



Java GUI Database and UML

Hands on Lab



September2011



Information in this document, including URL and other Internet Web site references, is subject to change without notice. This document supports a preliminary release of software that may be changed substantially prior to final commercial release, and is the proprietary information of Binus University.

This document is for informational purposes only. BINUS UNIVERSITY MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

The entire risk of the use or the results from the use of this document remains with the user. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Binus University.

Binus University may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Binus University, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2011 Binus University. All rights reserved.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Table of Content

OVERVIEW	iii
SYSTEM REQUIREMENT.....	iv
GUI Container.....	1
GUI Layout Manager.....	10
GUI Basic Component.....	22
GUI Intermediate Component.....	37
GUI Advanced Component.....	50
Event Driven Programming.....	63
Database Access	88
Database Operation.....	107
Java Applet	120
UML Tools	128
UML Design	146



OVERVIEW

Chapter 01

- GUI Container

Chapter 02

- GUI Layout Manager

Chapter 03

- GUI Basic Component

Chapter 04

- GUI Intermediate Component

Chapter 05

- GUI Advanced Component

Chapter 06

- Event Driven Programming

Chapter 07

- Database Access

Chapter 08

- Database Operation

Chapter 09

- Java Applet

Chapter 10

- UML Tools

Chapter 11

- UML Design



SYSTEM REQUIREMENT

- **Hardware:**

- Minimum:

- 1.6 GHz CPU, 192 MB RAM, 1024x768 display, 5400 RPM hard disk

- Recommended:

- 2.2 GHz, 384 MB, 1280x1024 display, 7200 RPM or higher.

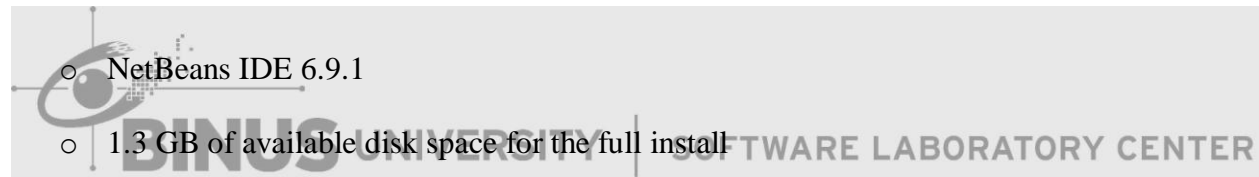
- On Windows Vista:

- 2.4 GHz CPU, 768 MB RAM

- **Software:**

- NetBeans IDE 6.9.1

- 1.3 GB of available disk space for the full install



Chapter 01

GUI Container



1.1. Graphical User Interface (GUI) Classes

GUI classes are classified into 3 groups: Component Classes, Container Classes, and Helper Classes.

1.2. Component Classes

Component Classes are an object that can be displayed on the screen, such as JButton, JTextField, JComboBox, etc.

1.3. Container Classes

Container Classes are a generic Abstract Window Toolkit (AWT) container object that can contain component classes. Components added to a container are tracked in a list. The order of the list will define the components' front-to-back stacking order within the container. If no index is specified when adding a component to a container, it will be added to the end of the list (and hence to the bottom of the stacking order).

Example: Create object Container class and set Container Layout to Border Layout.

(Note: We will discuss about Layout Manager in the next chapter)

```
Container c = frame.getContentPane();  
c.setLayout(new BorderLayout());
```

WARE LABORATORY CENTER

1.4. Helper Classes

Helper classes are used to describe the properties of GUI components, such as graphics context, colors, fonts, and dimension.

1.5. JFrame

JFrame class is an example of Container Class. It can contain other component classes. Default Layout JFrame is Border Layout. The frame is not displayed until **setVisible(true)** method is invoked.

Example: Create object JFrame

```
JFrame frame = new JFrame();
```

1.6. JInternalFrame

A lightweight object that provides many of the features of a native frame, including dragging, closing, becoming an icon, resizing, title display, and support for a menu bar. The JInternalFrame content pane is where you add child components. Generally, you add JInternalFrames to a JDesktopPane. The UI delegates the look-and-feel-specific actions to the DesktopManager object maintained by the JDesktopPane.

Example: Create object JInternalFrame

```
JInternalFrame intframe = new JInternalFrame();
```

1.7. JPanel

JPanel is a generic lightweight container. Default Layout JPanel is Flow Layout.


Example: Create object JPanel

```
JPanel panel = new JPanel();
```

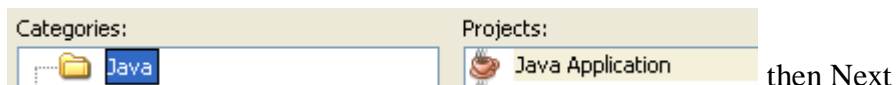
1.8. Exercise

a. Task 01 - Create Project JAVA in Netbeans

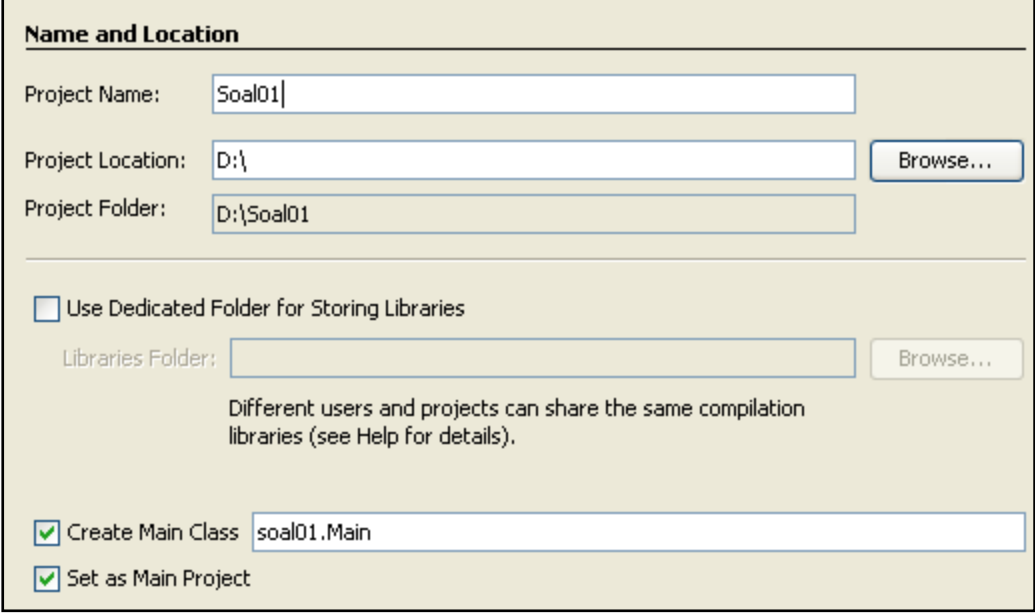
1. Run Netbeans from Start Menu

2. Open Menu File -> New Project or Click icon  on bottom menu Edit

3. On Project Window, choose Java on Categories and Java Application on Projects



4. Setting Project name, Project Location and don't forget to check "Create Main Class" like picture below



Name and Location

Project Name: Soal01

Project Location: D:\ Browse...

Project Folder: D:\Soal01

Use Dedicated Folder for Storing Libraries

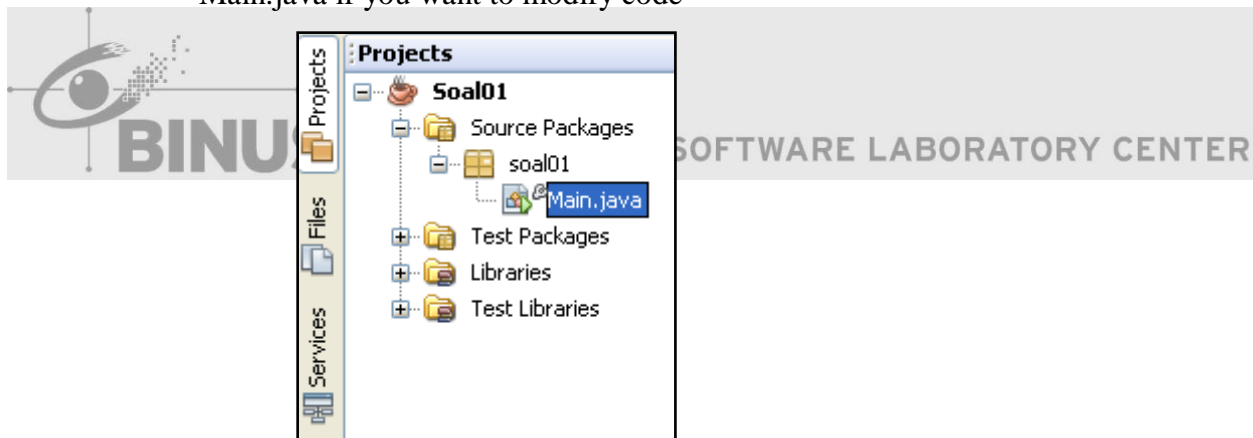
Libraries Folder: Browse...

Different users and projects can share the same compilation libraries (see Help for details).

Create Main Class soal01.Main

Set as Main Project

5. Then Press Finish
6. Finally, your project has been created like picture below, double click on file Main.java if you want to modify code



b. Task 02 – Introduction IDE Netbeans

1. Run program press **F6**
2. Netbeans can do autocomplete to help Programmer in order to import library and complete the syntax easily like example below:

Type “**JF**” and press **CTRL + Space** on keyboard, and the choices will be shown below.

```
public class Main {
    JF
    JFileChooser (javax.swing)
    JFormattedTextField (javax.swing)
    JFrame (javax.swing)
    JarFile (java.util.jar)
    JarFilter (sun.misc)
    JavaFileManager (javax.tools)
    JavaFileObject (javax.tools)
}
```

If you press enter on JFrame, the result will be shown below

```
import javax.swing.JFrame;

public class Main {
    JFrame
    public static void main(String[] args) {
    }
}
```

As you see, if you use autocomplete, library will be imported automatically.

c. Task 03 - Create JFrame and Set Properties JFrame

1. Create object JFrame, and give the name of object is "frame"

```
JFrame frame = new JFrame();
```

Explanation:

After you create object JFrame named frame, furthermore you will use the object to do something.

2. General Setting properties JFrame

```
//Give title JFrame
frame.setTitle("GUI Container");

//Setting size (Width, Height) of JFrame
frame.setSize(500, 500);

//Make location JFrame will be located center window on screen
frame.setLocationRelativeTo(null);

//Make JFrame exit when user close it
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

//Show JFrame
frame.setVisible(true);
```

Explanation:

- **setTitle([String](#) title)** is used to give title on JFrame. This method inherited from class **java.awt.[Frame](#)**.
- **setSize([int](#) width, [int](#) height)** is used to set your size of JFrame. This method inherited from class **java.awt.[Window](#)**.
- **setLocationRelativeTo([Component](#) c)** is method inherited from class **java.awt.[Window](#)**. Fill parameter with 'null' to located JFrame to center of window on the screen.
- **setDefaultCloseOperation([int](#) operation)** is method to set the operation that will happen by default when the user initiates a "close" on this frame. Fill parameter with some condition such as :
 - **[JFrame.EXIT ON CLOSE](#)**
The exit application default window close operation. If a window has this set as the close operation and is closed in an applet, a `SecurityException` may be thrown. It is recommended you only use this in an application.
 - **[JFrame.DISPOSE ON CLOSE](#)**
The dispose-window default window close operation.
 - **[JFrame.DO NOTHING ON CLOSE](#)**
The do-nothing default window close operation.
 - **[JFrame.HIDE ON CLOSE](#)**
The hide-window default window close operation.

d. Task 04 – Using Container Class

1. Create object Container and copy content from frame into object Container

```
Container c = frame.getContentPane();
```

Explanation:

Method **getContentPane()** is used to returns the `contentPane` object for this frame.

2. Set layout of Container with Border Layout

```
c.setLayout(new BorderLayout());
```

Explanation:

Set the layout manager for this container.

The conclusion is all content of JFrame will be copied into Container so if you use JFrame and Container at the same time you will see no different between Container and JFrame.

- e. Task 05 - Create JInternalFrame

Generally, you add JInternalFrames to a JDesktopPane. The UI delegates the look-and-feel-specific actions to the DesktopManager object maintained by the JDesktopPane.

1. Create object JDesktopPane and give the name of object is “dp”

```
JDesktopPane dp = new JDesktopPane();
```

Explanation:

After you make object JDesktopPane, furthermore you will use the object to do something.

2. Create object JInternalFrame and give the name of object is “intframe”

```
JInternalFrame intframe = new JInternalFrame();
```

3. Add object JInternalFrame into JDesktopPane

```
dp.add(intframe);
```

Explanation:

Method **add(Component comp)** is used to add other component from source into destination component. Which is in example, object from JinternalFrame is added into object from JDesktopPane. This method inherited from class **java.awt.Container**.

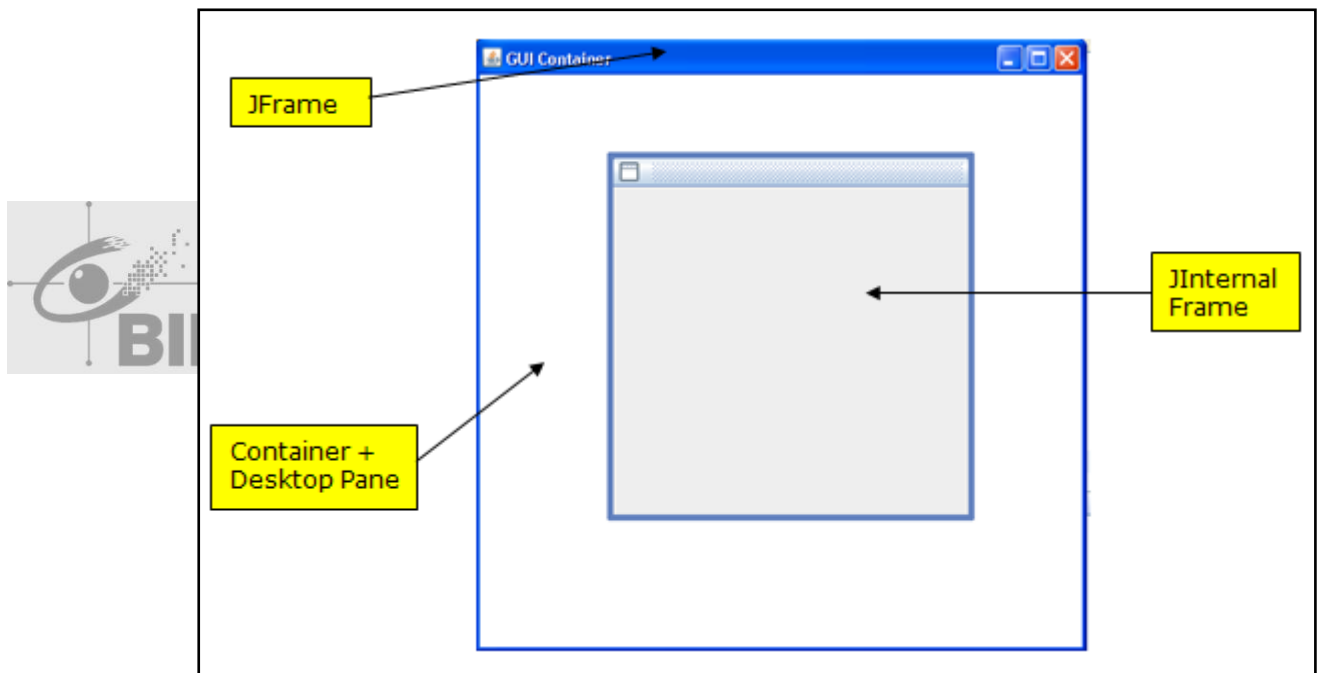
4. Add object `JDesktopPane` into Container

```
c.add(dp);
```

Explanation:

Method `add(Component comp)` is used to add other component from source into destination component. Which is in example, object from `JDesktopPane` is added into object from class `Container`. This method inherited from class `java.awt.Container`.

If you run (F6) program, you will be shown below



- f. Task 06 - Create JPanel

1. Create object JPanel

```
JPanel panel_1 = new JPanel();  
JPanel panel_2 = new JPanel();
```

Explanation:

After you create object `JPanel`, furthermore you will use the object to do something.

2. Set Background JPanel will give Color Background JPanel

```
panel_1.setBackground(Color.YELLOW);  
panel_2.setBackground(Color.GREEN);
```

Explanation:

Method **setBackground**([Color](#) bg) is used to set the background color of this component. The background color is used only if the component is opaque, and only by subclasses of [JComponent](#) or [ComponentUI](#) implementations. Direct subclasses of [JComponent](#) must override [paintComponent](#) to honor this property.

This method inherited from **class** [javax.swing.JComponent](#).

It is up to the look and feel to honor this property; some may choose to ignore it.

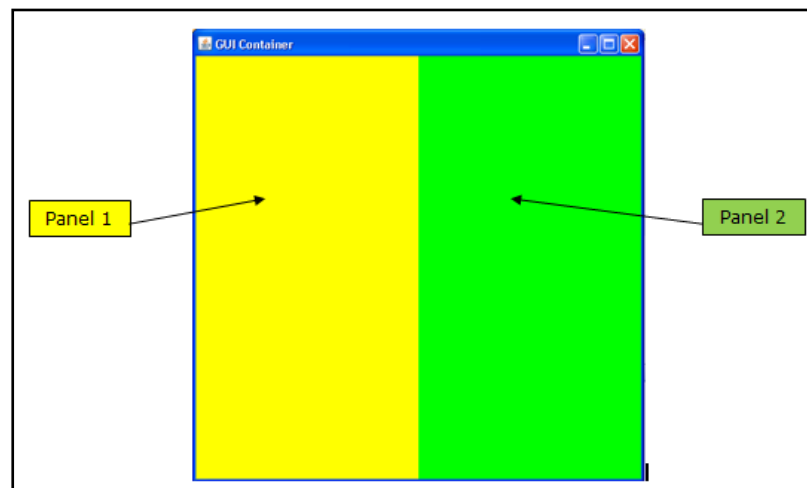
3. Add JPanel to Container

```
c.add(panel_1);  
c.add(panel_2);
```

Explanation:

Method **add**([Component](#) comp) is used to add other component from source into destination component. Which is in example, object from [JPanel](#) is added into object from class [Container](#). This method inherited from **class** [java.awt.Container](#).

If you run (F6) program, the result will be shown below



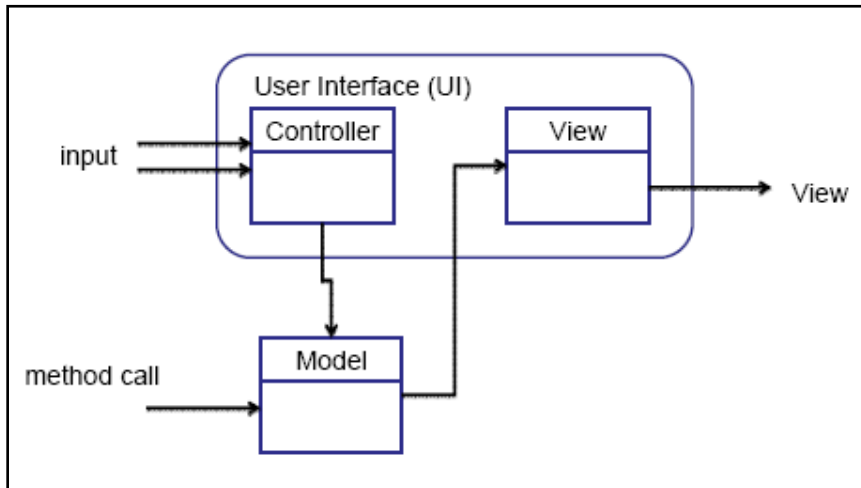
Chapter 02

GUI Layout Manager



2.1 Layout Manager

Swing provides many standard components to make GUI like button, textfield, etc. This component is built by Model View Controller (MVC) Concept.



Swing provides Container that can accept component. To arrange size and position of component, can be used Layout Manager. If Layout Manager is null then position and size of component will be given by programmer. Generally layout that always used areFlowLayout, BorderLayout, and GridLayout.

2.2 Border Layout

Arrange component in fixed position based on cardinal directions : NORTH, EAST, SOUTH, WEST and CENTER.

Generally, there are 2 constructors that always be used :

<code>BorderLayout ()</code>	Constructs a new border layout with no gaps between components.
<code>BorderLayout (int hgap, int vgap)</code>	Constructs a border layout with the specified gaps between components.

Parameter

int hgap : Horizontal gap between components

int vgap : Vertical gap between components

Example Border Layout



Source Code:

```
public class Main {  
  
    JFrame frame = new JFrame();  
  
    JButton btn_north = new JButton("North");  
    JButton btn_west = new JButton("West");  
    JButton btn_center = new JButton("Center");  
    JButton btn_east = new JButton("East");  
    JButton btn_south = new JButton("South");  
  
    public Main() {  
        Container c = frame.getContentPane();  
        c.setLayout(new BorderLayout());  
  
        c.add(btn_north, "North");  
        c.add(btn_west, "West");  
        c.add(btn_center, "Center");  
        c.add(btn_east, "East");  
        c.add(btn_south, "South");  
  
        frame.setTitle("GUI Layout Manager");  
        frame.pack();  
        frame.setLocationRelativeTo(null);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setVisible(true);  
    }  
  
    public static void main(String[] args) {  
        new Main();  
    }  
}
```

2.3 Grid Layout

Arrange component in grid position (Matrix)

Generally, there are 3 constructors that always be used:

<code>GridLayout()</code> Creates a grid layout with a default of one column per component, in a single row.
<code>GridLayout(int rows, int cols)</code> Creates a grid layout with the specified number of rows and columns.
<code>GridLayout(int rows, int cols, int hgap, int vgap)</code> Creates a grid layout with the specified number of rows and columns.

Parameter

int rows: the number of rows

int cols: the number of columns

int hgap: Gap horizontal between components

int vgap: Gap vertical between components

Example Grid Layout



Source Code:

```
public class Main {

    JFrame frame = new JFrame();

    JButton[] btn = new JButton[9];

    public Main() {
        Container c = frame.getContentPane();
        c.setLayout(new GridLayout(4,2));

        for(int i=1; i<=8; i++)
        {
            btn[i] = new JButton(""+i);
            c.add(btn[i]);
        }

        frame.setTitle("GUI Layout Manager");
        frame.pack();
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        new Main();
    }
}
```

2.4 Flow Layout

Arrange component position from left to right and put next component on new row if the width of panel is not enough.

Generally, there are 2 constructors that always be used:

[FlowLayout\(\)](#)

Constructs a new `FlowLayout` with a centered alignment and a default 5-unit horizontal and vertical gap.

[FlowLayout\(int align, int hgap, int vgap\)](#)

Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.

Parameter

int align : Fill with `FlowLayout.CENTER`, `FlowLayout.RIGHT`, `FlowLayout.LEFT`, `FlowLayout.LEADING`, `FlowLayout.TRAILING`

int hgap : Gap horizontal between components

int vgap : Gap vertical between components

Example Flow Layout



Source Code:

```
public class Main {

    JFrame frame = new JFrame();

    JButton[] btn = new JButton[6];

    public Main() {
        Container c = frame.getContentPane();
        c.setLayout(new FlowLayout());

        for(int i=1; i<=5; i++)
        {
            btn[i] = new JButton(""+i);
            c.add(btn[i]);
        }

        frame.setTitle("GUI Layout Manager");
        frame.pack();
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        new Main();
    }
}
```

2.5 Absolute Layout

AbsoluteLayout is a LayoutManager that works as a replacement for "null" layout to allow placement of components in absolute positions.

2.6 How to add Layout to Component

This section will guide you how to give layout step by step.

1. Analyze parameter function that filled by ([LayoutManager](#))

Example:

- a. When you create object JPanel, there are 4 constructors that you can use. One of them is filled by ([LayoutManager](#) layout). In this parameter, we can give layout to JPanel directly when we have created it.
- b. The Other way is we can use method **setLayout**([LayoutManager](#) manager) to give layout. Some components that have this method are JFrame, JPanel, JInternalFrame, etc.

2. Fill parameter with layout that you will use

Example:

- a. Create JPanel with GridLayout 3 Rows and 4 Cols.

First, use this constructor of JPanel

```
JPanel(LayoutManager layout)
Create a new buffered JPanel with the specified layout manager
```

Second, Fill parameter with GridLayout, there are several constructors of GridLayout, but generally this constructor is chosen.

```
JPanel panel = new JPanel(new GridLayout(rows, cols));
```

Third, Fill parameter with rows and cols

```
JPanel panel = new JPanel(new GridLayout(3, 4));
```

- b. Create object JInternalFrame and then change layout with BorderLayout.

First, create object JInternalFrame

```
JInternalFrame intframe = new JInternalFrame();
```

Second, use method `setLayout` to change the layout

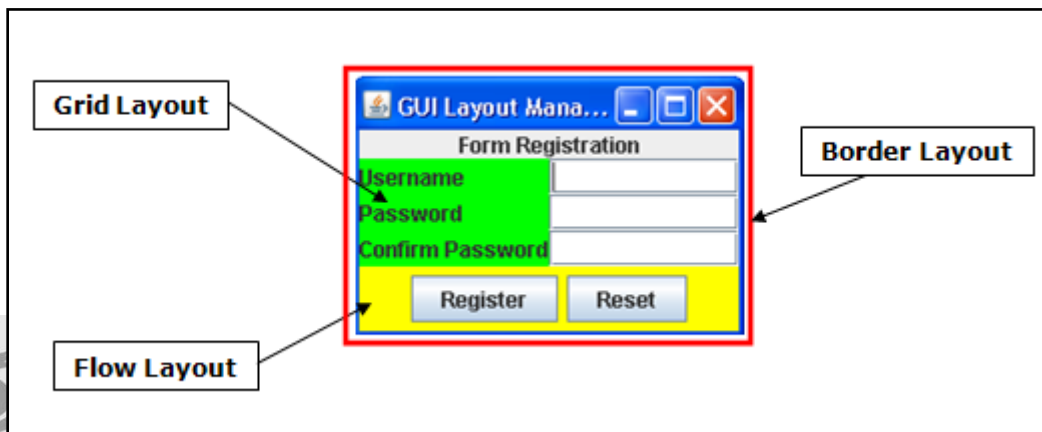
```
intframe.setLayout(manager);
```

Third, Fill parameter with `BorderLayout`

```
intframe.setLayout(new BorderLayout());
```


2.7. Exercise

Create Form Register



a. Task 01 - Create Project JAVA in Netbeans

1. Run Netbeans from Start Menu

2. Open Menu File -> New Project or Click icon  on bottom menu Edit

3. On Project Window, choose Java on Categories and Java Application on Projects



4. Next Setting Project name, Project Location and don't forget check "Create Main Class" like picture below

Name and Location

Project Name:

Project Location:

Project Folder:

Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

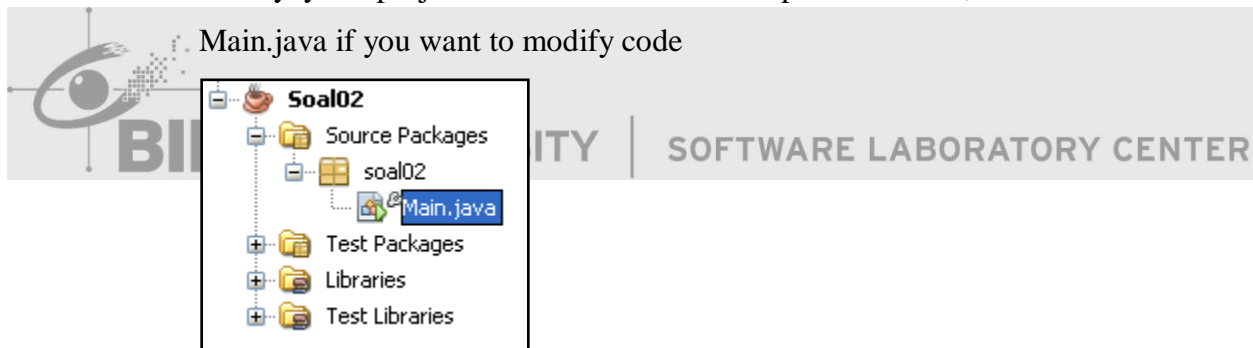
Create Main Class

Set as Main Project

5. Then Press Finish

6. Finally your project has been created like picture below, double click on file

Main.java if you want to modify code



b. Task 02 - Create JFrame and Set Properties JFrame

1. Create object JFrame

```
JFrame frame = new JFrame();
```

Explanation:

Same as Chapter 1

2. Setting properties JFrame

```
//Give title JFrame
frame.setTitle("GUI Layout Manager");

//Setting automatic fit size of JFrame
frame.pack();

//Make location JFrame will be located center window on screen
frame.setLocationRelativeTo(null);

//Make JFrame exit when user close it
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

//Show JFrame
frame.setVisible(true);
```

Explanation:

Same as Chapter 1

c. Task 03 – Using Container Class

1. Create object Container

```
Container c = frame.getContentPane();
```

Explanation:

Same as Chapter 1



2. Set layout of Container to BorderLayout

```
c.setLayout(new BorderLayout());
```

Explanation:

This code sets the layout manager for this container.

d. Task 04 - Create JPanel

1. Create object JPanel and set Layout

```
JPanel panel_center = new JPanel(new GridLayout(3,2));
JPanel panel_south = new JPanel(new FlowLayout());
```

Explanation:

In this section, we create two objects JPanel. Why two Objects? Not one or three? Because my assumption is with only two panels, we can make layout like exercise 03. One panel is set by GridLayout (3 rows,2 cols) and other panel is set by

FlowLayout. JPanel with GridLayout will be located on Center of Container, and the other will be located on South of Container.

2. Set Background Color JPanel

```
panel_center.setBackground(Color.green);
panel_south.setBackground(Color.yellow);
```

Explanation:

In this section, we will set background color of JPanel. First, we will set background color of 'panel_center' with Green. And second, we will set background color of 'panel_south' with Yellow.

You can set with other colors with use class Color.

e. Task 05 - Create Component

1. Create object Component Container direction "North"

```
JLabel lbl_title = new JLabel("Form Registration", JLabel.CENTER);
```

Explanation:

In this section, we will create object JLabel named 'lbl_title'. JLabel has constructor [JLabel\(String text, int horizontalAlignment\)](#). So we can directly make object JLabel with text and set position horizontal of JLabel. About position horizontal of JLabel, it will be explained in chapter 3

2. Create object Component Container direction "Center" (panel_center)

```
JLabel lbl_username = new JLabel("Username");
JLabel lbl_password = new JLabel("Password");
JLabel lbl_conf_password = new JLabel("Confirm Password");

JTextField txt_username = new JTextField();
JTextField txt_password = new JTextField();
JTextField txt_conf_password = new JTextField();
```

Explanation:

In this section, we create all components that will be located on center direction of Container. All of them will be added into 'panel_center'

3. Create object Component Container direction “South” (`panel_south`)

```
JButton btn_regis = new JButton("Register");
JButton btn_reset = new JButton("Reset");
```

Explanation:

In this section, we create all component that will be located on south direction of Container. All of them will be added into ‘`panel_south`’

- f. Task 06 – Add Component into JPanel

```
panel_center.add(lbl_username);
panel_center.add(txt_username);
panel_center.add(lbl_password);
panel_center.add(txt_password);
panel_center.add(lbl_conf_password);
panel_center.add(txt_conf_password);

panel_south.add(btn_regis);
panel_south.add(btn_reset);
```

Explanation:

In this section, we will add all components that we have created into each panel. Six lines earliest are statement to added component into ‘`panel_center`’ and two line latest are statement to added component into ‘`panel_south`’

- g. Task 07 – Add JPanel into Container and give direction

```
c.add(lbl_title, "North");
c.add(panel_center, "Center");
c.add(panel_south, "South");
```

Explanation:

This section is the final step. We will add each panel into Container based on border layout, so we need submit direction too when we add component into Container. At first we add object ‘`lbl_title`’ into North Direction. Second add ‘`panel_center`’ into Center Direction, and last add ‘`panel_south`’ into South Direction.

Chapter 03

GUI Basic Component



3.1. JLabel

A JLabel object can display a short text, an image, or both. You can specify where in the label's display area the label's contents are aligned by setting the vertical and horizontal alignment. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.

3.2. JButton

A JButton was used to give actions when JButton is “push” or clicked.

3.3. JTextField

JTextField is a lightweight component that allows the editing of a single line of text.

3.4. JTextArea

JTextArea is a multi-line area that displays plain text. JTextArea by default doesn't show the scrollbar. If you want to have the scrollbar showed on your JTextArea object, you can create JScrollPane object to hold an instance of JTextArea. Then the JScrollPane will handle scrolling for JTextArea.

TextArea has the ability to do line wrapping. This was controlled by the horizontal scrolling policy. Since scrolling is not done by JTextArea directly, backward compatibility must be provided another way. JTextArea has a bound property for line wrapping that controls whether or not it will wrap lines. By default, the line wrapping property is set to false (not wrapped).

3.5. JRadioButton

An implementation of a radio button, an item that can be selected or deselected, and which displays its state to the user. Used with a [ButtonGroup](#) object to create a group of buttons in which only one button at a time can be selected.

3.6. JCheckBox

An implementation of a check box, an item that can be selected or deselected, and which displays its state to the user. By convention, any number of check boxes in a group can be selected.

3.7. JComboBox

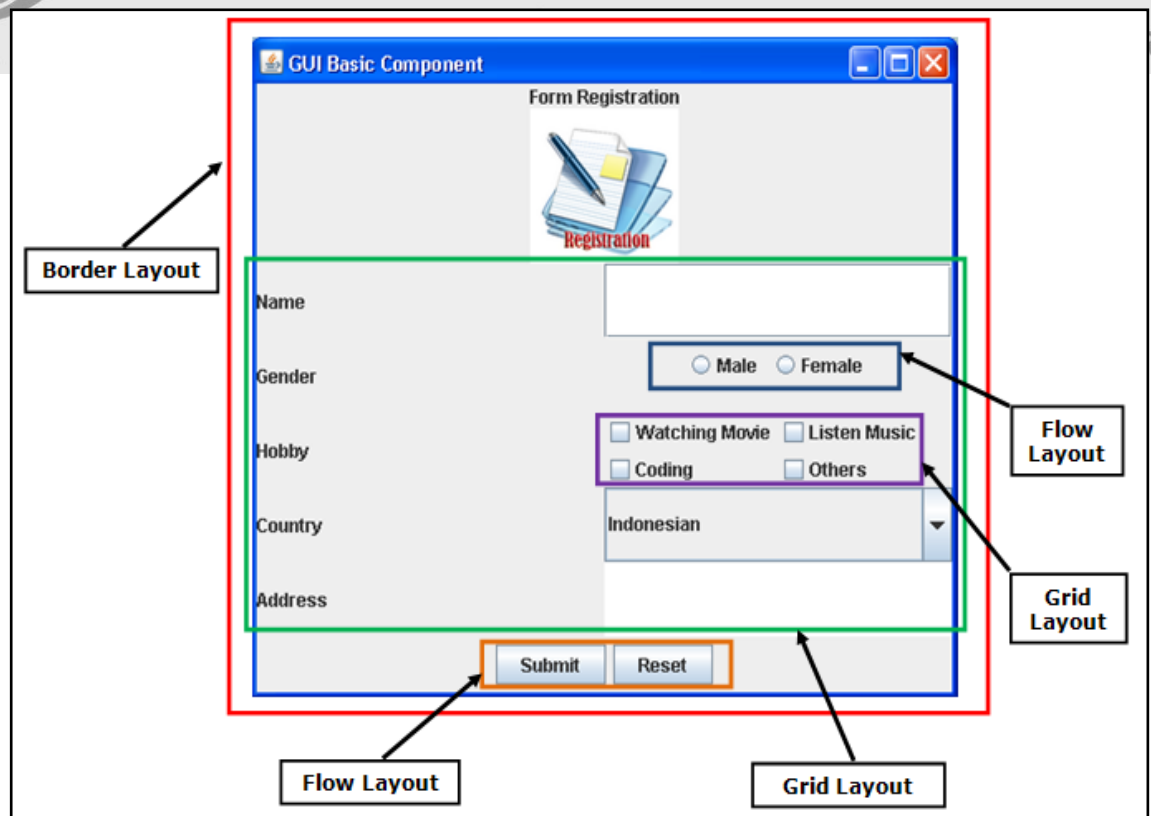
Described as a component which combined from button, drop-down list and editable field. The user can select a value from the drop-down list, which appears at the user's request.

3.8. ImageIcon

An implementation of the Icon interface that paints Icons from Images.Images that are created from a URL or filename.


3.9. Exercise

Create Form Register



a. Task 01 - Create Project JAVA in Netbeans

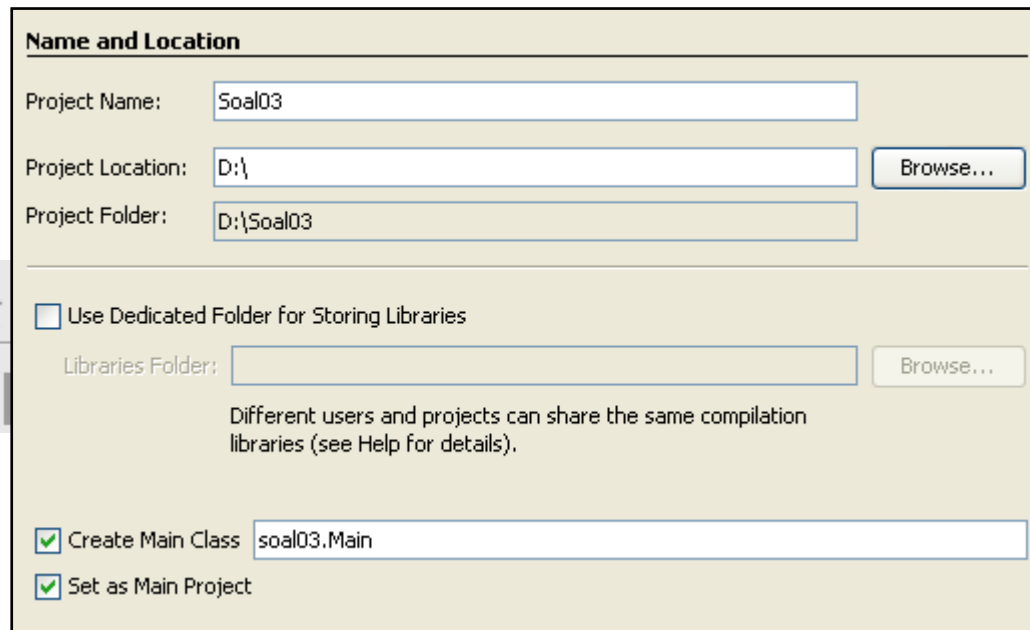
1. Run Netbeans from Start Menu

2. Open Menu File -> New Project or Click icon  on bottom menu Edit

3. On Project Window, choose Java on Categories and Java Application on Projects

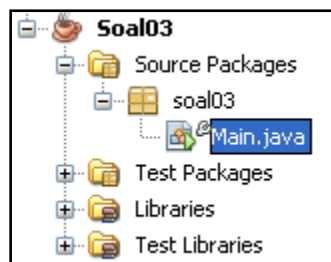


4. Next Setting Project name, Project Location and don't forget check "Create Main Class" like picture below



5. Then Press Finish

6. Finally your project has been created like picture below, double click on file Main.java if you want to modify code



b. Task 02 - Create JFrame and Set Properties JFrame

1. Create object JFrame

```
JFrame frame = new JFrame();
```

2. Setting properties JFrame

```
frame.setTitle("GUI Basic Component");  
frame.pack();  
frame.setLocationRelativeTo(null);  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
frame.setVisible(true);
```

Explanation:

Same as Chapter 1

c. Task 03 – Using Container Class

1. Create object Container

```
Container c = frame.getContentPane();
```

Explanation:

Same as Chapter 1

2. Set layout of Container to BorderLayout

```
c.setLayout(new BorderLayout());
```

Explanation:

Sets the layout manager of Container into Border Layout

d. Task 04 - Create JPanel

1. Create object JPanel and set Layout

```
JPanel panel_north = new JPanel(new BorderLayout());
```

Explanation:

Create object JPanel named 'panel_north' that will be located on North Direction of Container. Directly set layout into Border Layout when created object. Only object JLabel (Title) and JLabel (Icon) will be added into 'panel_north'.

```
JPanel panel_center = new JPanel(new GridLayout(5,2));
```

Explanation:

Create object JPanel named 'panel_center' that will be located on Center Direction of Container. Directly set layout into Grid Layout with 5 rows and 2 cols when created object. Core components such as JTextField, JRadioButton, JCheckBox, JComboBox, and JTextArea will be added into this panel.

```
JPanel panel_south = new JPanel(new FlowLayout());
```

Explanation:

Create object JPanel named 'panel_south' that will be located on South Direction of Container. Directly set layout into Flow layout when created object. Only two components of JButton will be added into this panel.

```
JPanel panel_radio = new JPanel(new FlowLayout());
```

Explanation:

Create object JPanel named 'panel_radio' that will be located on right side of Gender. Directly set layout into Flow layout when created object. Only two components of JRadioButton will be added into this panel.

```
JPanel panel_check = new JPanel(new GridLayout(2, 2));
```

Explanation:

Create object JPanel named 'panel_check' that will be located on right side of Hobby. Directly set layout into Grid Layout with 2 rows and 2 cols when created object. Four components of JCheckBox will be added into this panel.

e. Task 05 - Create Component

1. Create object Component Container direction “North”

```
JLabel lbl_title = new JLabel("Form Registration", JLabel.CENTER);
ImageIcon icon = new ImageIcon("src/registration.png");
JLabel lbl_icon = new JLabel(icon, JLabel.CENTER);
```

```
JLabel lbl_title = new JLabel("Form Registration", JLabel.CENTER);
```

Explanation:

Create object JLabel named ‘lbl_title’. JLabel has constructor [*JLabel\(String text, int horizontalAlignment\)*](#). So we can directly make object JLabel with text and set position horizontal of JLabel. About position horizontal of JLabel, there are several parameters that can be used.

➤ JLabel.[BOTTOM ALIGNMENT](#)

Ease-of-use constant for `getAlignmentY()`. It specifies an alignment to the bottom of the component.

➤ JLabel.[CENTER ALIGNMENT](#)

Ease-of-use constant for `getAlignmentY()` and `getAlignmentX()`. It specifies an alignment to the center of the component.

➤ JLabel.[LEFT ALIGNMENT](#)

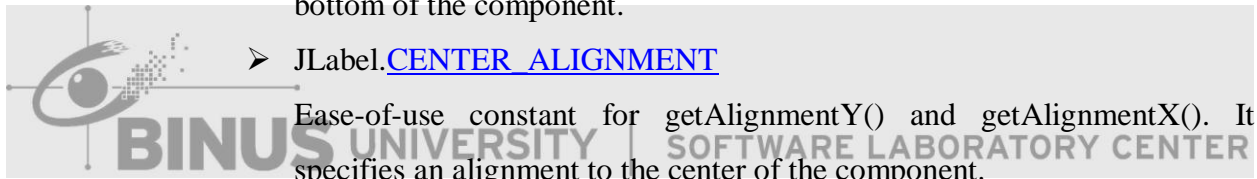
Ease-of-use constant for `getAlignmentX()`. It specifies an alignment to the left side of the component.

➤ JLabel.[RIGHT ALIGNMENT](#)

Ease-of-use constant for `getAlignmentX()`. It specifies an alignment to the right side of the component.

➤ JLabel.[TOP ALIGNMENT](#)

Ease-of-use constant for `getAlignmentY()`. It specifies an alignment to the top of the component.

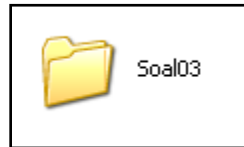


```
ImageIcon icon = new ImageIcon("src/registration.png");
```

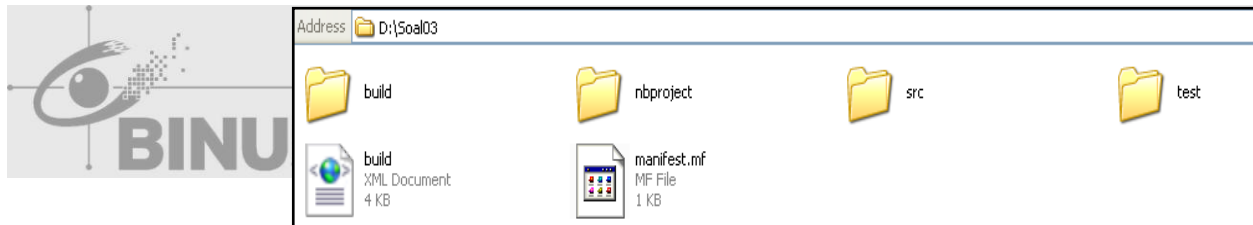
Explanation:

Create object ImageIcon named 'icon'. You can import file directly that have picture format such as .jpg, .bmp, .png, .gif and etc. with explain the location of file in parameter. This section will be explained step by step about directory file.

- First, you have project folder that named by "Soal03" like picture below



- Second, Double click the folder and you will get the location of project like picture below. This is "default first directory" if you want to import the file.



- Third, in case, picture file named "registration.png" has been located in folder "src". So it must be informed the location of file is "src/registration.png". Don't give space or make capital letter when the name of file is not capital letter because it's case sensitive.



```
JLabel lbl_icon = new JLabel(icon, JLabel.CENTER);
```

Explanation:

Create object JLabel named 'lbl_icon'. JLabel has constructor [JLabel\(Icon image, int horizontalAlignment\)](#). So we can directly create object JLabel with icon and set horizontal position of JLabel. Fill parameter *Icon image* with object ImageIcon named icon.

2. Create object Component Container direction "Center" (panel_center)

```
JLabel lbl_name = new JLabel("Name");
JLabel lbl_gender = new JLabel("Gender");
JLabel lbl_hobby = new JLabel("Hobby");
JLabel lbl_country = new JLabel("Country");
JLabel lbl_address = new JLabel("Address");

JTextField txt_name = new JTextField();
JRadioButton radio_male = new JRadioButton("Male");
JRadioButton radio_female = new JRadioButton("Female");
JCheckBox chk_listen = new JCheckBox("Listen Music");
JCheckBox chk_watch = new JCheckBox("Watching Movie");
JCheckBox chk_coding = new JCheckBox("Coding");
JCheckBox chk_others = new JCheckBox("Others");
JComboBox cmb_country = new JComboBox();
JTextArea txt_address = new JTextArea();

ButtonGroup btn_group = new ButtonGroup();
JScrollPane spane = new JScrollPane();
```

```
JLabel lbl_name = new JLabel("Name");
JLabel lbl_gender = new JLabel("Gender");
JLabel lbl_hobby = new JLabel("Hobby");
JLabel lbl_country = new JLabel("Country");
JLabel lbl_address = new JLabel("Address");
```

Explanation:

Create all objects JLabel as the picture above. JLabel has constructor [JLabel\(String text\)](#). So we can directly create object JLabel with text.

```
JTextField txt_name = new JTextField();
```

Explanation:

Create object JTextField named 'txt_name'. Use default constructor of JLabel with empty parameter.

```
JRadioButton radio_male = new JRadioButton("Male");
JRadioButton radio_female = new JRadioButton("Female");
```

Explanation:

Create object JRadioButton named 'radio_male' and 'radio_female'. JRadioButton has constructor [JRadioButton\(String text\)](#). So we can directly create object JRadioButton with text. **JRadioButton always combined with ButtonGroup** because ButtonGroup can create a group of buttons in which only one button at a time can be selected.

```
ButtonGroup btn_group = new ButtonGroup();
```

Explanation:

Create object ButtonGroup named 'btn_group'. ButtonGroup only has one default constructor [ButtonGroup\(\)](#).

```
JCheckBox chk_listen = new JCheckBox("Listen Music");
JCheckBox chk_watch = new JCheckBox("Watching Movie");
JCheckBox chk_coding = new JCheckBox("Coding");
JCheckBox chk_others = new JCheckBox("Others");
```

Explanation:

Create all objects JCheckBox like picture. JCheckBox has constructor [JCheckBox\(String text\)](#). So we can directly create object JCheckBox with text.

```
JComboBox cmb_country = new JComboBox();
```

Explanation:

Create object JComboBox named 'cmb_country'. Use default constructor of JComboBox with empty parameter.

```
JTextArea txt_address = new JTextArea();
```

Explanation:

Create object JTextArea named 'txt_address'. Use default constructor of JTextArea with empty parameter.

```
JScrollPane spane = new JScrollPane();
```

Explanation:

Create object JScrollPane named 'spane'. Use default constructor of JScrollPane with empty parameter. JScrollPane is always combined with JTextArea to create scroll bar in JTextArea automatically.

3. Create object Component Container direction "South" (panel_south)

```
JButton btn_submit = new JButton("Submit");
JButton btn_reset = new JButton("Reset");
```

Explanation:

Create object JButton named 'btn_submit' and 'btn_reset'. JButton has constructor [JButton\(String text\)](#). So we can directly create object JButton with text.

- f. Task 06 – Add Item into JComboBox

```
cmb_country.addItem("Indonesian");
cmb_country.addItem("Japan");
cmb_country.addItem("USA");
cmb_country.addItem("China");
cmb_country.addItem("Others");
```

Explanation:

Method **addItem**([Object](#) anObject) is a method JComboBox that used to add item to the item list. In example, add “Indonesian”, “Japan”, “USA”, “China”, and “Other” into item list of ‘cmb_country’.

g. Task 07 – Add Component into each JPanel

1. Add all components that located in the North of Container into ‘panel_north’

```
panel_north.add(lbl_title, "North");
panel_north.add(lbl_icon, "Center");
```

Explanation:

First, add object ‘lbl_title’ into ‘panel_north’ with direction “North”. Second, add object ‘lbl_icon’ into ‘panel_north’ with direction “Center”. We add direction in second parameter because ‘panel_north’ use BorderLayout.

2. Add all components that located in the Center of Container into ‘panel_center’

```
panel_center.add(lbl_name);
panel_center.add(txt_name);

btn_group.add(radio_male);
btn_group.add(radio_female);
panel_radio.add(radio_male);
panel_radio.add(radio_female);
panel_center.add(lbl_gender);
panel_center.add(panel_radio);

panel_check.add(chk_watch);
panel_check.add(chk_listen);
panel_check.add(chk_coding);
panel_check.add(chk_others);
panel_center.add(lbl_hobby);
panel_center.add(panel_check);

panel_center.add(lbl_country);
panel_center.add(cmb_country);

txt_address.setLineWrap(true);
spane.setViewportView(txt_address);
panel_center.add(lbl_address);
panel_center.add(spane);
```

Explanation:

Because 'panel_center' use GridLayout, therefore when we add each component into 'panel_center', we must add each component one by one according the order.

```
panel_center.add(lbl_name);  
panel_center.add(txt_name);
```

Explanation:

According to the order of components, add object lbl_name and txt_name into 'panel_center'.

```
btn_group.add(radio_male);  
btn_group.add(radio_female);  
panel_radio.add(radio_male);  
panel_radio.add(radio_female);
```

Explanation:

First, add radio_male and radio female into btn_group. Then, add radio_male and radio_female into panel_radio. Object btn_group in this section just created a group of buttons. Object panel_radio in this section is to create a button panel.

```
panel_center.add(lbl_gender);  
panel_center.add(panel_radio);
```

Explanation:

According to the order of components, add object lbl_gender and panel_radio into panel_center

```
panel_check.add(chk_watch);  
panel_check.add(chk_listen);  
panel_check.add(chk_coding);  
panel_check.add(chk_others);
```

Explanation:

Same as JRadioButton, add all JCheckBox into panel_check to create a button panel.

```
panel_center.add(lbl_hobby);  
panel_center.add(panel_check);
```

Explanation:

According to the order of components, add object lbl_hobby and panel_check into panel_center

```
panel_center.add(lbl_country);  
panel_center.add(cmb_country);
```

Explanation:

According to the order of components, add object lbl_country and cmb_country into panel_center

```
txt_address.setLineWrap(true);  
spane.setViewportViewView(txt_address);
```

Explanation:

Method [setLineWrap](#)(boolean wrap) is a method of JTextArea to set the line-wrapping policy of the text area. If set to true the lines will be wrapped if they are too long to fit within the allocated width. If set to false, the lines will always be unwrapped.

Method setViewportView will be explained later in chapter 4. We must combine JTextArea with JScrollPane to make correct Text Area.


```
panel_center.add(lbl_address);  
panel_center.add(spans);
```

Explanation:

According to the order of components, add object lbl_address and spans into panel_center

3. Add all components that located in the South of Container into 'panel_south'

```
panel_south.add(btn_submit);  
panel_south.add(btn_reset);
```

Explanation:

Add object btn_submit and btn_reset into panel_south

- h. Task 08 – Add JPanel into Container and give direction

```
c.add(panel_north, "North");  
c.add(panel_center, "Center");  
c.add(panel_south, "South");
```

Explanation:

This section is the final step. We will add each panel into Container based on border layout, so we need submit direction too when we add component into Container. At first, we add object 'panel_north' into North Direction. Second, add 'panel_center' into Center Direction, and last add 'panel_south' into South Direction.

Chapter 04

GUI Intermediate Component



4.1. **JMenuBar**

An implementation of a menu bar. You add JMenu objects to the menu bar to construct a menu. When the user selects a JMenu object, its associated JPopupMenu is displayed, allowing the user to select one of the JMenuItem objects on it.

4.2. **JMenu**

An implementation of a menu containing JMenuItem objects that is displayed when the user selects an item on the JMenuBar. In addition to JMenuItem objects, a JMenu can also contain JSeparator objects.

4.3. **JMenuItem**

An implementation of an item in a menu. A menu item is essentially a button sitting in a list. When the user selects the "button", the action associated with the menu item is performed. A JMenuItem contained in a JPopupMenu performs exactly that function.

4.4. **JSeparator**

JSeparator provides a general purpose component for implementing divider lines - most commonly used as a divider between menu items that breaks them up into logical groupings.

4.5. **JScrollPane**

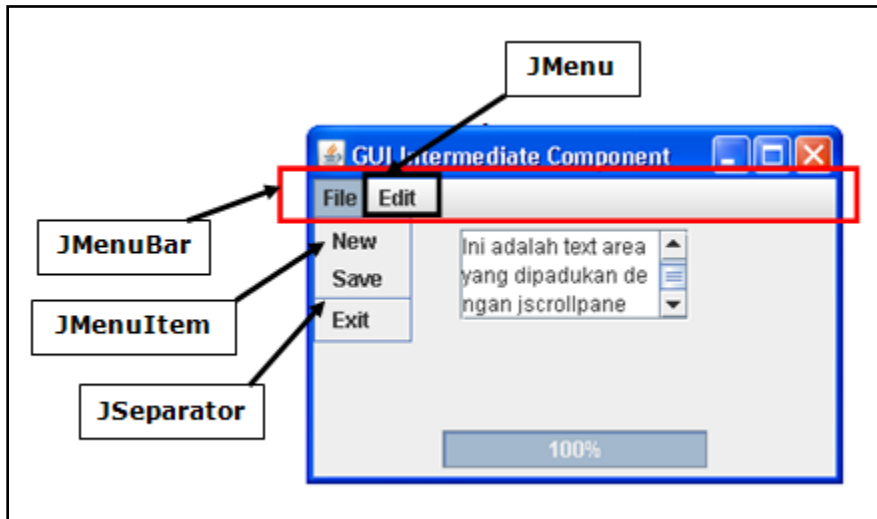
Provides a scrollable view of a lightweight component. A JScrollPane manages a viewport, optional vertical and horizontal scroll bars, and optional row and column heading viewports.

4.6. **JProgressBar**

A component that visually displays the progress of some task. As the task progresses towards completion, the progress bar displays the task's percentage of completion. This percentage is typically represented visually by a rectangle which starts out empty and gradually becomes filled in as the task progresses. In addition, the progress bar can display a textual representation of this percentage.

4.7. Exercise



Create Form With Menu, Progress Bar and Text Area



Note:

JProgressBar in the picture using Thread to show the workflow JProgressBar

a. Task 01 - Create Project JAVA in Netbeans

1. Run Netbeans from Start Menu
 2. Open Menu File -> New Project or Click icon  on bottom menu Edit
 3. On Project Window, choose Java on Categories and Java Application on Projects
- 
5. Next Setting Project name, Project Location and don't forget check "Create Main Class" like picture below

Name and Location

Project Name:

Project Location:

Project Folder:

Use Dedicated Folder for Storing Libraries

Libraries Folder:

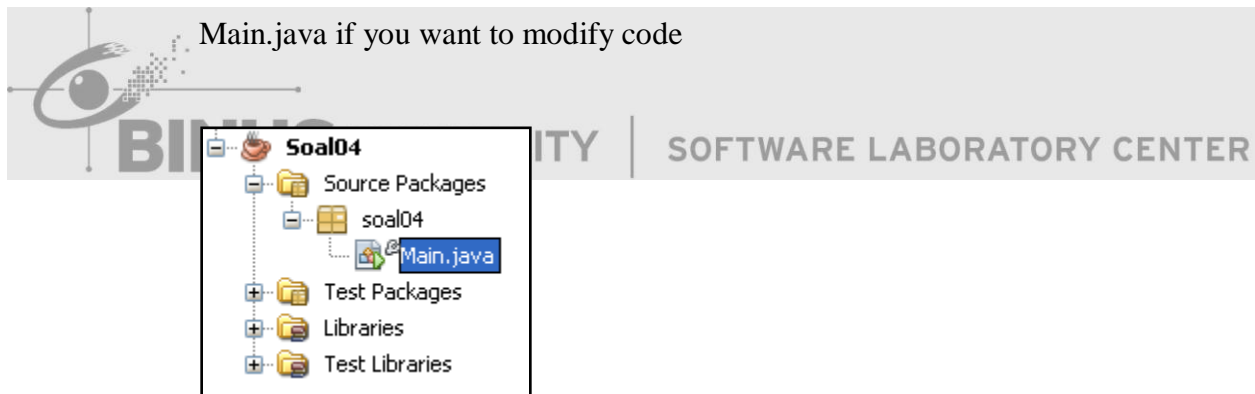
Different users and projects can share the same compilation libraries (see Help for details).

Create Main Class

Set as Main Project

6. Then Press Finish

7. Finally your project has been created like picture below, double click on file Main.java if you want to modify code



b. Task 02 - Create JFrame and Set Properties JFrame

1. Create object JFrame

```
JFrame frame = new JFrame();
```

Explanation:

Same as Chapter 1

2. Setting properties JFrame

```
frame.setTitle("GUI Basic Component");
frame.pack();
frame.setLocationRelativeTo(null);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```

Explanation:

Same as Chapter 1

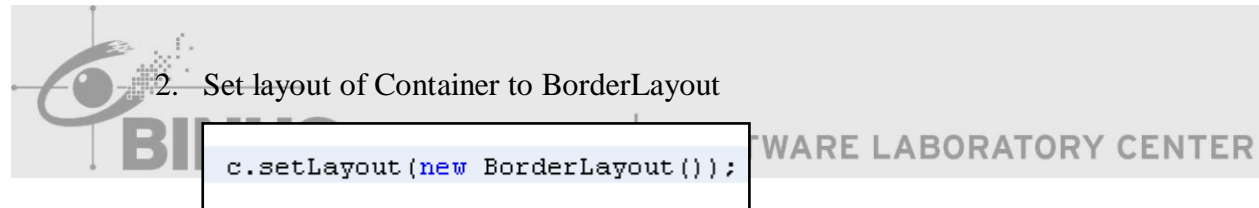
c. Task 03 – Using Container Class

1. Create object Container

```
Container c = frame.getContentPane();
```

Explanation:

Same as Chapter 1



2. Set layout of Container to BorderLayout

```
c.setLayout(new BorderLayout());
```

Explanation:

Sets the layout manager of Container into Border Layout

d. Task 04 - Create JPanel

1. Create object JPanel and set Layout

```
JPanel panel_center = new JPanel();
```

Explanation:

Create object JPanel named 'panel_center' that will be located on Center Direction of Container. Use default Layout JPanel (FlowLayout). Only object JTextArea and JScrollPane will be added into this panel.

```
JPanel panel_south = new JPanel();
```

Explanation:

Create object JPanel named 'panel_south' that will be located on South Direction of Container. Use default Layout JPanel (FlowLayout). Only object JProgressBar will be added into this panel.

e. Task 05 - Create Component

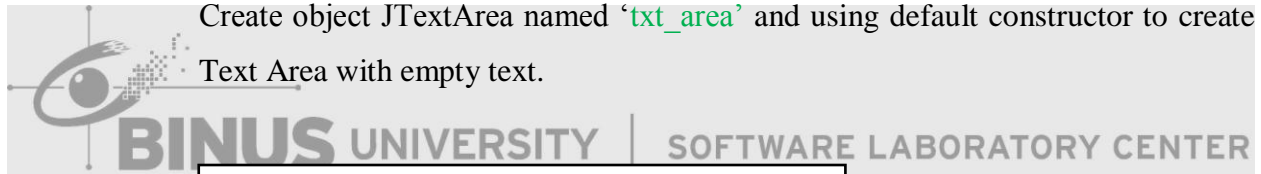
1. Create object Component Container direction "Center"

```
JTextArea txt_area = new JTextArea();
JScrollPane spane = new JScrollPane();
```

```
JTextArea txt_area = new JTextArea();
```

Explanation:

Create object JTextArea named 'txt_area' and using default constructor to create Text Area with empty text.



```
JScrollPane spane = new JScrollPane();
```

Explanation:

Create object JScrollPane named 'spane' and using default Constructor to create an empty (no viewport view) JScrollPane where both horizontal and vertical scrollbars appear when needed.

2. Create object Component Container direction "South"

```
Thread th;
JProgressBar pbar = new JProgressBar();
```

```
Thread th;
```

Explanation:

Just declare an object of Class Thread named 'Th'.

```
JProgressBar pbar = new JProgressBar();
```

Explanation:

Create object JProgressBar named 'pbar' and using default Constructor to create a horizontal progress bar that displays a border but no progress string. The initial and minimum values are 0, and the maximum is 100.

3. Create object Component Menu on the top of JFrame

```
JMenuBar menu_bar = new JMenuBar();

JMenu menu_file = new JMenu("File");
JMenu menu_edit = new JMenu("Edit");

JMenuItem item_new = new JMenuItem("New");
JMenuItem item_save = new JMenuItem("Save");
JMenuItem item_exit = new JMenuItem("Exit");

JMenuItem item_cut = new JMenuItem("Cut");
JMenuItem item_copy = new JMenuItem("Copy");
JMenuItem item_paste = new JMenuItem("Paste");

JSeparator separator = new JSeparator();
```

```
JMenuBar menu_bar = new JMenuBar();
```

Explanation:

Create object JProgressBar named 'menu_bar' and using default Constructor to create a new menu bar. Firstly, we must create JMenuBar before create JMenu and JMenuItem.

```
JMenu menu_file = new JMenu("File");
JMenu menu_edit = new JMenu("Edit");
```

Explanation:

Create object JMenu named 'menu_file' and 'menu_edit'. Use constructor **JMenu(String text)** to construct a new JMenu with the supplied string as its text.


```
JMenuItem item_new = new JMenuItem("New");
JMenuItem item_save = new JMenuItem("Save");
JMenuItem item_exit = new JMenuItem("Exit");
```

Explanation:

Create all objects JMenuItem named and using constructor **JMenuItem**([String](#) text) to create a JMenuItem with the specified text. All of them will be located in menu_file.

```
JMenuItem item_cut = new JMenuItem("Cut");
JMenuItem item_copy = new JMenuItem("Copy");
JMenuItem item_paste = new JMenuItem("Paste");
```

Explanation:

Create all objects JMenuItem named and using constructor **JMenuItem**([String](#) text) to create a JMenuItem with the specified text. All of them will be located in menu_edit.

```
JSeparator separator = new JSeparator();
```

Explanation:

Create object JSeparator named 'separator' (for new horizontal separator).

f. Task 06 – Create Menu in JFrame

```
menu_file.add(item_new);
menu_file.add(item_save);
menu_file.add(separator);
menu_file.add(item_exit);

menu_edit.add(item_cut);
menu_edit.add(item_copy);
menu_edit.add(item_paste);

menu_bar.add(menu_file);
menu_bar.add(menu_edit);

frame.setJMenuBar(menu_bar);
```

```
menu_file.add(item_new);  
menu_file.add(item_save);  
menu_file.add(separator);  
menu_file.add(item_exit);
```

Explanation:

Method [add\(Action a\)](#) is a method JMenu that used to creates a new menu item attached to the specified Action object and appends it to the end of this menu. Fill parameter with object JMenuItem that have created before. When insert JMenuItem, you should pay attention to the order of menu item.

```
menu_edit.add(item_cut);  
menu_edit.add(item_copy);  
menu_edit.add(item_paste);
```

Explanation:

Method [add\(Action a\)](#) is a method JMenu that used to creates a new menu item attached to the specified Action object and appends it to the end of this menu. Fill parameter with object JMenuItem that have created before. When insert JMenuItem, you should pay attention to the order of menu item.

```
menu_bar.add(menu_file);  
menu_bar.add(menu_edit);
```

Explanation:

Method [add\(JMenu c\)](#) is a method JMenuBar that used to appends the specified menu to the end of the menu bar. Fill parameter JMenu with object JMenu that have created before named menu_file and menu_edit.

```
frame.setJMenuBar(menu_bar);
```

Explanation :

Method [setJMenuBar\(JMenuBar menubar\)](#) is a method JFrame that used to add JMenuBar into top of JFrame. Fill parameter menubar with object JMenuBar that have created before named menu_bar.

g. Task 07 – Setting Properties JTextArea and Making Animation Progress

JProgressBar

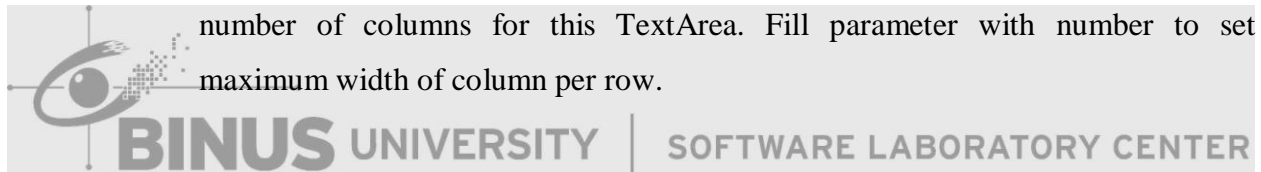
1. Setting properties JTextArea

```
txt_area.setColumns(10);
txt_area.setRows(3);
txt_area.setLineWrap(true);
spane.setViewportView(txt_area);
```

```
txt_area.setColumns(10);
```

Explanation:

Method [setColumns\(int columns\)](#) is a method JTextArea that used to set the number of columns for this TextArea. Fill parameter with number to set maximum width of column per row.



```
txt_area.setRows(3);
```

Explanation:

Method [setRows\(int rows\)](#) is a method JTextArea that used to set the number of rows for this TextArea. Fill parameter with number to adjust the row that will be displayed in the text area.

```
txt_area.setLineWrap(true);
```

Explanation:

Method [setLineWrap\(boolean wrap\)](#) is a method JTextArea that used to set the line-wrapping policy of the text area. If set to true the lines will be wrapped if they are too long to fit within the allocated width. If set to false, the lines will always be unwrapped.

```
spane.setViewportView(txt_area);
```

Explanation:

Method [setViewportView\(Component view\)](#) is a method JScrollPane that used to creates a viewport if necessary and then sets its view. This setting properties will make object txt_area has scrollbar if the row of text area is not enough.

2. Making animation progress JProgressBar

```
th = new Thread()
{
    @Override
    public void run() {
        super.run();

        while(true)
        {
            if(pbar.getValue() <= pbar.getMaximum())
            {
                pbar.setString(pbar.getValue() + "%");
                pbar.setValue(pbar.getValue() + 1);
            }

            try {
                th.sleep(10);
            } catch (InterruptedException ex) {
                Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
};
th.start();
pbar.setStringPainted(true);
```

```
if(pbar.getValue() <= pbar.getMaximum())
{
    pbar.setString(pbar.getValue() + "%");
    pbar.setValue(pbar.getValue() + 1);
}
```

Explanation:

1. Method [getValue\(\)](#) is a method JProgressBar that returns the progress bar's current value.

- Method [getMaximum\(\)](#) is a method JProgressBar that returns the progress bar's maximum value (100)

```
if(pbar.getValue() <= pbar.getMaximum())
```

- Do selection with condition if Progress Bar's current value smaller or same as Progress Bar's maximum value then doing statement.

```
pbar.setString(pbar.getValue() + "%");
pbar.setValue(pbar.getValue() + 1);
```

- Method [setString\(String s\)](#) is a method JProgressBar that used to sets the value of the progress string. Fill parameter like sample to get Progress Bar's current value and show the value.

- Method [setValue\(int n\)](#) is a method JProgressBar that used to sets the progress bar's current value to n. This method forwards the new value to the model. Fill parameter like sample to do increment value one by one until reach the Progress Bar's maximum value.

```
th.sleep(10);
```

Explanation:

Method [sleep\(long millis\)](#) is a method Thread that causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers. In short term, it delays the program execution by 10 milliseconds.

```
th.start();
```

Explanation:

Method [start\(\)](#) is a method Thread that causes this thread to begin execution. Don't forget to start the thread when you used it.

```
pbar.setStringPainted(true);
```

Explanation:

Method [setStringPainted](#)(boolean b) is a method JProgressBar that used to sets the value of the stringPainted property, which determines whether the progress bar should render a progress string.

h. Task 08 – Add Component into each JPanel

1. Add all components that located in the Center of Container into 'panel_center'

```
panel_center.add(spave);
```

Explanation:

Add object 'spave' into 'panel_center'. Object txt_area is include in the object spave.

2. Add all components that located in the South of Container into 'panel_south'

```
panel_south.add(pbar);
```

Explanation:

Add object 'pbar' into 'panel_south'.

i. Task 09– Add JPanel into Container and give direction

```
c.add(panel_center, "Center");
c.add(panel_south, "South");
```

Explanation:

This section is final step. We will add each panel into Container based on border layout, so we need submit direction too when we add component into Container. Add object 'panel_center' into Center Direction. Then add 'panel_south' into South Direction.

Chapter 05

GUI Advanced Component



5.1 JList

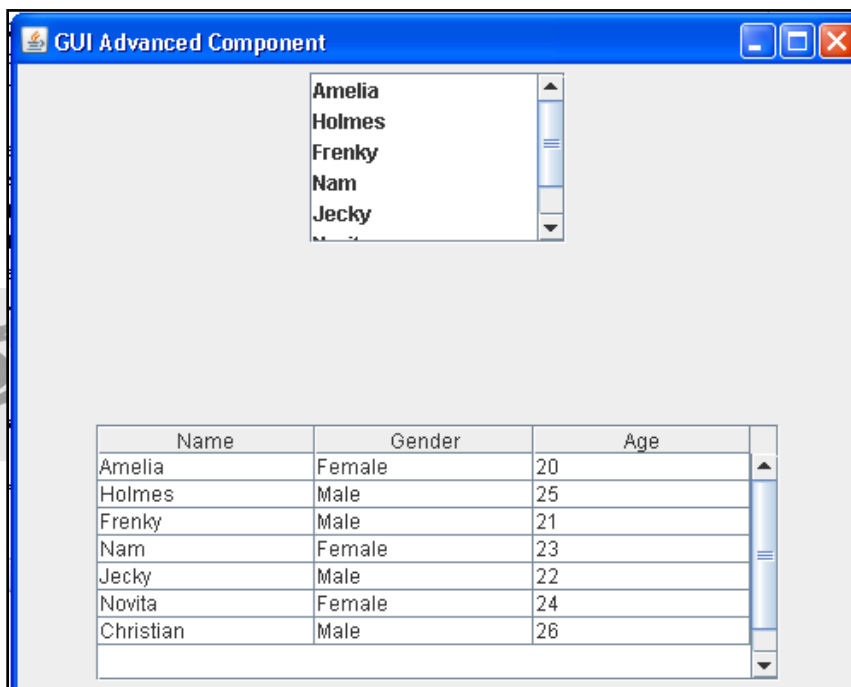
Described as a component that displays a list of objects and allows the user to select one or more items.

5.2 JTable

The JTable is used to display and edit regular two-dimensional tables of cells.


5.3 Exercise

Create Form with JList and JTable

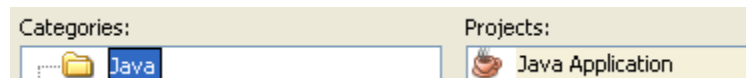


a. Task 01 - Create Project JAVA in Netbeans

1. Run Netbeans from Start Menu

2. Open Menu File -> New Project or Click icon  on bottom menu Edit

3. On Project Window, choose Java on Categories and Java Application on Projects



4. Next Setting Project name, Project Location and don't forget check "Create Main Class" like picture below

Name and Location

Project Name:

Project Location:

Project Folder:

Use Dedicated Folder for Storing Libraries

Libraries Folder:

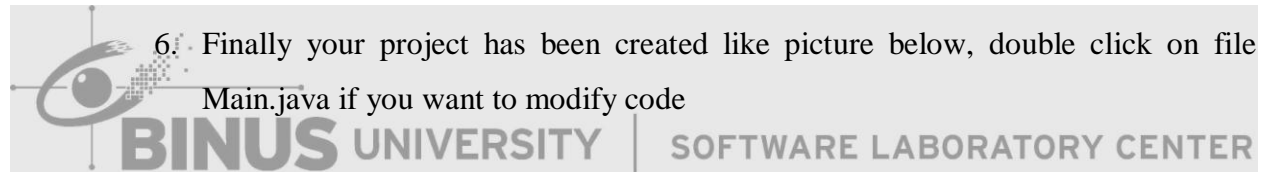
Different users and projects can share the same compilation libraries (see Help for details).

Create Main Class

Set as Main Project

5. Then Press Finish

6. Finally your project has been created like picture below, double click on file Main.java if you want to modify code




b. Task 02 - Create JFrame and Set Properties JFrame

1. Create object JFrame

```
JFrame frame = new JFrame();
```

Explanation:

Same as Chapter 1

2. Setting properties JFrame

```
frame.setTitle("GUI Basic Component");
frame.pack();
frame.setLocationRelativeTo(null);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```

Explanation:

Same as Chapter 1

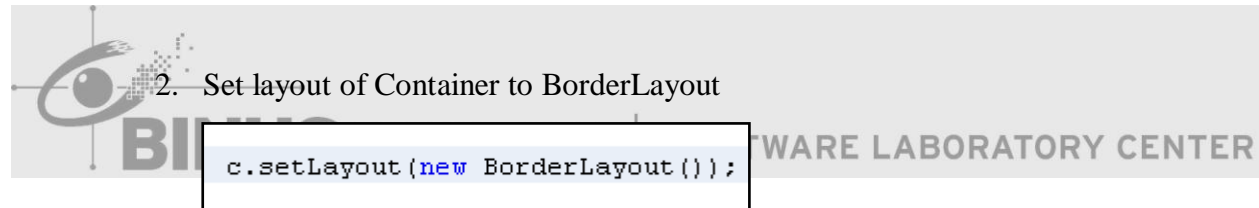
c. Task 03 – Using Container Class

1. Create object Container

```
Container c = frame.getContentPane();
```

Explanation:

Same as Chapter 1



2. Set layout of Container to BorderLayout

```
c.setLayout(new BorderLayout());
```

Explanation:

Sets the layout manager of Container into Border Layout

d. Task 04 - Create JPanel

1. Create object JPanel and set Layout

```
JPanel panel_center = new JPanel();
```

Explanation:

Create object JPanel named 'panel_center' that will be located on Center Direction of Container. Use default Layout JPanel (FlowLayout). Only object JList will be added into this panel.

```
JPanel panel_south = new JPanel();
```

Explanation:

Create object JPanel named 'panel_south' that will be located on South Direction of Container. Use default Layout JPanel (FlowLayout). Only object JLabel will be added into this panel.

e. Task 05 - Create Component

1. Create object Component Container direction "Center"

```
JList list = new JList();
JScrollPane spaneList = new JScrollPane();
```

```
JList list = new JList();
```

Explanation:

Create object JList named 'list' and using default Constructor to constructs a JList with an empty, read-only, model.

```
JScrollPane spaneList = new JScrollPane();
```

Explanation:

Create object JScrollPane named 'spanelist' and using default Constructor to creates an empty (no viewport view) JScrollPane where both horizontal and vertical scrollbars appear when needed.

2. Create object Component Container direction "South"

```
JTable tabel = new JTable();
JScrollPane spaneTabel = new JScrollPane();
```

```
JTable tabel = new JTable();
```

Explanation:

Create object JTable named 'tabel' and using default Constructor to constructs a default JTable that is initialized with a default data model, a default column model, and a default selection model.

```
JScrollPane spaneTabel = new JScrollPane();
```

Explanation:

Create object JScrollPane named 'spanetabel' and using default Constructor to creates an empty (no viewport view) JScrollPane where both horizontal and vertical scrollbars appear when needed.

f. Task 06 – How to insert data into Jlist

1. Initialization data

```
String[] name = {"Amelia", "Holmes", "Frenky", "Nam", "Jecky", "Novita", "Christian"};
```

Explanation:

Declaration of all data into Array String

2. Create object Vector

```
Vector<String> vec_name = new Vector<String>();
```

Explanation:

Create object Vector (String data type) named 'vec_name' and using default Constructor to constructs an empty vector.

```
for (int i = 0; i < name.length; i++) {
    vec_name.add(name[i]);
}

list.setListData(vec_name);
spaneList.setPreferredSize(new Dimension(150, 100));
spaneList.setViewPortView(list);
```

3. Insert data into Vector

```
for (int i = 0; i < name.length; i++) {  
    vec_name.add(name[i]);  
}
```

Explanation:

Method [add\(E e\)](#) is a method `Vector<E>` that used to Appends the specified element to the end of this Vector. Do looping array object 'name' from first index (0) until last index (`name.length`) to add all data from array object 'name' into Vector object 'vec_name'

4. Set list data with Vector

```
list.setListData(vec_name);
```

Explanation:

Method [setListData\(Vector<?> listData\)](#) is a method `JList` that used to constructs a read-only `ListModel` from a `Vector` then show data from `Vector` to `JList`. Fill parameter with object `Vector` named 'vec_name'.

5. Set size for JScrollPane

```
spaneList.setPreferredSize(new Dimension(150, 100));
```

Explanation:

Method [setPreferredSize\(Dimension preferredSize\)](#) is a inherited method from class `javax.swing.JComponent` that used to sets the preferred size of this component.

6. Set view for JScrollPane

```
spaneList.setViewportViewView(list);
```

Explanation:

Method [setViewportView\(Component view\)](#) is a method JScrollPane that used to creates a viewport if necessary and then sets its view. This setting property will make object 'list' has scrollbar if the row of text area is not enough.

g. Task 07 - How to insert data into JTable


1. Initialization data

```
String[] name = {"Amelia", "Holmes", "Frenky", "Nam", "Jecky", "Novita", "Christian"};
String[] gender = {"Female", "Male", "Male", "Female", "Male", "Female", "Male"};
String[] age = {"20", "25", "21", "23", "22", "24", "26"};
```

Explanation:

Declaration of all data into One Dimension Array String

2. Create some objects Vector for table



```
Vector<String> vec_header = new Vector<String>();
Vector<Vector> vec_data = new Vector<Vector>();
Vector<String> vec_detail;

DefaultTableModel dtm = new DefaultTableModel(vec_data, vec_header)
{
    @Override
    public boolean isCellEditable(int row, int column) {
        return false;
    }
};
```

```
Vector<String> vec_header = new Vector<String>();
```

Explanation:

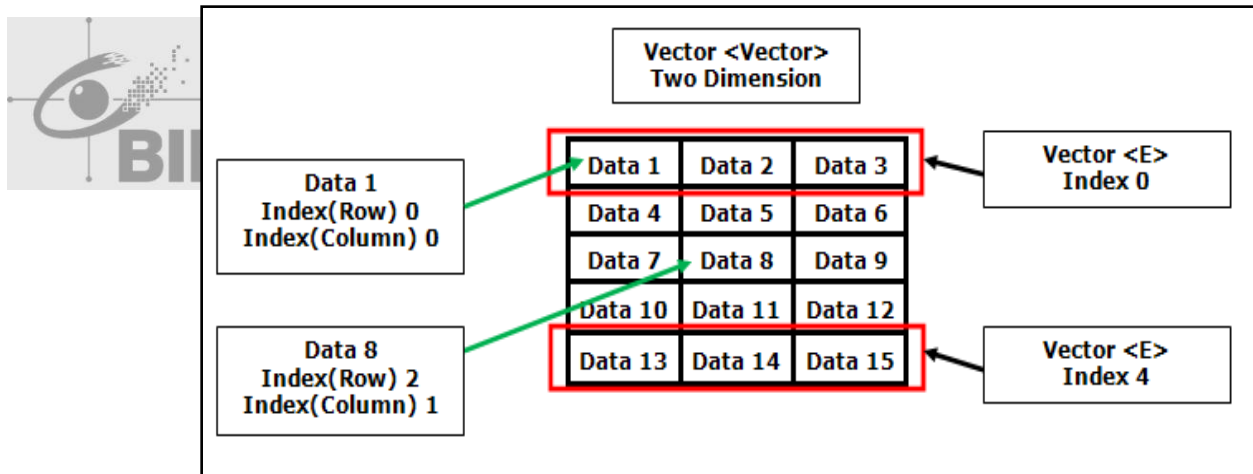
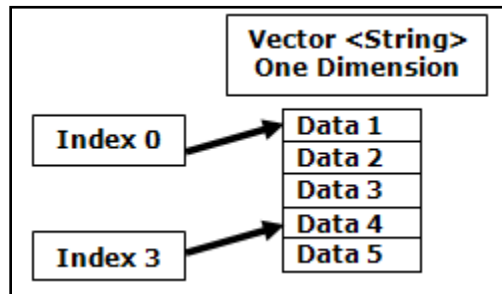
Create object Vector (String data type) named 'vec_header' and using default Constructor to constructs an empty vector. It will be used as parameter when creating DefaultTableModel and it will be as a container of header table.

```
Vector<Vector> vec_data = new Vector<Vector>();
```

Explanation:

Create object Vector (Vector data type) named 'vec_data' and using default Constructor to constructs an empty vector. It will be used as parameter when creating DefaultTableModel and it will be as a container of data table.

Different Vector<String> and Vector<Vector> will be explained picture below



```
Vector<String> vec_detail;
```

Explanation:

Just declare an object of Class Vector type String name 'vec_detail'

```

DefaultTableModel dtm = new DefaultTableModel(vec_data, vec_header)
{
    @Override
    public boolean isCellEditable(int row, int column) {
        return false;
    }
};

```

```

DefaultTableModel dtm = new DefaultTableModel(vec_data, vec_header)

```

Explanation:

Create object DefaultTableModel named 'dtm' and using Constructor [DefaultTableModel\(Vector data, Vector columnNames\)](#) to Constructs a DefaultTableModel and initializes the table by passing data and columnNames to the setDataVector method. Or the other word is creates content of JTable using DefaultTableModel.

Fill parameter Vector data (Content Table) with object 'vec_data' and Vector columnNames (Header Table) with object 'vec_header'

```

@Override
public boolean isCellEditable(int row, int column) {
    return false;
}

```

Explanation:

Method [isCellEditable\(int row, int column\)](#) is a method from DefaultTableModel that used to sets cell of table whether or not to edit.

3. Make table with data

```

vec_header.add("Name");
vec_header.add("Gender");
vec_header.add("Age");

for (int i = 0; i < name.length; i++) {
    vec_detail = new Vector<String>();
    vec_detail.add(name[i]);
    vec_detail.add(gender[i]);
    vec_detail.add(age[i]);

    vec_data.add(vec_detail);
}

tabel.setModel(dtm);
tabel.setPreferredSize(new Dimension(300, 150));
spaneTabel.setPreferredSize(new Dimension(400, 150));
spaneTabel.setViewportView(tabel);

```

```

vec_header.add("Name");
vec_header.add("Gender");
vec_header.add("Age");

```

Explanation:

Add objects into 'vec_header' as the header of table

Name	Gender	Age
Amelia	Female	20
Holmes	Male	25

```

for (int i = 0; i < name.length; i++) {
    vec_detail = new Vector<String>();
    vec_detail.add(name[i]);
    vec_detail.add(gender[i]);
    vec_detail.add(age[i]);

    vec_data.add(vec_detail);
}

```

Explanation:

```

for (int i = 0; i < name.length; i++) {

```

Do looping from first index (0) until last index (name.length) to add data from object array into each of vector detail.

```
vec_detail = new Vector<String>();
```

Constructs an empty vector

```
vec_detail.add(name[i]);
vec_detail.add(gender[i]);
vec_detail.add(age[i]);
```

Add data from each array into vector detail

```
vec_data.add(vec_detail);
```

Add object 'vec_detail' (One dimension Vector) into 'vec_data' (Two Dimension Vector)

```
tabel.setModel(dtm);
```

Explanation:

Method [setModel\(TableModel dataModel\)](#) is a method from `JTable` that used to sets the data model for this table to `newModel` and registers with it for listener notifications from the new data model. Fill parameter with Object `DefaultTableModel` named 'dtm'

```
tabel.setPreferredSize(new Dimension(300, 150));
```

```
spaneTabel.setPreferredSize(new Dimension(400, 150));
```

Explanation:

Method `setPreferredSize(Dimension preferredSize)` is an inherited method from class `javax.swing.JComponent` that used to sets the preferred size of this component.

```
spaneTabel.setViewportView(tabel);
```

Explanation:

Method [setViewportView\(Component view\)](#) is a method JScrollPane that used to creates a viewport if necessary and then sets its view. This setting properties will make object 'tabel' has scrollbar if the row of table is not enough.

h. Task 08 – Add Component into each JPanel

1. Add all components that located in the Center of Container into 'panel_center'

```
panel_center.add(speneList);
```

Explanation:

Add object 'spenelist' into 'panel_center'. Object 'list' is include in the object 'speneList'.

2. Add all component that located in the South of Container into 'panel_south'

```
panel_south.add(speneTabel);
```

Explanation:

Add object 'spenetable' into 'panel_south'. Object 'tabel' is include in the object 'speneTabel'.

i. Task 09 – Add JPanel into Container and give direction

```
c.add(panel_center, "Center");
c.add(panel_south, "South");
```

Explanation:

This section is final step. We will add each panel into Container based on border layout, so we need submit direction too when we add component into Container. Add object 'panel_center' into Center Direction. Second then add 'panel_south' into South Direction.

Chapter 06

Event Driven Programming



6.1 Event Handling

EventHandling is a method to handle an event/action given user to a GUI component. Event can be defined as a signal to the program that something has happened. Event can be triggered by a user on a component, such as the button is pressed. Two packages are commonly used to handle events are `java.swing.event` and `java.awt.event`. Eventhandling can be divided into 3groups: **EventSource**, **EventListener**, and **EventHandler**.

1. Event Source

EventSource is a component that getting the event which is then captured by the EventListener. EventSource can be distinguished by name of the component itself, such as the Save button, delete button, and others. By naming this Event Source, an Event Listener will be able to detect the source which it originates.

2. Event Listener

EventListener is used to capture events that occur in components.

3. Event Handler

EventHandler is a method contains blocks of program that determines the next process after the component get an event. For example, when Save button is pressed, the Event Listener will catch the event from the Source, then the Event Handler will store the data.

6.2 Action Listener

The listener interface for receiving action events. The class that might be processing an action event implements this interface, and the object created within that class is registered with an **addActionListener** method. When the action event occurs, that object's `actionPerformed` method is invoked.

Method Summary	
<code>void</code>	<code>actionPerformed(ActionEvent e)</code> Invoked when an action occurs.

6.3 Key Listener

The listener interface for receiving keyboard events (keystrokes). The class that is interested in processing a keyboard event either implements this interface (and all the

methods it contains) or extends the abstract `KeyAdapter` class (overriding only the methods that will be used).

The listener object created from that class is then registered with a component using the component's `addKeyListener` method. A keyboard event is generated when a key is pressed, released, or typed. The relevant method in the listener object is then invoked, and the `KeyEvent` is passed to it.

Method Summary	
void	<code>keyPressed(KeyEvent e)</code> Invoked when a key has been pressed.
void	<code>keyReleased(KeyEvent e)</code> Invoked when a key has been released.
void	<code>keyTyped(KeyEvent e)</code> Invoked when a key has been typed.

6.4 Mouse Listener

The listener interface for receiving mouse events (press, release, click, enter, and exit) on a component.



Method Summary	
void	<code>mouseClicked(MouseEvent e)</code> Invoked when the mouse button has been clicked (pressed and released) on a component.
void	<code>mouseEntered(MouseEvent e)</code> Invoked when the mouse enters a component.
void	<code>mouseExited(MouseEvent e)</code> Invoked when the mouse exits a component.
void	<code>mousePressed(MouseEvent e)</code> Invoked when a mouse button has been pressed on a component.
void	<code>mouseReleased(MouseEvent e)</code> Invoked when a mouse button has been released on a component.

6.5 Window Listener

The listener interface for receiving window events.

Method Summary	
void	windowActivated (WindowEvent e) Invoked when the Window is set to be the active Window.
void	windowClosed (WindowEvent e) Invoked when a window has been closed as the result of calling dispose on the window.
void	windowClosing (WindowEvent e) Invoked when the user attempts to close the window from the window's system menu.
void	windowDeactivated (WindowEvent e) Invoked when a Window is no longer the active Window.
void	windowDeiconified (WindowEvent e) Invoked when a window is changed from a minimized to a normal state.
void	windowIconified (WindowEvent e) Invoked when a window is changed from a normal to a minimized state.
void	windowOpened (WindowEvent e) Invoked the first time a window is made visible.

6.6 Item Listener

The listener interface for receiving item events.

Method Summary	
void	itemStateChanged (ItemEvent e) Invoked when an item has been selected or deselected by the user.

6.7 JOptionPane

JOptionPane makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something.

While the JOptionPane class may appear complex because of the large number of methods, almost all uses of this class are one-line calls to one of the static methods shown below:

Method Name	Description
showConfirmDialog	Asks a confirming question, like yes/no/cancel.
showInputDialog	Prompt for some input.
showMessageDialog	Tell the user about something that has happened.
showOptionDialog	The Grand Unification of the above three.

6.8 Exercise

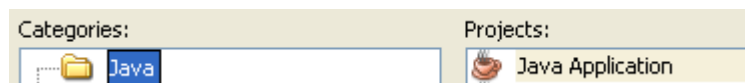
Create Form Register

a. Task 01 - Create Project JAVA in Netbeans

1. Run Netbeans from Start Menu

2. Open Menu File -> New Project or Click icon  on bottom menu Edit

3. On Project Window, choose Java on Categories and Java Application on Projects



4. Next Setting Project name, Project Location and don't forget check "Create Main Class" like picture below

Name and Location

Project Name:

Project Location:

Project Folder:

Use Dedicated Folder for Storing Libraries

Libraries Folder:

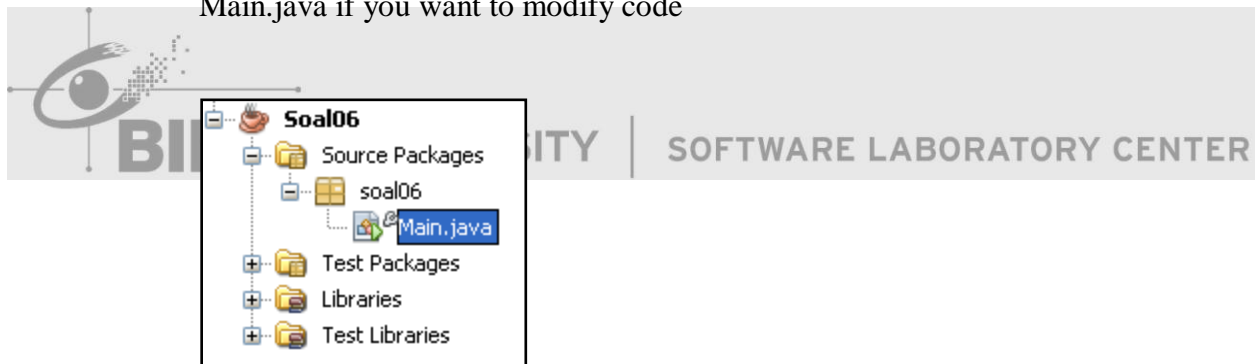
Different users and projects can share the same compilation libraries (see Help for details).

Create Main Class

Set as Main Project

5. Then Press Finish

6. Finally your project has been created like picture below, double click on file Main.java if you want to modify code



b. Task 02 - Create JFrame and Set Properties JFrame

1. Create object JFrame

```
JFrame frame = new JFrame();
```

2. Setting properties JFrame

```
frame.setTitle("GUI Basic Component");
frame.pack();
frame.setLocationRelativeTo(null);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```

c. Task 03 – Using Container Class

1. Create object Container

```
Container c = frame.getContentPane();
```

2. Set layout of Container to BorderLayout

```
c.setLayout(new BorderLayout());
```

Explanation:

Sets the layout manager of Container into Border Layout

d. Task 04 - Create JPanel

1. Create object JPanel and set Layout

```
JPanel panel_north = new JPanel(new BorderLayout());
```

Explanation:

Make object JPanel named 'panel_north' that will be located on North Direction of Container. Directly set layout into Border Layout when created object. Only object JLabel (Title) and JLabel (Icon) will be added into 'panel_north'.

```
JPanel panel_center = new JPanel(new GridLayout(4,2));
```

Explanation:

Create object JPanel named 'panel_center' that will be located on Center Direction of Container. Directly set layout into Grid Layout with 4 rows and 2 cols when created object.

```
JPanel panel_south = new JPanel(new FlowLayout());
```

Explanation:

Create object JPanel named 'panel_south' that will be located on South Direction of Container. Directly set layout into Flow layout when created object. Only two JButton will be added into this panel.

```
JPanel panel_radio = new JPanel();
```

Explanation:

Create object JPanel named 'panel_radio' that will be located on right side of Gender. Still make default Layout of JPanel (Flow Layout) when created object. Only two JRadioButton will be added into this panel.

```
JPanel panel_check = new JPanel(new GridLayout(2, 2));
```

Explanation:

Create object JPanel named 'panel_check' that will be located on right side of Hobby. Directly set layout into Grid Layout with 2 rows and 2 cols when created object. Four JCheckBox will be added into this panel.



e. Task 05 - Create Component

1. Create object Component Container direction "North"

```
JLabel lbl_title = new JLabel("Form Registration", JLabel.CENTER);
ImageIcon icon = new ImageIcon("src/registration.png");
JLabel lbl_icon = new JLabel(icon, JLabel.CENTER);
```

```
JLabel lbl_title = new JLabel("Form Registration", JLabel.CENTER);
```

Explanation:

Create object JLabel named 'lbl_title'. JLabel has constructor *JLabel(String text, int horizontalAlignment)*. So we can directly make object JLabel with text and set position horizontal of JLabel. About position horizontal of JLabel, there are several parameter that can be used.

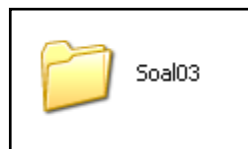
- JLabel.[BOTTOM ALIGNMENT](#)
Ease-of-use constant for `getAlignmentY()`. It specifies an alignment to the bottom of the component
- JLabel.[CENTER ALIGNMENT](#)
Ease-of-use constant for `getAlignmentY()` and `getAlignmentX()`. It specifies an alignment to the center of the component
- JLabel.[LEFT ALIGNMENT](#)
Ease-of-use constant for `getAlignmentX()`. It specifies an alignment to the left side of the component.
- JLabel.[RIGHT ALIGNMENT](#)
Ease-of-use constant for `getAlignmentX()`. It specifies an alignment to the right side of the component.
- JLabel.[TOP ALIGNMENT](#)
Ease-of-use constant for `getAlignmentY()`. It specifies an alignment to the top of the component.



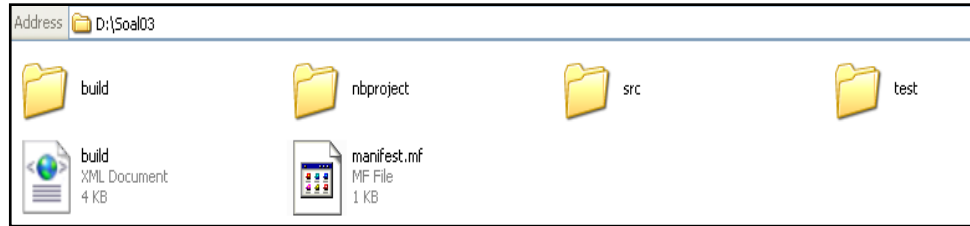
Explanation:

Create object ImageIcon named 'icon'. You can import directly file that have picture format such as .jpg, .bmp, .png, .gif and etc with explain the location of file in parameter. This section will be explained step by step about directory file.

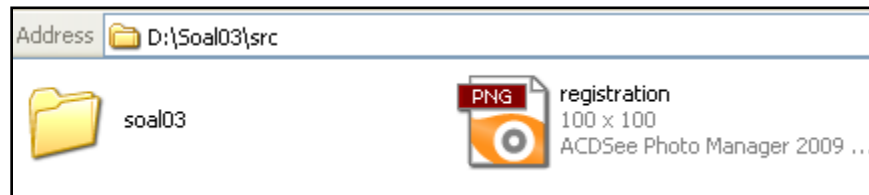
- First, you have project folder that named by "Soal03" like picture below



- Second, Double click the folder and you will get the location like picture. This is "default first directory" if you want to import the file.



- Third, in case, picture file named “[registration.png](#)” has been located in folder “[src](#)”. So it must be informed the location of file is “[scr/registration.png](#)”. Don’t give space or make capital letter when name of file is not capital letter because it’s case sensitive.



```
JLabel lbl_icon = new JLabel(icon, JLabel.CENTER);
```

Explanation :

Create object JLabel named ‘`lbl_icon`’. JLabel has constructor `JLabel(Icon image, int horizontalAlignment)`. So we can directly make object JLabel with icon and set position horizontal of JLabel. Fill parameter `Icon image` with object ImageIcon named `icon`.

2. Create object Component Container direction “`Center`” (`panel_center`)

```
JLabel lbl_name = new JLabel("Name (Letter)");
JLabel lbl_age = new JLabel("Age (Numeric)");
JLabel lbl_gender = new JLabel("Gender");
JLabel lbl_hobby = new JLabel("Hobby");

JTextField txt_name = new JTextField();
JTextField txt_age = new JTextField();
JRadioButton radio_male = new JRadioButton("Male");
JRadioButton radio_female = new JRadioButton("Female");
JCheckBox chk_listen = new JCheckBox("Listen Music");
JCheckBox chk_watch = new JCheckBox("Watching Movie");
JCheckBox chk_coding = new JCheckBox("Coding");
JCheckBox chk_others = new JCheckBox("Others");

ButtonGroup btn_group = new ButtonGroup();
```

```
JLabel lbl_name = new JLabel("Name (Letter)");
JLabel lbl_age = new JLabel("Age (Numeric)");
JLabel lbl_gender = new JLabel("Gender");
JLabel lbl_hobby = new JLabel("Hobby");
```

Explanation:

Create all objects JLabel like picture. JLabel has constructor [JLabel\(String text\)](#). So we can directly make object JLabel with text.

```
JTextField txt_name = new JTextField();
JTextField txt_age = new JTextField();
```

Explanation:

Create object JTextField named 'txt_name' and 'txt_age'. Use default constructor of JLabel with empty parameter.

```
JRadioButton radio_male = new JRadioButton("Male");
JRadioButton radio_female = new JRadioButton("Female");
```

Explanation:

Create object JRadioButton named 'radio_male' and 'radio_female'. JRadioButton has constructor [JRadioButton\(String text\)](#). So we can directly make object JRadioButton with text. **JRadioButton always combined with ButtonGroup** because ButtonGroup can create a group of buttons in which only one button at a time can be selected.

```
ButtonGroup btn_group = new ButtonGroup();
```

Explanation:

Create object ButtonGroup named 'btn_group'. ButtonGroup only has one default constructor [ButtonGroup\(\)](#).

```
JCheckBox chk_listen = new JCheckBox("Listen Music");
JCheckBox chk_watch = new JCheckBox("Watching Movie");
JCheckBox chk_coding = new JCheckBox("Coding");
JCheckBox chk_others = new JCheckBox("Others");
```

Explanation:

Create all objects JCheckBox like picture. JCheckBox has constructor [JCheckBox\(String text\)](#). So we can directly create object JCheckBox with text.

3. Create object Component Container direction “South” (`panel_south`)

```
JButton btn_submit = new JButton("Submit");
JButton btn_reset = new JButton("Reset");
```

Explanation:

Create object JButton named ‘`btn_submit`’ and ‘`btn_reset`’. JButton has constructor [JButton\(String text\)](#). So we can directly create object JButton with text.

f. Task 06 – Add Component into each JPanel

1. Add all components that located in the North of Container into ‘`panel_north`’

```
panel_north.add(lbl_title, "North");
panel_north.add(lbl_icon, "Center");
```

Explanation:

First add object ‘`lbl_title`’ into ‘`panel_north`’ with direction “North”. Second, add object ‘`lbl_icon`’ into ‘`panel_north`’ with direction “Center”. We add direction in second parameter because ‘`panel_north`’ use BorderLayout.

2. Add all components that located in the Center of Container into 'panel_center'

```
panel_center.add(lbl_name);
panel_center.add(txt_name);

panel_center.add(lbl_age);
panel_center.add(txt_age);

btn_group.add(radio_male);
btn_group.add(radio_female);
panel_radio.add(radio_male);
panel_radio.add(radio_female);
panel_center.add(lbl_gender);
panel_center.add(panel_radio);

panel_check.add(chk_watch);
panel_check.add(chk_listen);
panel_check.add(chk_coding);
panel_check.add(chk_others);
panel_center.add(lbl_hobby);
panel_center.add(panel_check);
```

Explanation:

Because 'panel_center' use GridLayout, therefore when we add each component into 'panel_center', we must add each component one by one according the order.

```
panel_center.add(lbl_name);
panel_center.add(txt_name);
```

Explanation:

According the order, add object lbl_name and txt_name into panel_center

```
panel_center.add(lbl_age);
panel_center.add(txt_age);
```

Explanation:

According the order, add object lbl_age and txt_age into panel_center


```
btn_group.add(radio_male);
btn_group.add(radio_female);
panel_radio.add(radio_male);
panel_radio.add(radio_female);
```

Explanation:

Add object radio_male and radio female into btn_group first, then add radio_male and radio_female into panel_radio. Object btn_group in this section just created a group of buttons. Object panel_radio in this section is to create a button panel.

```
panel_center.add(lbl_gender);
panel_center.add(panel_radio);
```

Explanation:

According the order, add object lbl_gender and panel_radio into panel_center

```
panel_check.add(chk_watch);
panel_check.add(chk_listen);
panel_check.add(chk_coding);
panel_check.add(chk_others);
```

Explanation:

Same as JRadioButton, add all JCheckBox into panel_check to make a button panel.

```
panel_center.add(lbl_hobby);
panel_center.add(panel_check);
```

Explanation:

According the order, add object lbl_hobby and panel_check into panel_center

3. Add all component that located in the South of Container into 'panel_south'

```
panel_south.add(btn_submit);
panel_south.add(btn_reset);
```

Explanation:

A object btn_submit and btn_reset into panel_south

- g. Task 07 – Add JPanel into Container and give direction

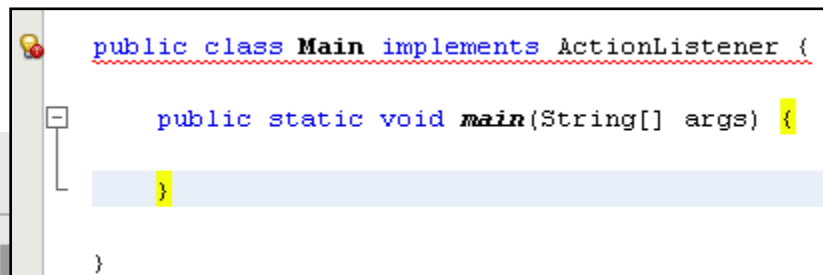
```
c.add(panel_north, "North");
c.add(panel_center, "Center");
c.add(panel_south, "South");
```

Explanation:

This section is final step. We will add each panel into Container based on border layout, so we need submit direction too when we add component into Container. At first, we add object 'panel_north' into North Direction. Second, add 'panel_center' into Center Direction, and last add 'panel_south' into South Direction.

- h. Task 08 – Make event ActionListener & implemented it to JButton

1. Implements Interface ActionListener



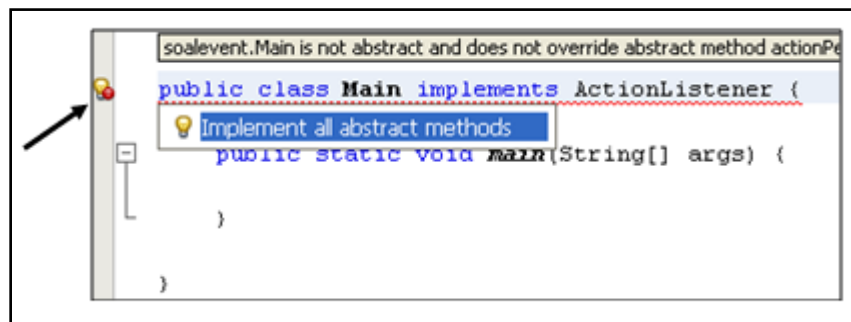
```
public class Main implements ActionListener {

    public static void main(String[] args) {

    }

}
```

2. Click yellow lamp on left side to implement all methods



```
soalevent.Main is not abstract and does not override abstract method actionPerformed() in ActionListener
public class Main implements ActionListener {
    Implement all abstract methods
    public static void main(String[] args) {

    }

}
```

3. Method from Interface ActionListener will be added automatically

```
public void actionPerformed(ActionEvent e) {
    throw new UnsupportedOperationException("Not supported yet.");
}
```

4. Erase all statement in method actionPerformed

```
public void actionPerformed(ActionEvent e) {
}

```

5. Add ActionListener to object btn_submit and btn_reset

```
btn_submit.addActionListener(this);
btn_reset.addActionListener(this);

```

Explanation:

addActionListener([ActionListener l](#))

➔ This function will adds an ActionListener to the button.

6. Check validation on btn_submit

```
public void actionPerformed(ActionEvent e) {
    if(e.getSource() == btn_submit)
    {
        if(txt_name.getText().equals(""))
        {
            JOptionPane.showMessageDialog(frame.getContentPane(), "Fill name!", "Attention",
            JOptionPane.ERROR_MESSAGE);
        }
        else if(txt_age.getText().equals(""))
        {
            JOptionPane.showMessageDialog(frame.getContentPane(), "Fill age!", "Attention",
            JOptionPane.ERROR_MESSAGE);
        }
        else if (radio_male.getSelectedObjects() == null && radio_female.getSelectedObjects() == null)
        {
            JOptionPane.showMessageDialog(frame.getContentPane(), "Choose gender!", "Attention",
            JOptionPane.ERROR_MESSAGE);
        }
        else if (chk_coding.getSelectedObjects() == null && chk_listen.getSelectedObjects() == null
        && chk_others.getSelectedObjects() == null && chk_watch.getSelectedObjects() == null)
        {
            JOptionPane.showMessageDialog(frame.getContentPane(), "Choose Hobby!", "Attention",
            JOptionPane.ERROR_MESSAGE);
        }
    }
}

```

```
if(e.getSource() == btn_submit)
{
}

```

Explanation:

[Object getSource\(\)](#)

➔ This function will return object on which the Event initially occurred.

```

if(txt_name.getText().equals(""))
{
    JOptionPane.showMessageDialog(frame.getContentPane(), "Fill name!", "Attention",
    JOptionPane.ERROR_MESSAGE);
}

```

Explanation:

If the object txt_name don't have text, then print alert "Fill name"

```

else if(txt_age.getText().equals(""))
{
    JOptionPane.showMessageDialog(frame.getContentPane(), "Fill age!", "Attention",
    JOptionPane.ERROR_MESSAGE);
}

```

Explanation:

If the object txt_age don't have text, then print alert "Fill age"

```

else if (radio_male.getSelectedObjects() == null && radio_female.getSelectedObjects() == null)
{
    JOptionPane.showMessageDialog(frame.getContentPane(), "Choose gender!", "Attention",
    JOptionPane.ERROR_MESSAGE);
}

```

Explanation:

If the object radio_male and radio_female haven't been chosen, then print alert "Choose gender"

```

else if (chk_coding.getSelectedObjects() == null && chk_listen.getSelectedObjects() == null
&& chk_others.getSelectedObjects() == null && chk_watch.getSelectedObjects() == null)
{
    JOptionPane.showMessageDialog(frame.getContentPane(), "Choose Hobby!", "Attention",
    JOptionPane.ERROR_MESSAGE);
}

```

Explanation:

If the object chk_coding, chk_listen, chk_other and chk_watch haven't been chosen, then print alert "Choose hobby"

```

else if(e.getSource() == btn_reset)
{
    txt_name.setText("");
    txt_age.setText("");
    btn_group.clearSelection();
    chk_coding.setSelected(false);
    chk_listen.setSelected(false);
    chk_others.setSelected(false);
    chk_watch.setSelected(false);
}

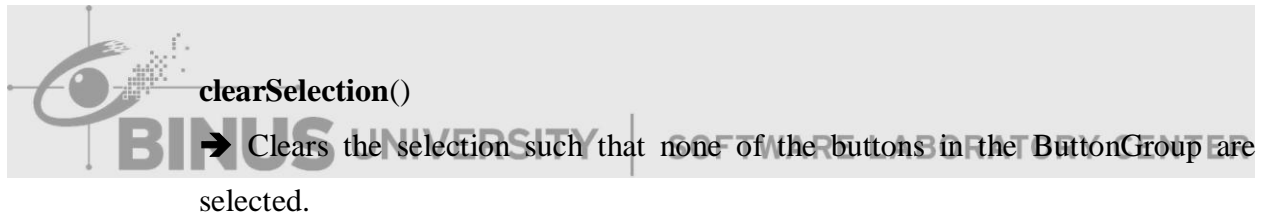
```

Explanation:**Object** `getSource()`

➔ This function will return object on which the Event initially occurred.

setText() (`String` text)

➔ Defines the single line of text this component will display. If the value of text is null or empty string, nothing is displayed.



clearSelection()

➔ Clears the selection such that none of the buttons in the ButtonGroup are selected.

setSelected() (boolean b)

➔ Selects or deselects the button.

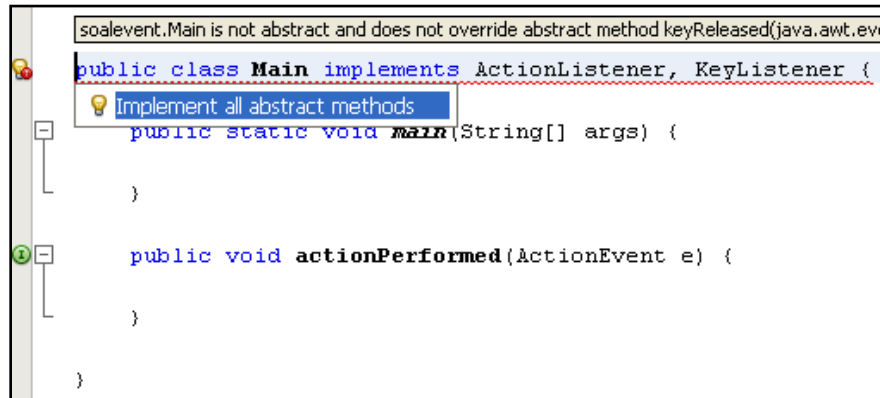
- i. Task 09 – Make event KeyListener & implemented it to JTextField
 1. Implements Interface KeyListener

```

public class Main implements ActionListener, KeyListener {
    public static void main(String[] args) {
    }
    public void actionPerformed(ActionEvent e) {
    }
}

```

2. Click yellow lamp on left side to implement all methods



```

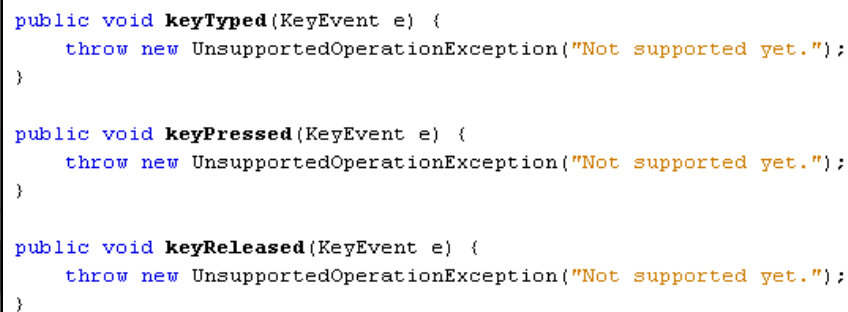
soalevent.Main is not abstract and does not override abstract method keyReleased(java.awt.ev
public class Main implements ActionListener, KeyListener {
    Implement all abstract methods
    public static void main(String[] args) {
        }

    public void actionPerformed(ActionEvent e) {
        }

}

```

3. Method from Interface KeyListener will be added automatically



```

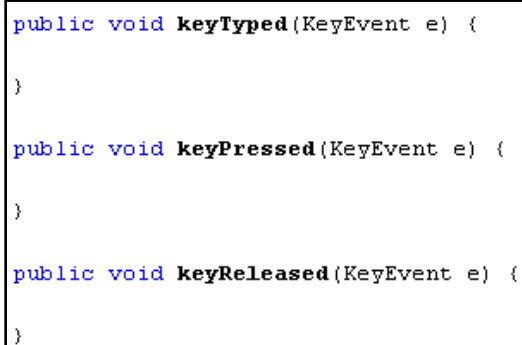
public void keyTyped(KeyEvent e) {
    throw new UnsupportedOperationException("Not supported yet.");
}

public void keyPressed(KeyEvent e) {
    throw new UnsupportedOperationException("Not supported yet.");
}

public void keyReleased(KeyEvent e) {
    throw new UnsupportedOperationException("Not supported yet.");
}

```

4. Erase all statement in all method



```

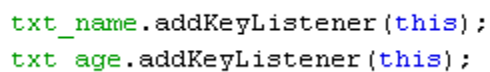
public void keyTyped(KeyEvent e) {
}

public void keyPressed(KeyEvent e) {
}

public void keyReleased(KeyEvent e) {
}

```

5. Add KeyListener to object txt_name and txt_age



```

txt_name.addKeyListener(this);
txt_age.addKeyListener(this);

```

addKeyListener([KeyListener](#) l)

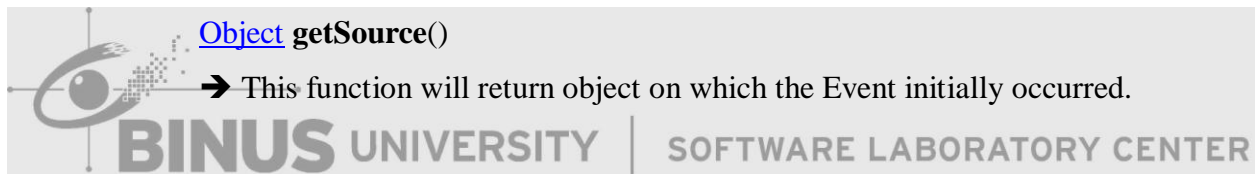
→ this function will adds an ActionListener to the button.

6. Create validation in object txt_name

```
if(e.getSource() == txt_name)
{
    if((e.getKeyChar() < 'a' || e.getKeyChar() > 'z') && (e.getKeyChar() < 'A'
        || e.getKeyChar() > 'Z') && e.getKeyChar() != e.VK_BACK_SPACE
        && e.getKeyChar() != e.VK_SPACE)
    {
        JOptionPane.showMessageDialog(frame.getContentPane(), "Input must be Letter", "Alert",
            JOptionPane.ERROR_MESSAGE);
        e.setKeyChar('\0');
    }
}
```

Explanation:

If the input doesn't between 'a-z' or 'A-Z' or BACK_SPACE or SPACE, then print JOptionPane "Input must be letter"



[Object](#) **getSource()**

→ This function will return object on which the Event initially occurred.

Char **getKeyChar()**

→ Returns the character associated with the key in this event.

setKeyChar(char keyChar)

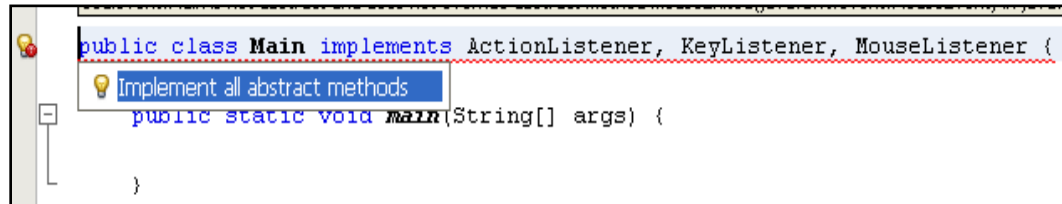
→ Set the keyChar value to indicate a logical character.

j. Task 10 – Make event MouseListener & implemented it to JCheckBox

1. Implements Interface KeyListener

```
public class Main implements ActionListener, KeyListener, MouseListener {
    public static void main(String[] args) {
    }
}
```

2. Click yellow lamp on left side to implement all methods



```
public class Main implements ActionListener, KeyListener, MouseListener {  
    public static void main(String[] args) {  
    }  
}
```

3. Method from Interface MouseListener will be added automatically

```
public void mouseClicked(MouseEvent e) {  
    throw new UnsupportedOperationException("Not supported yet.");  
}  
  
public void mousePressed(MouseEvent e) {  
    throw new UnsupportedOperationException("Not supported yet.");  
}  
  
public void mouseReleased(MouseEvent e) {  
    throw new UnsupportedOperationException("Not supported yet.");  
}  
  
public void mouseEntered(MouseEvent e) {  
    throw new UnsupportedOperationException("Not supported yet.");  
}  
  
public void mouseExited(MouseEvent e) {  
    throw new UnsupportedOperationException("Not supported yet.");  
}
```

4. Erase all statement in all method

```
public void mouseClicked(MouseEvent e) {  
}  
  
public void mousePressed(MouseEvent e) {  
}  
  
public void mouseReleased(MouseEvent e) {  
}  
  
public void mouseEntered(MouseEvent e) {  
}  
  
public void mouseExited(MouseEvent e) {  
}
```


5. Add MouseListener to all object JCheckBox

```
chk_watch.addMouseListener(this);
chk_listen.addMouseListener(this);
chk_coding.addMouseListener(this);
chk_others.addMouseListener(this);
```

addMouseListener([MouseListener l](#))

➔ Adds the specified mouse listener to receive mouse events from this component.

6. Create event when click mouse in JcheckBox

```
if(e.getSource() == chk_watch)
{
    JOptionPane.showMessageDialog(frame.getContentPane(), "Watching Movie");
}
else if(e.getSource() == chk_listen)
{
    JOptionPane.showMessageDialog(frame.getContentPane(), "Listen Music");
}
else if(e.getSource() == chk_coding)
{
    JOptionPane.showMessageDialog(frame.getContentPane(), "Coding");
}
else if(e.getSource() == chk_others)
{
    JOptionPane.showMessageDialog(frame.getContentPane(), "Others");
}
```

k. Task 11 – Make event WindowListener & implemented it to JFrame

1. Implements Interface WindowListener

```
public class Main implements WindowListener, ActionListener, KeyListener, MouseListener {
```

2. Click yellow lamp on left side to implement all methods

```
soalevent.Main is not abstract and does not override abstract method windowDeactivated(java.awt.event.WindowEvent) in java.awt.event.WindowList
```

```
public class Main implements WindowListener, ActionListener, KeyListener, MouseListener {
```

Implement all abstract methods

- Method from Interface WindowListener will be added automatically

```

public void windowOpened(WindowEvent e) {
    throw new UnsupportedOperationException("Not supported yet.");
}

public void windowClosing(WindowEvent e) {
    throw new UnsupportedOperationException("Not supported yet.");
}

public void windowClosed(WindowEvent e) {
    throw new UnsupportedOperationException("Not supported yet.");
}

public void windowIconified(WindowEvent e) {
    throw new UnsupportedOperationException("Not supported yet.");
}

public void windowDeiconified(WindowEvent e) {
    throw new UnsupportedOperationException("Not supported yet.");
}

public void windowActivated(WindowEvent e) {
    throw new UnsupportedOperationException("Not supported yet.");
}

public void windowDeactivated(WindowEvent e) {
    throw new UnsupportedOperationException("Not supported yet.");
}

```

- Erase all statement in all method
- Add WindowListener to object JFrame named frame

```
frame.addWindowListener(this);
```

addWindowListener([WindowListener](#) l)

➔ Adds the specified window listener to receive window events from this window.

- Make event when Windows JFrame has been closed

```

public void windowClosing(WindowEvent e) {
    JOptionPane.showMessageDialog(frame.getContentPane(), "Goodbye Friend :)", "Welcome",
    JOptionPane.INFORMATION_MESSAGE);
}

```

1. Task 10 – Make event ItemListener & implemented it to JRadioButton

1. Implements Interface ItemListener

```
public class Main implements ItemListener, WindowListener, ActionListener,
```

2. Click yellow lamp on left side to implement all methods

```
soalevent.Main is not abstract and does not override abstract method itemStateChanged(java.awt.event.ItemEvent) in ja
public class Main implements ItemListener, WindowListener, ActionListener,
Implement all abstract methods
```

3. Method from Interface ItemListener will be added automatically

```
public void itemStateChanged(ItemEvent e) {
    throw new UnsupportedOperationException("Not supported yet.");
}
```

4. Erase all statement in all method

```
public void itemStateChanged(ItemEvent e) {
}
```

5. Add ItemListener to object JRadioButton

```
radio_male.addItemListener(this);
radio_female.addItemListener(this);
```

addItemListener([ItemListener](#) l)

→ Adds an ItemListener to the checkbox.

6. Make event when object `radio_male` and `radio_female` has been chosen or not

```
public void itemStateChanged(ItemEvent e) {  
    if(e.getSource() == radio_male)  
    {  
        JOptionPane.showMessageDialog(frame.getContentPane(), "Male");  
    }  
    else if(e.getSource() == radio_female)  
    {  
        JOptionPane.showMessageDialog(frame.getContentPane(), "Female");  
    }  
}
```

Chapter 07

Database Access



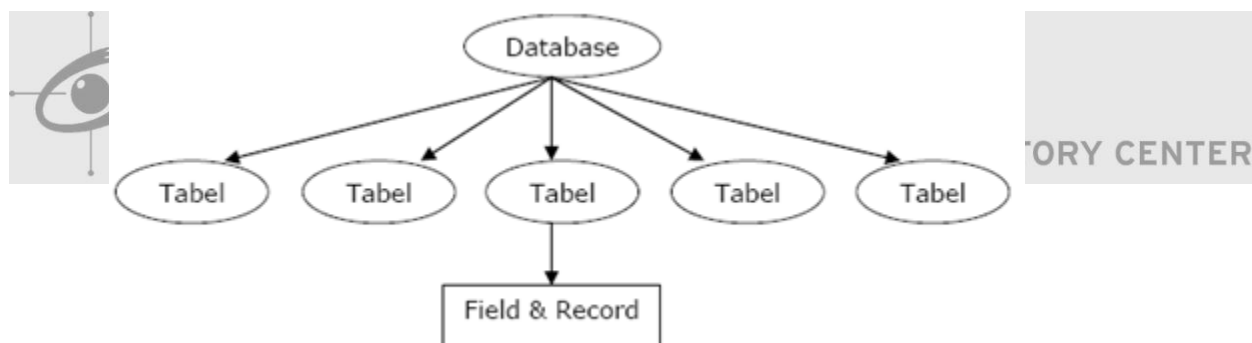
7.1 Introduction to Database

A database is a collection of information that is organized so that it can easily be accessed, managed, and updated. In one view, databases can be classified according to types of content: bibliographic, full-text, numeric, and images.

In computing, databases are sometimes classified according to their organizational approach. The most prevalent approach is the relational database, a tabular database in which data is defined so that it can be reorganized and accessed in a number of different ways. A distributed database is one that can be dispersed or replicated among different points in a network. An object-oriented programming database is one that is congruent with the data defined in object classes and subclasses.

Computer databases typically contain aggregations of data records or files, such as sales transactions, product catalogs and inventories, and customer profiles.

Here is the hierarchy of the database:



7.2 Introduction to JDBC

JDBC API is a Java API that can access any data tabulation, especially data stored in Relational Database. JDBC helps us to create Java programs that broadly cover the following three programming activities:

1. Connect to a data source, such as databases.
2. Send queries and update statements to the database. Doing the command query (SELECT) and command updates (Insert, Update, and Delete) to the database.
3. Receive and process the results from the database response to a query that we make.

JDBC is divided into four essential components.

1. JDBC API

JDBC API provides access from Java programming language to relational database. Using the JDBC API, applications can run SQL commands, receive results, and make the changes back to the data. JDBC API can also interact with multiple data sources contained in a dispersed and diverse environment.

2. JDBC Driver Manager

JDBC Driver Manager class load all the drivers found in the system properly as well as to select the most appropriate driver from opening a connection to a database. Driver Manager has traditionally formed the backbone for the JDBC architecture.

3. JDBC Test Suite

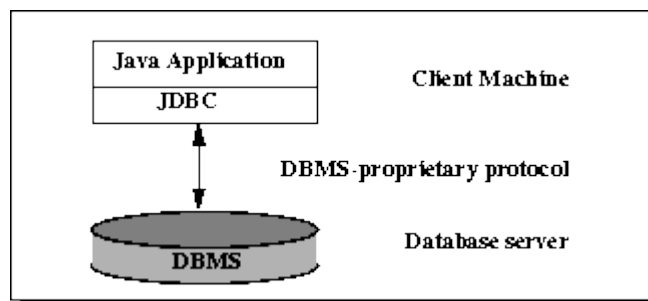
JDBC driver test suite helps us determine whether the JDBC drivers will run the programs that we make or not. This test is not fully comprehensive or complete, but these tests provide important information about many features in the JDBC API.

4. JDBC-ODBC Bridge

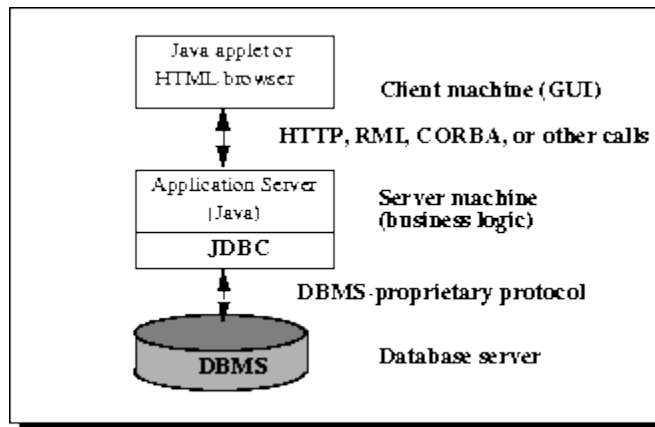
JDBC-ODBC Bridge is a database driver that utilizes the ODBC driver to connect the database. Note worthy is that we have to load ODBC binary code into the client used by the driver.

JDBC Architecture can be seen from his tier level modelling of the Process for accessing the database.

Two tier



Three tier



7.3 Database Connection in Java

Before performing the database operations, the first thing to do is create a connection to the database. To create a connection to the database we need to create a new JDBC (Java Database Connectivity) which can be connected to many sources of data derived from a Java application.

JDBC helps us to create a java application which regulates some activities of programming, among others:

1. Connect to data sources, such as Database
2. Sending query and update data in the Database
3. Provide the results of the database as an answer to the query.

The steps are:

1. Loads its driver class

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

2. Createa database connection by passing the URL connection (in this case database we use is MicrosoftAccess)

```
Connection con = DriverManager.getConnection("jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ="+databaseName+".mdb");
```

3. Createa statement that we want

```
Statement st = con.createStatement(TYPE_SCROLL_INSENSITIVE, CONCUR_UPDATABLE);
```


or

```
Statement st = con.createStatement(1004,1008);
```

Statement is an interface that is used to accommodate queries and st is an object created from the interface Statement. Then create Statement used to receive the query results by the type which can be determined. Then we choose TYPE_SCROLL_INSENSITIVE and CONCUR_UPDATABLE.

TYPE_SCROLL_INSENSITIVE mean that the result of query can be scrollable and CONCUR_UPDATABLE mean query results can be updated.

FieldSummary-java.sql.ResultSet

<u>CLOSE_CURSORS_AT_COMMIT</u>	2	The constant indicating that open ResultSet object with this hold ability will be closed when the current transaction is committed.
<u>CONCUR_READ_ONLY</u>	1007	The constant indicating the concurrency mode for a ResultSet object that may NOT be updated.
<u>CONCUR_UPDATABLE</u>	1008	The constant indicating the concurrency mode for a ResultSet object that may be updated.
<u>FETCH_FORWARD</u>	1000	The constant indicating that the rows in a resultset will be processed in a forward direction;first-to-last.
<u>FETCH_REVERSE</u>	1001	The constant indicating that the rows in a resultset will be processed in a reverse direction; last-to-first.

FETCH_UNKNOWN	1002	The constant indicating that the order in which rows in a resultset will be processed is unknown.
HOLD_CURSORS_OVER_COMMIT	1	The constant indicating that open ResultSet objects with this hold ability will remain open when the current transaction is committed.
TYPE_FORWARD_ONLY	1003	The constant indicating the type for a ResultSet object whose cursor may move only forward.
TYPE_SCROLL_INSENSITIVE	1004	The constant indicating the type for a ResultSet object that is scrollable but generally not sensitive to changes to the data that underlies the ResultSet
TYPE_SCROLL_SENSITIVE	1005	The constant indicating the type for a ResultSet object that is scrollable and generally sensitive to changes to the data that underlies the ResultSet.



4. Create ResultSet to accommodate the query results.

```
ResultSet rs = st.executeQuery("select*fromMasterBarang");
```

This code executes the given SQL statement, which may be an INSERT, UPDATE, or DELETE statement or an SQL statement that returns nothing, such as an SQLDDL statement.

5. Close the connection.

```
con.close();
```

ResultSet

By default, the object will execute the statement and retrieve all the results query then hold it on ResultSet objects. An object from ResultSet, has a cursor that points to the currentline. Tables used:



KodeBarang	NamaBarang	HargaBarang	Click to Add
K001	Pensil	1000	
K002	Pulpen	2000	
K003	Buku	2500	
K004	Penggaris	1500	
K005	Gunting	3000	
K006	Tas	15000	

ResultSet rs:

Record set has several methods to designate data

- ✓ rs.first() → move the cursor to the first data. In the case above, the resultset will appoint the data in row1(K001Pencil1000)
- ✓ rs.next() → move the cursor to the next data. When the first stater will point(K001Pencil1000), after the method rs.next() invoked, then the cursor will point to then extrow(K002pen2000)
- ✓ rs.prev() → move the cursor to the previous data. If the situation now is pointing to the row(K002pen2000) then after method rs.prev() invoked, he would point to the previous row(K001Pencil1000)
- ✓ rs.last() → move the cursor to the latest data. In the case above, the cursor will point the data in row6(K006Bag15000)
- ✓ rs.getString(intcolumnIndex) → Retrieves the value of the designated column in the current row as a String.Index starts from1.

Example:

If the current rs is the first line, then your code is:

```
String code = rs.getString(1);
```

So String code will contain the "K001"

If you write:

```
String name = rs.getString(2);
```

So the name will contain the string "Pencil", it applies to the rest.

7.4 SimpleSQLQuery

After we create the connection then let's make the simple SQL Query. Let's make the select simple query.

SimpleSELECT

```
SELECT[column_name]FROM[table_name]
```

```
SELECT[column_name1],[column_name2],[...]FROM[table_name]
```

```
SELECT*FROM[table_name]
```

Example:

```
SELECT Username FROM MsUser // this query for see column Username from
```

table MsUser

```
SELECT Username, Password FROM MsUser
```

```
SELECT * FROM MsUser // this query for see all the column from a table
```

SELECT Filters

In select filter, you can have rules in your queries. These rules are tested for each row your query produces. If the rule is true, the row is displayed. If the rule is false, the row is not displayed. The rule starts with WHERE.

FORMAT:

```
SELECT [column_name] FROM [table_name] WHERE [condition]
```

Example:

```
SELECT Username FROM MsUser WHERE Username like '%budi%'
```

```
//this query can be used if we want to see the username that contain the word
```

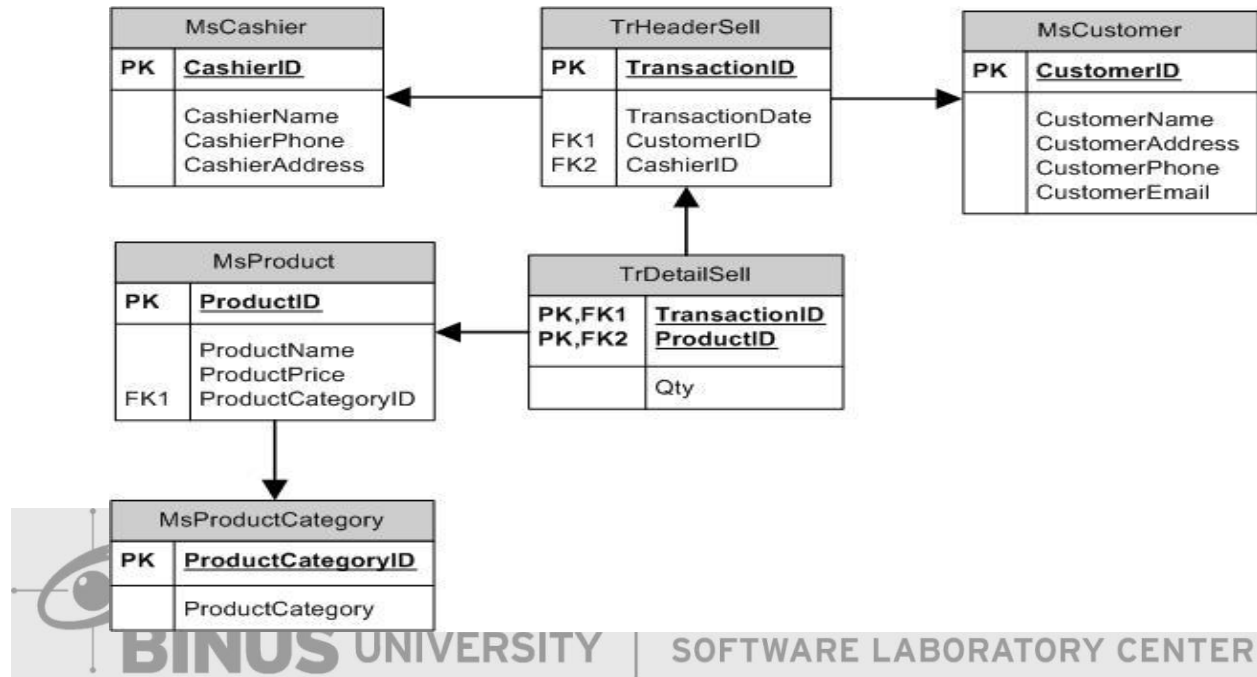
budi.

```
SELECT Username FROM MsUser WHERE Username = ' budi'
```

```
//this query can be used if we want to see the username budi.
```

7.5 ViewData

In view data, we can only use simple select query. But how about if our manager want to see a few table? Is it possible to use simple select query? Therefore we need merging tables. So for example table A combined with table B, it will generate a new virtual table contains the contents of table A and table B.



If the manager wants to see the contents of the table MsProduct Category and MsProduct simultaneously, it is necessary to merge the table. So, this is the query examples:

```
SELECT ProductName, ProductPrice, ProductCategory FROM MsProduct mp,
MsProductCategory pc WHERE mp.ProductID = pc.ProductID.
```

Mp from MsProduct and pc from MsProductCategory is the alias name that is used to combine tables. And ProductID is used as a bridge to combine the tables. So the process of merging table cannot be do near bit rarely. There is a requirement to merge two tables.

7.6 Exercise

Exercise 01 – Make a Database with MicrosoftAccess

➤ Task 01 – Create Blank Database in Microsoft Office 2007

1. Run Microsoft Office 2008 from Start Menu

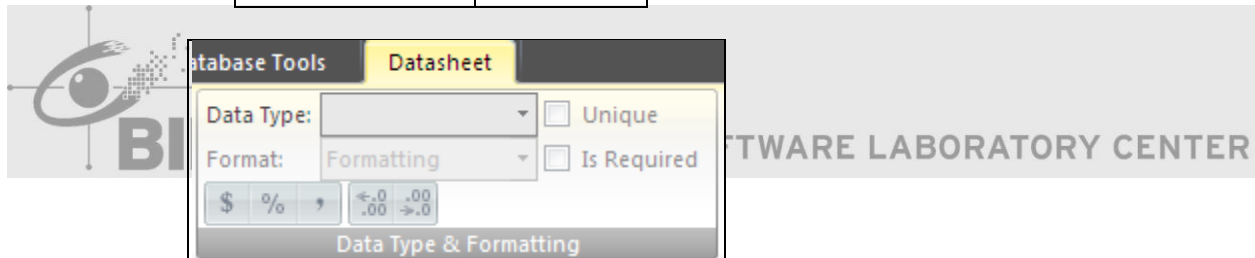
2. Choose Blank Database
3. Then on the right side, Enter database name “database”
4. Browse location of database in D:\
5. Then press Create

➤ Task 02 – Table declaration

1. Fill the header on the word “AddNewField”
2. Then specify the data type on menu Datasheet–Data Type&Formatting–DataType.
You can choose many data type based on ColumnNames.

Example:

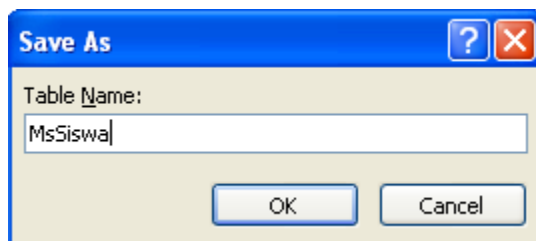
Column Names	Data Type
Name	Text
Dateof Birth	Date/Time
Price	Number



3. Then you can fill the content of the row column

ID	Nama	TanggalLahi	Umur	Add New Field
2	poipoi	11/16/1990	20	
3	yoyo	2/15/1991	19	
*	(New)			

4. After filling the contents of the column line, then save it with CTRL+S or click on the save icon, then Enter the name of your table “MsSiswa”

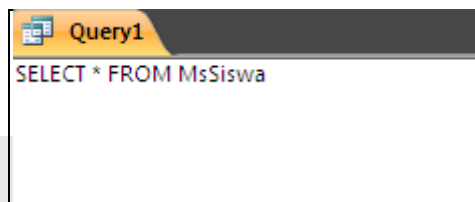


5. This is the place where we can see table what has been made



➤ Task 03 – SQL SimpleQuery

1. Choose Menu select Create-QueryDesign. If there is a dialogbox "ShowTable", then choose close. Then choose SQL menu.
2. Then you can perform a simple query in here



3. After you create this query then you can run you show the result of your query. Choose Run on the menu

The screenshot shows a window titled 'Query1' displaying a table with the following data:

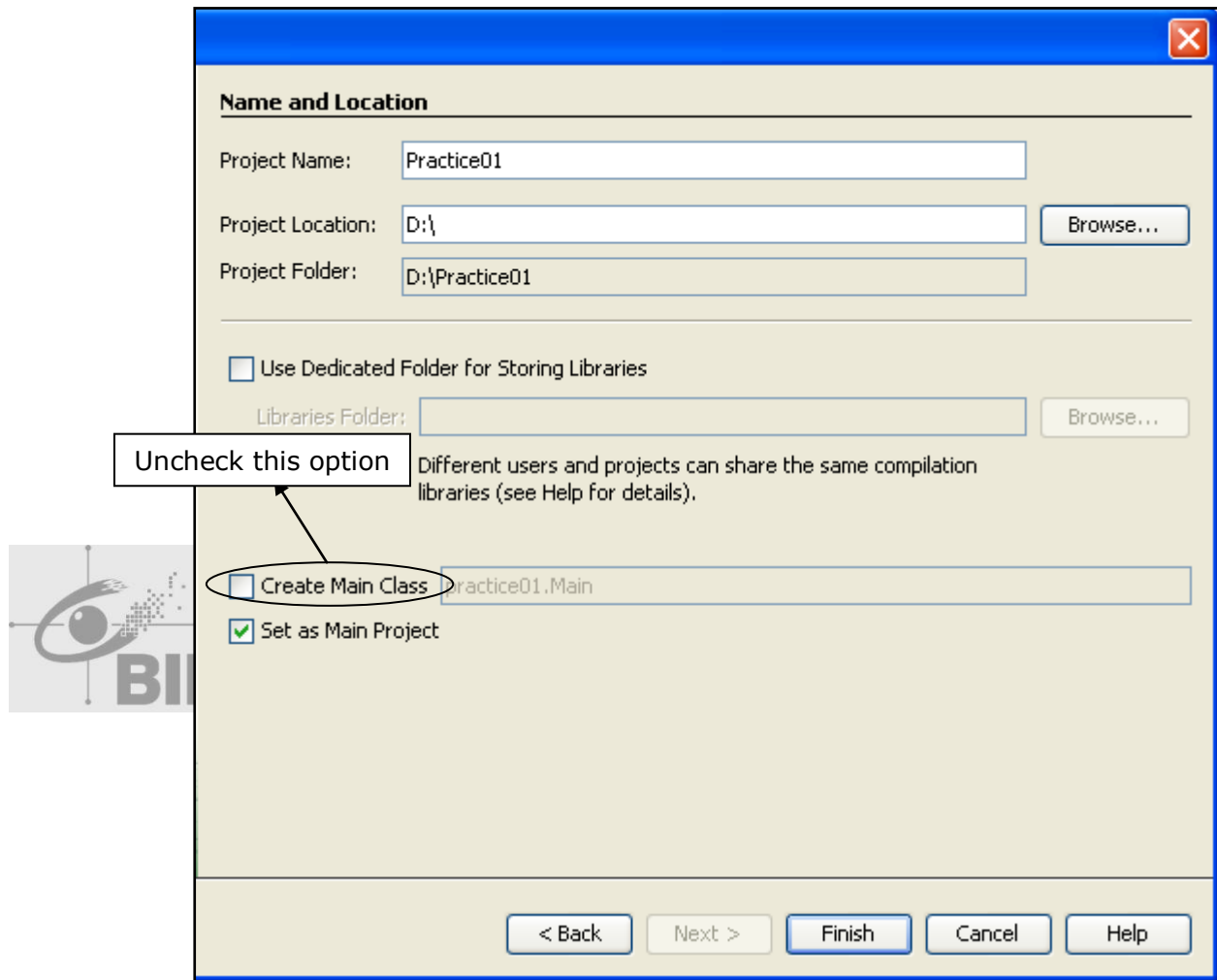
ID	Nama	TanggalLahi	Umur
2	poi poi	11/16/1990	20
3	yoyo	2/15/1991	19
*	(New)		

Exercise 02 – How to using ResultSet

a. Task 01 – Create Project Java in NetBeans

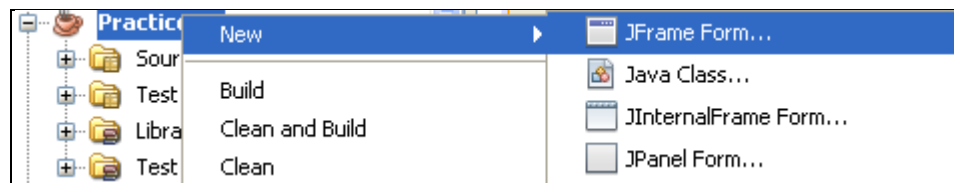
1. Run NetBeans from Start Menu
2. Open Menu File -> NewProject
3. On New Project Window, choose categories Java then choose Projects JavaApplication
4. Then Press Next

5. Enter your project name “Practice01”
6. Browse Location of your project in D:\
7. Unchecklist the CreateMainClass
8. Then Press Finish

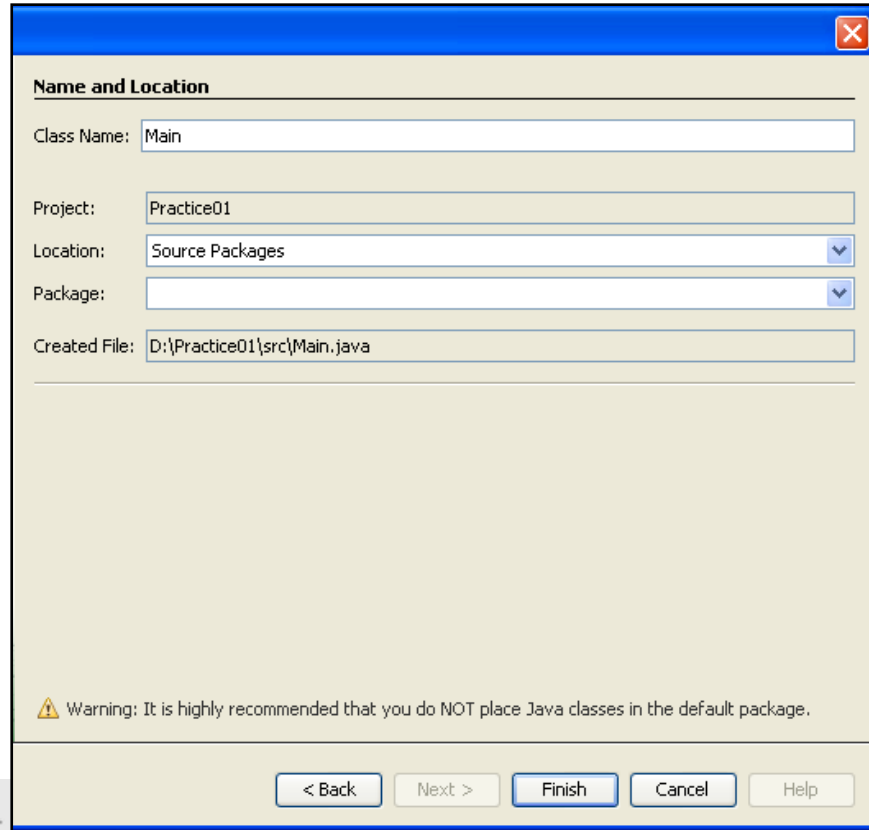


b. Task 02 – Create Make a JFrame in your project

1. On the left side, right click on your mouse then choose New–JFrameForm

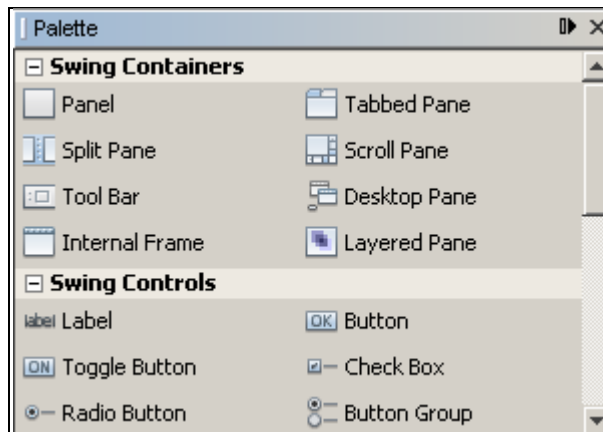


2. Then fill your ClassName“Main”
3. Then Press Finish



c. Task 03 – Create User Interface in JFrameForm

1. Drag component that is served in the palette window



- Choose Textfield 3 pieces, Label 3 pieces, Button 4 pieces until like this image

- On New Project Window, choose categories Java the choose Projects JavaApplication

➤ Task 04 – Create Connection from Java to Access

- Type import for SQLConnection

```
import java.sql.*;
```

- Create Connection Method

```
public void connect()
{
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection("jdbc:odbc:Driver=(Microsoft Access Driver (*.mdb));DBQ=db1.mdb");
        st = con.createStatement(1004, 1008);
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, "Connection Failed");
    }
}
```

If an error occurs when creating a connection, just click the error, then the error will be handled by try catch. Everything connected with the database should be handled by the try catch.

➤ Task 05 – Create select query for the data can be selected

- Create query select at the constructor of JFrameForm

```
try {
    rs = st.executeQuery("SELECT * FROM MsMahasiswa");
} catch (SQLException ex) {
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
}
```

➤ Task 05 – Create the contents of the function button first

1. Double Click at button first

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

2. Fill content from function of the first button

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        rs.first();
        jTextField1.setText(rs.getString(1));
        jTextField2.setText(rs.getString(2));
        jTextField3.setText(rs.getString(3));
    } catch (SQLException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

rs.first() means that the cursor can point the first rows.

MsMahasiswa			
	NIM	Name	Address
	1100456888	Grok	Jl. Bandengan
	1200123456	Rapenda	Jl. Jambi
	1200456789	Sean Young	Jl. Binus
	1300100896	Mahenda	Jl. Cengkareng
*			

The first row will be selected

jTextField1.setText(rs.getString(1))

- setText means, to set text from rs.getString(1).
- rs.getString(1) it means the index column in your query. So, index in query is started from 1.

➤ Task 06 – Create the contents of the function button previous

1. Double Click at button previous

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

2. Fill content from function of the prev button

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        if(rs.isFirst() == false)
        {
            rs.previous();
            jTextField1.setText(rs.getString(1));
            jTextField2.setText(rs.getString(2));
            jTextField3.setText(rs.getString(3));
        }
    } catch (SQLException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Explanation:

The validation above is to prove whether this row is the first row. If not, then the cursor can point the previous row.

- i. rs.isFirst() will return the false value if the row is not the first row and will return the true value if the row is the first row.
- ii. rs.previous(); means that the cursor can point previous row.

MsMahasiswa			
	NIM	Name	Address
	1100456888	Grok	Jl. Bandengan
	1200123456	Rapenda	Jl. Jambi
	1200456789	Sean Young	Jl. Binus
	1300100896	Mahenda	Jl. Cengkareng
*			

While Cursor at the third row

After press PrevButton

	NIM	Name	Address
	1100456888	Grok	Jl. Bandengan
	1200123456	Rapenda	Jl. Jambi
	1200456789	Sean Young	Jl. Binus
	1300100896	Mahenda	Jl. Cengkareng
*			

The second row will be selected

➤ Task 07 – Create the contents of the function button next

1. Double Click at button next

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

2. Fill content from function of the next button

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        if(rs.isLast() == false)
        {
            rs.next();
            jTextField1.setText(rs.getString(1));
            jTextField2.setText(rs.getString(2));
            jTextField3.setText(rs.getString(3));
        }
    } catch (SQLException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Explanation:

The validation above is to prove whether this row is the last row. If not, then the cursor can point the next row.

- i. `rs.isLast();` will return the false value if the row is not the last row and will return the true value if the row is the last row.
- ii. `rs.next();` means that the cursor can point next row.

	NIM	Name	Address
	1100456888	Grok	Jl. Bandengan
	1200123456	Rapenda	Jl. Jambi
	1200456789	Sean Young	Jl. Binus
	1300100896	Mahenda	Jl. Cengkareng
*			

While Cursor at the first row

After press Next Button

	NIM	Name	Address
	1100456888	Grok	Jl. Bandengan
	1200123456	Rapenda	Jl. Jambi
	1200456789	Sean Young	Jl. Binus
	1300100896	Mahenda	Jl. Cengkareng
*			

The second row will be selected

➤ Task 08 – Create the contents of the function button last

1. Double Click at button last

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

2. Fill content from function of the next button

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        rs.last();
        jTextField1.setText(rs.getString(1));
        jTextField2.setText(rs.getString(2));
        jTextField3.setText(rs.getString(3));
    } catch (SQLException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

rs.last(); means that the cursor can point the last rows.

	NIM	Name	Address
	1100456888	Grok	Jl. Bandengan
	1200123456	Rapenda	Jl. Jambi
	1200456789	Sean Young	Jl. Binus
	1300100896	Mahenda	Jl. Cengkareng
*			

The first row will be selected

	NIM	Name	Address
	1100456888	Grok	Jl. Bandengan
	1200123456	Rapenda	Jl. Jambi
	1200456789	Sean Young	Jl. Binus
	1300100896	Mahenda	Jl. Cengkareng
*			

The last row will be selected

Chapter 08

Database Operation



8.1 Data Manipulation

Data manipulation is the way in which data can be manipulated and changed. This type of operation is divided into several sections, among them:

a. Insert Data

Query insert is used to enter data into a table in a database in accordance with LinkConnection.

1. Example query for insert data:

```
INSERT INTO (table_name) [(column_name)] VALUES (Data)
```

2. Example below to enter data NIM, StudentName, and Programs in Table MsMahasiswa:

```
INSERT INTO MsMahasiswa VALUES ('1400123456' , 'Java',
'ComputerScience')
```

```
try{
st.executeUpdate("INSERTINTOMsMahasiswaVALUES('1400123456','Java','Co
mputerScience'");
}
catch(Exceptione){
    e.printStackTrace();
    System.out.println("ErrorDetected");
}
```

b. Delete Data

Query Delete is one of the Data Manipulation Language is used to delete existing data in the database. In the case of here we will be using MsAccess as the database and java as programming language.

Delete Query

To perform Delete Queries can be written as follows:

```
FORMAT:
DELETE FROM [table_name]
```

Example:

```
DELETE FROM MsUser
```

If we do the query as above, then the result, MsUser will be empty because we do not give the condition of the query. So that it will immediately delete the contents of table MsUser.

So if we only want to delete 1 line data only can we write:

FORMAT:

```
DELETE FROM [table_name] WHERE
[column_name]='value'
```

Example:

```
DELETE FROM MsUser WHERE Username='haha'
```

If we do the query above, then the result is a line that has the content 'haha' in the Username column is deleted.



BINUS UNIVERSITY | SOFTWARE LABORATORY CENTER

If 'value' of numbers then we do not need to add sign".

Example:

```
DELETE FROM MsUser WHERE Age=10
```

DeleteinJava

```
try{
st.executeUpdate("DeletefromMsUserwhereUsername='haha'");
}
catch(Exceptione){
e.printStackTrace();
System.out.println("ErrorDetected");
}
```

c. Update Data

To update data from existing table we use:

Format:

```
UPDATE [table_name] SET[column_name]=[value]
WHERE[condition]
```

Example:

```
UPDATE FROM MsUser Setname='bluejack' WHERE
Username LIKE'%haha%'
```

This query is mean, update from table MsUser, then set name“bluejack” where the condition for the username contain “haha”.

```
try{
    st.executeUpdate("UPDATEFROMMsUserSetname='bluejack'WHEREUs
ernameLIKE'%haha%'");
}
catch(Exceptione){
    e.printStackTrace();
    System.out.println("ErrorDetected");
}
```

8.2 PreparedStatement

Sometimes it is more convenient to use a PreparedStatement object for sending SQL statements to the database. This special type of statement is derived from the more general class, statement that you already know.

If you want to execute a Statement object many times, you better use PreparedStatement. Usually it reduces execution time.

The main feature of a PreparedStatement object is that, unlike a Statement object, it is given a SQL statement when it is created. The advantage to this is that in most cases, this SQL statement is sent to the DBMS right away, where it is compiled. As a result, the

PreparedStatement object contains not just a SQL statement, but a SQL statement that has been precompiled. This means that when the PreparedStatement is executed, the DBMS can just run the PreparedStatement SQLstatement without having to compile it first.

Although PreparedStatement objects can be used for SQL statements with no parameters, you probably use them most often for SQL statements that take parameters. The advantage of using SQL statements that take parameters is that you can use the same statement and supply it with different values each time you execute it. Examples of this are in the following sections.

To create a PreparedStatement same as using a Statement, except that were place the Statement with PreparedStatement like the example below

This is the example when we use Statement like usually

```
Statementst=con.createStatement(TYPE_SCROLL_INSENSITIVE,CONCUR_UPDAT
ABLE);
```

Then this is the example when we use PreparedStatement for Insert Query

```
PreparedStatementps = con.prepareStatement("INSERTINTOMsUserVALUES(?,?)");
ps.setString(1,"ichigo");//1meansindexforthefirstsign?
ps.setString(2,"ganteng");//2meansindexforthesecondsign?
ps.executeUpdate();//forexecutethequery
```

If using PreparedStatement, it is identical with a question mark(?), because the question mark determines the order of the data to be entered into the database.

- ps.setString(intparameterIndex,Stringx); means to sets the designated parameter to the given Java String value.
- ps.executeUpdate(); it's for executes the SQLstatement such an SQL Data Definition Language(DDL) statement, like INSERT, UPDATE or DELETE

Update Query

```
PreparedStatementps =
con.prepareStatement("UPDATERMsMahasiswaSETIPK=?WHEREINIM=?");
ps.setString(1,"3.4");
```

```
ps.setString(2, "1200123456");  
ps.executeUpdate();
```

Delete Query

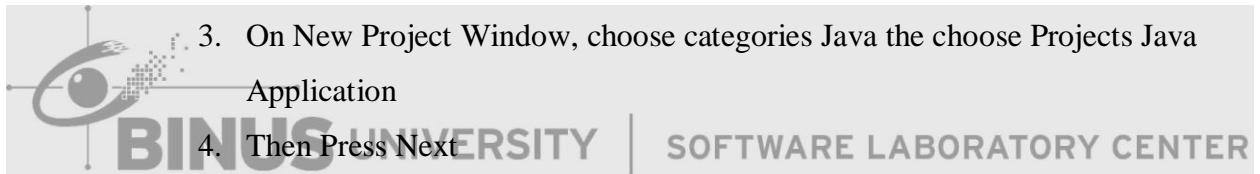
```
PreparedStatement ps =  
con.prepareStatement("DELETE FROM MsMahasiswa WHERE NIM=?");  
ps.setString(1, "1200123456");  
ps.executeUpdate();
```

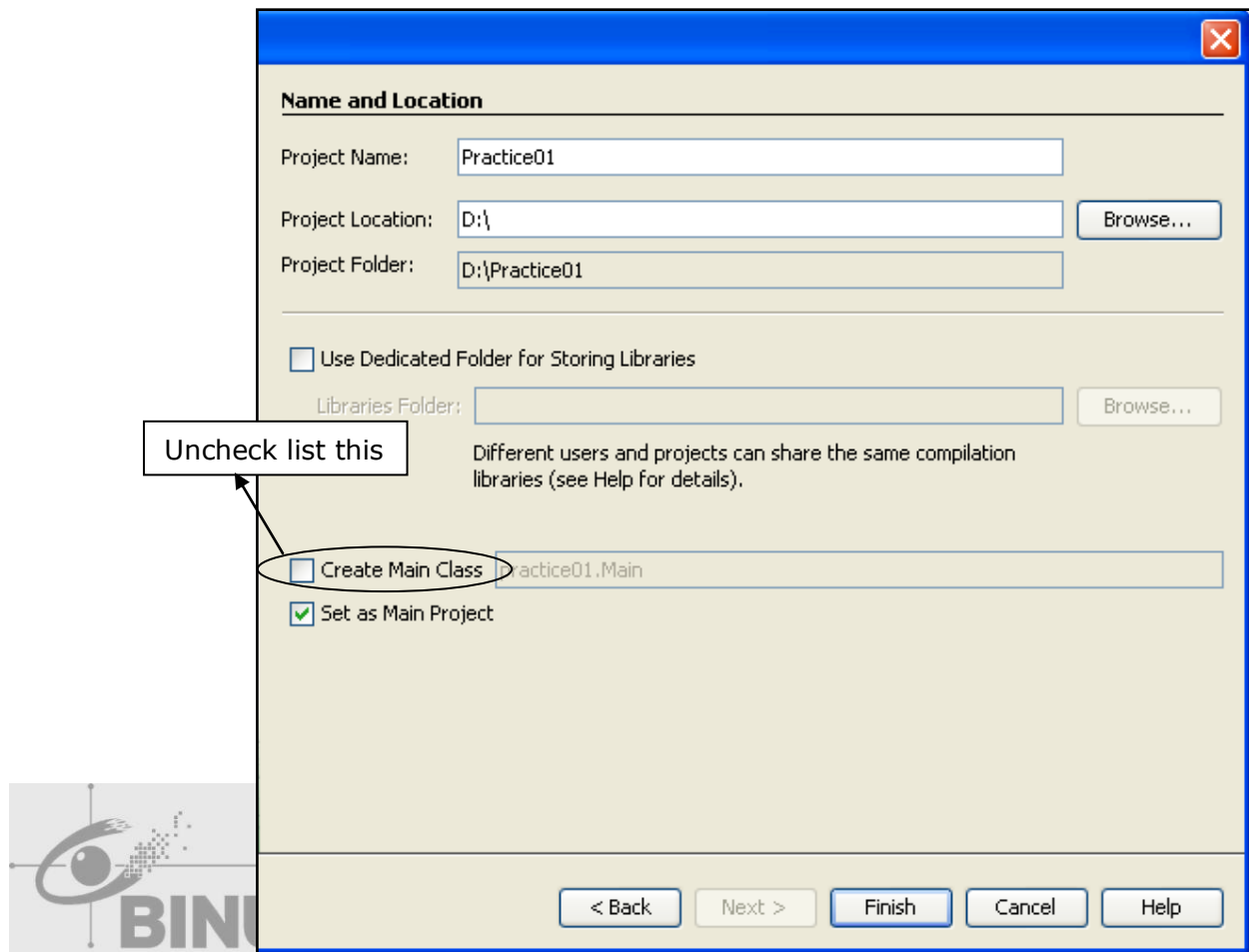
8.3 Exercise

Let's make a Simple Database Operation

a. Task 01 – Create Project Java in NetBeans

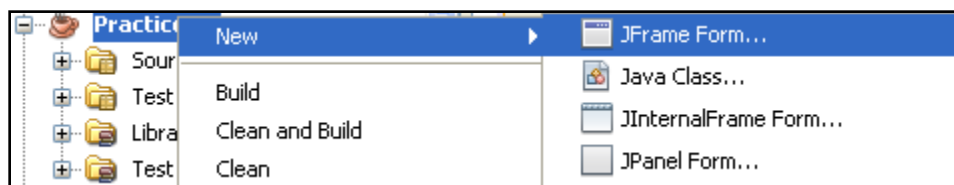
1. Run NetBeans from Start Menu
2. Open Menu File -> New Project
3. On New Project Window, choose categories Java the choose Projects Java Application
4. Then Press Next
5. Enter your project name "Practice01"
6. Browse Location of your project in D:\
7. Unchecklist the Create Main Class
8. Then Press Finish



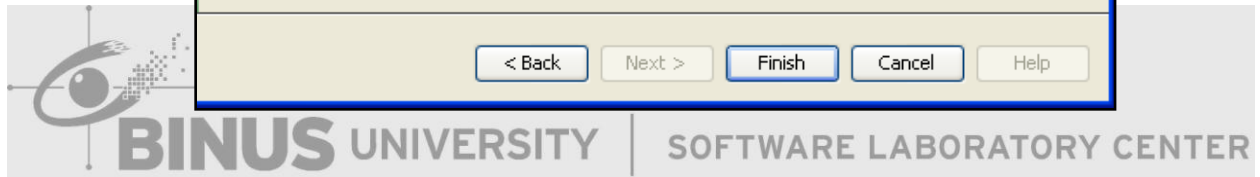
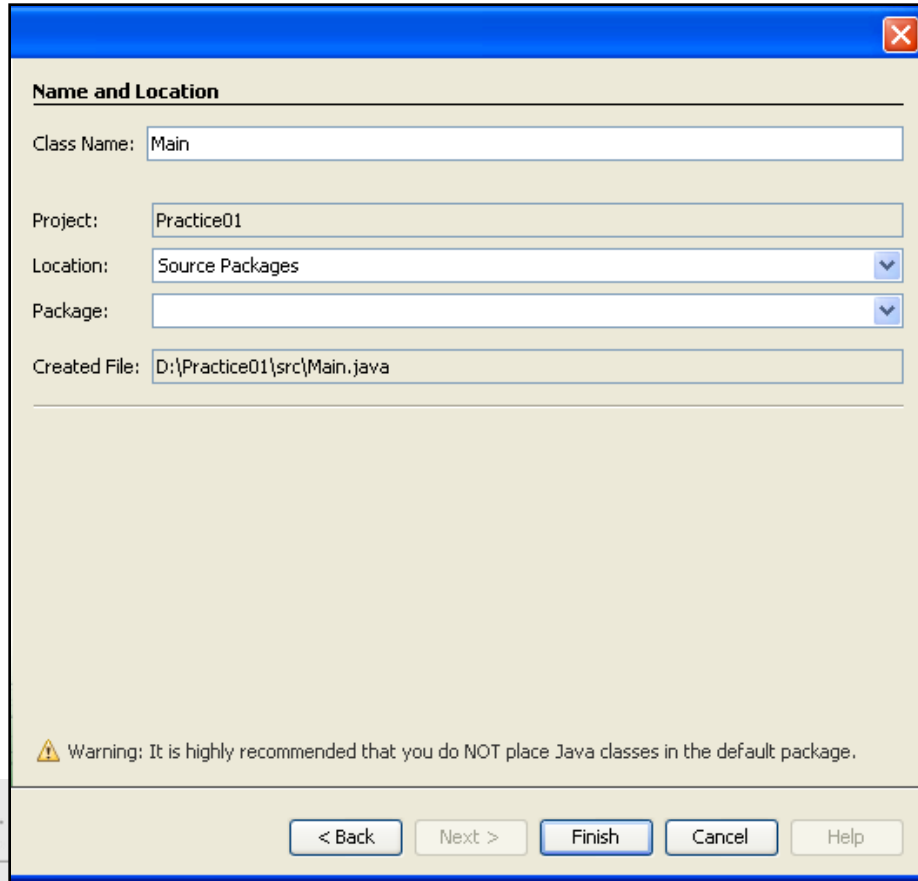


b. Task 02 – Create Make a JFrame in your project

1. On the left side, right click on your mouse then choose New–JFrameForm

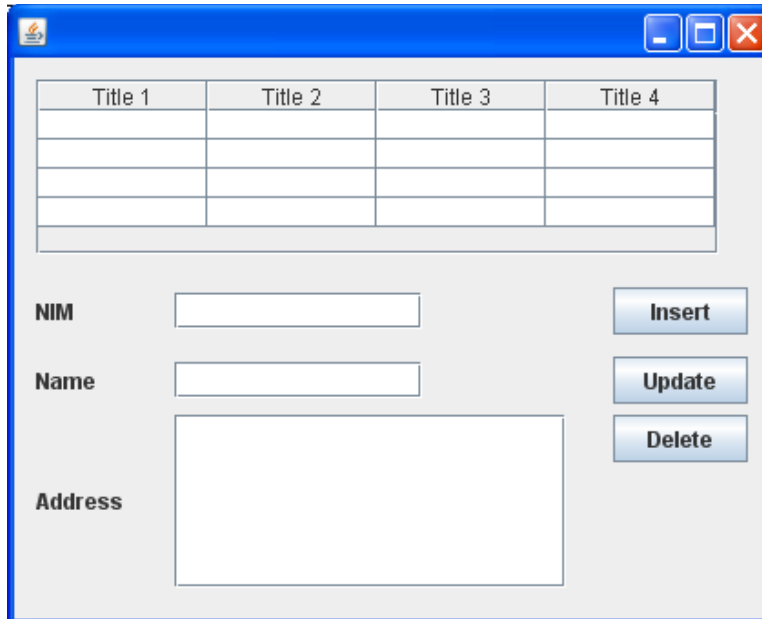


2. Then fill your ClassName “Main”
3. Then Press Finish



c. Task 03 – Make User Interface in JFrameForm

1. Type the import for SqlConnection



d. Task 03 – Make a connection in Java

1. Type the import for SqlConnection

```
import java.sql.*;
```

2. Create Connection Method

```
public void connect()
{
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection("jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=db1.mdb");
        st = con.createStatement(1004, 1008);
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, "Connection Failed");
    }
}
```

e. Task 04 – Create Method for fill the Table



```
public void fillTable()
{
    Vector <String> kolom = new Vector<String>();
    Vector<Vector> baris = new Vector<Vector>();
    Vector isi;
    DefaultTableModel tm = new DefaultTableModel(baris, kolom);

    kolom.add("NIM");
    kolom.add("Name");
    kolom.add("Address");
    try {
        rs = st.executeQuery("SELECT * FROM MsMahasiswa");

        while(rs.next())
        {
            isi = new Vector();
            isi.add(rs.getString("NIM"));
            isi.add(rs.getString("Name"));
            isi.add(rs.getString("Address"));
            baris.add(isi);
        }
    } catch (SQLException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
    jTable1.setModel(tm);
}
```

- `Vector<String> kolom = new Vector<String>();` This vector is use to accommodate the name of the column.

- `Vector <Vector> baris = new Vector <Vector>();` This vector is use to accommodate the content of the table.
- `Vector isi;` This vector is use to hold temporary data that will be transferred to the `Vectorbaris`.
- `DefaultTableModel tm = new DefaultTableModel (Vector rows, Vector column);` This `DefaultTableModel` is an implementation of `TableModel` that uses a `Vector` of `Vectors` to store the cell value objects.
- `Rs = st.executeQuery(“SELECT * FROM MsMahasiswa”);` means we do a simple query that will select its contents and will be accommodated into the `ResultSet rs`.
- `While(rs.next()){ }` This statement is performed a useful loop to include all the data in to the `Vector baris`.
- `Isi = new Vector();` We have to initialize (with keyword ‘new’) the object that we want to use.
- `Isi.add(Objectdata);` Appends the specified element to the end of this `Vector`
- `rs.getString(StringcolumnLabel);` Retrieves the value of the designated column in the current row of this `ResultSet` object as a `String` in the Java programming language.
- Or you can use `rs.getString (intcolumnIndex);` Retrieves the value of the designated column in the current row of this `ResultSet` object as a `String` in the Java programming language.
- `jTable1.setModel(tm);` it sets the data model for this table to new `Model` and registers with it for listener notifications from the new `datamodel`.

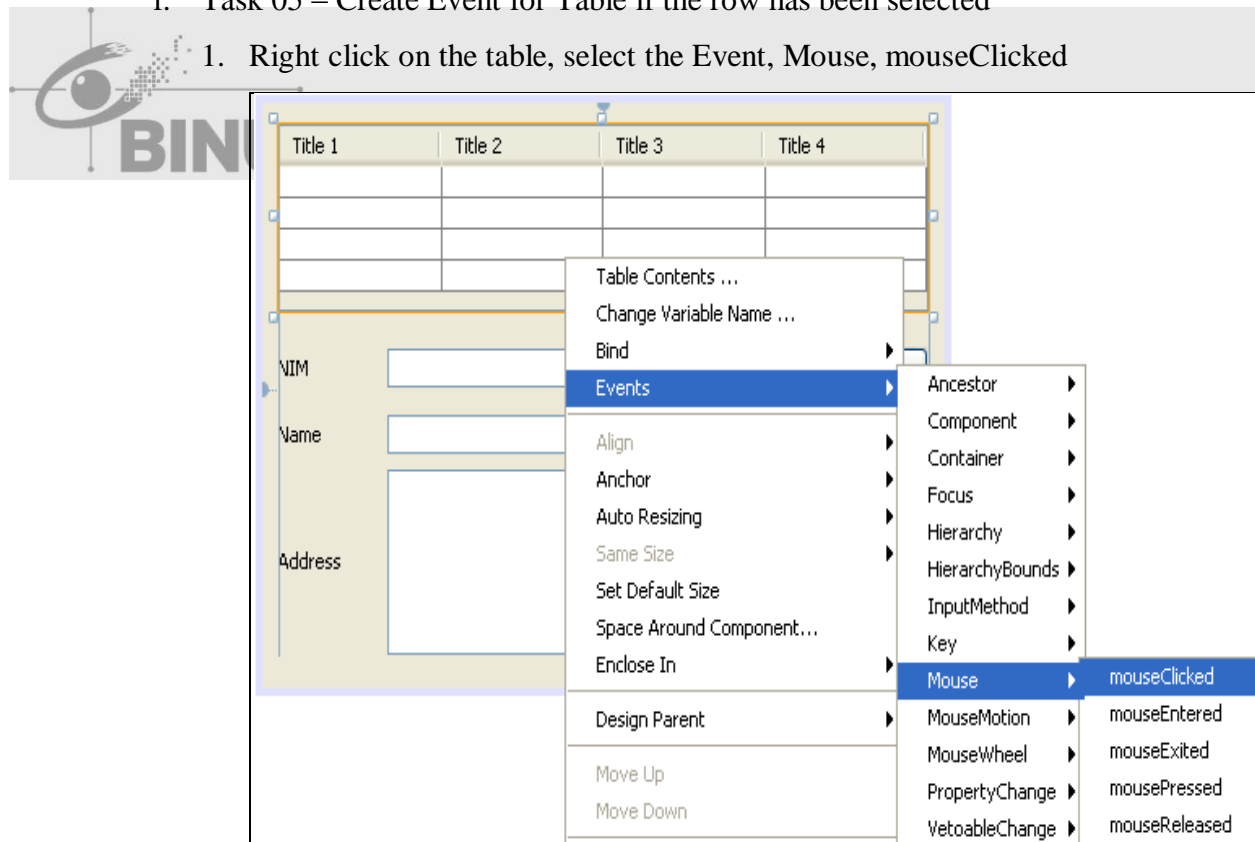


NIM	Name	Address
1200123456	Rapenda	Jl. Jambi
1200190101	Sonny Salim	Jl. Bekasi
1200919101	Gomantit	Jl. Kebun Jeruk
1200952472	Sean Young	Jl. Binus
1300101021	Mahenda	Jl. Cengkareng

NIM **Insert**
Name **Update**
Address **Delete**

f. Task 05 – Create Event for Table if the row has been selected

1. Right click on the table, select the Event, Mouse, mouseClicked



2. Fill the content of the method jTable1MouseClicked

```
private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {
    int baris = jTable1.getSelectedRow();
    jTextField1.setText(jTable1.getValueAt(baris, 0).toString());
    jTextField2.setText(jTable1.getValueAt(baris, 1).toString());
    jTextField3.setText(jTable1.getValueAt(baris, 2).toString());
}
```

- jTable1.getSelectedRow(); it's use to accommodate the selected row from table.
- jTextField1.setText(); it's use to set the value at the jTextField
- jTable1.getValueAt(baris,0); it's use to get the value from the selected row and data from column0.
- toString(); it's use to convert the object from get ValueAt (int row, int column) to String.

g. Task 06 – Create Insert Query in Button Insert



1. Double click at Insert Button on your JFrameForm

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

2. Fill the method at Button Insert

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        String nim = jTextField1.getText();
        String name = jTextField2.getText();
        String address = jTextField3.getText();
        String query = "INSERT INTO MsMahasiswa VALUES('" + nim + "', '" + name + "', '" + address + "')";
        st.executeUpdate(query);
    } catch (SQLException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

h. Task 07 – Create Update Query in Button Update

1. Double click at Update Button on your JFrameForm

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

2. Fill the method at Update Button

```

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        String nim = jTextField1.getText();
        String name = jTextField2.getText();
        String address = jTextField3.getText();
        String query = "UPDATE MsMahasiswa SET Name = '"+name+"', Address = '"+address+"' WHERE NIM = '"+nim+"'";
        st.executeUpdate(query);
    } catch (SQLException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

i. Task 08 - Create Delete Query in Button Delete

1. Double click at Delete Button on your JFrameForm

```

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

```

2. Fill the method at Delete Button

```

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        String nim = jTextField1.getText();
        String query = "DELETE FROM MsMahasiswa WHERE NIM = '"+nim+"'";
        st.executeUpdate(query);
    } catch (SQLException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```



Chapter 09

Java Applet



9.1 Introduction

Applet is java program that can be embedded into HTML pages. Java applets runs on the java enables web browsers such as Mozilla and internet explorer. Applet is designed to run remotely on the client browser, so there are some restrictions on it. Applet can't access system resources on the local computer. Applets are used to make the website more dynamic and entertaining.

Advantages of Applet:

- Applets are cross platform and can run on Windows, Mac OS and Linux platform
- Applets can work all the version of Java Plug-in
- Applets run in a sandbox. Having applets run in the sandbox means that they cannot read/write local valuable resources (such as your files). So the user does not need to trust the code, so it can work without security approval
- Applets are supported by most web browsers
- Applets are cached in most web browsers, so will be quick to load when returning to a webpage
- User can also have full access to the machine if user allows

Disadvantages of Java Applet:

- Java plug-in is required to run applet
- Java applet requires JVM, so first time will takes significant startup time
- If applet is not already cached in the machine, it will be downloaded from internet and will take time
- It's difficult to design and build good user interface in applets compared to HTML technology

1. Applet Class

An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application. The Applet class must be the superclass of any applet that is to be embedded in a Webpage or viewed by the JavaApplet Viewer. Any class that uses the applet must reduce its class of java.applet.Applet. The Applet class provides a standard interface between applets and their environment.

Java applets do not have a main function. When applet is started, create an instant web browser applet by calling the constructor of the applet that contains no arguments or parameters.

To control the applet, the browser uses the functions:

- Init
- Start
- Stop
- Destroy

Init function:

- Called when the applet is created.
- Derived class must override this function.
- Usually used for the initialization includes setting user interface components.

Start function:

- Called after init function or each time you visit a webpage.
- Running the functions to be run in an applet, such as animation.
- Derived class must override this function.

Stop function:

- Called when the user leaves a webpage containing the applet.
- Applet becomes inactive.
- Derived class must override this function.

Destroy function:

- Called when the user closes the browser containing the applet.
- All source and object of the object is deleted.
- Called after a stop run first.
- Derived class must override this function.

Framework derived class using the applet:

```
public class Practice01 extends java.applet.Applet
{
    /*there is no argument to the constructor called by the browser when
    a web that contains the applet is initialized, or run*/
    public Practice01()
    {
    }

    /*called when the browser after running the applet*/
    public void init()
    {
    }

    /*called when the web pages you visit*/
    public void start()
    {
    }

    /*called when a web page leaved and become inactive*/
    public void stop()
    {
    }

    /*called when the web browser is closed*/
    public void destroy()
    {
    }
}
```

Must
override
derived
class

Applet class is not designed to work with Swing components.

To use the Swing components of the Applet (java.applet.Applet) you better use JApplet (javax.swing.JApplet)

Default layout of the JApplet: BorderLayout.

Example Using JApplet

```
import javax.swing.*;

public class Practice01 extends JApplet
{
    /*initialized Applet*/
    public void init()
    {
        add(new JLabel("Welcome to Java Applet", JLabel.CENTER));
    }
}
```


The above class cannot be executed simply because they do not have a function main. Must create an HTML file using applet tag<applet> that references to the applet class. The browser will automatically create a GUI component that handles frames, the framesize also must be given and change the framevisibility (so it can be displayed).

9.2 Java Applet in Web

If you want to display java applet into a web browser, then we have to make HTML script first. HTML tags used are <applet> tag. If you want to know about and want to learn about HTML, you can read book about HTML.

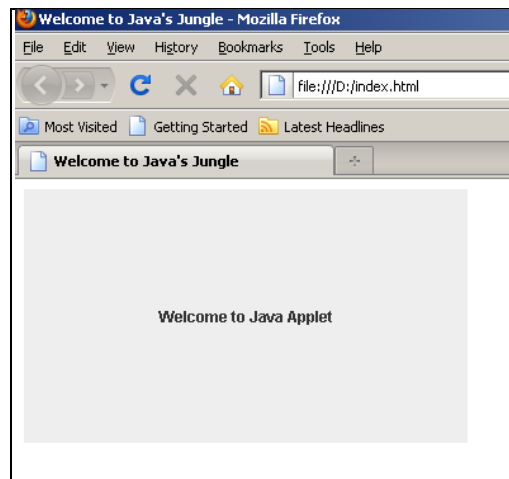
Examples of when the applet is called by the Web.

```
<html>
  <head>
    <title>Welcome to Java's Jungle </title>
  </head>

  <body>
    <applet code="Practice01.class" width = 350 height = 200<</applet>
  </body>
</html>
```

After you make this HTML script, then you can run this HTML script on your web browser such a MozillaFirefox, GoogleChrome, etc.

- Code = "Practice01.class" used to display our applet class, which is 'Practice01.class'.
- Width = 350it's for specifies the width of an applet
- Height = 200it's for specifies the height of an applet

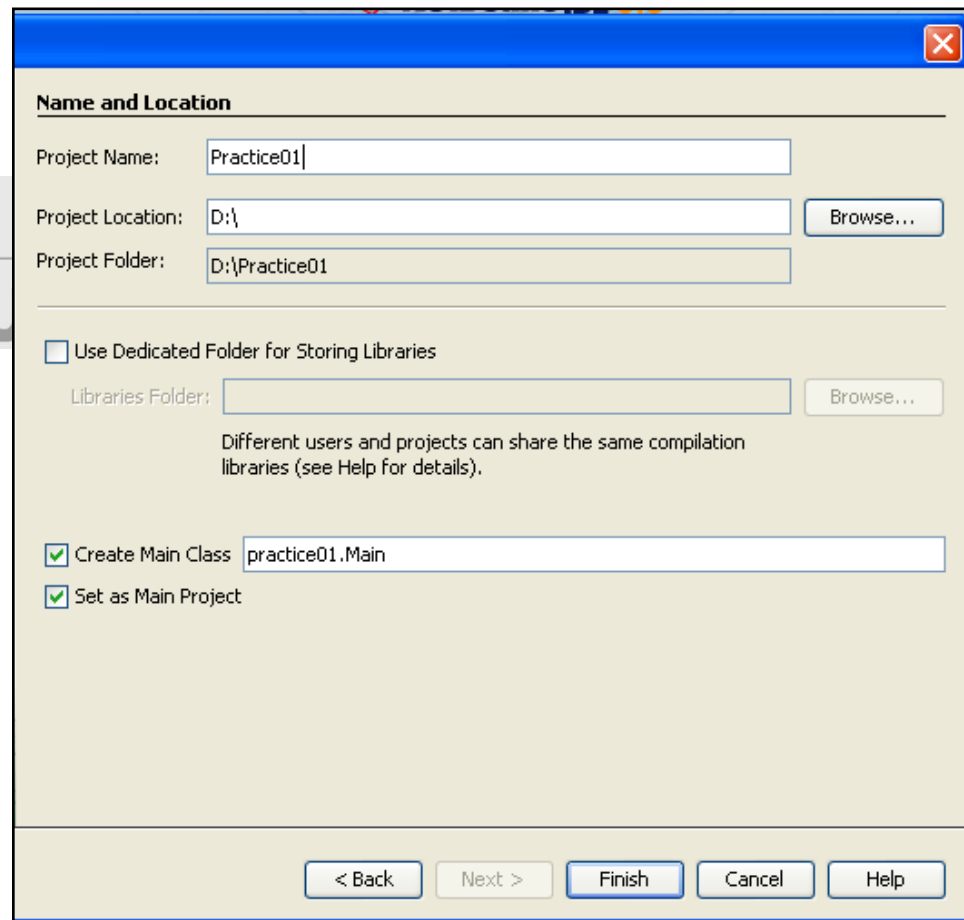


9.3 Exercise

Exercise 01 – make a simple User Interface in Java Applet

➤ Task 01 - Create Project Java in NetBeans

1. Run NetBeans from Start Menu
2. Open Menu File -> New Project
3. On NewProject Window, choose categories Java the choose Projects JavaApplication
4. Then Press Next
5. Enter your projectname “Practice01”
6. Browse Location of your project in D:\
7. Then Press Finish



The screenshot shows the 'Name and Location' dialog box in NetBeans. The 'Project Name' field contains 'Practice01'. The 'Project Location' field contains 'D:\' and has a 'Browse...' button next to it. The 'Project Folder' field contains 'D:\Practice01'. There is an unchecked checkbox for 'Use Dedicated Folder for Storing Libraries' with a 'Libraries Folder:' field and a 'Browse...' button below it. Below that, there is a note: 'Different users and projects can share the same compilation libraries (see Help for details)'. There are two checked checkboxes: 'Create Main Class' with the value 'practice01.Main' in the adjacent field, and 'Set as Main Project'. At the bottom, there are five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

➤ Task 02 – Create the Java Applet

```
import java.awt.*;
import javax.swing.JApplet;
public class Main extends JApplet{

    Font f = new Font("Verdana", Font.BOLD, 20);
    String name;

    public void init()
    {
        this.name = getParameter("name");
        if (this.name == null)
            this.name = "Laura";

        this.name = "Hello "+this.name;
    }

    public void paint(Graphics g)
    {
        Font f = new Font("Verdana", Font.BOLD, 20);
        g.setFont(f);
        g.setColor(Color.RED);
        init();
        g.drawString("Welcome to Java Applet\n", 20, 50);
        g.drawString(this.name, 20, 100);
    }

    public static void main(String[] args) {
        new Main();
    }
}
```

- Font f = new Font ("Verdana", Font.BOLD, 20); means to create a new Font from the specified name, style and point size.
- g.setFont(f); it's use to set this graphics context's font to the specified font.
- g.setColor(Color.RED); it's use to set this graphics context's current colour to the specified colour.
- g.drawString(Stringname, int x, int y); it's use to draw the text given by the specified string, using this graphics context's current font and colour.
- init(); this function init is used to validate whether the name is NULL or not?
- If NULL, it will be filled by "Laura"
- If NOT NULL, it will be filled by the existing value in the HTML
- getParameter(Stringname); it's use to return the value of the named parameter in the HTMLtag

➤ Task 03 – Create HTML Script

```
<html>
  <head>
    <title>Welcome to Java's Jungle </title>
  </head>

  <body>
    <applet code = "Main.class" width = 500 height = 500>
      <param name = "name" value = "poipoi">
    </applet>
  </body>
</html>
```

- `<paramname="name" value="poipoi">` it's used to define parameters or variables for an object or applet element.
- Name = "name" it's used to defines the name for a parameter
- value = "poipoi" it's used to give the specifies the value of a parameter

Chapter 10

UML Tools



10.1 Theory

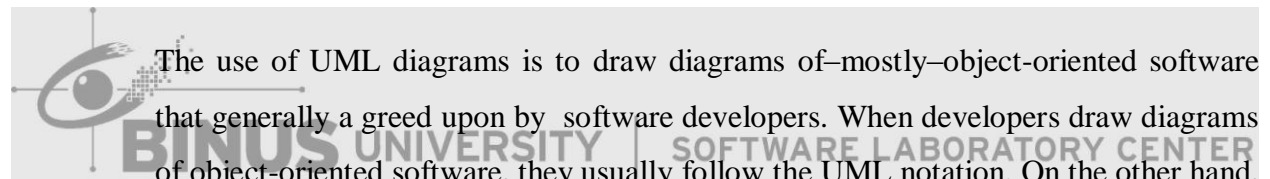
A **UML tool** or **UML modelling tool** is a software application that supports some or all of the notation and semantics associated with the **Unified Modelling Language(UML)**, which is the industry standard general purpose modelling language for software engineering.

UML tool is used broadly here to include application programs which are not exclusively focused on UML, but which support some functions of the Unified Modelling Language, either as an *add-on*, as a *component* or as a *part* of their overall functionality.

UML tools support the following kinds of functionality:

- **Diagramming**

Diagramming in this context means *creating* and *editing* UML diagrams; that is diagrams that follow the graphical notation of the Unified Modelling Language.



The use of UML diagrams is to draw diagrams of—mostly—object-oriented software that generally a greed upon by software developers. When developers draw diagrams of object-oriented software, they usually follow the UML notation. On the other hand, it is often debated whether those diagrams are needed at all, during what stages of the software development process they should be used, and how (if at all) they should be kept up to date. The primacy of software code often leads to the diagrams being deprecated. But the diagrams help developers to maintain or develop the systems.

- **Round-TripEngineering**

Round-trip engineering refers to the ability of a UML tool to perform code generation from models, and model generation from code(a.k.a.,reverse engineering), while keeping both the model and the code semantically consistent with each other. Code generation and reverse engineering are explained in more detail below.

- **CodeGeneration**

Code generation in this context means that the user creates UML diagrams, which have some connoted model data, and the UML tool derives from the diagrams part or all of the

source code for the software system. In some tools the user can provide a skeleton of the program source code, in the form of a source code template, where predefined tokens are then replaced with program source code parts during the code generation process.

- **ReverseEngineering**

Reverse engineering in this context means, that the UML tool reads program source code as input and *derives* model data and corresponding graphical UML diagrams from it (as opposed to the somewhat broader meaning described in the article "Reverse engineering").

Some of the challenges of reverse engineering are:

- The source code often has much more detailed information than one would want to see in design diagrams. This problem is addressed by software architecture reconstruction.
- Diagram data is normally not contained with the program source, such that the UML tool, at least in the initial step, has to create some *random layout* of the graphical symbols of the UML notation or use some automatic *layout algorithm* to place the symbols in a way that the user can understand the diagram. For example, the symbols should be placed at such locations on the drawing pane that they don't overlap. Usually, the user of such a functionality of a UML tool has to manually edit those automatically generated diagrams to attain some meaningfulness. It also often doesn't make sense to draw diagrams of the whole program source, as that represents just too much detail to be of interest at the level of the UML diagrams.
- There are language features of some programming languages, like *class-or function templates* of the C++ programming language, which are notoriously hard to convert automatically to UML diagrams in their full complexity.

- **Model and Diagram Interchange**

XML Metadata Interchange (XMI) is the format for UML model interchange. XMI does not support UML Diagram Interchange, which allows you to import UML diagrams from one model to another.

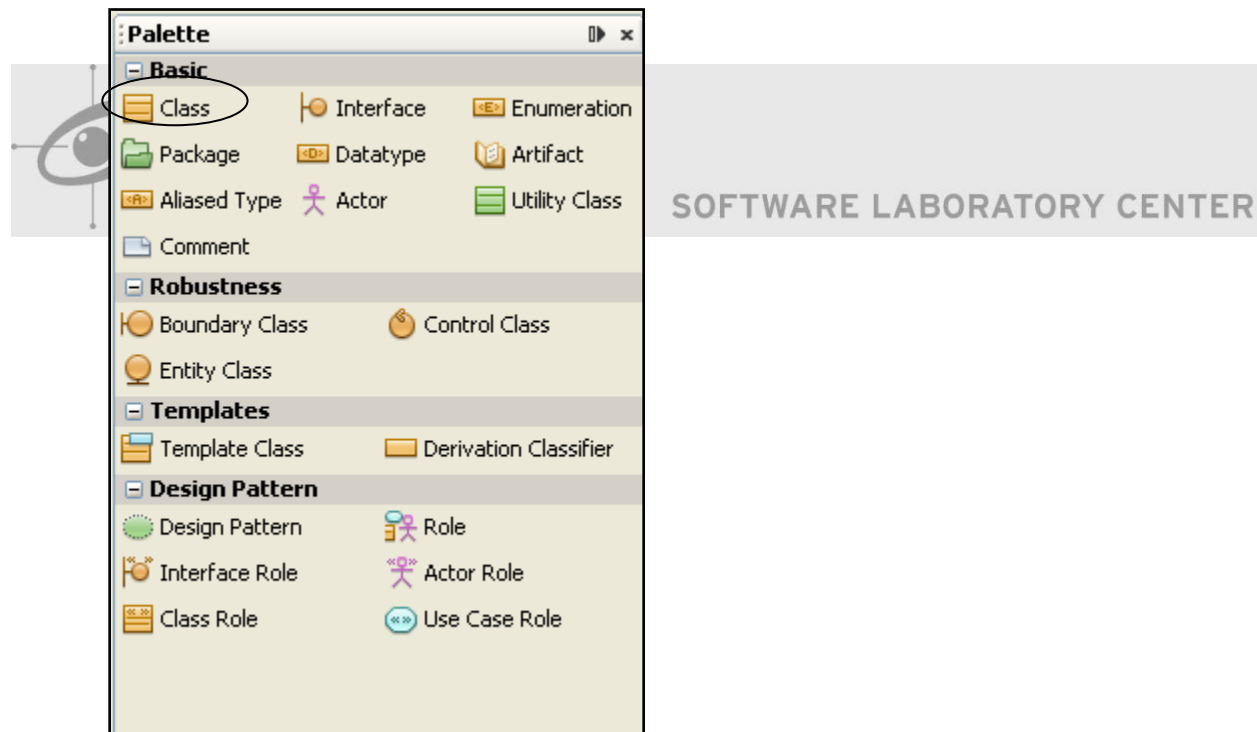
- **Model Transformation**

A key concept associated with the Model-driven architecture initiative is the capacity to transform a model into another model. For example, one might want to transform a platform-independent domain model into a Java platform-specific model for implementation. It is also possible to refactor UML models to produce more concise and well-formed UML models. Finally, it is possible to generate UML models from other modelling notations, such as BPMN. The standard that supports this is called QVT(Queries/Views/Transformations). One example of an open-source QVT-solution is the ATL language built by INRIA.

UML Tools commonly used are as follows:

1. Class Diagram

Notation Class Diagram



10.2 Class Definition

A class describes a set of objects that share the same specifications of features, constraints, and semantics. Class is a kind of classifier whose features are attributes and operations.

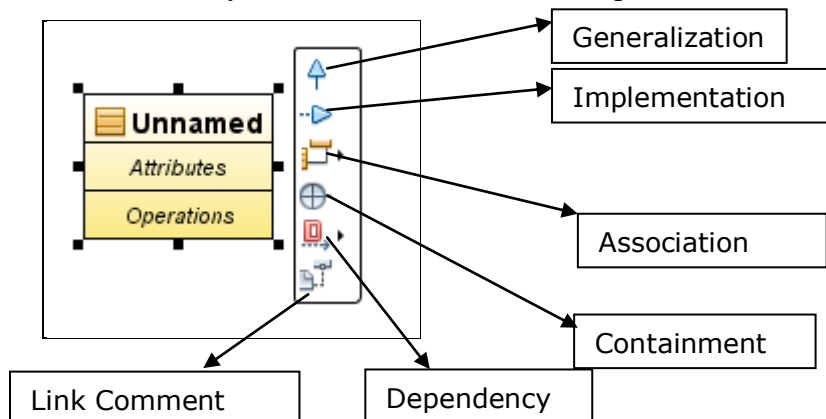
Attributes of a class are represented by instances of property that are owned by the class. Some of these attributes may represent the navigable ends of binary associations.

Properties

Name	The name of class.
Parent	The model element that owns the class.
Visibility	Determines where the class appears within different name spaces within the overall model, and its accessibility.
Documentation	Description of class.
Abstract	If true, the class does not provide a complete declaration and typically cannot be instantiated. An abstract class is intended to be used by other classes.
Leaf	Indicates whether it is possible to further specialize a class. If the value is true, then it is not possible to further specialize the class.
Root	Indicates whether the class has no ancestors. (true for no ancestors)
Active	Determines whether an object specified by this class is active or not. If true, then the owning class is referred to as an active class. If false, then such a class is referred to as a passive class.
Business Model	Set it to make the class become a "business class"
Attributes	Refers to all of the Properties that are direct (i.e., not inherited or imported) attributes of the class.
Operations	An operation is a behavioural feature of a class that specifies the name, type, parameters, and constraints for invoking an associated behaviour. Operations here refer to the operations owned by the class.
Template Parameters	A Template able Element that has a template signature is a specification of a template. A template is a parameterized element that can be used to generate other model elements using Template Binding relationships. The template parameters for the template signature specify the formal parameters that will be substituted by actual parameters (or the default) in a binding.

	<p>A template parameter is defined in the namespace of the template, but the template parameter represents a model element that is defined in the context of the binding.</p> <p>A template able element can be bound to other templates. This is represented by the bound element having bindings to the template signatures of the target templates. In a canonical model a bound element does not explicitly contain the model element simplified by expanding the templates it binds to, since those expansions are regarded as derived. The semantics and well-formedness rules for the bound element must be evaluated as if the bindings were expanded with the substitutions of actual elements for formal parameters</p>
Class Code Details	Properties of class in implementation (code) level. Settings in this page is programming language specific, and will affect the code being generated.
Java Annotations	A Java annotation is a metadata that can be added to Java source code for annotation purposes.
ORM Query	Available only to ORM Persistable class, ORM Query lets you define the ORM Qualifiers and named queries of the class.

There are many tools that can be used in the palette, but that is often used only class.



10.3 Class Level Relationship

- Generalization

The Generalization relationship indicates that one of the two related classes (the *subtype*) is considered to be a specialized form of the other (the *supertype*) and super type is considered as '**Generalization**' of subtype. In practice, this means that any instance of the subtype is also an instance of the supertype. An exemplary tree of generalizations of this form is found in binomial nomenclature: human beings are a subtype of simian, which are a subtype of mammal, and soon. The relationship is most easily understood by the phrase 'an A is a B ' (a human is a mammal, a **mammal** is an animal).

The UML graphical representation of a Generalization is a hollow triangle shape on the supertype end of the line (or tree of lines) that connect sit to one or more subtypes.

The generalization relationship is also known as the inheritance or "*is a*" relationship. The supertype in the generalization relationship is also known as the "*parent*", *superclass*, *baseclass*, or *base type*. The subtype in the specialization relationship is also known as the "*child*", *subclass*, *derived class*, *derived type*, *inheriting class*, or *inheriting type*.

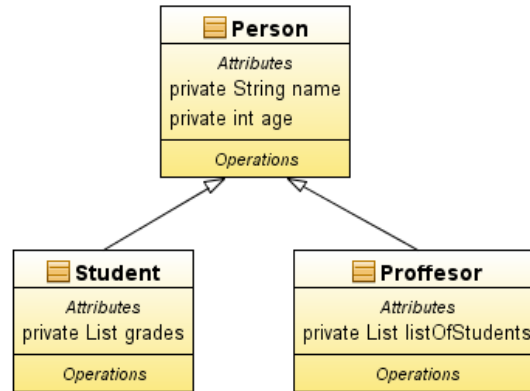
Note that this relationship bears no resemblance to the biological parent/child relationship: the use of these terms is extremely common, but can be misleading.

- Generalization-Specialization relationship

A is a type of B

E.g. "an oak is a type of tree", "an automobile is a type of vehicle"

Generalization can only be shown on class diagrams and on Use case diagrams.



Class diagram showing generalization between one superclass and two subclasses

Instance Level Relationships

- Association

An *Association* represents a family of links. Binary associations (with two ends) are normally represented as a line, with each end connected to a class box. Higher order associations can be drawn with more than two ends. In such cases, the ends are connected to a central diamond.

An association can be named, and the ends of an association can be adorned with role names, ownership indicators, multiplicity, visibility, and other properties. There are five different types of association. Bi-directional and uni-directional associations are the most common ones. For instance, a flight class is associated with a plane class bi-directionally. Associations can only be shown on class diagrams. Association represents the static relationship shared among the objects of two classes. Example: "department offers courses", is an association relation.



Class diagram example of association between two classes

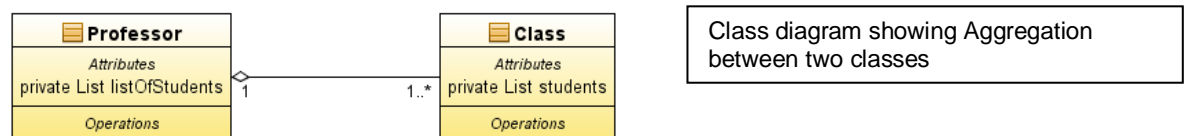
- Aggregation

Aggregation is a variant of the "has a" or association relationship; aggregation is more specific than association. It is an association that represents apart-whole or part-of relationship. As a type of association, an aggregation can be named and

have the same adornments that an association can. However, an aggregation may not involve more than two classes.

Aggregation can occur when a class is a collection or container of other classes, but where the contained classes do not have a strong *life cycle dependency* on the container—essentially, if the container is destroyed, its contents are not.

In UML, it is graphically represented as a *hollow diamond shape* on the containing class end of the tree of lines that connect contained class(es) to the containing class.

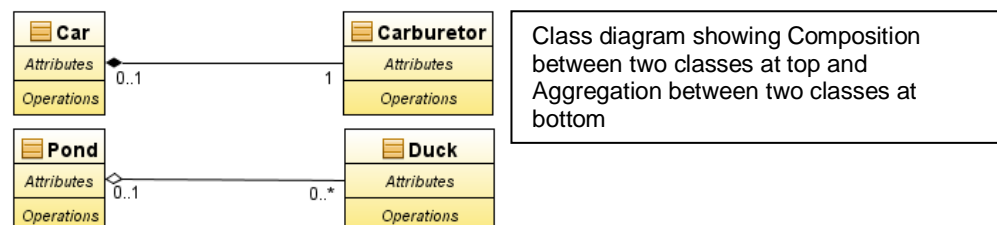


o Composition

Composition is a stronger variant of the "owns a" or association relationship; composition is more specific than aggregation.

Composition usually has a strong *life cycle dependency* between instances of the container class and instances of the contained class(es). If the container is destroyed, normally every instance that it contains is destroyed as well. Note that a part can (where allowed) be removed from a composite before the composite is deleted, and thus not be deleted as part of the composite.

The UML graphical representation of a composition relationship is a *filled diamond shape* on the containing class end of the tree of lines that connect contained class(es) to the containing class.



Differences between Composition and Aggregation

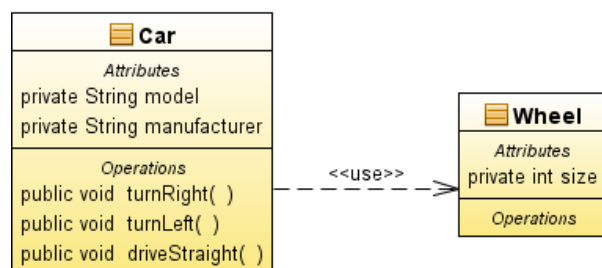
When attempting to represent real-world whole-part relationships, e.g., an engine is part of a car, the composition relationship is most appropriate. However, when representing software or database relationship, e.g., car model engine ENG01 is part of a car model CM01, an aggregation relationship is best, as the engine, ENG01 maybe also part of a different car model. Thus the aggregation relationship is often called "catalog" containment to distinguish it from composition's "physical" containment.

The whole of a composition must have a multiplicity of 0..1 or 1, indicating that a part must belong to only one whole; the part may have any multiplicity. For example, consider University and Department classes. A department belongs to only one university, so University has multiplicity 1 in the relationship. A university can (and will likely) have multiple departments, so Department has multiplicity 1..*.

General Relationship

- Dependency

Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point of time. Dependency exists if a class is a parameter variable or local variable of a method of another class.



Class diagram showing dependency between "Car" class and "Wheel" class

- Multiplicity

The association relationship indicates that (at least) one of the two related classes makes reference to the other. In contrast with the generalization relationship, this is most easily understood through the phrase 'A has a B' (a mother cat has kittens, kittens have a mother cat).

The UML representation of an association is a line with an optional arrow head indicating the *role* of the object(s) in the relationship, and an optional notation at each end indicating the *multiplicity* of instances of that entity (the number of objects that participate in the association).

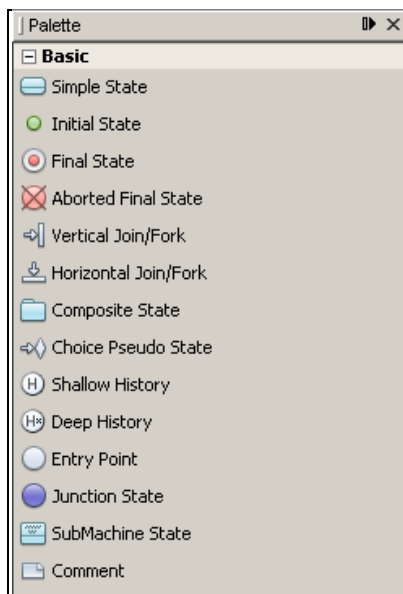
0..1	No instances, or one instance (optional, may)
1	Exactly one instance
0..*or*	Zero or more instances
1..*	One or more instances (at least one)

2. State Diagram

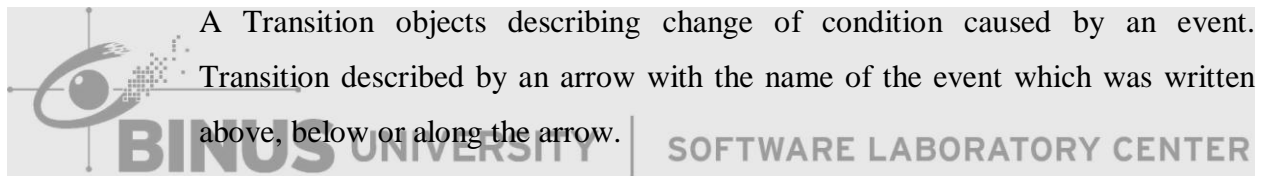
State chart diagrams, or commonly also called a state diagram used to document the various conditions/circumstances that could happen to a class and any activity that may alter the conditions/circumstances.

State diagrams model the transition would usually only happens only in a class. In general state chart diagram describes a certain class (one class can have more than one state chart diagram).

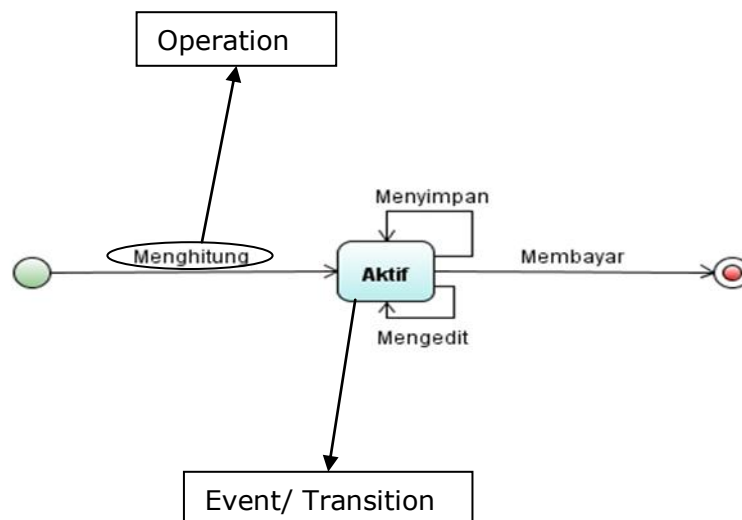
Notation State Diagram



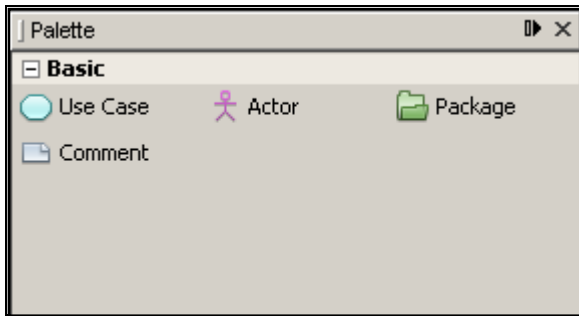
- State
State notation describing the condition of an entity, and illustrated with blunted gessquare with state name init.
- Initial State
Initial State is a state in the beginning of an object prior to any changes in circumstances. Initial State represented by a solid circle. Only one Initial State is permitted in a diagram.
- Final State
Final State describing the object stopped giving the response to an event. Final State represented by a solid circle inside an empty circle.
- Event/Transition



- Operation
An Operation is described in a verb form of the function or the running system.



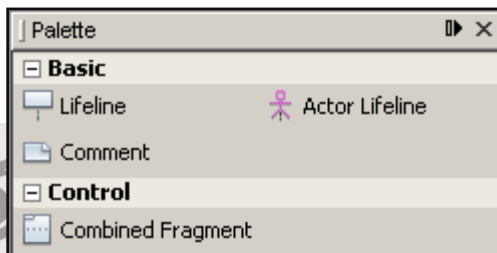
3. Use Case Diagram



Tools commonly used in use case diagrams, among others:

- Actor
- UseCases

4. Sequence Diagram



There are some tools which commonly used. They are:

- Life line

This is a tool to make a life line in a sequence diagram. A life line shows the life time of a class while sending some messages to the other classes. While the class is the window, or the grid, or the list, or anything that is not the class from the database, it will be destroyed in the end of the life line.
- Actor Life line

This is a tool to make a life line for the actor. Its characteristics are same with the common life line. It sends the messages to the other classes too. The messages are usually the instructions from the actor while running the system. Remember that it can't be destroyed too.
- Comment

This is a tool to insert any comments.
- Combined Fragment

This is a tool to make a combine fragment in a sequence diagram. Combine fragment groups some messages into a fragment which defines some conditions of a fragment by use operator.

There are some operators that commonly used. They are:

➤ alt

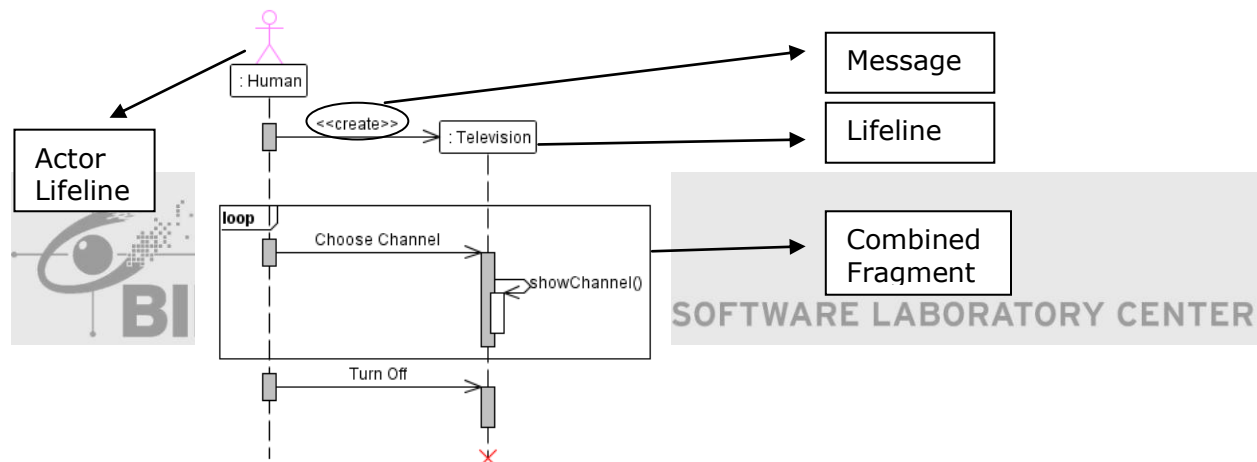
It is to make the alternate conditions from a fragment. So, if the first condition doesn't match, it will go to the next condition.

➤ opt

It is to make a fragment becomes optional.

➤ loop

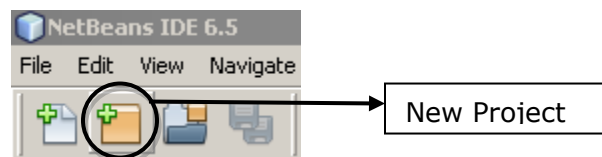
It is to make a fragment become slooping the message(s).



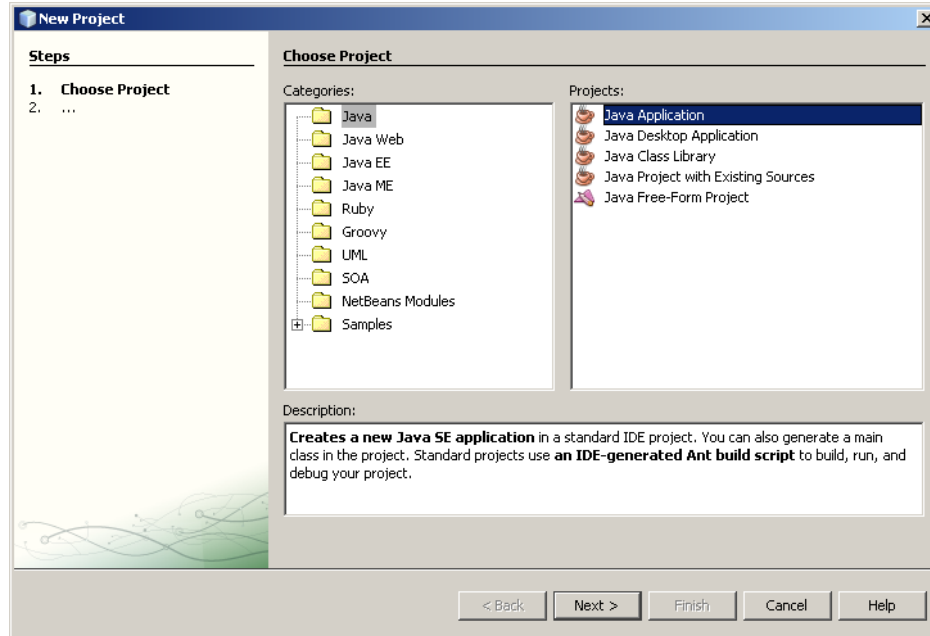
5. Generate Code

When you make a class diagram in the Netbeans, you can generate code from the diagram directly. These are the steps to generate the codes from a class diagram:

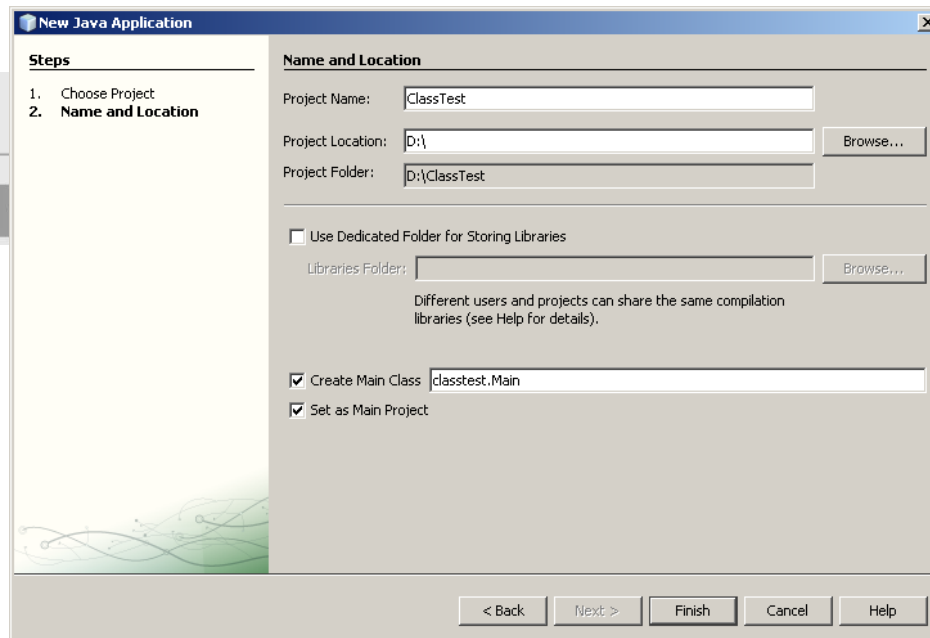
- First, make a new project (make sure that you have made the class diagram) by clicking the New Project (Ctrl+Shift+Nfortheshortcut).



- Then, choose **Java Category**, choose **Java Application**, then click **Next**.

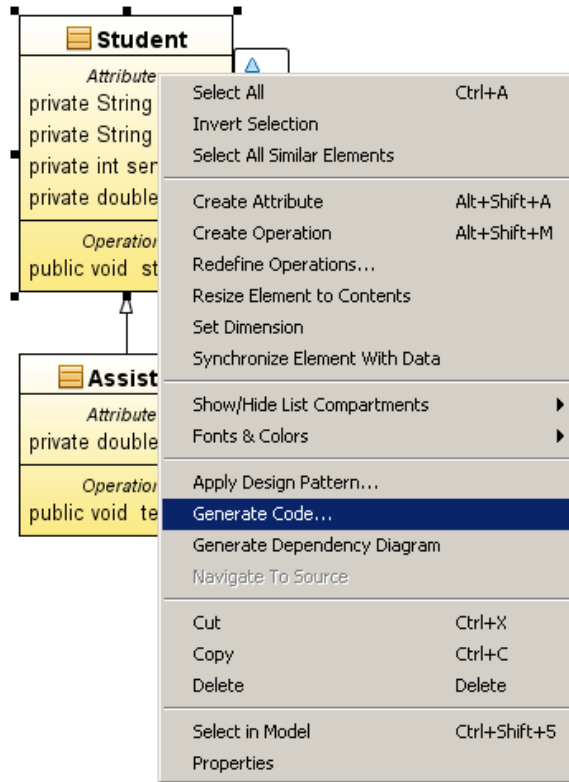


- Give the **Project Name** and choose the **Project Location**, then click **Finish**.

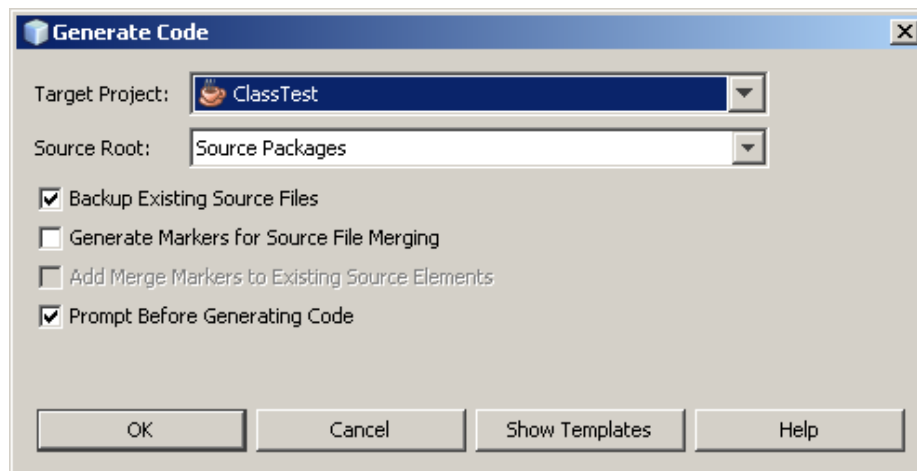


- Back to your project that contains the class diagram. Right click on the class, and then choose **GenerateCode**.

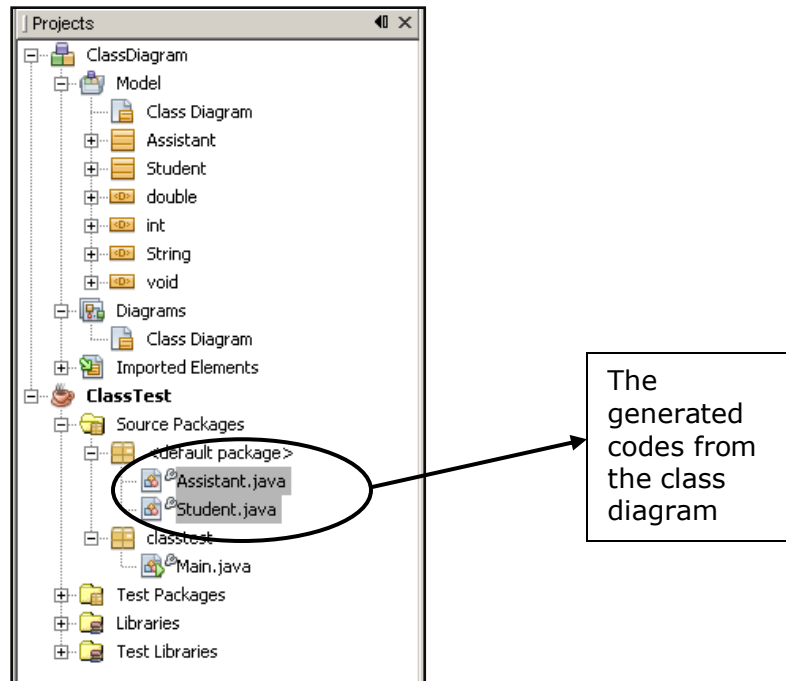
(For the example we create the Student class which has NIM, Name, Semester, and GPA attribute, and also the Study operation. And it has the subclass which named Assistant which has Salary attribute and Teach method.)



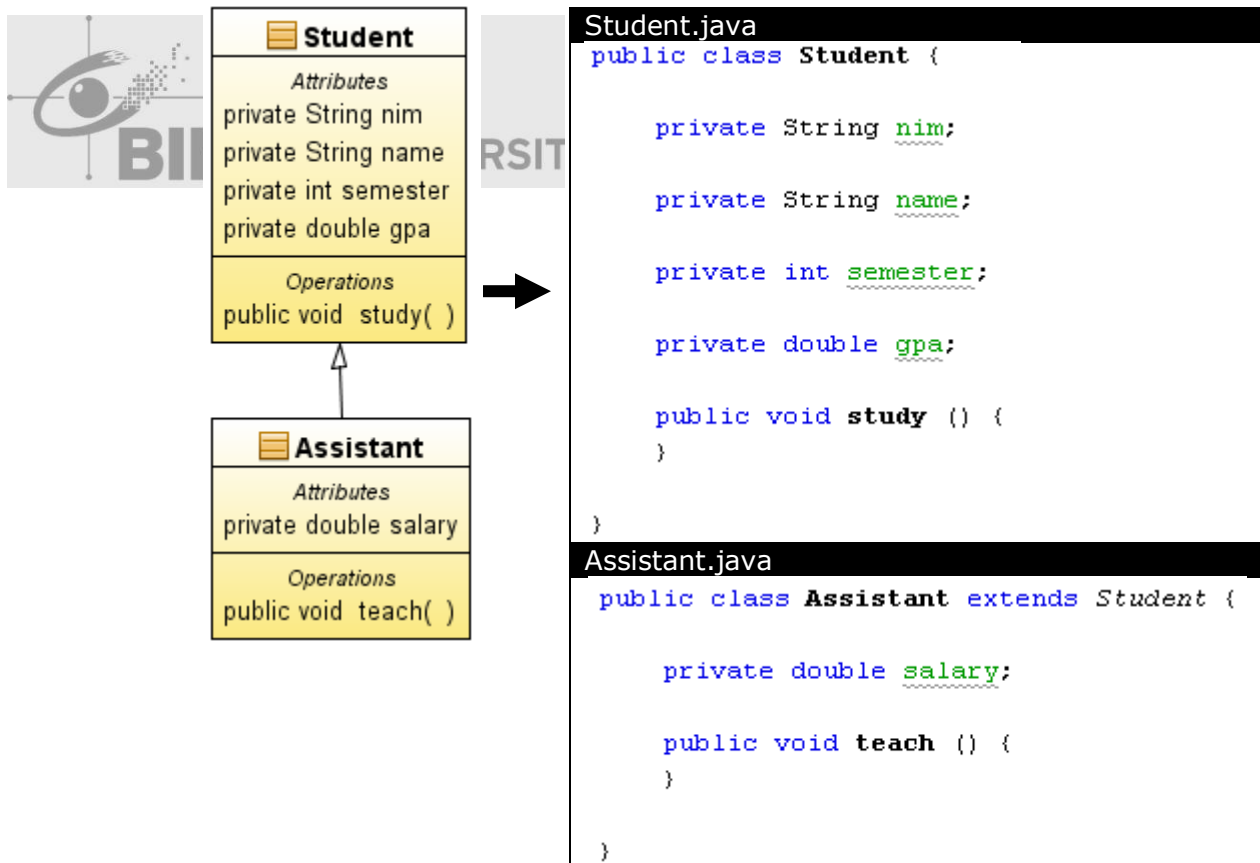
Choose the **Target Project** and choose **Source Packages** to be the **SourceRoot**. You can uncheck the **Generate Markers for Source File Merging** to make your codes don't contain any comment (it is optional). Then, click **OK**.



- Repeat it for the other classes until all classes are generated to be the codes.
- Now, you can check your **Project Explorer**, choose your **Java Application** project which has been the source for your generated codes. Then, you will see that the class diagram has been generated to be the codes.



- o So, it is the Class Diagram and its generated codes.



10.4 Exercise

- a) Mention the function of these diagrams below:
 - Class Diagram
 - State Chart
- b) Mention the difference of Life line and Actor Life line in the Sequence diagram!
- c) If the system has a **Purchase Order Transaction** and when the all fields in the form have been filled, then the **Purchase Division** can click **Submit** button to submit the data to database, or click **Cancel** button to cancel the transaction, what is the operators which we must choose to explain this case?
- d) What is the difference of **Aggregation** and **Composition**?
- e) What is the requirements if we want to make some classes to be generalized!

Answer

- a) **Class Diagram:** a tool from UML that modelling the classes which have been take from the abstraction of the real object in the solutions of the problems in Information System.
State Chart: a tool from UML which used to document the various conditions/circumstances that could happen to a class and any activity that may alter the conditions/circumstances.
- b) A life line defines a life time of class, grid, list, form, or window and some of them must be destroyed in the end of sequence. Where as an actor life line defines a life time of an actor in a sequence diagram and it can't be destroyed.
- c) ALT operators.
- d) Aggregation is a variant of the "has a" or association relationship; aggregation is more specific than association. Composition is a stronger variant of the "owns a" or association relationship; composition is more specific than aggregation.
- e) The classes must have the same characteristic(s) (whether attributes or operations).

Chapter 11

UML Design

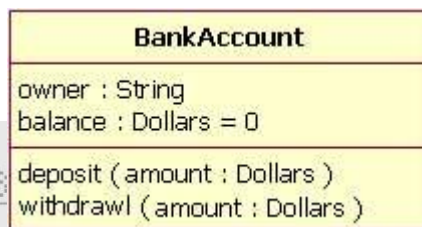


11.1 Class Diagram

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

- **Overview**

The class diagram is the main building block in object oriented modelling. It is used both for general conceptual modelling of the systematic of the application, and for detailed modelling translating the models into programming code. The classes in a class diagram represent both the main objects and or interactions in the application and the objects to be programmed. In the class diagram these classes are represented with boxes which contain three parts:



A class with three sections:

- The upper part holds the name of the class
- The middle part contains the attributes of the class
- The bottom part gives the methods or operations the class can take or under take

In the system design of a system, a number of classes are identified and grouped together in a class diagram which helps to determine the static relations between those objects. With detailed modeling, the classes of the conceptual design are often split in to a number of subclasses.

In order to further describe the behavior of systems, these class diagrams can be complemented by state diagram or UML state machine. Also instead of class diagrams object role modeling can be used if you just want to model the classes and their relationships.

- **Members**

UML provides mechanisms to represent class members, such as attributes and methods, and additional information about them.

- **Visibility**

To specify the visibility of a class member (i.e., any attribute or method) there are the following notations that must be placed before the member's name:

+ Public
- Private
Protected
~ Package
/ Derived
Underline Static

11.2 State Diagram

A **state diagram** is a type of diagram used in computer science and related fields to describe the behavior of systems. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. There are many forms of state diagrams, which differ lightly and have different semantics.

- **Overview**

State diagrams are used to give an abstract description of the behavior of a system. This behavior is analyzed and represented in series of events that could occur in one or more possible states. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system.

State diagrams can be used to graphically represent finite state machines. This was introduced by Taylor Booth in his 1967 book "Sequential Machines and Automata Theory". Another possible representation is the State transition table.

11.3 UseCaseDiagram

A **use case diagram** in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

Use Case diagrams are formally included in two modeling languages defined by the OMG: the Unified Modeling Language (UML) and the Systems Modeling Language (SysML).

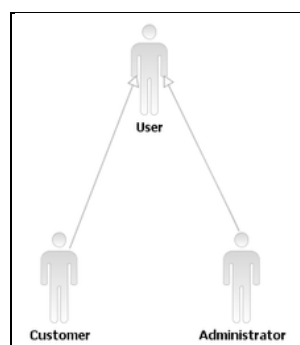
- **Diagram Building Block**

Interaction among actors is not shown on the use case diagram. If this the system or use case boundaries should be re-examined. Alternatively, interaction among actors can be part of the assumptions used in the use case.

○ Use cases: A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse.

○ Actors: An actor is a person, organization, or external system that plays a role in one or more interactions with the system.

○ System boundary boxes (optional): A rectangle is drawn around the use cases, called the system boundary box, to indicate the scope of system. Anything within the box represents functionality that is inscope and anything outside the box is not.



Actor Inheritance

- **Actor Generalization**

One popular relationship between Actors is Generalization/Specialization. This is useful in defining overlapping roles between actors. The notation is a solid line ending in a hollow triangle drawn from the specialized to the more general actor.

- **Use Case Relationships**

Four relationships among use cases are used often in practice:

- Include

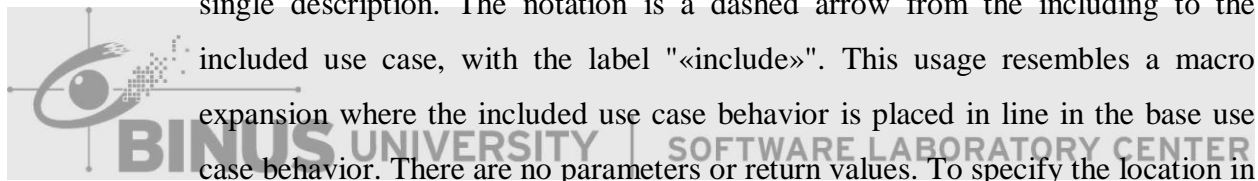
In one form of interaction, a given use case may *include* another. "Include is a Directed Relationship between two use cases, implying that the behaviour of the included use case is inserted in to the behaviour of the including use case".

The first use case often depends on the outcome of the included use case. This is useful for extracting truly common behaviors from multiple use cases in to a single description. The notation is a dashed arrow from the including to the included use case, with the label "«include»". This usage resembles a macro expansion where the included use case behavior is placed in line in the base use case behavior. There are no parameters or return values. To specify the location in a flow of events in which the base use case includes the behavior of another, you simply write *include* followed by the name of use case you want to include, as in the following flow for *track order*.

- Extend

In another form of interaction, a given use case (the extension) may *extend* another. This relationship indicates that the behaviour of the extension use case may be inserted in the extended use case under some conditions. The notation is a dashed arrow from the extension to the extended use case, with the label "«extend»". The notes or constraints may be associated with this relationship to illustrate the conditions under which this behaviour will be executed.

Modellers use the «extend» relationship to indicate use cases that are "optional" to the base use case. Depending on the modeler's approach "optional" may mean



"potentially not executed with the base use case" or it may mean "not required to achieve the base use case goal".

- Generalization

In the third form of relationship among use cases, a *generalization/specialization* relationship exists. A given use case may have common behaviours, requirements, constraints, and assumptions with a more general use case. In this case, describe them once, and deal with it in the same way, describing any differences in the specialized cases. The notation is a solid line ending in a hollow triangle drawn from the specialized to the more general use case (following the standard generalization notation).

- Associations

Associations between actors and use cases are indicated in use case diagrams by solid lines. An association exists whenever an actor is involved with an interaction described by a use case. Associations are modelled as lines connecting use cases and actors to one another, with an optional arrow head on one end of the line. The arrow head is often used to indicate the direction of the initial invocation of the relationship or to indicate the primary actor within the use case. The arrow head simply control flow and should not be confused with data flow.

11.4 Sequence Diagram

A **sequence diagram** in Unified Modeling Language (UML) is a kind of interaction diagram that show processes operate with one another and in what order. It is a construct of a Message Sequence Chart.

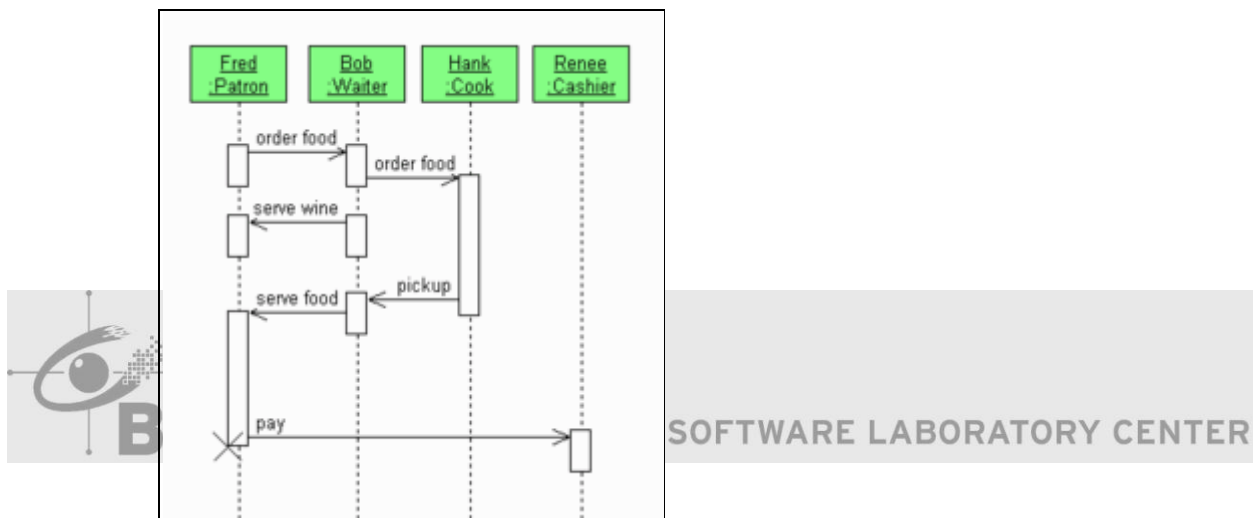
Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

- **Overview**

A sequence diagram shows, as parallel vertical lines (*life lines*), different processor objects that live simultaneously, and, as horizontal arrows, the messages exchanged

between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

For instance, the UML 1.x diagram on the right describes the sequences of messages of a (simple) restaurant system. This diagram represents a Patron ordering food and wine, drinking wine then eating the food, and finally paying for the food. The dotted lines extending down wards indicate the time line. Time flows from top to bottom. The arrows represent messages (stimuli) from an actor or object to other objects. For example, the Patron sends message 'pay' to the Cashier. Half arrows indicate a synchronous method calls.

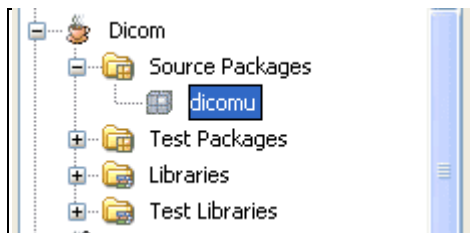


SimpleRestaurantSequenceDiagram

11.5 Generate Code

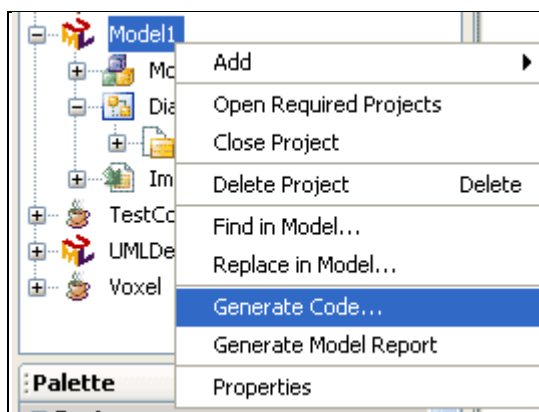
In this part, Its going to show **how to generate code from existing diagram** so that it can reduce a lot of work if you have an well-defined diagram. Also, it can make sure that the coding structure of your project will conform as in a diagram.

1. Suppose I have an existing class DicomImage in Model 1 Project as in figure below.

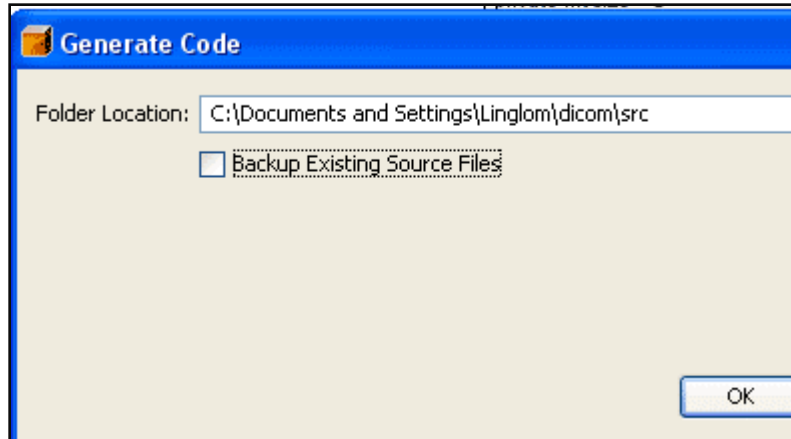


DicomImage
<i>Attributes</i>
private int width = 0
private int height = 0
private int size = 0
private int highBit = 0
private int bitsStored = 0
private int bitsAllocated = 0
private boolean pixelRepresentation = true
private int samplesPerPixel = 0
private String photometricInterpretation = "MONOCHROME2"
private int numberOfFrames = 0
private byte pixelData[0..*]
private int thumbWidth = 128
private int thumbHeight = 128
<i>Operations</i>
public DicomImage(DicomInformation di)
public Image getImage()
public ArrayList<Image> getMultipleImages()
private ColorModel grayColorModel()
private byte[0..*] to8PerPix(byte pixData[0..*])
private byte[0..*] signedTo8PerPix(byte pixData[0..*])
private Image scaleImage()
<u>public boolean hasAlpha(Image image)</u>
<u>public BufferedImage toBufferedImage(Image image)</u>
public Image createThumbnail(Image image)

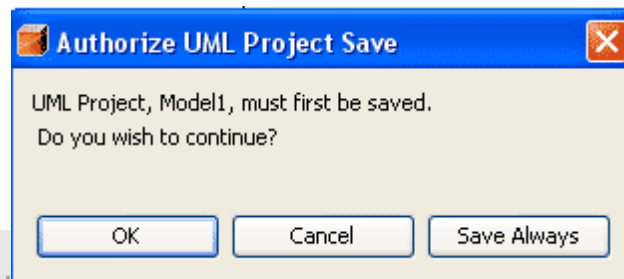
- Before we start, let choose where to save source code file. Let say, I want to put source code in Source Packages, Dicomproject. (“C:\Documents and Settings\Linglom\Dicom\src”)
- Now, let generate code. Right click on Model 1 Project, select Generate Code...



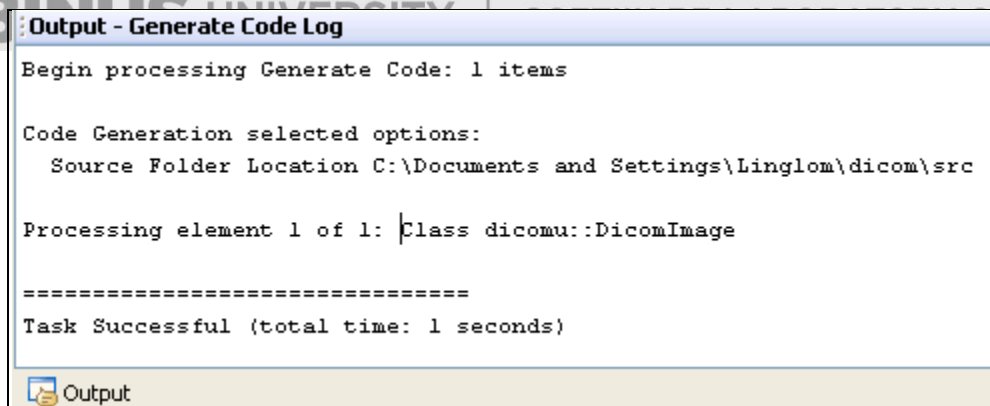
- Select destination to put the source code, I use location from2. Click OK. If you don't want to backup file, just uncheck the box.



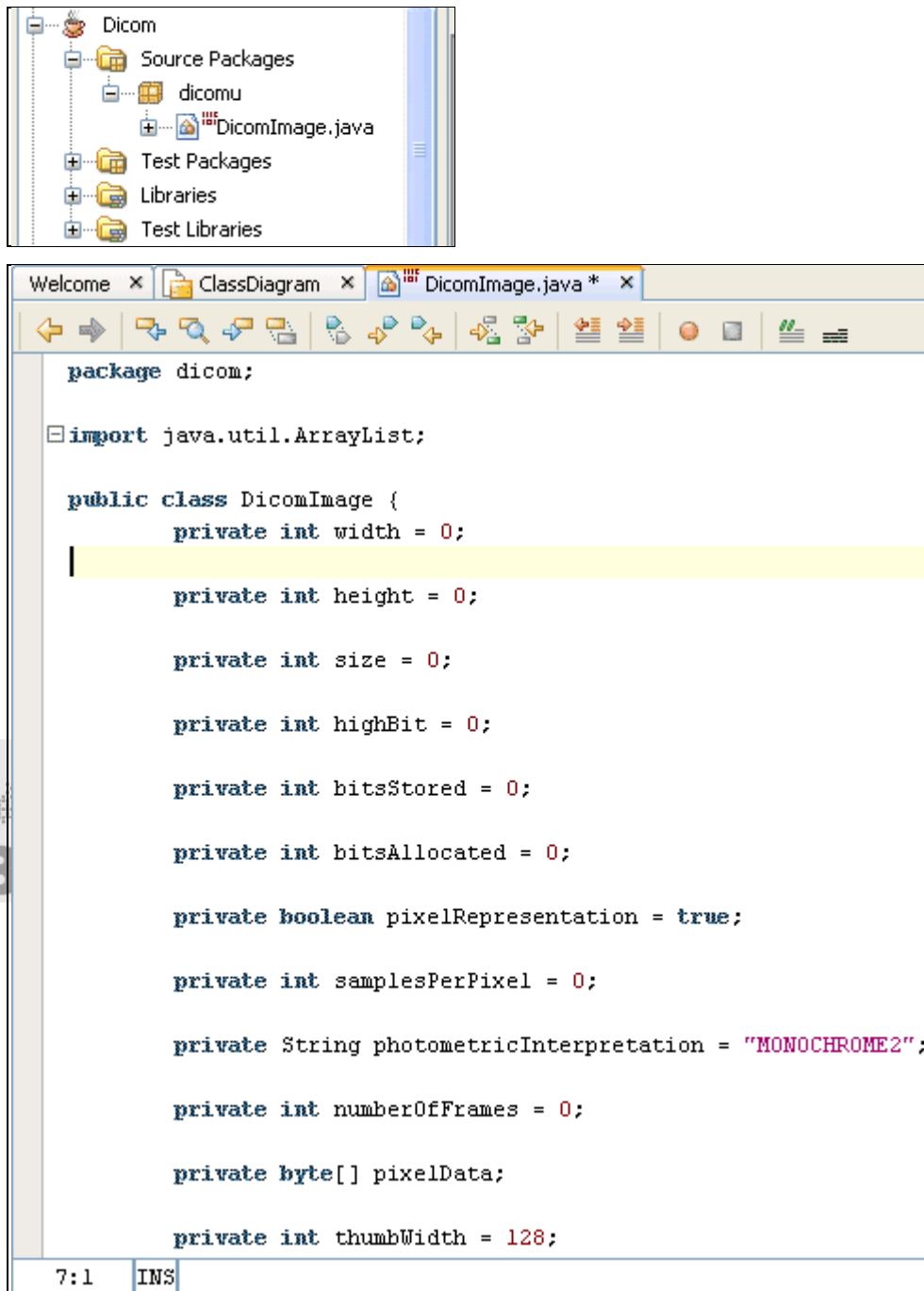
5. If you have just created a diagram, it will ask to save it first. Click OK.



6. When code are generating, you will see the progress in the output window.



7. When it finished, you will see the new file in destination Project was created. Try open it and you will see some source code that it has generated for you.

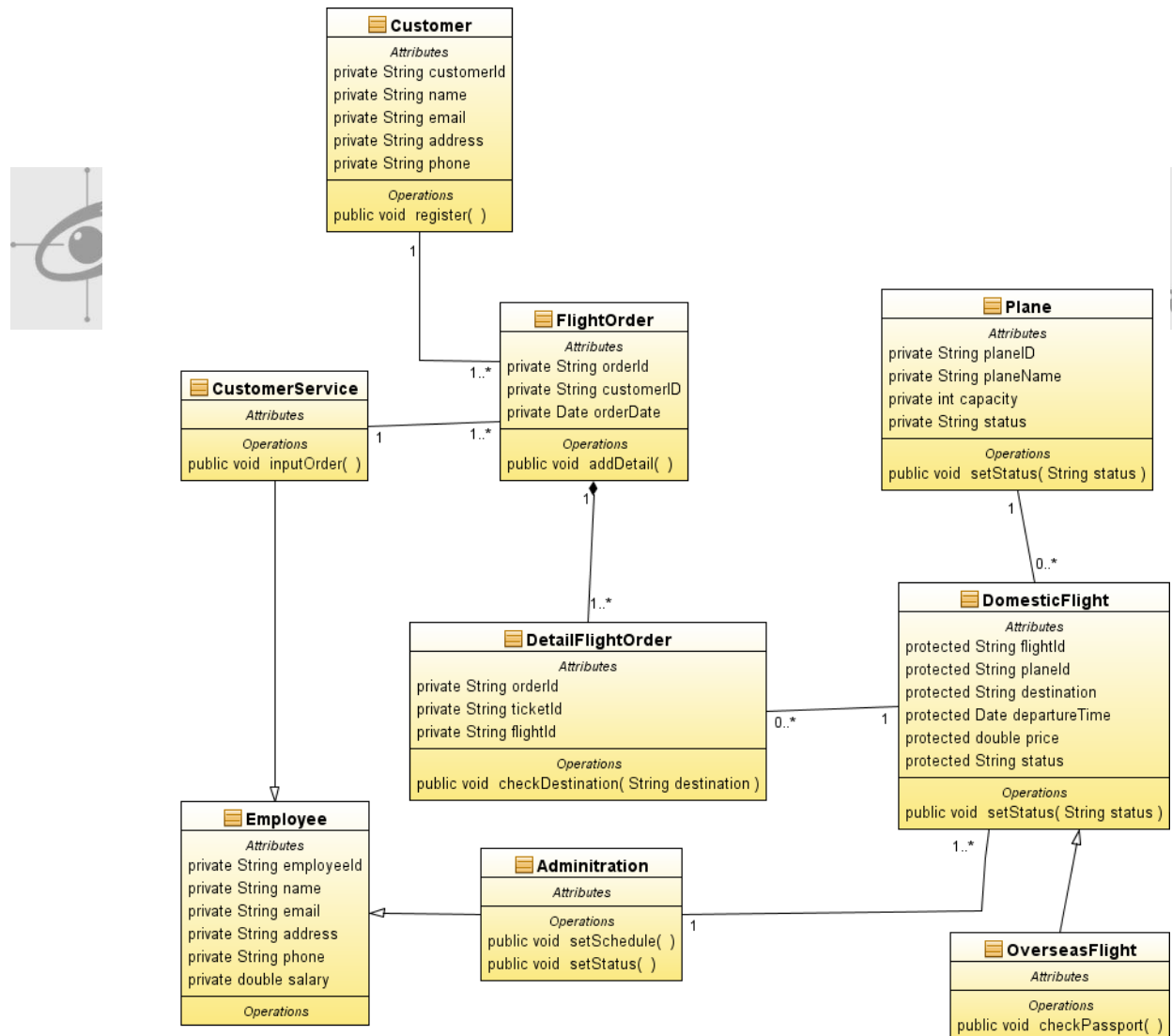


This tool is useful if you have a clearly well-defined diagram. But keep in mind that if your diagrams contain some errors, your generated code may effect from that, too.

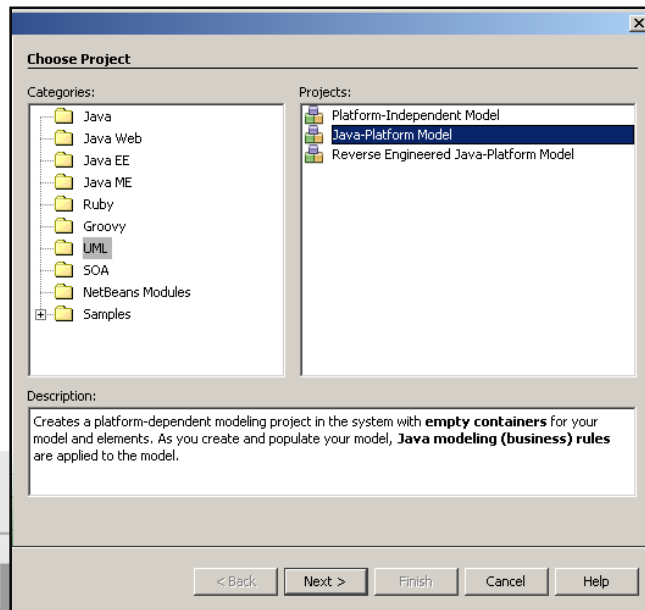
11.6 Exercise

Tiger Air is a flight service company. They have a problem with their business, because it is not computerized. Their business process is like, customer comes to this company and can register their information, so they can order the flight ticket(s). When they order the flight ticket(s), they must pay the ticket(s) based on the destination and the flight type (Domestic Flight and Overseas Flight). When customer orders the ticket(s), Customer Service inputs the order to the their database. The flight schedule (whether Domestic Flight or Overseas Flight) is set by the Administration Division. Administration Division can set the status of the flight schedule too. It can be delayed, depart, or cancelled.

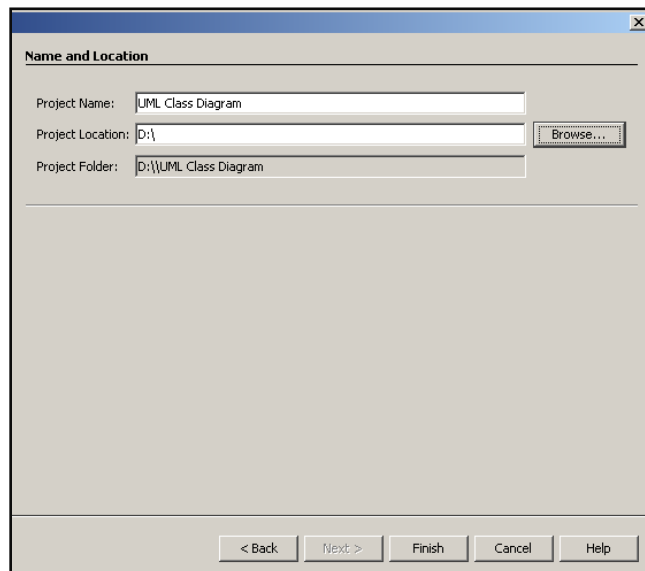
Exercise 01 – Make a Class Diagram



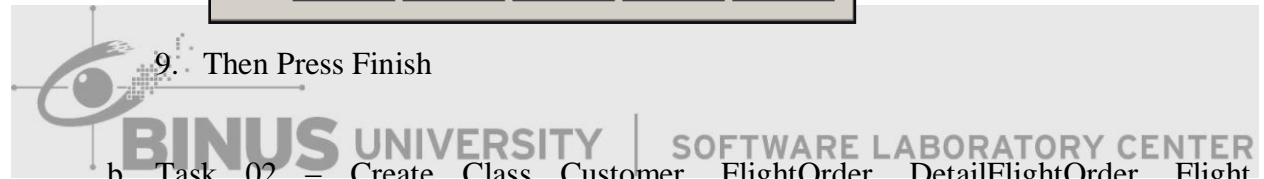
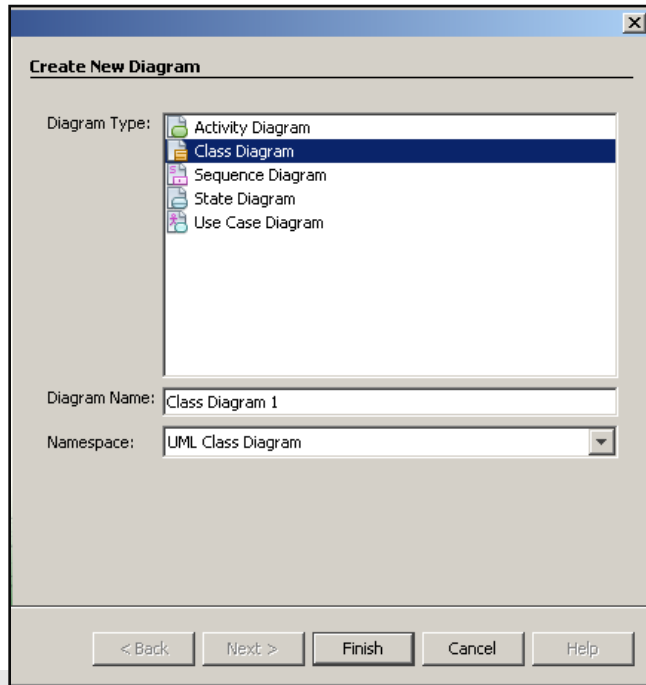
- a. Task 01 – Create Project in NetBeans
 1. Run NetBeans from Start Menu
 2. Open Menu File -> New Project
 3. On New Project Window, choose categories UML the choose Projects Java-Platform Model



4. Then Press Next
5. Enter your project name “UML Class Diagram”
6. Browse Location of your project in D:\



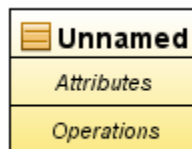
7. Then Press Finish
8. At Diagram Type, choose Class Diagram



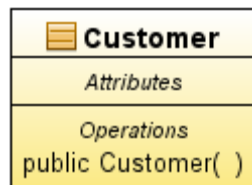
9. Then Press Finish

b. Task 02 – Create Class Customer, FlightOrder, DetailFlightOrder, Flight, GarudaFlight, LionFlight

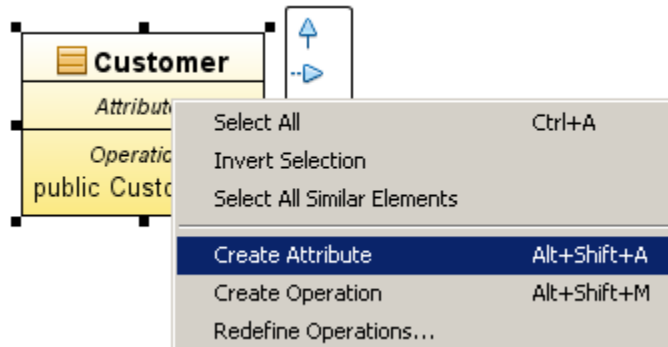
1. Drag component Class from Pallette Basic to your worksheet



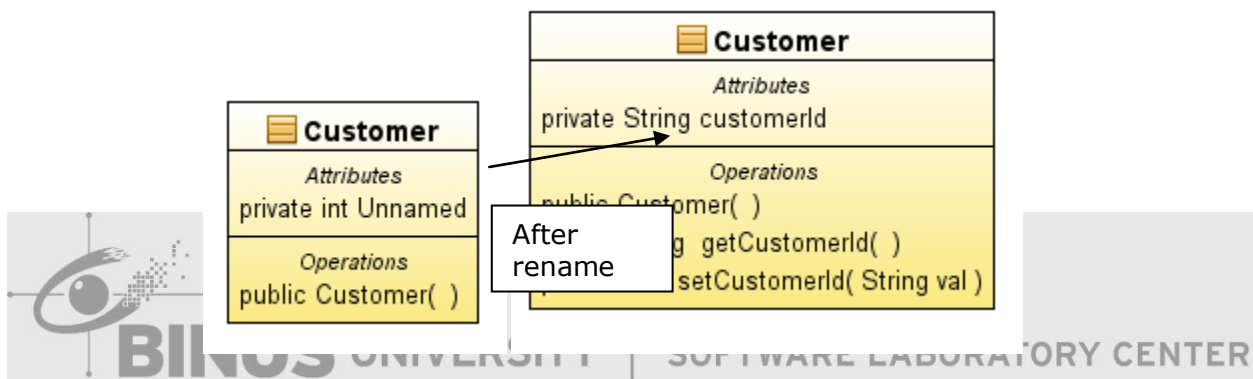
2. Double Click on the text “Unnamed”, then rename the class



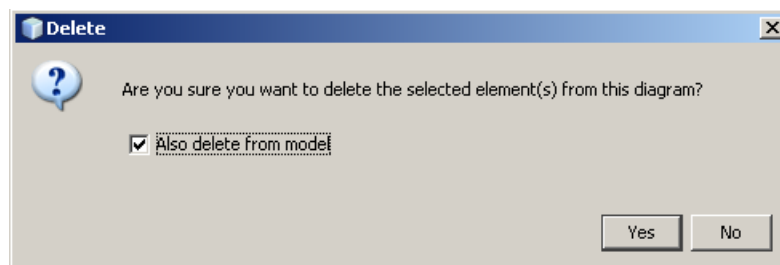
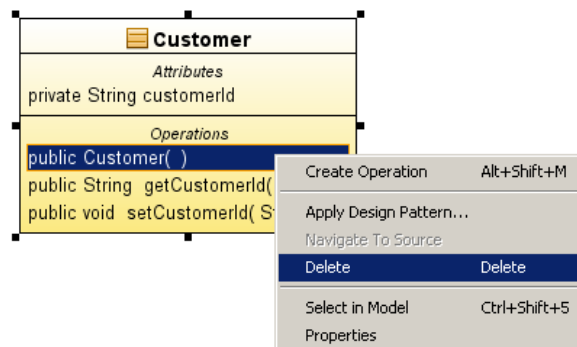
3. Create attribute from each Class by right click on the class then choose Create Attributes.



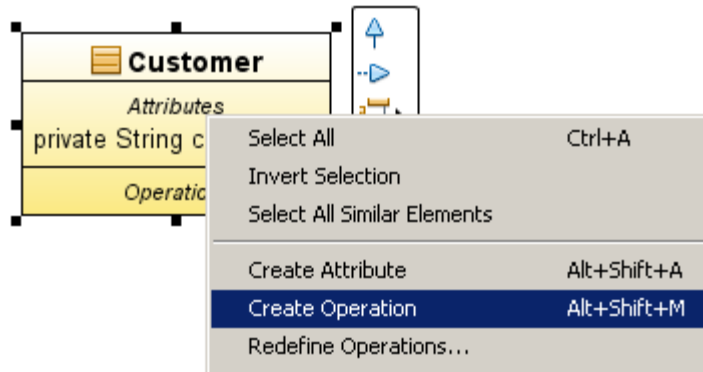
4. Then Double Click the attribute do you have make it then rename it.



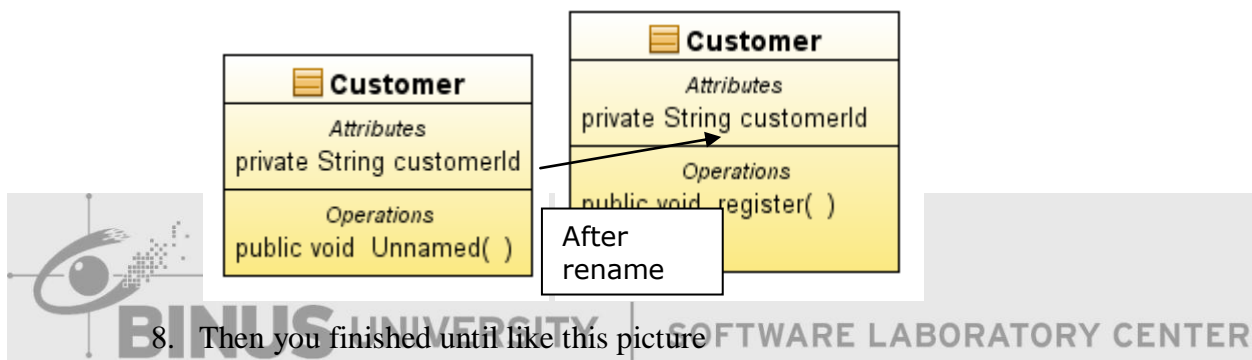
5. Delete the operation are made of the attributes and if it appears the window is deleted, then the check list “Also delete from the model”, then select yes.



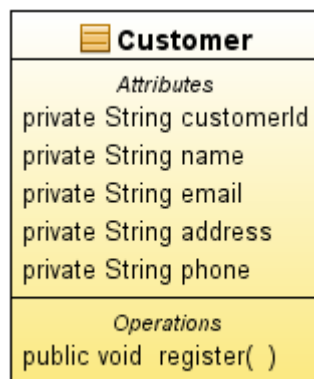
6. Then Create the operation from each Class by right click on the class then choose Create Operation



7. Then Double Click the operation do you have make it then rename it.



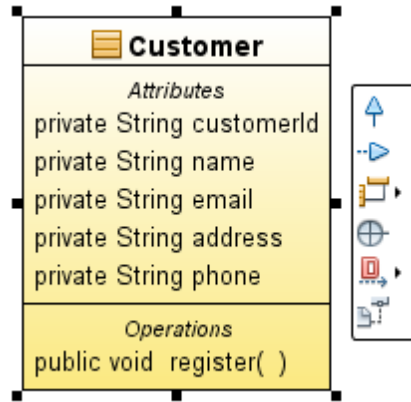
8. Then you finished until like this picture



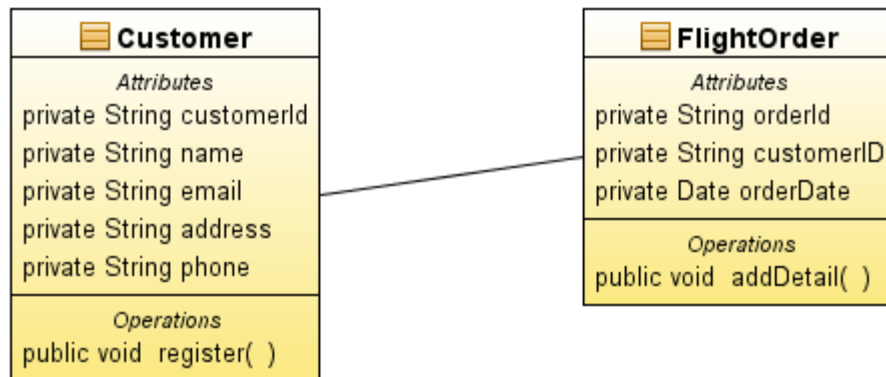
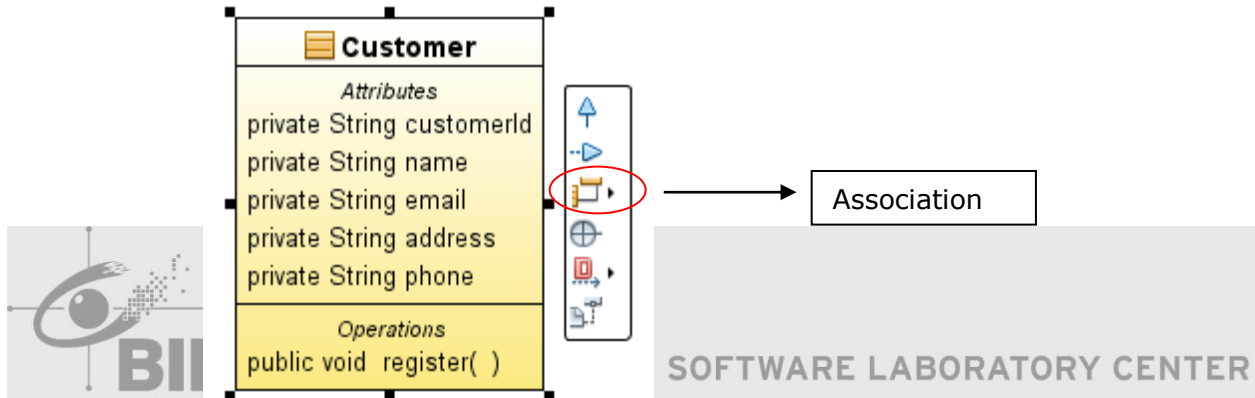
9. Then you finished class for FlightOrder, Plane, DetailFlightOrder, DomesticFlight, OverseasFlight

c. Task 03 – Create the Association

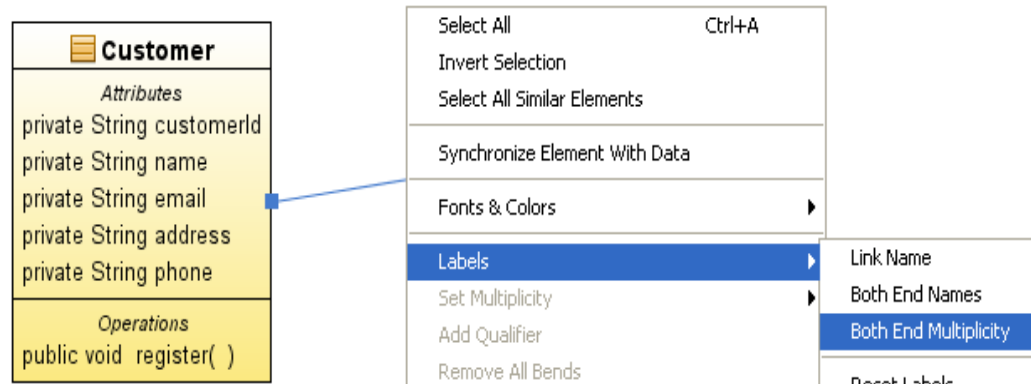
1. For relationship from Class Customer to Class FlightOrder, choose Class Customer first until show tools for the relationship



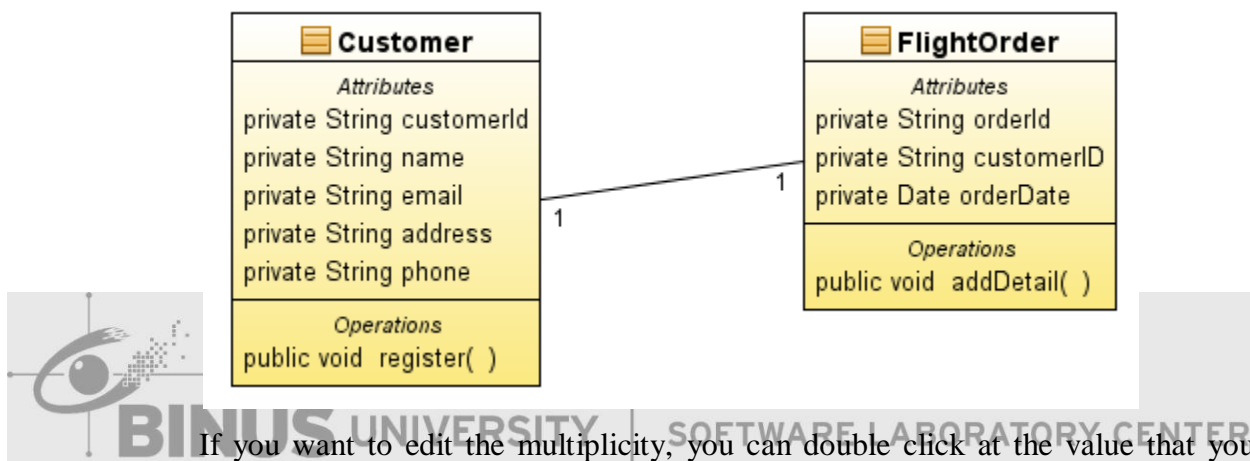
2. Then choose association and drag it to Class FlightOrder



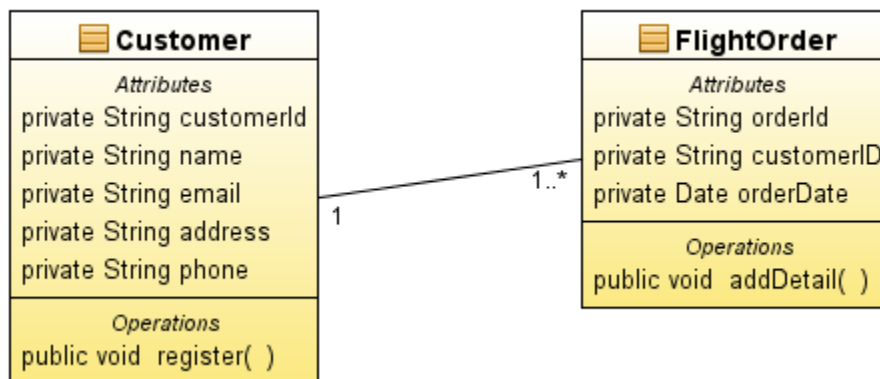
Then you can give the multiplicity by right click at the relationship, then choose Labels → Both End Multiplicity



Then it will show the multiplicity from Class Customer to Class FlightOrder.



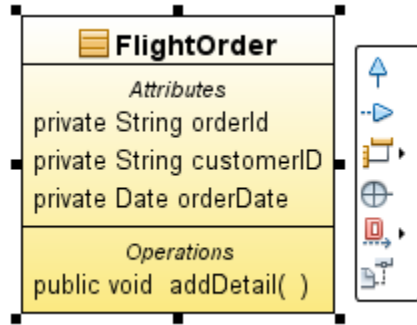
If you want to edit the multiplicity, you can double click at the value that you want to edit.



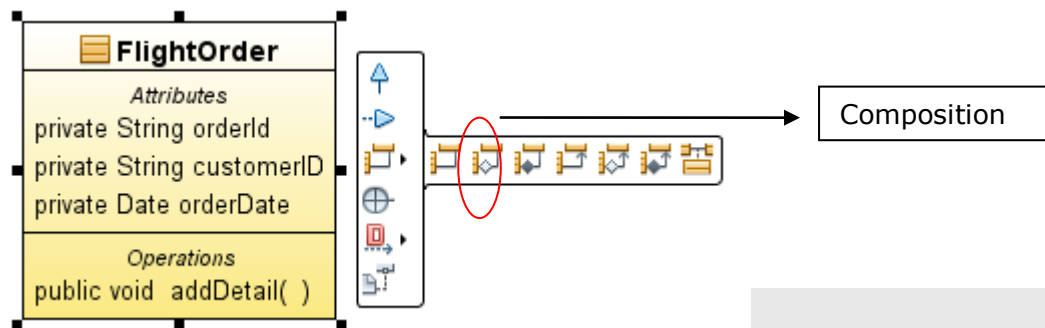
Then you can make classes that use association relationship.

d. Task 04 – Create Composite

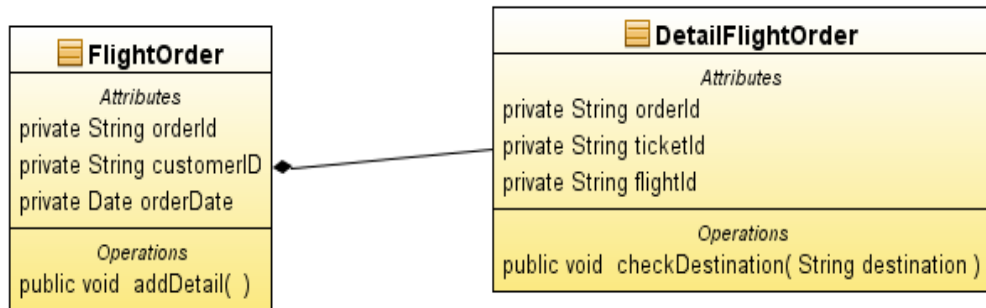
1. For relationship from Class FlightOrder to Class DetailFlightOrder, choose Class FlightOrder first until show tools for the relationship



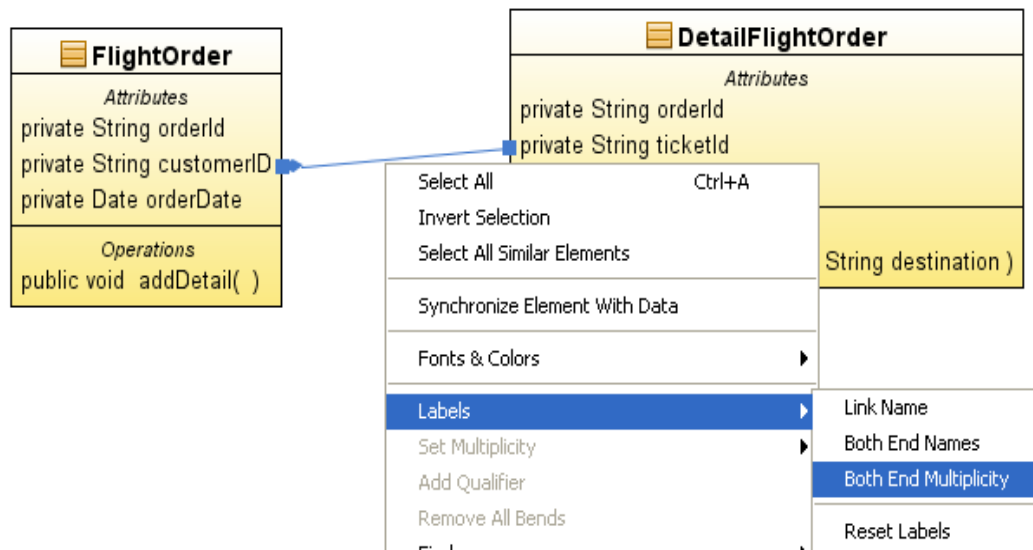
2. Composition position it was in the association. So, if you want to change it, click the arrow until show like this picture



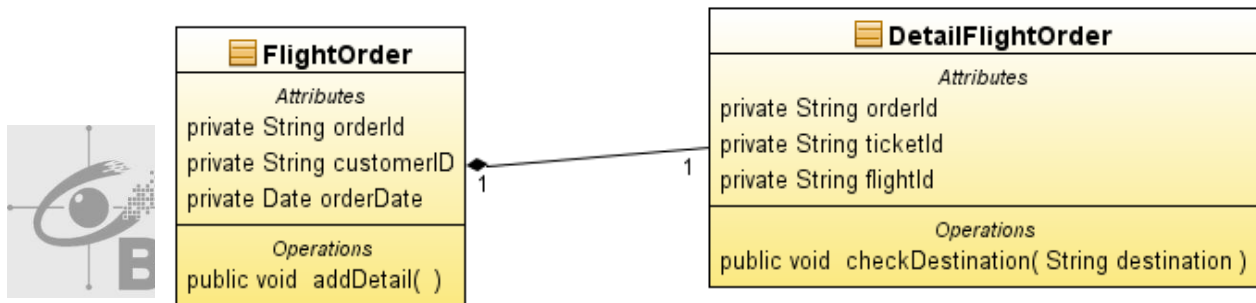
Then drag it to the Class DetailFlightOrder



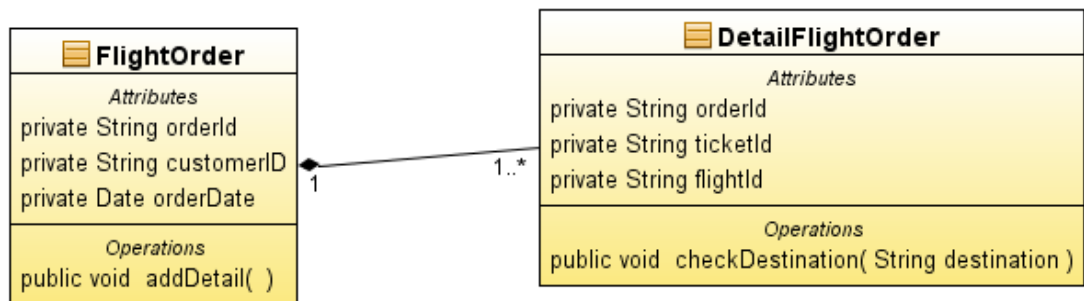
Then you can give the multiplicity by right click at the relationship, then choose Labels → Both End Multiplicity



Then it will show the multiplicity from Class FlightOrder to Class DetailFlightOrder.



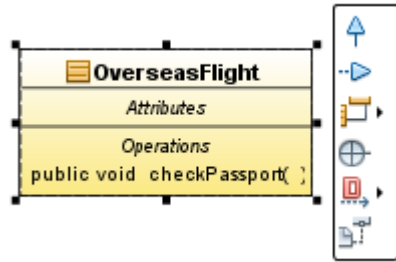
If you want to edit the multiplicity, you can double click at the value that you want to edit.



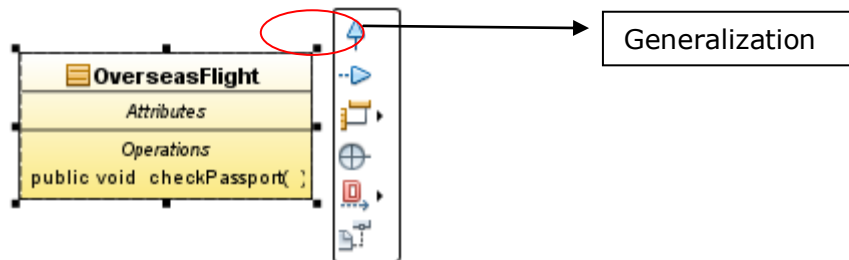
Then you can make classes that use composition relationship.

e. Task 05 – Create Generalization

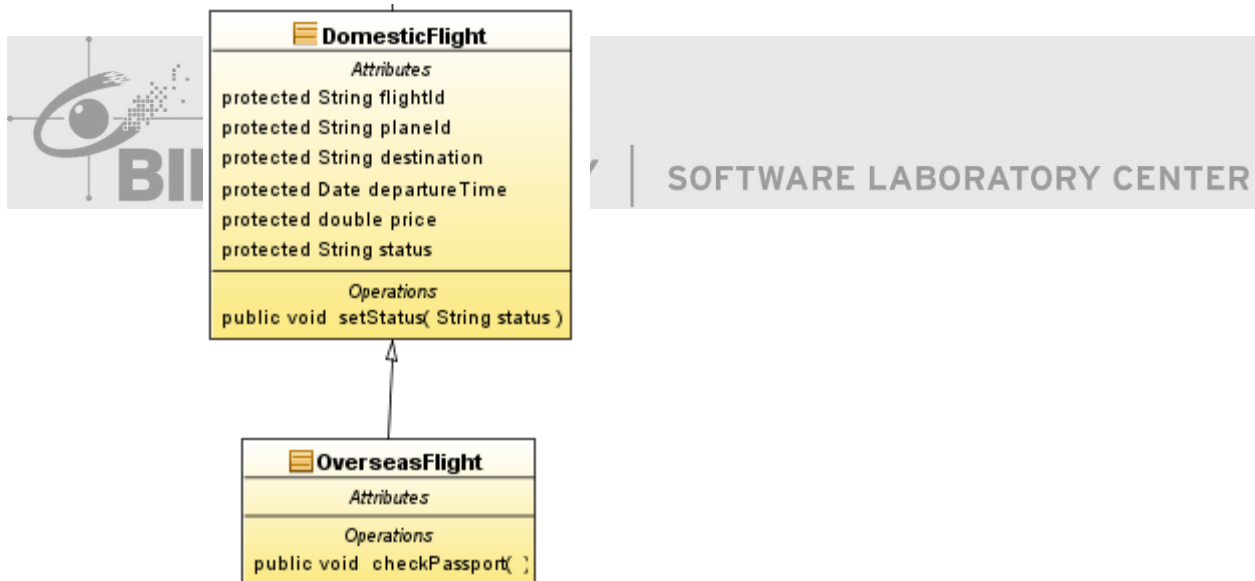
1. For relationship from Class DomesticFlight to Class OverseasFlight, choose Class OverseasFlight first until show tools for the relationship



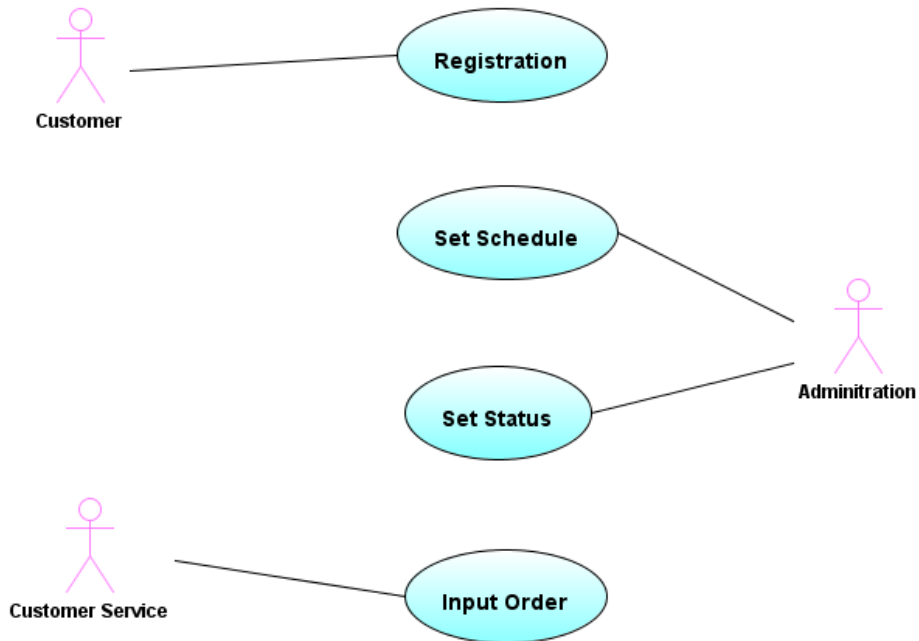
2. Then choose generalization



And drag it to Class DomesticFlight

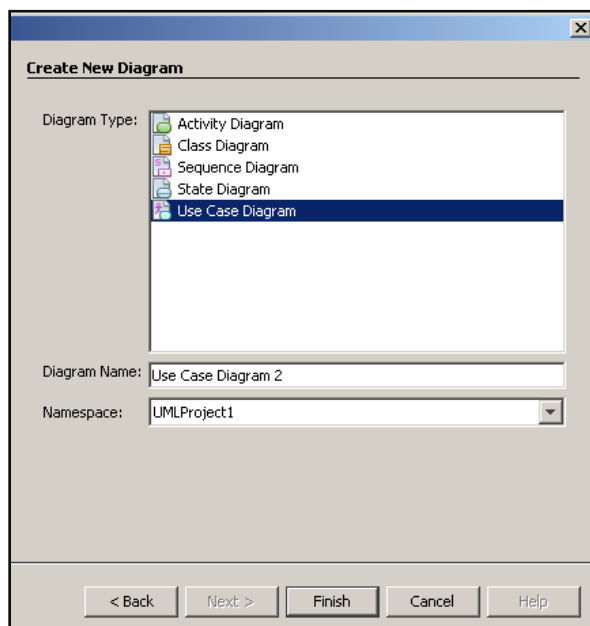


Exercise 02 – Make a Use Case Diagram



a. Task 01 – Create Use Case Diagram

1. Right click on the project
2. Then choose New → Diagram
3. Then at Diagram Type, choose Use Case Diagram
4. Then Press Finish



b. Task 02 – Create Actor

1. Drag the Actor from Pallete Basic to your worksheet



2. Then rename the text “Unnamed” with double click on the text



3. Then make the Actor for Customer Service and Administration

c. Task 03 – Create Use Cases

1. Drag the Use Cases from Pallete Basic to your worksheet



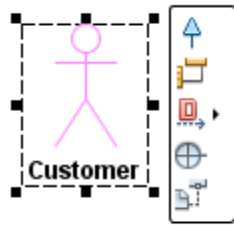
2. Then rename the text “Unnamed” with double click on the text



3. Then make other Use Cases

d. Task 04 – Create Relationship

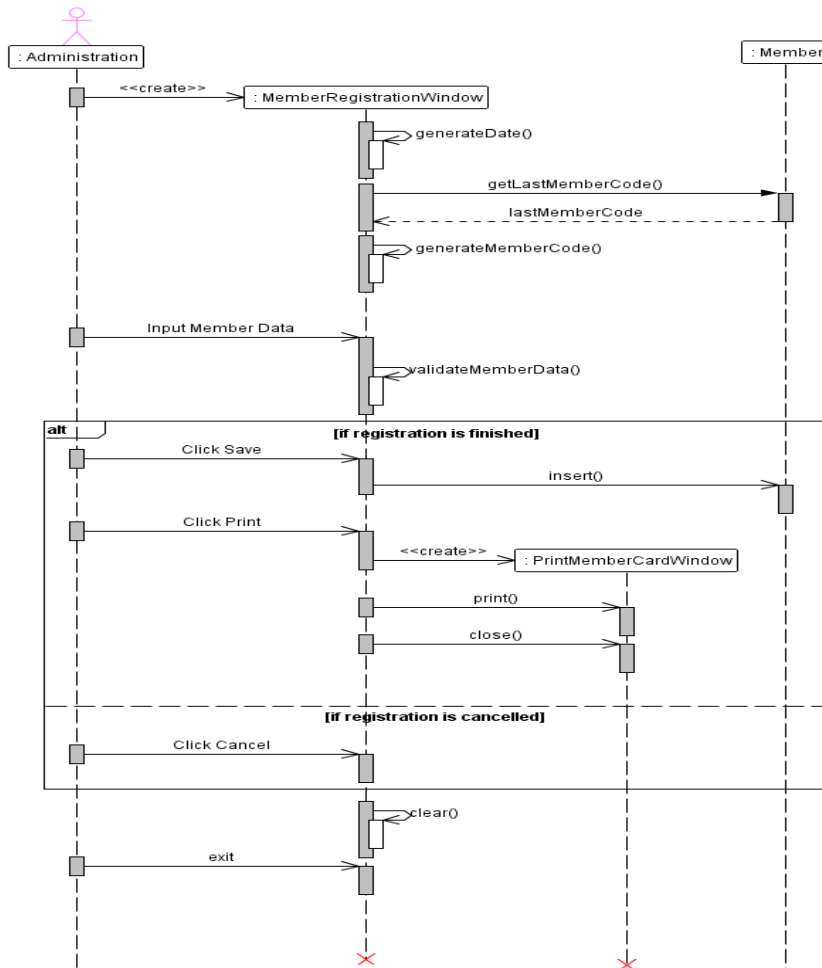
1. Click on your actor that you want to make relationship until show this tools



2. Then choose Association and drag to "Registration" use cases



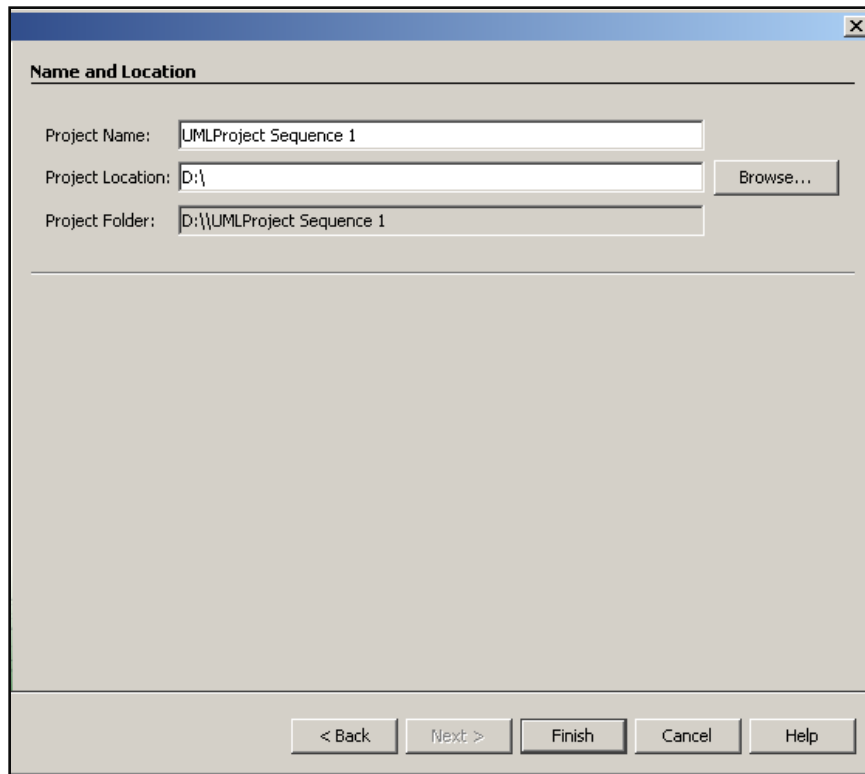
3. Then the other Association like above example



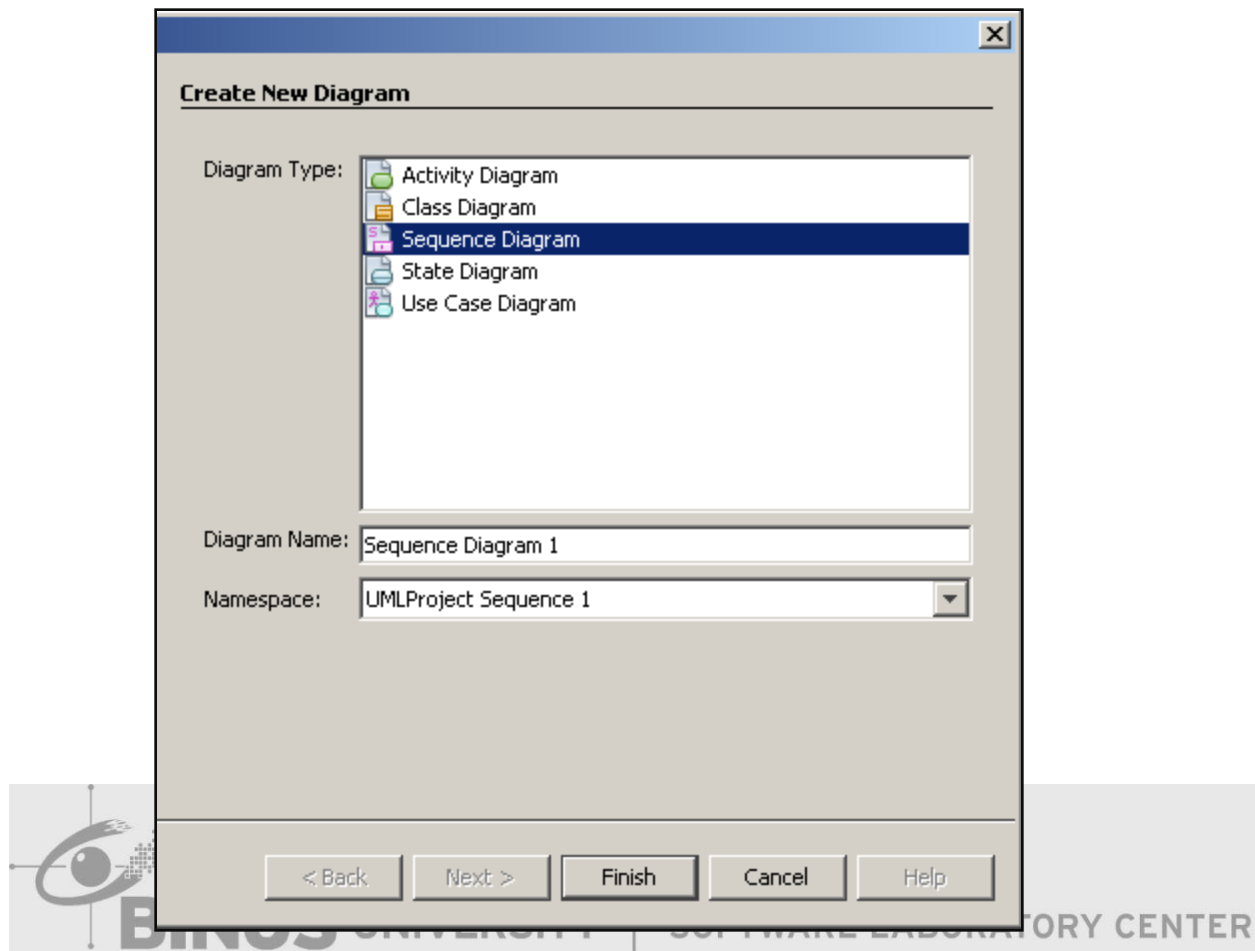
LABORATORY CENTER

a. Task01-CreateNewProjectSequenceDiagram

1. RunNetBeansfromStartMenu
2. OpenMenuFile->NewProject
3. OnNewProjectWindow,choosecategoriesUMLthechooseProjectsJava-PlatformModel
4. ThenPressNext
5. Enteryourprojectname“UMLSequenceDiagram1”
6. BrowseLocationofyourprojectinD:\
7. ThenPressFinish
8. AtDiagramType,chooseSequenceDiagram
9. ThenPressFinish

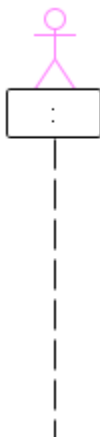


The image shows a dialog box titled "Name and Location" with a close button (X) in the top right corner. It contains three input fields: "Project Name" with the text "UMLProject Sequence 1", "Project Location" with "D:\", and "Project Folder" with "D:\\UMLProject Sequence 1". A "Browse..." button is positioned to the right of the "Project Location" field. At the bottom of the dialog, there are five buttons: "< Back", "Next >", "Finish", "Cancel", and "Help".

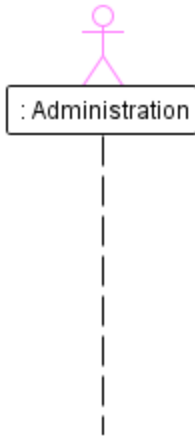


b. Task02-CreateActorLifeline

1. DragtheActorLifelinefromPallette–Basictoyourworksheet

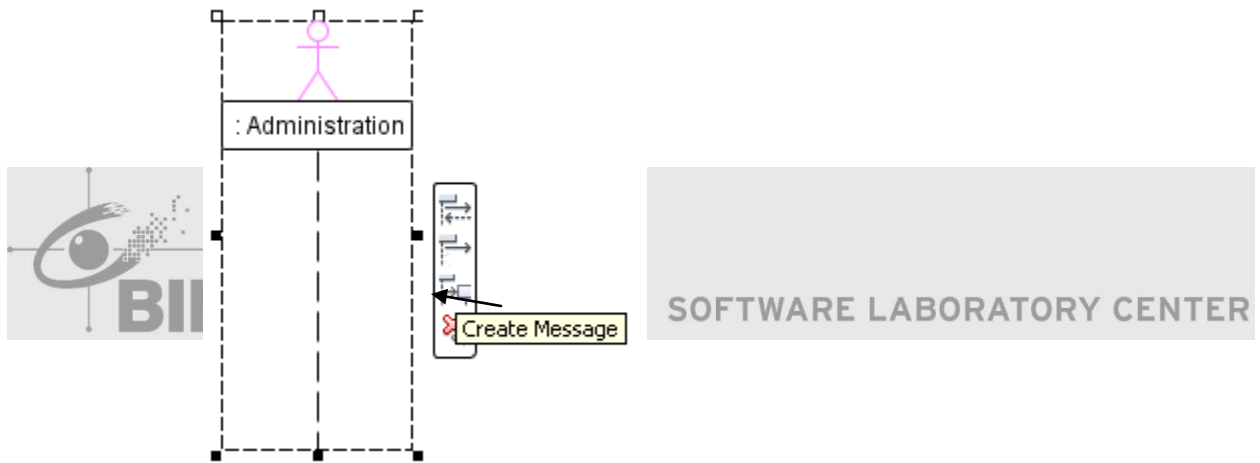


2. ThenyoucangivethenametoyourActorLifelinewithdoubleclickontherightofthecolon (:):thengiveitthename

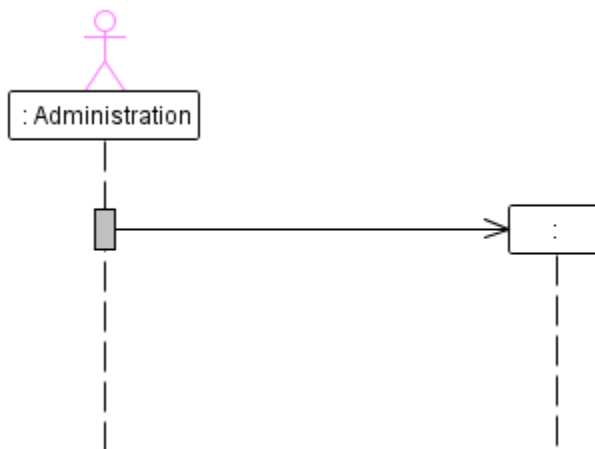


c. Task03-CreateMessage

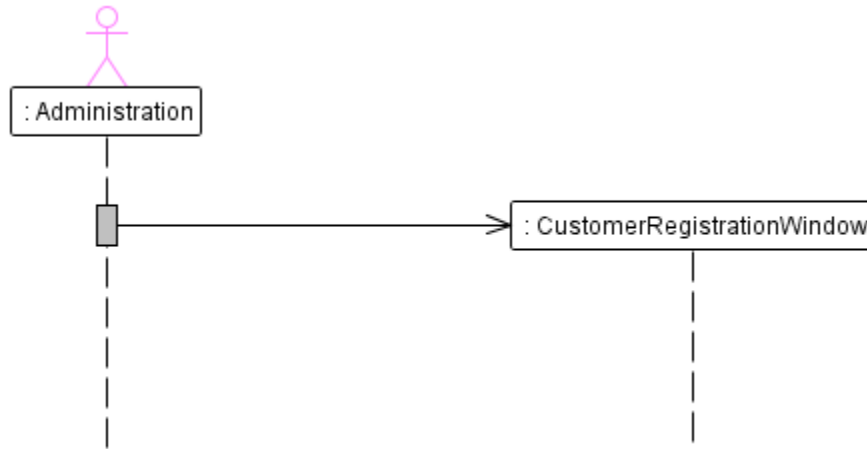
1. OnceclickonyourActorLifelineuntilshowthistools.



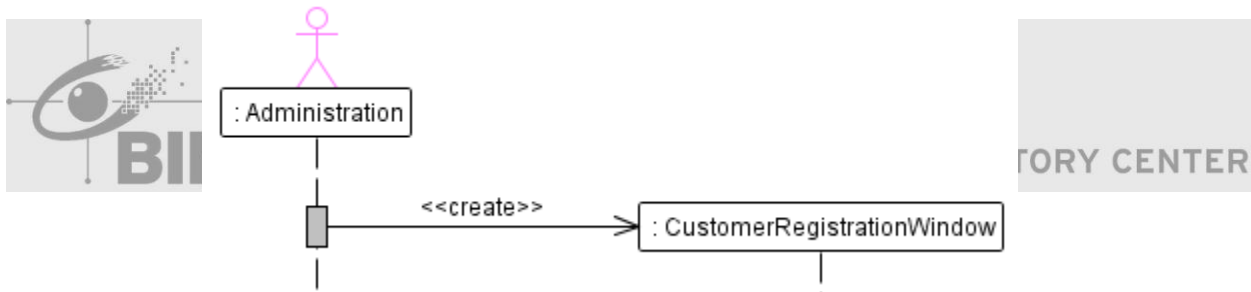
2. Thendragittotherightsideuntillikethis



- Then give the name for your message with double click at the right of the colon(:) then give its name

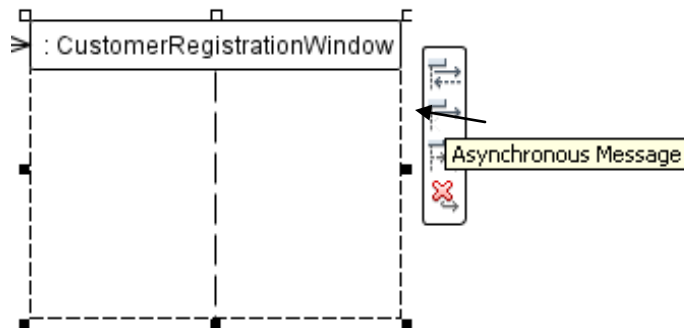


- Then give it the name for your message with click on the arrow of your message, the look at the Properties of the Message. Fill the name property

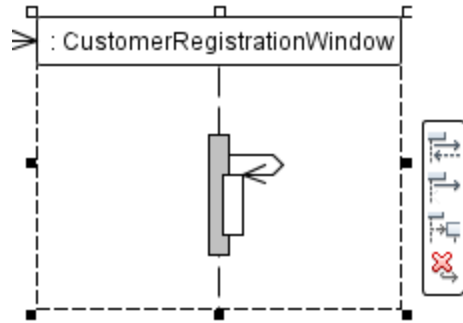


d. Task 04 – Create Recursive Asynchronous Message

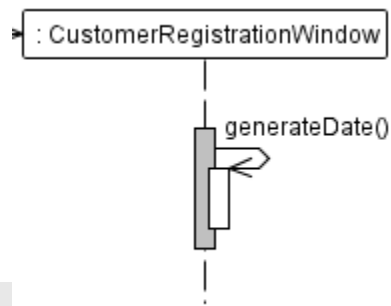
- Once click on your life line then click create Asynchronous Message



- Then and point it to itself.



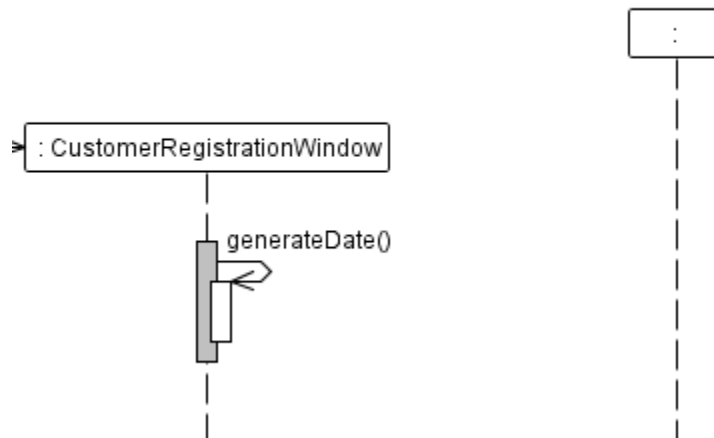
3. Then give it the name for your message with click on the arrow of your message, the look at the Properties of the Message. Fill the name property



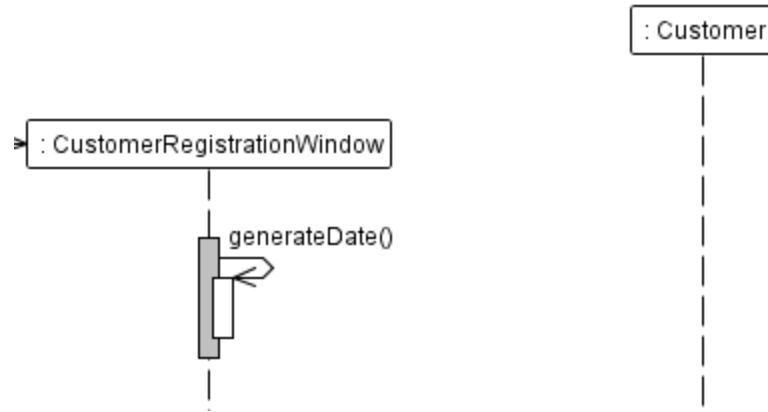
e. Task 05 - Create Synchronous Message

1. Drag the life line to your worksheet

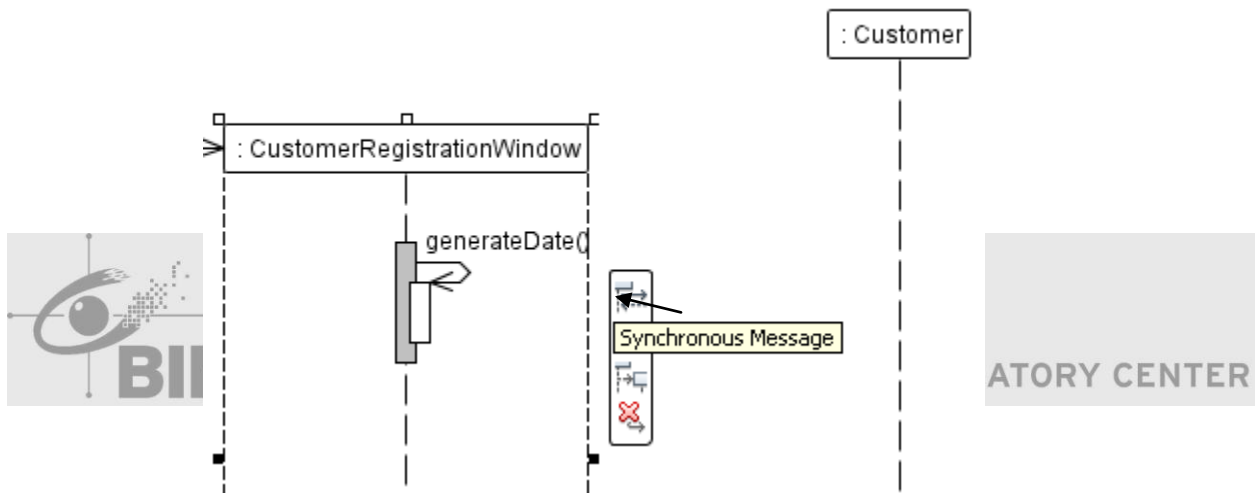
BINUS UNIVERSITY | SOFTWARE LABORATORY CENTER



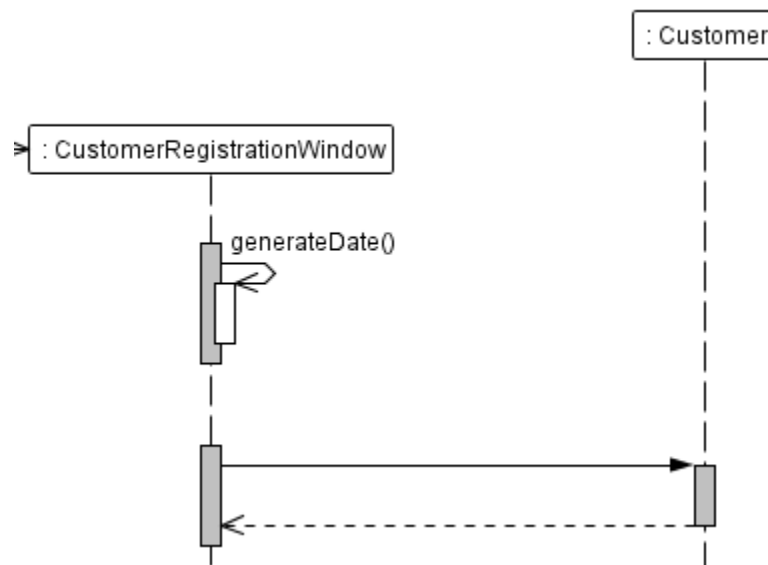
2. Then give it the name for your life line with double click on the right of the colon(:) then give its name



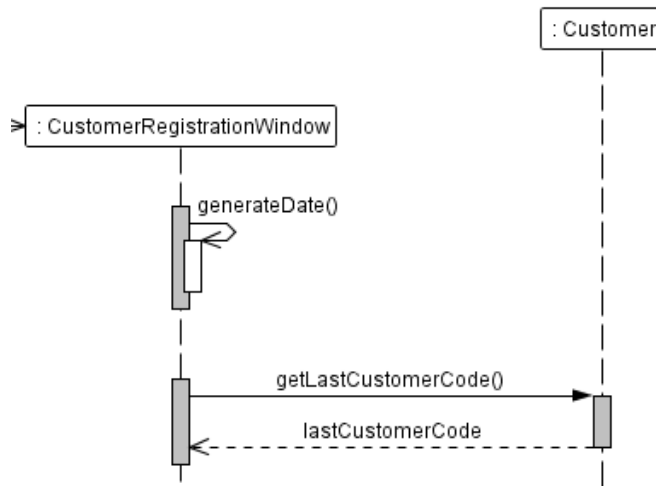
3. Then from Customer Registration Window, choose Synchronous Message



4. Then drag it to Customer Life line



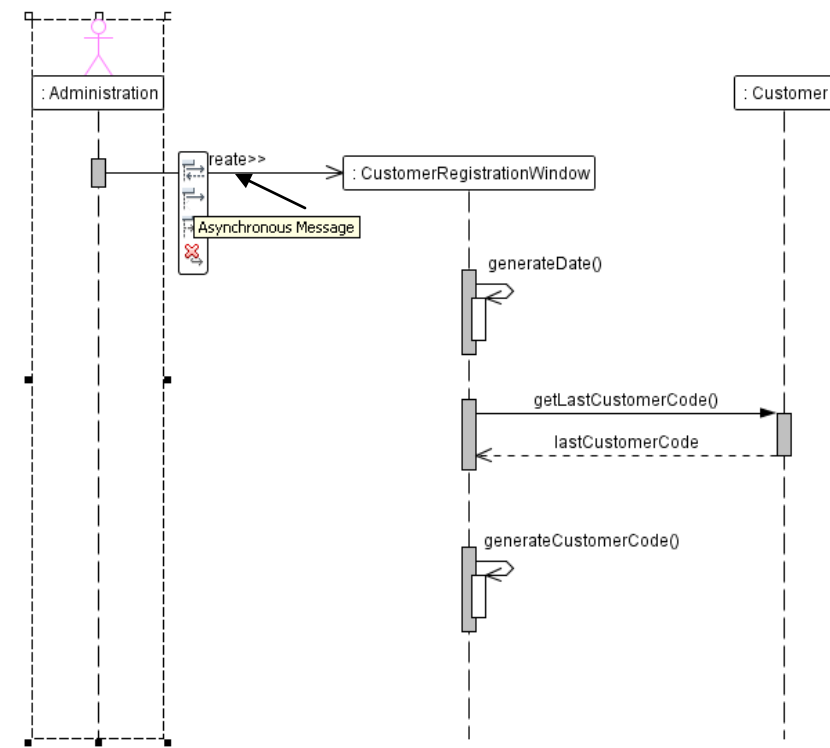
5. Then give it the name for your message with click on the arrow of your message, the look at the Properties of the Message. Fill the name property



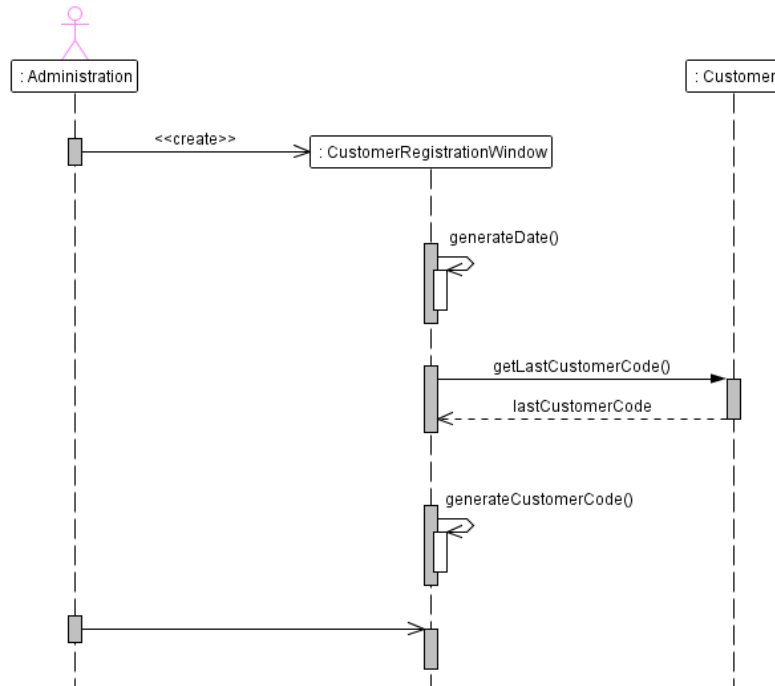
Remember that the function(the arrow with the black head) must be named with the parenthesis(e.g: getLastCustomerCode()) and the return value must be named without the parenthesis(e.g: lastCustomerCode).

f. Task 06 – Create Asynchronous with no Recursive Message

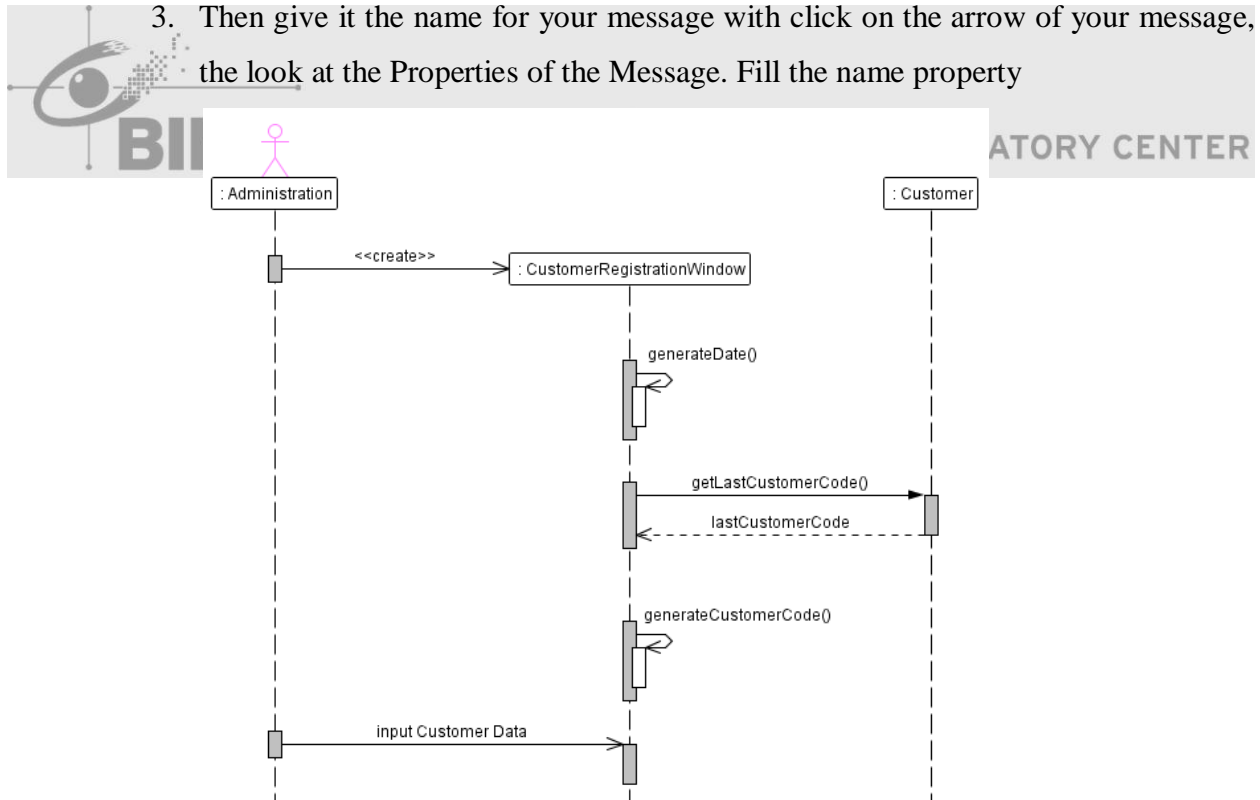
1. Once click on your Actor Life line then click create Asynchronous Message



2. Then and point it to Customer Registration Window

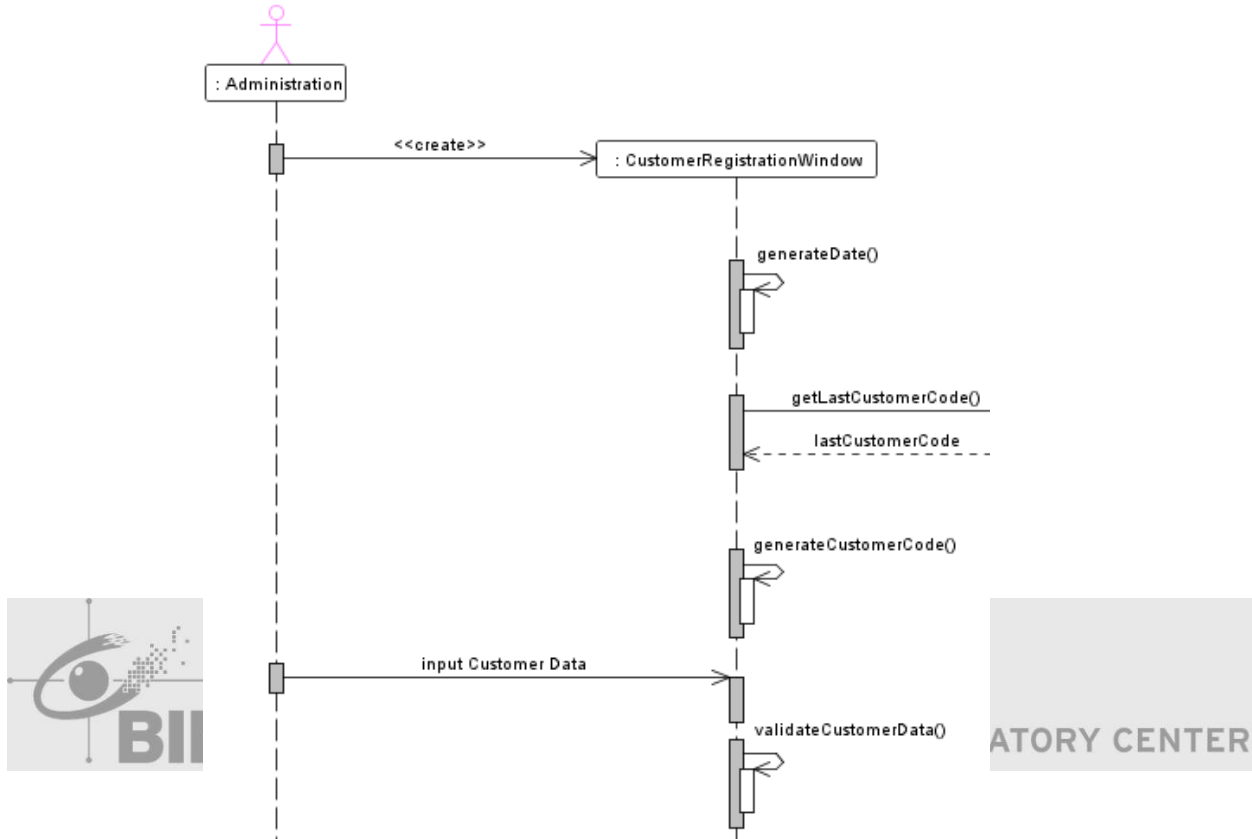


3. Then give it the name for your message with click on the arrow of your message, the look at the Properties of the Message. Fill the name property



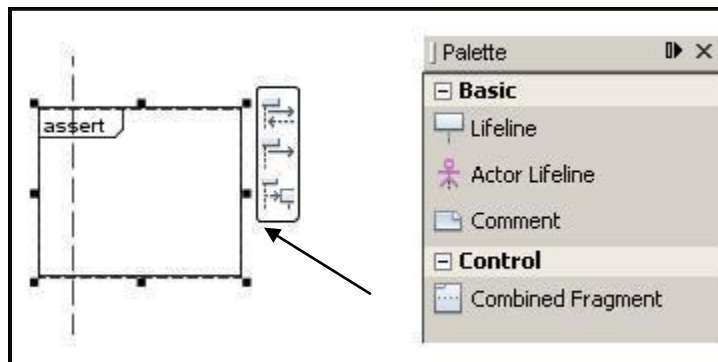
g. Task 07 – Create Recursive Asynchronous Message for validateCustomerData()

1. Once click on your customer Registration Window life line then click create Asynchronous Message then and point it to itself.

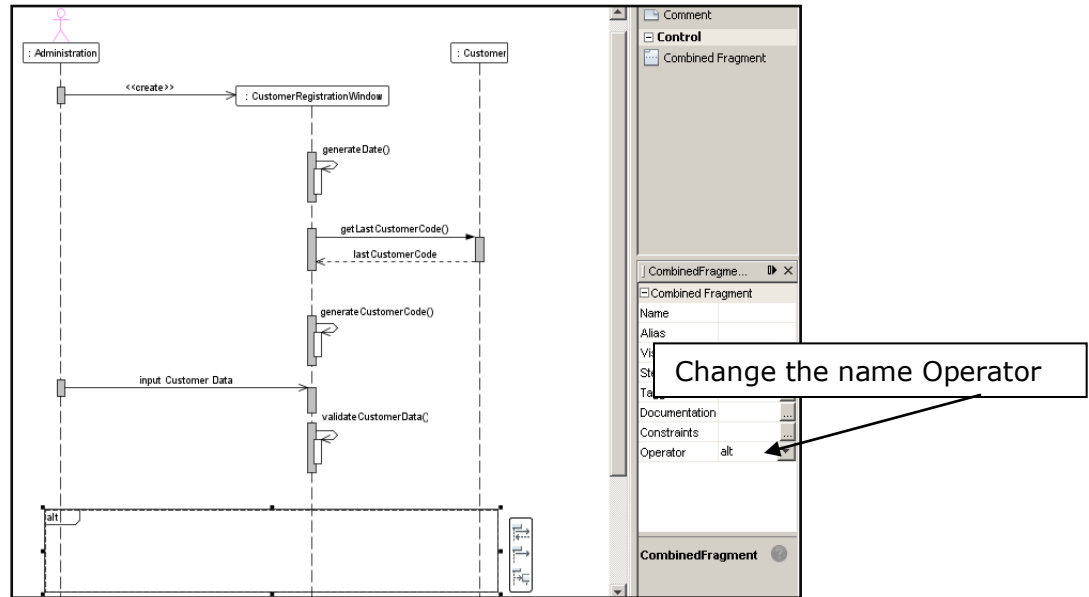


h. Task 08 – Create Combined Fragment

1. Drag the combined fragment from Palette–Control to your worksheet then adjust the width

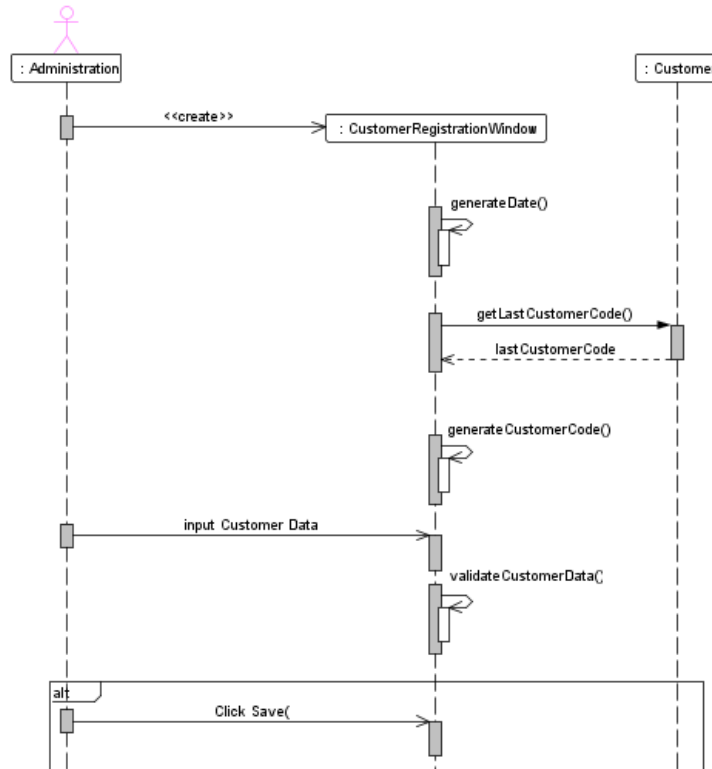


2. Choose the Operator from assert to alt with click on your Combined Fragment and find the Property that name Operator and change the content



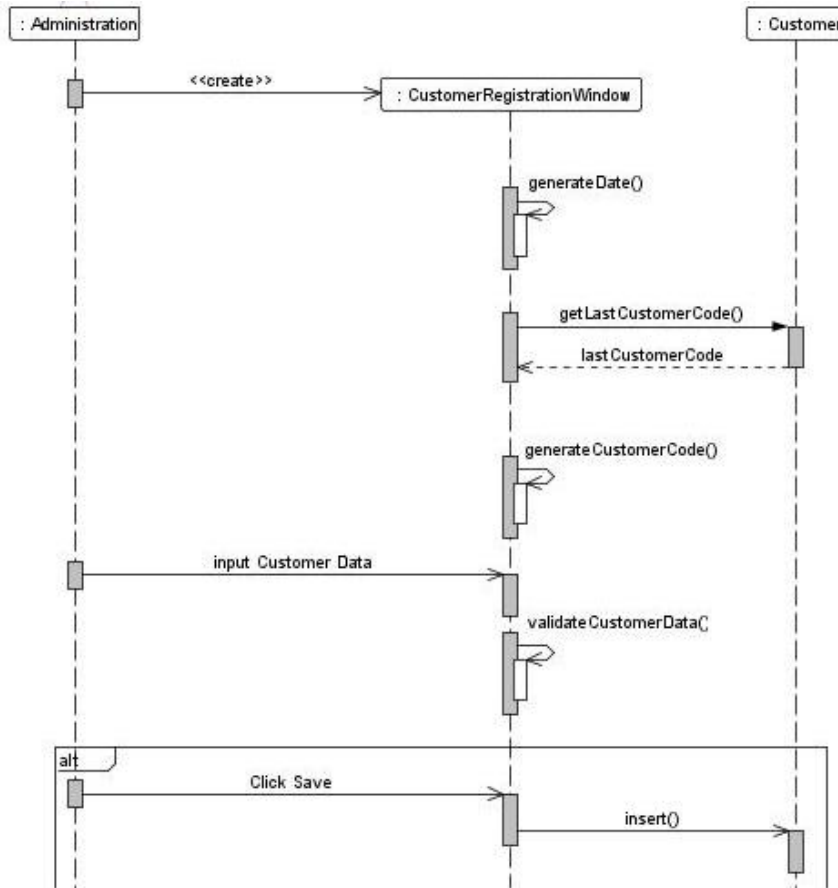
i. Task 09 - Create Asynchronous Message for Click Save

1. Click Administration Actor Life line then choose Asynchronous Message and drag it to CustomerRegistrationWindow and give the name



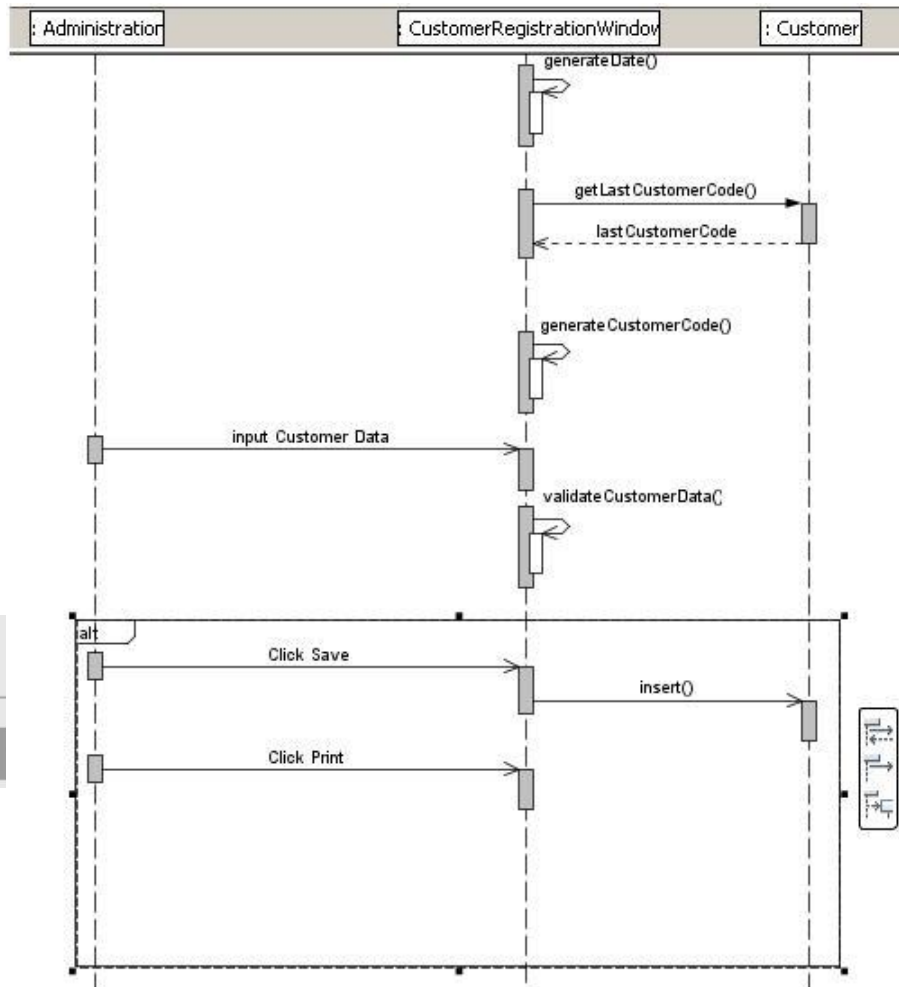
j. Task 09 – Create Asynchronous Message for insert()

1. Once click on your customer RegistrationWindow life line then click create Asynchronous Message then and point it to Customer life line.



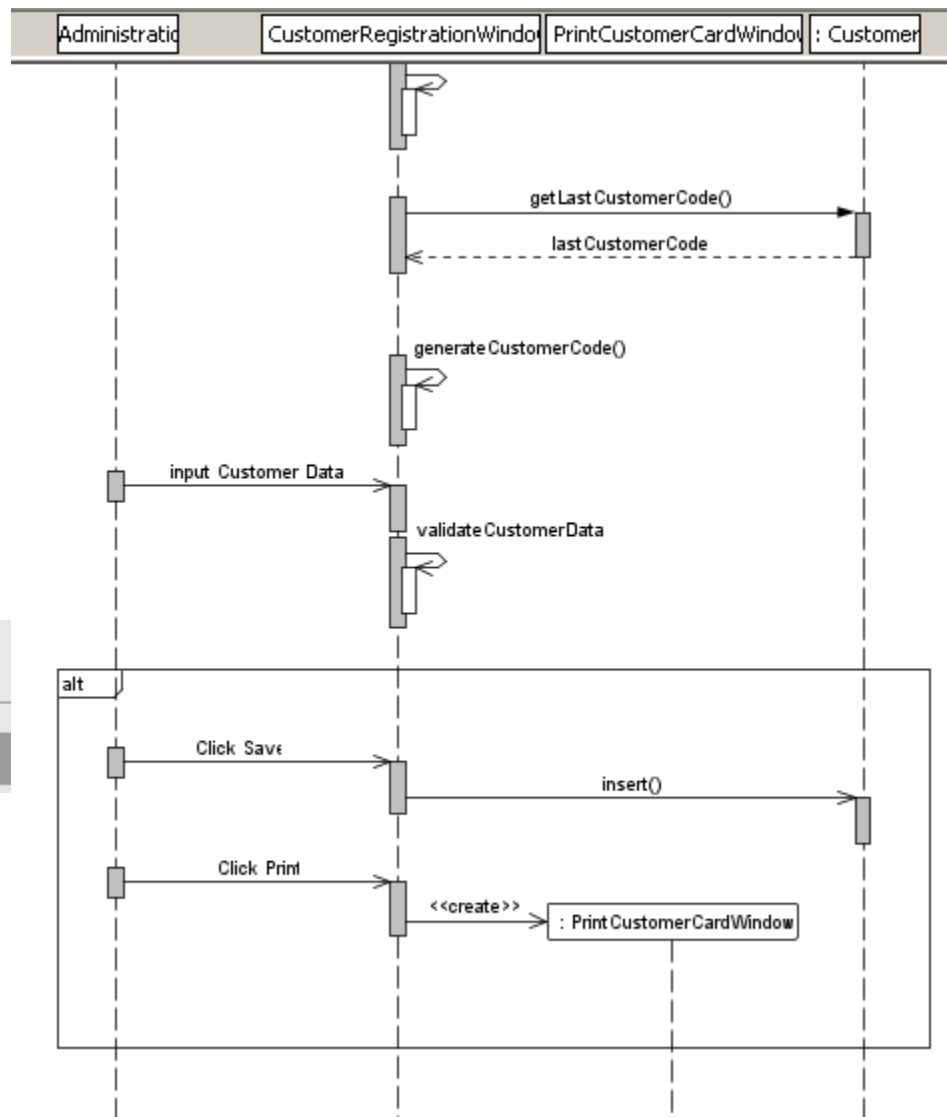
k. Task 09 – Create Asynchronous Message for Click Print

1. Once click on your Administration life line then click create Asynchronous Message then and point it to customer RegistrationWindow.



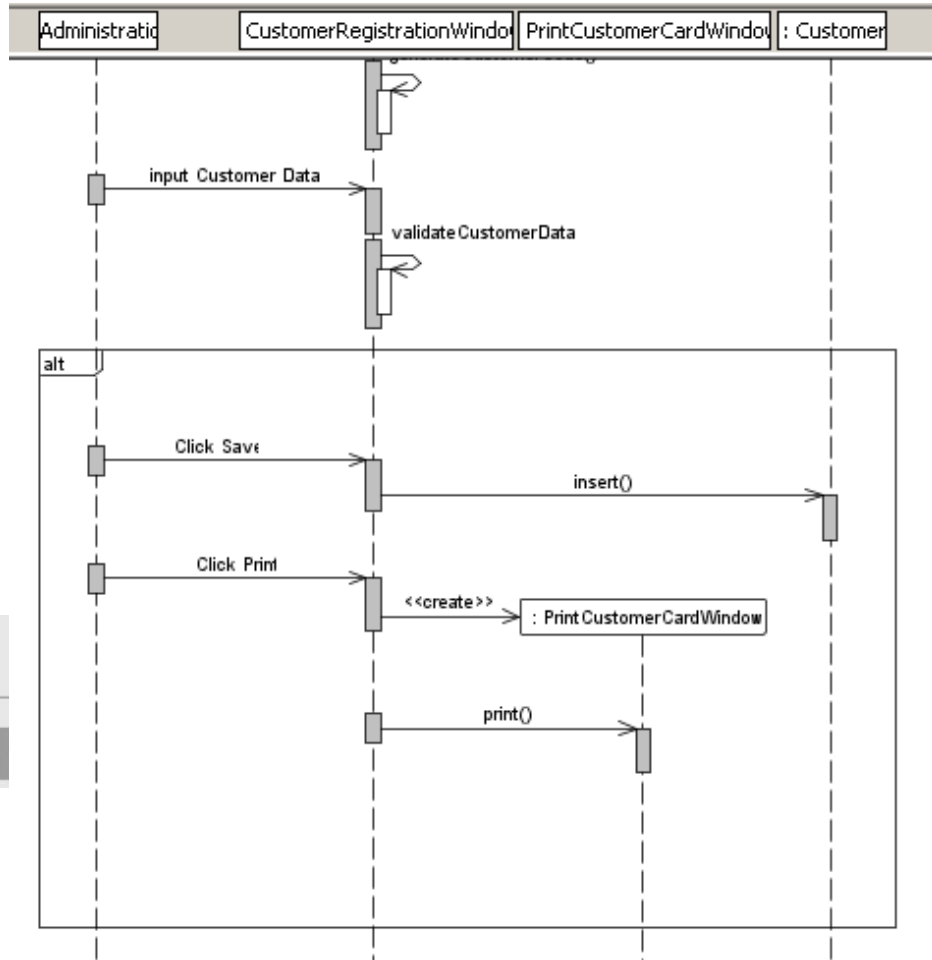
1. Task 10 – Create Message for PrintCustomerCardWindow

1. Once click on your customer RegistrationWindow life line then click Create Message then and point it between customer RegistrationWindow and Customer.



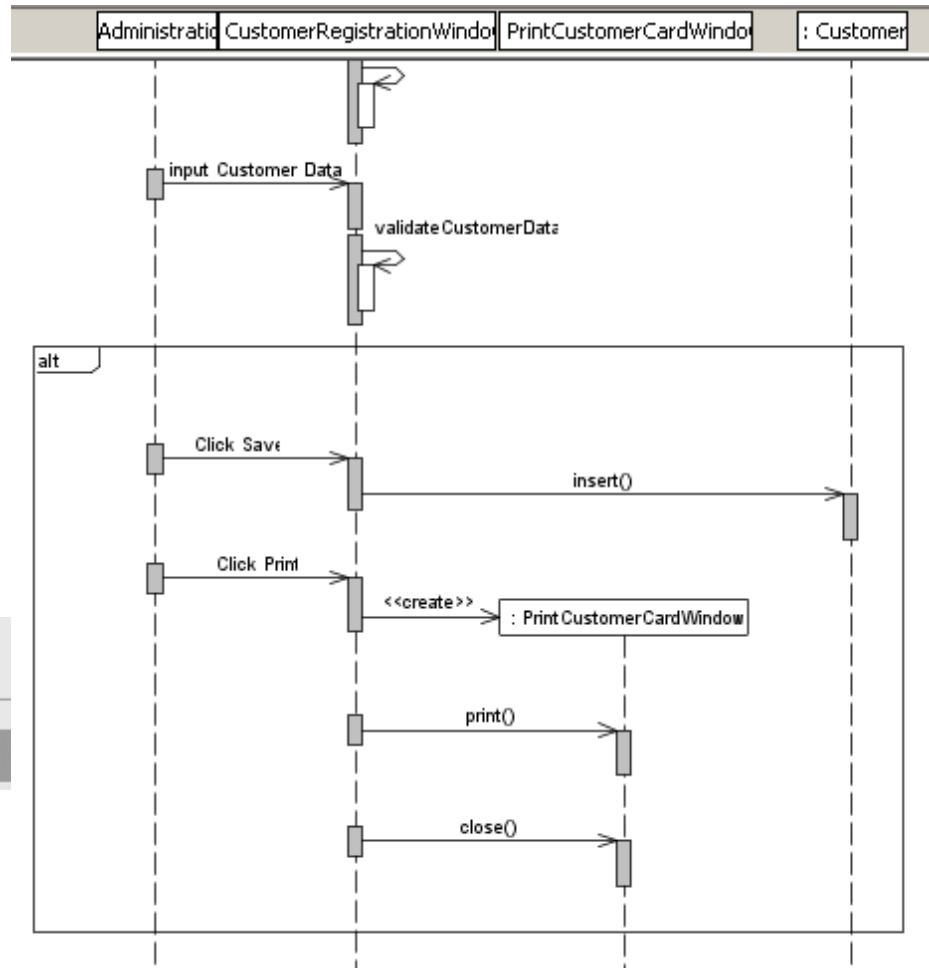
m. Task 11 – Create Asynchronous Message for print()

1. Once click on your customer RegistrationWindow life line then click create Asynchronous Message then and point it to PrintCustomerCardWindow.



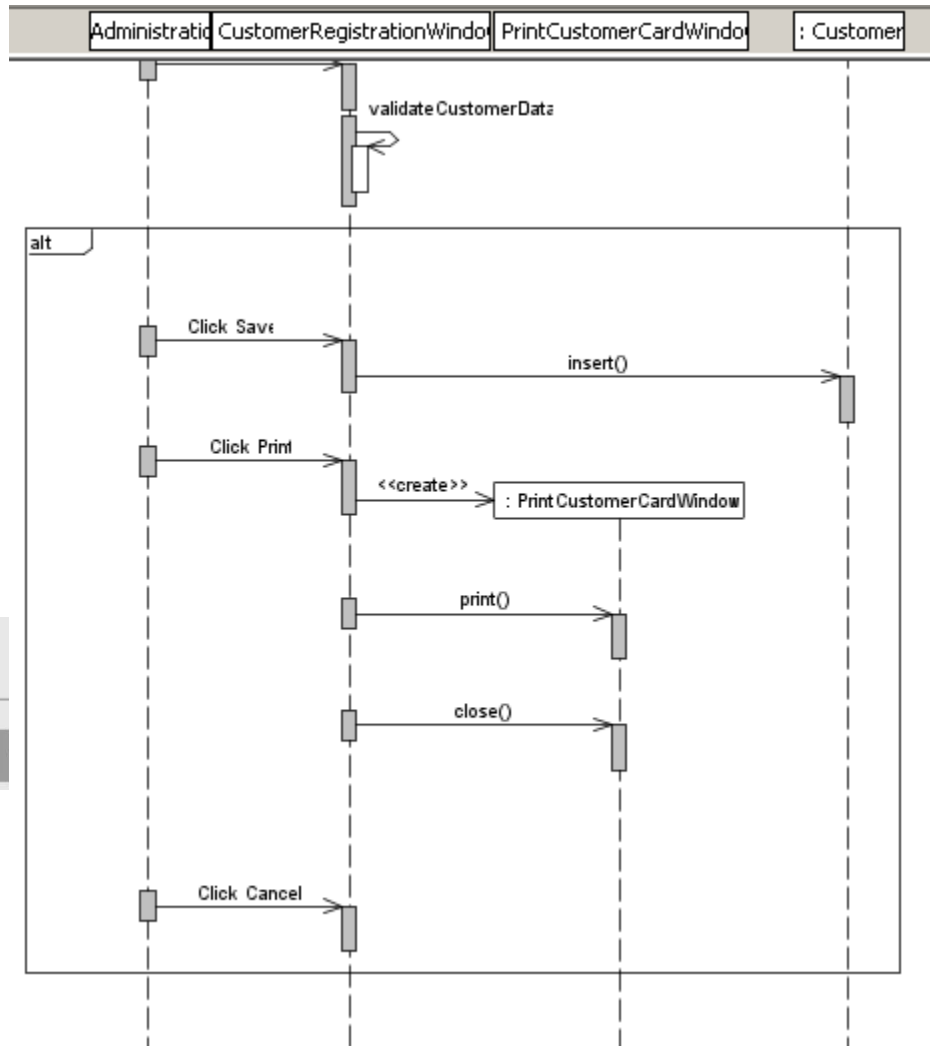
n. Task 12 – Create Asynchronous Message for close()

1. Once click on your CustomerRegistrationWindow life line then click create Asynchronous Message then and point it to PrintCustomerCardWindow.

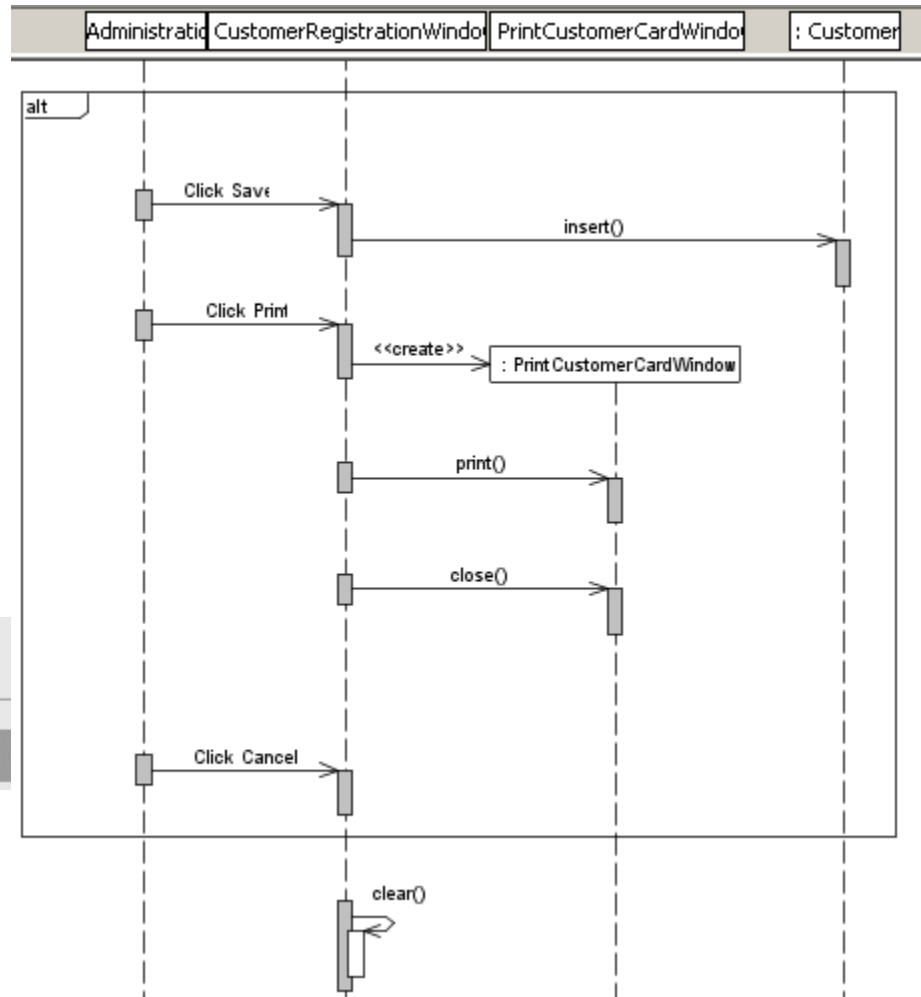


p. Task 13 – Create Asynchronous Message for Click Cancel

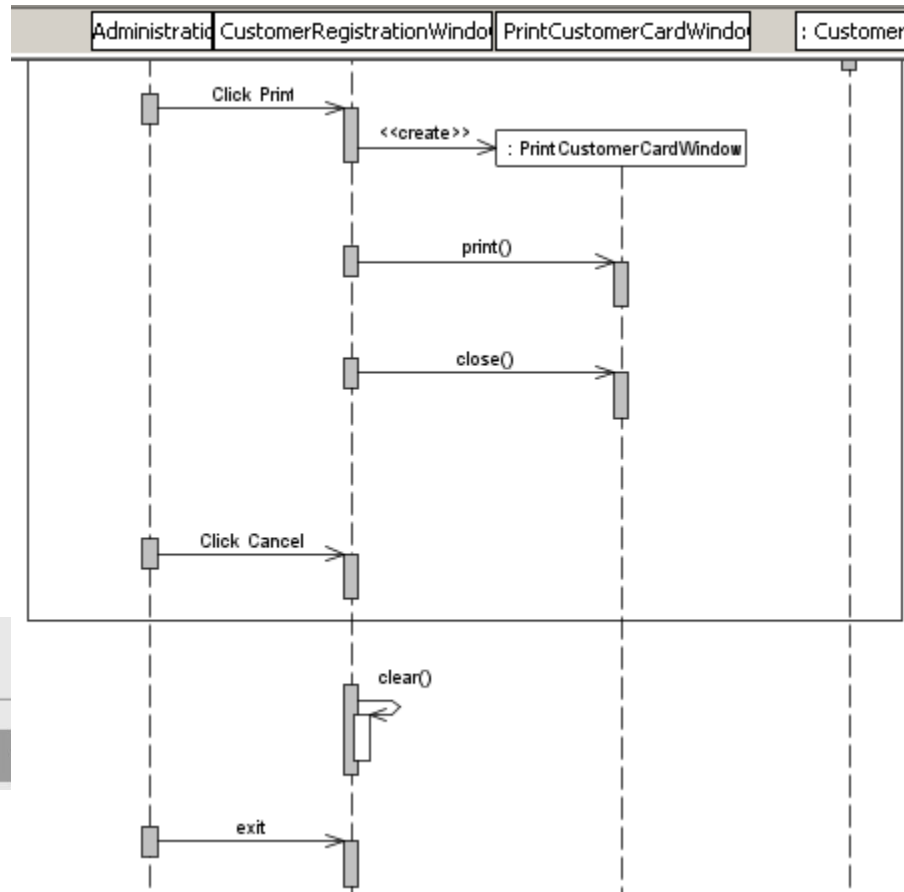
1. Once click on your Administration life line then click create Asynchronous Message then and point it to CustomerRegistrationWindow.



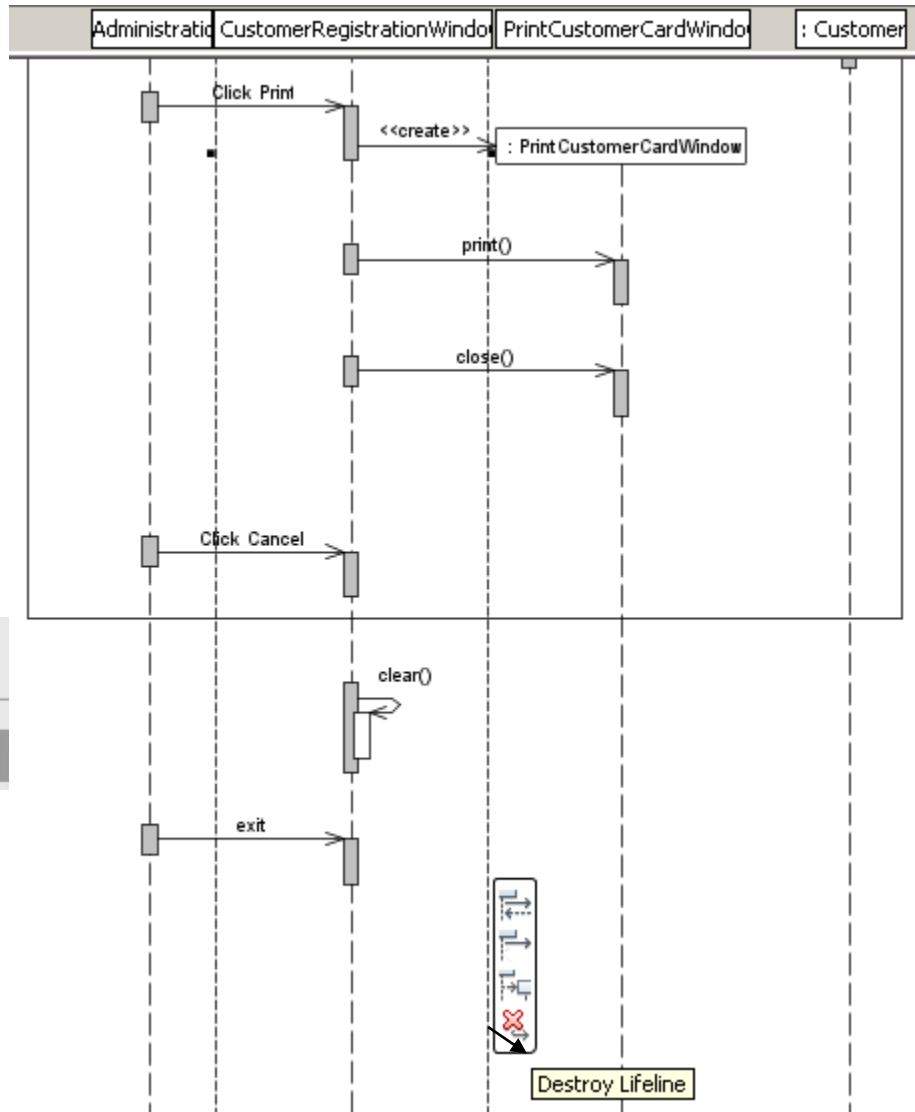
- q. Task 14 – Create Recursive Asynchronous Message for clear()
1. Once click on your CustomerRegistrationWindow life line then click create Asynchronous Message then and point it to itself.



- s. Task 15 - Create Asynchronous Message for exit
 - 1. Once click on your Administration life line then click create Asynchronous Message then and point it to CustomerRegistrationWindow.

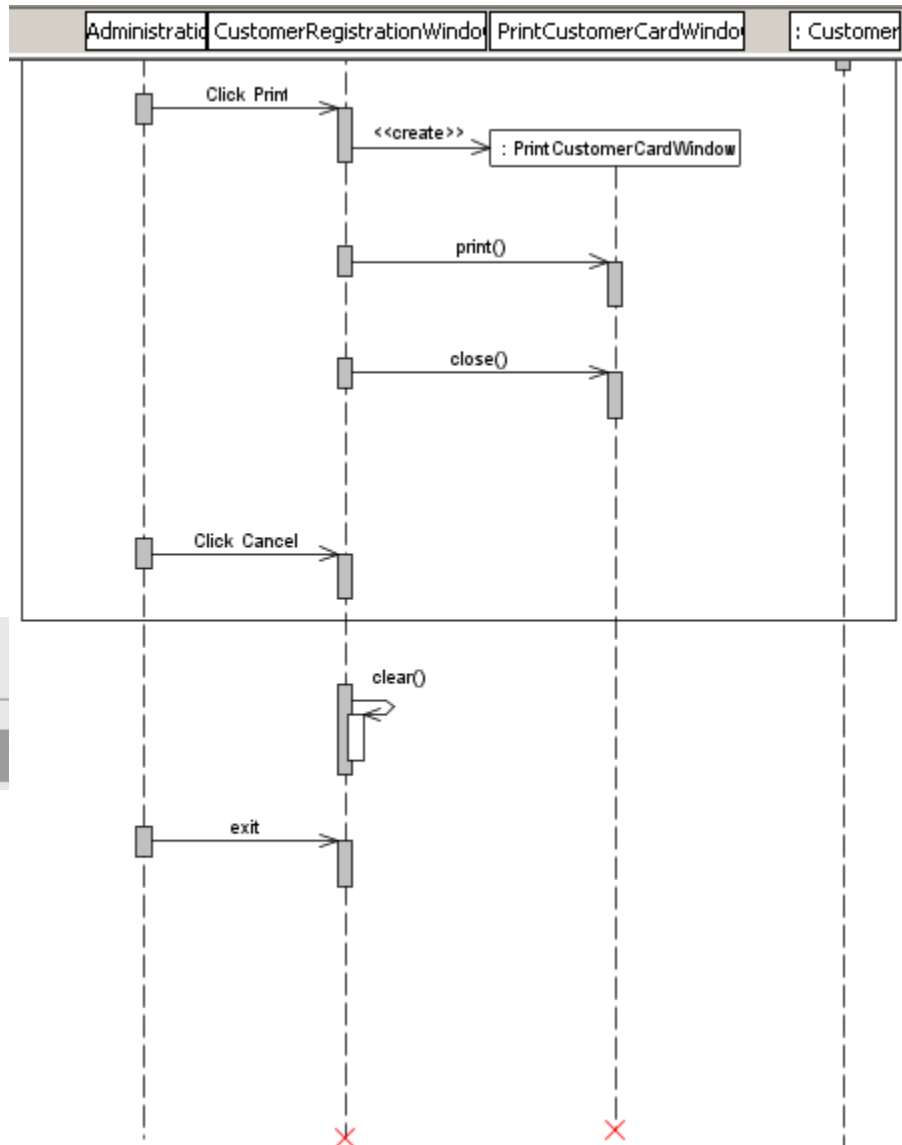


- t. Task 16 – Create Destroy Life line for CustomerRegistrationWindow
 - 1. Once click on your CustomerRegistrationWindow life line then click Destroy Life line.



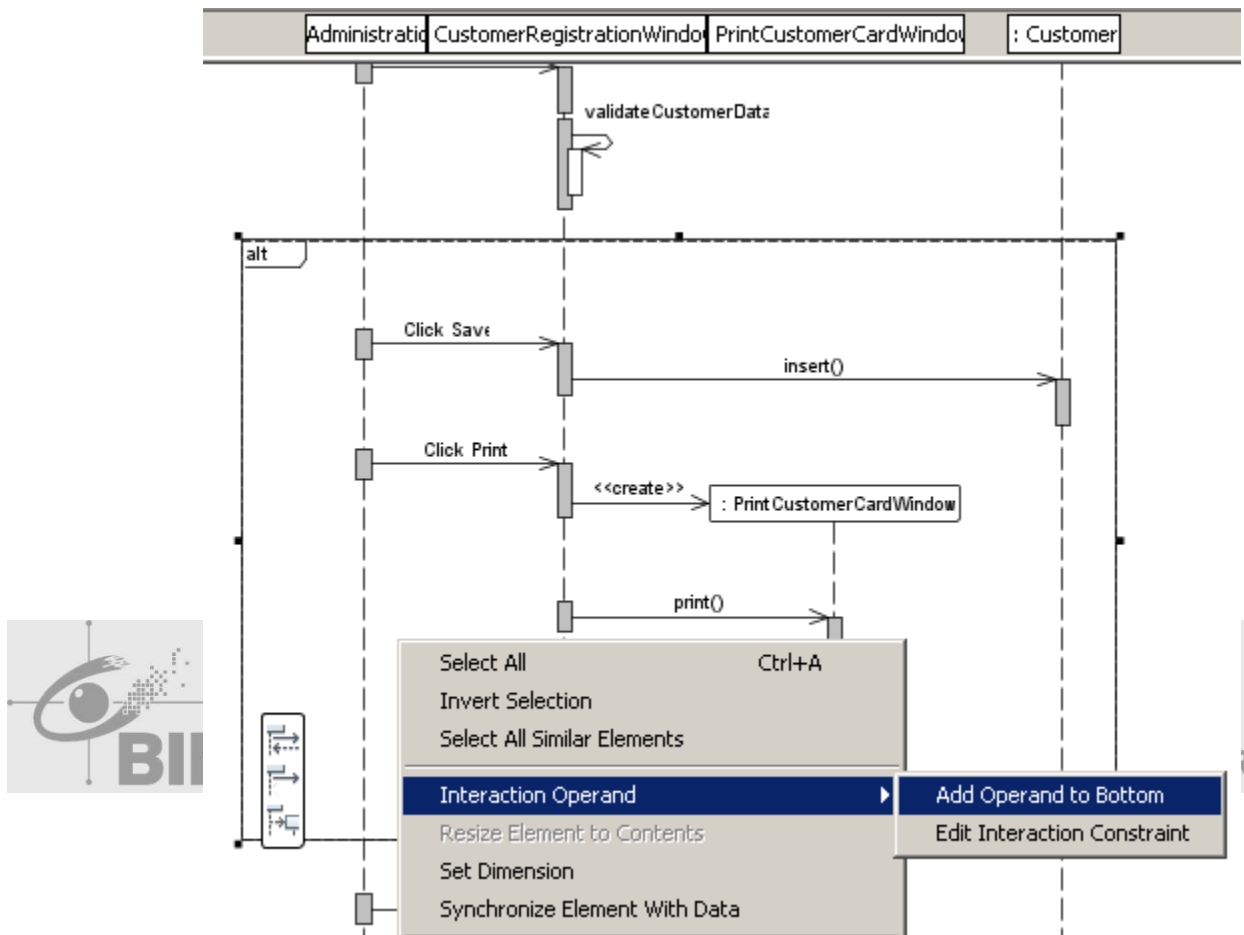
u. Task 17 – Create Destroy Life line for PrintCustomerCardWindow

1. Once click on your PrintCustomerCardWindow life line then click Destroy Life line.



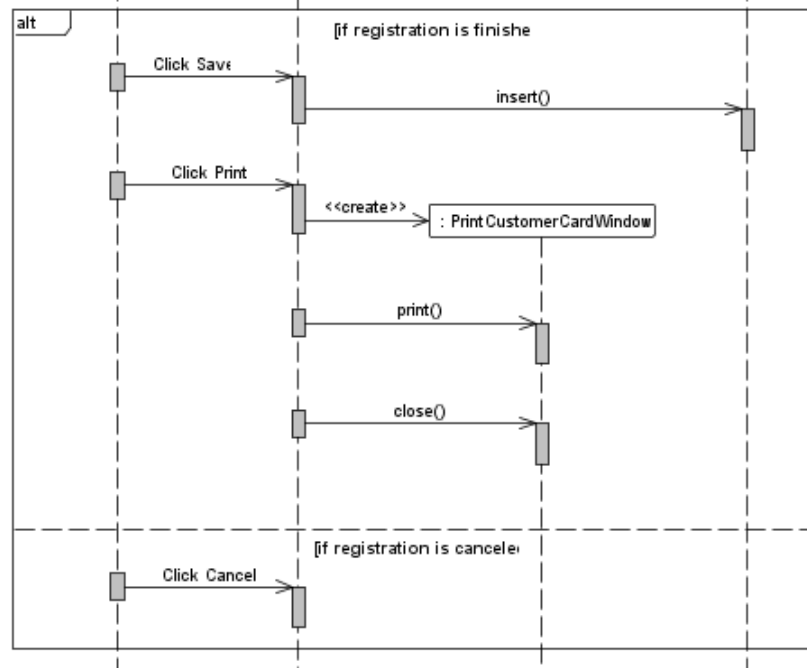
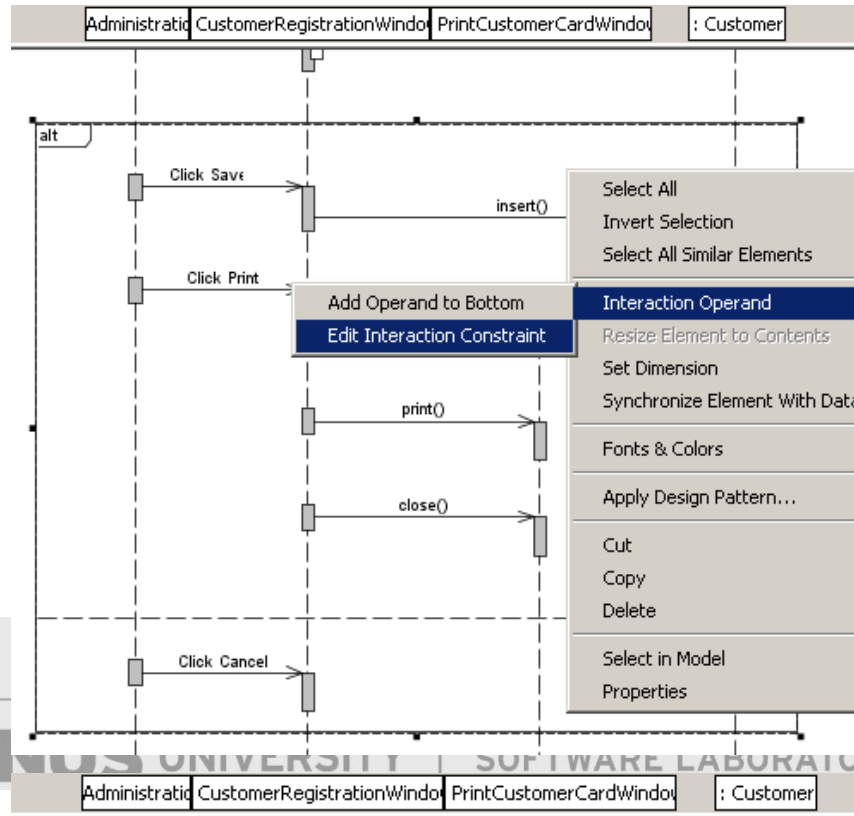
v. Task 17 – Create Destroy Interaction Operand

1. Once click on your Combined Fragment life line then right click, choose Interaction Operand – Add Operand to Bottom.



w. Task 17 – Create Interaction Constraint

1. Once click on your Combined Fragment life line then right click, choose Interaction Operand – Edit Interaction Constraint. Then rename it.

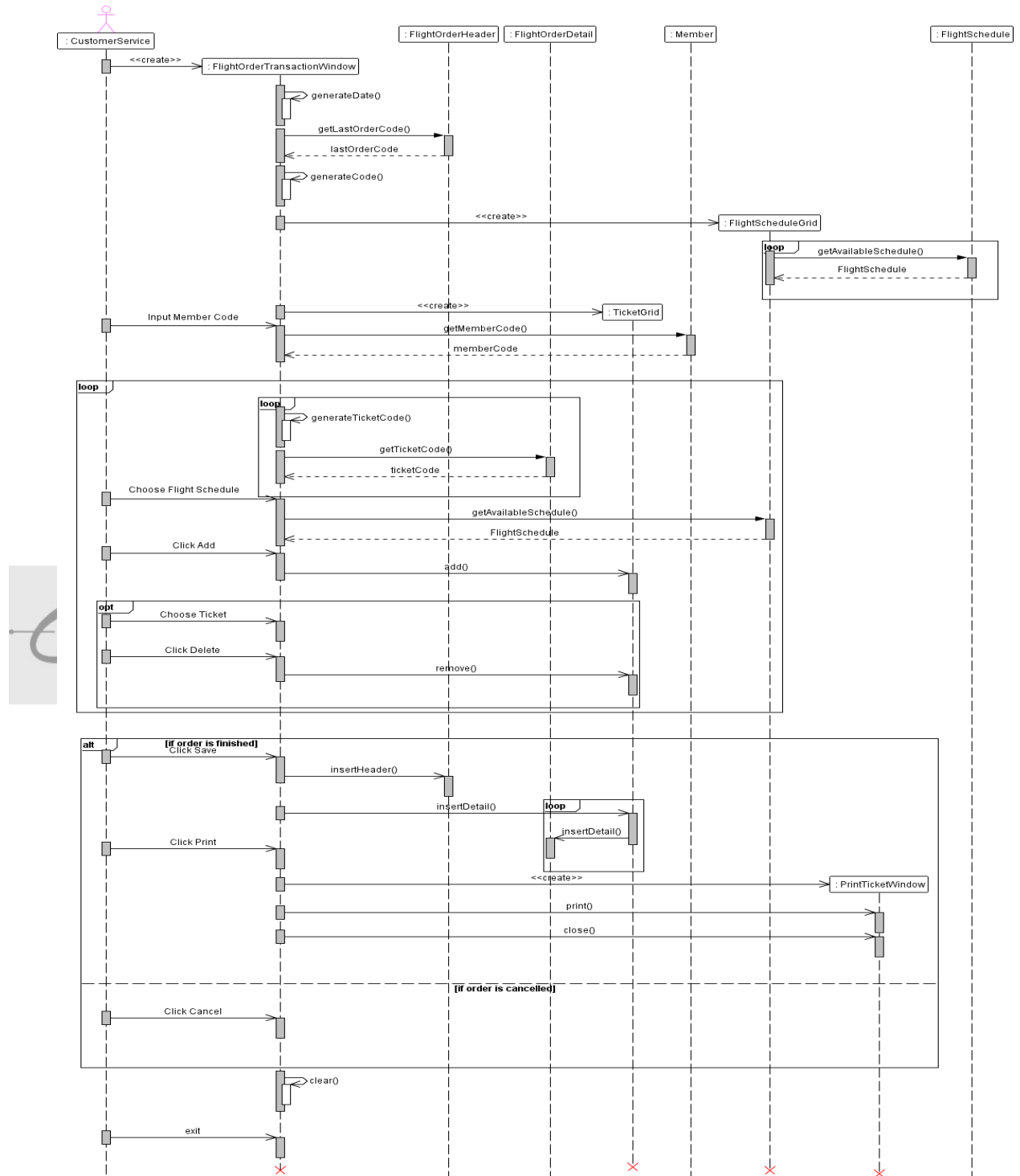


- x. Task 18 – Create Recursive Asynchronous Message at CustomerRegistrationWindow
- y. Task 19 – Create Asynchronous Message to CustomerRegistrationWindow from Administration
- z. Task 20 – Create Destroy for both CustomerRegistration and PrintCustomer Card

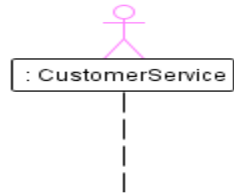
Explanation of above Sequence Diagram

1. Administration Division opens the MemberRegistrationWindow to input the Member Data.
2. The window generates current date.
3. The window retrieves the last Member Code in order to generate Member Code, and then it generates the code.
4. Administration Division input the member data, and then the window validates the member data(like phone number must be a number, and etc.).
5. If the registration is finished, Administration Division can click Save button to insert member data to the database, and then click Print button to print the Member Card(the window will show the PrintMemberCardWindow, then print the card, after that, it will be closed automatically).
6. Or if the registration is cancelled, Administration Division can click Cancel button and it won't insert anything or print anything, because it has been cancelled.
7. After it, all of data or field in the window will be cleared and the window will exit.
8. After all of the sequence processes end, the MemberRegistrationWindow and the PrintMemberCardWindow will be destroyed.

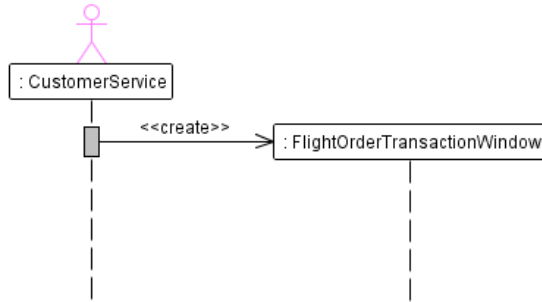
Exercise 04 – Make Sequence Diagram 2



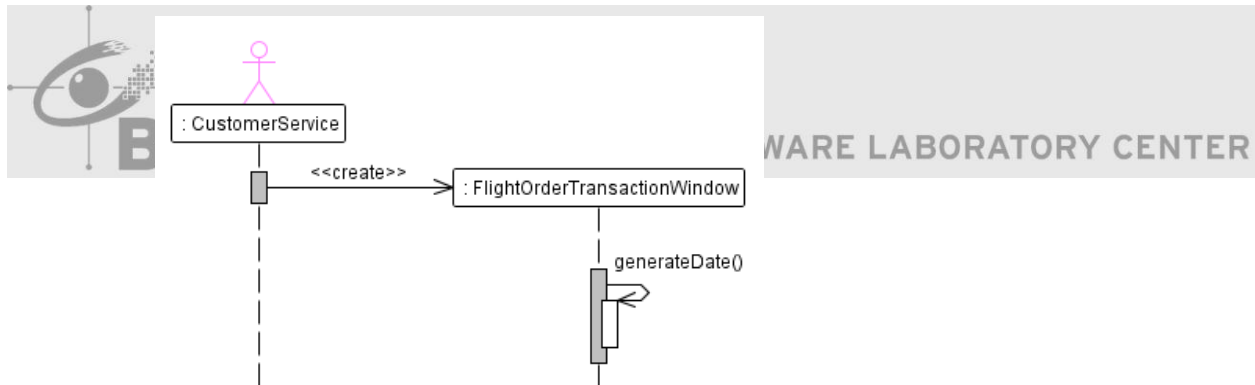
a. Task 01 – Create CustomerService Actor Life line



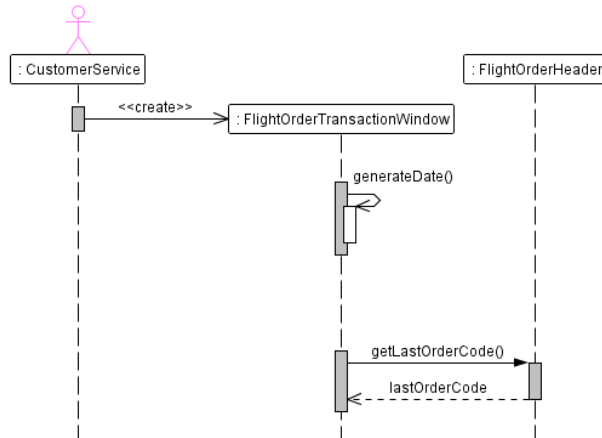
b. Task 02 – Create Message for FlightOrderTransactionWindow



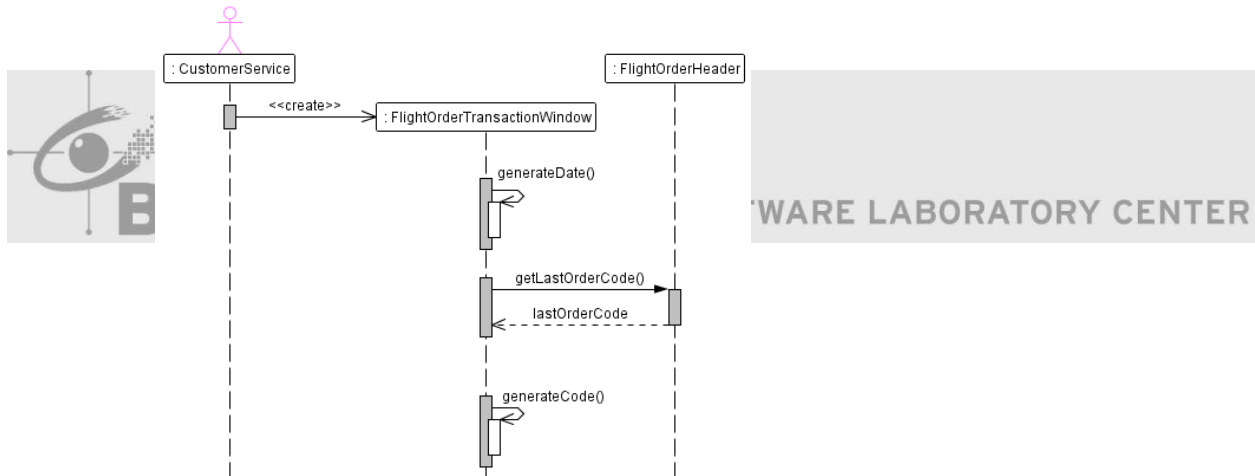
c. Task 03 – Create Recursive Asynchronous Message for generateDate()



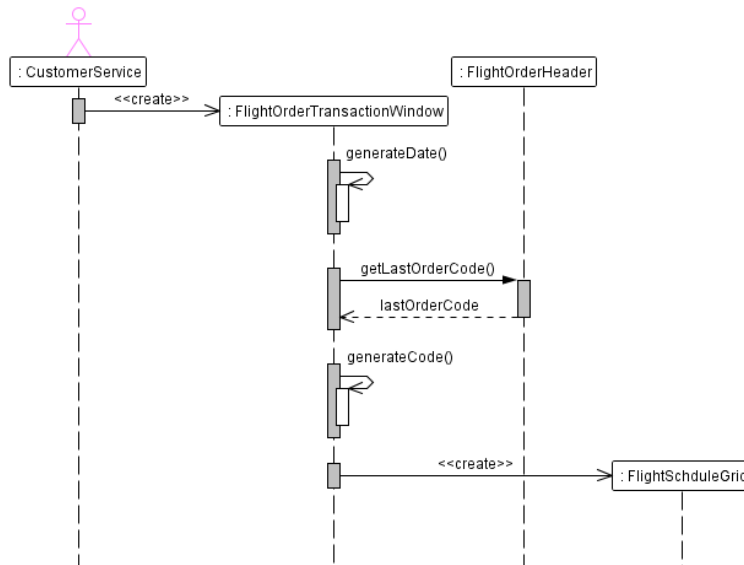
- d. Task 04 – Create Synchronous Message for FlightOrderHeader then give the name getLastOrderCode() and lastOrderCode



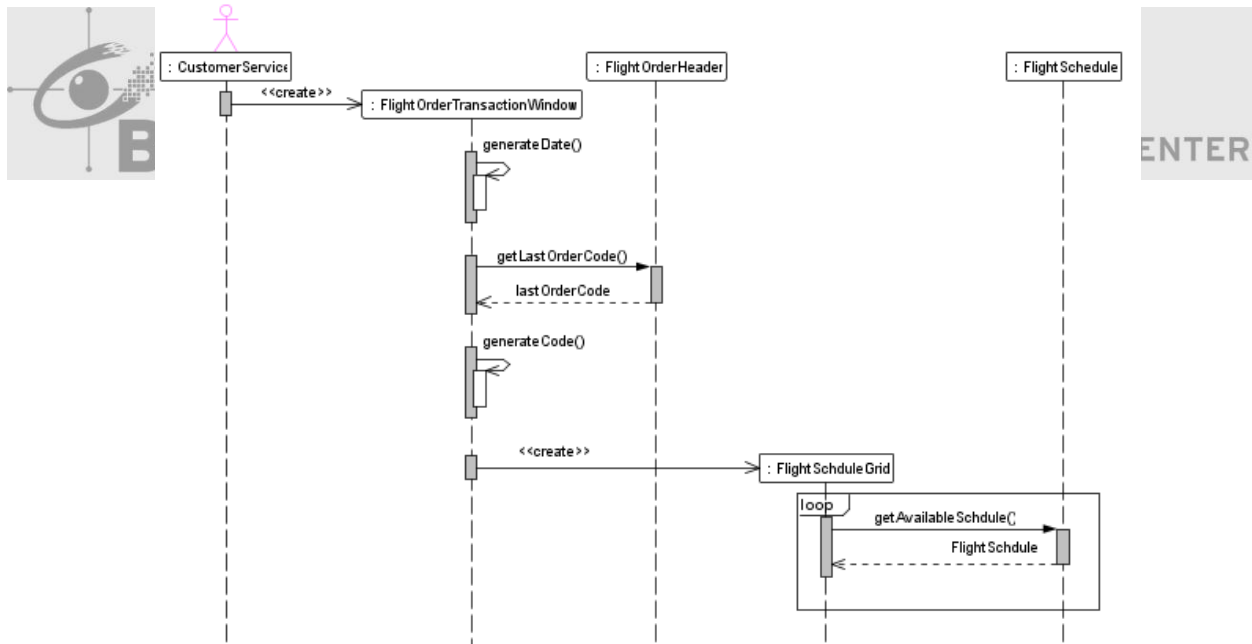
- e. Task 05 – Create recursive Asynchronous Message generateCode() for FlightOrderTransactionWindow



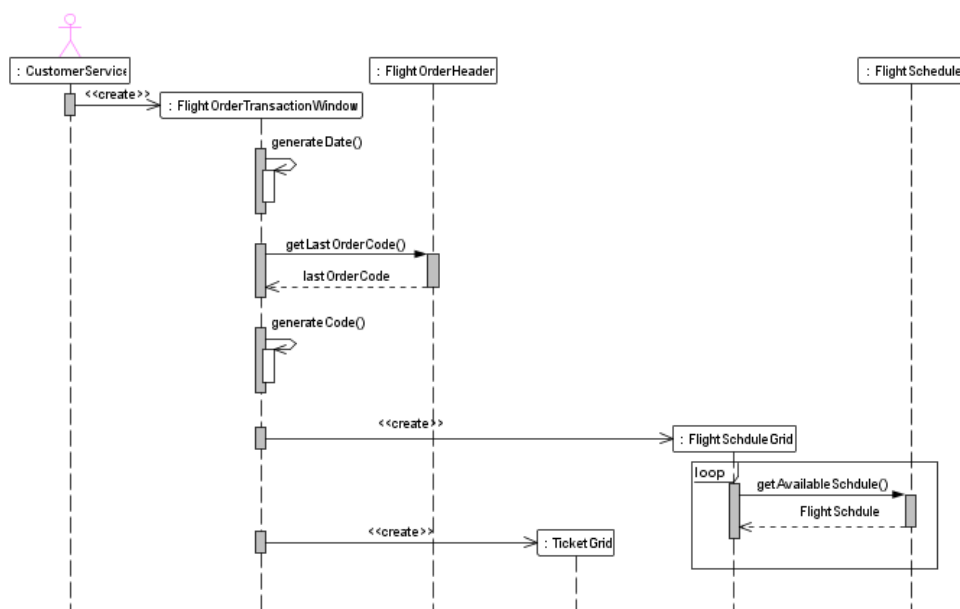
f. Task 06 – Create Message from FlightOrderTransactionWindow to FlightSchdule Grid



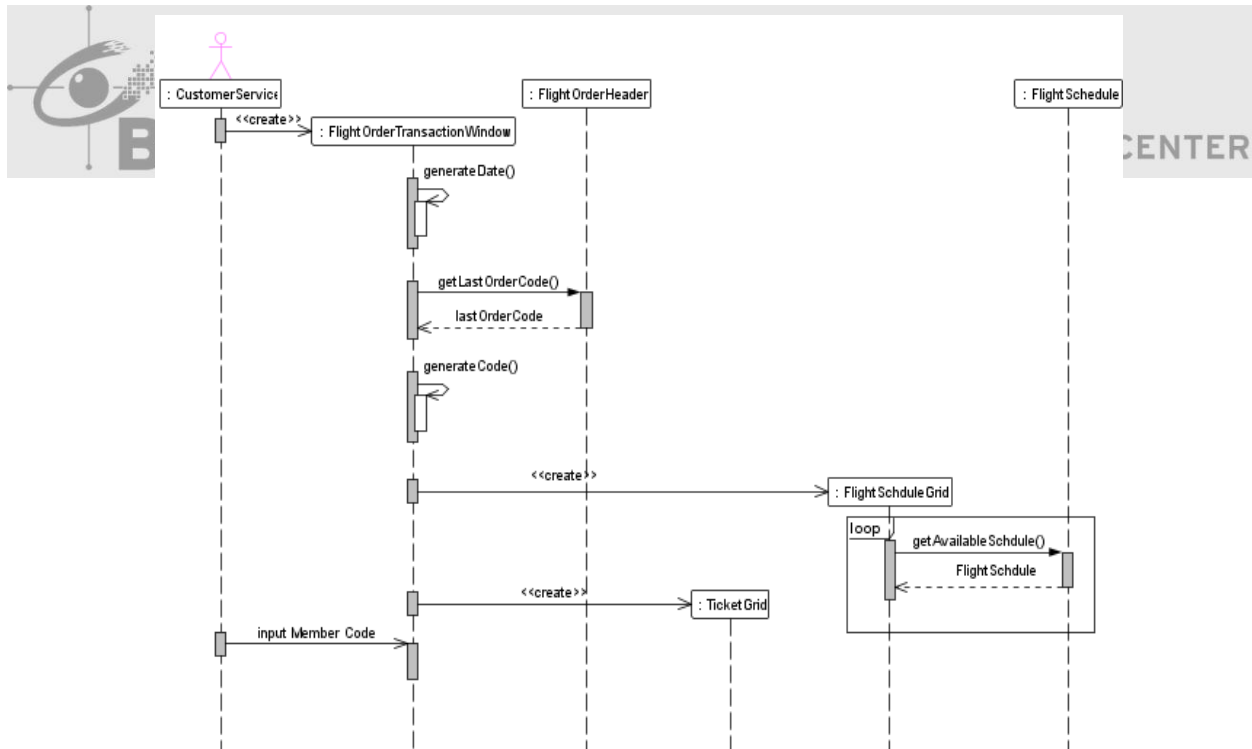
g. Task 07 – Create Synchronous for FlightSchedule



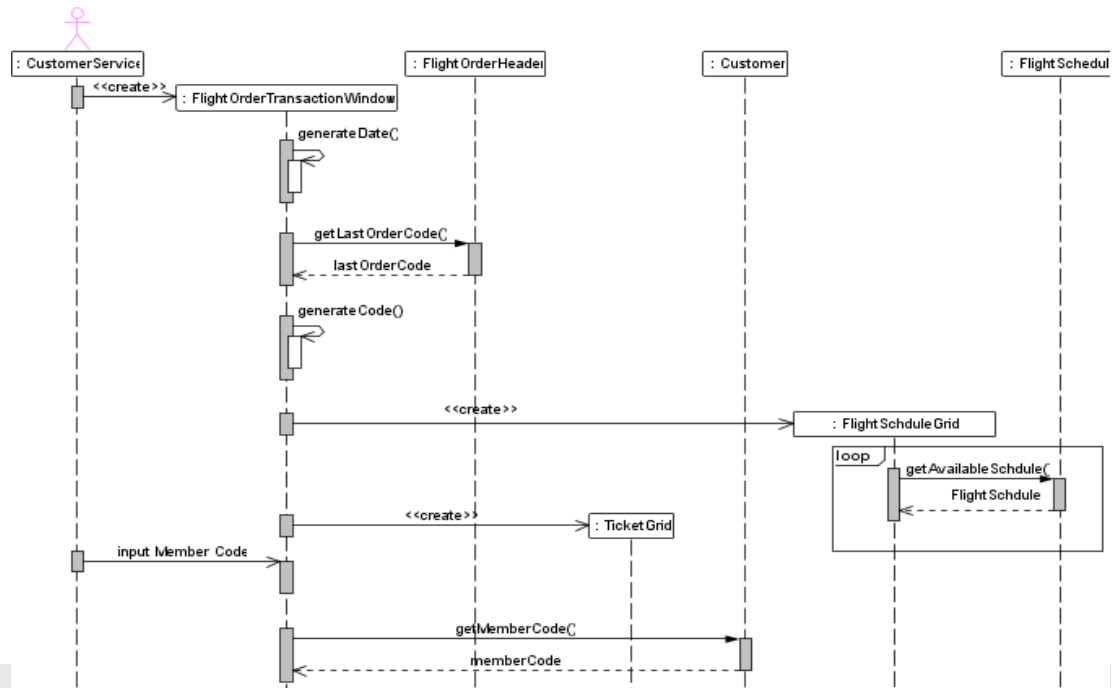
h. Task 08 – Create Message for TicketGrid from FlightOrderTransactionWindow



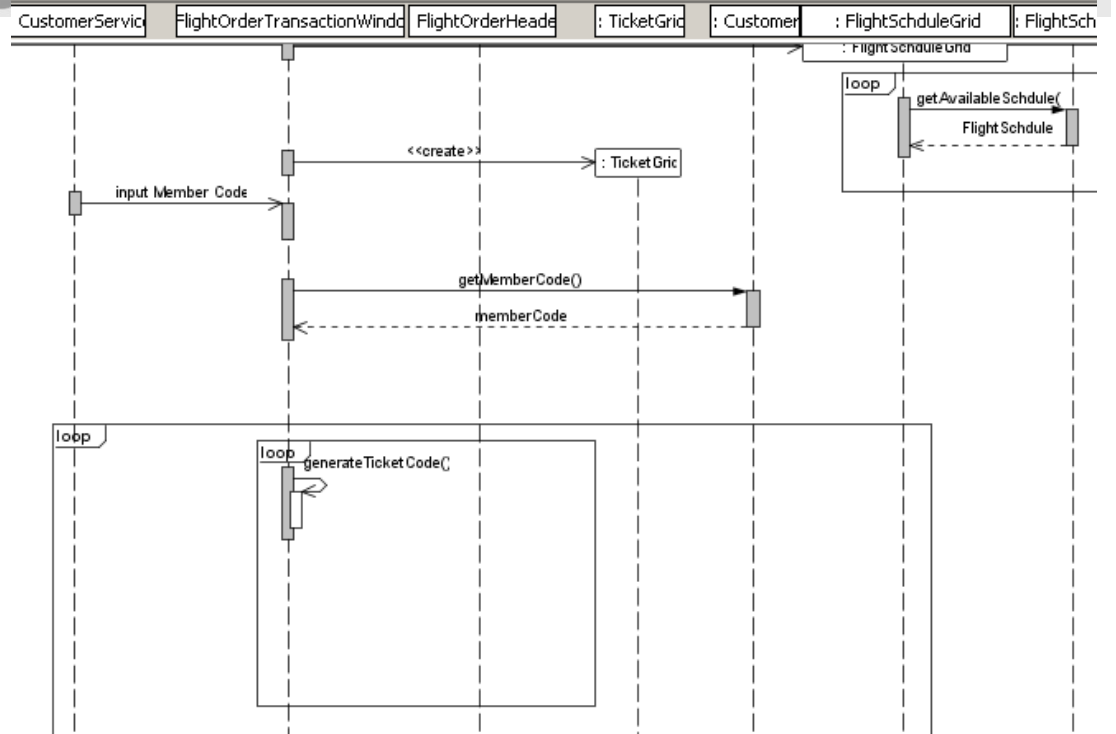
i. Task 09 – Create input MemberCode to FlightOrderTransactionWindow from Customer Service with Asynchronous Message



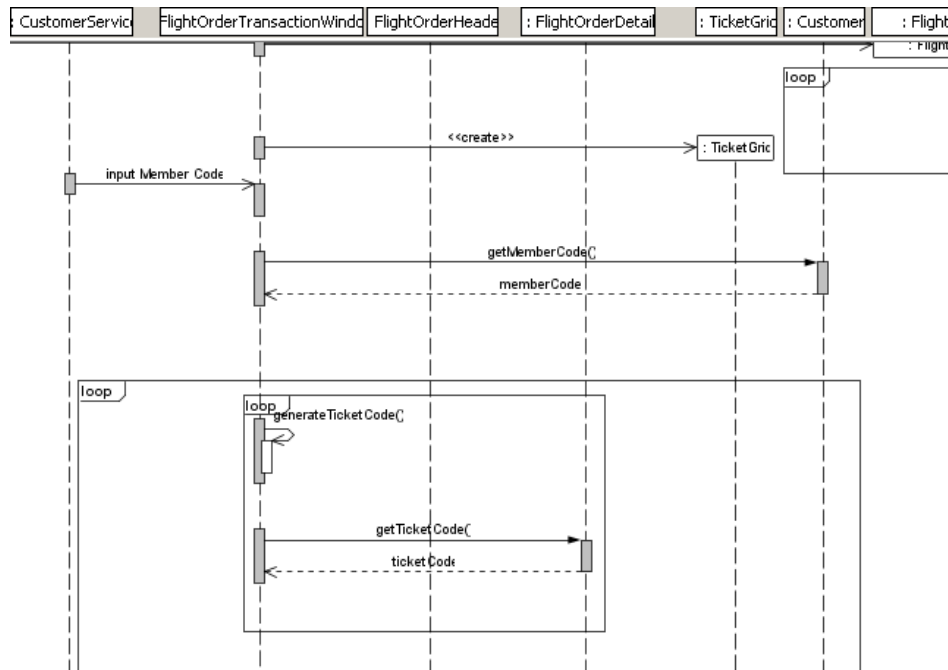
j. Task 10 – Create Synchronous Message from FlightOrderTransactionWindow to Customer



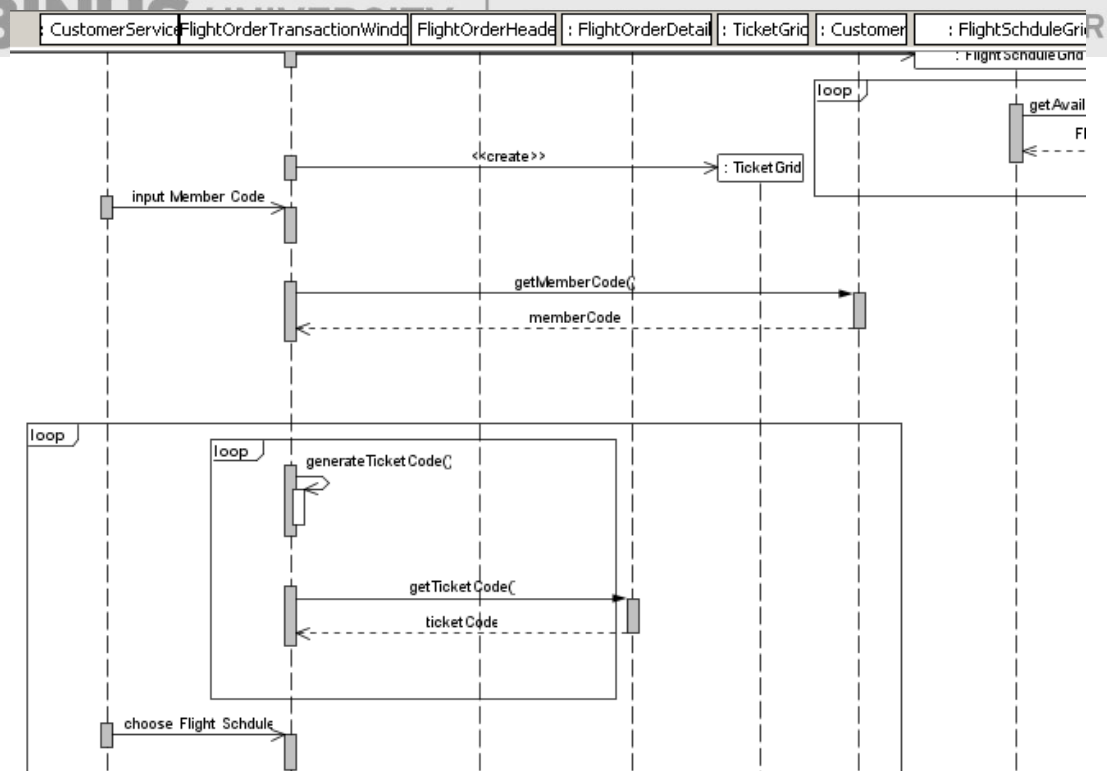
k. Task 11 – Create Combined Fragment and recursive Asynchronous Message GenerateTicketCode() for FlightOrderTransactionWindow



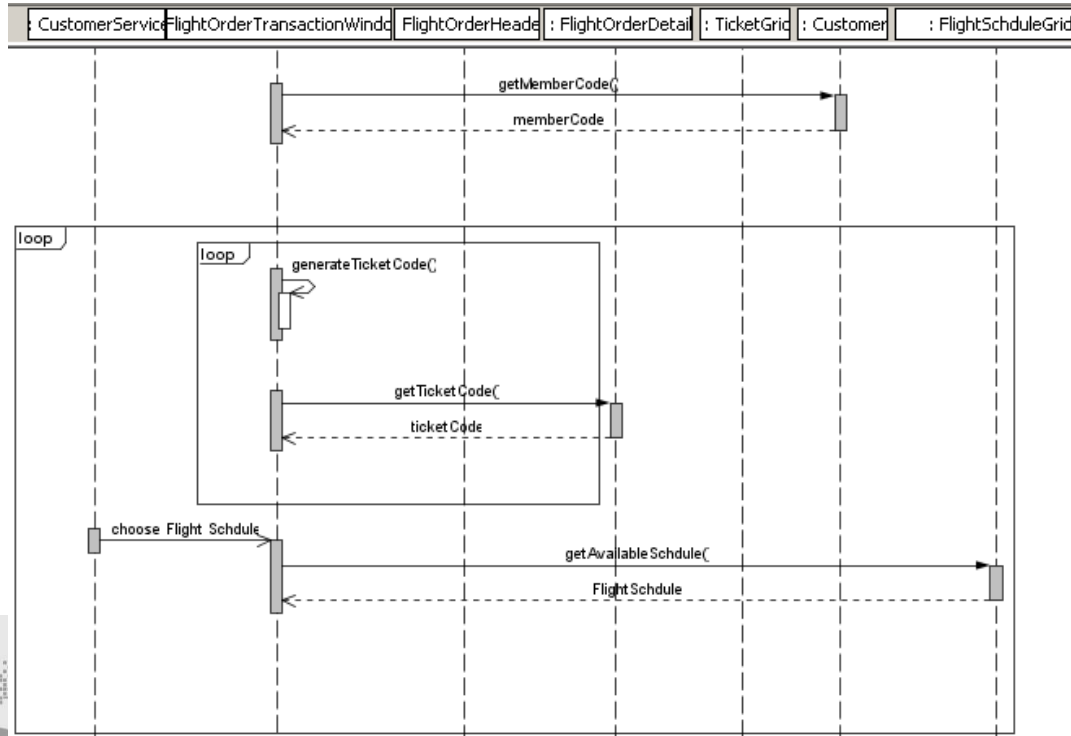
1. Task 12 – Create Synchronous Message for FlightOrderTransactionWindow to FlightOrderDetail



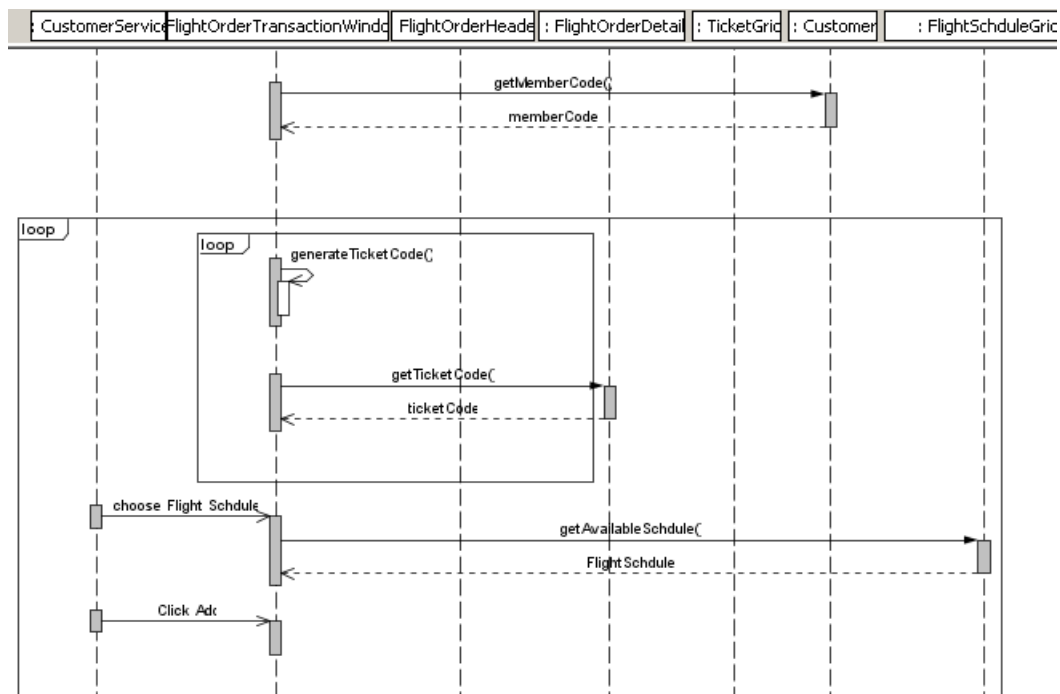
m. Task 13 – Create Asynchronous Message to FlightOrderTransactionWindow from CustomerService



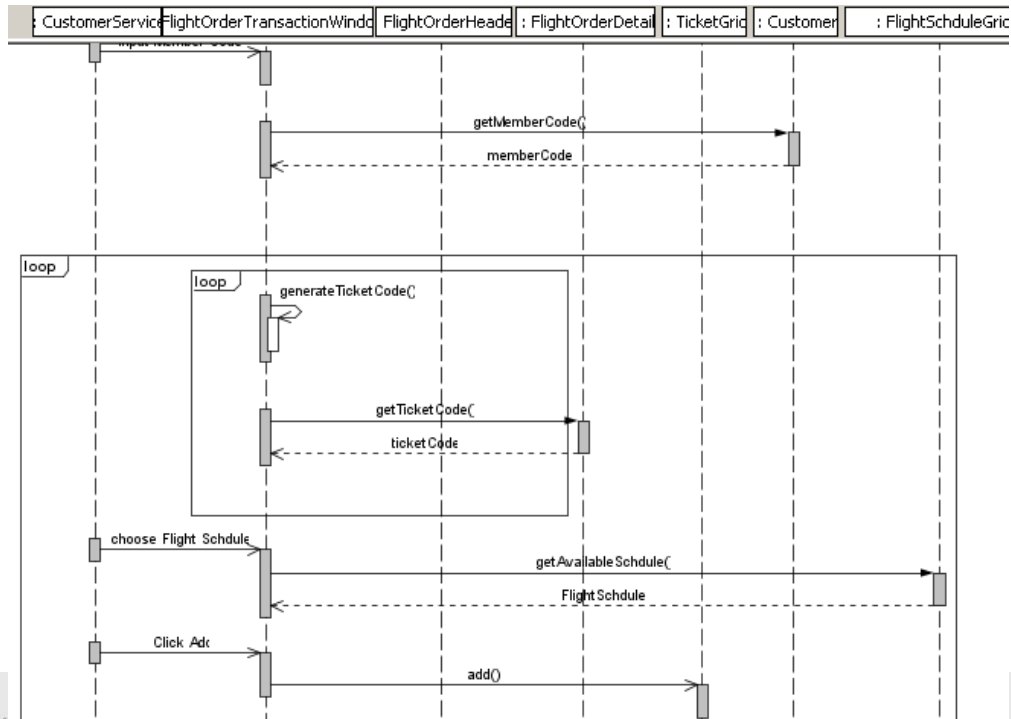
n. Task 14 – Create Synchronous Message from FlightOrderTransactionWindow to FlightSchduleGrid



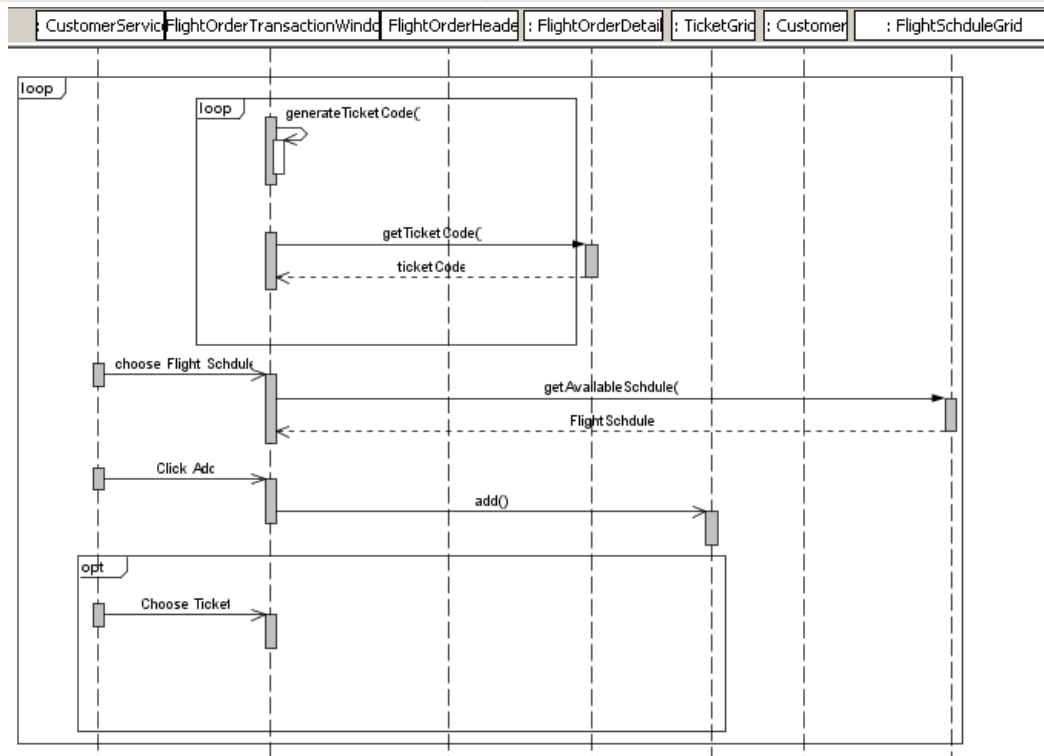
o. Task 15 – Create Asynchronous Message from CustomerService to FlightOrderTransactionWindow



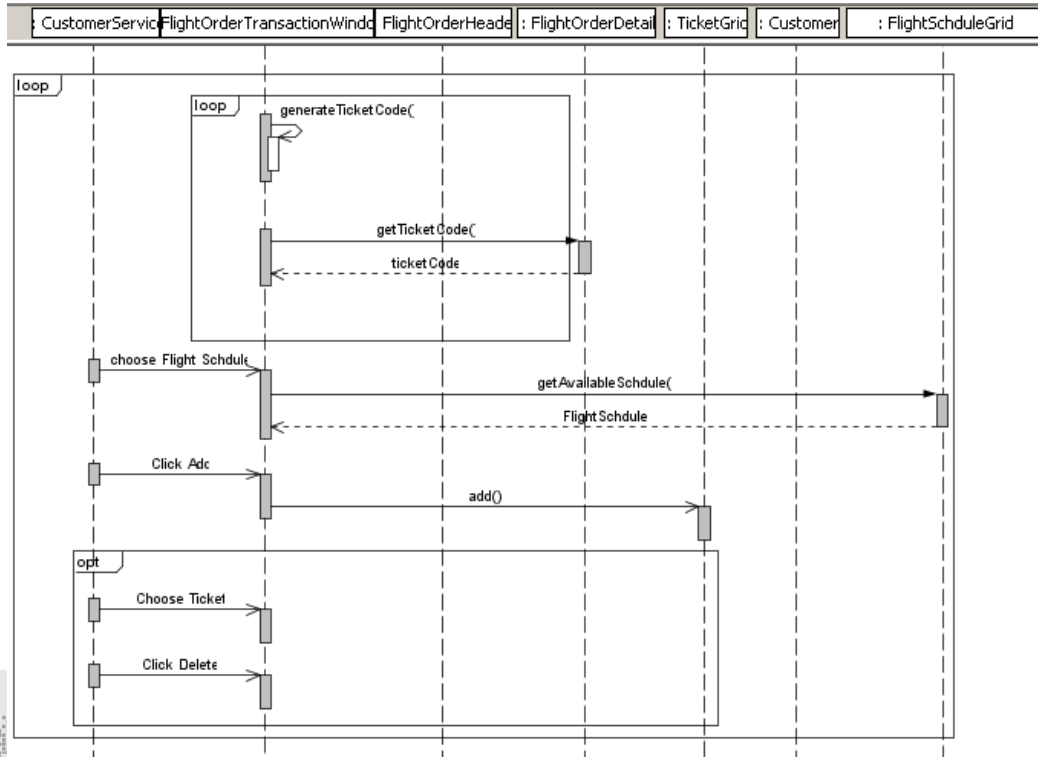
p. Task 16 – Create Asynchronous Message from FlightOrderTransactionWindow to TicketGrid



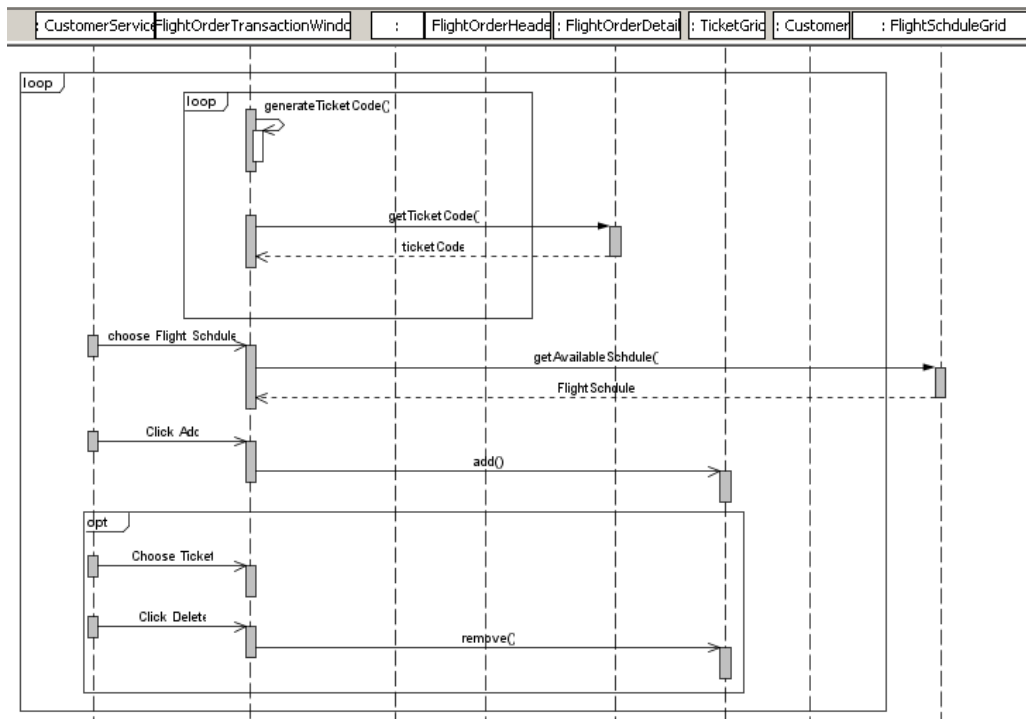
q. Task 17 – Create Asynchronous Message to FlightOrderTransactionWindow from CustomerService for ChooseTicket



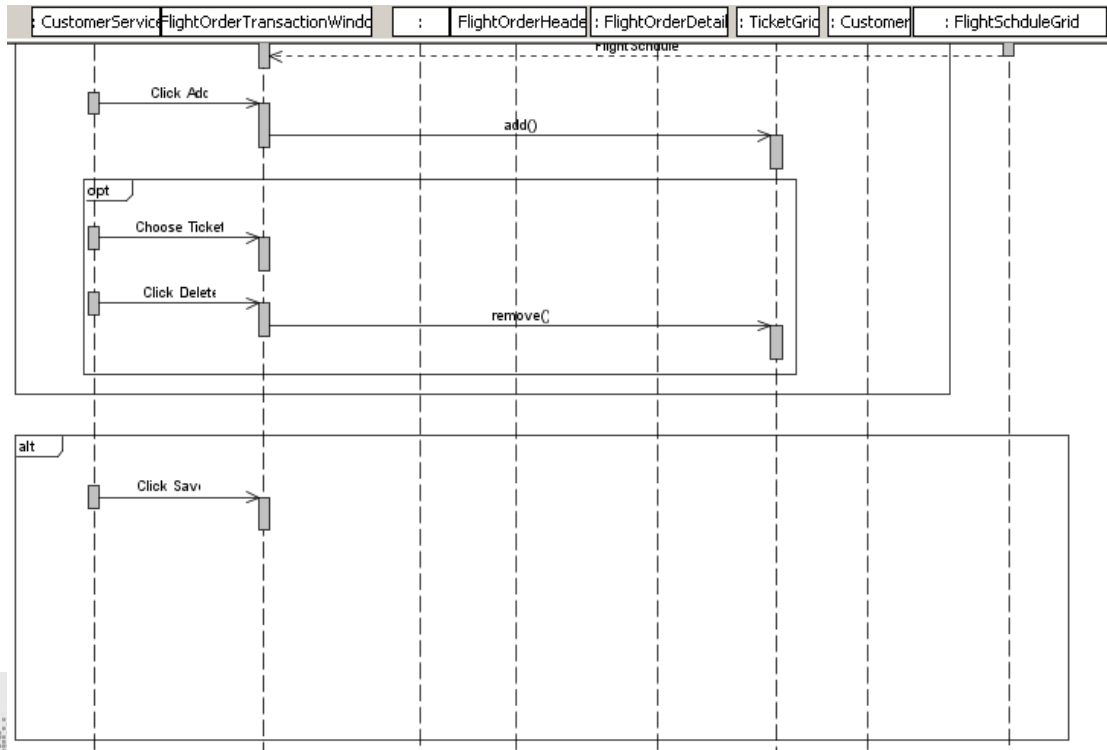
- r. Task 18 – Create Asynchronous Message to FlightOrderTransactionWindow from CustomerService for ClickDelete



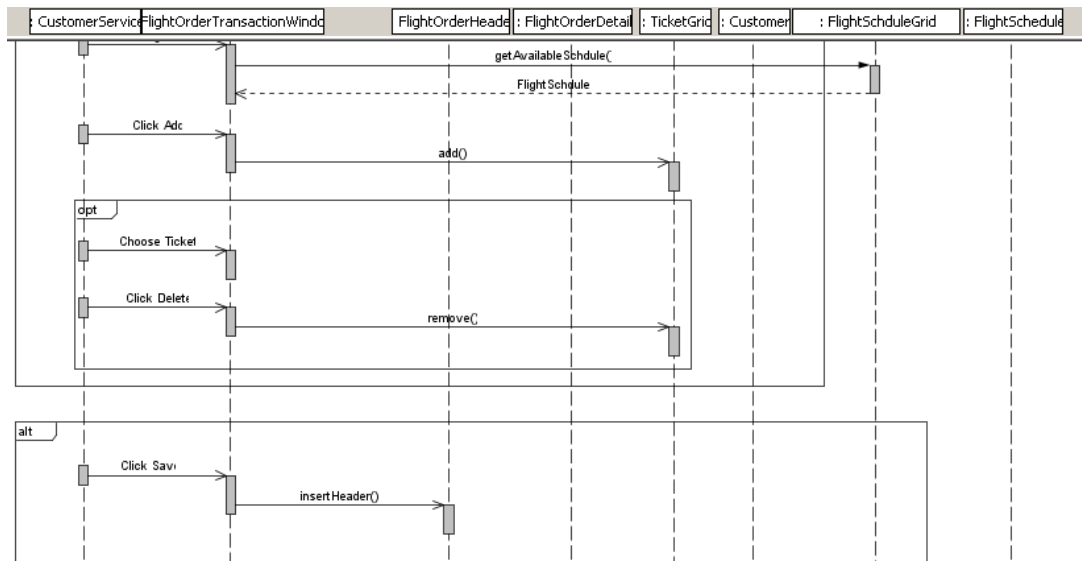
- s. Task 19 – Create Asynchronous Message from FlightOrderTransactionWindow to TicketGridforremove



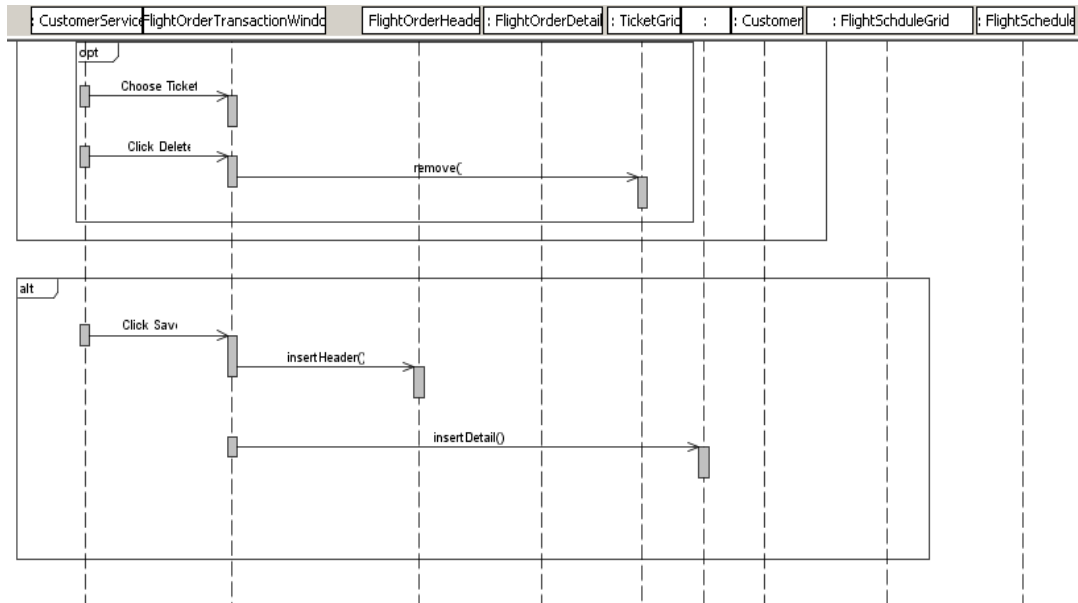
- t. Task 20 – Create Asynchronous Message to FlightOrderTransactionWindow from CustomerService for ClickSave



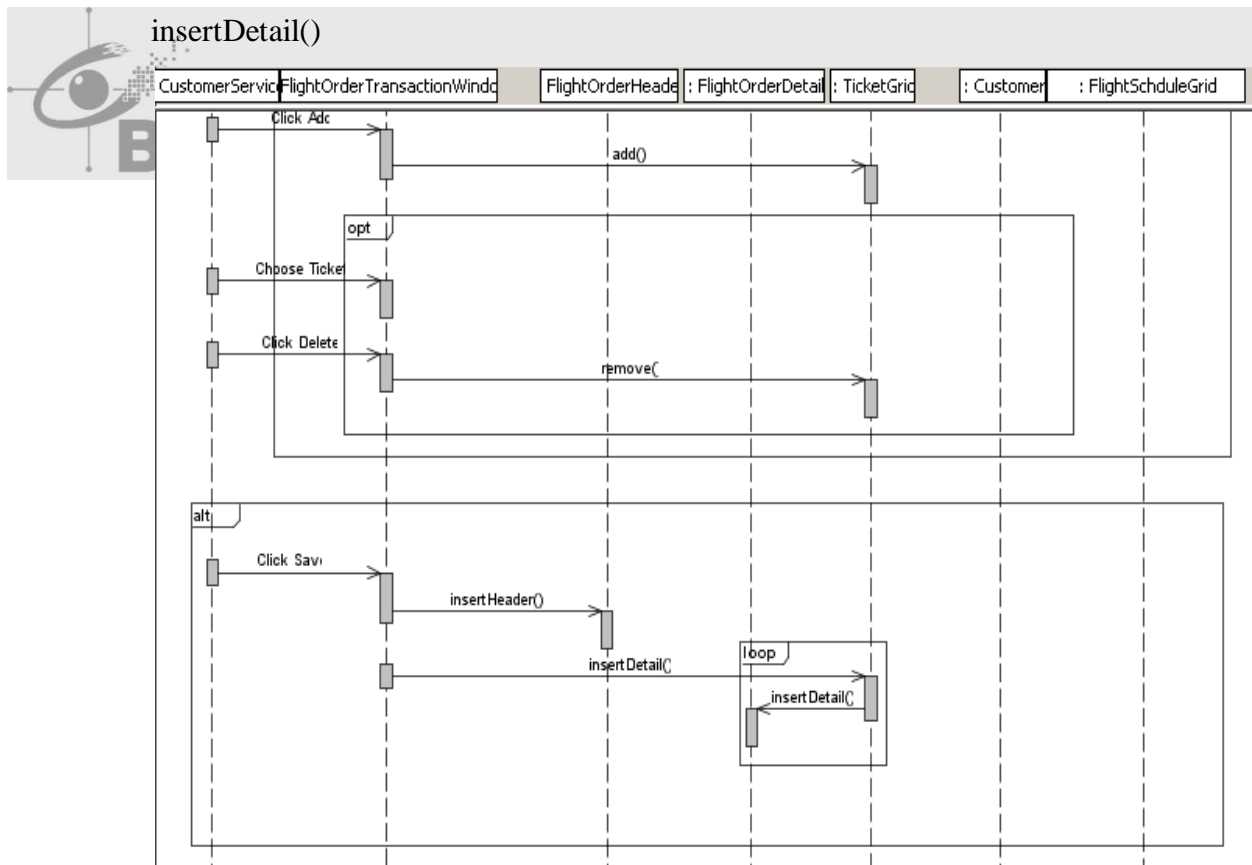
- u. Task 21 – Create Asynchronous Message from FlightOrderTransactionWindow to FlightOrderHeader for insertHeader()



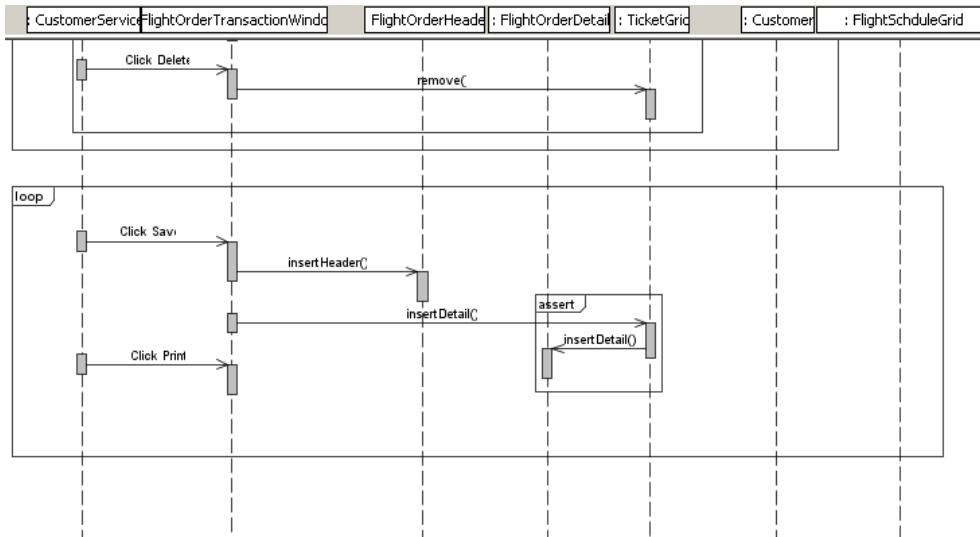
- v. Task 22 – Create Asynchronous Message from FlightOrderTransactionWindow to TicketGrid for insertDetail()



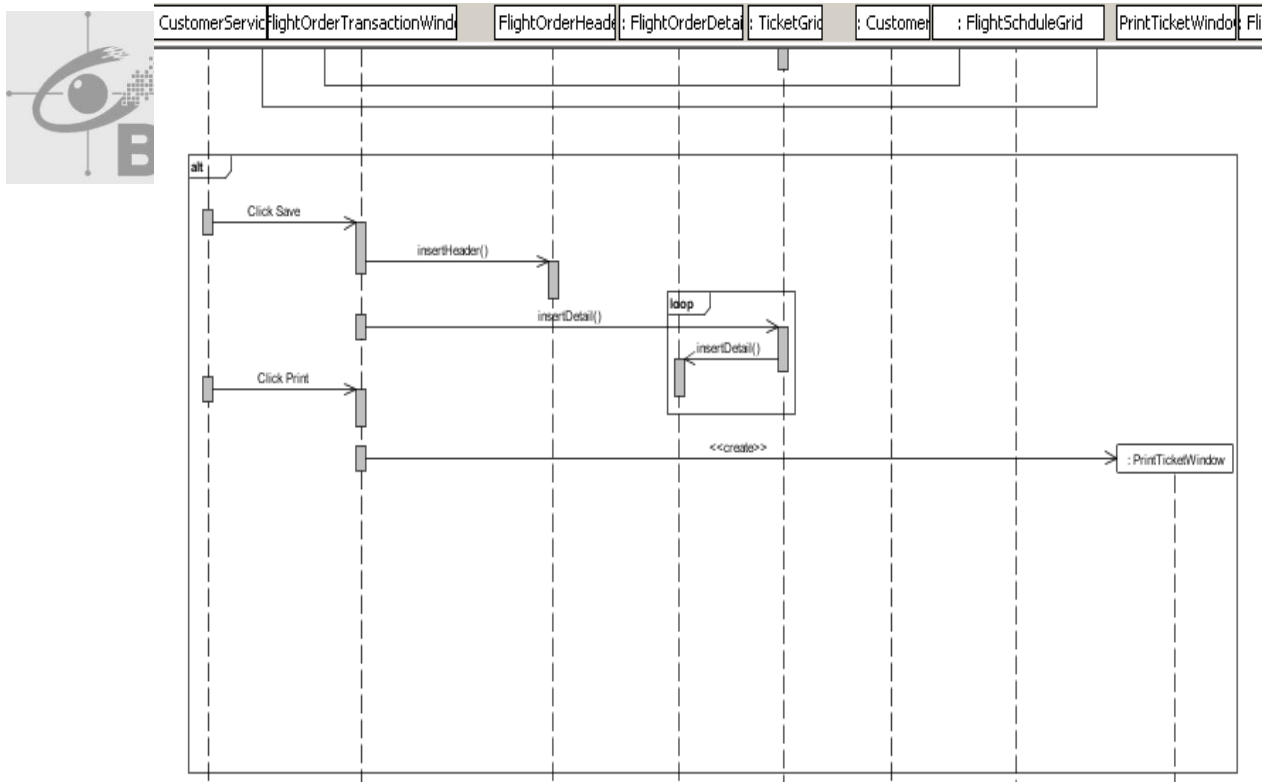
- w. Task 23 – Create Asynchronous Message to FlightOrderDetail from TicketGrid for insertDetail()



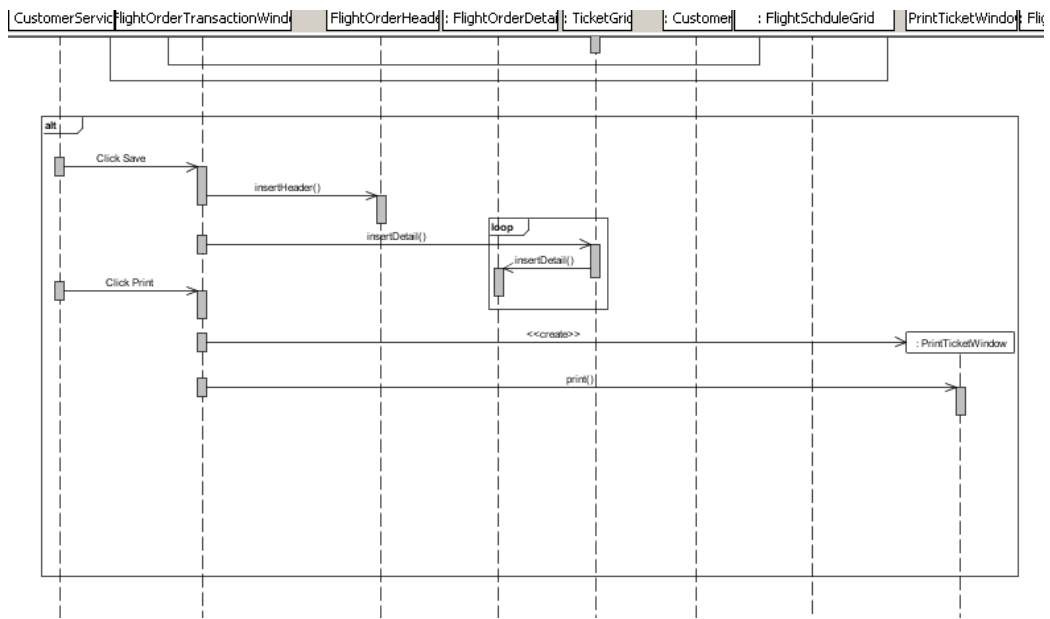
- x. Task 24 – Create Asynchronous Message to FlightOrderTransactionWindow from CustomerService for Click Print



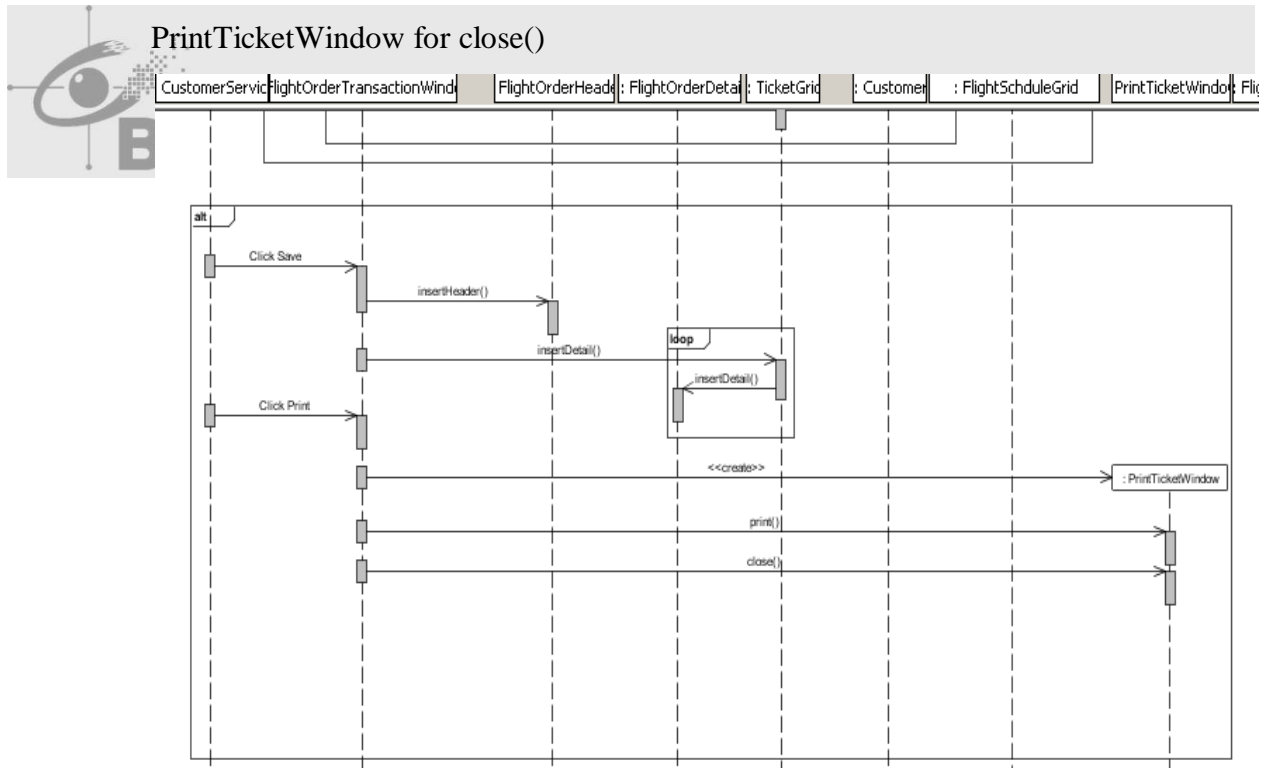
- y. Task 25 – Create Message from FlightOrderTransactionWindow to PrintTicketWindow



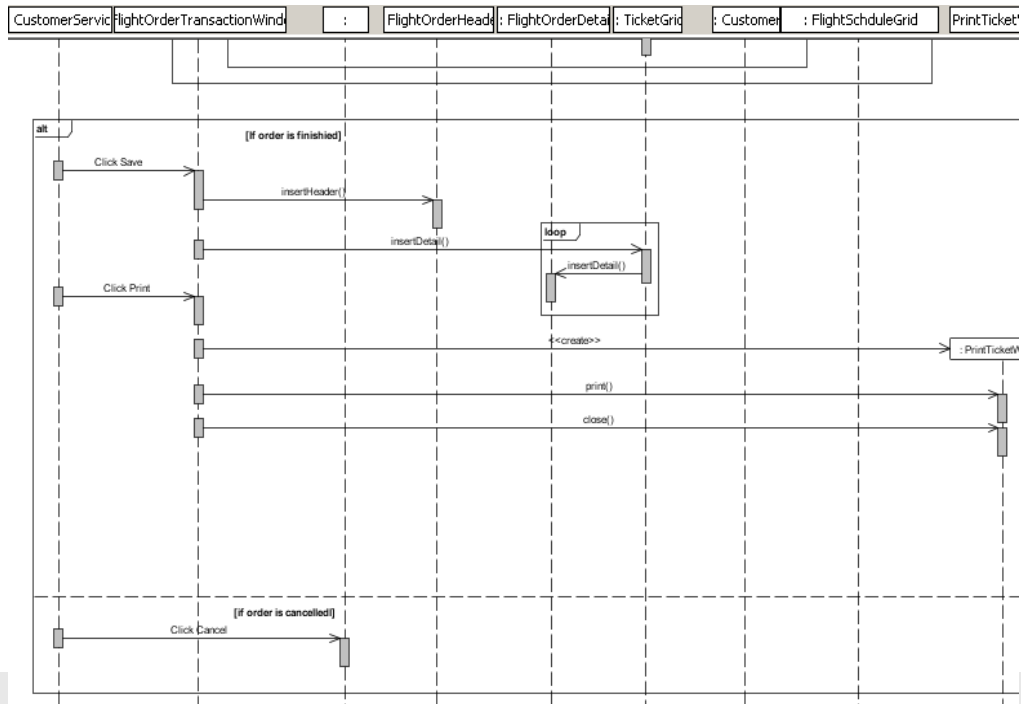
- z. Task 26 – Create Asynchronous Message from FlightOrderTransactionWindow to PrintTicketWindow for print()



- aa. Task 27 – Create Asynchronous Message from FlightOrderTransactionWindow to PrintTicketWindow for close()

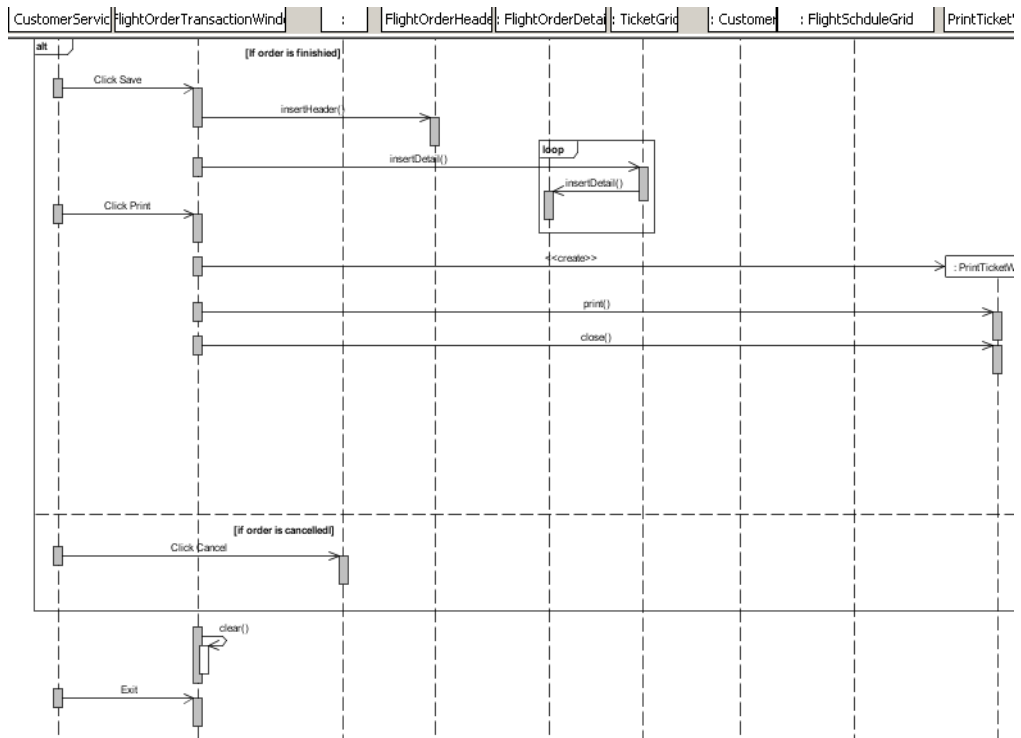


bb. Task 28 – Create Asynchronous Message to FlightOrderTransactionWindow from CustomerService for Click Cancel

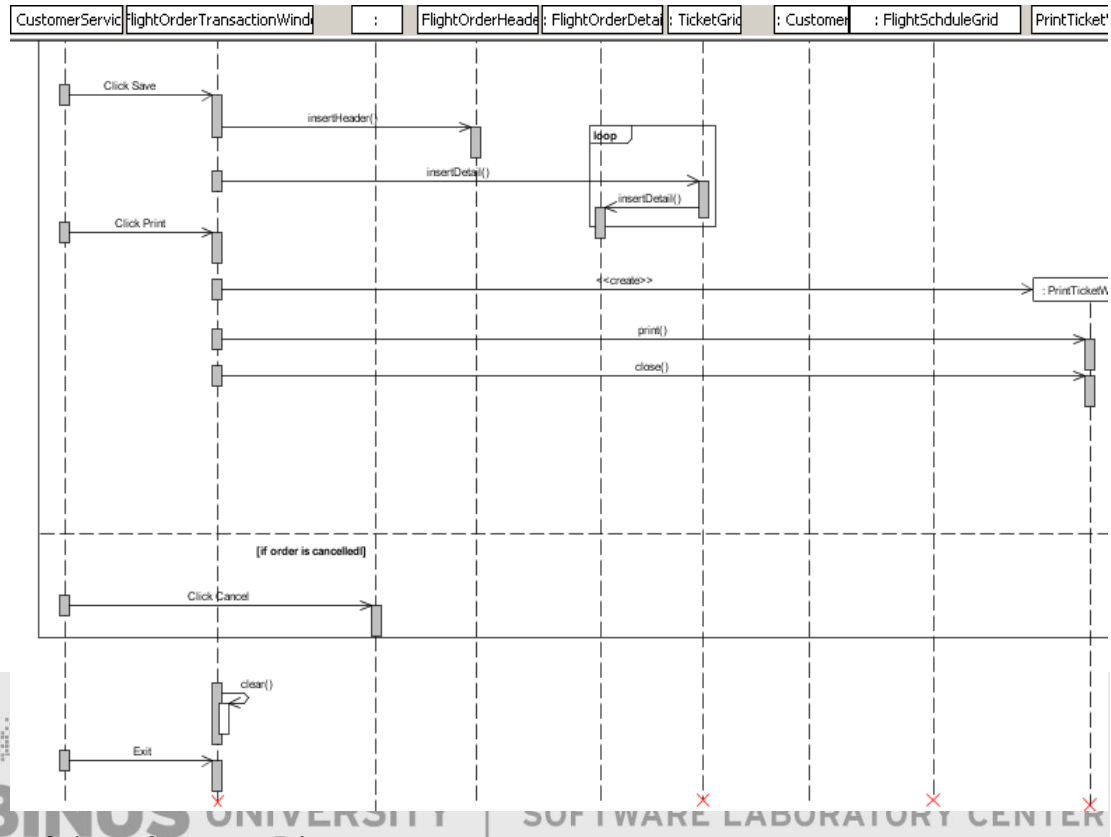


cc. Task 29 – Create Recursive clear() at FlightOrderTransactionWindow

dd. Task 30 – Create Asynchronous Message to FlightOrderTransactionWindow from CustomerService for exit



ee. Task 31 – Create destory for FlightOrderTransactionWindow, TicketGrid, FlightSchduleGrid, PrintTicketWindow



Explanation of above Sequence Diagram

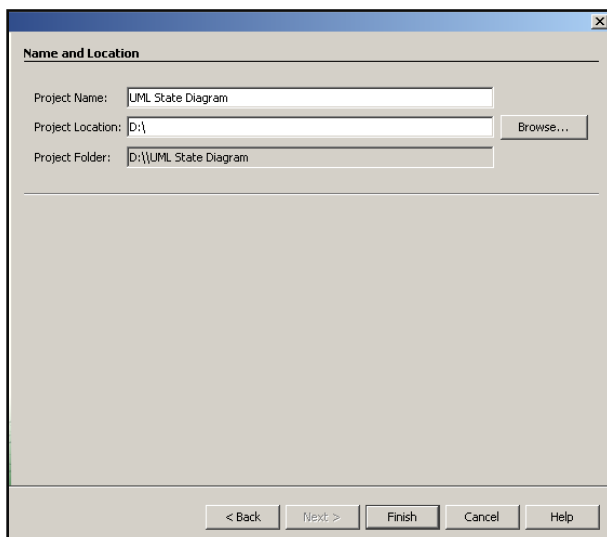
1. CustomerServiceDivision opens the FlightOrderTransactionWindow in order to input the FlightOrderTransaction.
2. The window generates current date.
3. The window retrieves the lastOrderCode in order to generate OrderCode, and then it generates the code.
4. The window creates the FlightSchduleGrid to show the available flight schedule and TicketGrid to show the ticket that member orders based on the schedule that they choose.
5. Then, the window retrieves the available flight schedule from the database.
6. After it, CustomerServiceDivision inputs the MemberCode and the window will check from the database if the code is valid or not.
7. Then, the window will generate the TicketCode and check from the database if there is this generated code or not. If it's in the database, it will be generated again.

8. Then, CustomerServiceDivision chooses the available schedule and click Add button to add the schedule and the generated TicketCode to the TicketGrid.
9. CustomerServiceDivision can delete the ticket that has been added to the grid too by chooses it from the TicketGrid then click Delete.
10. If the order transaction is finished, CustomerServiceDivision can click Save button to insert the order transaction data to the database, and then click Print button to print the tickets(the window will show the PrintTicketWindow, then print each ticket, after that, it will be closed automatically).
11. Or if the order transaction is cancelled, CustomerServiceDivision can click Cancel button and it won't insert anything or printany thing, because it has been cancelled.
12. After it, all of data or field in the window will be cleared and the window will exit.
13. After all of the sequence processes end, the OrderTransaction Window, Ticket Grid, FlightSchedule Grid and the Print Ticket Window will be destroyed.

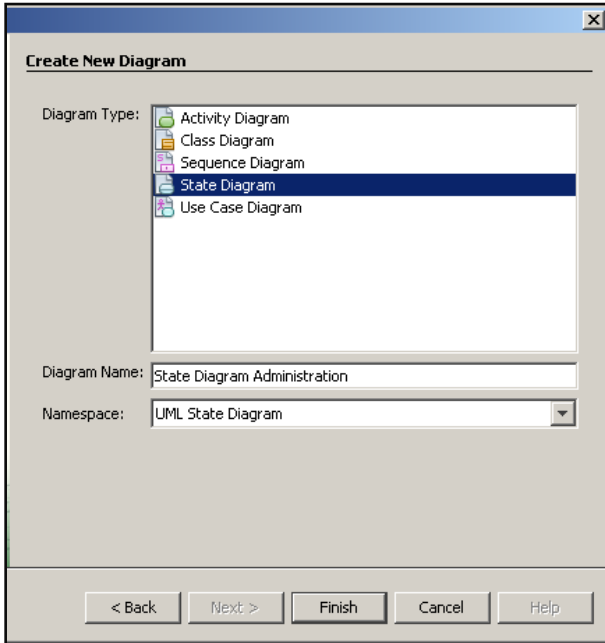
Exercise 05 – Make State Diagram

a. Task 01 – Create New Project SequenceDiagram

1. Run NetBeans from Start Menu, Open MenuFile -> NewProject
2. On New Project Window, choose categories UML the choose Projects Java-PlatformModel, Then Press Next
3. Enter your project name “UMLStateDiagram”
4. Browse Location of your project in D:\

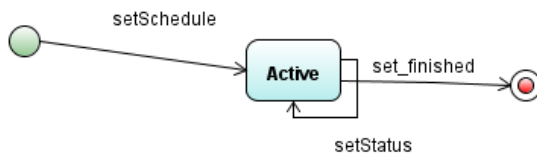


5. Then Press Finish
6. At Diagram Type, choose StateDiagram

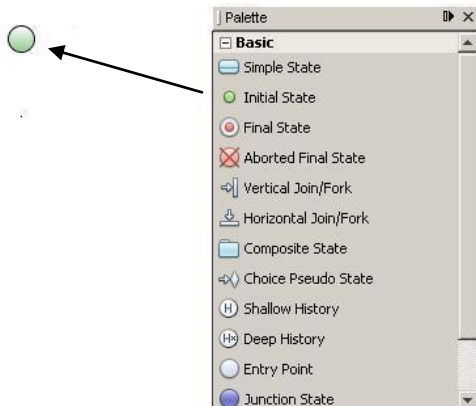


7. Then Press Finish

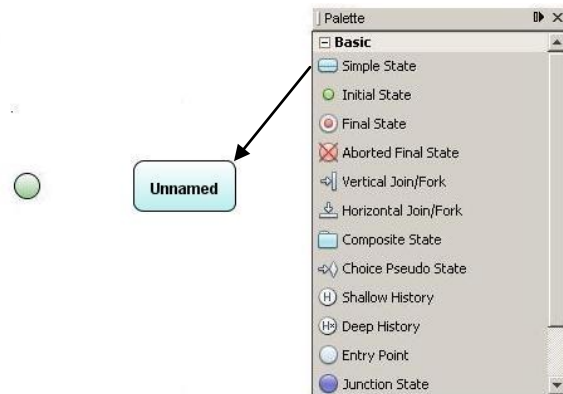
b. Task 02 – Create StateDiagram for Administration



1. Drag InitialState from Pallete–Basic to your worksheet



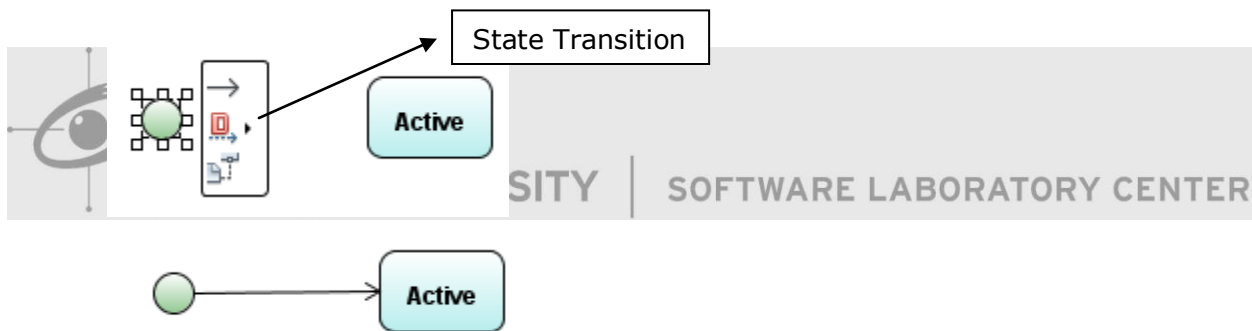
2. Then drag SimpleState from Palette–Basic to your worksheet



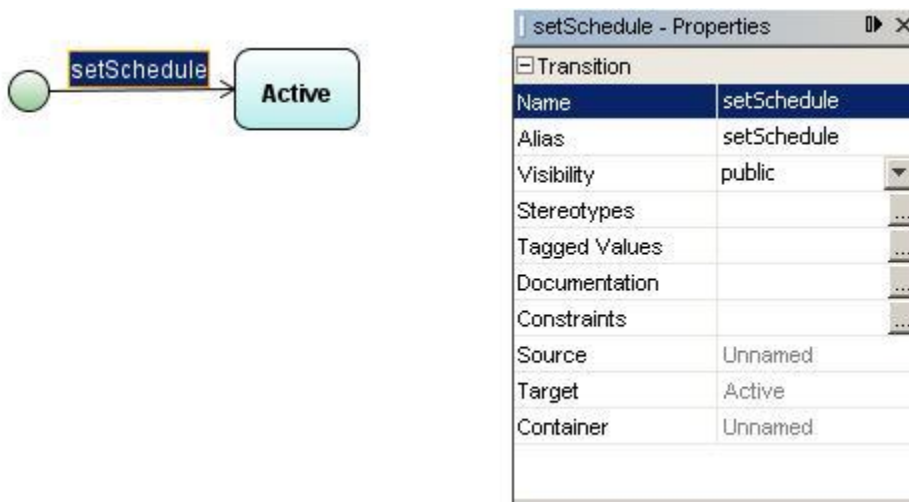
3. Then rename the text“Unnamed” with doubleclick at that text



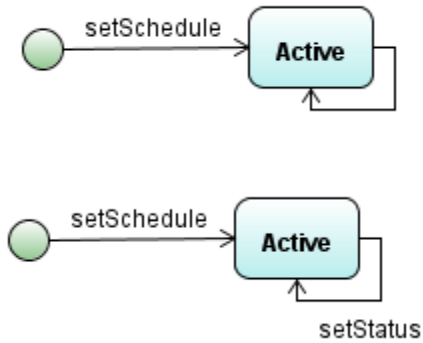
4. Then give statetransition from initialstate to Active



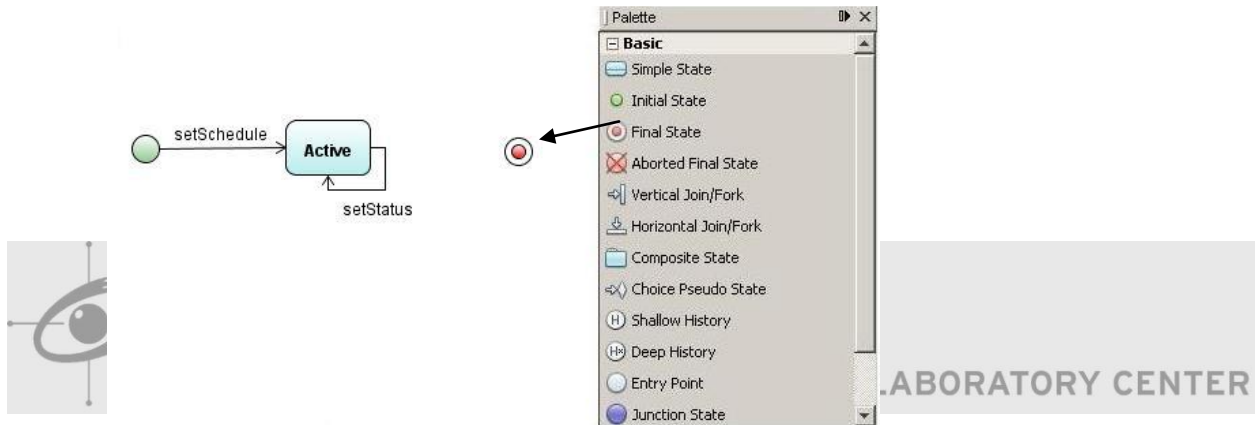
And give the operation that you have made from clasdiagram with click at the arrow, and you will see the properties. And give the name it.



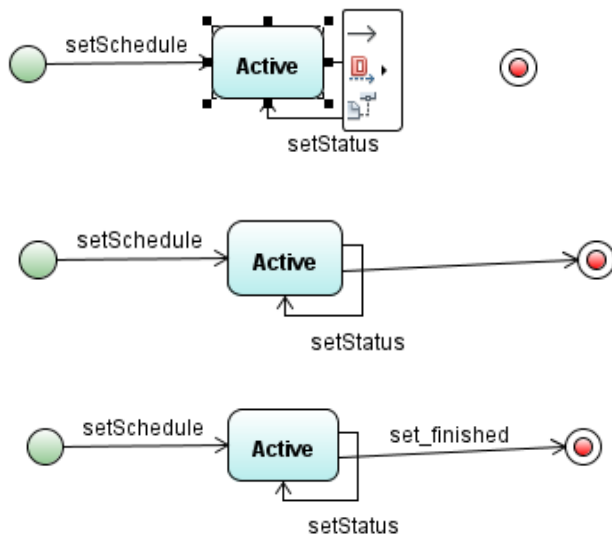
Then create the operation from Active to itself and give the name for the operation.



5. Then drag final state from Pallet–Basic to your worksheet



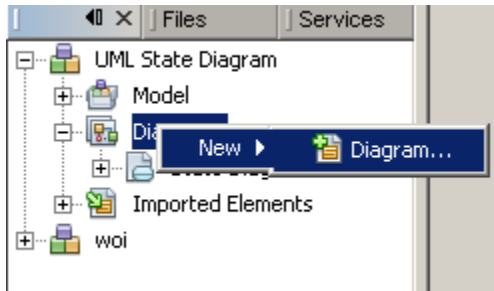
6. Then give the your assumptions because the operation has been completed from Active to finalstate. Drag it from Active to finalstate



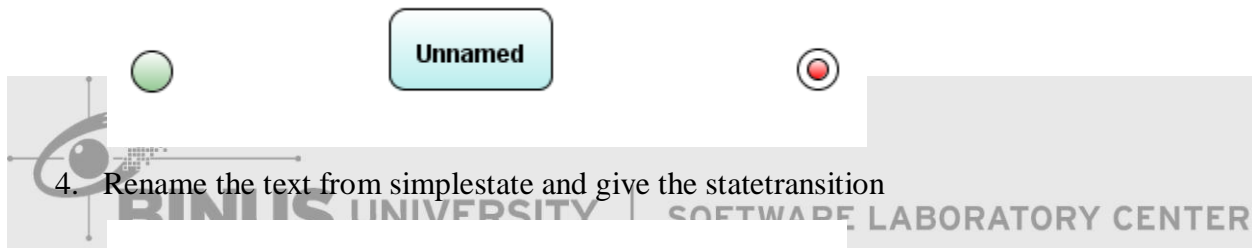
c. Task 03 – Create StateDiagram for CustomerService



1. Create New Diagram with right click on folder Diagram, choose New



2. Then repeat such steps have been performed above
3. Drag InitialState, SimpleState, FinalState to your worksheet,



4. Rename the text from simplestate and give the statetransition



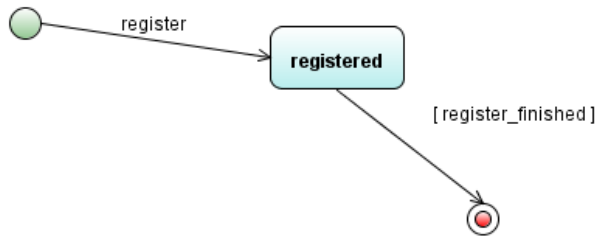
5. Give the operation name that from your classdiagram CustomerService



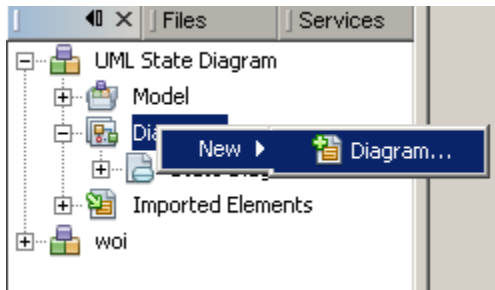
6. Give the assumptions from inserted to finalstate



d. Task 04 – Create StateDiagram for Customer



1. Create New Diagram with right click on folder Diagram, choose New



2. Then repeat such steps have been performed above
3. Drag InitialState, SimpleState, FinalState to your worksheet,



4. Rename the text from simplestate and give the statetransition



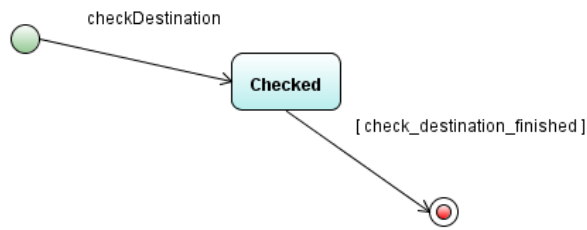
5. Give the operation name that from your classdiagram Customer



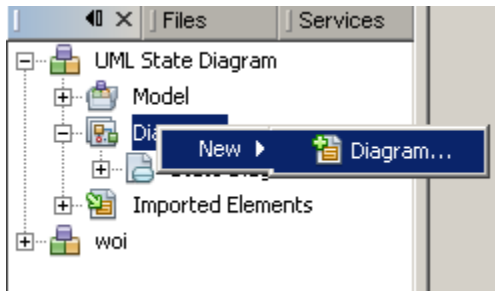
6. Give the assumptions from registered to finalstate



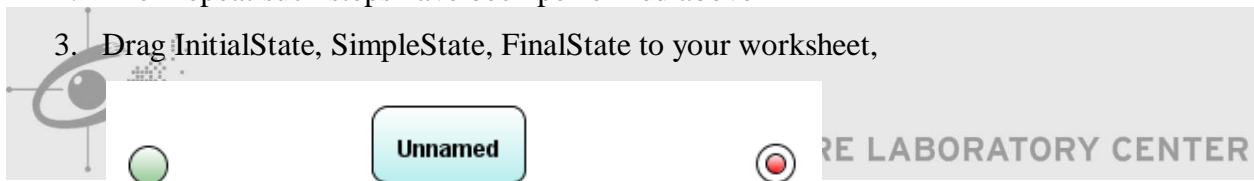
e. Task 05 – Create StateDiagram for DetailFlight



1. Create New Diagram with right click on folder Diagram, choose New



2. Then repeat such steps have been performed above
3. Drag InitialState, SimpleState, FinalState to your worksheet,



4. Rename the text from simplestate and give the statetransition



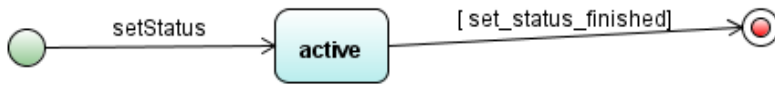
5. Give the operation name that from your classdiagram DetailFlight



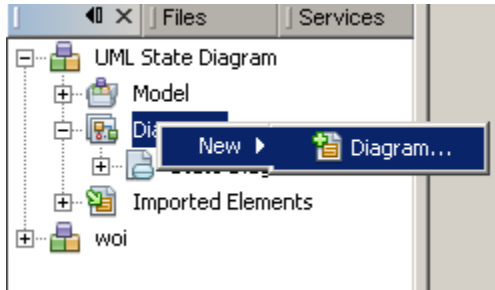
6. Give the assumptions from checked to finalstate



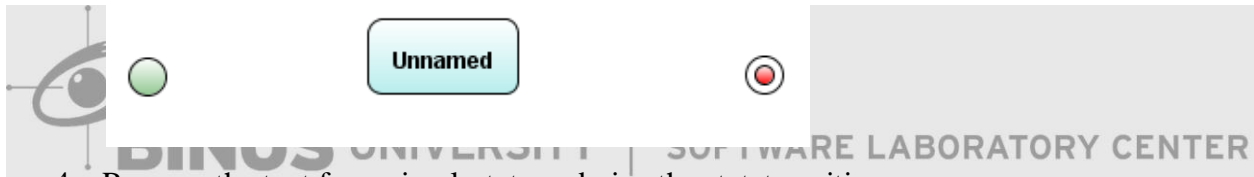
f. Task 06 – Create StateDiagram for DomesticFlight



1. Create New Diagram with right click on folder Diagram, choose New



2. Then repeat such steps have been performed above
3. Drag InitialState, SimpleState, FinalState to your worksheet,



4. Rename the text from simplestate and give the statetransition



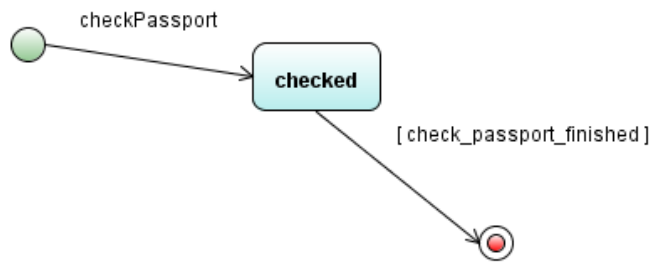
5. Give the operation name that from your classdiagram DomesticFlight



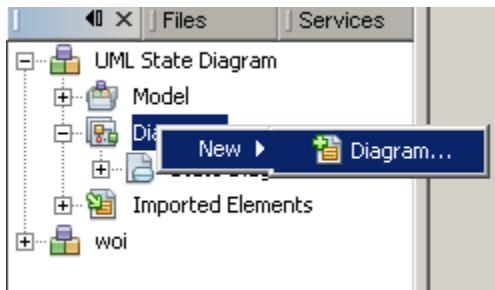
6. Give the assumptions from active to finalstate



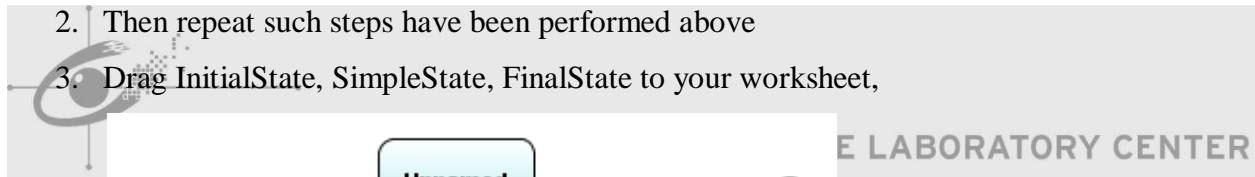
g. Task 07 – Create StateDiagram for OverseasFlight



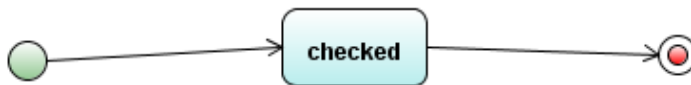
1. Create New Diagram with right click on folder Diagram, choose New



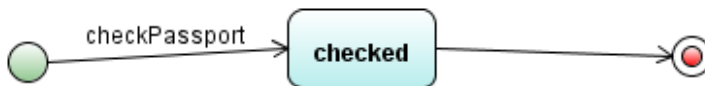
2. Then repeat such steps have been performed above
3. Drag InitialState, SimpleState, FinalState to your worksheet,



4. Rename the text from simplestate and give the statetransition



5. Give the operation name that from your classdiagram OverseasFlight



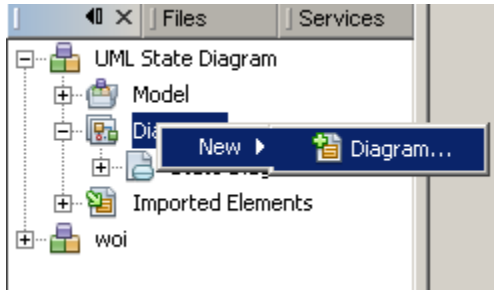
6. Give the assumptions from checked to finalstate



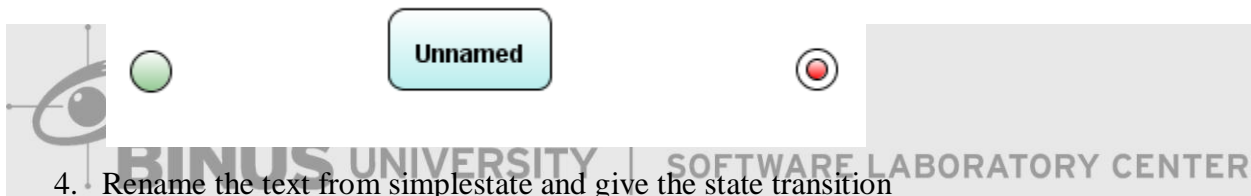
h. Task08 – Create StateDiagram for Plane



1. Create New Diagram with right click on folder Diagram, choose New



2. Then repeat such steps have been performed above
3. Drag InitialState, SimpleState, FinalState to your worksheet,



4. Rename the text from simplestate and give the state transition



5. Give the operation name that from your classdiagram Plane



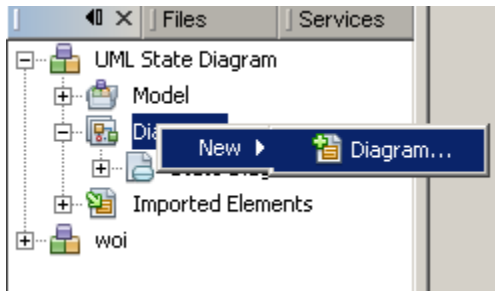
6. Give the assumptions from active to finalstate



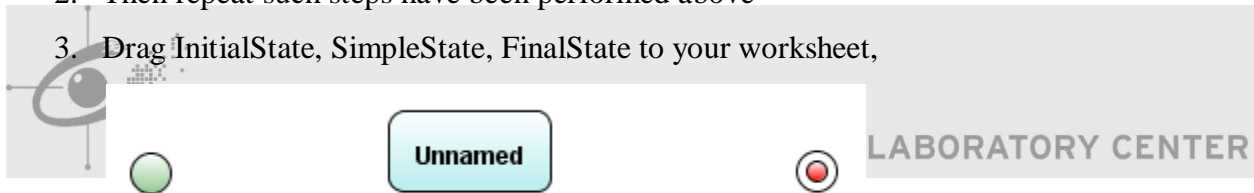
i. Task09 – Create StateDiagram for FlightOrder



1. Create New Diagram with right click on folder Diagram, choose New



2. Then repeat such steps have been performed above
3. Drag InitialState, SimpleState, FinalState to your worksheet,



4. Rename the text from simple state and give the state transition



5. Give the operation name that from your class diagram FlightOrder



6. Give the assumptions from added to finalstate

