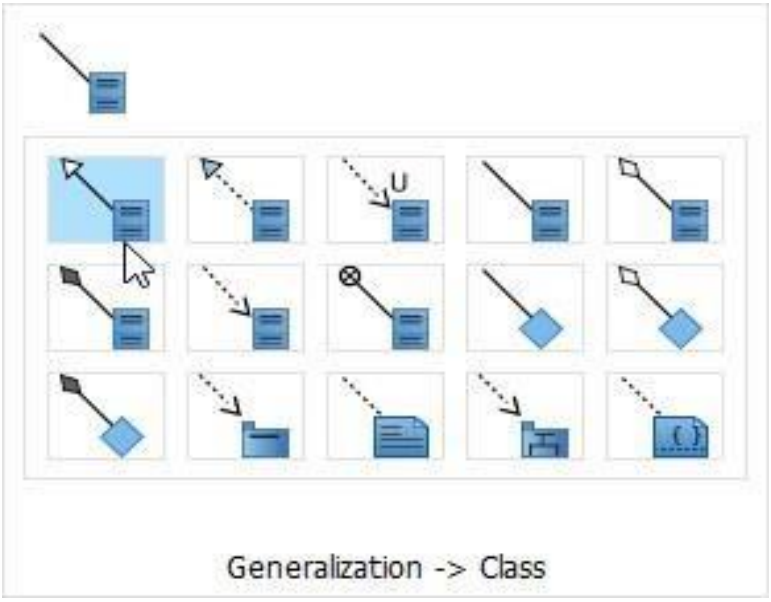
Creating generalization

To create a subclass:

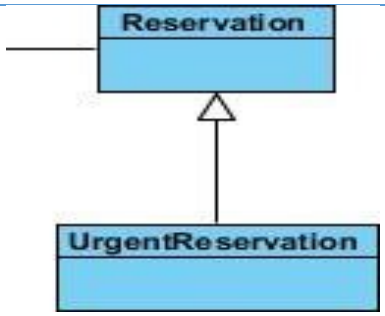
1. Move your mouse pointer over the super-class.
2. Press on the **Resource Catalog** button and drag it out.



3. Release the mouse button at the place where you want the subclass to be created. If you want to connect to an existing class, drop at that class. Otherwise, drop an empty space (either at the diagram background or container shape like package).
4. If you are connecting to an existing class, select **Generalization** from Resource Catalog. If you are creating a new class, select **Generalization -> Class** from Resource Catalog.

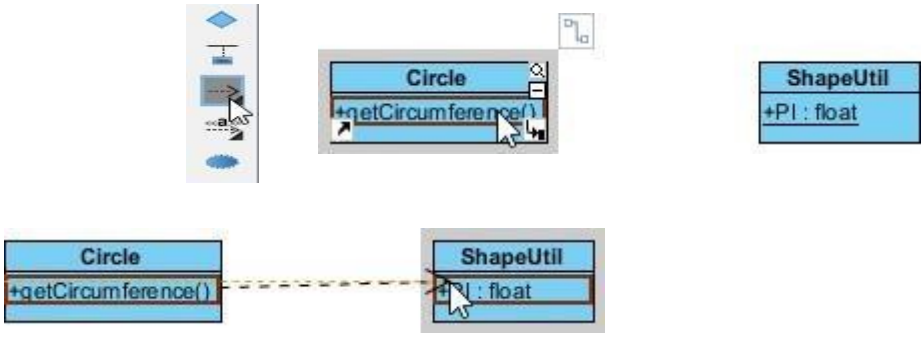


If you are creating a new class, you should see the class now and it is connected to the source shape with a generalization. Enter its name and press **Enter** to confirm editing.



Creating dependency from/to attribute/operation

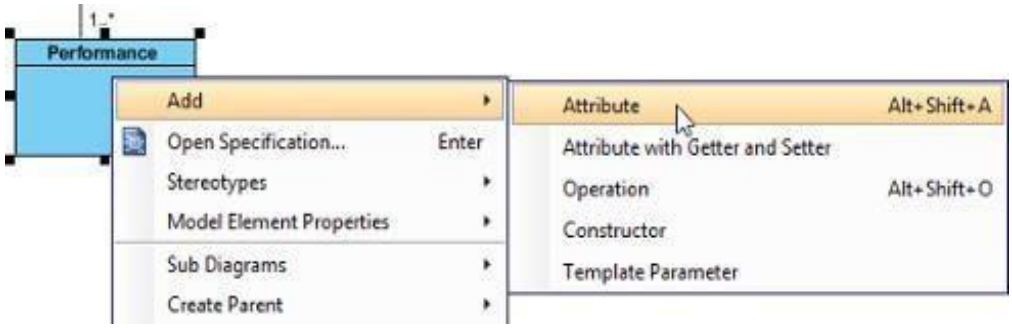
You can also add a dependency from and/or to an attribute or operation in class. To create such a dependency.



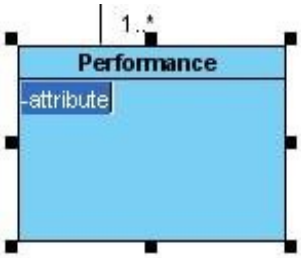
1. Select **Dependency** from the diagram toolbar.
2. Press on the source shape or a class member.
3. Drag to the target shape, or a class member.
4. Release the mouse button to create the dependency.

Creating attribute

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

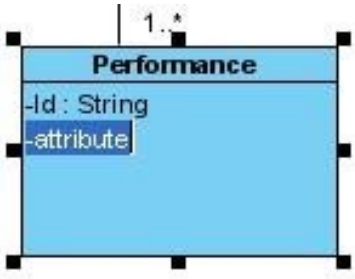


An attribute is created.



Creating attribute with enter key

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



STUDENT TASK-01

Draw a Class diagram of below explained system. Assume simple attributes and operations where necessary.

Hospital Management System:

Person could be associated with different *Hospitals*, and a Hospital could employ or serve multiple Persons. *Person* class has attributes *name, birthDate, gender* and *homeAddress*. Name represents full name and could be combined from title, first name, middle name, and last name. *Patient* class has derived attribute *age* which could be calculated based on her or his birth date and current date or hospital admission date. The *Patient* class inherits attributes from the *Person* class. Several inherited attributes *name*, *gender*, and *birthDate.* **Ward** is a division of a hospital or a suite of rooms shared by patients. In a hospital, there are a number of wards, each of which may be empty or have on it one or more **patients**. Each ward has a unique name and id. Wards are differentiated by **gender** of its patients, i.e. male wards and female wards. A ward can only have patients of the gender admitted to it. Ward and patient have constraint on Gender. Every ward has a fixed capacity, which is the maximum number of patients that can be on it at one time.

The doctors in the hospital are organized into **teams**. Each team has a unique name and is headed by a **consultant doctor.** Consultant doctor is the senior doctor who has completed all of his or her specialist training, residency and practices medicine in a clinic or hospital, in the specialty learned during residency. She or he can supervise fellows, residents, and medical students. The rest of the team are all junior doctors. Each doctor could be a member of no more than one team. Each **patient** is on a single ward and is under the care of a single team of doctors. A patient may be treated by any number of doctors but they must all be in the team that cares for the patient. A doctor can treat any number of patients.

LAB-06 (I)

OBJECTIVE

After this lab Students will be able to

- Understand the concepts underlining Activity Model
- Model Activity diagram and its application

PRE-LAB READING ASSIGNMENT

Basic concepts of activity diagram

LAB RELATED CONTENT

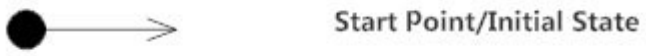
When to Use Activity Diagram

Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction. Typically, an event needs to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination, or how the events in a single use case relate to one another, in particular, use cases where activities may overlap and require coordination. It is also suitable for modeling how a collection of use cases coordinate to represent business workflows

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.

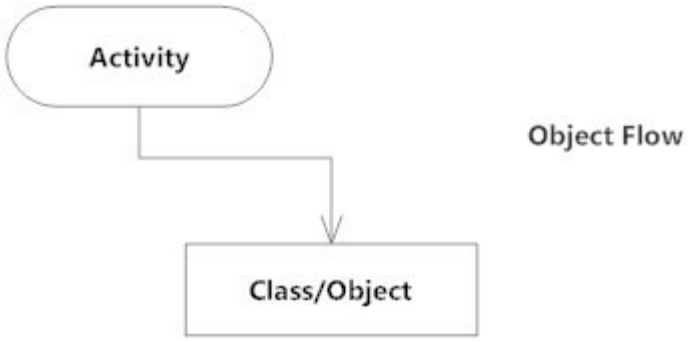
Action Flow

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."

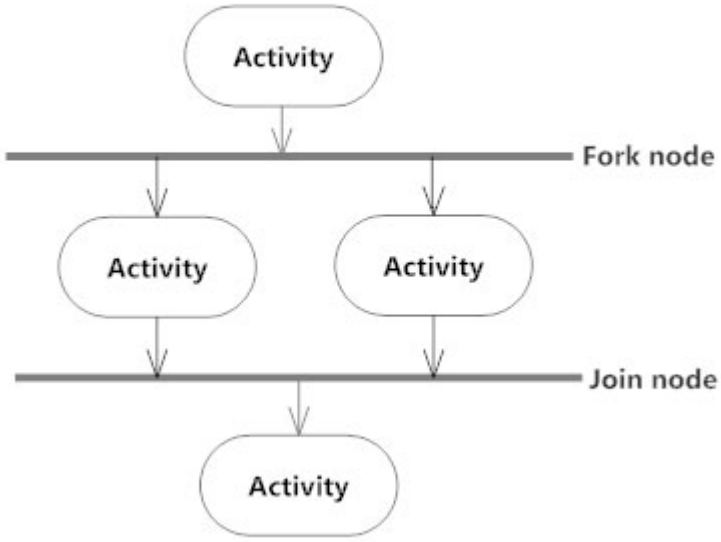
Guards

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.

Synchronization

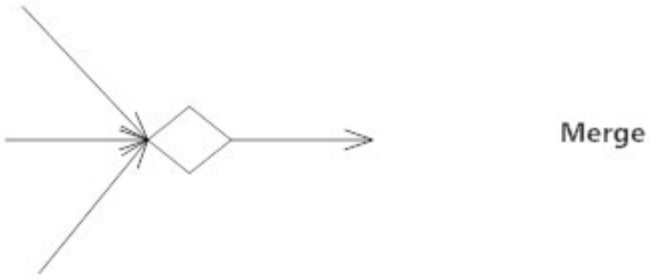


Time Event

This refers to an event that stops the flow for a time; an hourglass depicts it. ایک گھنٹہ کا گلاس اس کی عکاسی کرتا ہے۔

Merge Event

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

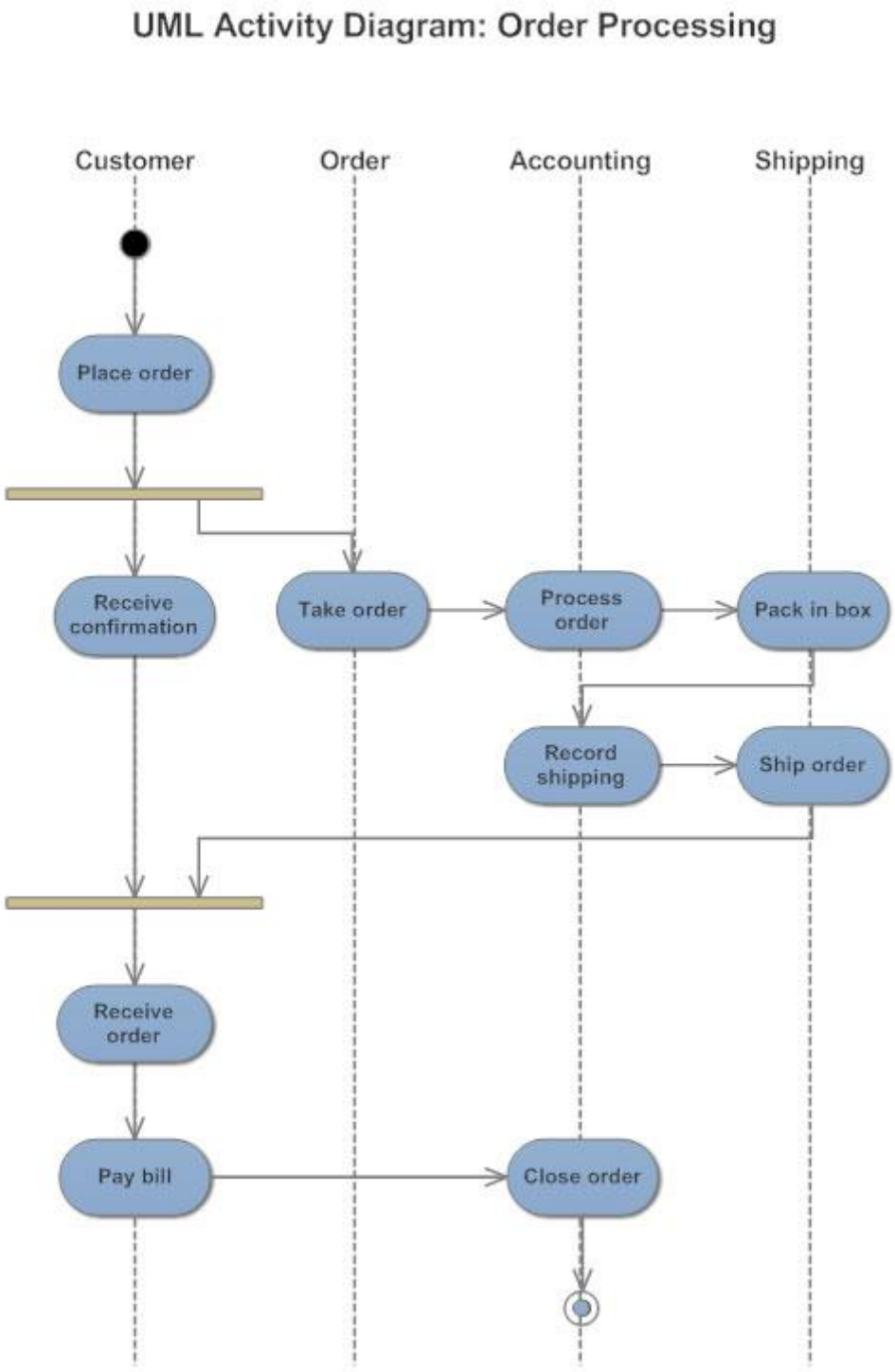


Sent and Received Signals

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 a sequence diagram. For example, an authorization of payment is needed before an order can be completed.

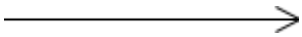
Swimlanes

Swimlanes group related activities into one column.



Final State or End Point

An arrow pointing to a filled circle nested inside another circle represents the final action state.

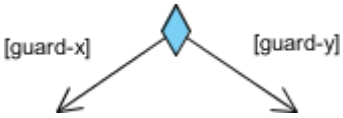
Notation Description	UML Notation
Activity Is used to represent a set of actions	
Action A task to be performed	
Control Flow Shows the sequence of execution	
Object Flow Show the flow of an object from one activity (or action) to another activity (or action).	
Initial Node Portrays the beginning of a set of actions or activities	
Activity Final Node Stop all control flows and object flows in an activity (or action)	

Object Node

Represent an object that is connected to a set of Object Flows

Decision Node

Represent a test condition to ensure that the control flow or object flow only goes down one path

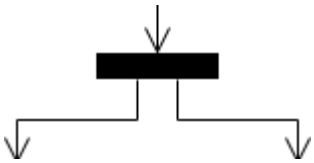


Merge Node

Bring back together different decision paths that were created using a decision-node.

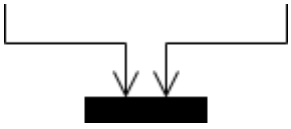
Fork Node

Split behavior into a set of parallel or concurrent flows of activities (or actions)



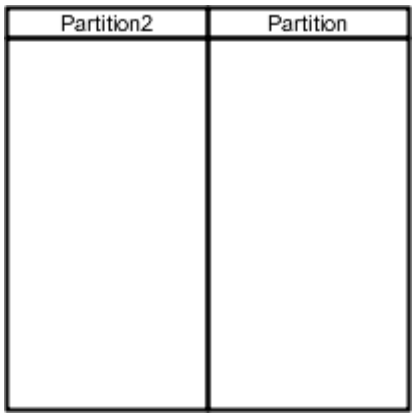
Join Node

Bring back together a set of parallel or concurrent flows of activities (or actions).



Swimlane and Partition

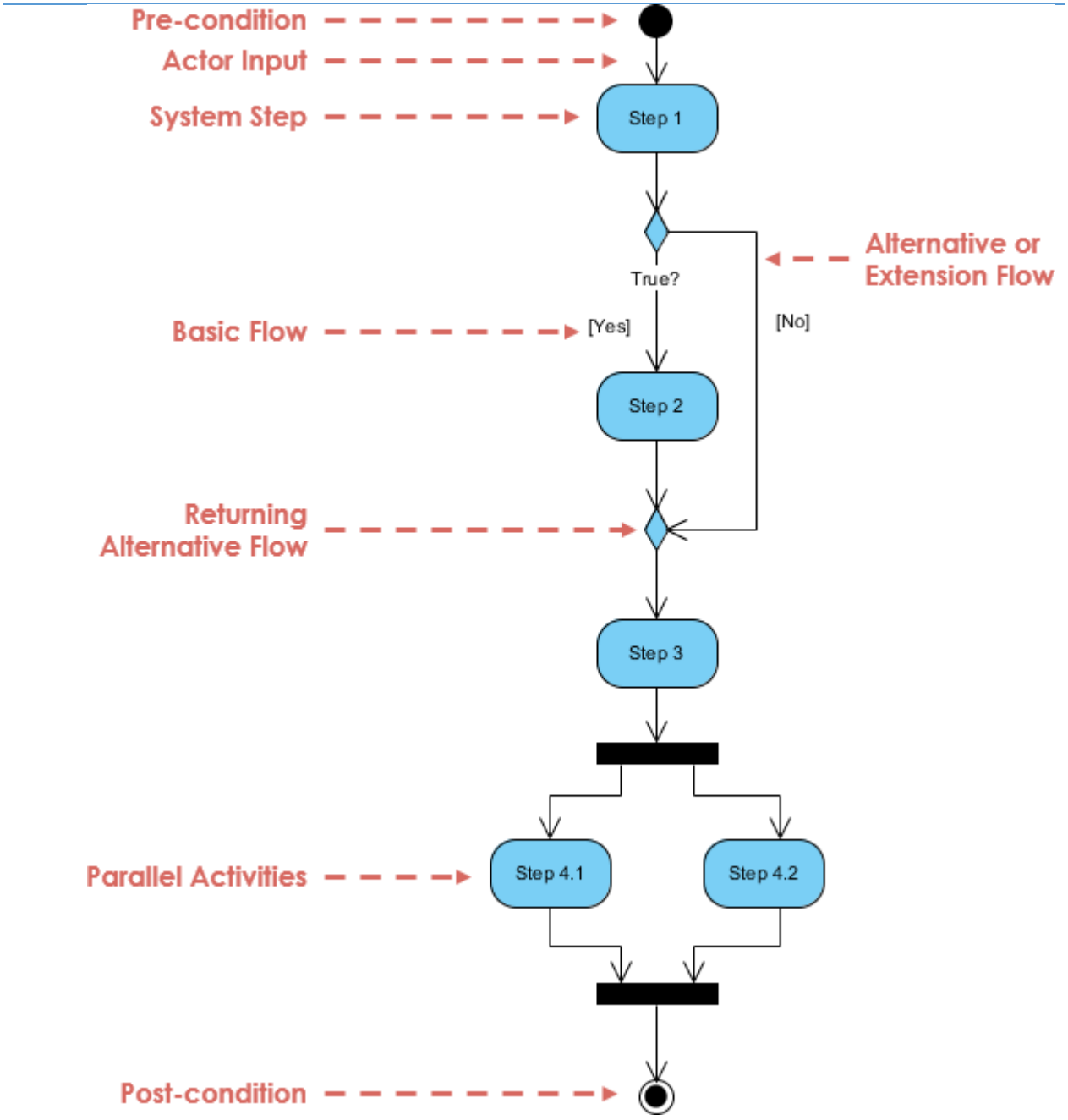
A way to group activities performed by the same actor on an activity diagram or to group activities in a single thread



1. Identify candidate use cases, through the examination of business workflows
2. Identify pre- and post-conditions (the context) for use cases
3. Model workflows between/within use cases
4. Model complex workflows in operations on objects
5. Model in detail complex activities in a high-level activity Diagram

Activity Diagram - Learn by Examples

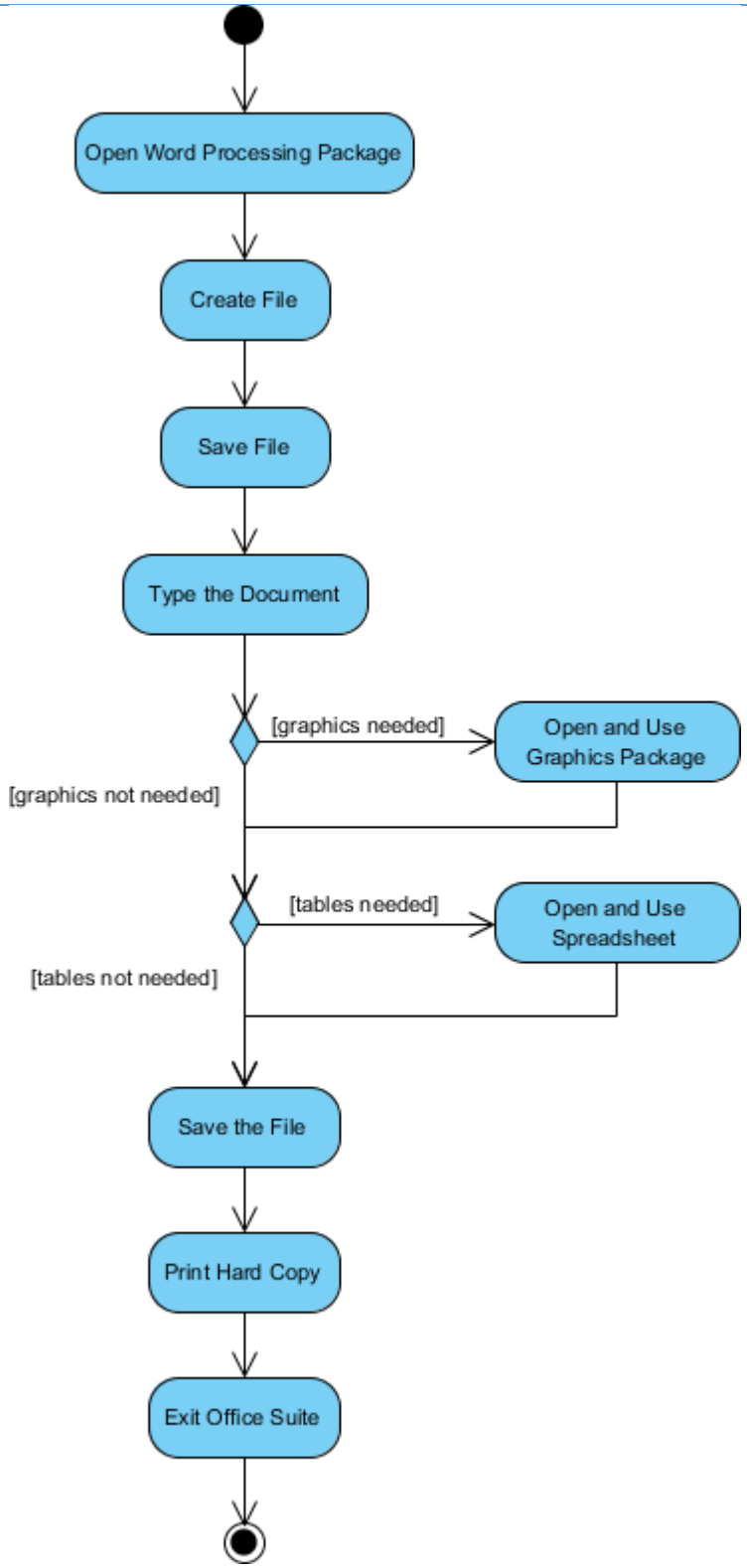
A basic activity diagram - flowchart like


Pre-condition
Actor Input
System Step
Step 1
True?
Alternative or
Extension Flow
Basic Flow
[Yes]
[No]
Step 2
Returning
Alternative Flow
Step 3
Parallel Activities
Step 4.1
Step 4.2
Post-condition

Activity Diagram - Modeling a Word Processor

The activity diagram example below describes the workflow for a word process to create a document through the following steps:

- Open the word processing package.
- Create a file.
- Save the file under a unique name within its directory.
- Type the document.
- If graphics are necessary, open the graphics package, create the graphics, and paste the graphics into the document.
- If a spreadsheet is necessary, open the spreadsheet package, create the spreadsheet, and paste the spreadsheet into the document.
- Save the file.
- Print a hard copy of the document.
- Exit the word processing package.


Open Word Processing Package
Create File
Save File
Type the Document
[graphics needed]
Open and Use Graphics Package
[graphics not needed]
[tables needed]
Open and Use Spreadsheet
[tables not needed]
Save the File
Print Hard Copy
Exit Office Suite

Activity Diagram Example - Process Order

Given the problem description related to the workflow for processing an order, let's model the description in visual representation using an activity diagram:

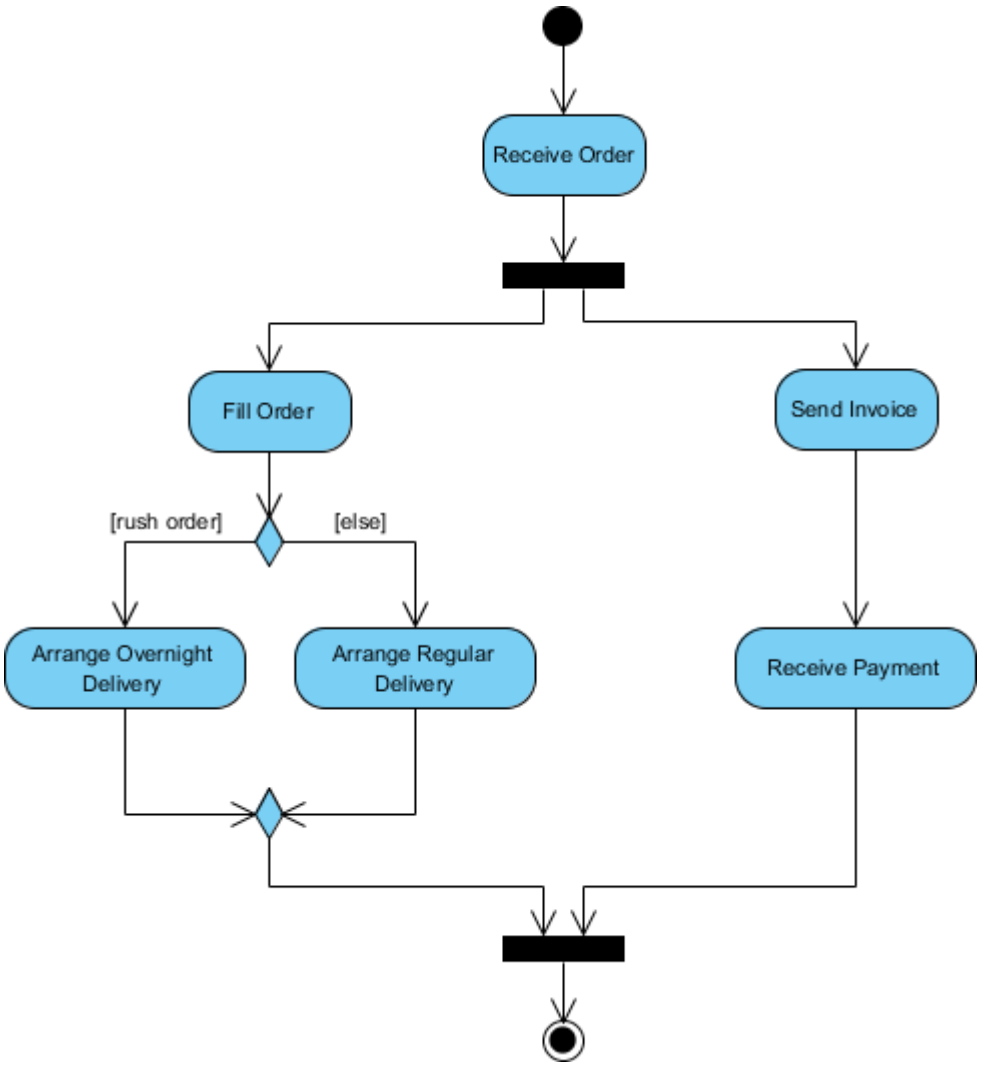
Process Order - Problem Description

Once the order is received, the activities split into two parallel sets of activities. One side fills and sends the order while the other handles the billing.

On the Fill Order side, the method of delivery is decided conditionally. Depending on the condition either the Overnight Delivery activity or the Regular Delivery activity is performed.

Finally the parallel activities combine to close the order.

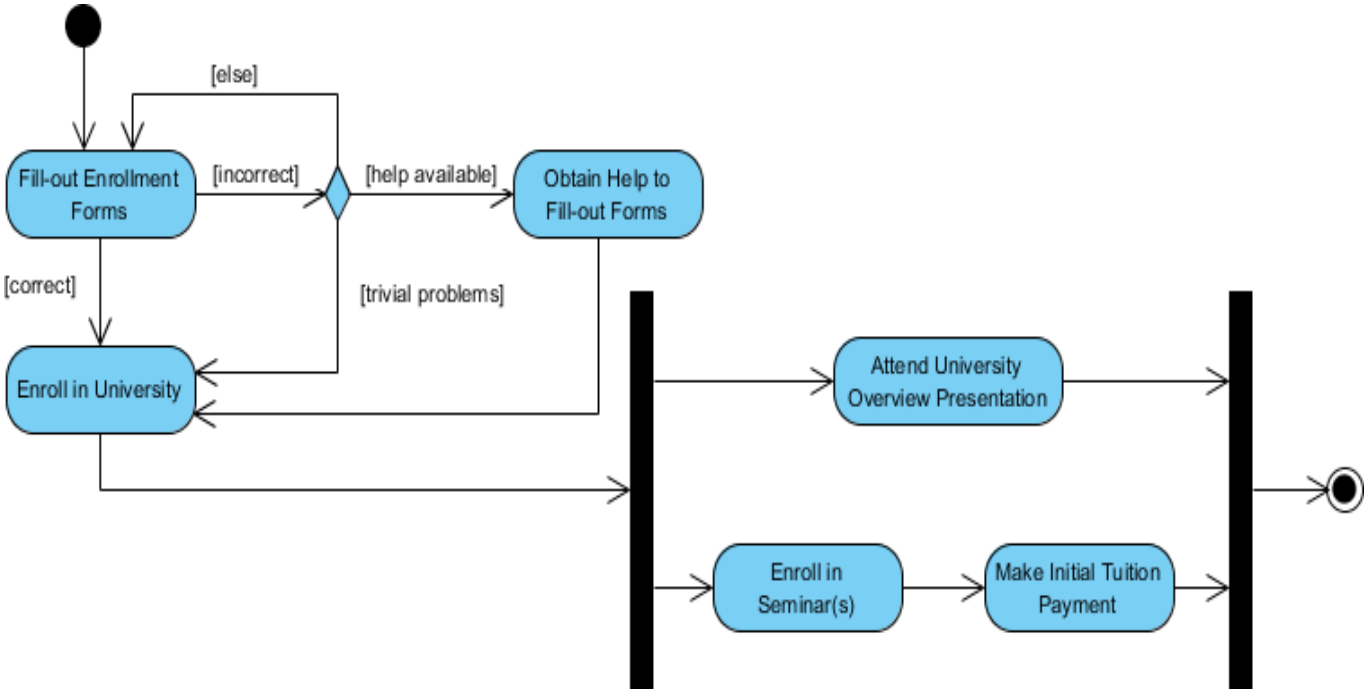
The activity diagram example below visualize the flow in graphical form.



Activity Diagram Example - Student Enrollment

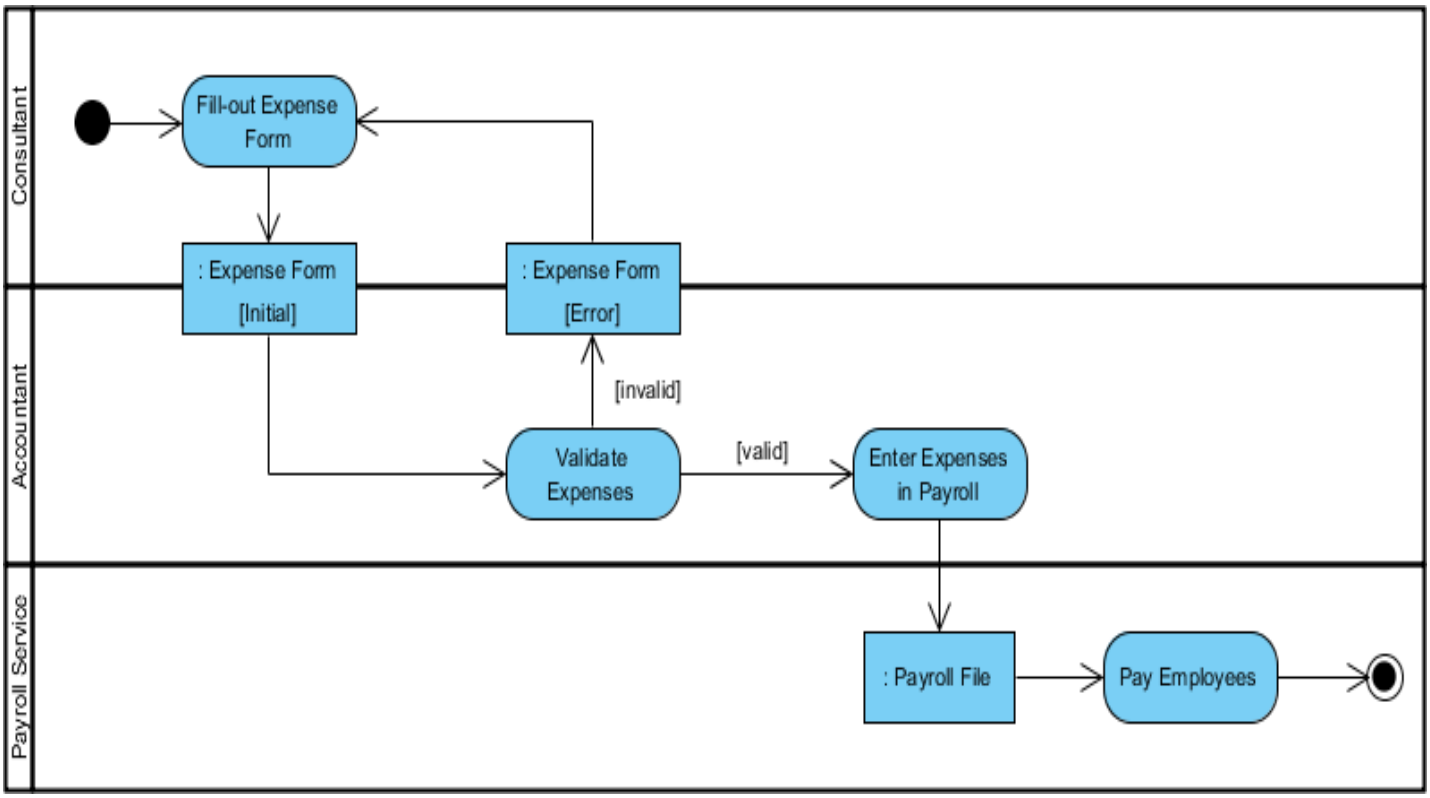
This UML activity diagram example describes a process for student enrollment in a university as follows:

- An applicant wants to enroll in the university.
- The applicant hands a filled out copy of Enrollment Form.
- The registrar inspects the forms.
- The registrar determines that the forms have been filled out properly.
- The registrar informs student to attend in university overview presentation.
- The registrar helps the student to enroll in seminars
- The registrar asks the student to pay for the initial tuition.



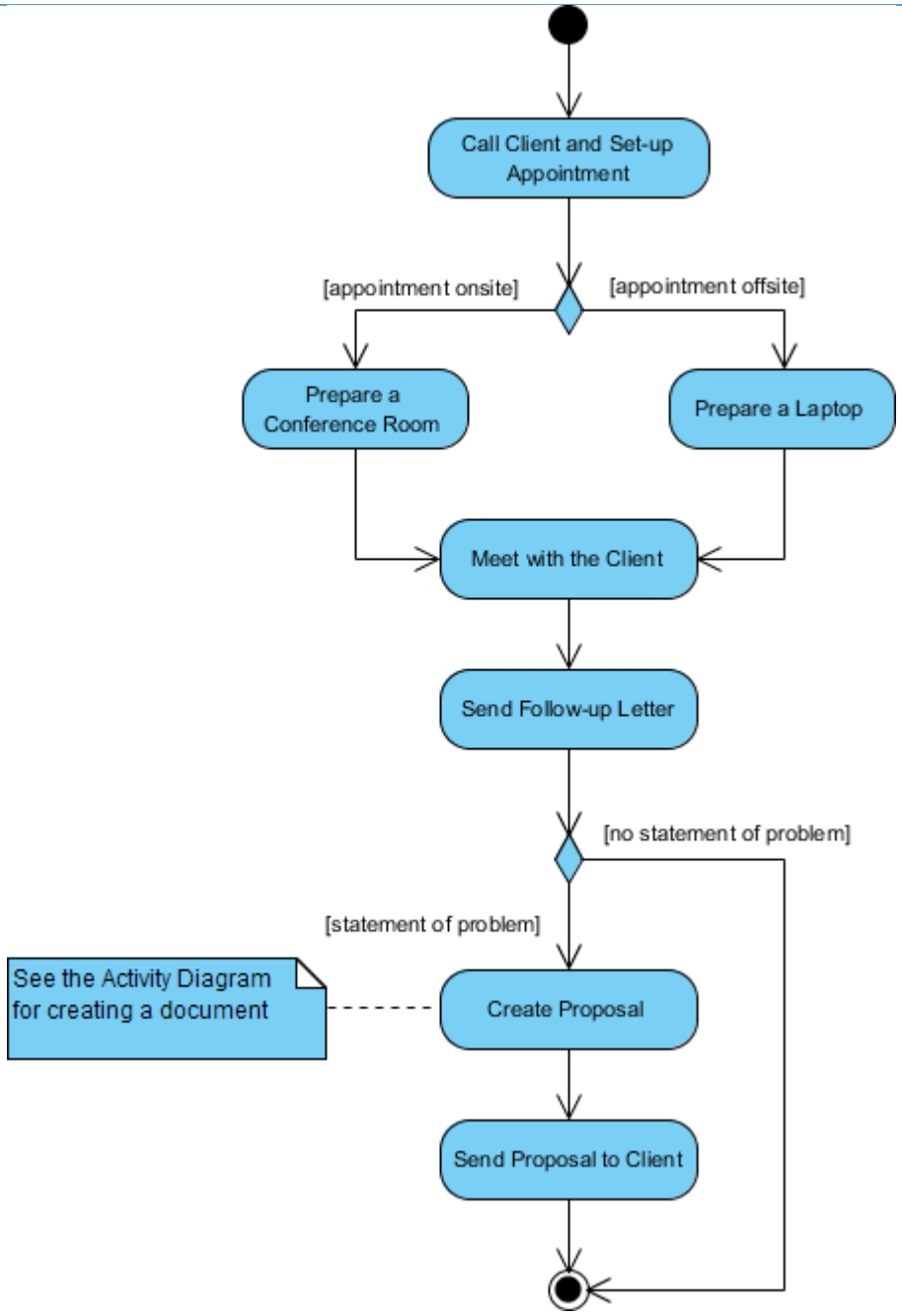
Activity Diagram - Swinlane

A swimlane is a way to group activities performed by the same actor on an activity diagram or activity diagram or to group activities in a single thread. Here is an example of a swinlane activity diagram for modeling Staff Expenses Submission:

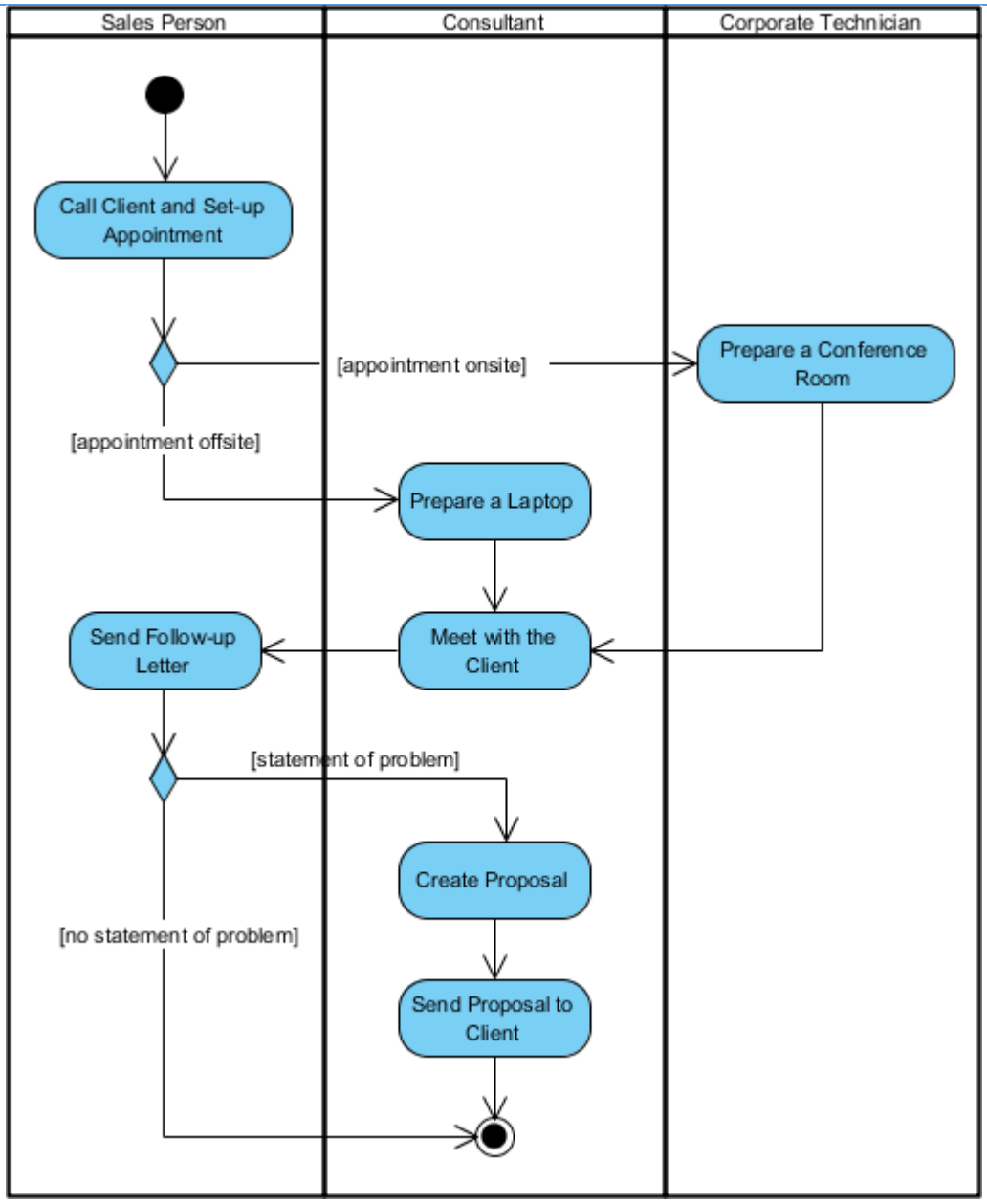


Swinlane and Non-Swinlane Activity Diagram

The activity diagram example below describes the business process for meeting a new client using an activity Diagram without swinlane.



This figure below describes the business process for meeting a new client using an activity Diagram with swinlane.



Activity Diagram Notation Summary

Activity diagrams can be thought of as the object oriented equivalent of flow charts and data flow diagrams. They are used to describe how activities happen and flow of control from one activity to other. Starting and stopping points in diagrams are shown using filled in circles. Activities are shown in **boxes**. Decisions are shown as **diamonds**. Activities can also be done at the same time. This is shown in the diagram by the **vertical black bars**.

SOFTWARE TOOL

Visual paradigm

EXPERIMENTS

EXAMPLE EXPERIMENT-01

Draw an activity diagram for online form submission system.

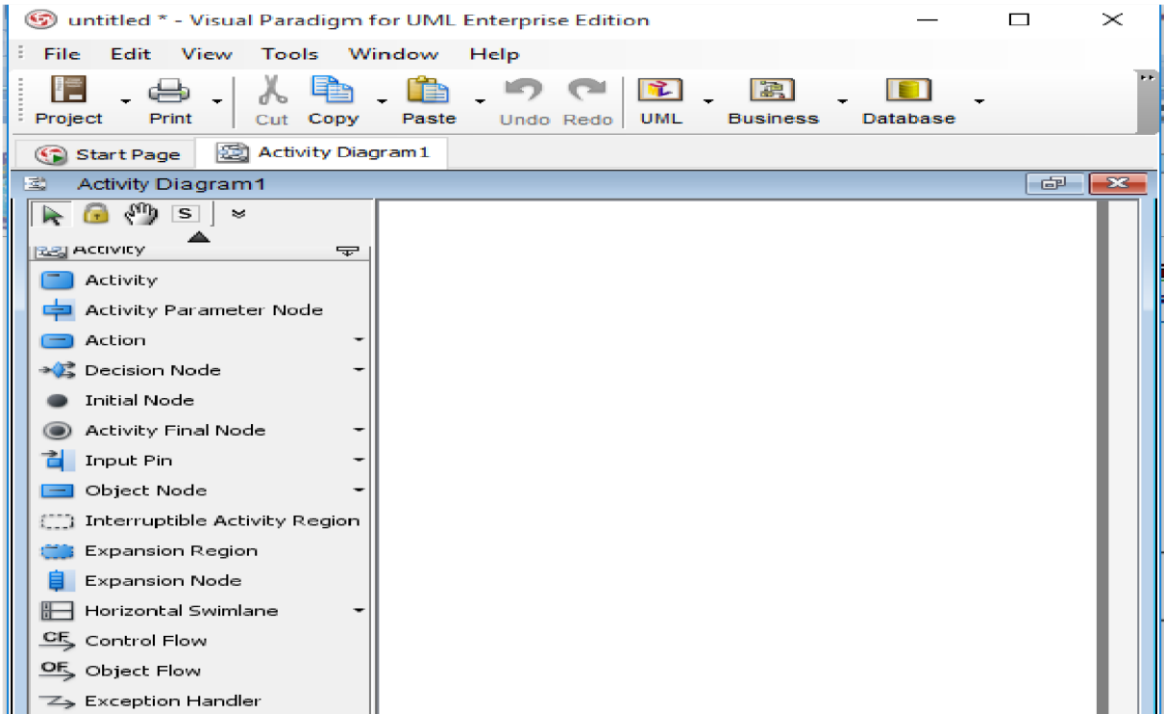
PROCEDURE

Perform the steps below to create a UML Activity diagram in Visual Paradigm.

1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Activity Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.

Tool Box support for Activity Diagram

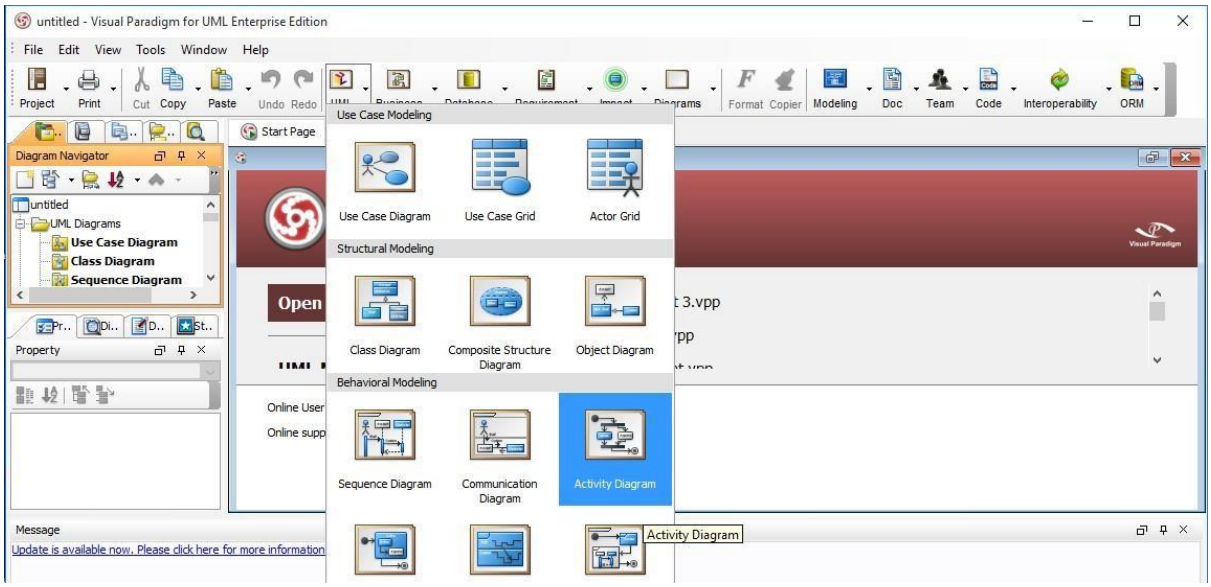
Tool box contain all the required model elements required for drawing activity diagram on the drawing panel.



Example of Activity Diagram

A person wants to attend a Seminar on health and safety talk. Firstly he or she has to complete the required form. If person made major mistakes while filling the form, he can get help from the office.

5. Click **OK**.



Whereas if mistakes done by him/her are minor then he/she is requested to complete it accurately. Once his/her form is accepted on the list, a person needs to attend a health and safety talk as well as check equipment and rent anything they don't have.

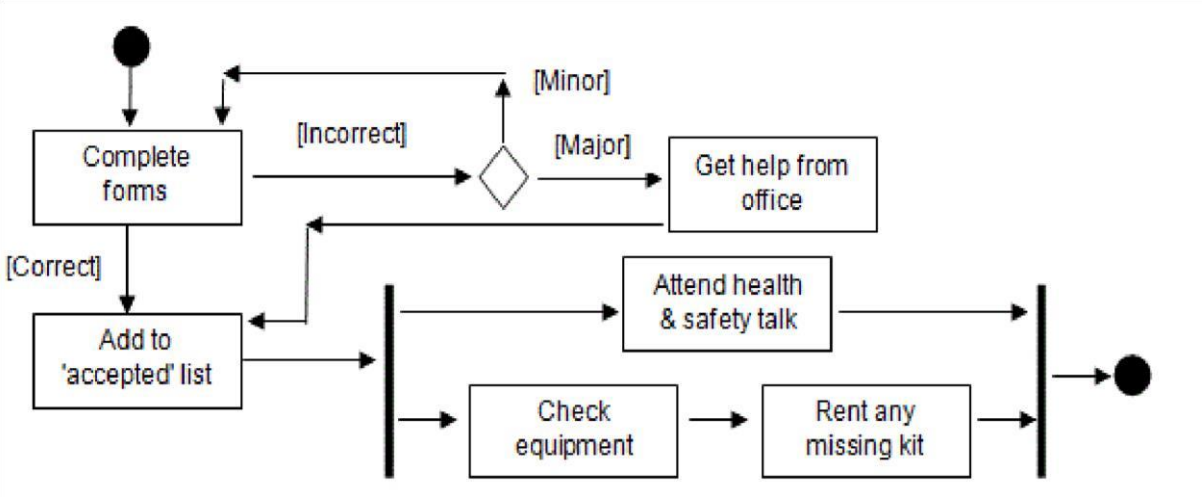


Figure 21: Activity Diagram of Attending Seminar

STUDENT TASK-01

Draw an Activity diagram of below explained system.

Sometimes, patients who are suffering from mental health problems may be a danger to others or to themselves. They may therefore have to be detained against their will in a hospital so that treatment can be administered. Such detention is subject to strict legal safeguards—for example, the decision to detain a patient must be regularly reviewed so that people are not held indefinitely without good reason.

STUDENT TASK-02

Based on your experience with a bank ATM, draw an activity diagram that models the data processing involved when a customer withdraws cash from the machine.

STUDENT TASK-03

You have been asked to develop a system that will help with planning large-scale events and parties such as weddings, graduation celebrations, birthday parties, etc. Using an activity diagram, model the process context for such a system that shows the activities involved in planning a party (booking a venue, organizing invitations, etc.) and the system elements that may be used at each stage.

LAB-06 (II)

Working with State Machine Models

After this lab students will be able to:

- Familiarize with the concepts underlining State Machine Model
- How to model State Machine diagram and its application

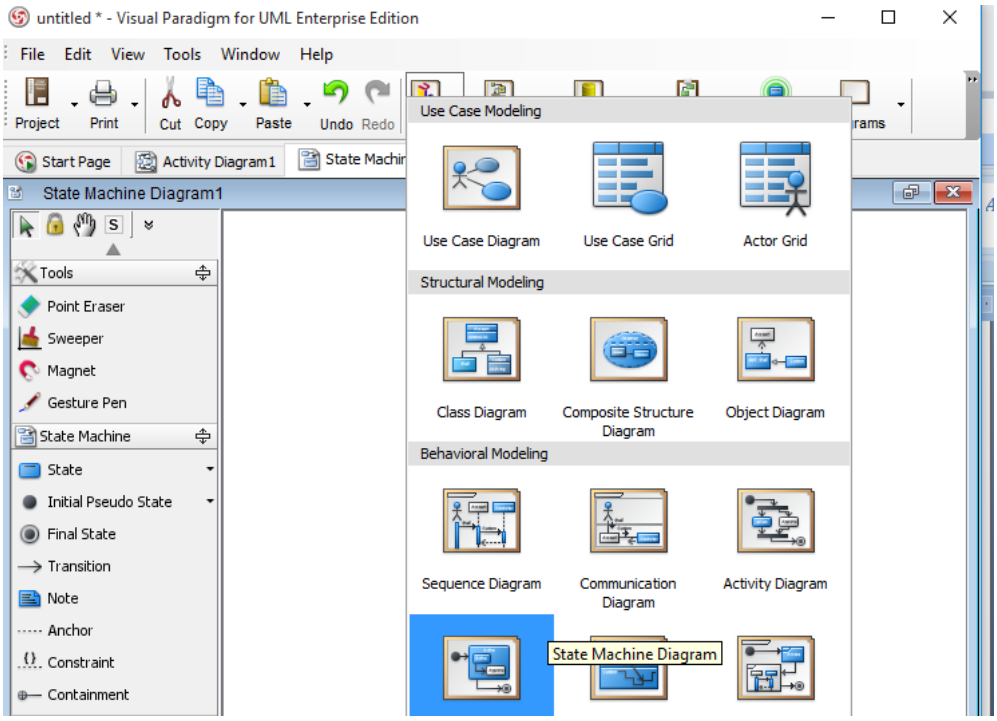
State Machine Diagram

State Machine diagrams model the behaviour of the system in response to external and internal events. They show the system's responses to stimuli so are often used for modelling real-time systems. State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another. State charts are an integral part of the UML and are used to represent state machine models.

Creating State Machine diagram

Perform the steps below to create a UML State Machine diagram in Visual Paradigm.

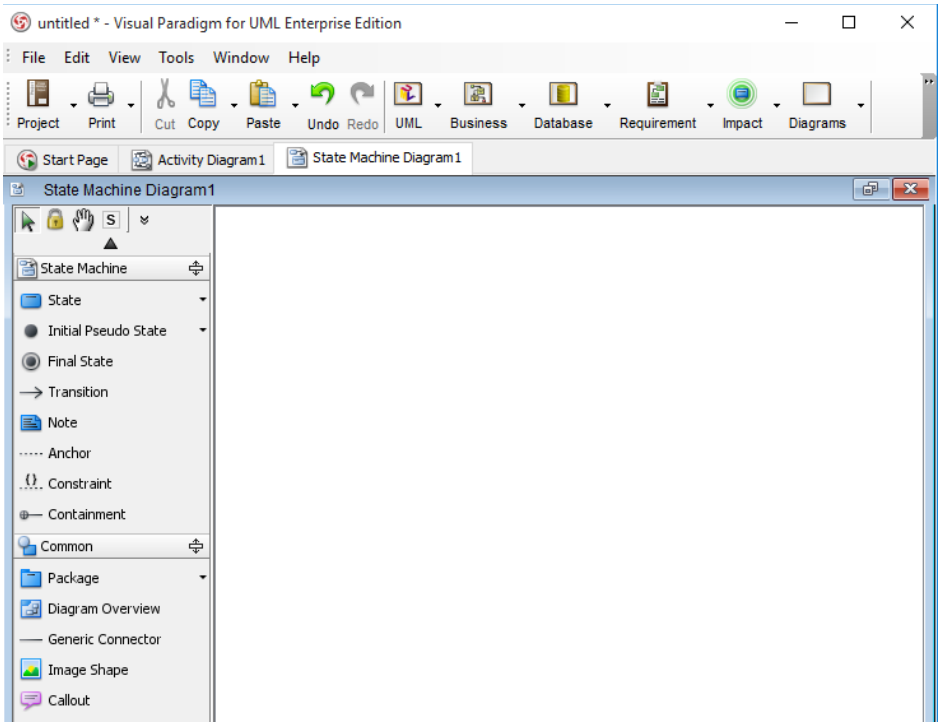
1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **State Machine Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.



Creating State Machine diagram

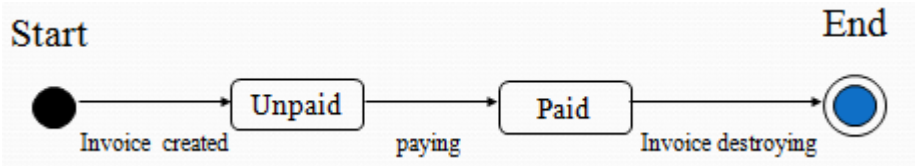
Tool Box support for State Machine Diagram

Tool box contain all the required model elements required for drawing state machine diagram on the drawing panel.

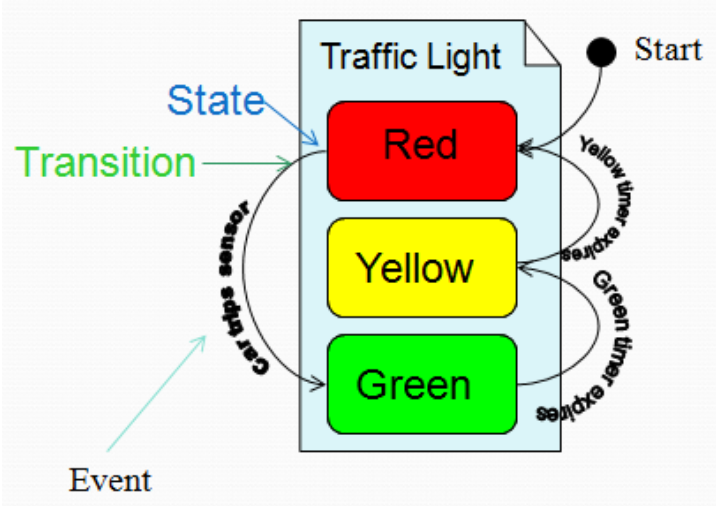


Model elements for State Machine Diagram

Examples of State Machine Diagram



State Machine Model of Paying Bill



State Machine Model of Traffic Light Signal

Lab Task 1

Draw State Machine diagram of below explained systems.

a. An automatic washing machine that has different programs for different types of clothes.
b. The software for a DVD player
c. A telephone answering system that records incoming messages and displays the number of accepted messages on an LED. The system should allow the telephone customer to dial in from any location, type a sequence of numbers (identified as tones) and play any recorded messages

LAB-07 (I)

OBJECTIVE

After this lab students will be able to

- Familiarize with the concepts underlining **Deployment** Model
- Model **Deployment** diagram and its application

PRE-LAB READING ASSIGNMENT

Basic concepts of deployment diagram

LAB RELATED CONTENT

A deployment diagram shows the assignment of concrete software artifacts (such as executable files) to computational nodes (something with processing services). It shows the deployment of software elements to the physical architecture and the communication (usually on a network) between physical elements. Deployment diagrams are useful to communicate the physical or deployment architecture. Deployment diagrams are used for describing the hardware components where software components are deployed

SOFTWARE TOOL

Visual paradigm

EXPERIMENTS

EXAMPLE EXPERIMENT-01

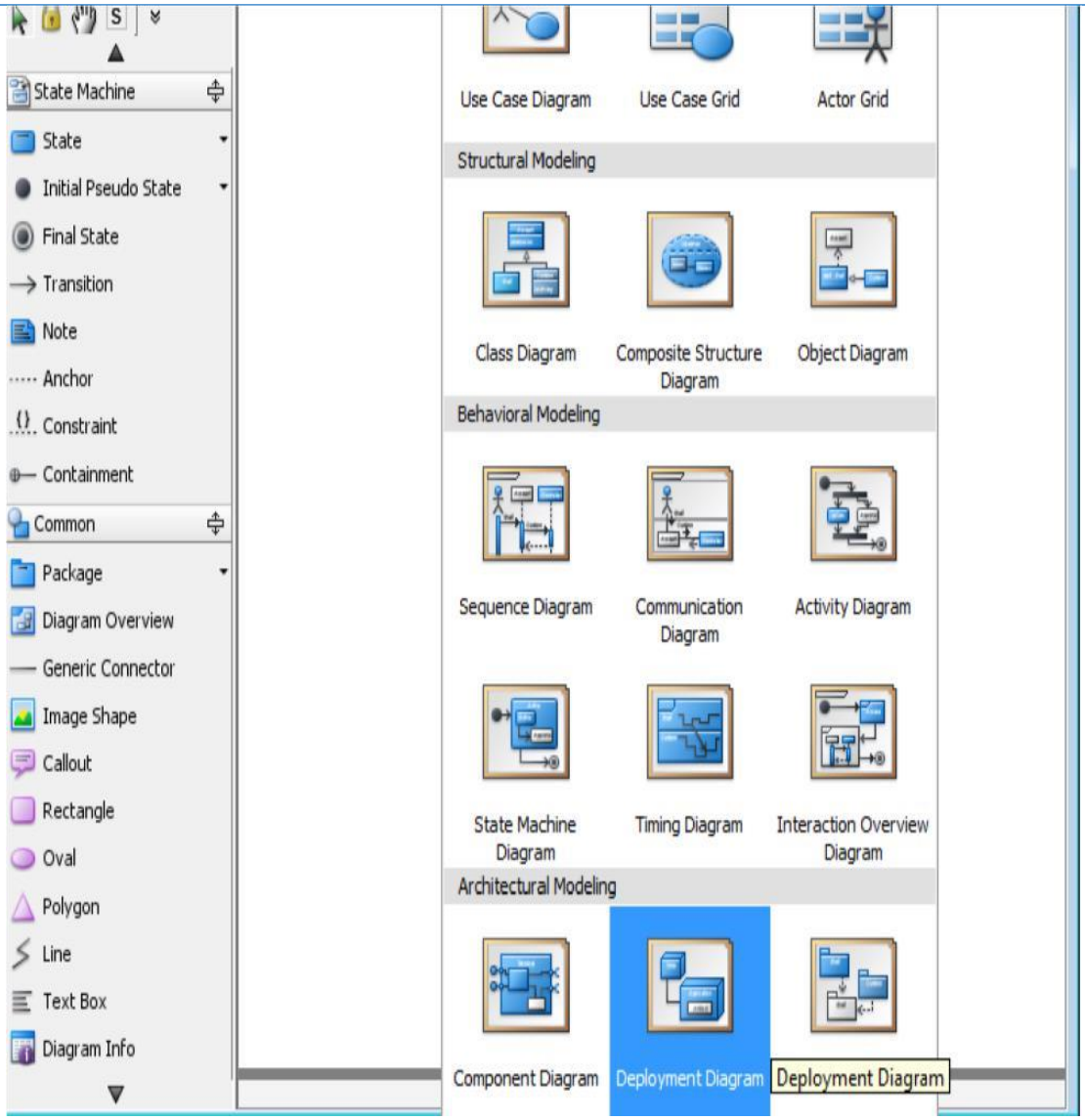
Draw a deployment diagram for any known system.

PROCEDURE

Creating Deployment diagram

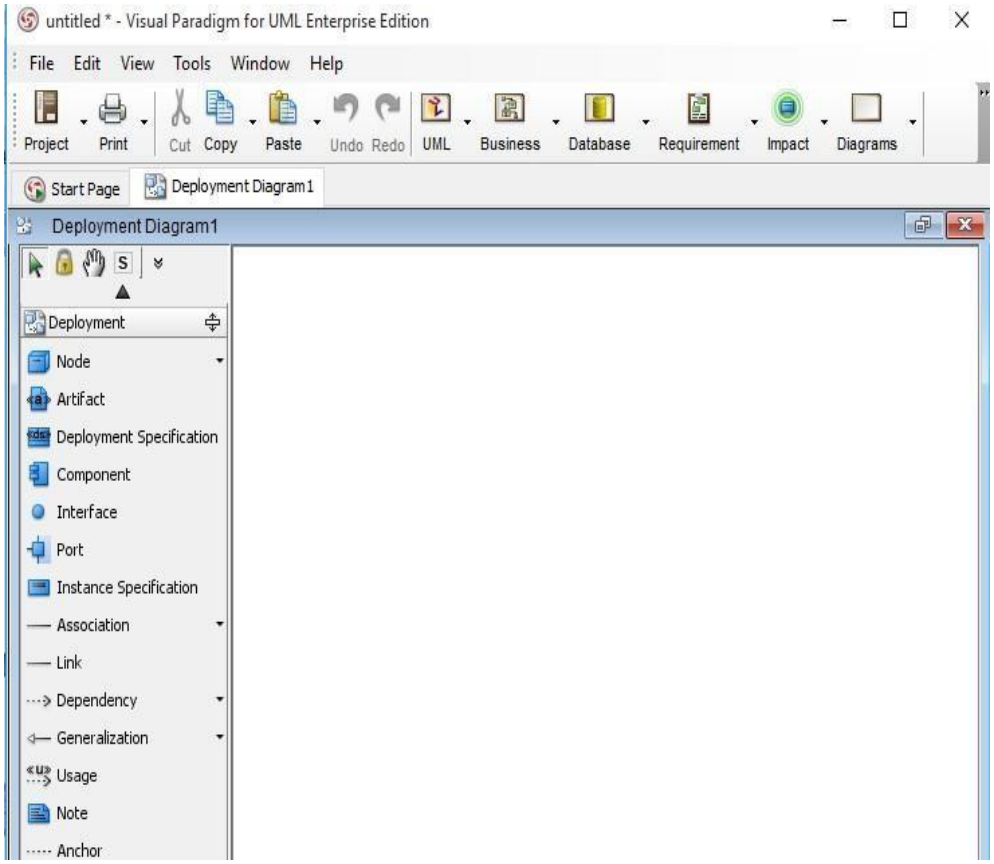
Perform the steps below to create a UML **Deployment** diagram in Visual Paradigm.

1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Deployment Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.



Tool Box support for Deployment Diagram

Tool box contain all the required model elements required for drawing **Deployment** diagram on the drawing panel



Elements of Deployment Diagram

The basic element of a deployment diagram is a node, of two types:

1. **Device node** (or device): A physical (e.g., digital electronic) computing resource with processing and memory services to execute software, such as a typical computer or a mobile phone

2. **Execution Environment Node (EEN):** This is a software computing resource that runs within an outer node (such as a computer) and which itself provides a service to host and execute other executable software elements. **For example:**

    - An operating system (OS) is software that hosts and executes programs
    - A virtual machine (VM, such as the Java or .NET VM) hosts and executes programs
    - A database engine (such as PostgreSQL) receives SQL program requests and executes them, and hosts/executes internal stored procedures (written in Java or a proprietary language)
    - A web browser hosts and executes JavaScript, Java applets, Flash, and other executable technologies

Examples of Deployment Diagram

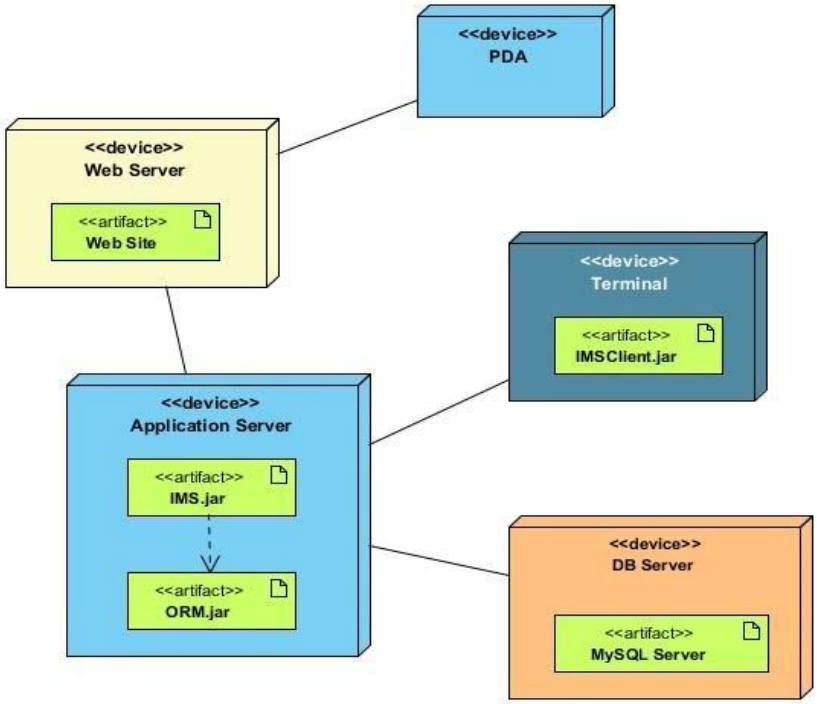


Figure 22: Deployment Diagram of Java based Application

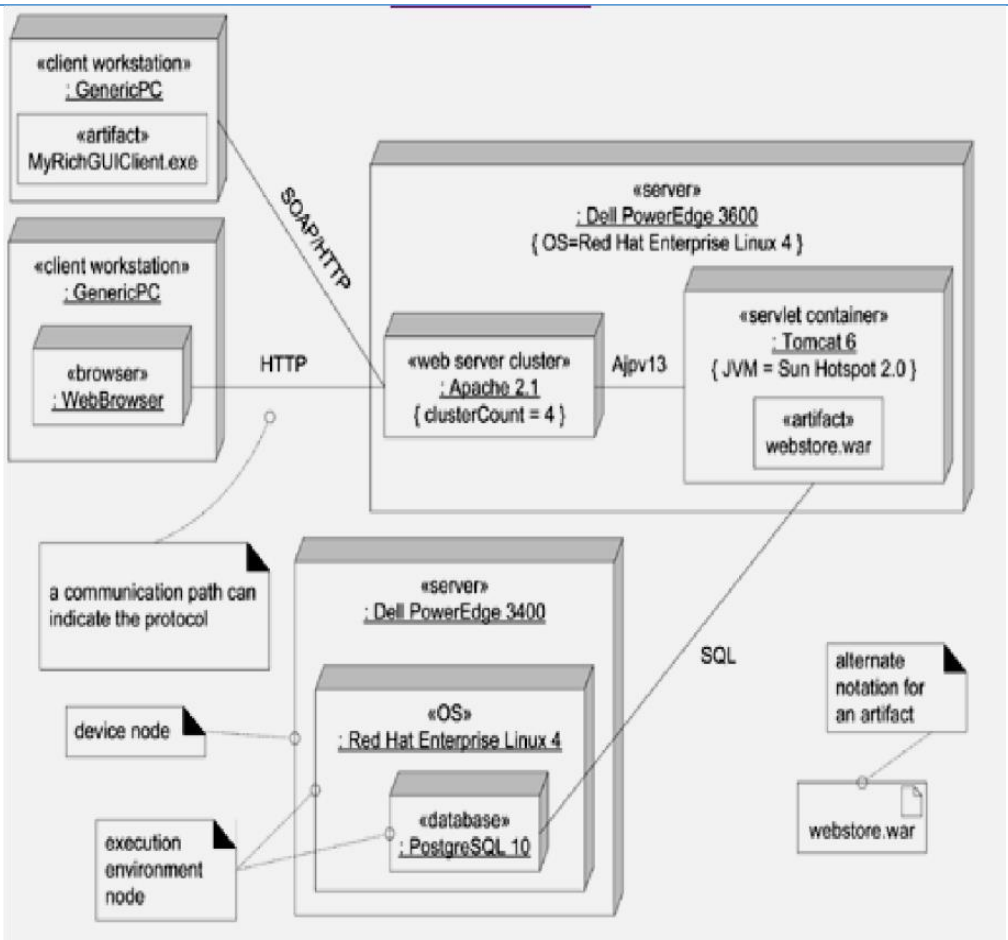


Figure 24: Generic Application of Deployment Diagram

STUDENT TASK-01

Draw a **Deployment Diagram** of below scenario**.**

You are developing a Web-based PEC portfolio/interactive application. User sitting on his personal computer (PC) use some browser to access the web based application developed in HTML 5. HTML 5 is installed on his PC. Website (.html file) resides on web server which is accessed by user through HTTP protocol. For data related transactions, database server is linked to web server through SQL interface. Database server resides SQL database for any entertaining any data queries.

LAB-07 (II)

OBJECTIVE

After this lab students will be able to:

- Familiarize with the concepts underlining **Component** Model
- Model **Component** diagram and its application

PRE-LAB READING ASSIGNMENT

Basic concepts of component diagram

LAB RELATED CONTENT

Component Diagram

Component diagrams illustrate the pieces of software, embedded controllers, etc., that will make up a system. A component diagram has a higher level of abstraction than a Class Diagram - usually a component is implemented by one or more classes (or objects) at runtime. They are building blocks so a component can eventually encompass a large portion of a system. The Component Diagram helps to model the physical aspect of an Object-Oriented software system. It illustrates the architectures of the software components and the dependencies between them. Those software components including run-time components, executable components also the source code components.

SOFTWARE TOOL

Visual paradigm

EXPERIMENTS

EXAMPLE EXPERIMENT-01

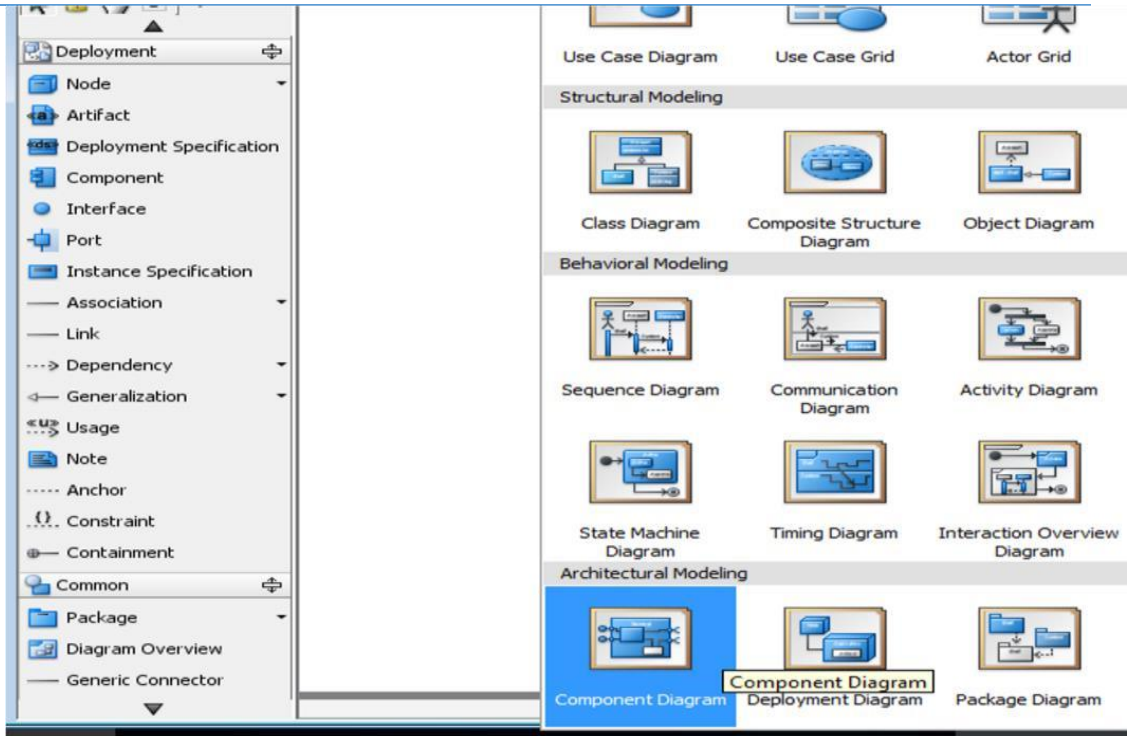
Draw a component diagram for ATM transactions.

PROCEDURE

Creating Component diagram

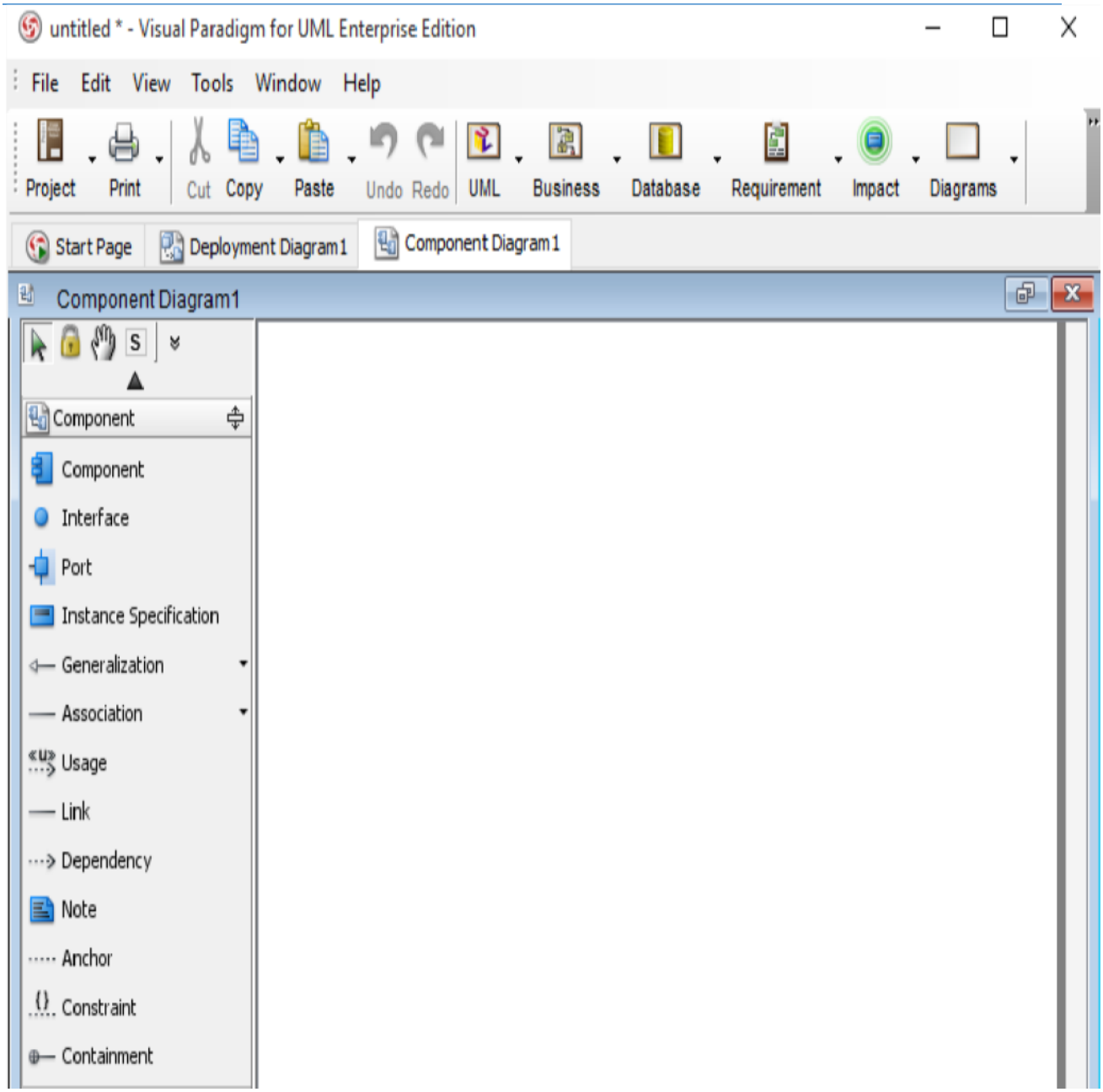
Perform the steps below to create a UML **Component** diagram in Visual Paradigm.

1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Component Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

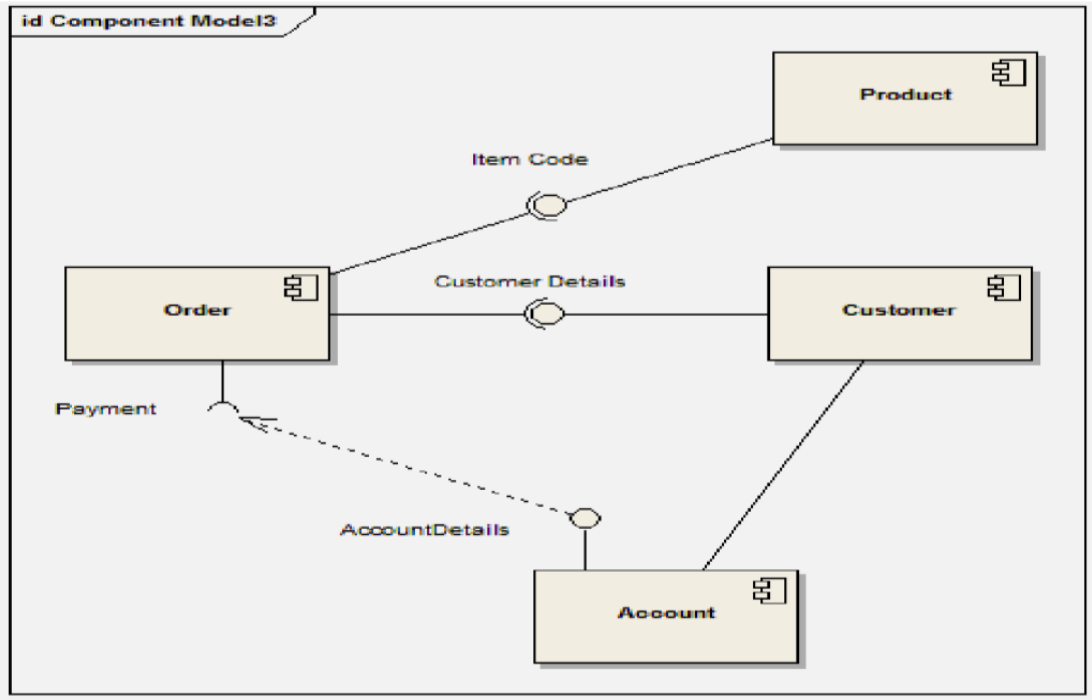


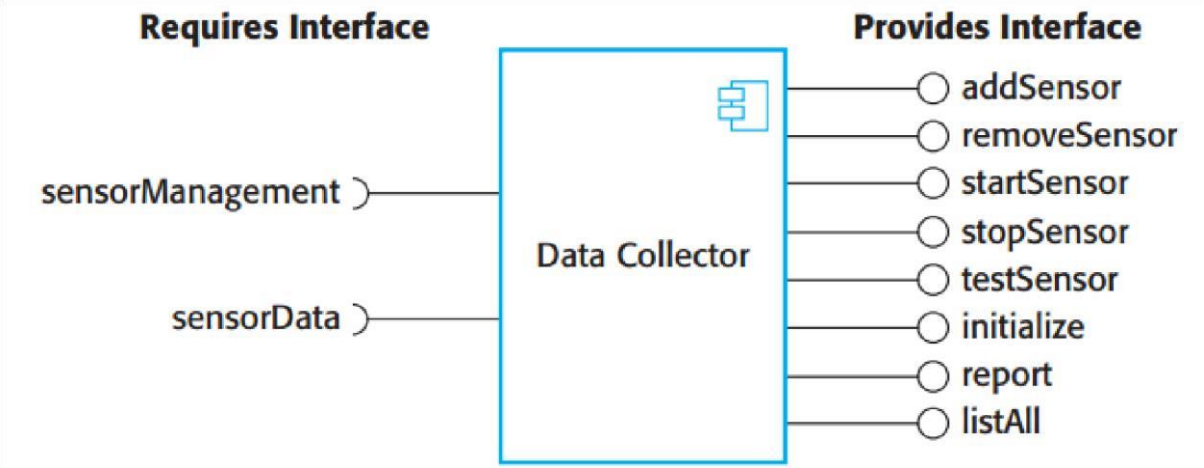
Tool Box support for Component Diagram

Tool box contain all the required model elements required for drawing **Component** diagram on the drawing panel.


untitled * - Visual Paradigm for UML Enterprise Edition
File Edit View Tools Window Help
Project Print Cut Copy Paste Undo Redo UML Business Database Requirement Impact Diagrams
Start Page
Deployment Diagram1
Component Diagram1
Component Diagram1
Component
Component
Interface
Port
Instance Specification
Generalization
Association
Usage
Link
Dependency
Note
Anchor
Constraint
Containment

Examples of Component Diagram





STUDENT TASK-01

Draw a Component Diagram of below scenario.

Design the interfaces of components that might be used in a system for an emergency control room. You should design interfaces for a call-logging component that records calls made, and a vehicle discovery component that, given a post code (zip code) and an incident type, finds the nearest suitable vehicle to be dispatched to the incident.

LAB-08

OBJECTIVE

After this lab students will be able to:

- Familiarize with the concepts underlining **Communication** Model
- Model **Communication** diagram and its application

PRE-LAB READING ASSIGNMENT

Basic concepts of communication diagram

LAB RELATED CONTENT

Communication diagrams illustrate object interactions in a graph or network format, in which objects can be placed anywhere on the diagram. Communication diagram is more focused on showing the collaboration of objects rather than the time sequence. Communication diagrams have advantages when applying "UML as sketch" to draw on walls because they are much more space-efficient. This is because the boxes can be easily placed or erased anywhere horizontal or vertical. Modifying wall sketches is easier with communication diagrams it is simple to erase a box at one location, draw a new one elsewhere, and sketches a line to it. In contrast, new objects in a sequence diagrams must always be added to the right edge, so it quickly consumes and exhausts right-edge space on a page; free space in the vertical dimension is not efficiently used. Developers doing sequence diagrams on walls rapidly feel the drawing pain when contrasted with communication diagrams

SOFTWARE TOOL

Visual paradigm

EXPERIMENTS

EXAMPLE EXPERIMENT-01

Draw communication diagram of room reservation in hotel

PROCEDURE

Creating Communication diagram

Perform the steps below to create a UML **Communication** diagram in Visual Paradigm.

1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Communication Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

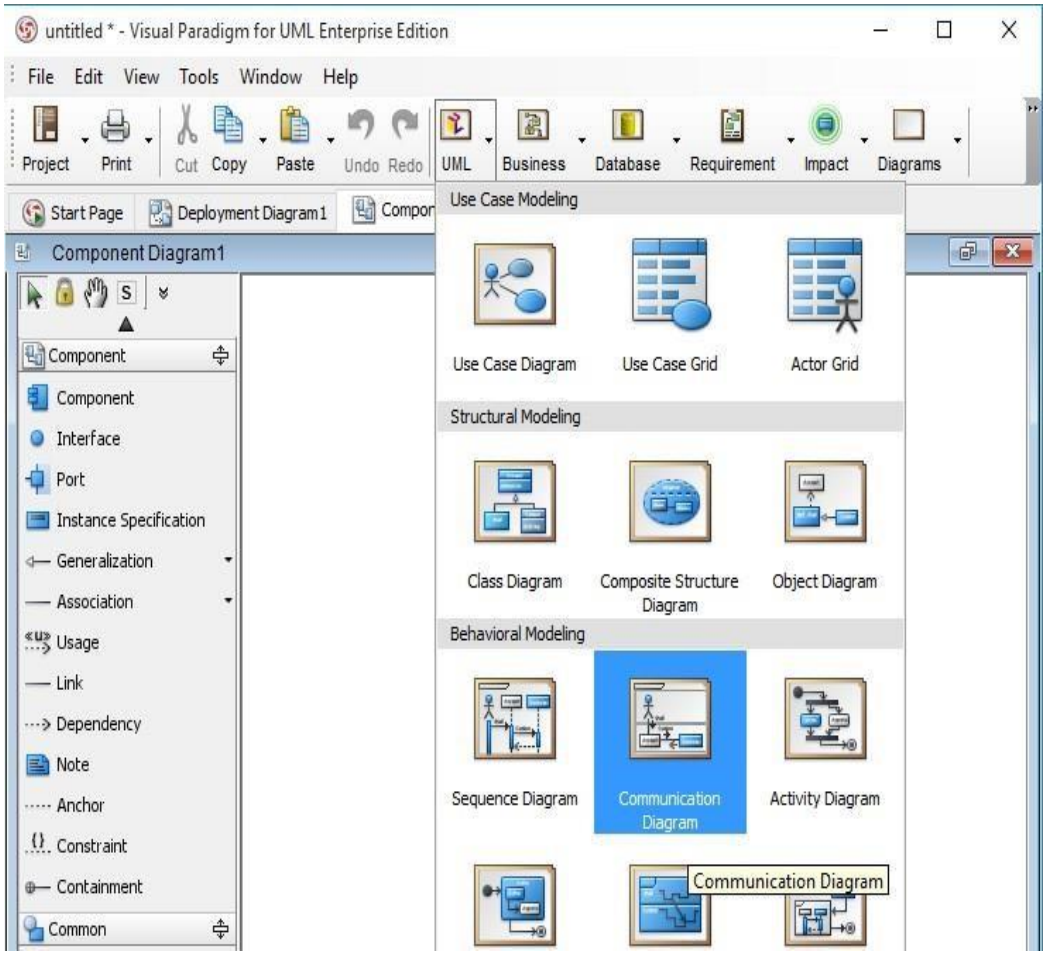

untitled * - Visual Paradigm for UML Enterprise Edition
File Edit View Tools Window Help
Project Print Cut Copy Paste Undo Redo UML Business Database Requirement Impact Diagrams
Start Page
Deployment Diagram1
Component Diagram1
Component
Component
Interface
Port
Instance Specification
Generalization
Association
Usage
Link
Dependency
Note
Anchor
Constraint
Containment
Common
Use Case Modeling
Use Case Diagram
Use Case Grid
Actor Grid
Structural Modeling
Class Diagram
Composite Structure Diagram
Object Diagram
Behavioral Modeling
Sequence Diagram
Communication Diagram
Activity Diagram
Communication Diagram

Figure 26: Selecting Communication Diagram

Tool Box support for Communication Diagram

Tool box contain all the required model elements required for drawing **Communication** diagram on the drawing panel.

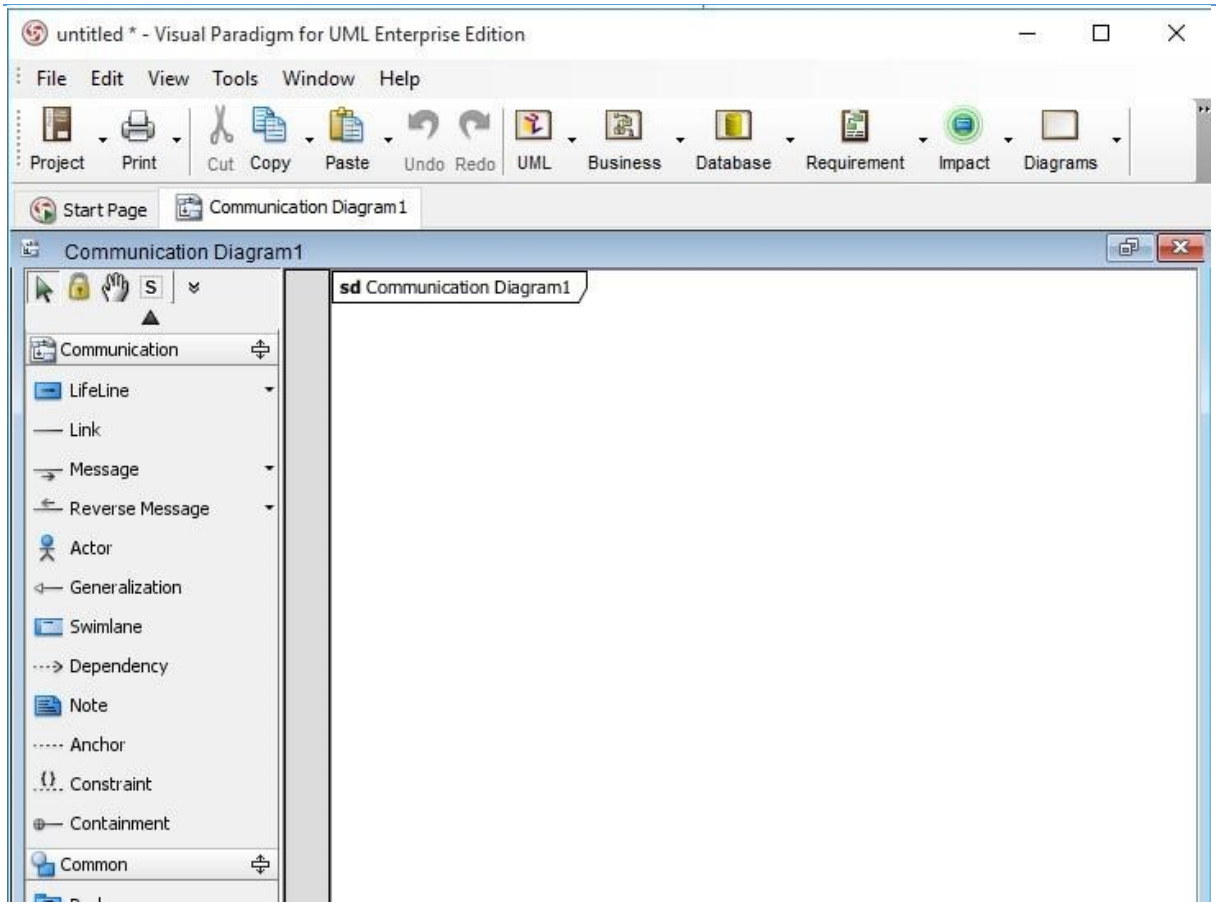


Figure 27: Model Elements of Communication Diagram

STUDENT TASK-01

Draw a **Communication Diagram** of below scenario**.**

Reporting structure in a hierarchical organization. Status information is reported to the project manager, and corrective decisions are communicated back to the teams by the team leaders. The team leaders and the project manager are called the management team.

STUDENT TASK-02

Draw a **communication diagram** of below explained online book shop.

Communication starts with **find_books()** - iterative message which could be repeated some unspecified number of times. Client searches inventory of books, and if he/she is interested in some book he/she can view description of the book **view_book()**. If client decides to buy, he/she can add book to the shopping cart **add_to_cart()**. Checkout includes getting list of books from shopping cart, creating order, and updating inventory, if order was completed.

LAB-09 (I)

OBJECTIVE

After this lab students will be able to:

- Familiarize with the concepts underlining **Package** Model

- Model **Package** diagram and its application

PRE-LAB READING ASSIGNMENT

Basic concepts of package diagram

LAB RELATED CONTENT

UML Package Diagrams provide a way to group elements. A UML package can group anything, e.g., classes, use-cases, other packages, etc. Packages as components can be nested inside other packages. Dependencies among different packages can be shown with the dependency line. UML package is shown as a tabbed folder. Subordinate packages may be shown within it. The package name is within the tab if the package depicts its elements; otherwise, it is centered within the folder itself.

An element is owned by the package within which it is defined, but may be referenced in other packages. In that case, the element name is qualified by the package name using the pathname format **PackageName::ElementName**. A class shown in a foreign package may be modified with new associations, but must otherwise remain unchanged.

Packages should be cohesive. Anything you put into the package should make sense when considered with the rest of the contents of the package. To determine whether a package is cohesive, a good test is you should be able to give your package a short, descriptive name.

Following types of classes can typically be grouped together in one package:

- Classes in the same inheritance hierarchy
- Classes related to one another via composition
- Classes that collaborate with each other (a lot of information reflected by your sequence diagrams and communication diagrams)

SOFTWARE TOOL

Visual paradigm

EXPERIMENTS

EXAMPLE EXPERIMENT-01

Draw package diagram of Model View Controller (MVC) Architectural Pattern

PROCEDURE

Creating Package diagram

Perform the steps below to create a UML **Package** diagram in Visual Paradigm.

1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Package Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

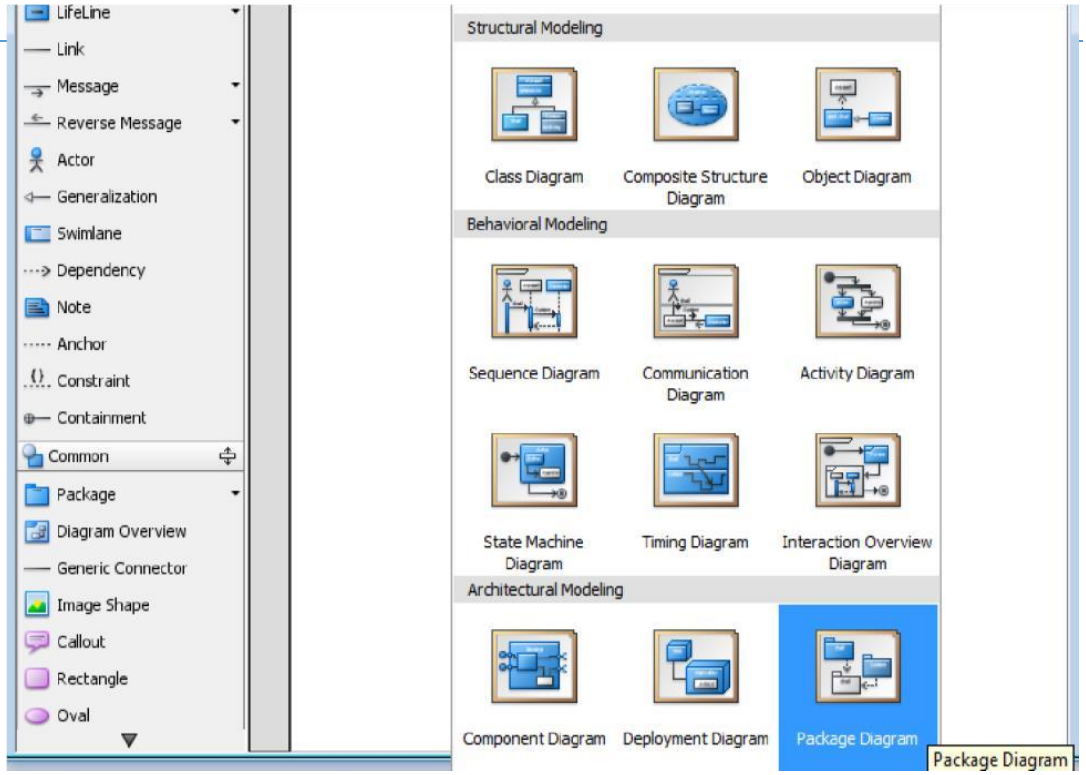


Figure 28: Selecting Package Diagram

Tool Box support for Package Diagram

Tool box contain all the required model elements required for drawing **Package** diagram on the drawing panel.

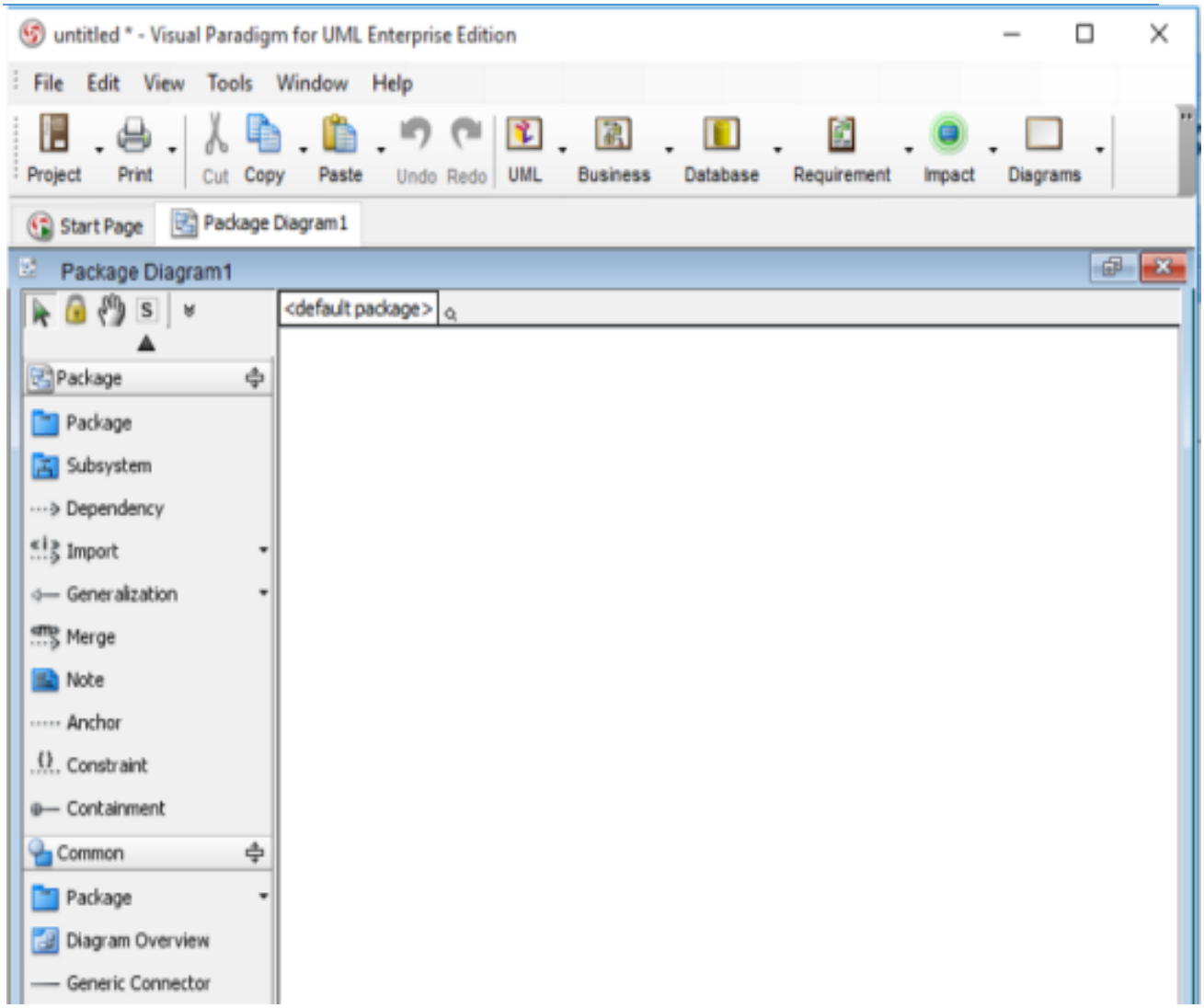


Figure 29: Model Elements of Package Diagram

Examples of Package Diagram

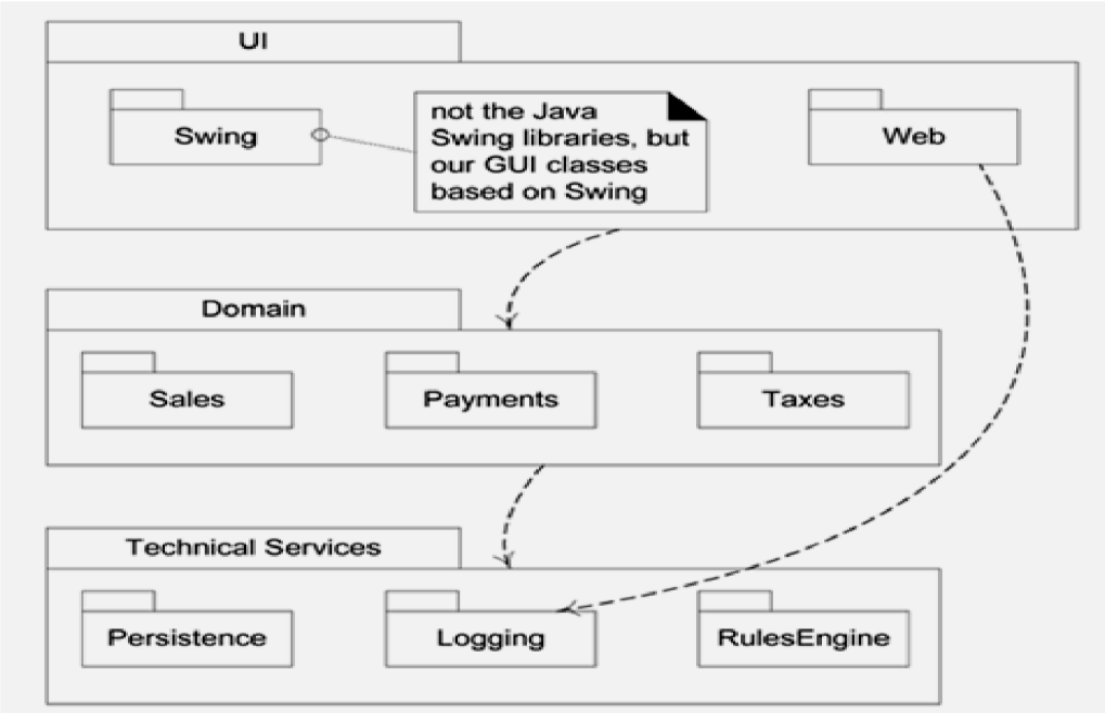


Figure 30: Example of Package Diagram

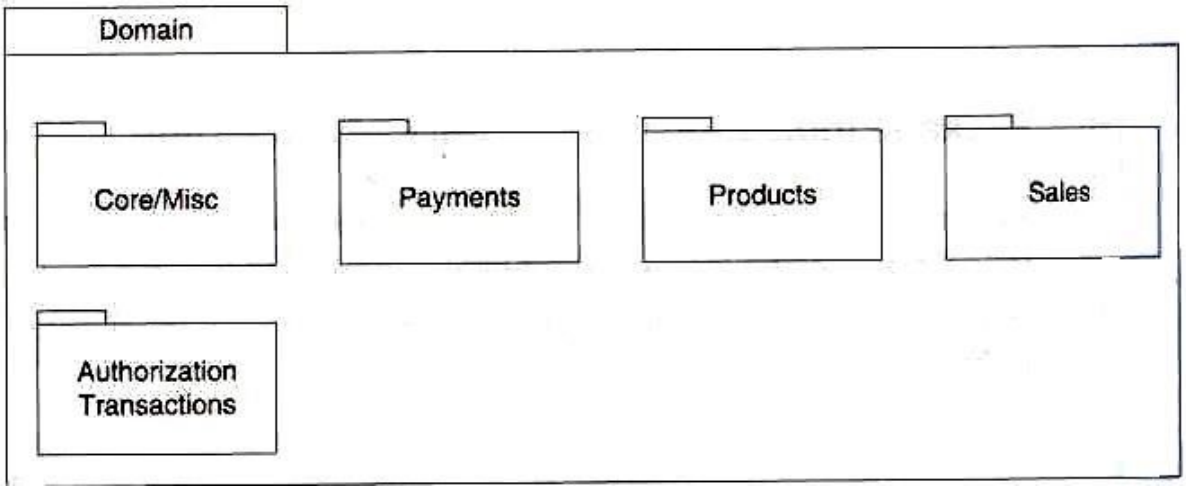


Figure 31: Package Diagram of POS System

STUDENT TASK-01

Draw a **Package Diagram** of Online Shopping. Show appropriate sub-packages or classes.

LAB-09 (II)

OBJECTIVE

- Familiarize with the concepts underlining **Data Flow** Model
- Model **Data Flow** diagram and its application

PRE-LAB READING ASSIGNMENT

Basic concepts of data flow diagram

LAB RELATED CONTENT

The DFD is mainly used for describing a very problem-oriented view of the workings of a system. It provides a description based on modeling the flow of information around a network of operational elements, with each element making use of or modifying the information flowing into that element.

The DFD is a graphical representation, and it makes use of only four basic symbols. Because of its highly abstract nature, in terms of the level of description provided, it is chiefly used during the early design stages that are often termed 'analysis' – at a time when the designer is likely to be making major architectural design decisions.

So essentially this DFD provides a top-level 'model' of how the designer intends the auto-teller to operate. It is expressed in terms that are part of the problem domain (customer, PIN, transaction), rather than of the solution, and as such it identifies the main architectural tasks for the auto-teller system.

SOFTWARE TOOL

Visual paradigm

EXPERIMENTS

EXAMPLE EXPERIMENT-01

Draw data flow diagram (Level-0 and Level-1) of Food Ordering System

PROCEDURE

Creating dataflow diagram

Perform the steps below to create a UML **Data Flow** diagram in Visual Paradigm.

1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Data Flow Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

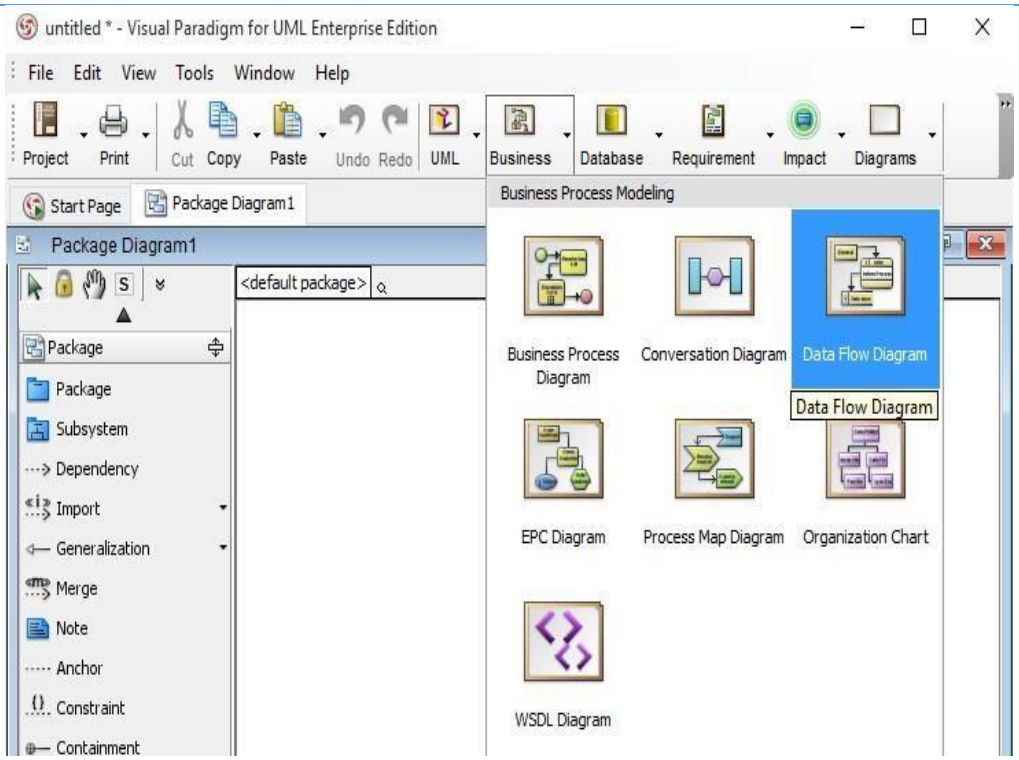


Figure 32: Selecting Data Flow Diagram

Tool Box support for Data Flow Diagram

Tool box contain all the required model elements required for drawing **Data Flow** diagram on the drawing panel.

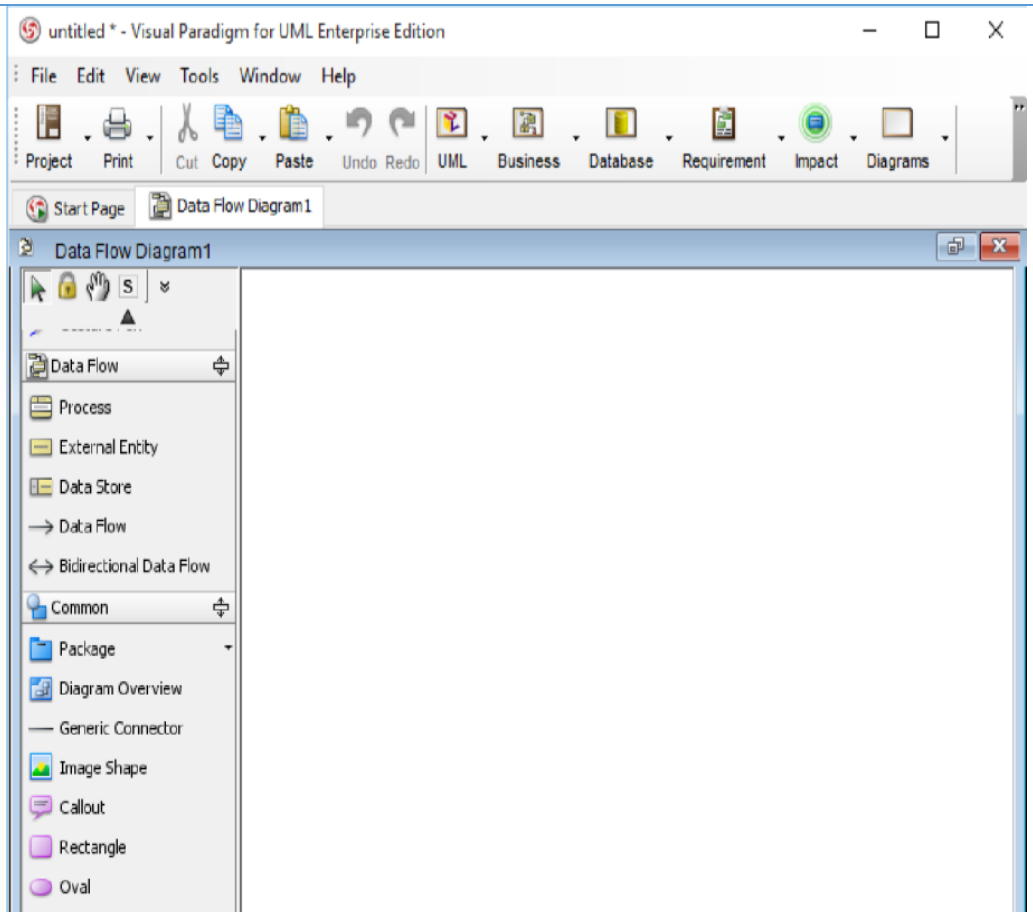


Figure 33: Model Elements of Data Flow Diagram

Examples of Data Flow Diagram

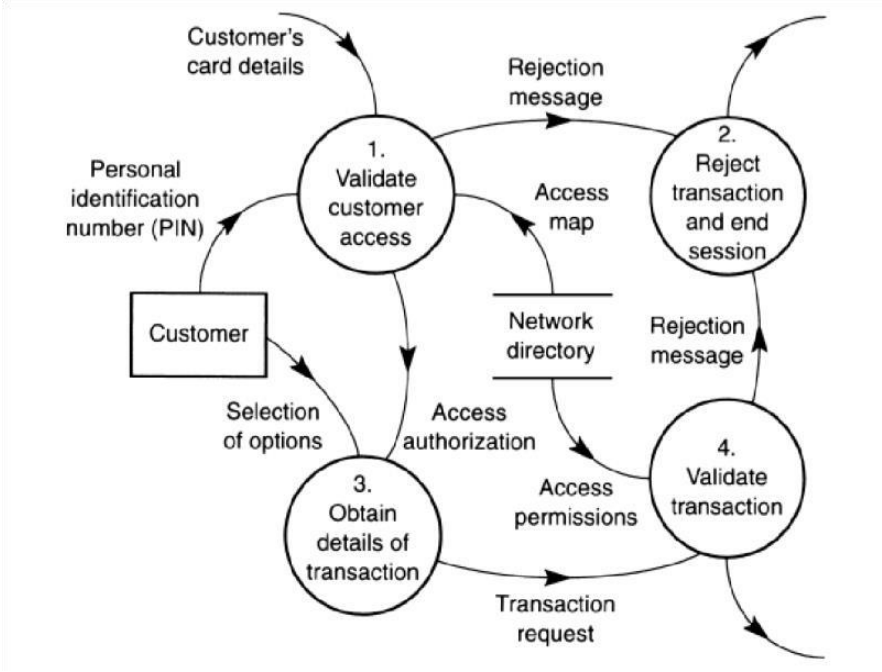


Figure 34: DFD of ATM

STUDENT TASK-01

Draw a **Data Flow Diagram** of travel agency booking system showing its different processes.

LAB-09 (III)

OBJECTIVE

After this lab students will be able to:

- Familiarize with the concepts underlining **Entity Relation** Model
- Model **Entity Relation** diagram and its application

PRE-LAB READING ASSIGNMENT

Basic concepts of data flow diagram

LAB RELATED CONTENT

An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database.

There are two reasons to create a database diagram. You're either designing a new schema or you need to document your existing structure.

If you have an existing database you need to to document, you create a database diagram using data directly from your database. You can export your database structure as a CSV file (there are some scripts on how to this here), then have a program generate the ERD automatically.

SOFTWARE TOOL

Visual paradigm

EXPERIMENTS

EXAMPLE EXPERIMENT-01

Draw entity relation (ER) diagram of Hospital Billing System

PROCEDURE

Perform the steps below to create a UML **ER** diagram in Visual Paradigm.

1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **ER Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

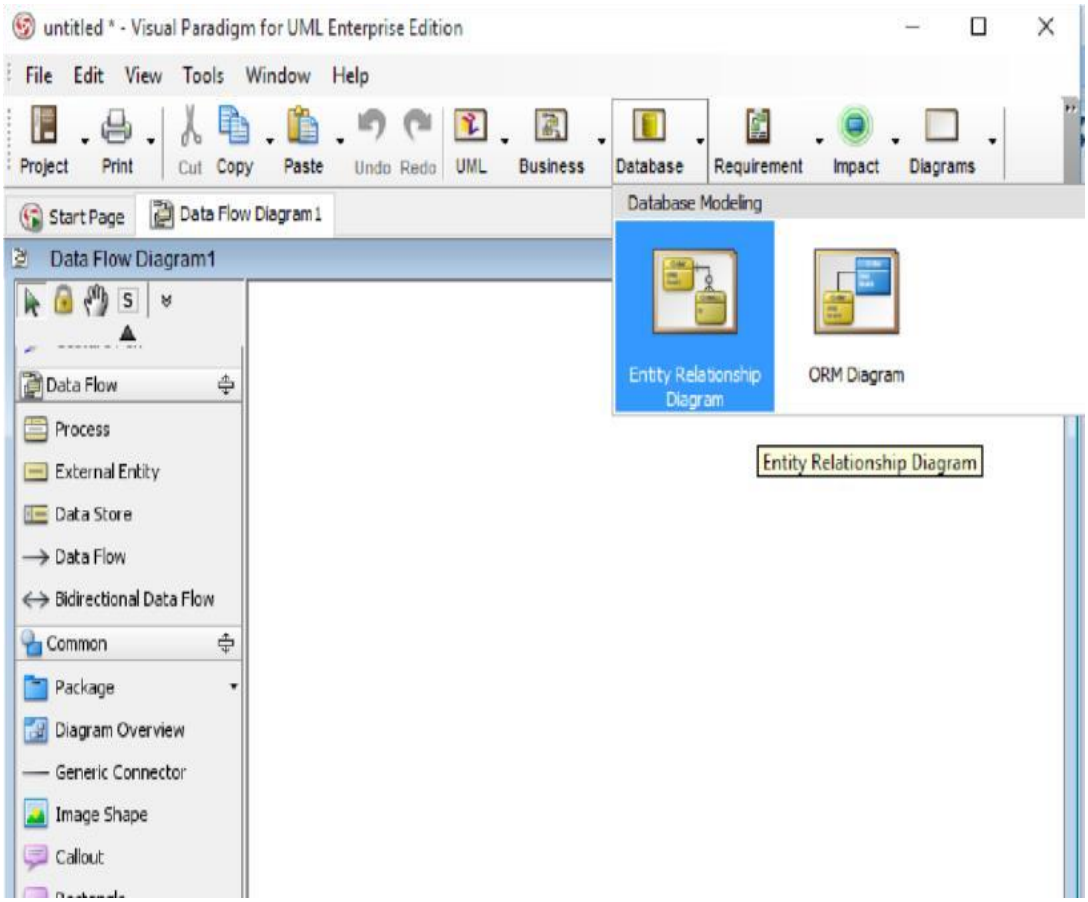


Figure 35: Selecting ER Diagram

Tool Box support for Data Flow Diagram

Tool box contain all the required model elements required for drawing **Data Flow** diagram on the drawing panel.

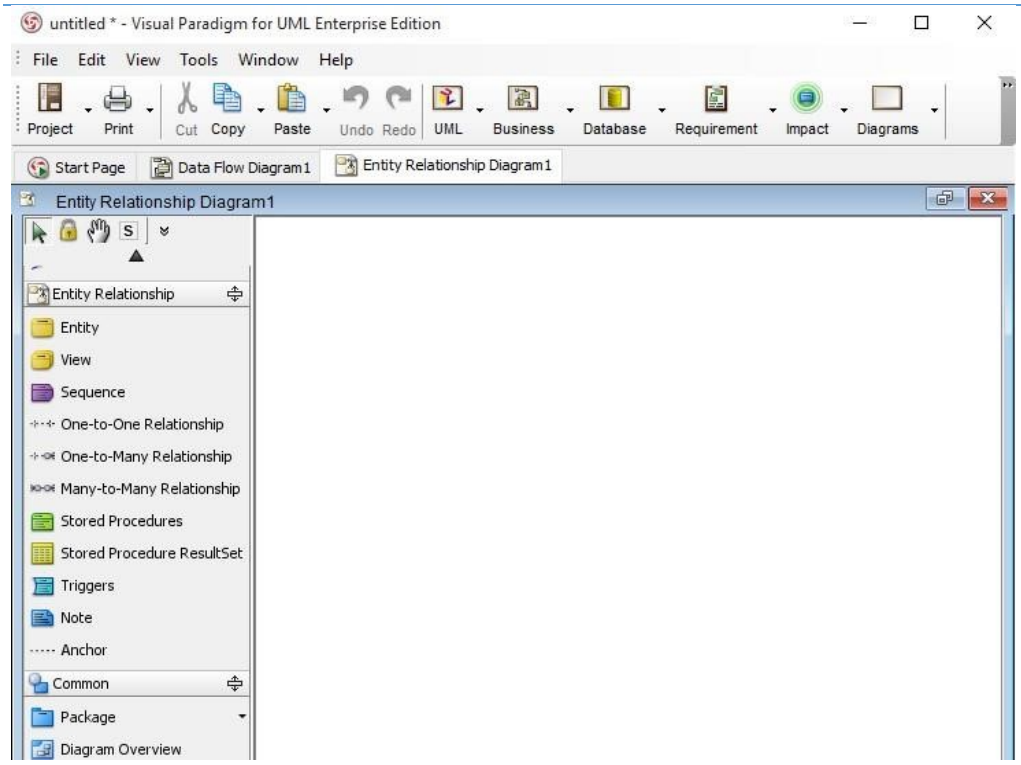


Figure 36: Model Elements of ER Diagram

Examples of Data Flow Diagram

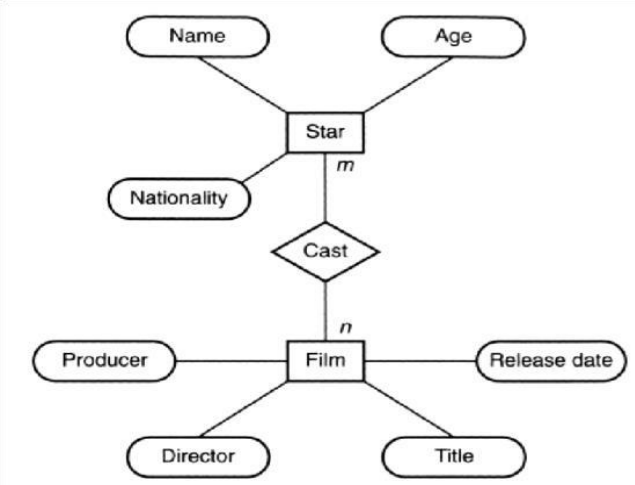


Figure 37: ERD showing relationship between two entities

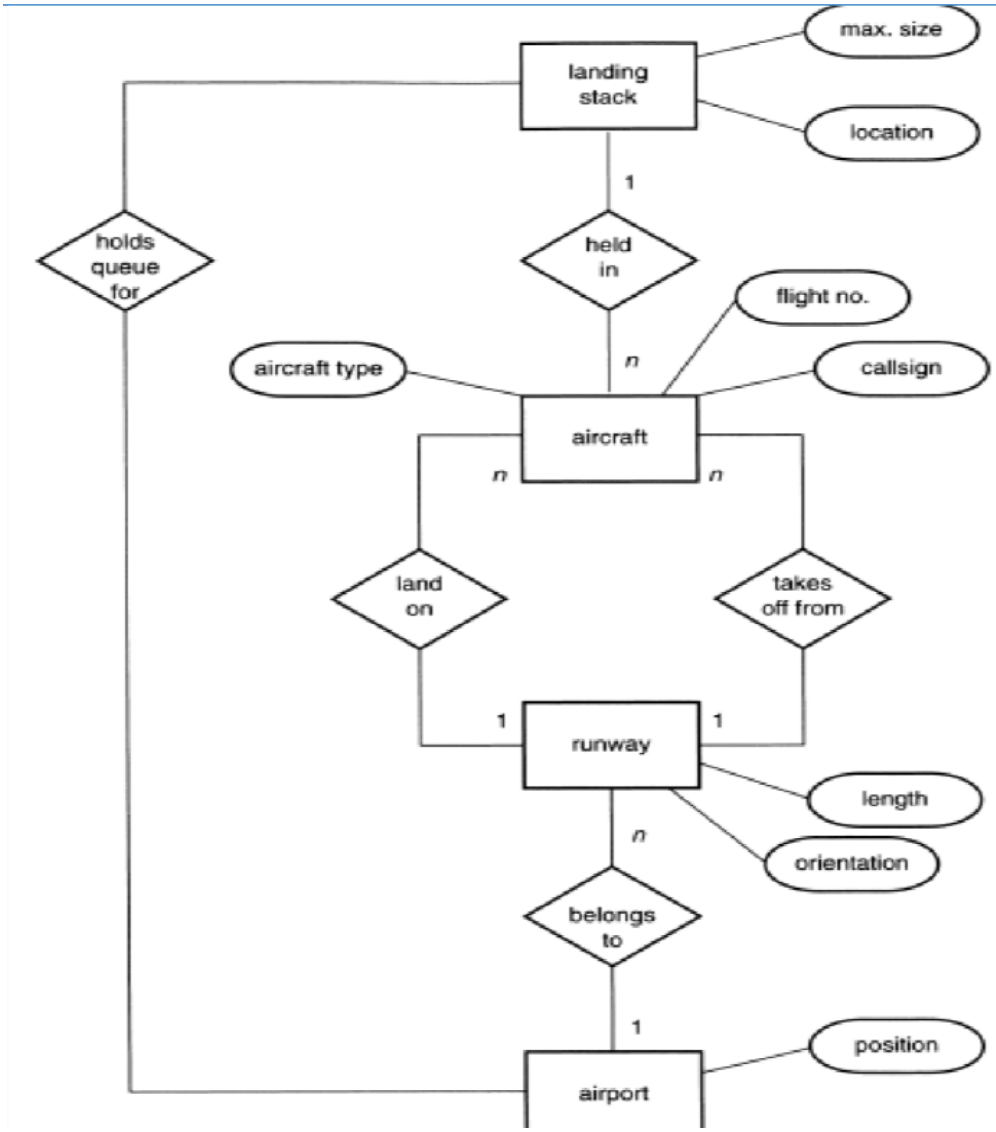


Figure 38: ERD of Air Traffic Control System

STUDENT TASK-01

Draw an **ER Diagram** of university library system showing its different entities.

LAB-10 (I)

OBJECTIVE

After this lab student will be able to:

Understand the concept of object diagram.

PRE-LAB READING ASSIGNMENT

Basics of Object Diagram

LAB RELATED CONTENT

Object diagram is a kind of UML diagram that shows a snapshot of instances of things in class diagram. Similar to class diagram, it shows the static design of system from the real or prototypical perspective.

SOFTWARE TOOL

Visual paradigm

PROCEDURE

Creating object diagram

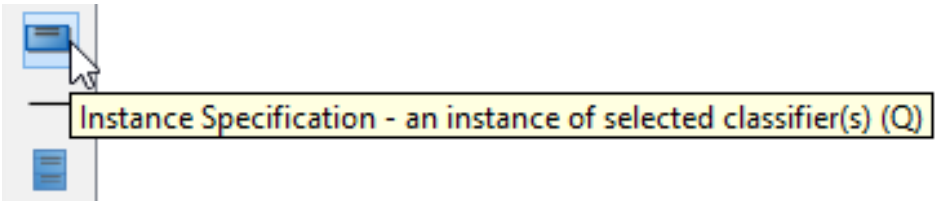
Perform the steps below to create a UML object diagram in Visual Paradigm.

1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Object Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

Creating instance specification

To create instance specification in object diagram:

1. Select **Instance Specification** from the diagram toolbar.



Create instance specification

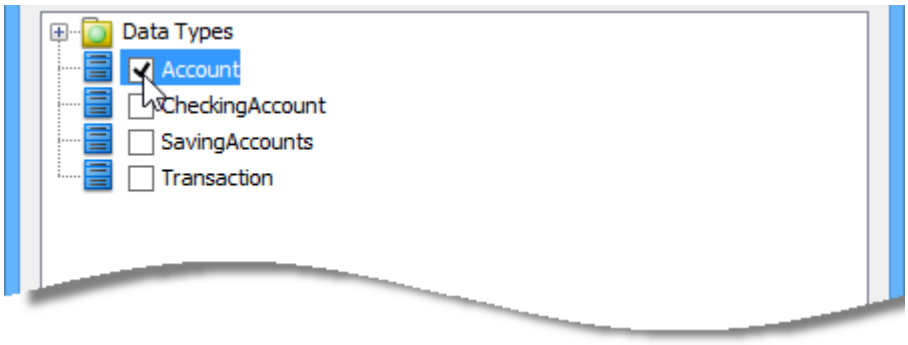
1. Click on the diagram to create an instance specification shape. Name it.

Instance specification created

Selecting classifiers

To specify classifiers for an instance specification:

1. Right-click on the desired instance specification shape and select **Select Classifier** > **Select Classifier...** from the pop-up menu.
2. This opens the **Classifiers** tab. Click **Add...** in it.
3. In the **Select Classifier** window, select the class(es) to be the classifier of the instance specification. If you are referencing another project, you can select its model element to be the classifier. Just change the **from project** selection at the top of the window.



Selecting classifier

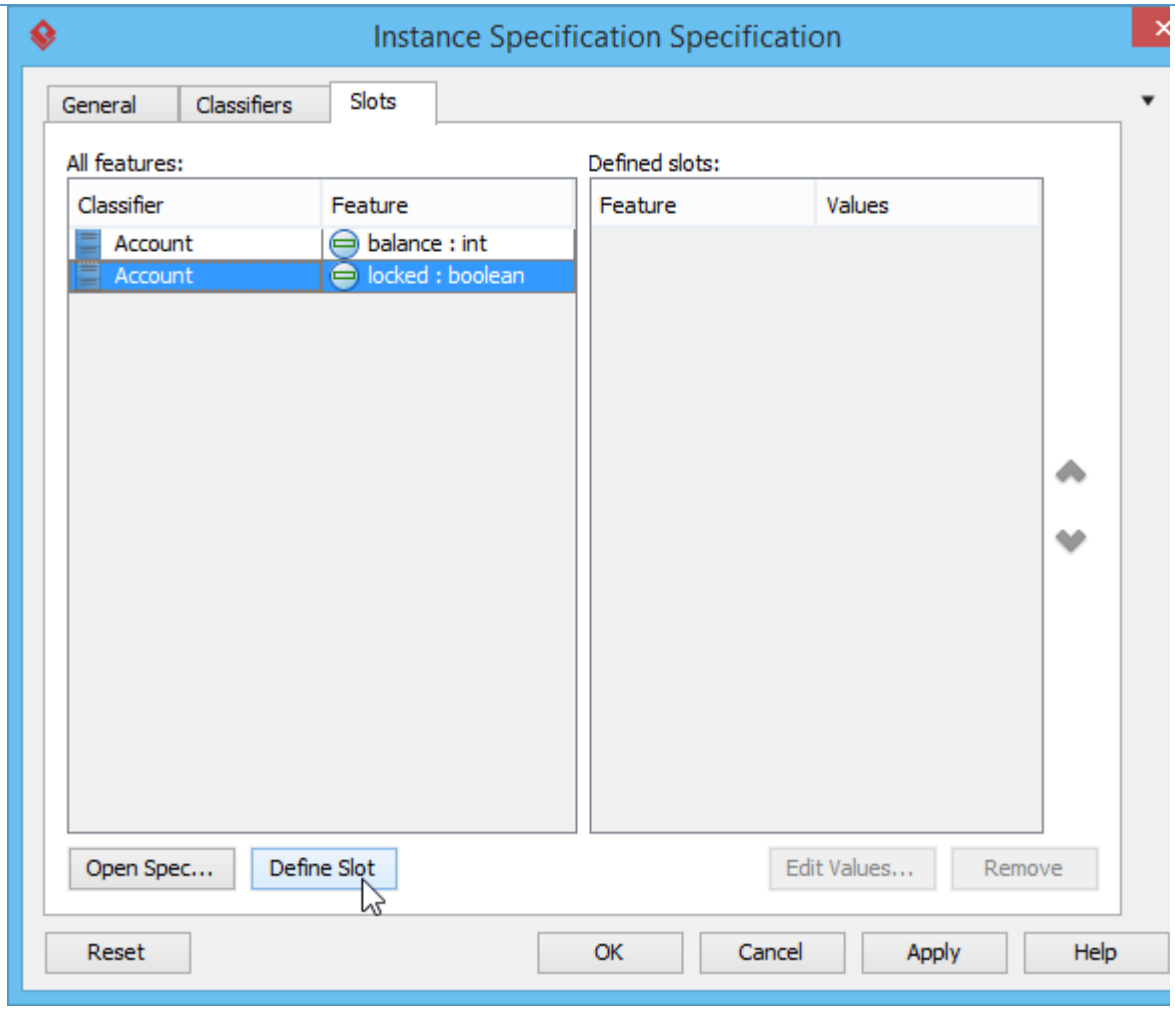
4. Click **OK** to return to the **Instance Specification Specification** window.
5. Click **OK** to return to the diagram.

Classifier selected

Defining slots

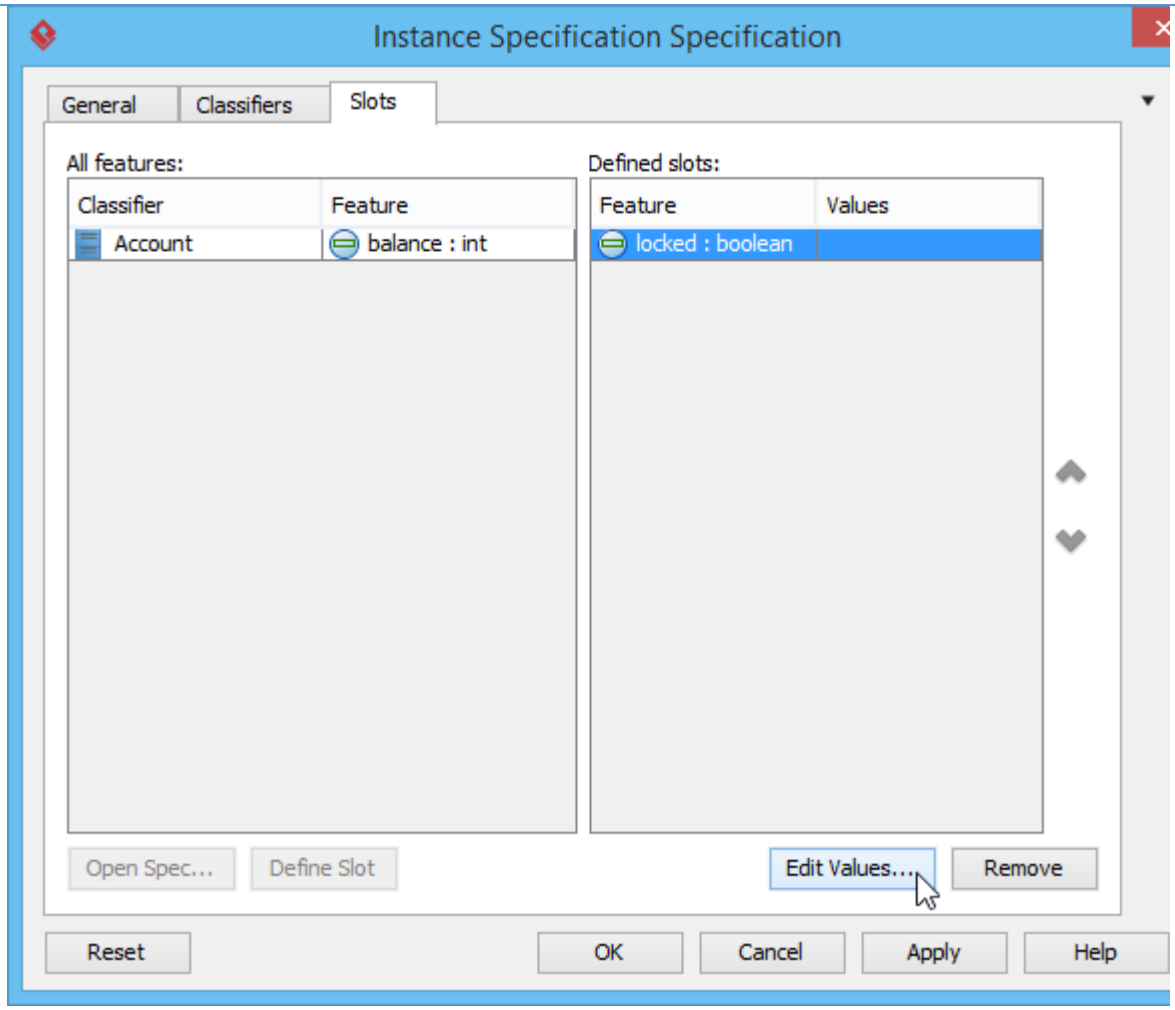
To define slots for an instance specification:

1. Right-click on the desired instance specification shape and select **Slots...** from the pop-up menu.
2. The **Instance Specification Specification** window appears with the **Slots** tab selected. Select the features that you want to define slots on the left and click **Define Slot**.



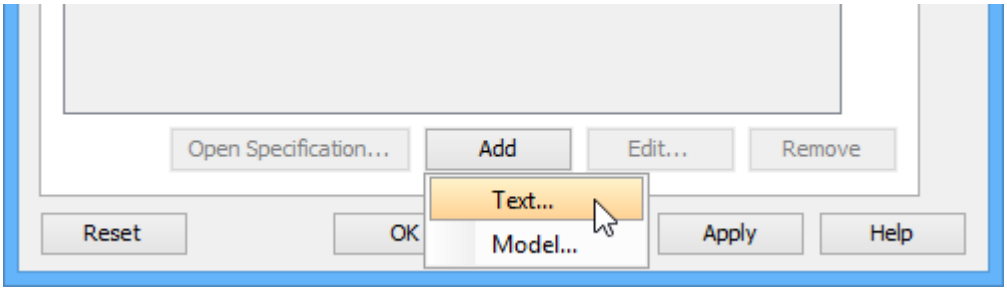
Defining slot

3. Select a defined slot and click **Edit Values...** at bottom right.



Edit values

4. The **Slot Specification** window pops out, the **Values** tab is opened by default. Click **Add** button and select **Text** from the pop-up menu.



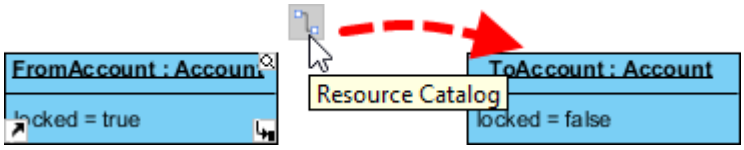
Add values to defined slot

5. Enter the slot value and click **OK** to confirm.
6. Click **OK** again in the **Instance Specification Specification** window to return to the diagram.

Creating link

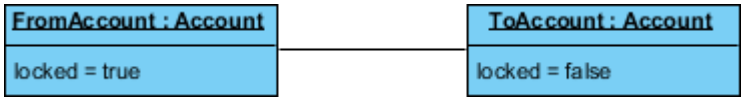
To create link between instance specifications:

1. Move the mouse pointer over the source instance specification.
2. Press on the Resource Catalog button and drag it out. Drag to the target instance specification and release the mouse button.



Create a link

3. Select Link from Resource Catalog. A link is created.



Link created

STUDENT TASK:

LAB-10 (II)

OBJECTIVE
After this lab student will be able to:
Understand the Composite Structure Diagram

PRE-LAB READING ASSIGNMENT
Basics of Composite Structure Diagram

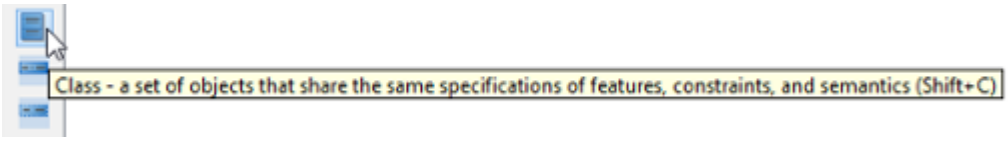
LAB RELATED CONTENT
Composite structure diagram is a kind of UML diagram that visualizes the internal structure of a class or collaboration. It is a kind of component diagram mainly used in modeling a system at micro point-of-view.

SOFTWARE TOOL
Visual paradigm

PROCEDURE

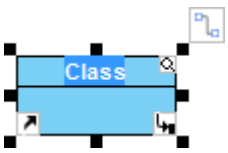
Creating composite structure diagram
Perform the following steps to create a UML composite structure diagram.
1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Composite Structure Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

Creating class
To create a class in composite structure, click **Class** on the diagram toolbar and then click on the diagram.



Create class

A class will be created.



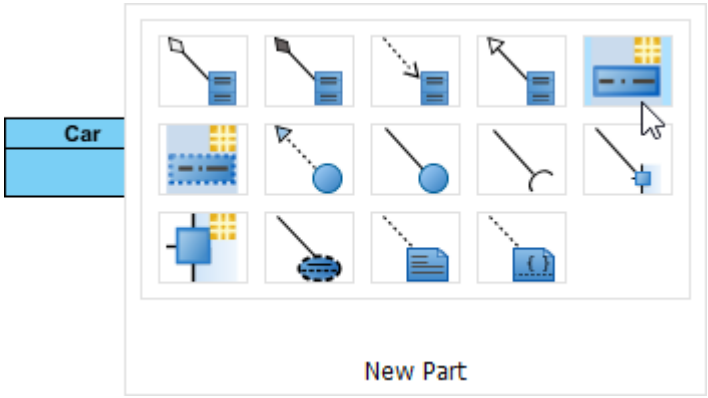
Class created

Creating part
To create a part inside a class:
1. Move your mouse pointer over the class.

2. Click on the **Resource Catalog** button.

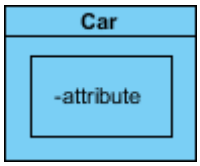
Clicking on Resource Catalog button

3. Select **New Part** from Resourcce Catalog.



To create part

4. A part is created.

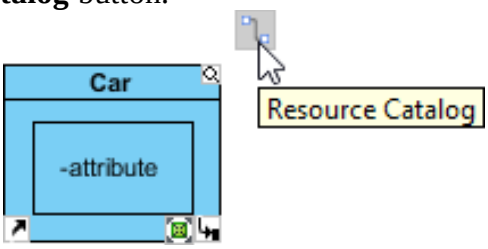


Part created

Creating port

To create a port that attaches to a class:

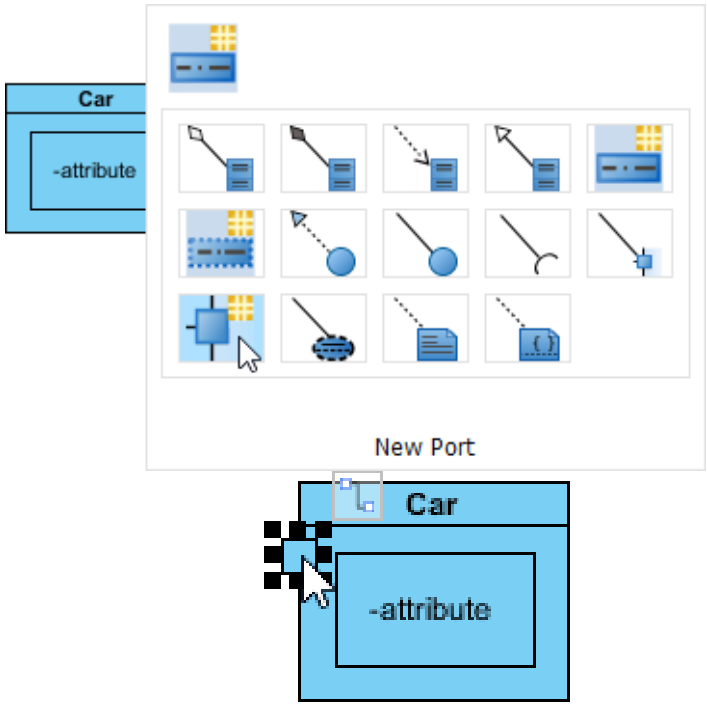
1. Move your mouse pointer over the class.
2. Click on the **Resource Catalog** button.



Clicking on Resource Catalog button

3. Select **New Port** from Resourcce Catalog.

4. A port is created.

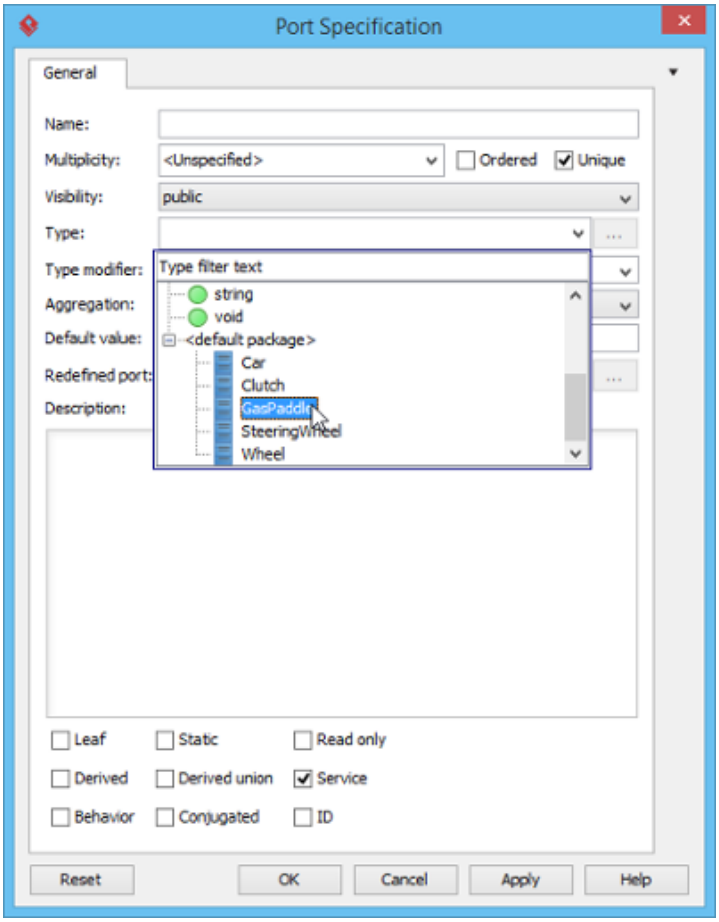


Port created

Specifying type of port

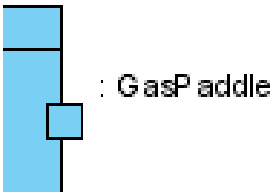
Right-click the port and select **Open Specification...** from the pop-up menu. The **Port Specification** window appears.

Click the combo box of **Type** and select a class.



Select type

Click **OK** button to apply the changes. Type will be shown on the caption of the port.



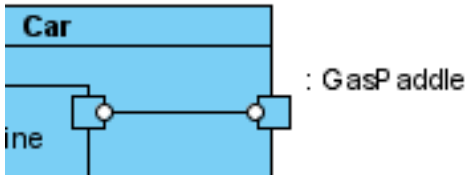
Type shown on port

Creating connector

To create connector, click **Connector** on the diagram toolbar.

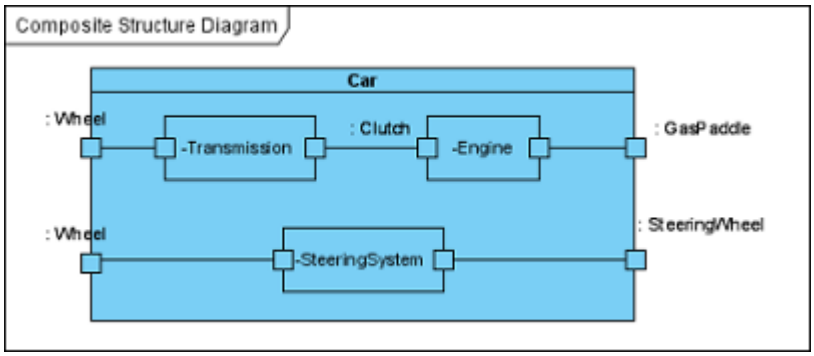
Create connector

Drag from the source shape, move the mouse over the target shape and then release the mouse button to create the connector.



Connector created

Continue to complete the diagram.



Completed diagram

STUDENT TASK-01

LAB-11 (I)

OBJECTIVE
After this lab students will be able to get;
- Familiarize with the concepts underlining Timing Diagram

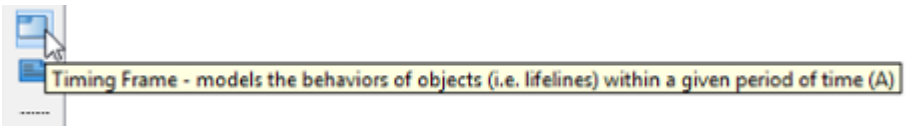
PRE-LAB READING ASSIGNMENT
Basics of Timing Diagram

LAB RELATED CONTENT
Timing diagram is a kind of UML diagram that shows time, event, space and signal for real-time and distributed system.

SOFTWARE TOOL
Visual paradigm

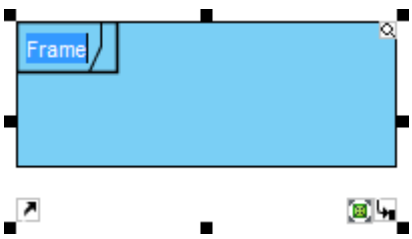
PROCEDURE
Creating timing diagram
Perform the steps below to create a UML timing diagram in Visual Paradigm.
1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Timing Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enables you to select a model to store the diagram.
5. Click **OK**.

Creating timing frame
To create timing frame in a timing diagram, click **Timing Frame** on the diagram toolbar and then click on the diagram.

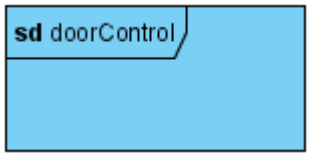


Create timing frame

Double click on the top left corner of the frame to rename it.



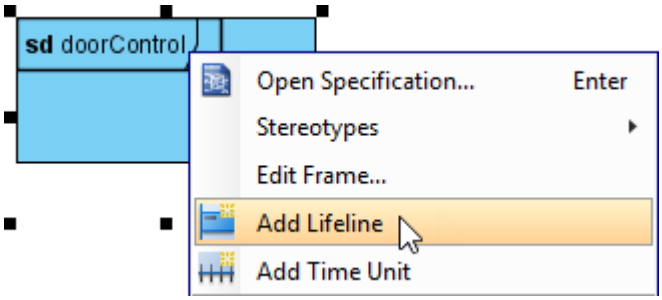
The name of a timing frame is usually preceded by the **sd** keyword.



Frame renamed

Adding lifeline to frame

To add lifeline to a timing frame, right-click the frame and select **Add Lifeline** from the pop-up menu.

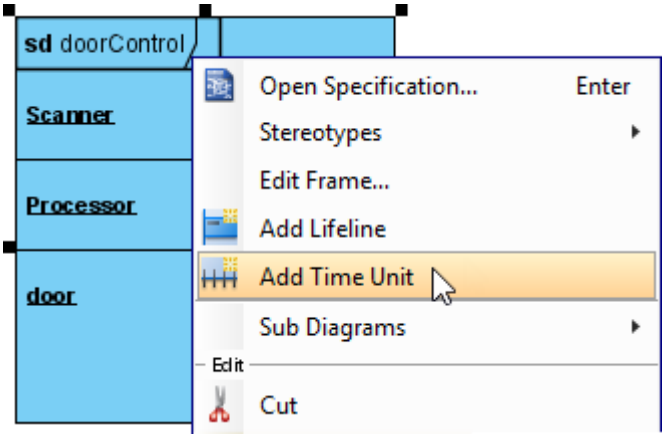


Add lifeline

Double-click on the name of the lifeline to rename it.

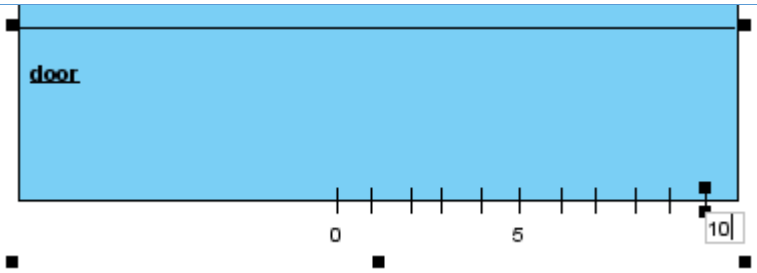
Adding time unit to frame

To add time unit to a timing frame, right-click the frame and select **Add Time Unit** from the pop-up menu.



Add time unit

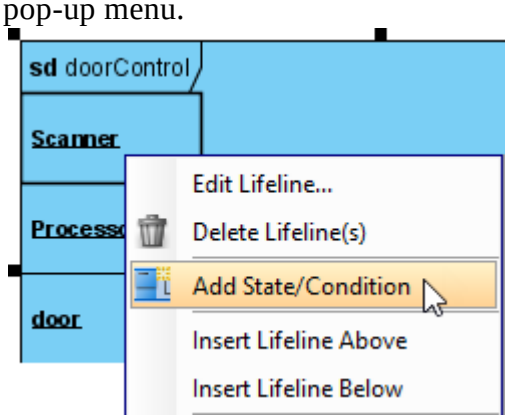
Repeat the step to add as many as time units you need. Double-click on a time unit to rename it.



Rename time unit

Adding state/condition to lifeline

To add state/condition to a lifeline, right-click the lifeline and select **Add State/Condition** from the pop-up menu.

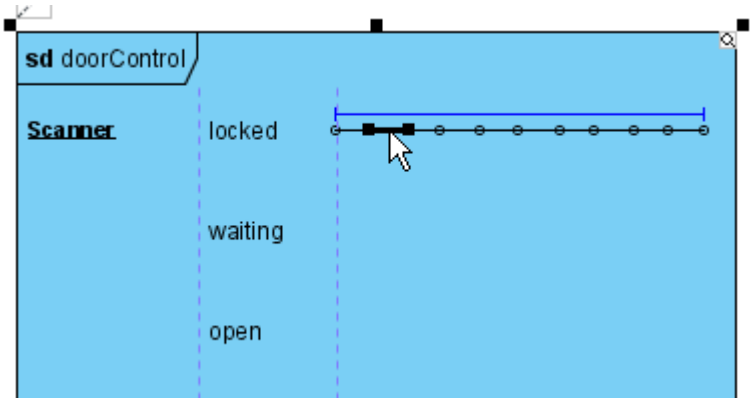


Add state/condition

Double click on the name of the state/condition to rename it.

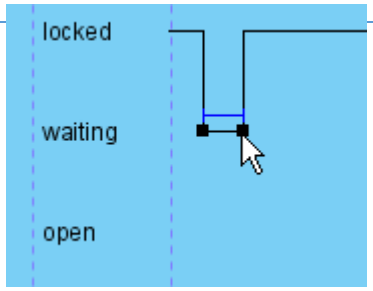
Dragging time instance

Move your mouse pointer over the line segment of a time instance, click and drag it.



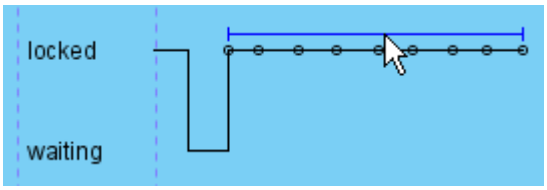
Drag time instance

Release the mouse button when reached the target state/condition.



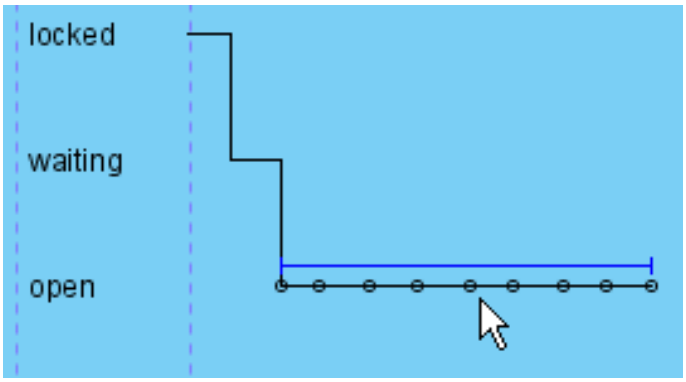
Dragged time instance

You can also move a group of time instances that are at the same state/condition. Mouse over the time instances and you will see a blue line above them, click and drag on the blue line.



Move a group of time instances

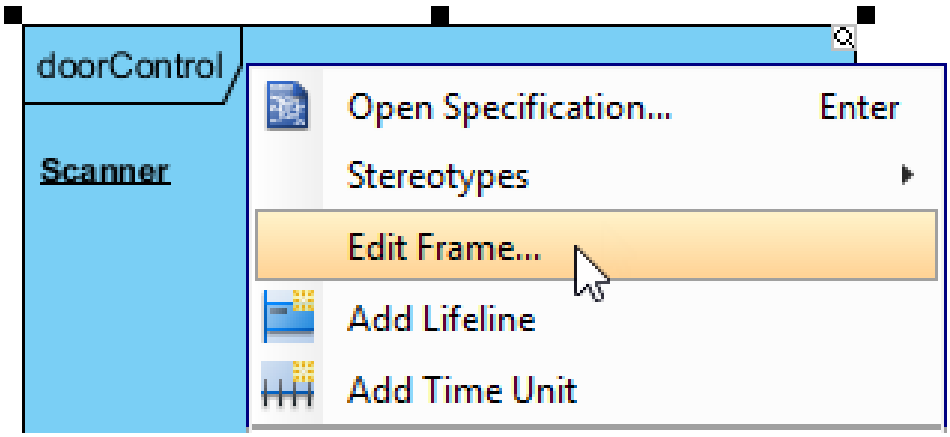
Release the mouse button when reached the target state/condition. The group of time instances is moved at once.



Moved group of time instances

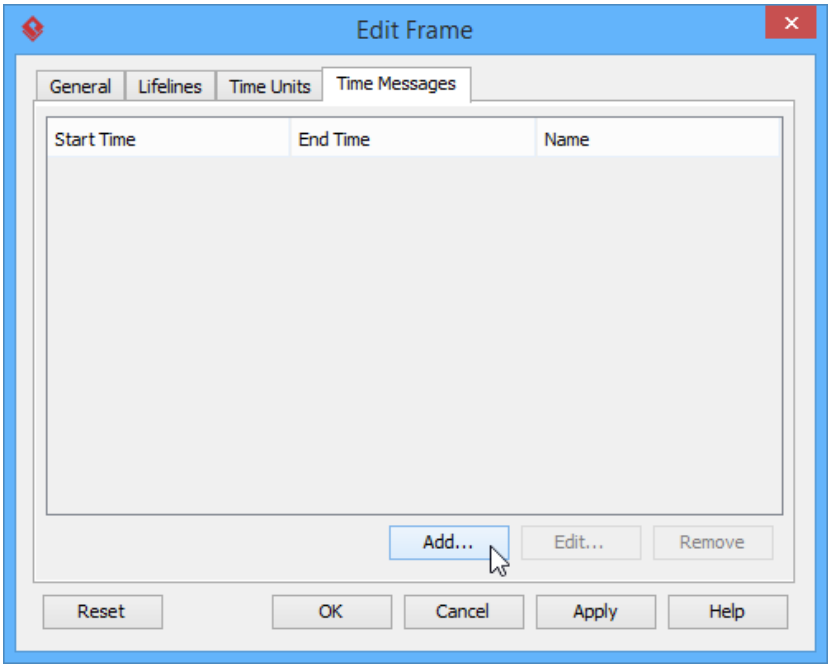
Adding time messages to frame

To add time messages to frame, right-click the timing frame and select **Edit Frame...** from the pop-up menu.



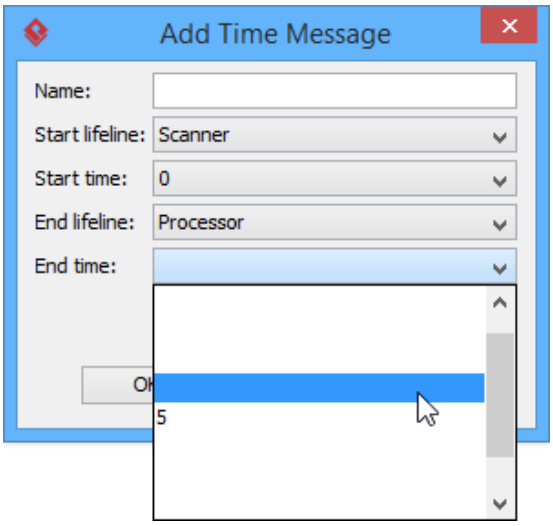
Edit frame

In the **Edit Frame** window, open the **Time Messages** tab and click **Add...** button.



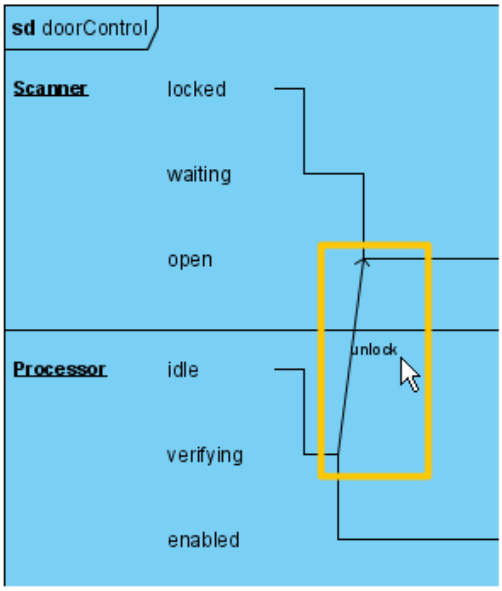
Add time message

When the **Add Time Message** window pops out, enter name and select the start lifeline, start time, end lifeline and end time for this time message. Note that as time units may be unnamed, when selecting start/end time you should check the relative position of the time unit in the list.



Select end time of time message

The time message is shown on the frame.

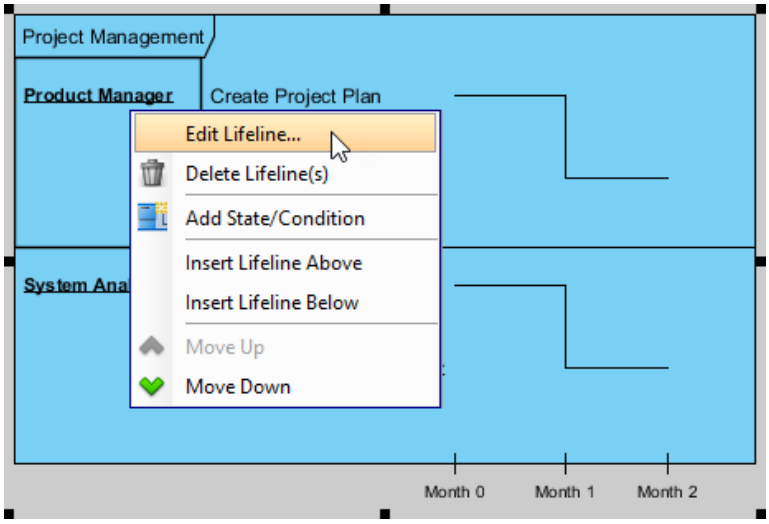


Time message

Adding duration constraint

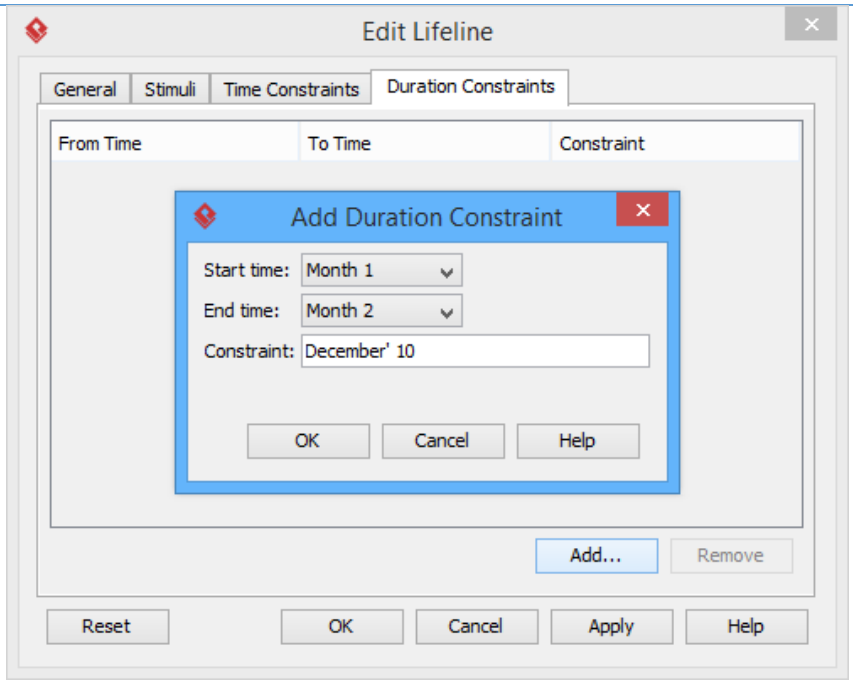
Duration constraint is used to show the duration limitation of a particular lifeline over a period of time.

1. To set the duration constraints of a lifeline, right-click on the lifeline and select **Edit Lifeline...** from the pop-up menu.



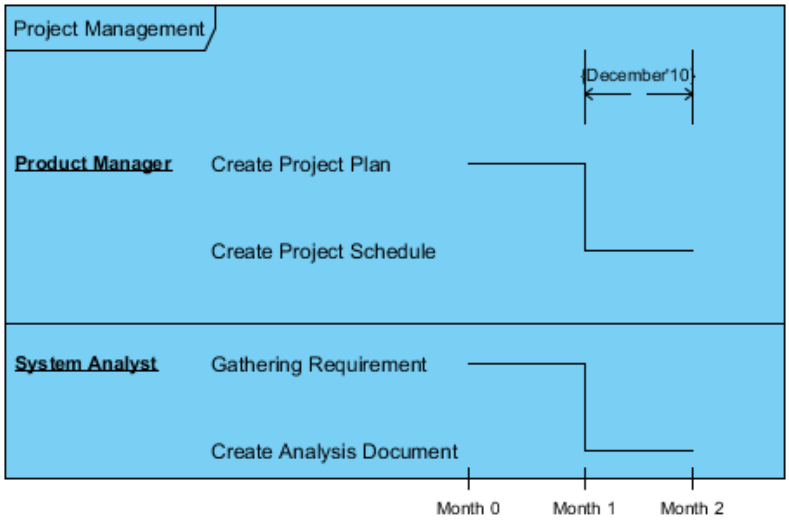
Edit lifeline

2. In the **Duration Constraints** tab, click on the **Add...** button. In the **Add Duration Constraint** window, select the appropriate **Start time** and **End time** from the drop down menu. Fill in the duration constraint of the selected time on the **Constraint** field. Click on the **OK** button to close the window.



Add duration constraint

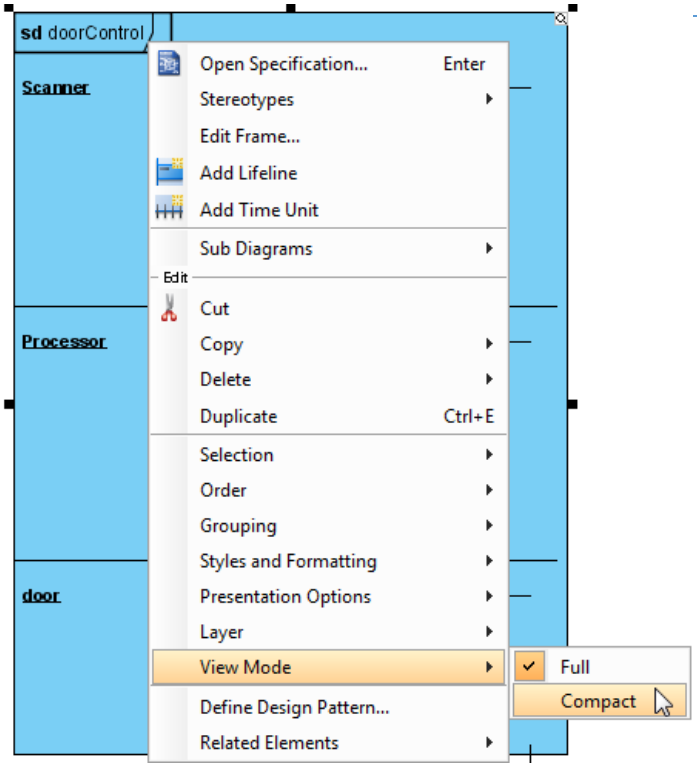
3. Click **OK** to return to diagram.



Duration constraint is added

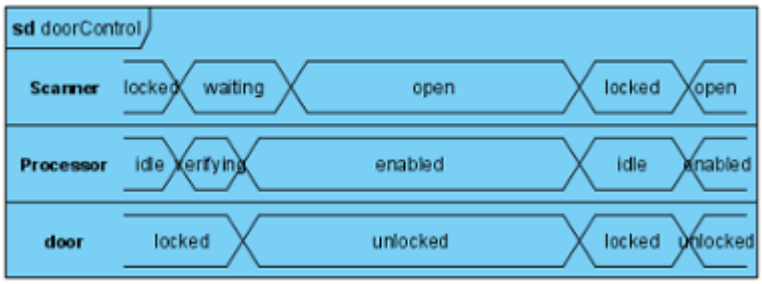
Switching to compact view mode

To switch to compact view mode, right-click the frame and select **View Mode** > **Compact** from the popup menu.



Switch to compact view mode

The frame will be shown in compact mode.



Frame shown in compact mode

STUDENT TASK-01

LAB 11 (II)

OBJECTIVE

After this lab students will be able to get;

- Familiarize with the concepts underlining Interaction overview diagram

PRE-LAB READING ASSIGNMENT

Basics of Interaction overview diagram

LAB RELATED CONTENT

Interaction overview diagram is a kind of UML diagram. It is the variant of UML activity diagram that shows specifically the flow of internation diagrams like sequence diagrams.

SOFTWARE TOOL

Visual paradigm

PROCEDURE

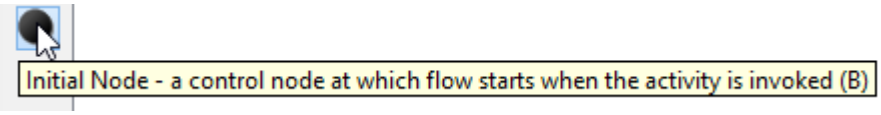
Creating interaction overview diagram

Perform the steps below to create an UML interaction overview diagram in Visual Paradigm.

1. Select **Diagram > New** from the application toolbar.
2. In the **New Diagram** window, select **Interaction Overview Diagram**.
3. Click **Next**.
4. Enter the diagram name and description. The **Location** field enab enables you to select a model to store the diagram.
5. Click **OK**.

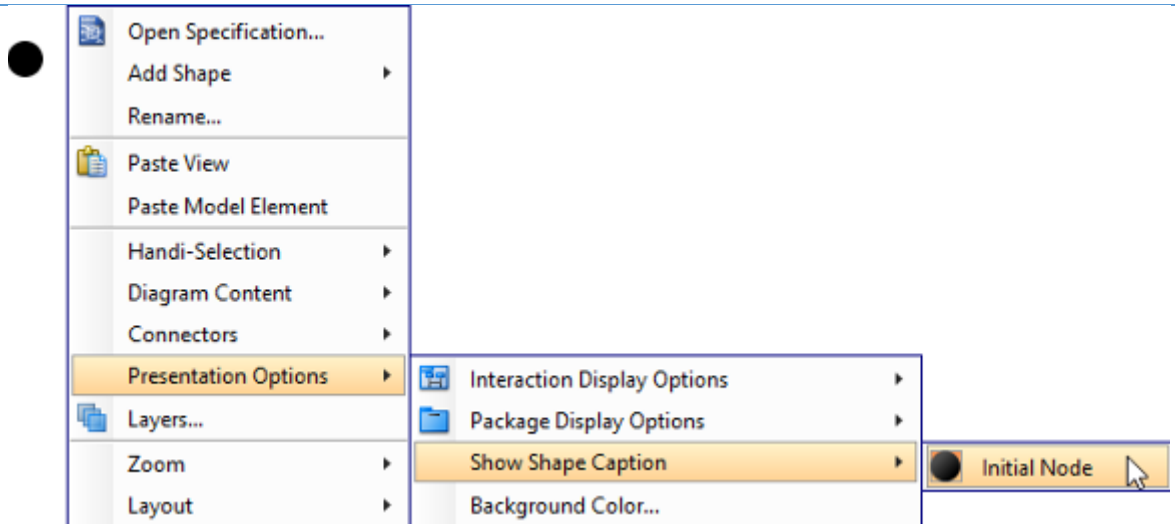
Creating initial node

Initial node is the beginning of a control flow. To create initial node in interaction overview diagram, click **Initial Node** on the diagram toolbar and then click on the diagram.



Create initial node

An initial node is created. The caption of initial node is hidden by default, to show it, right-click on the diagram and select **Presentation Options** > **Show Shape Caption** > **Initial Node** from the pop-up menu.

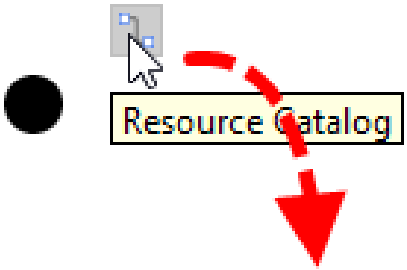


Show caption of initial node

Creating decision node

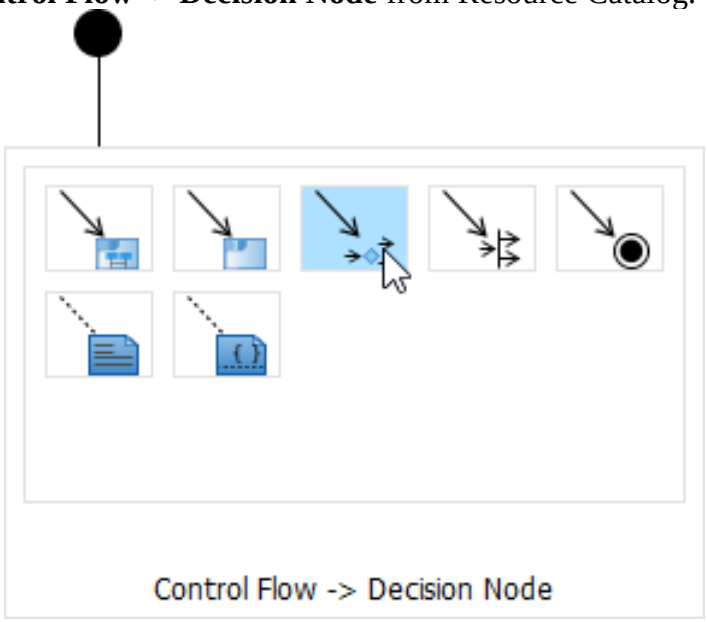
To create a decision node from an initial node:

1. Move your mouse pointer over the initial node.
2. Press on the **Resource Catalog** button and drag it out.



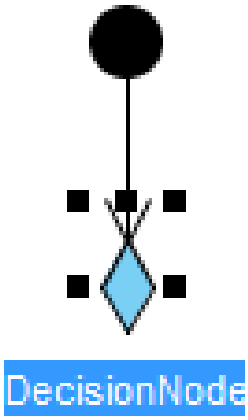
Using Resource Catalog

3. Release the mouse button at the place where you want the decision node to be created.
4. Select **Control Flow -> Decision Node** from Resource Catalog.



To create a decision node

5. A new decision node will be created and is connected to the initial node. Enter its name and press **Enter** to confirm editing.

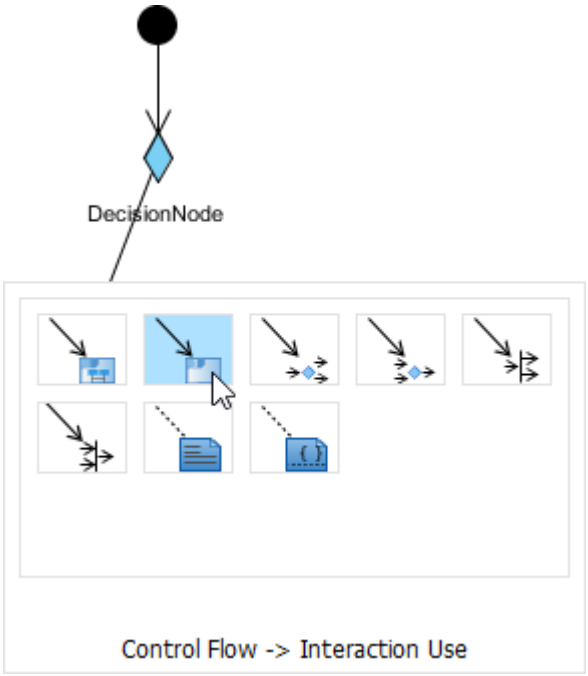


Decision node created

Creating interaction use

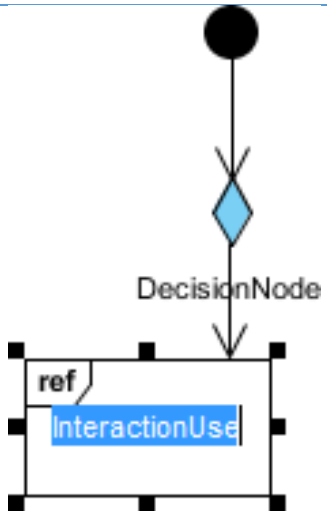
To create an interaction use:

1. Move your mouse pointer over the source shape.
2. Press on the **Resource Catalog** button and drag it out.
3. Release the mouse button at the place where you want the interaction use to be created.
4. Select **Control Flow -> Interaction Use** from Resource Catalog.



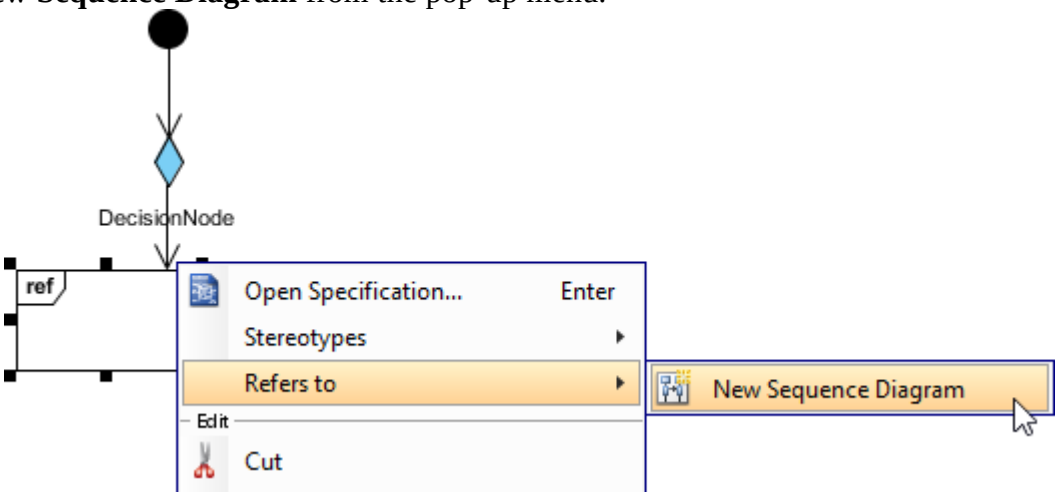
To create an interaction use

5. A new interaction use will be created and is connected to the source node. Enter its name and press **Enter** to confirm editing.



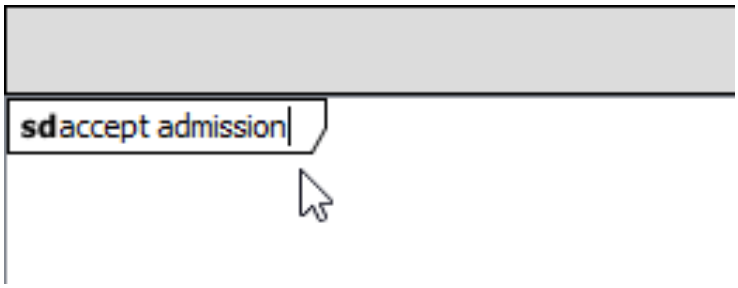
Interaction use created

You can make the interaction use refers to a diagram by right clicking on it and select **Refers to** > **New Sequence Diagram** from the pop-up menu.



Make interaction use refers to diagram

When sequence diagram is created, rename the diagram.



Rename sequence diagram

When you return to the interaction overview diagram, you can see the interaction use caption shows the diagram it refers to.

Interaction use caption updated

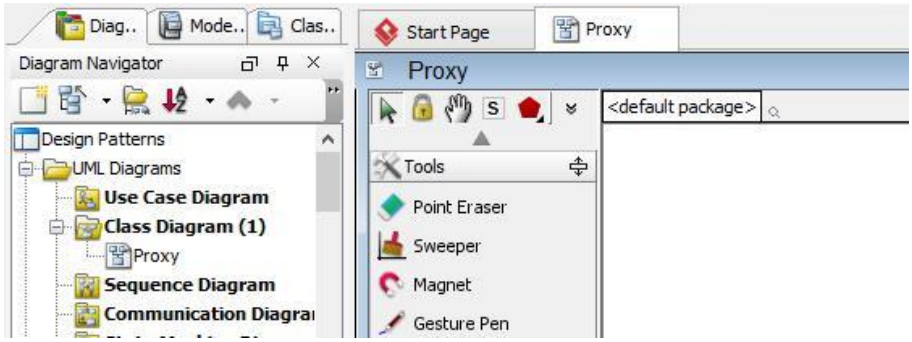
STUDENT TASK-01

LAB-12

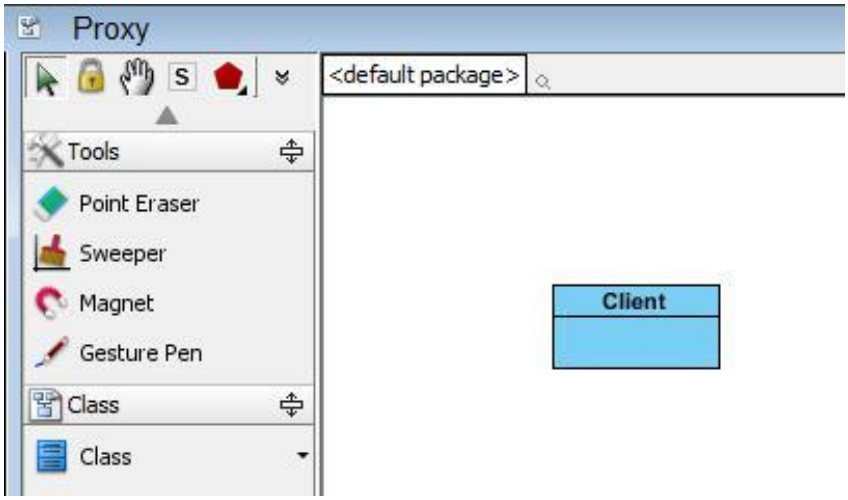
Modeling Design Pattern with Class Diagram

1) Create a new project *Design Patterns*.

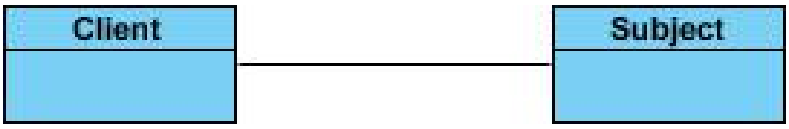
2) Create a class diagram *Proxy*.



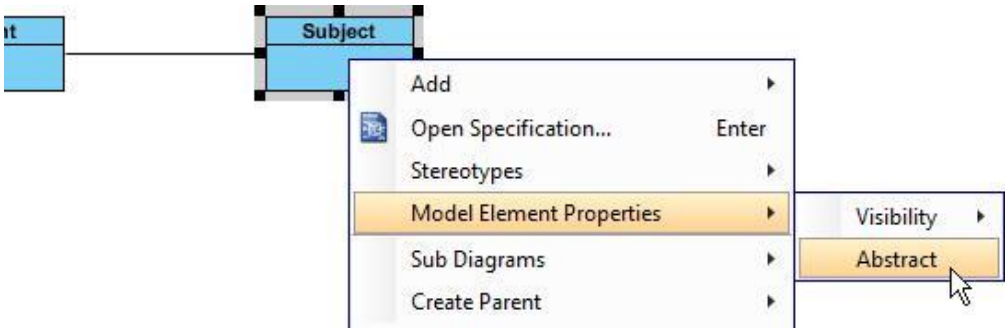
3. Select **Class** from diagram toolbar. Click on the diagram to create a class. Name it as *Client*.



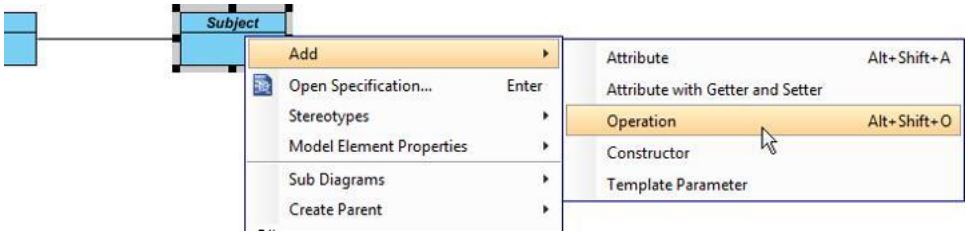
4. Move the mouse cursor over the *Client* class, and drag out **Association** > **Class** to create an associated class *Subject*.



5. Right-click on *Subject*, and select **Model Element Properties** > **Abstract** to set it as abstract.

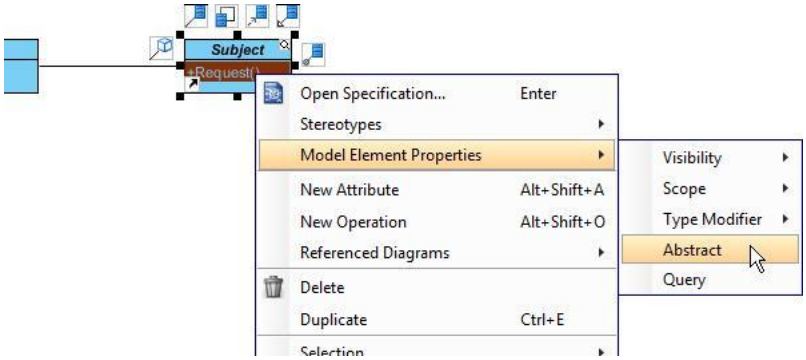


6. Right-click on the *Subject* class, and select **Add** > **Operation** from the popup menu.

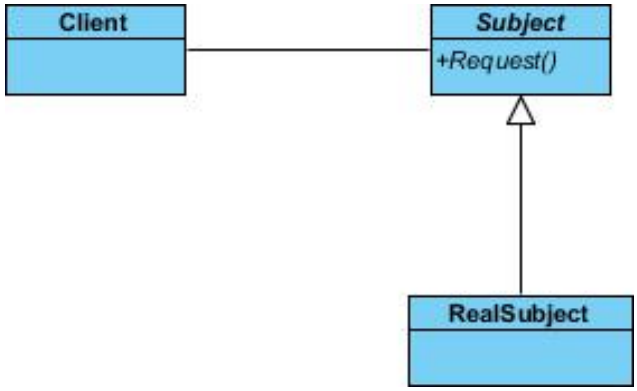


7. Name the operation *Request()*.

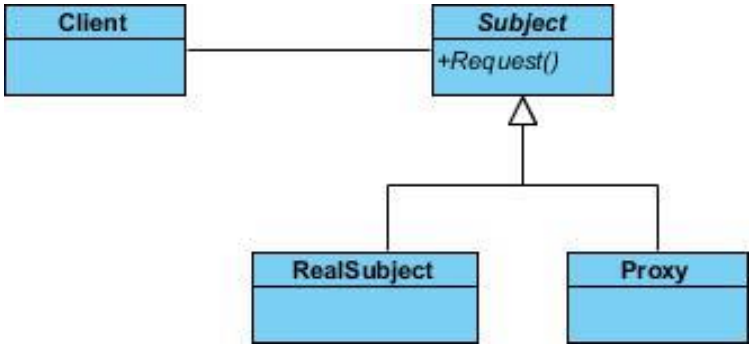
8. Right-click on *Request*, and select **Model Element Properties** > **Abstract** to set it as abstract.



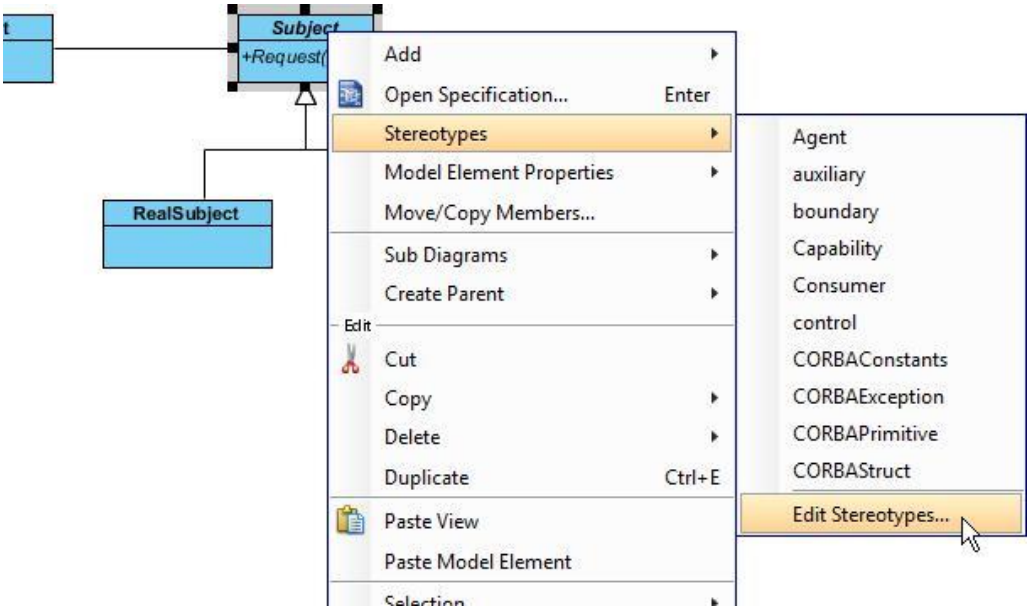
9. Move the mouse cursor over the *Subject* class, and drag out **Generalization** > **Class** to create a subclass *RealSubject*.



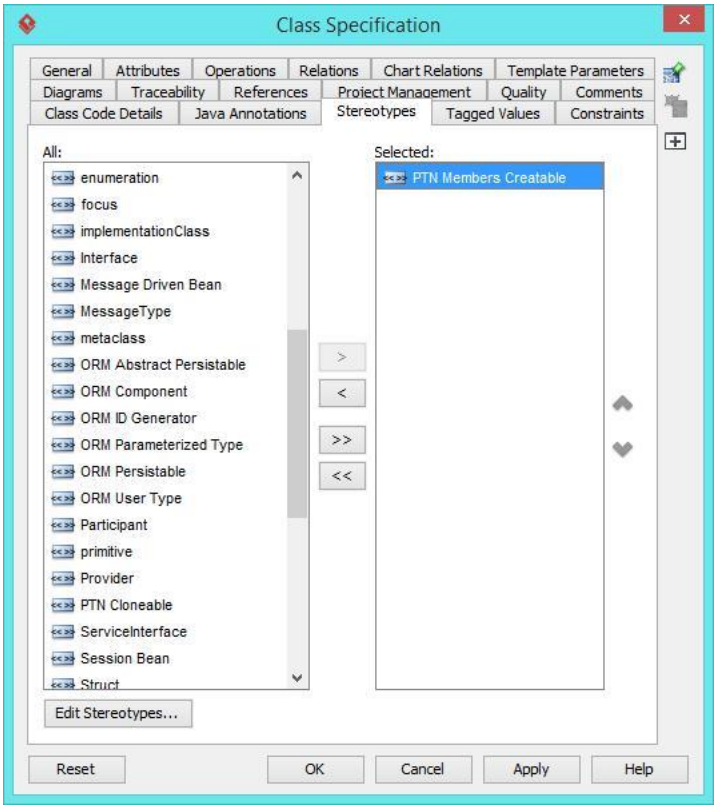
10. Repeat the previous step to create another subclass from *Subject*, namely *Proxy*.



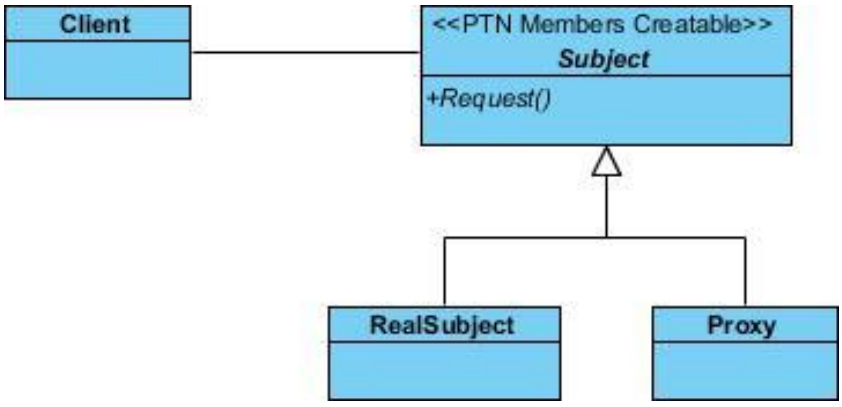
11. In practice, there may be multiple requests. To represent this, stereotype the class *Subject* as **PTN Members Creatable**. Right-click on *Subject* and select **Stereotypes** > **Stereotypes...** from the popup menu.



12. In the **Stereotypes** tab of the **Class Specification** dialog box, select **PTN Members Creatable** and click > to assign it to *Subject* class. Click **OK** to confirm.

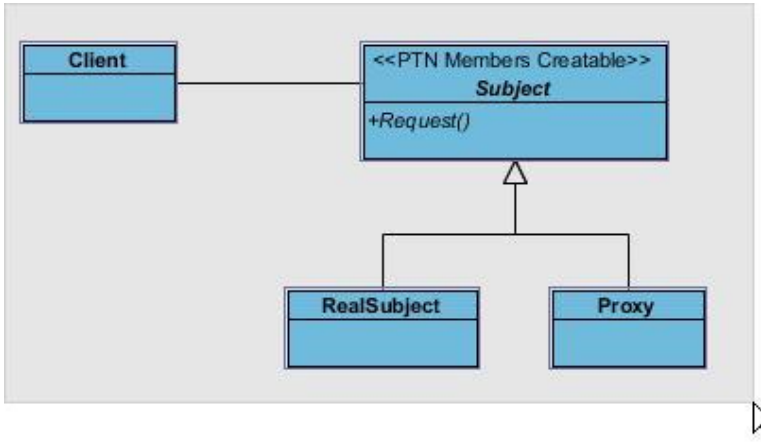


Up to now, the diagram should look like this:

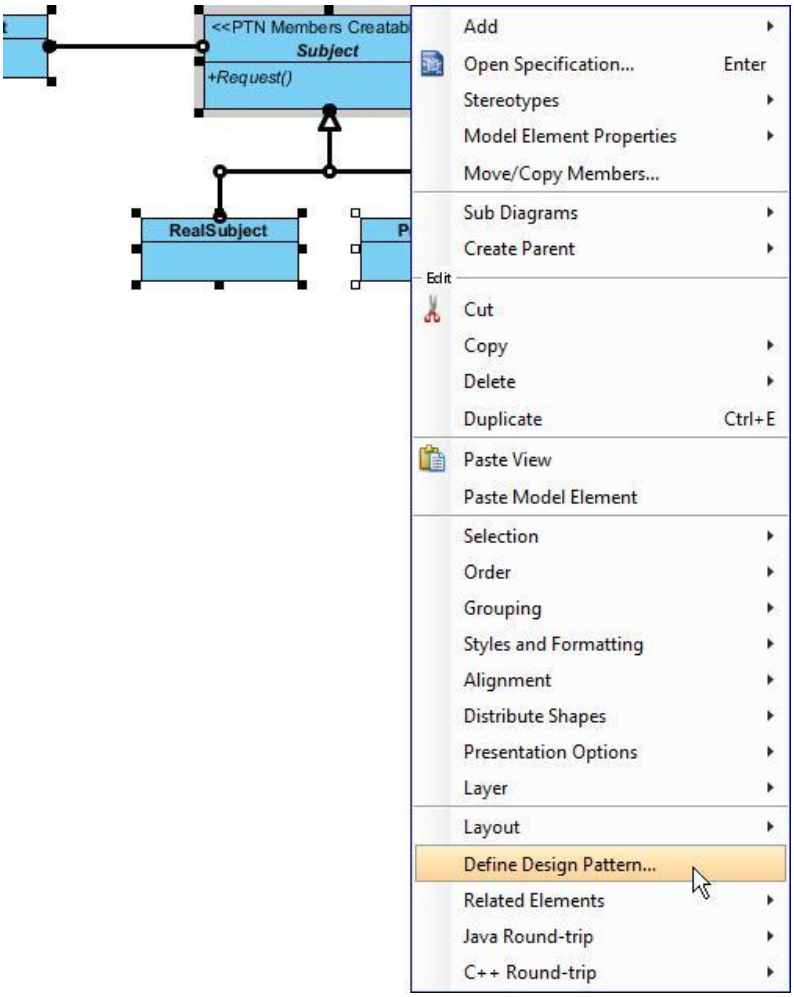


Defining Pattern

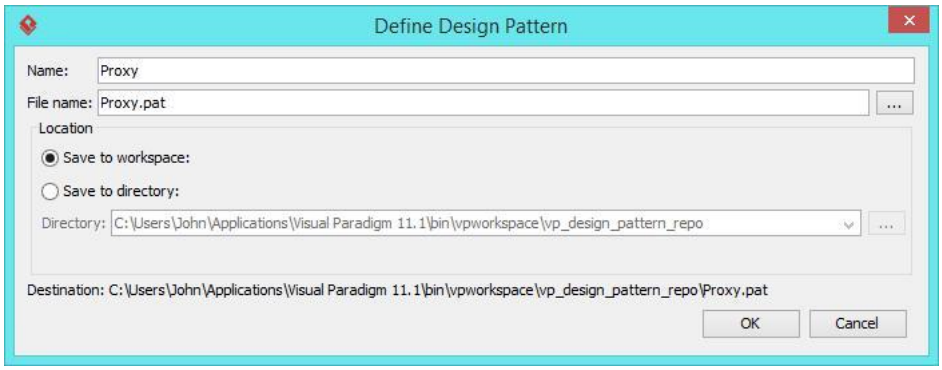
1. Select all classes on the class diagram.



2. Right-click on the selection and select **Define Design Pattern...** from the popup menu.



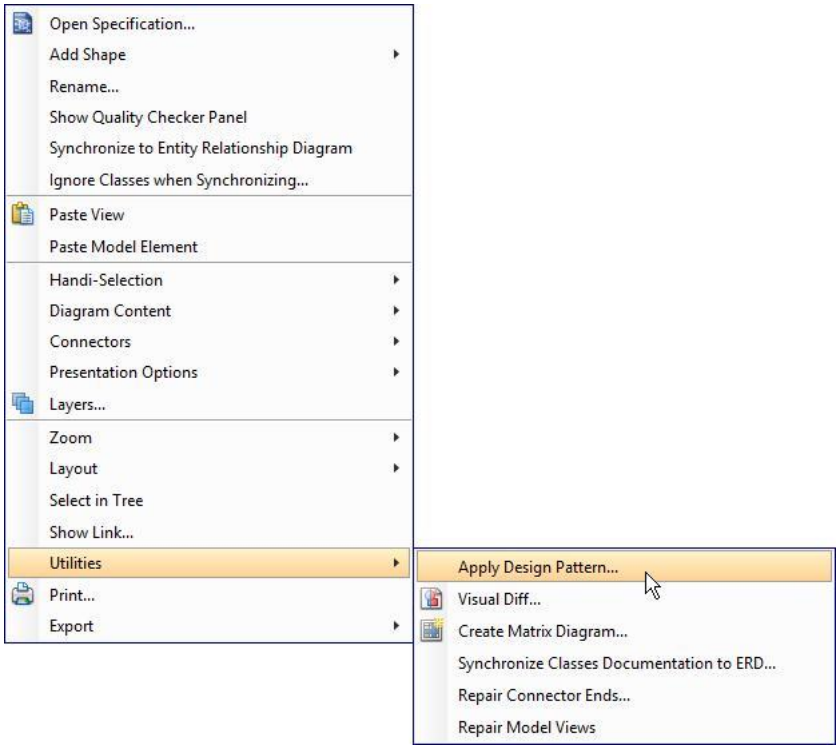
3. In the **Define Design Pattern** dialog box, specify the pattern name *Proxy*. Keep the file name as it. Click **OK** to proceed.



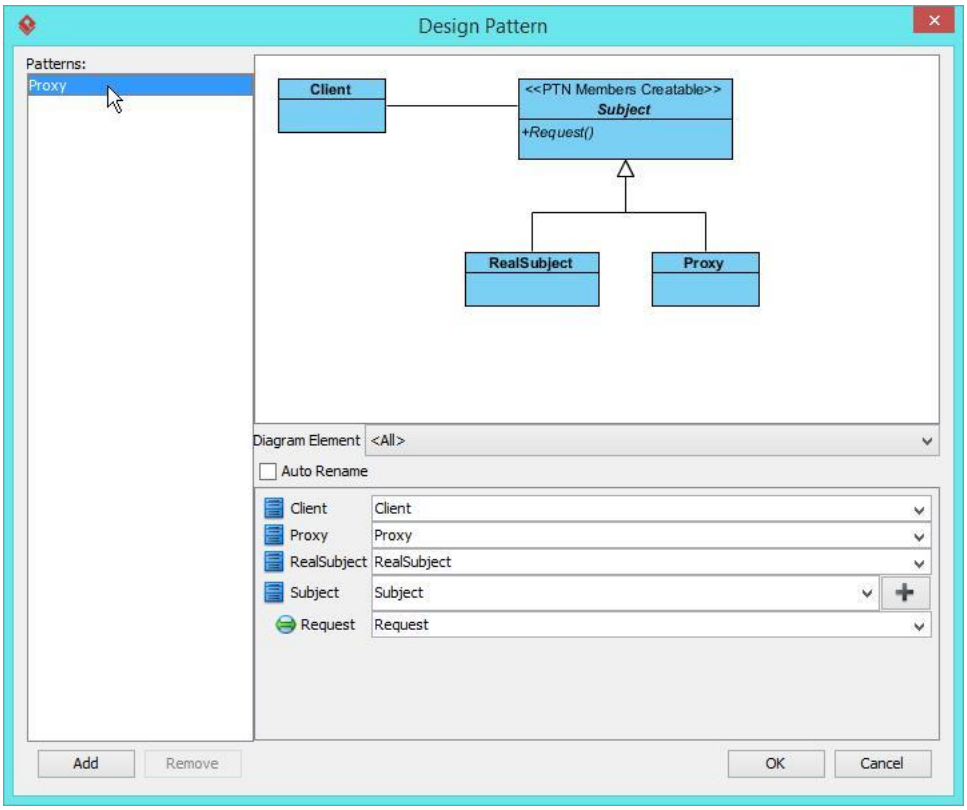
Applying Design Pattern on Class Diagram

In this section, we are going to apply the proxy pattern in modeling a client class that talks to a proxy account class.

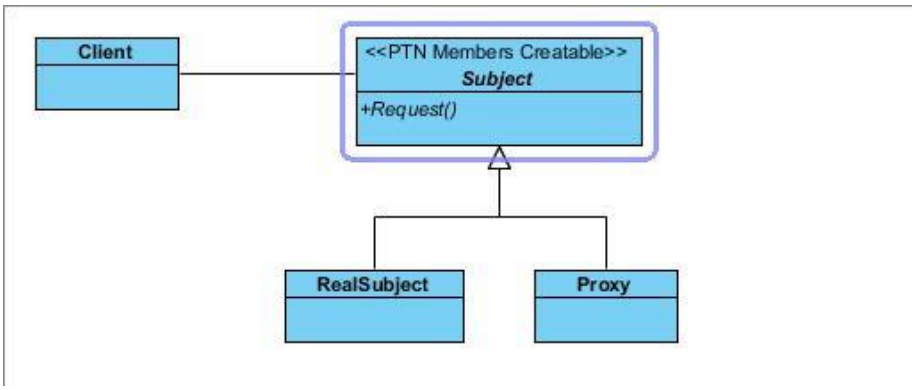
1. Create a new project *Account Management*.
2. Create a class diagram *Domain Model*.
3. Right-click on the class diagram and select **Utilities** > **Apply Design Pattern...** from the popup menu.



4. In the **Design Pattern** dialog box, select *Proxy* from the list of patterns.



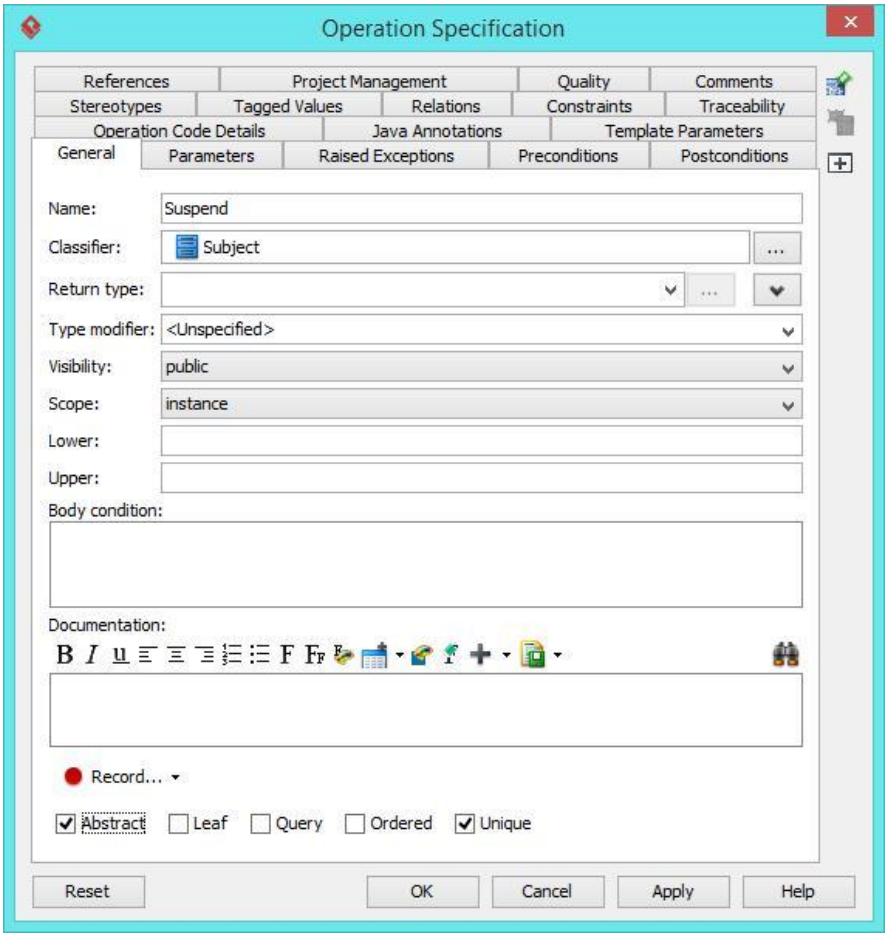
5. Click on *Subject* in the overview.



6. Rename *Subject* to *Account*, and operation *Request* to *Create* at the bottom pane.

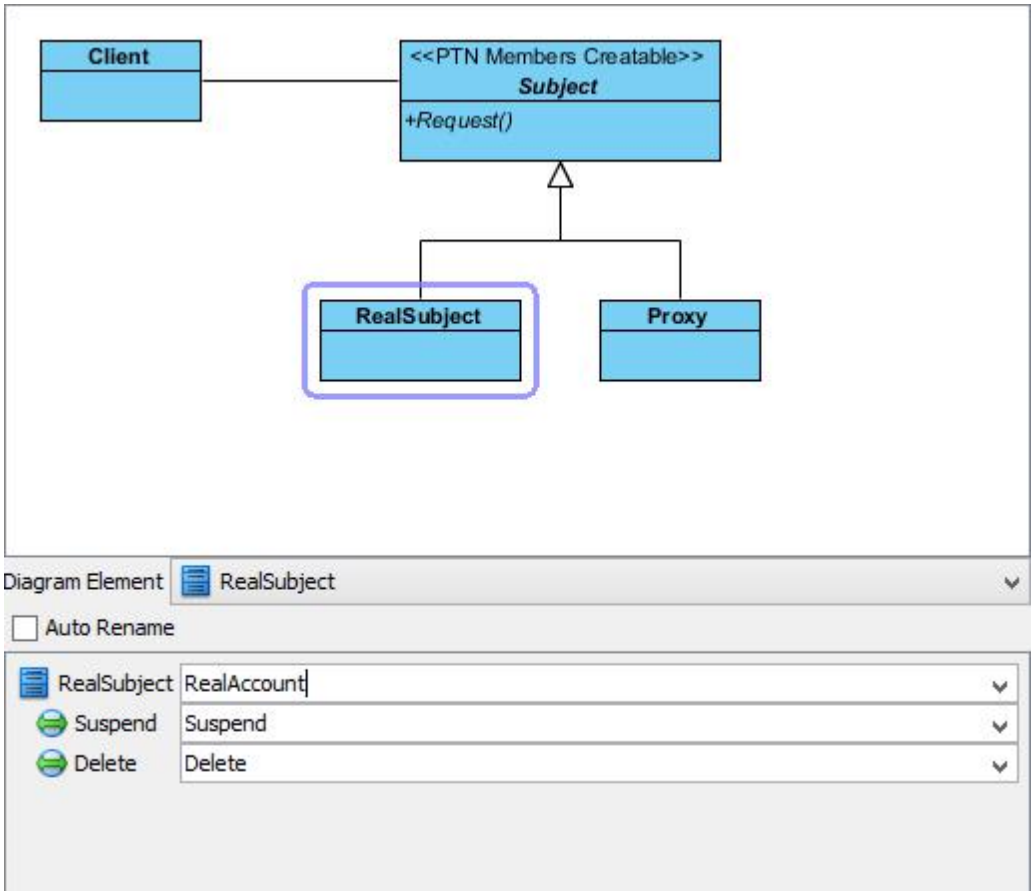
1. Besides the operation *Create*, we also need two more operations for *Suspend* and *Delete*. Keep *Subject* selected, click on the + button at the bottom pane, and select **New Operation...** from the popup menu.

8. In the **Operation Specification** dialog box, name the operation *Suspend*.

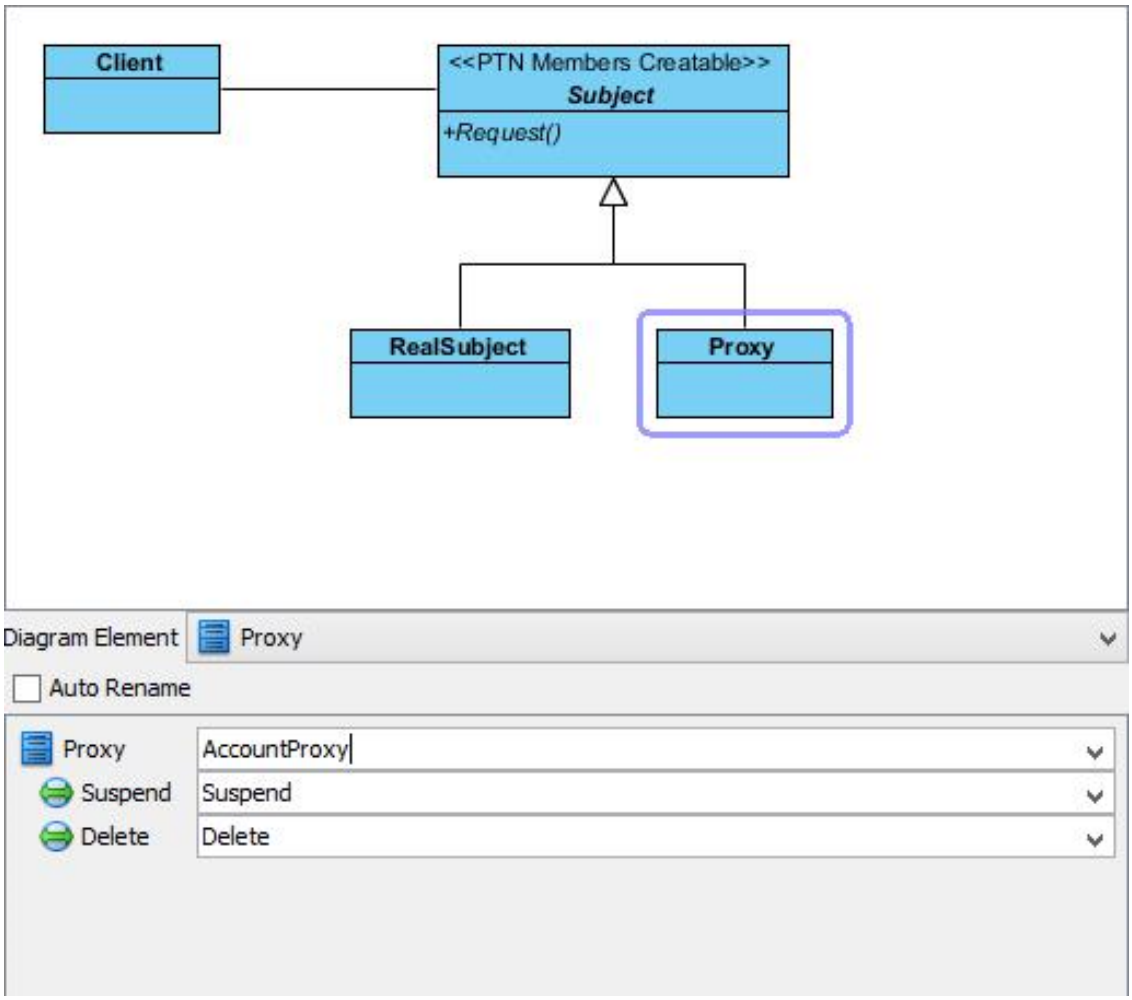


9. Repeat steps 7 and 8 to create operation *Delete*.

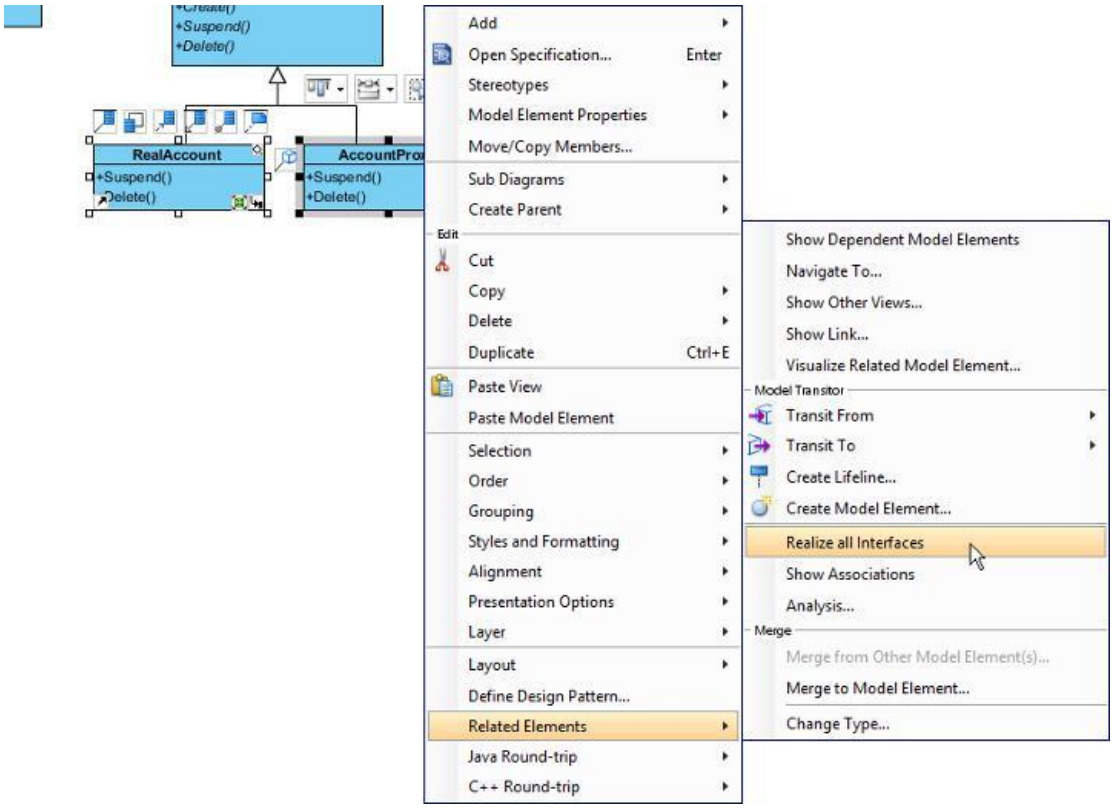
10. Select *RealSubject* in overview, and rename it as *RealAccount* at the bottom pane.



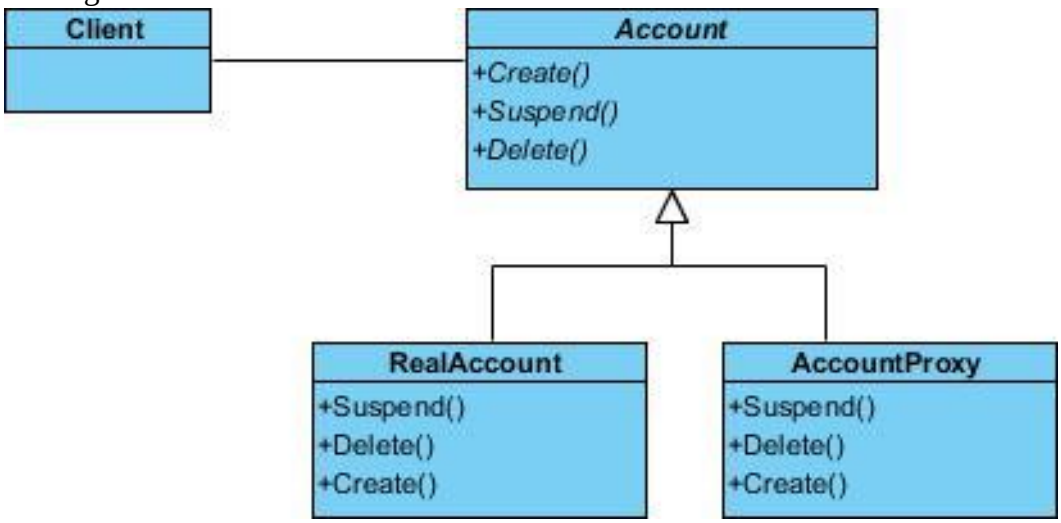
11. Select *Proxy* in overview, and rename it as *AccountProxy* at the bottom pane. Click **OK** to apply the pattern to diagram.



12. We need to make the real object and the proxy class inherit operations from the subject class. Right-click on *RealAccount* and *AccountProxy*, and select **Related Elements** > **Realize all Interfaces** from the popup menu.



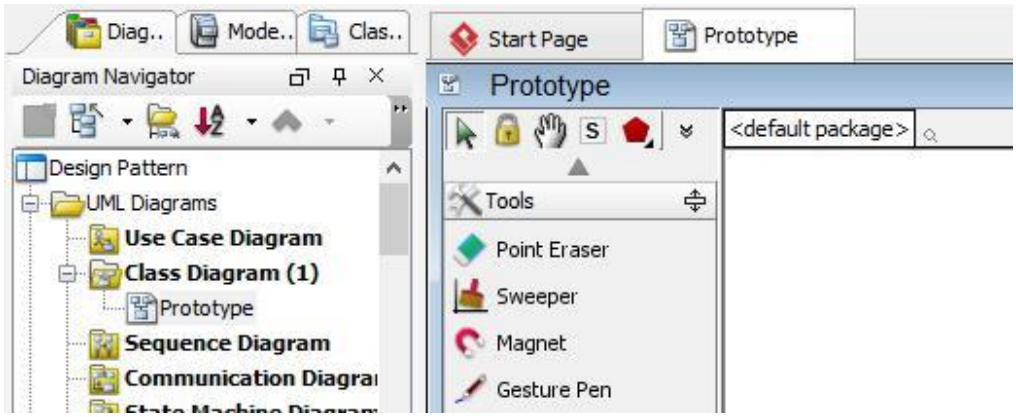
The diagram should look like:



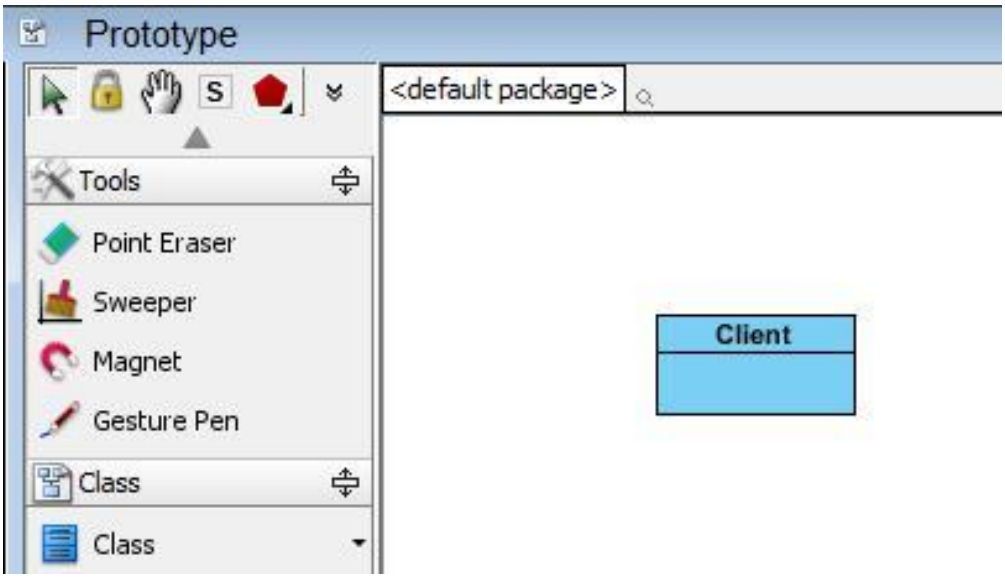
LAB-13

Modeling Design Pattern with Class Diagram

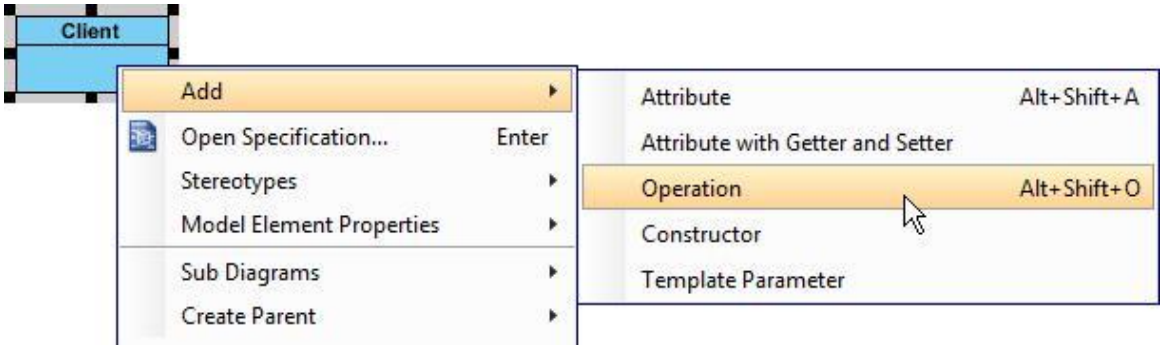
1. Create a new project *Design Pattern.*
2. Create a class diagram *Prototype*.



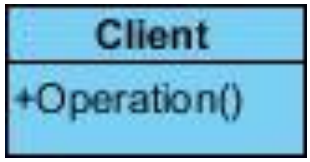
3. Select **Class** from diagram toolbar. Click on diagram to create a class. Name it as *Client*.



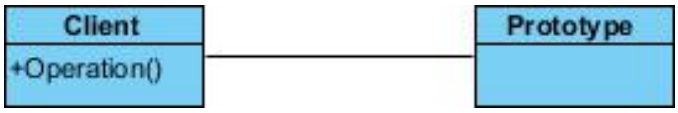
4. Right-click on the *Client* class, and select **Add** > **Operation** from the popup menu.



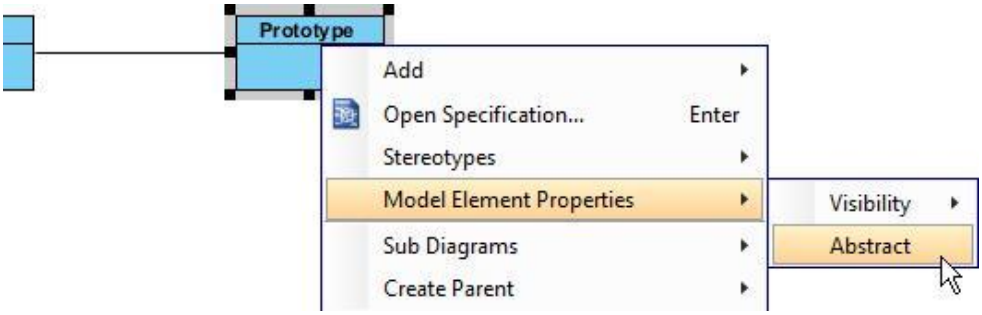
5. Name the operation *Operation()*.



6. Move the mouse cursor over the *Client* class, and drag out **Association** > **Class** to create an associated class *Prototype*.

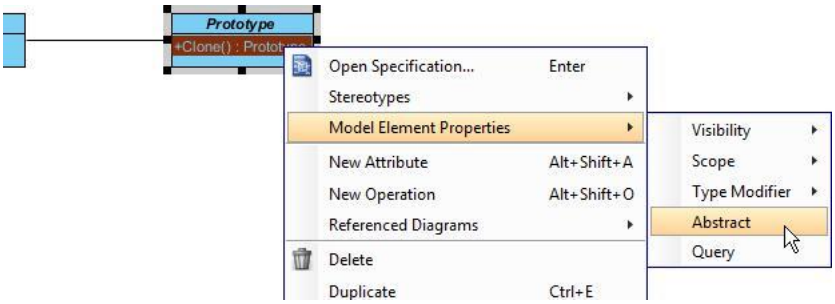


7. Right-click on *Prototype*, and select **Model Element Properties** > **Abstract** to set it as abstract.

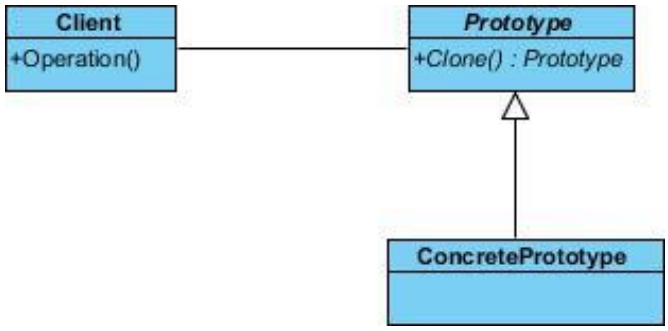


8. Add an operation *Clone()* to *Prototype*. Make it return *Prototype*.

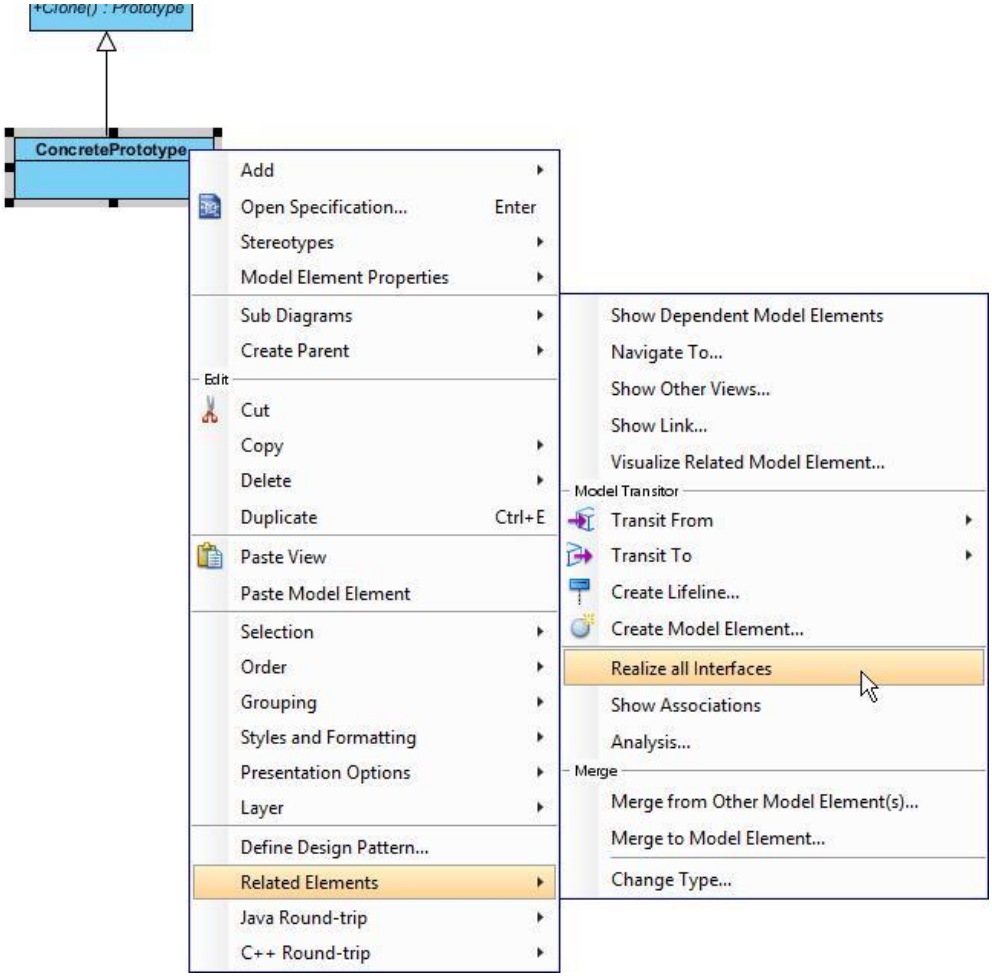
9.Right-click on *Clone()*, and select **Model Element Properties** > **Abstract** to set it as abstract.



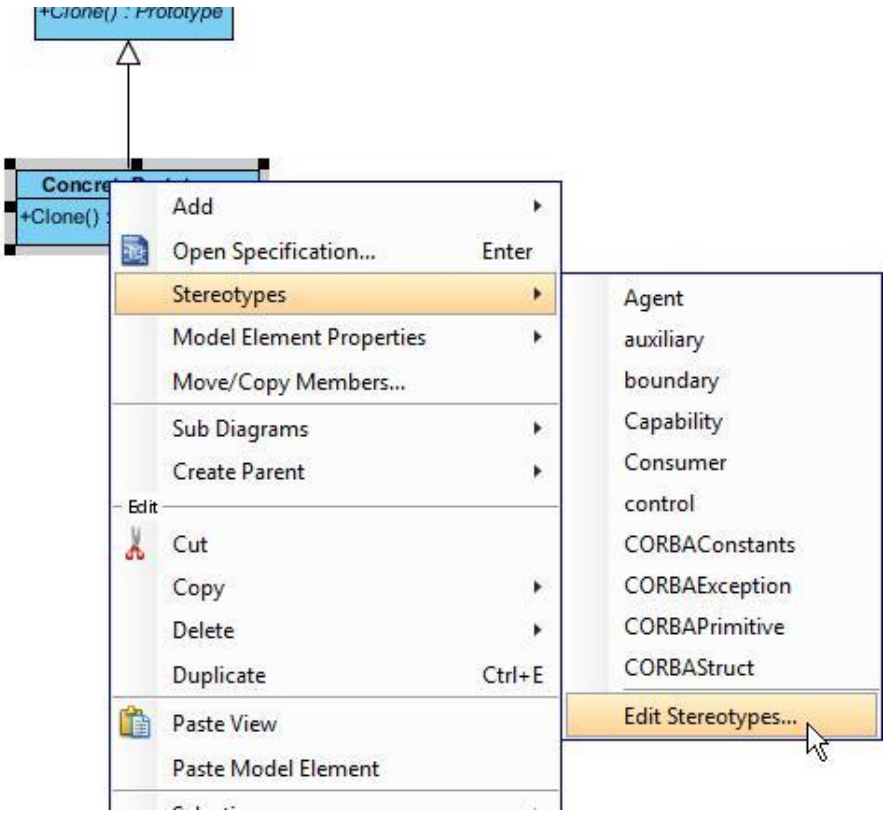
10. Move the mouse cursor over the *Prototype* class, and drag out **Generalization** > **Class** to create a subclass *ConcretePrototype*.



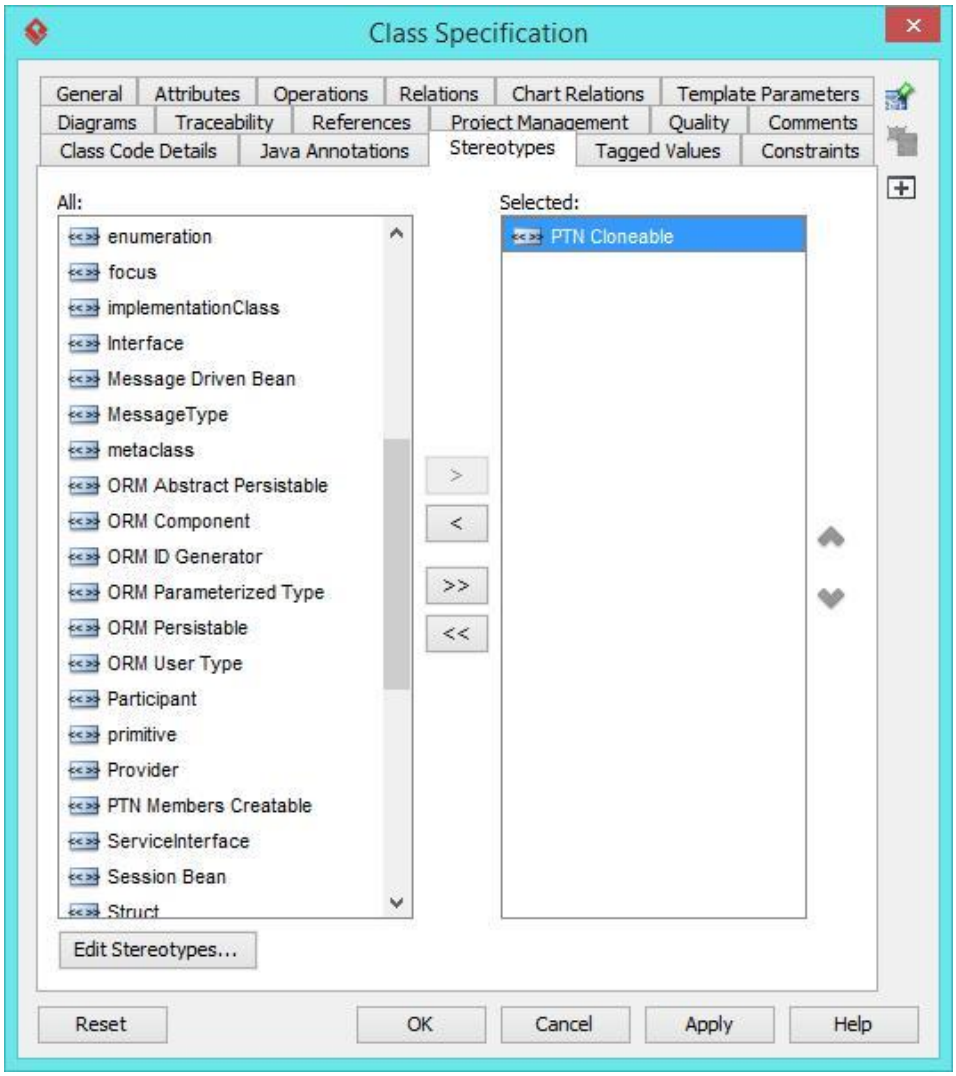
11. Make *ConcretePrototype* inherit the abstract operations provided from *Prototype* by right clicking on *ConcretePrototype*, and selecting **Related Elements** > **Realize all Interfaces** from the popup menu.



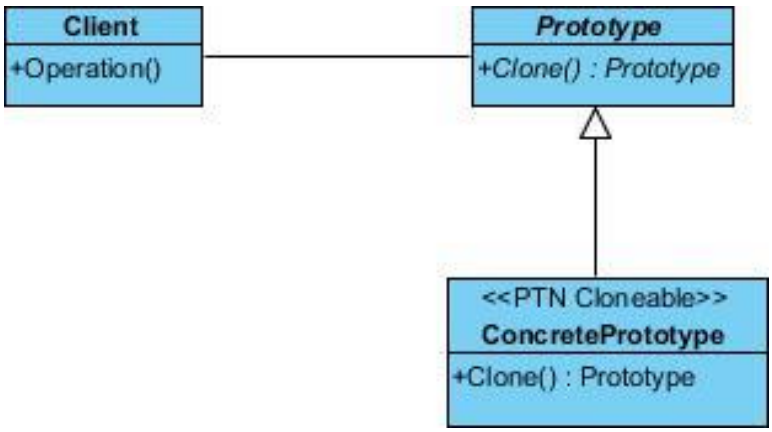
12. In practice, there may be multiple *ConcretePrototype* classes. To represent this, stereotype the *ConcretePrototype* class as **PTN Cloneable**. Right-click on *ConcretePrototype* class and select **Stereotypes** > **Stereotypes...** from the popup menu.



13. In the **Stereotypes** tab of class specification, select **PTN Cloneable** and click > to assign it to the class. Click **OK** to confirm.

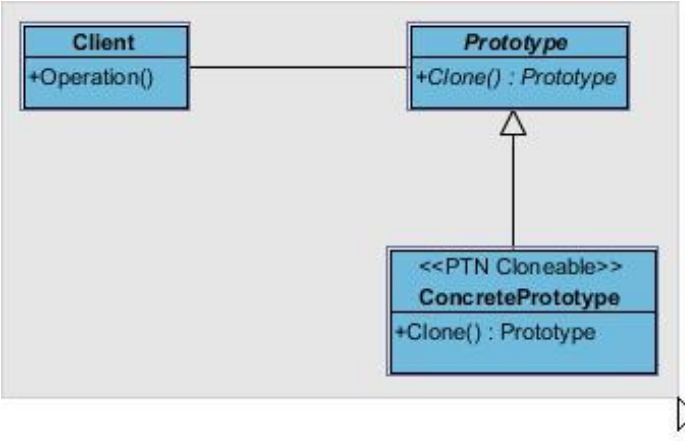


The diagram should look like this:

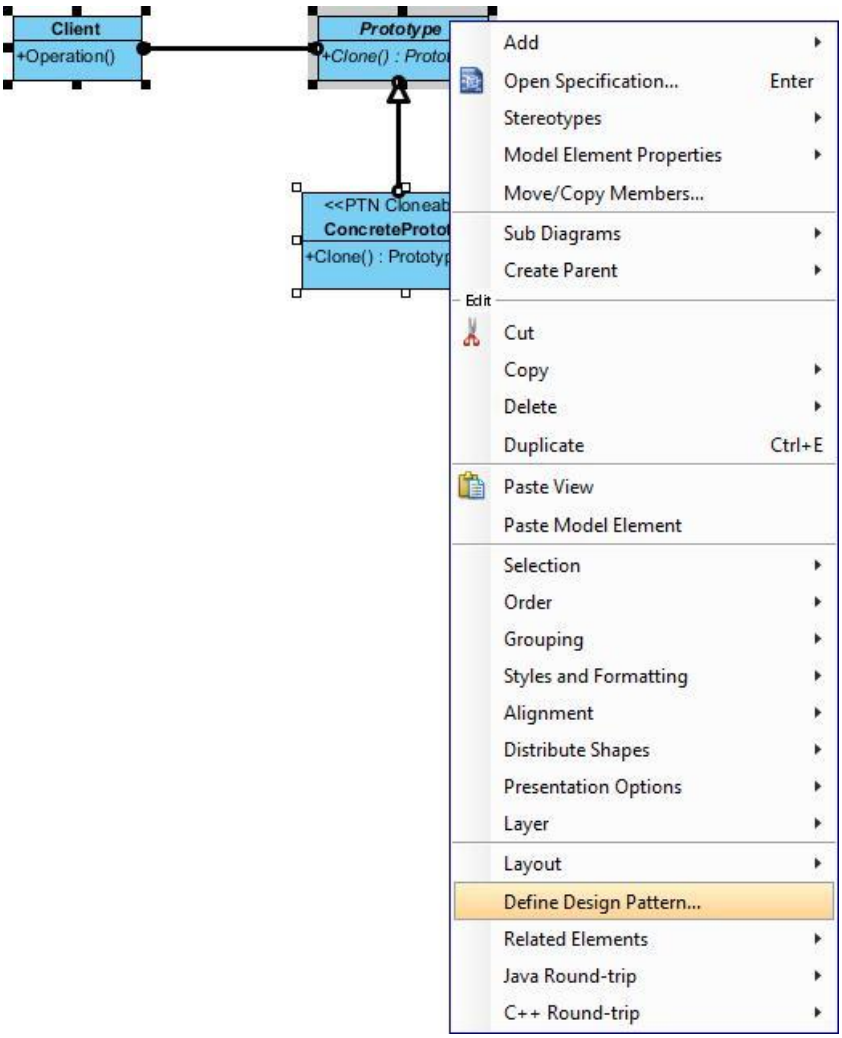


Defining Pattern

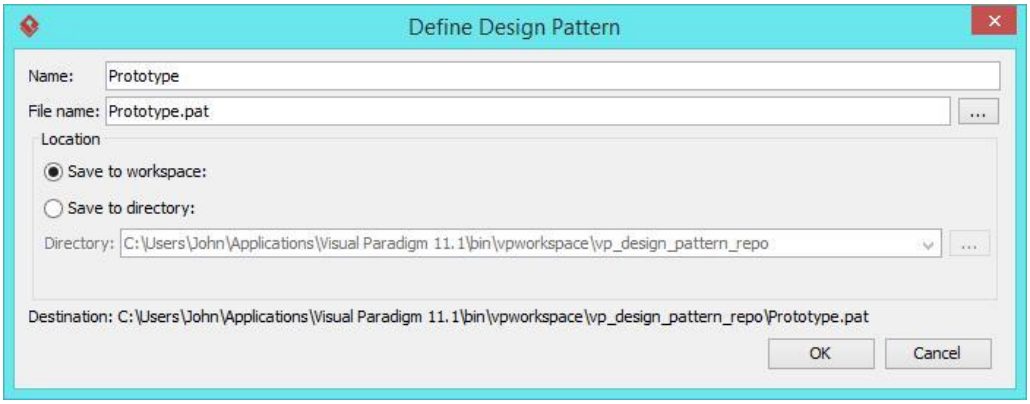
1. Select all classes on the class diagram.



2. Right-click on the selection and select **Define Design Pattern...** from the popup menu.



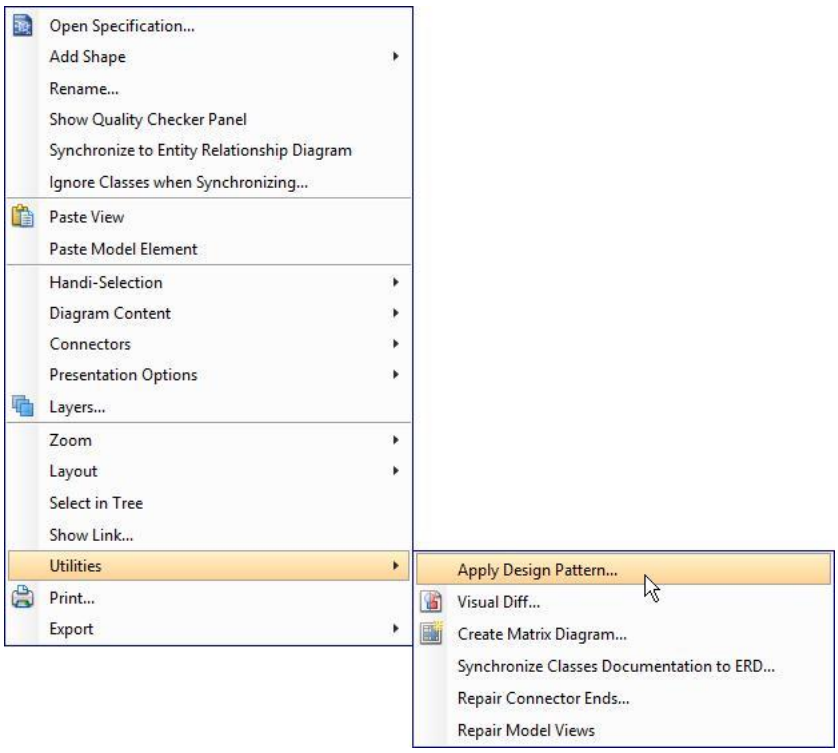
3. In the **Define Design Pattern** dialog box, specify the pattern name *Prototype*. Keep the file name as it. Click **OK** to proceed.



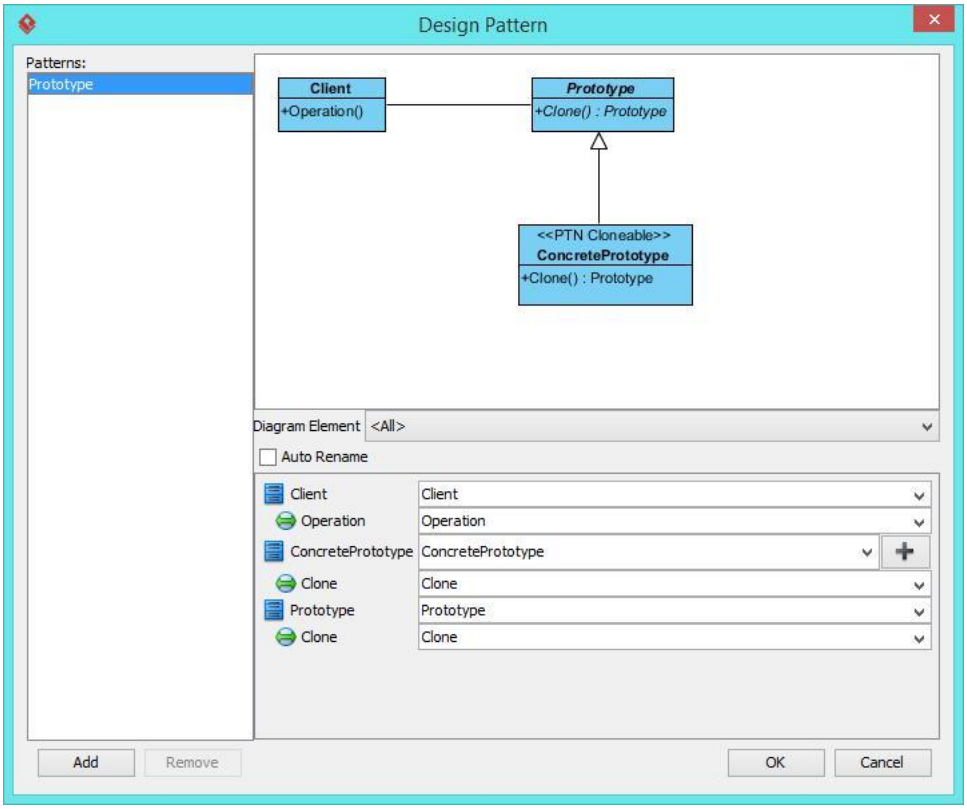
Applying Design Pattern on Class Diagram

In this section, we will try to make use of the prototype pattern to model a part of diagram editor.

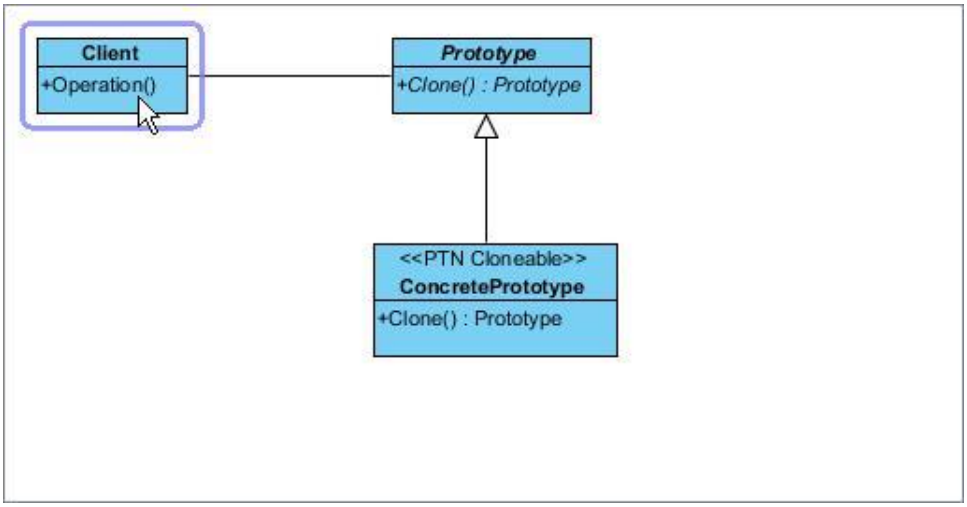
1. Create a new project *My Diagram Tool*.
2. Create a class diagram *Domain Model*.
3. Right-click on the class diagram and select **Utilities** > **Apply Design Pattern...** from the popup menu.



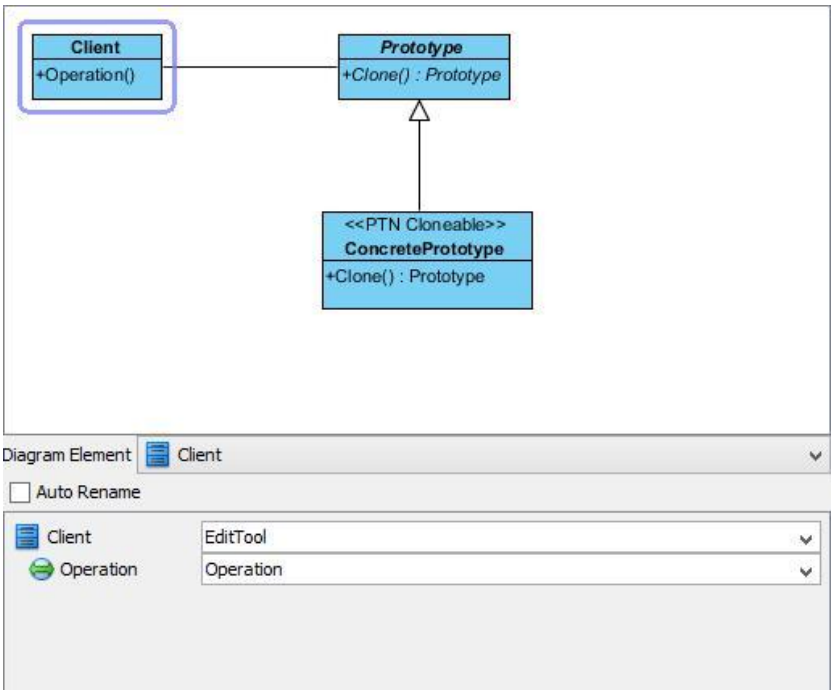
4. In the **Design Pattern** dialog box, select *Prototype* from the list of patterns.



5. Click on *Client* in the overview.



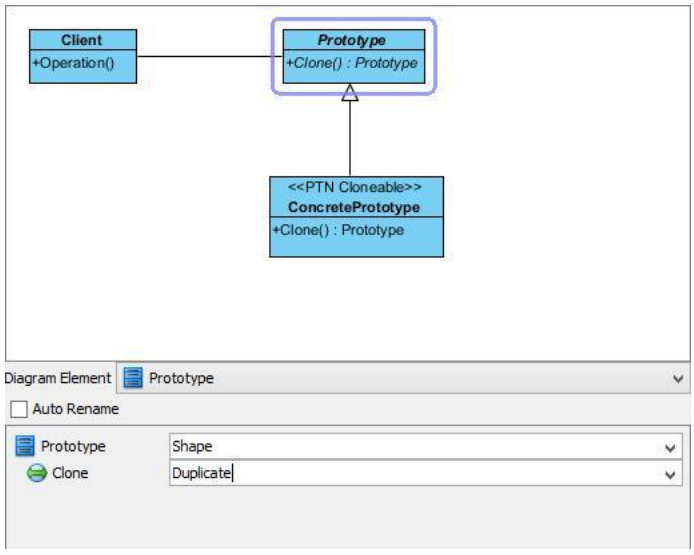
6. Rename it to *EditTool* at the bottom pane.



7. Rename *Operation* to *Duplicate*.

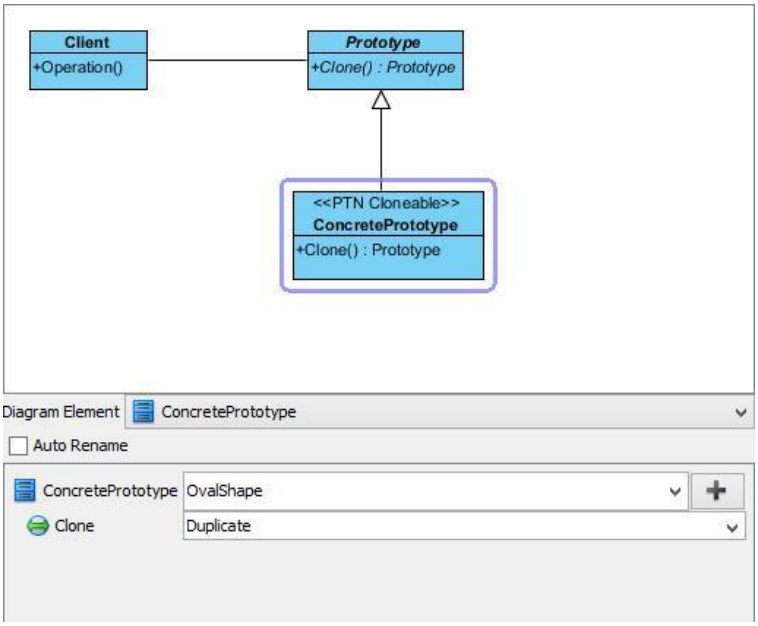
8. Select *Prototype* in overview.

9. Rename *Prototype* to *Shape* at the bottom pane, and rename the operation *Clone* to *Duplicate*.



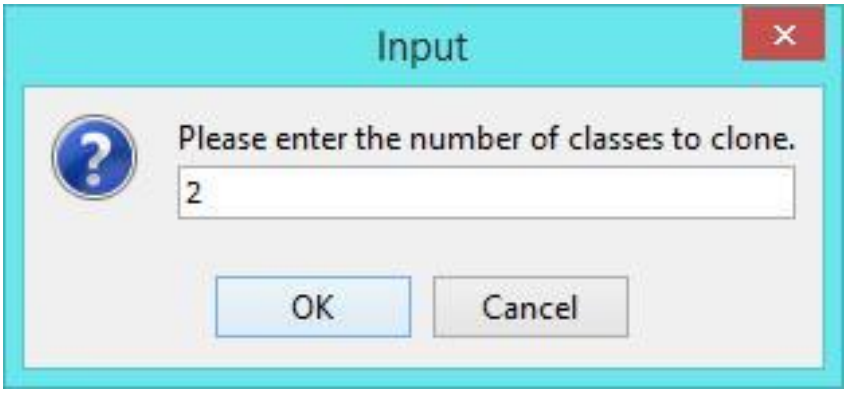
10. Select *ConcretePrototype* in overview.

11. Rename *ConcretePrototype* to *OvalShape* at the bottom pane, and rename the operation *Clone* to *duplicate*.

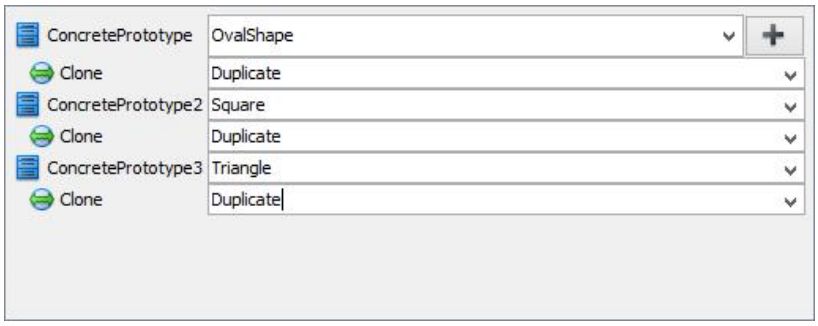


12. We need to have two more concrete prototype classes for square and triangle. Keep *ConcretePrototype* selecting, click the + button, and select **Clone...** from the popup menu.

13. Enter *2* to be the number of classes to clone.

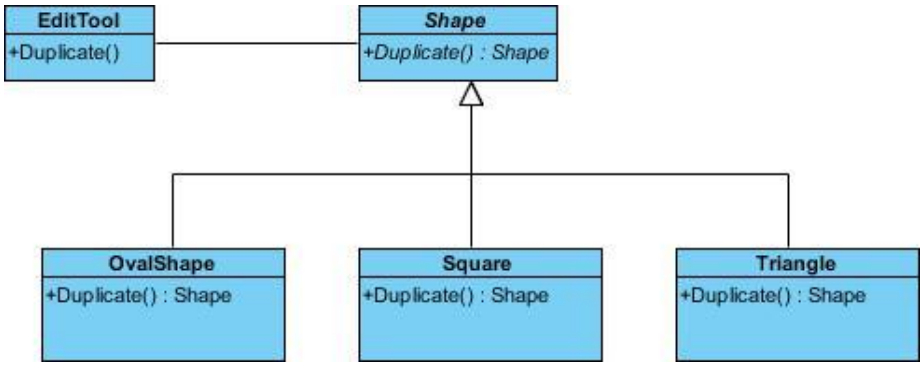


14. Rename *ConcretePrototype2* and *ConcretePrototype3* to *Square* and *Triangle* respectively. Rename the two *Clone* operations to duplicate.



15. Click **OK** to confirm editing and apply the pattern to diagram.

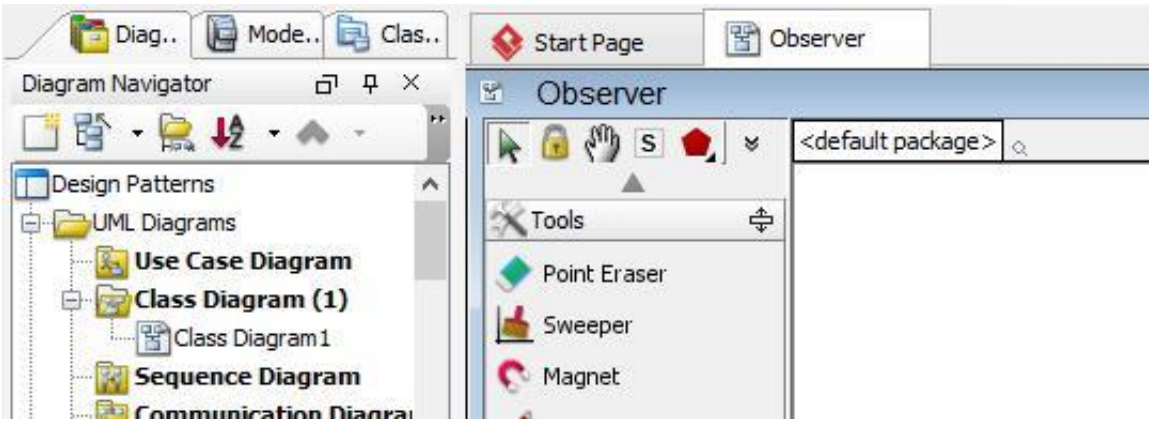
16. Tidy up the diagram. It should become:



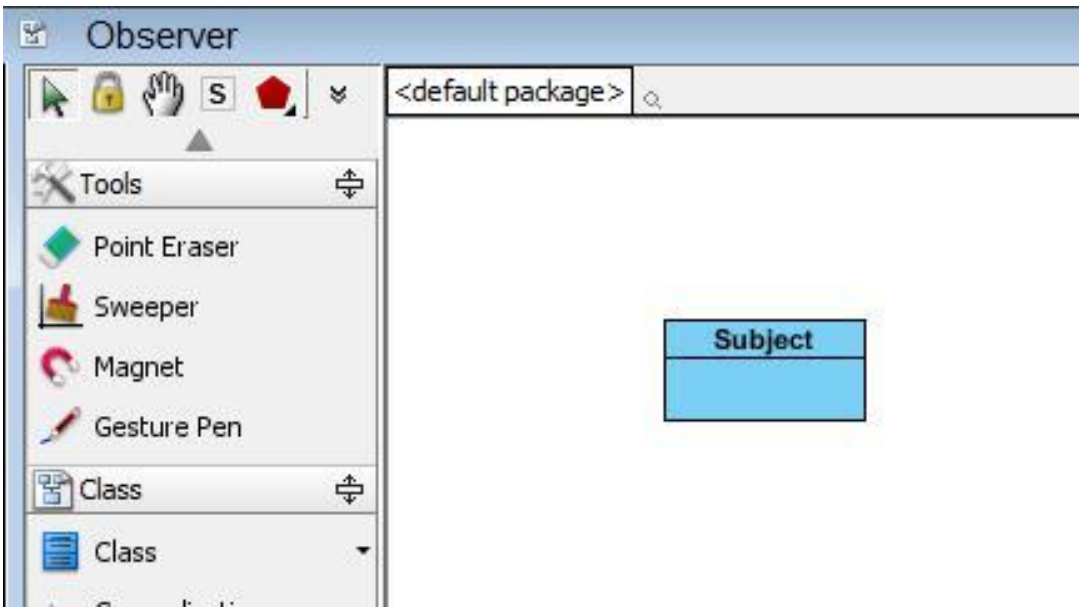
LAB-14

Modeling Design Pattern with Class Diagram

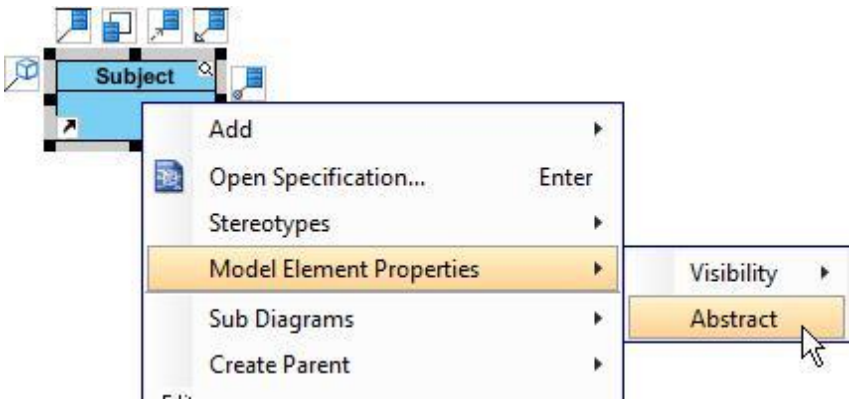
1. Create a new project *Design Patterns*.
2. Create a class diagram *Observer*.



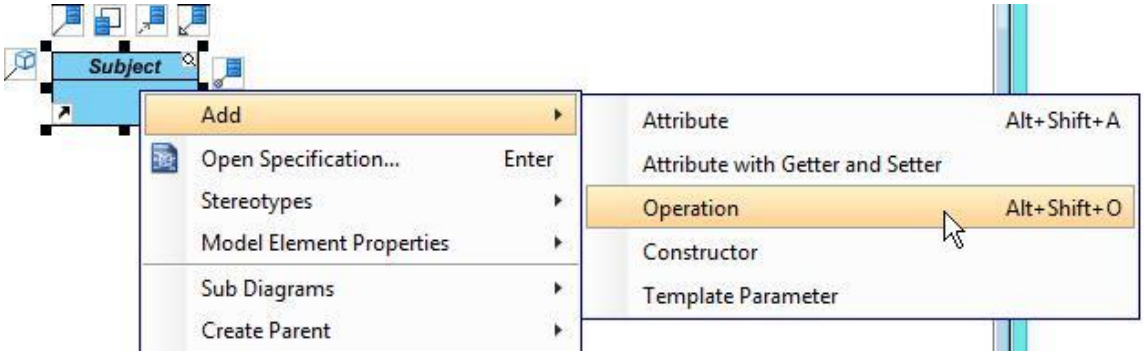
3. Select **Class** from diagram toolbar. Click on the diagram to create a class. Name it as *Subject*.



4. Right-click on *Subject*, and select **Model Element Properties** > **Abstract** to set it as abstract.

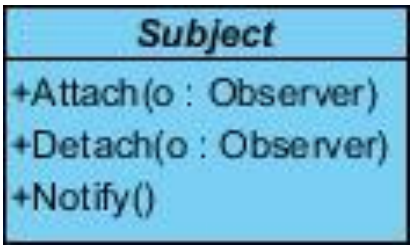


5. Right-click on *Subject* class, and select **Add** > **Operation** from the popup menu.

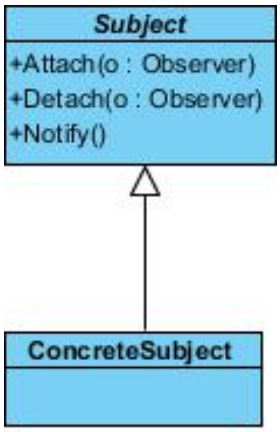


6. Name the operation *Attach(o : Observer)*.

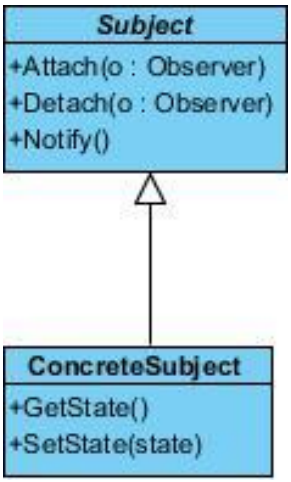
7. Repeat step 5 and 6 to create the remaining two operations : *Detach (o : Observer)*, *Notify()*.



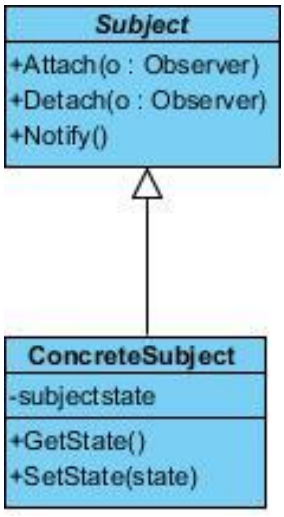
9. Move the mouse cursor over the *Subject* class, and drag out **Generalization** > **Class** to create a subclass *ConcreteSubject*.



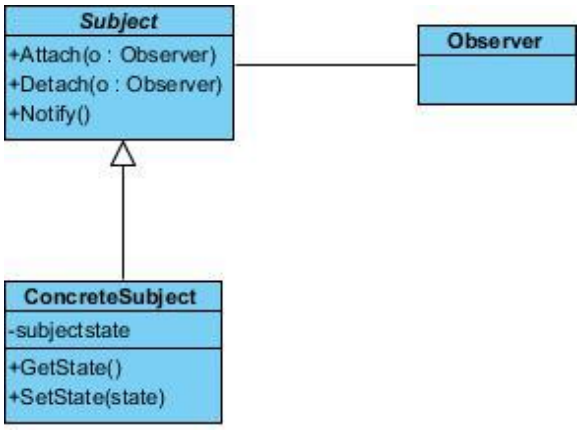
10. Repeat steps 5 and 6 to create the following operations in *ConcreteSubject: GetState(), SetState(state).*



11. Right-click on the *ConcreteSubject* class, and select **Add** > **Attribute** from the popup menu. Name the attribute *subjectstate*.

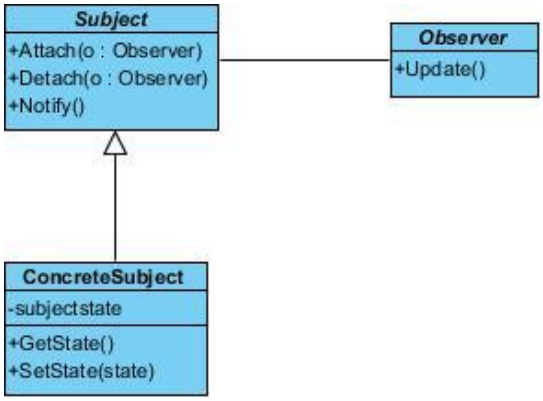


12. Move the mouse cursor over the *Subject* class, and drag out **Association** > **Class** to create an associated class *Observer*.

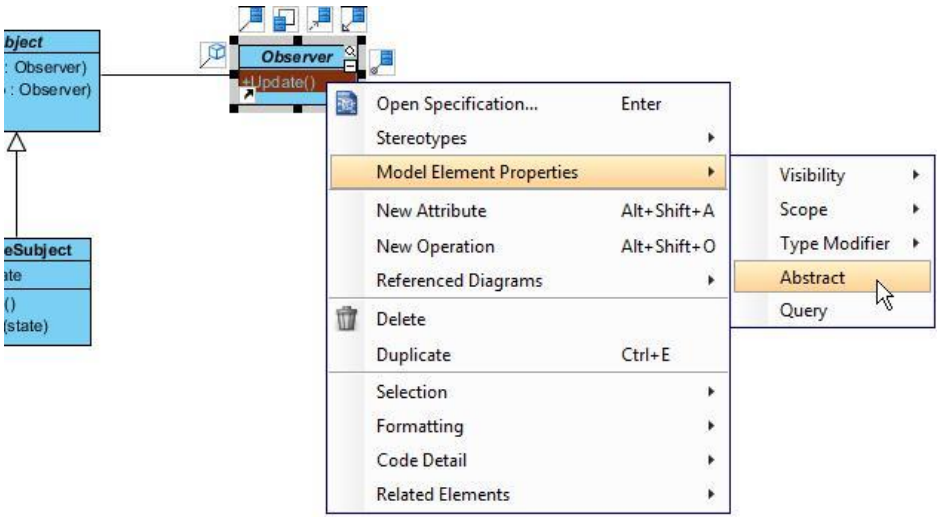


13. Right-click an *Observer*, and select **Model Element Properties** > **Abstract** to set it as abstract.

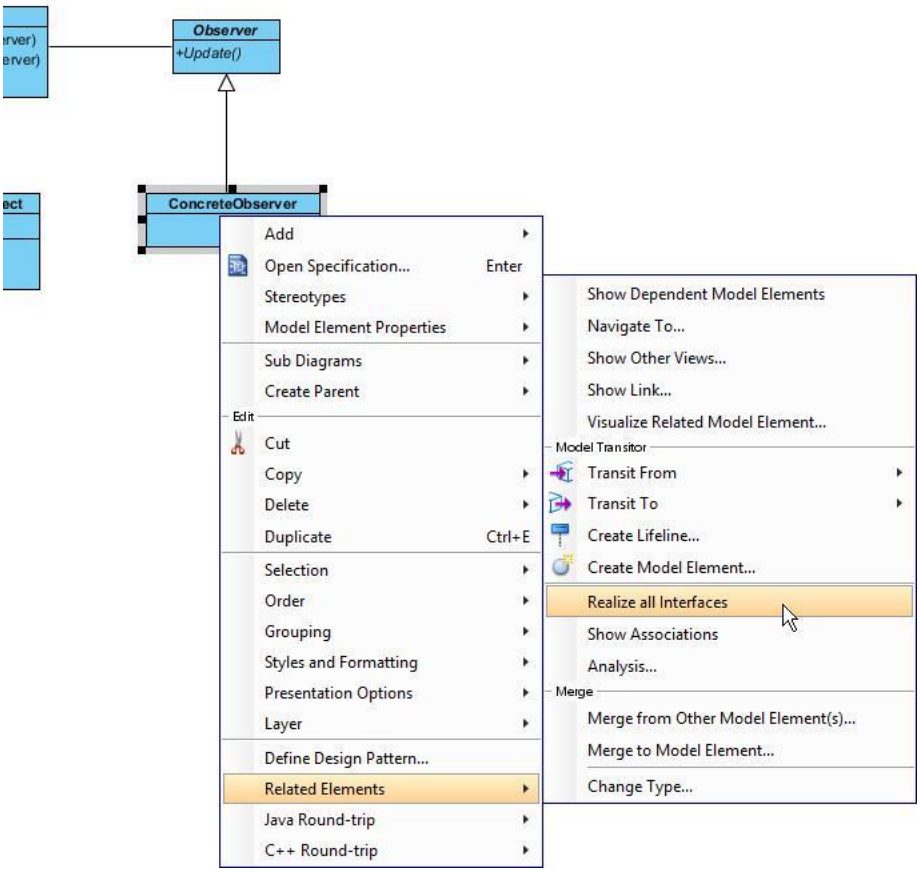
14. Repeat steps 5 and 6 to create the following operation in *Observer: Update()*.



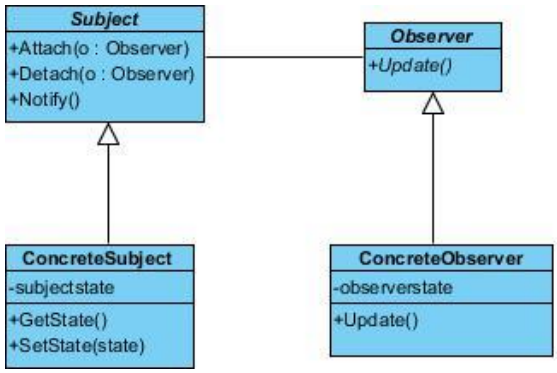
15. Right-click on *Update()*, and select **Model Element Properties** > **Abstract** to set it as abstract.



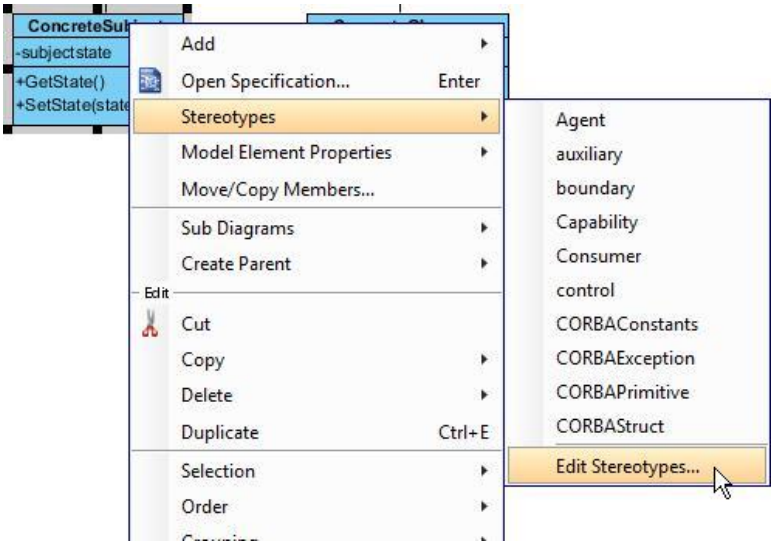
16. Move the mouse cursor over the *Observer* class, and drag out **Generalization** > **Class** to create a subclass *ConcreteObserver*.
17. *ConcreteObserver* will inherits the operations from *Observer*. Right-click on *ConcreteObserver* and select **Related Elements** > **Realize all Interfaces** from the popup menu.



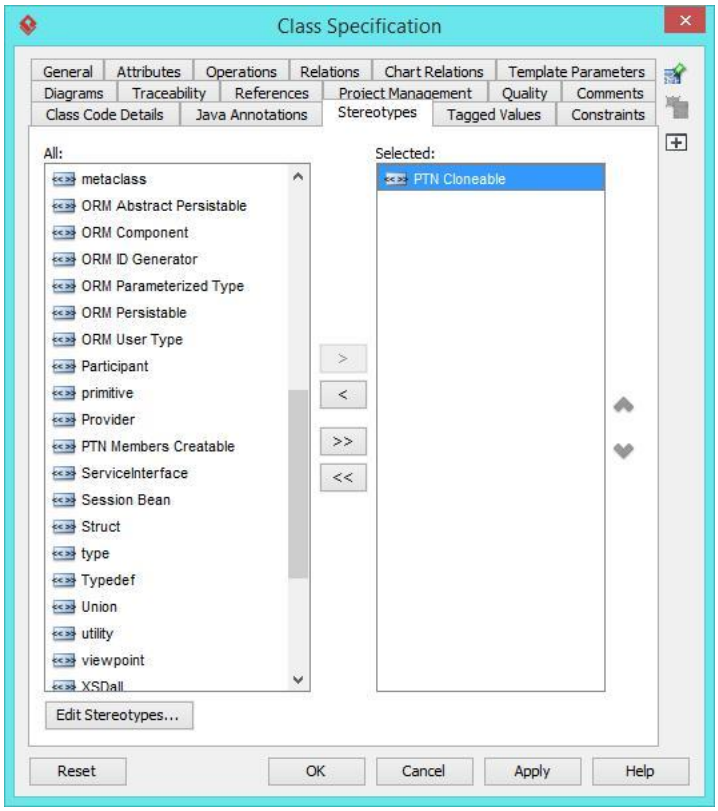
18. Right-click on the *ConcreteObserver* class, and select **Add** > **Attribute** from the popup menu. Name the attribute *observerstate*. Up to now, the diagram should look like:



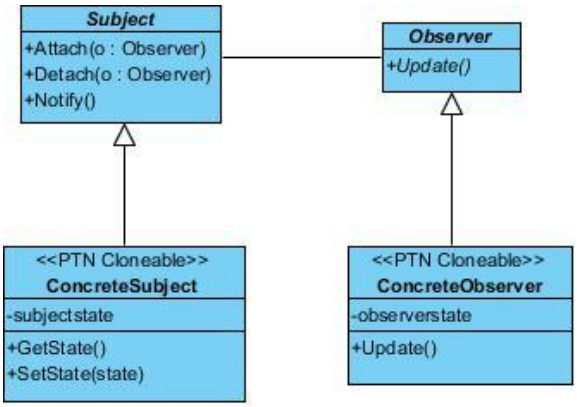
19. In practice, there may be multiple concrete subjects and observers. To represent this, stereotype the class *ConcreteSubject* and *ConcreteObserver* as **PTN Cloneable**. Right-click on *ConcreteSubject* and select **Stereotypes** > **Stereotypes...** from the popup menu.



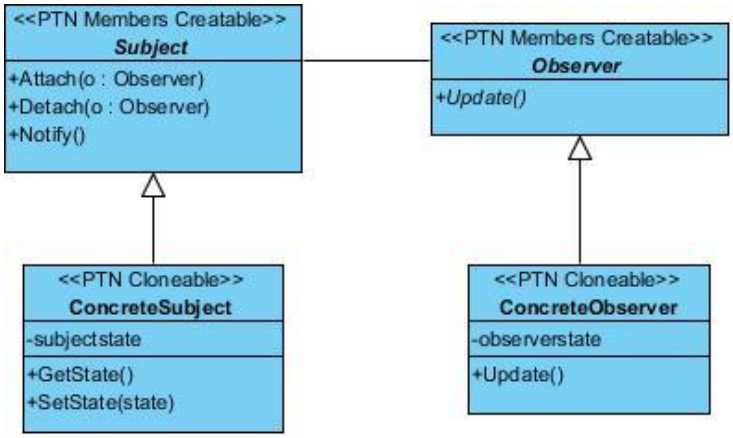
20. In the **Stereotypes** tab of the **Class Specification** dialog box, select **PTN Cloneable** and click > to assign it to *ConcreteSubject* class. Click **OK** to confirm.



21. Repeat steps 18 and 19 on *ConcreteObserver*.

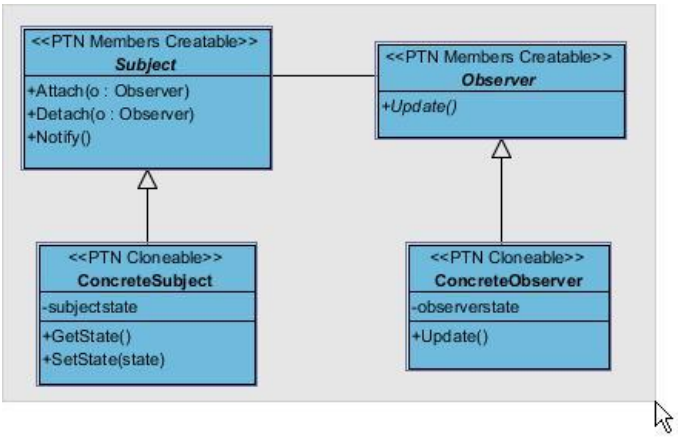


22. In practice, there are domain specific operations in subject and observer. To represent this, stereotype the class *Subject* and *Observer* as **PTN Members Creatable**. Repeat steps 18 and 19 to stereotype *Subject* and *Observer* as **PTN Members Creatable**.

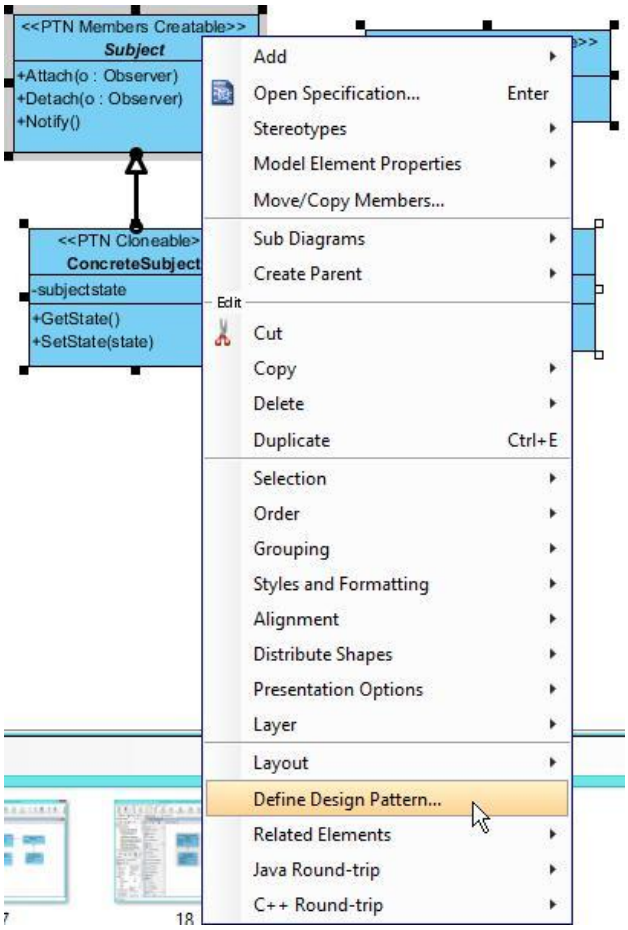


Defining Pattern

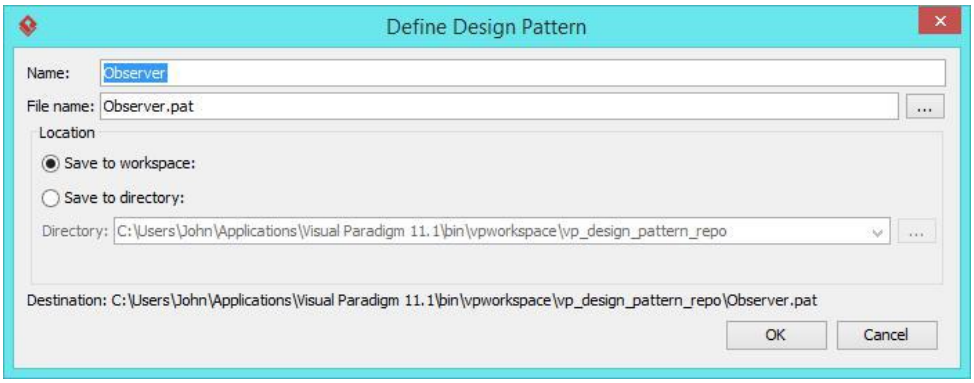
1. Select all classes on the class diagram.



2. Right-click on the selection and select **Define Design Pattern...** from the popup menu.



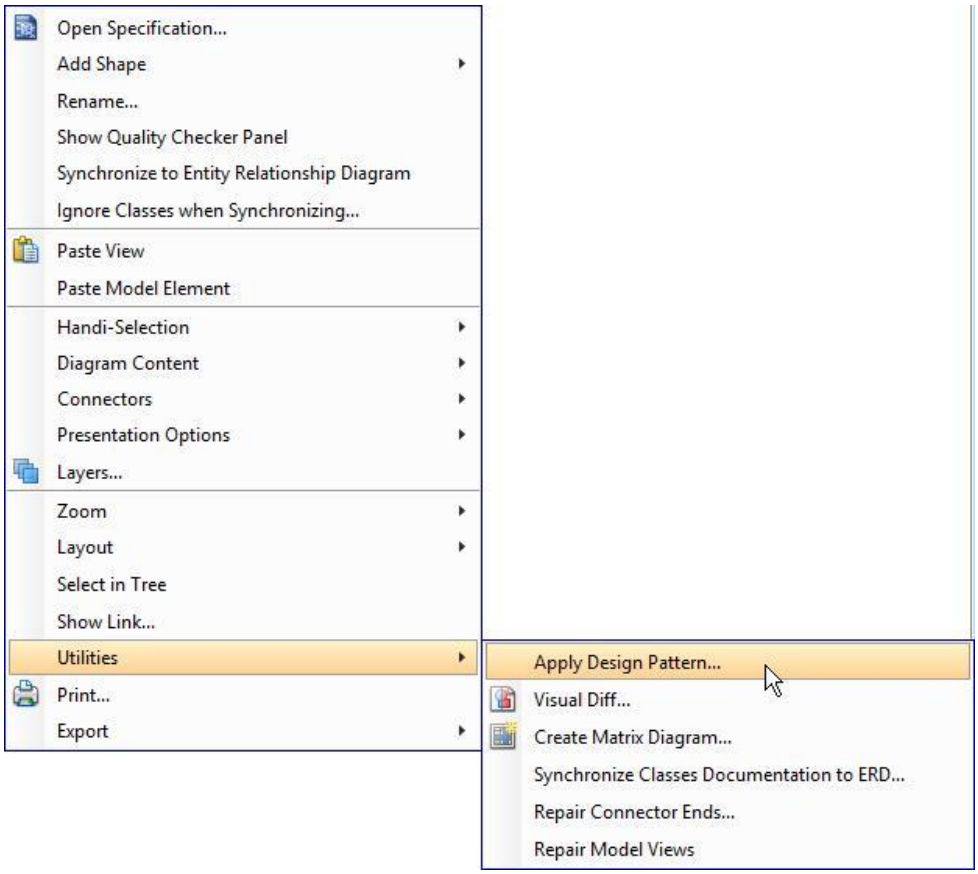
13. In the **Define Design Pattern** dialog box, specify the pattern name *Observer*. Keep the file name as is. Click **OK** to proceed.



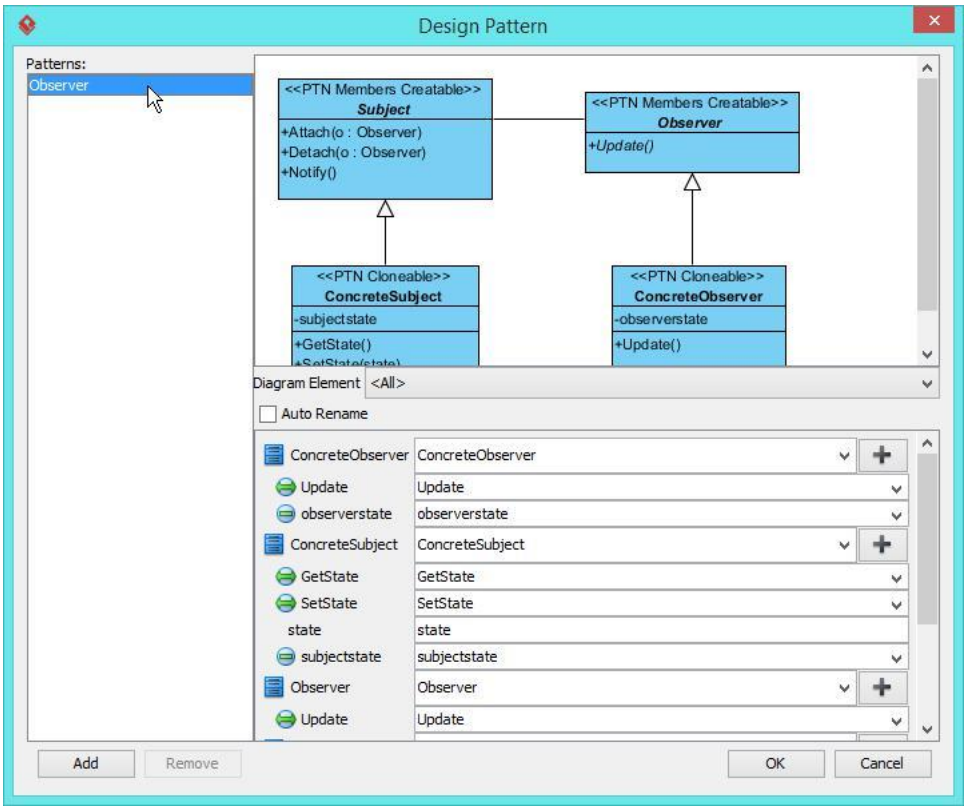
Applying Design Pattern on Class Diagram

In this section, we are going to apply the observer pattern to model a diagram editor for observing changes of model, and calling various panes like the property and overview panes to update their content.

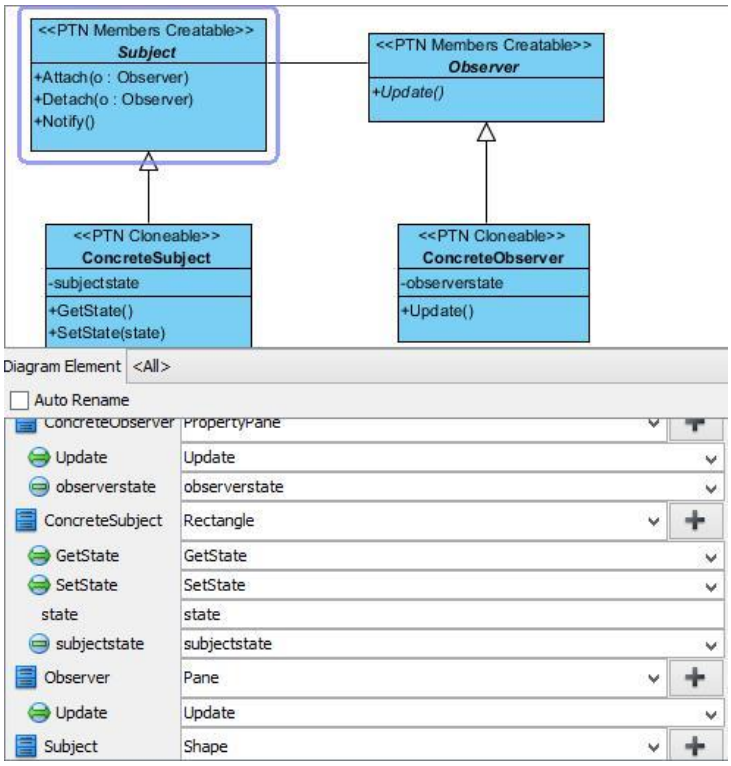
1. Create a new project *Diagram Editor*.
2. Create a class diagram *Domain Model*.
3. Right-click on the class diagram and select **Utilities** > **Apply Design Pattern...** from the popup menu.



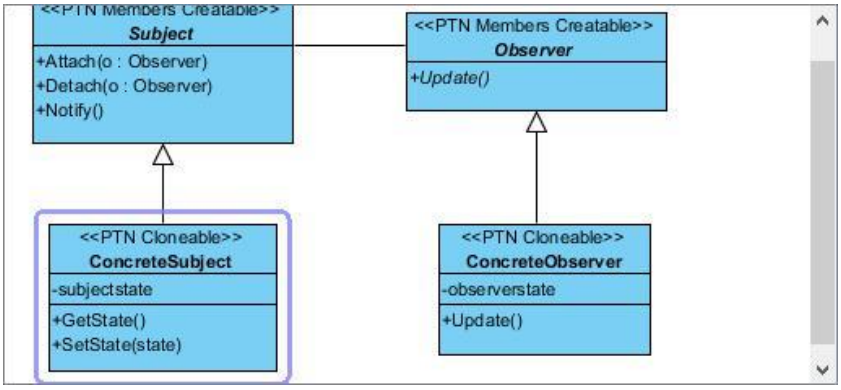
4. In the **Design Pattern** dialog box, select *Observer* from the list of patterns.



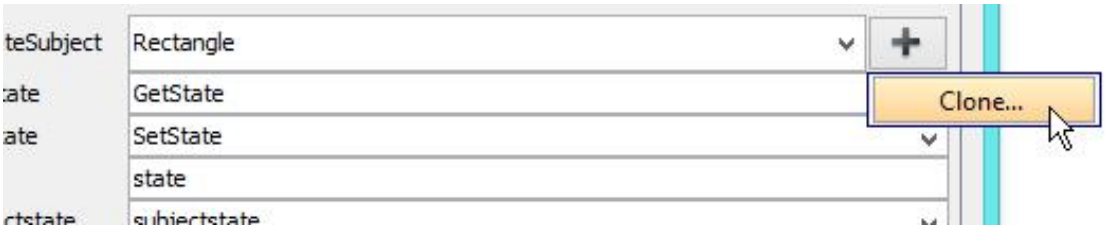
5. At the bottom pane, rename classes *Subject*, *Observer*, *ConcreteSubject* and *ConcreteObserver* to *Shape*, *Pane*, *Rectangle* and *PropertyPane* respectively.



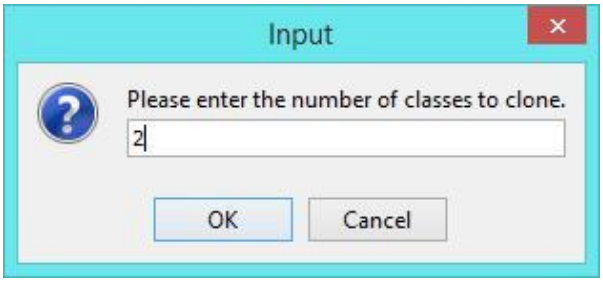
6. Besides rectangle, there are more types of shape like circle and triangle.
   Select *ConcreteSubject* from the overview pane.



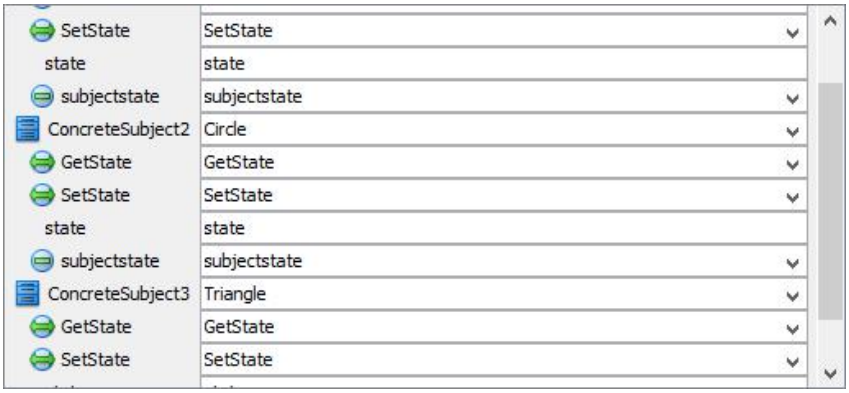
7. Click on the + button next to the class name and select **Clone...** from the popup menu.



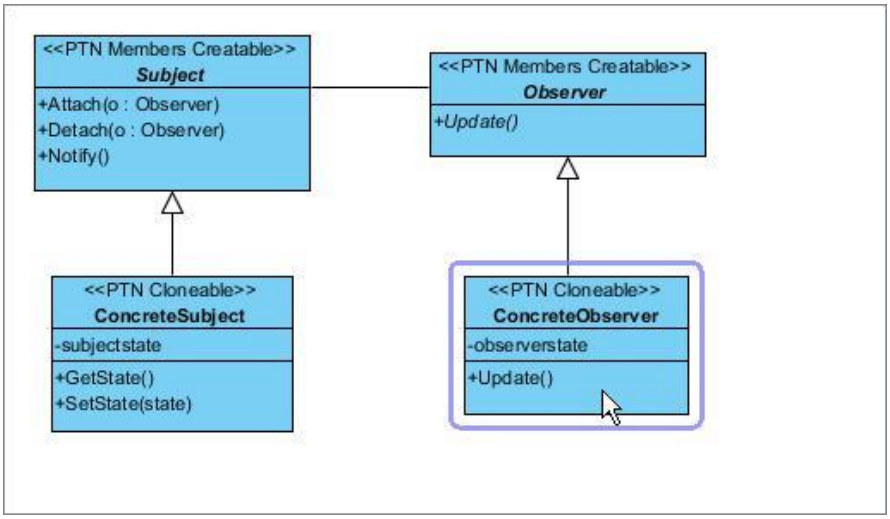
8. Enter 2, which is the number of classes to clone, and click **OK** to confirm.



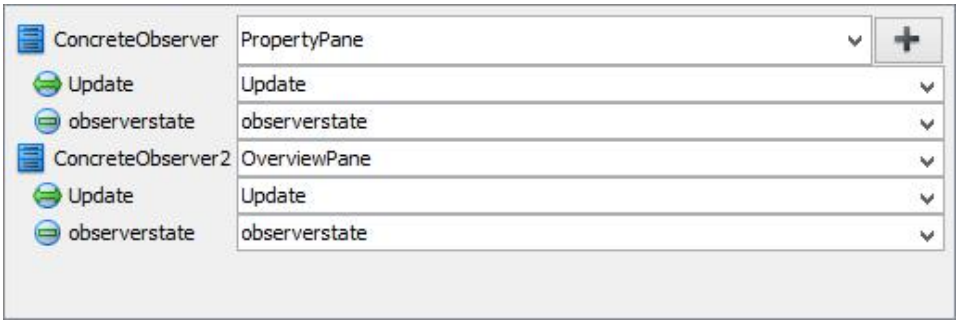
9. At the bottom pane, rename *ConcreteSubject2* and *ConcreteSubject3* to *Circle* and *Triangle*.



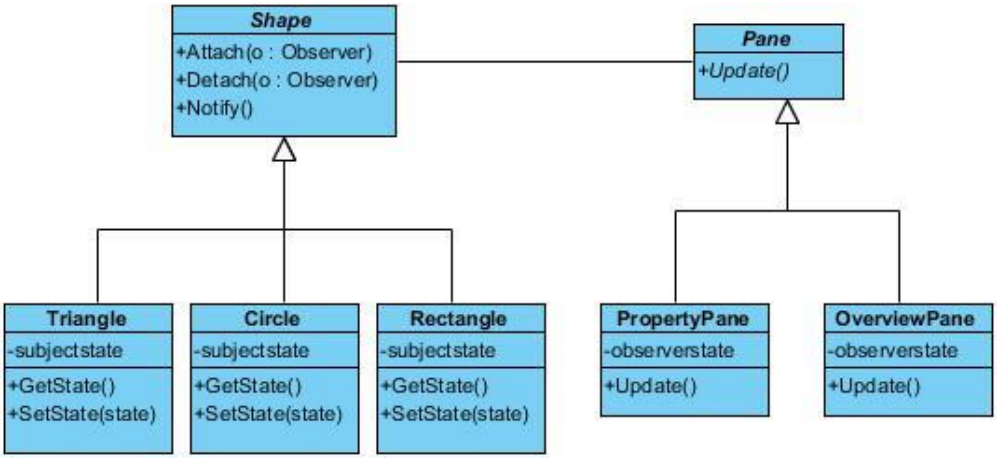
10. For observers, there are also panes like overview pane. Select *ConcreteObserver* from the overview pane.



11. Click on the + button next to the class name and select **Clone...** from the popup menu.
12. Enter 1, which is the number of classes to clone, and click **OK** to confirm.
13. At the bottom pane, rename *ConcreteObserver2* to *OverviewPane*.



14. Click **OK** to confirm. Here is the diagram formed:

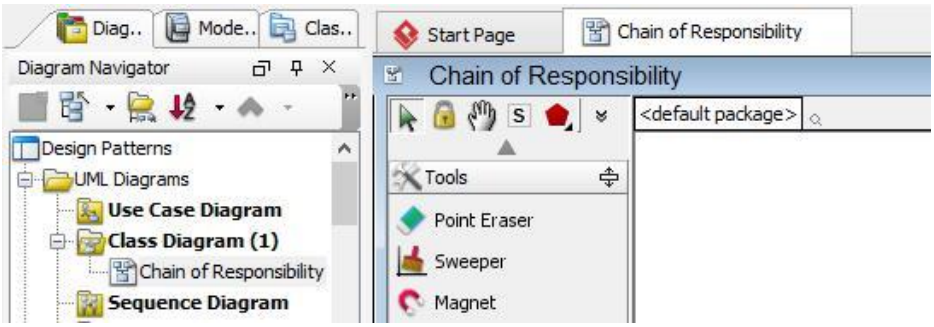


LAB-15

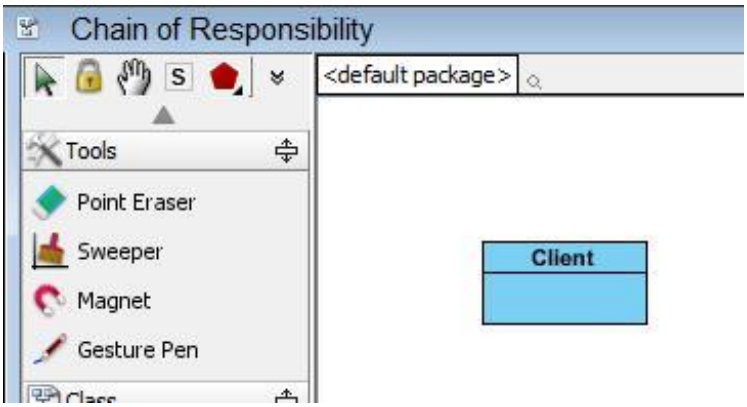
Modeling Design Pattern with Class Diagram

1. Create a new project *Design Patterns*.

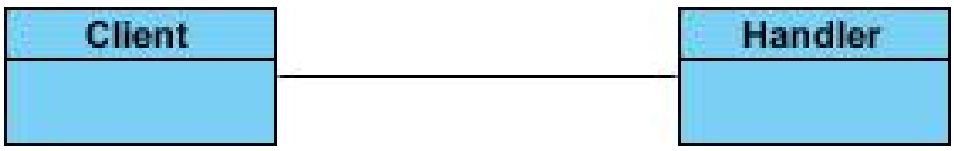
2. Create a class diagram *Chain of Responsibility*.



3. Select **Class** from diagram toolbar. Click on the diagram to create a class. Name it as *Client*.

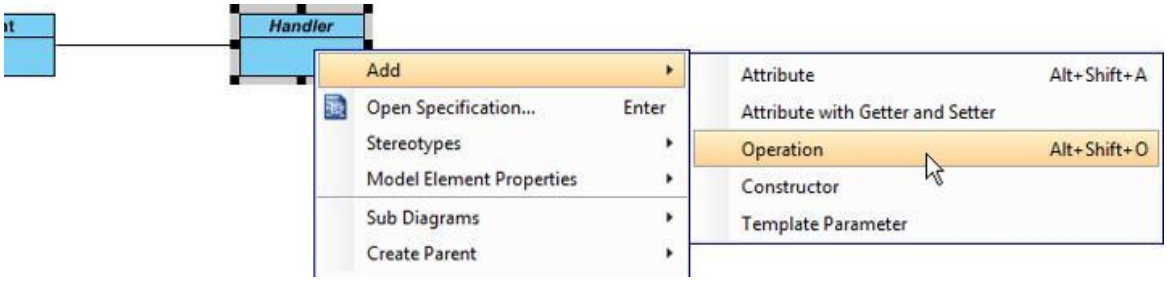


4. Move the mouse cursor over the *Client* class, and drag out **Association** > **Class** to create an associated class *Handler*.



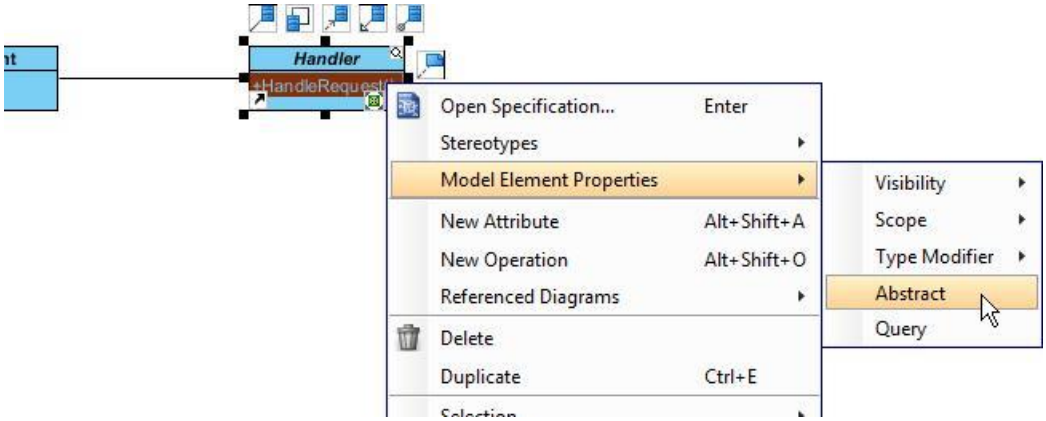
6. Right-click on *Handler*, and select **Model Element Properties** > **Abstract** to set it as abstract.

7. Right-click on *Handler* class, and select **Add** > **Operation** from the popup menu.

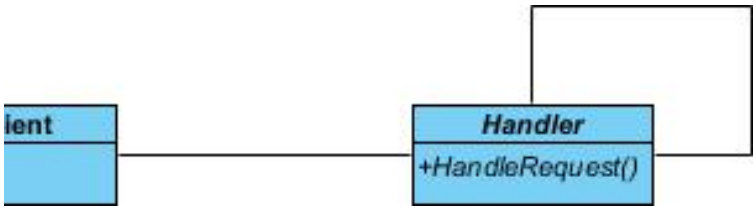


8. Name the operation *HandleRequest().*

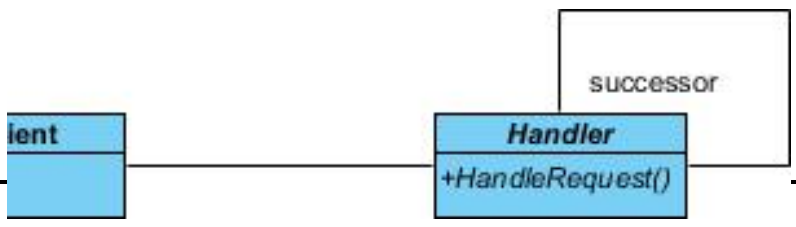
9. Right-click on *HandleRequest*, and select **Model Element Properties** > **Abstract** to set it as abstract.



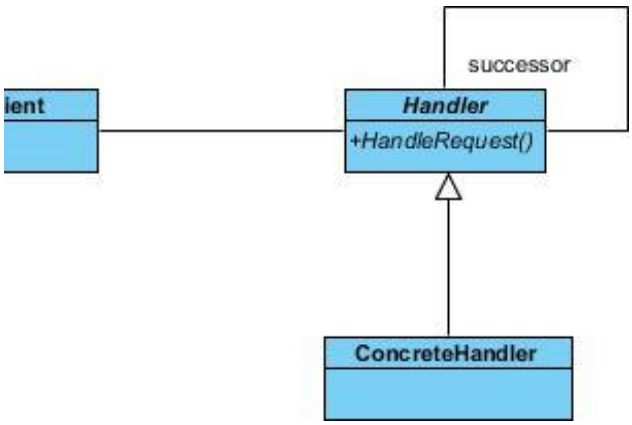
10. Move the mouse cursor over the *Handler* class, and click on the resource icon **Self Association** to create a self-association.



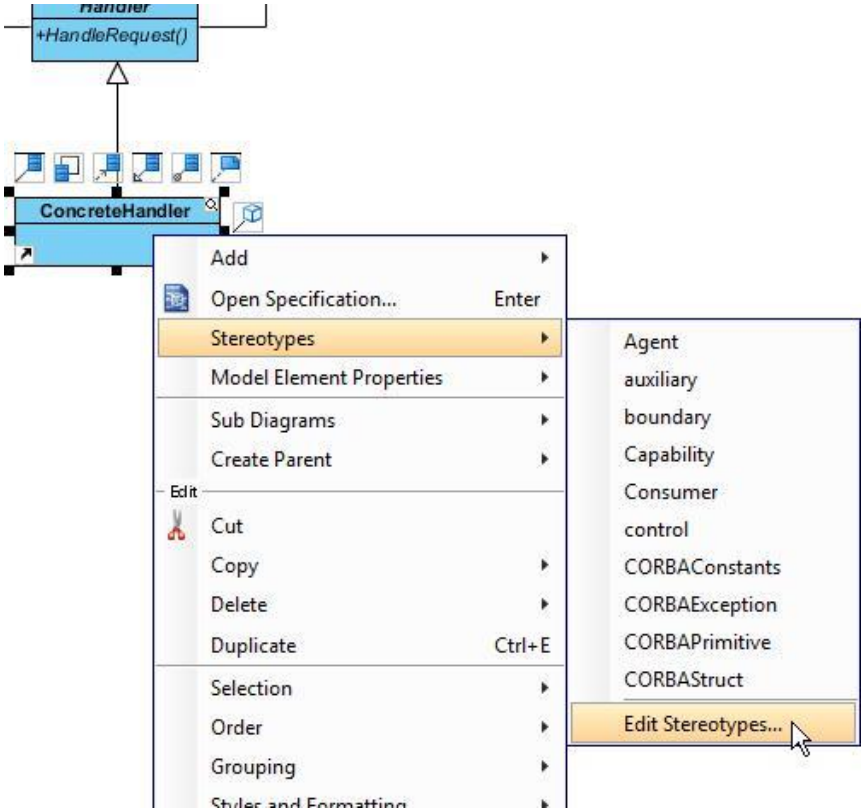
11. Name the association end successor.



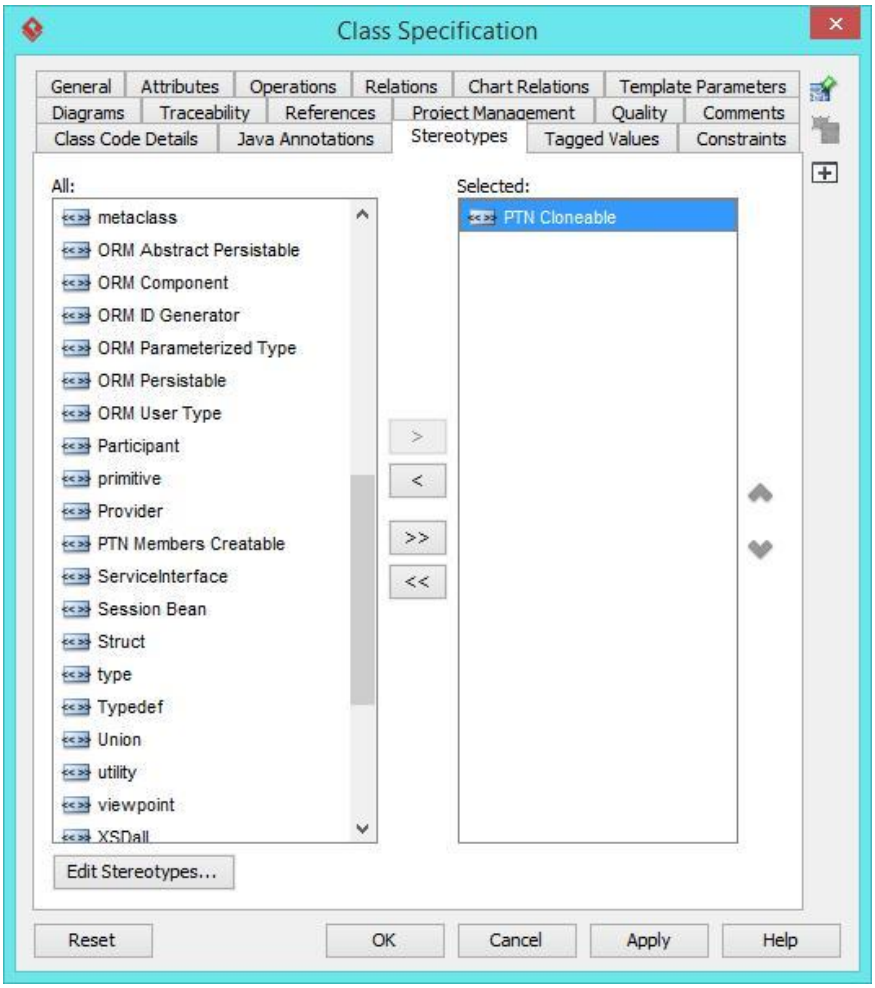
12. Move the mouse cursor over the *Handler* class, and drag out **Generalization** > **Class** to create subclasses *ConcreteHandler*.



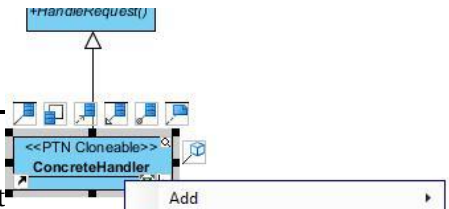
13. In practice, there may be multiple concrete handlers. To represent this, stereotype the class *ConcreteHandler* as **PTN Cloneable**. Right-click on *ConcreteHandler* and select **Stereotypes** > **Stereotype...** from the popup menu.



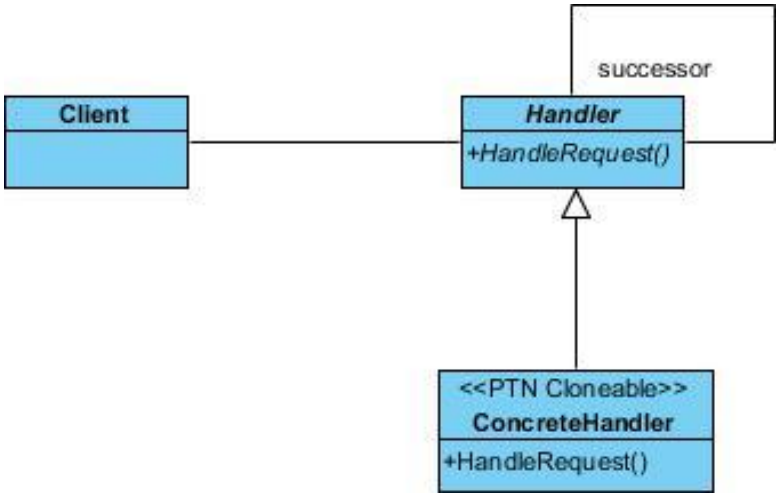
14. In the **Stereotypes** tab of the **Class Specification** dialog box, select **PTN Cloneable** and click > to assign it to *ConcreteHandler* class. Click **OK** to confirm.



15. We need make the concrete handlers inherit operations from the handle class. Right-click on *ConcreteHandler* and select **Related Elements** > **Realize all Interfaces** from the popup menu.

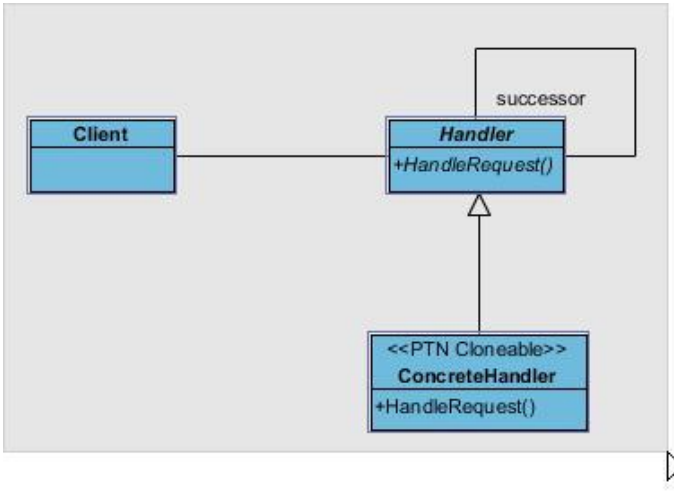


Up to now, the diagram should look like this:

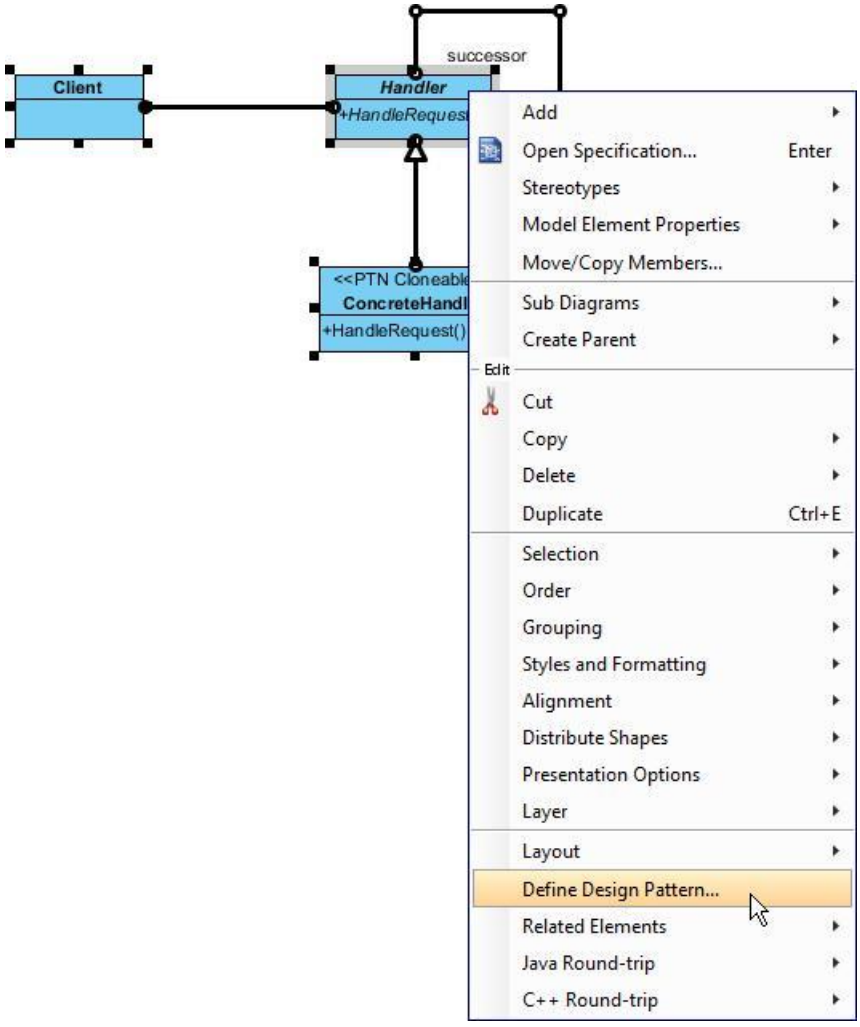


Defining Pattern

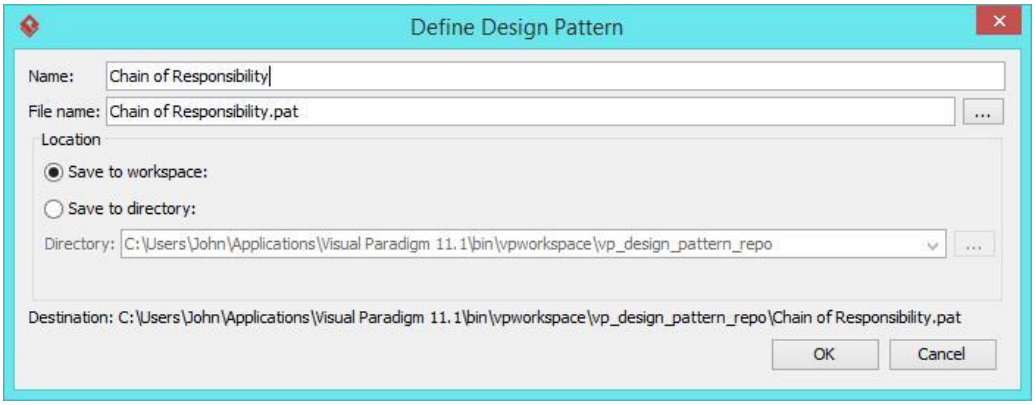
1. Select all classes on the class diagram.



2. Right-click on the selection and select **Define Design Pattern...** from the popup menu.



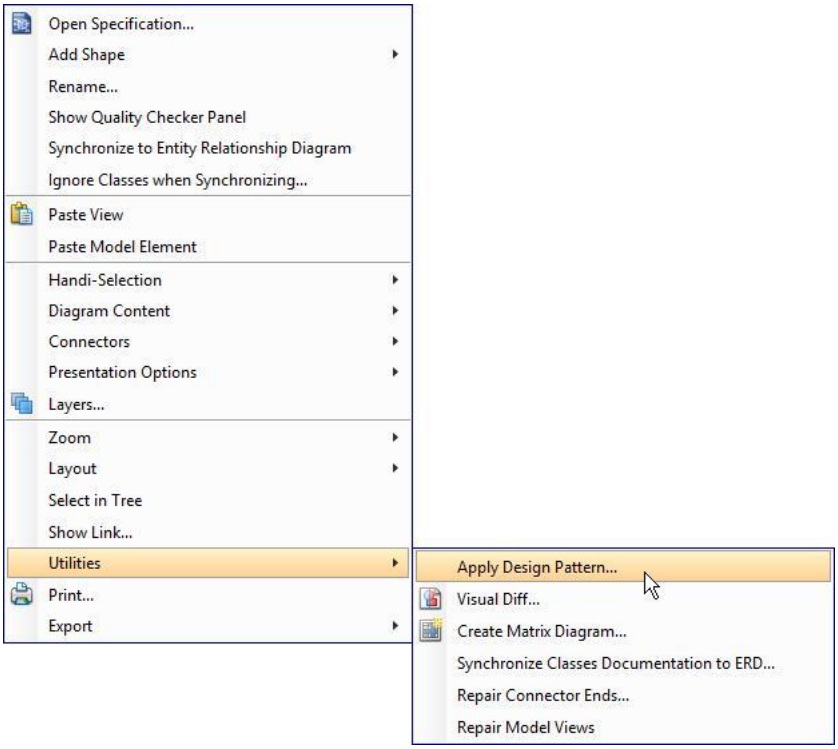
3. In the **Define Design Pattern** dialog box, specify the pattern name *Chain of Responsibility*. Keep the file name as is. Click **OK** to proceed.



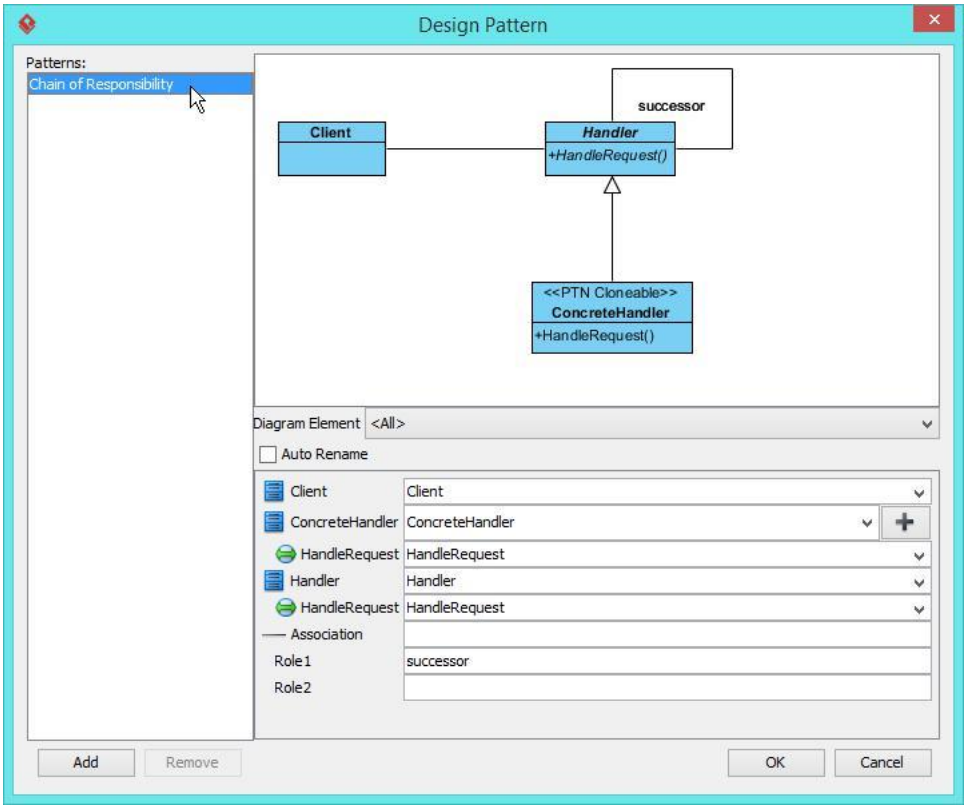
Applying Design Pattern on Class Diagram

In this section, we are going to apply the chain of responsibility pattern in modeling a coin dispenser.

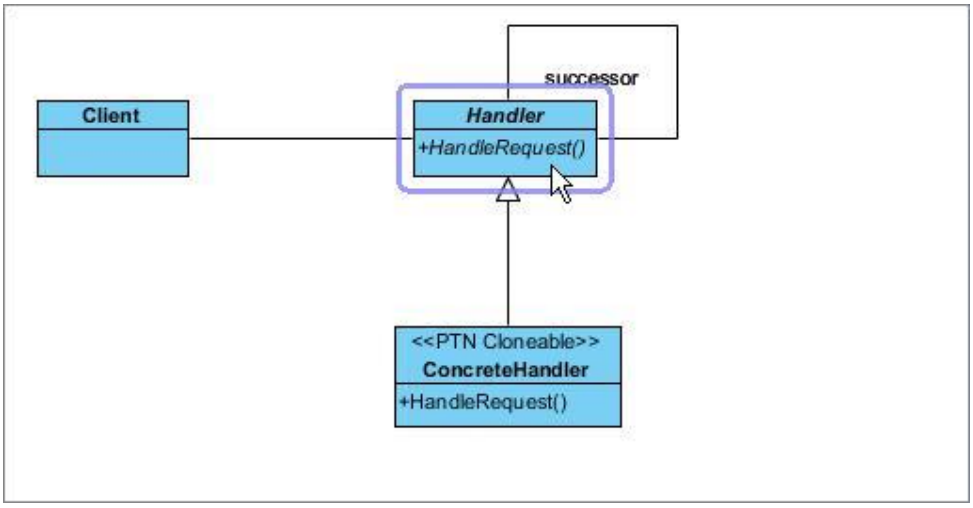
1. Create a new project *Coin Dispenser*.
2. Create a class diagram *Domain Model*.
3. Right-click on the class diagram and select **Utilities** > **Apply Design Pattern...** from the popup menu.



4. In the **Design Pattern** dialog box, select *Chain of Responsibility* from the list of patterns.

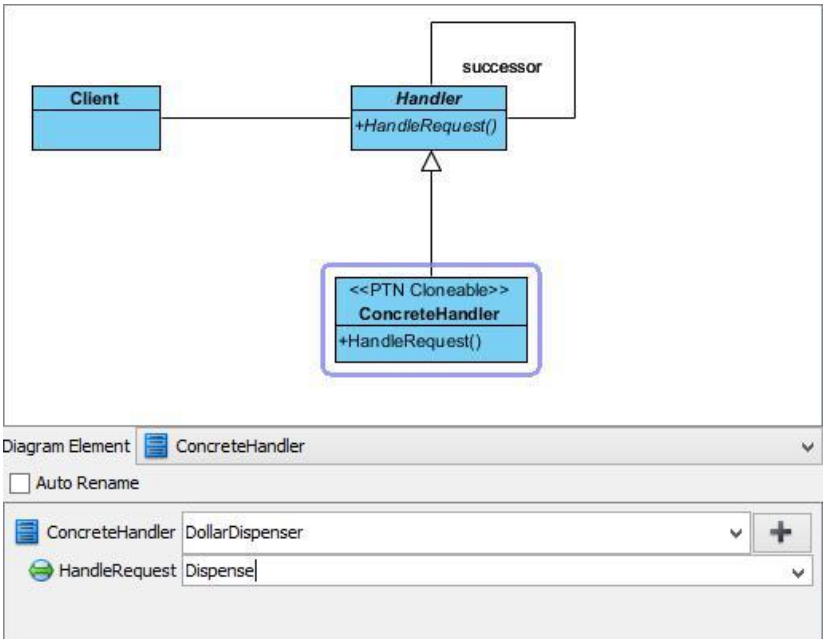


5.Click on *Handler* in the overview.



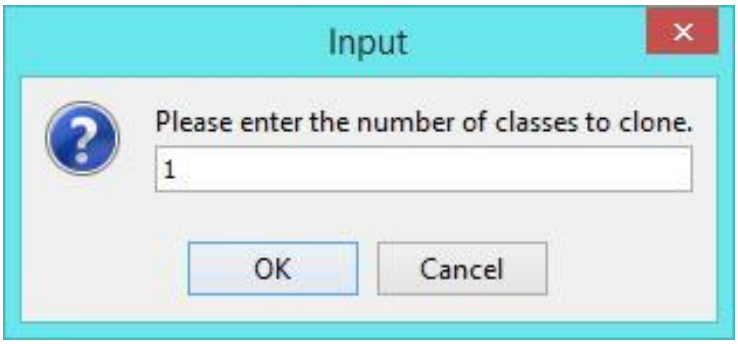
6. Rename *Handler* to *CoinDispenser*, and operation *HandleRequest* to *Dispense* at the bottom pane.

7. Click on *ConcreteHandler* in overview, and rename it to *DollarDispenser*, and operation *HandleRequest* to *Dispense*.



8. We need one more concrete handler for dispensing cents. Keep *ConcreteHandler* selected, click on + and select **Clone...** from the popup menu.

9. Enter *1* to be the number of classes to clone. Click **OK** to confirm.



10. Rename *ConcreteHandler2* to *CentDispenser*, and operation *HandleRequest* to *Dispense*.

11. Click **OK** to apply the pattern to diagram.
12. Tidy up the diagram. Here is the result: