Unit 1

Tags	Self note

Unit 1: GUI Programming (12 Hrs.)
Introducing Swing; Creating a Frame; Displaying Information in a Component; Working with 2D Shapes; Using Color; Using Special Fonts for Text; Displaying Images; Event Handling: Event Handling Basics, Event Classes, Event Listeners and Adapter Classes; Swing and the MVC Design Pattern; Layout Management; Basic Swing Components

JFC
- ⇒ AWT
- ⇒ Swing
- ⇒ Difference between AWT and Swing

Basic Termiologies
- ⇒ Components
- ⇒ Container
  - → Window
  - → Panel
  - → Frame
- ⇒ Common methods of component class (VIMP)
  - → Code exmaple inside Main method()
  - → Code example inside Constructor()

Swing Components / Classes
- ⇒ JFrame
- ⇒ JLabel
- ⇒ JTextFiled
- ⇒ JPasswordField
- ⇒ JButton

Event Handling (IMP)
- ⇒ Event
- ⇒ Event Handling Introduction
- ⇒ Event Classes and its description
- ⇒ Event Listener Interfaces
- ⇒ Java Event Classes and Listner Interfaces
- ⇒ Registration Methods
- ⇒ Code For Handling Event
  - → Note:
  - → Action Listner Interface
  - → Mouse Listener Interface
  - → Item Listener Interface

Java Adapter Classes
- ⇒ Code for adapter

- → Mouse Adapter
- → Mouse Motion Adapter
- → Window Adapter

⇒ Pros of Using Adapter Classes

Action Command

⇒ Process:

⇒ Code

Layout Management

⇒ Border Layout

- → Fields are :
- → Code for border layout

⇒ Flow Layout

- → Fields are:
- → Constructors are:
- → Code for Flow layout

⇒ Grid Layout

- → Constructor are:
- → Code for Grid layout

⇒ GridBagLayout

- → Code for Grid Bag Layout

⇒ Group Layout

- → Methods
- → Code

2D shape

⇒Graphics context

⇒ Method

⇒ Color Class

- → Code for 2d shapes
- → code for Font adding

Displaying Image

⇒ URL image load

⇒ File image load

⇒ Preloading images

- → Code

MVC Architecture

⇒ Model

⇒ View

⇒ Controller

- → Advantage / Benefits
- → Disadvantage

JFC

- Java foundation classes

- set of GUI components which simplifiy development of desktop application
- Components :
  - Swing
  - AWT
  - Java 2D
  - Drag & Drop

⇒ AWT

- Java Abstract Window Toolkit
- Its a API to develop GUI or window based application
- Its a platform dependent, components are displayed according to view of operating system
- Its heavy wight
- **java.awt package** provides classess for AWT api such as
  - Label
  - TextField
  - TextArea
  - Radio Button
  - CheckBox
  - List etc..

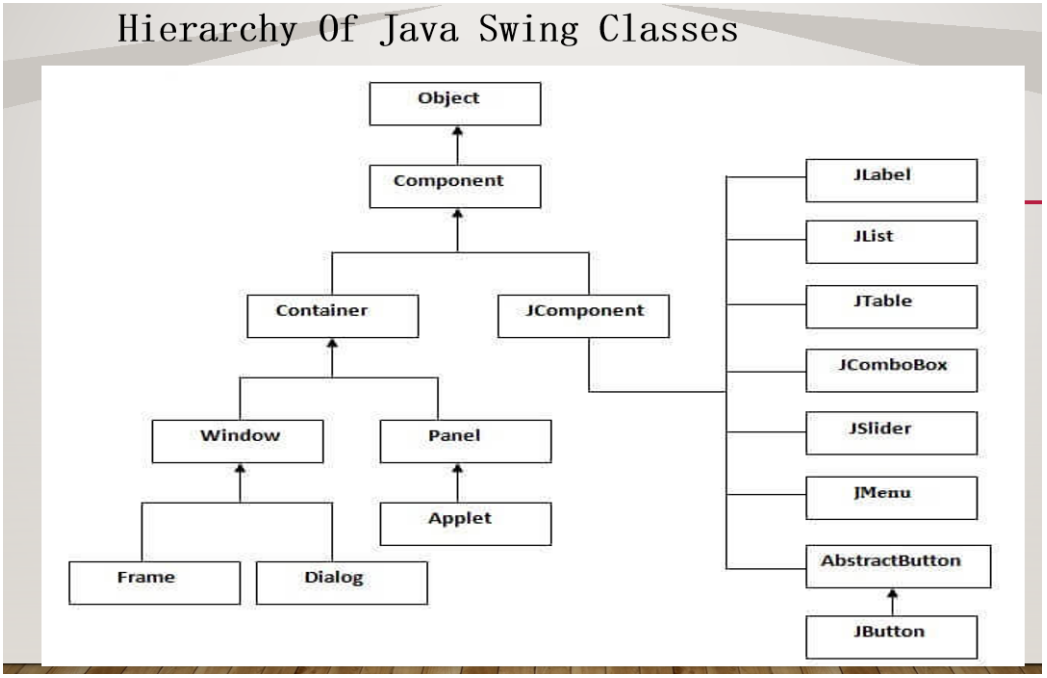
⇒ Swing

- Its a part of JFC that is used to create GUI or Window based applications
- Its built on top of AWT
- Its Lightweight & Platform independent
- java.swing package provies classes for swing api such as
  - JLabel
  - JTextFiled
  - JTextArea
  - JRadioButton

- JCheckbox
- JMenu

⇒ Difference between AWT and Swing

AWT	Swing
Platform dependent	Platform independent
Componets are heavyweight	Components are lightweight
Doesnot support pluggable look and feel	Support’s pluggable look and feel
Has less componets than swing	Has more components than awt
Does not follow MVC	Follows MVC
EG: Label,TextField , TextArea etc	EG: JLabel, JTextField, JTextArea



Basic Termiologies

⇒ Components

- All elements like button, text fields, text area, scroll bar etc are called components
- Every components mut be inside the container to show on the screen

⇒ Container

- A component that can contain another components like buttons, textfield, textarea, lables etc.
- It extends Container class are known as container such as Frame, Dialog and Panel
- Four types of containers in AWT:
  - Window
    - Frame
    - Dialog
  - Panel

→ Window

- A container that have no borders and menu bars
- Must use frame, dialog for creating window
- We need to create instance of Window class to create this container.

→ Panel

- A container that doesnot contain title bar or menu bar
- It can contain other components like button, text field etc

---

→ Frame

- A container that contain titile bar and border and can have menu bars.
- It can contain other componets like button, text field, scroll bar etc.
- There are two ways to create frame:
  - ***Creating object of Frame class***

```
Frame f = new Frame();
```

> If we use object then we have to do i.e `f.setLayout(null)`, `f.setVisible(true)`

- ***Extending Frame class***

```
class Example extends Frame{}
```

> If we use extend then we can directly set i.e `setLayout(null)`, `setVisible(true)`.

⇒ Common methods of component class (VIMP)

Method	Description
public void **add(Component c)**	add component on another component
public void **setSize(int width, int height)**	sets size of component
public void **setLayout(LayoutManager m)**	sets layout manager for component
public void **setVisible(boolean b)**	set visibility of component its false by default.
public void **setBounds(int x,int y,int width, int height)**	automatically decide the position and size of the added components.

→ Code exmaple inside Main method()

```
class Example{
    public static void main(String [] args){
        Jframe f = new JFrame();
        f.setSize(400,500);
        JButton b = new JButton("click");
        b.setBounds(130,100,100,40);
        f.add(b);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

→ Code example inside Constructor()

```
class Example{
    public Example(){
        Jframe f = new JFrame();
        f.setSize(400,500);
        JButton b = new JButton("click");
        b.setBounds(130,100,100,40);
        f.add(b);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String [] args){
        new Example();
    }
}
```

Swing Components / Classes

⇒ JFrame

- Jframe is a type of container which inherits the java.awt.Frame class
- It works like main window where other components are added i.e labels, button, textfileds

→Constructor

Constructor	Description
JFrame()	Creates frame with no title which is initially invisible
JFrame(GraphicsConfiguration gc)	Creates frame in specified GraphicsConfiguration of a screen device and blank title
JFrame(String title)	Creates frame with specified title which is initially invisible
JFrame(String title, GraphicsConfiguration gc)	It creates frame with specified title and specified GraphicsConfiguration of screen device.

⇒ JLabel

- Its a component for placing text in a container
- Used to display single line of read only text
- Text can be change by application (java program logic) but a user cannot edit it directly.

→Constructor

Constructor	Description
JLabel()	Creates Jlabel instance with empty string for title and no image
JLabel(String s)	Creates Jlabel instance with spcified title
JLabel(Icon i)	Creates Jlabel instance with specified image
JLabel(String s,Icon i,int horizontalAlignment)	It creates Jlabel instance with specified title,image and horizontal alignment

→Methods

Methods	Description
String getText()	returns text of the displayed label
void setText(String s)	set text in the label
void setHorizontalAlignment(int alignment)	set alignment of label’s content along x axis
Icon getIcon()	returns graphic image of displayed label
int getHorizontalAlignment()	returns alignment of label’s content along x axis

⇒ JTextFiled

- A component that allows editing of single line text.

→ Constructor

Methods	Description
JTextField()	creates Text Field

Methods	Description
JTextField(String text)	create text field initialized with specified text
JTextField(int columns)	create text field initialized with specified columns
JTextField(String text, int columns)	create text field initialized with specified text and columns

→ Methods

Methods	Description
void **addActionListner(ActionListner l)**	To add action listner to text field
Action getAction()	To returns Action for specified ActionEvent source, or null if no action is set
void setFont(Font f)	To Set current font
void **removeActionListner(ActionListner l)**	To remove spcified action listner so it no longer recieve action events

---

⇒ JPasswordField

- A component that is specialized text component for password entry

→ Constructor

Constructor	Description
JPasswordField()	creates JPasswordField, with a default document, null starting with text string and 0 column width
JPasswordField(String text)	create JPasswordField initialized with specified text
JPasswordField(int columns)	create JPasswordField with specified number of columns
JPasswordField(String text, int columns)	create JPasswordField with specified text and number of columns

---

⇒ JButton

- A component with labled button that has platform independent implementation.

→ Constructor

Constructor	Description
JButton()	creates button with no text and icon
JButton(String text)	create button with specified text
JButton(Icon i)	create button with specified icon

→ Methods

Methods	Description
void setText(String s)	To set text on button
String getText()	To return text of that button
void setEnabled(boolean b)	To enable or disable button
void setIcon(Icon b)	To set icon on button
Icon getIcon()	To return icon of that button
void addActionListner(ActionListner a)	To add action listner to button

---

Event Handling (IMP)

⇒ Event

- Event are actions that happens within a program, such as
  - clicking a button
  - typing in text field
  - closing window

→ Event Sources:

- A source is an object that generates event.
- Occurs when state of object changes
- Sources may generate more than one type of event
- EG:
  - Button
  - Combo boxes etc.

→ Event Listener:

- A object that is notified when an event occurs.
- They are
  - ActionListener
  - KeyListener

→ Key Event Classes

- ***Event Object***

  > Super class for all event classes.

- ***AWTEvent***

  > Subclass of EventObject
  > Superclass for all awt events.

⇒ Event Handling Introduction

- It is a mechanisum that controls what happns when an event occurs.
- Java uses **Delegation Event Model** to handle events,
  - It defines **standard mechanism** to generate and handle events.
  - In this model, a source generates event and sends it to one or more listeners.
  - listner simply wait until it receives event, after receiving the listner processes event and returns.
  - **Benefit, is that UI logic is completly seperated from logic that generate event.**

→ Steps to perform event handling

1. Implement Listner interface and override its method (i.e : implements ActionListner)
2. Register component with Listener (i.e: **.addActionListner**)
3. Put proper code in required overriden method

⇒ Event Classes and its description

Event Classes	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected
MouseEvent	Generated when mouse is dragged, moved, clicked, pressed or released. Also generated when the mouse enters or exit a component.
MouseWheelEvent	Generated when the mouse wheel is moved
KeyEvent	Generated when input is received from the keyboard.
ItemEvent	Generated when a check box, list item or checkable menu item is selected
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed,deactivated,open or quit.
ComponentEvent	Generated when a component is hidden, moved, resized or becomes visible.
ContainerEvent	Generated when a component is added to or removed from container.
FocusEvent	Generated when a component gains or losses keyboard focus

> Focus : ActionEvent, Mouse, Key, Item, Component, Container

---

⇒ Event Listener Interfaces

- To handle events we must implement event listner interface.
- Two major Requirements:
  1. It must be registered with one or more sources (button,textfield) to receive notification
  2. It must implement methods to recieve and process these notification.

→ Interfaces and their methods

> If you remember the Events then you can easily remember listner

Easy to remember

- `ActionEvent` → `ActionLisnter`
- `MouseEvent` → `MouseListner`
- `KeyEvent` → `KeyListner`

- `ItemEvent` → `ItemListner`
- `ComponentEvent` → `ComponentListner`
- `ContainerEvent` → `ContainerListner`

Listener Interfaces	Description
ActionListener	Declares one method to receive action events. Action events are generated when button is pressed
MouseListener and MouseMotionListener	Defines five methods to recongnize when the mouse is clicked, enters a component, exits a component, is pressed or released. void mouseClicked(MouseEvent me) void mouseEntered(MouseEvent me) void mouseExited(MouseEvent me) void mousePressed(MouseEvent me) void mouseReleased(MouseEvent me)
MouseWheelListener	MouseWheelListener
KeyListener	KeyListener
ItemListener	ItemListener
TextListener	TextListener
AdjustmentListener	AdjustmentListener
WindowListener	WindowListener
ComponentListener	ComponentListener
ContainerListener	ContainerListener
FocusListener	FocusListener

⇒ Java Event Classes and Listner Interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

⇒ Registration Methods

> This methods are added to register the action events

```
Jbutton button = new JButoon("Click me")
button.addActionListner(this);
```

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

⇒ Code For Handling Event

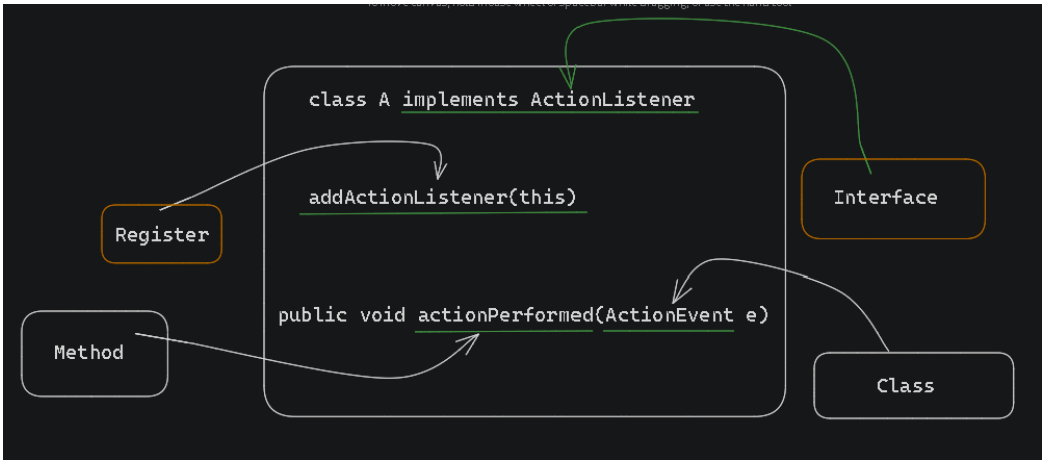
After implementing interface and registering source objects with listener, we can hande event by code in following places:

1. Same class
2. Inner class
3. Outer class

→ **Note:**

> Always remember we have three things when we code for event handling as they are:

- Implement Interface
- Register
- Method for performing on that event



The structure is same, just gets changed when the Action/listener is different.

→ Action Listner Interface

- we need to use method `public void actionPerform(ActionEvent e)`

→ WAP to implement ActionListener in Same Class

```
class Example extends JFrame implements ActionListner{
    JTextField text = new JTextField(); //this should be outsid
e as it beign used inside constructor and actionperformed
    public Example(){
        JButton button = new JButton("Click me");
        add(text);
        add(button);
        button.addActionListner(this);
        text.setBounds(10,10,100,100);
        button.setBounds(20,20,50,50);
        setSize(500,500);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
        text.setText("Welcome");
    }
    public static void main(String[] args){
        new Example();
    }
}
```

→ WAP to implement by Inner Class

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class Example extends JFrame {
    JTextField text = new JTextField(); // This should be outsi
de as it is being used inside the constructor and actionPerform
ed and inner class

    class Handler implements ActionListener {
        public void actionPerformed(ActionEvent e) {
        text.setText("Welcome");
        }
    }
    public Example() {
        JButton button = new JButton("Click me");
        add(text);
        add(button);
        button.addActionListener(new Handler(this));
        text.setBounds(10, 10, 100, 100);
        button.setBounds(20, 120, 100, 50); // Adjusted positio
n and size
        setSize(500, 500);
        setLayout(null);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Ensu
```

```
re the application exits on close
    }

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

→ WAP to handle by Outer Class

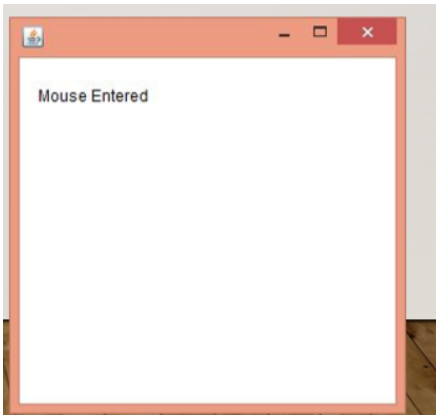
```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class Example extends JFrame {
    JTextField text = new JTextField(); // This should be outsi
de as it is being used inside the constructor and actionPerform
ed
    public Example() {
        JButton button = new JButton("Click me");
        add(text);
        add(button);
        button.addActionListener(new Handler(this));
        text.setBounds(10, 10, 100, 100);
        button.setBounds(20, 120, 100, 50); // Adjusted positio
n and size
        setSize(500, 500);
        setLayout(null);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Ensu
re the application exits on close
    }

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

class Handler implements ActionListener {
    Example example;

    Handler(Example e) {
```

```
        example = e;
    }

    public void actionPerformed(ActionEvent e) {
        example.text.setText("Welcome");
    }
}
```

→ Mouse Listener Interface

> We need to override the methods mouseClicked, mouseEnter, mouseExit, mousePress, mouseRelease

```
class Example extens JFrame implements MouseListener{
    JLabel label = new Jlabel();
        public Example(){
            add(label);
            label.setBounds(10,10,50,50);
            setSize(500,500);
            setLayout(null);
            setVisible(true);
        }

        public void mouseClicked(MouseEvent e){
            label.setText("Mouse clicked");
        }
        public void mouseEnter(MouseEvent e){label.setText("Mouse
entered"); }
        public void mouseExit(MouseEvent e){label.setText("Mouse e
xit");}
        public void mousePress(MouseEvent e){label.setText("Mouse
press");}
        public void mouseRelease (MouseEvent){label.setText("Mouse
release");}
        public static void main(String[] args){
            new Example();
        }
}
```



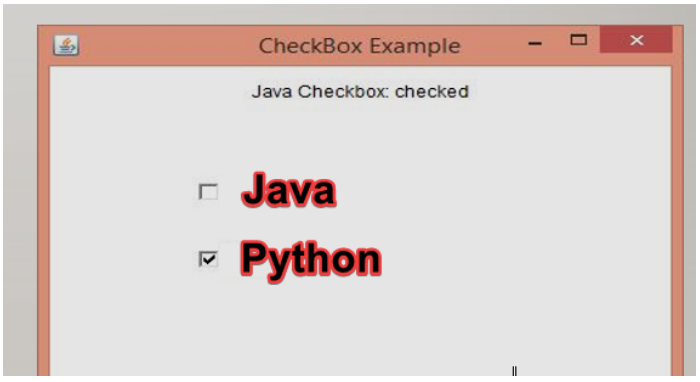
→ Item Listener Interface

We need to remember two things

- e.getSource()
- e.getStateChange() → 0 or 1

```
class Example extens JFrame implements ItemListener{
    JLabel label = new Jlabel();
    Checkbox c1 = new Checkbox("Java");
    Checkbox c2 = new Checkbox("Python");
        public Example(){
            label.setBounds(10,10,50,50);
            c1.setBounds(20,20,50,50);
            c2.setBounds(30,30,50,50);
            add(label);
            add(c1);
            add(c2);
            c1.addItemListener(this);
            c2.addItemListener(this);
            setSize(500,500);
            setLayout(null);
            setVisible(true);
        }

        public void itemStateChanged(ItemEvent e){
            if(e.getSource() == c1){
                label.setText("Java:"+ (e.getStateChange() == 1 ?
"Checked" : "Unchecked" ));
            }
            if(e.getSource() == c2){
                label.setText("Python:" e.getStateChange() == 1 ?
```

```
"checked" : "Unchecked"));
            }
        }

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



Java Adapter Classes

- It provides default implementation of listener interface.
- We will not be forced to implement all methods of listener interfaces.
- No need to implement all methods.
- When we use adapter class we need to extend as its class not interface.

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener

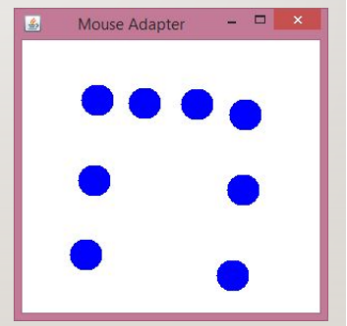
> ***Note:*** **If you were using JFrame by extending then you cannot do that as java doesnot support multiple inheritance**

```
class Example extends JFrame,MouseAdapter { //error
}
```

⇒ Code for adapter

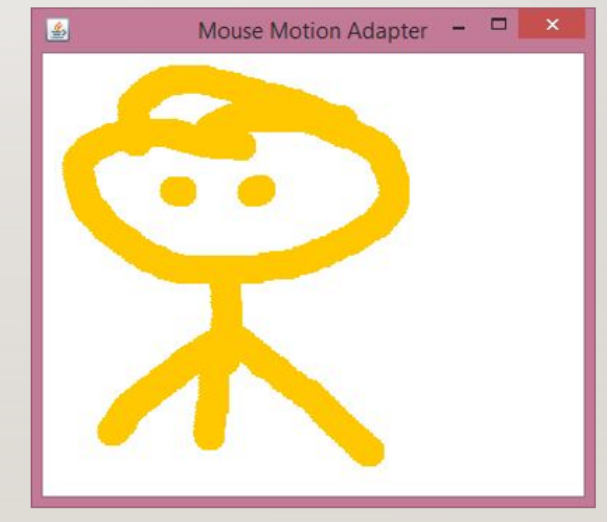
→ Mouse Adapter

```
class Example extends MouseAdapter {

    public Example(){
        JFrame f = new JFrame("Mouse Adapter");
        f.addMouseListener(this);
        f.setSize(500,500);
        f.setLayout(null);
        f.setVisible(true);
    }

    public void mouseClicked(MouseEvent e){
        Graphics g = f.getGraphics();  //graphics instance / objec
t  created
        g.setColor(Color.BLUE);        //setting color blue
        g.fillOval(e.getX(),e.getY(),30,30);  //geting the coordin
ate values of x and y : where it's clicked
    }

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



→ Mouse Motion Adapter

```
class Example extends MouseMotionAdapter{

    public Example(){
        JFrame f = new JFrame("Mouse Motion Adapter");
        f.addMouseMotionListener(this);
        f.setSize(500,500);
        f.setLayout(null);
        f.setVisible(true);
    }

    public void mouseDragged(MouseEvent e){
        Graphics g = f.getGraphics();  //graphics instance / objec
t  created
        g.setColor(Color.Orange);        //setting color blue
        g.fillOval(e.getX(),e.getY(),20,20);  //geting the coordin
ate values of x and y : where it's clicked
    }

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



→ Window Adapter

```
class Example{
    JFrame f;
    Example(){
        f = new JFrame("Window Adapter");
        f.addWindowListener( new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                f.dispose();
            }
        });

        f.setSize(500,500);
        f.setLayout(null);
        f.setVisible(true);

    }

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

⇒ Pros of Using Adapter Classes

1. **Simplified Event Handling**

You can override only methods that are needed.

2. **Reduce boilerplate code.**
3. **Consistent default behavior**

   If you forget to implement override method, it wont cause unexpected behavior or error.

4. **Improve code readability and maintainability**

   Only focusing on methods that we need.

Action Command

- A string that helps to identify component(button, menu or other) that performed action.
- To differentiate between multiple component that might trigger same event listener.

⇒ Process:

- **Setting the Action Command:**
  - You can set the action command for a component using the `setActionCommand(String command)` method.
  - This method associates a custom string (identifier) with the component.
- **Getting the Action Command:**
  - When the event listener is invoked (e.g., button clicked, menu item selected), you can retrieve the action command from the `ActionEvent` object passed to the listener method.
  - You can use the `getActionCommand()` method of the `ActionEvent` object.
- **Using the Action Command:**
  - Inside the event listener's method (e.g., `actionPerformed` for `ActionListener`), you can use the retrieved action command to determine which specific component triggered the event.
  - This allows you to write different code blocks based on the action command, providing more granular control over your application's behavior.

⇒ Code

```
class Example extends JFrame implements ActionListner{
    JTextField tf;
    Example(){
        tf = new JTextField();
        tf.setBounds(10,10,50,100);
        JButton b = new JButton("Click me");
        b.setBounds(20,20,50,100);
        b.addActionListner(this);
        b.setActionCommand("FirstButton");
        add(tf);
        add(b);
        setSize(500,500)
        setLayout(null);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e){
        tf.setText("Welcome"); //not necessary just to show that i
ts working
        String command = e.getActionCommand(); //getting the text
"FirstButton"
        system.out.println("Clicked" + command);
    }

    public static void main(String[] arg){
        new Example();
    }

}
```

Layout Management

- It is used to arrange components in a particular manner.
- It is an interface that is implemented by all classes of layout managers.
- Following are layout managers classes:
  - BorderLayout
  - FlowLayout
  - GridLayout
  - GridBagLayout
  - GroupLayout

⇒ Border Layout

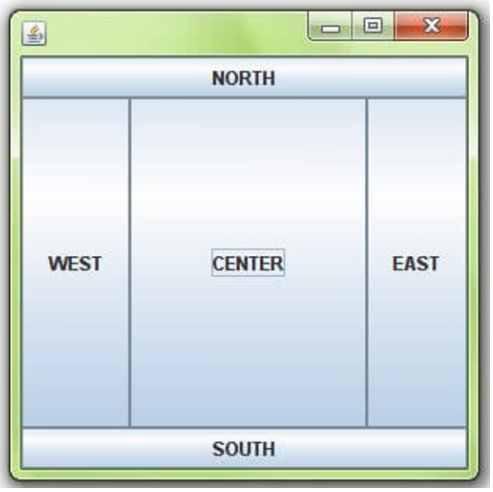
- To arrange components in five regions:
  - North
  - South
  - East
  - West
  - Center
- It is default layout for frame or window.

→ Fields are :

- public static final int North
- public static final int South
- public static final int East
- public static final int West
- public static final int Center

→ Code for border layout

```
import javax.swing.*;
import java.awt.*;

class Example extends JFrame {
    Example() {
        JButton b1 = new JButton
("North");
        JButton b2 = new JButton
("South");
        JButton b3 = new JButton
("East");
        JButton b4 = new JButton
("West");
        JButton b5 = new JButton
("Center");

        add(b1, BorderLayout.NORT
H);
        add(b2, BorderLayout.SOUT
H);
        add(b3, BorderLayout.EAS
T);
        add(b4, BorderLayout.WES
T);
        add(b5, BorderLayout.CENTE
R);

        setSize(500, 500);
        setVisible(true);
    }

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



⇒ Flow Layout

- To arrange component in line, one after another (in flow).

- Default layout for panel or applet.

→ Fields are:

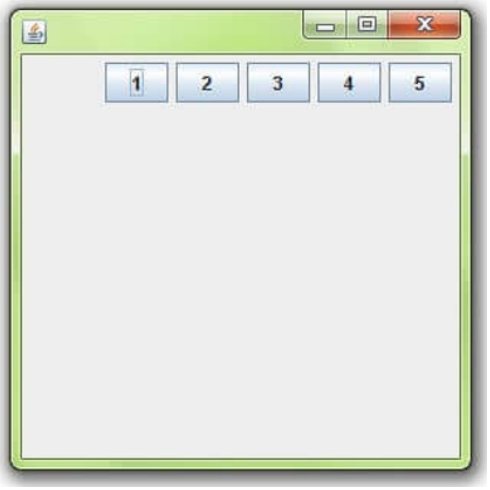
- public static final int LEFT
- public static final int RIGHT
- public static final int Center
- public static final int LEADING
- public static final int TRAILING

→ Constructors are:

- `FlowLayout():` Creates center alignment
- `FlowLayout(int align):` creates with given alignment
- `FlowLayout(int align, int hgap, int vgap):` create with given alignment and with horizontal and vertical gap

→ Code for Flow layout

```
import javax.swing.*;
import java.awt.*;

class Example extends JFrame {
    Example() {
        // Create buttons
        JButton b1 = new JButton
("1");
        JButton b2 = new JButton
("2");
        JButton b3 = new JButton
("3");
        JButton b4 = new JButton
("4");
        JButton b5 = new JButton
("5");

        // Add buttons to the fram
e
```



```
        add(b1);
        add(b2);
        add(b3);
        add(b4);
        add(b5);

        // Set frame properties
        setSize(500, 500);
        setLayout(new FlowLayout(F
lowLayout.RIGHT));
        setVisible(true);
    }

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

⇒ Grid Layout

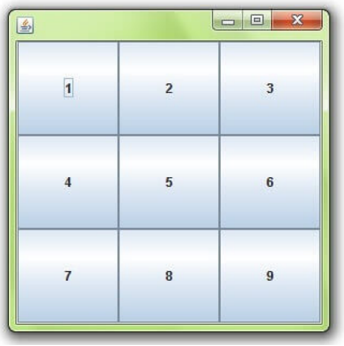
- To arrange components in rectangular grid.
- One component is displayed in each rectangle.

→ Constructor are:

- `GridLayout();` : layout with one column per component in row
- `GridLayout(int rows,int columns);` layout with given rows & columns but no gaps
- `GridLayout(int rows, int columns, int hgap, int vgap);` layout with row & colmuns with given gaps

→ Code for Grid layout

```
class Example extends JFrame{
    Example(){
        JButton b1 = new JButton("1");
        JButton b2 = new JButton("2");
        JButton b3 = new JButton("3");
        JButton b4 = new JButton("4");
        JButton b5 = new JButton("5");
        JButton b6 = new JButton("6");
        JButton b7 = new JButton("7");
        JButton b8 = new JButton("8");
        JButton b9 = new JButton("9");
        add(b1);add(b2);add(b3);add(b4);ad
d(b5);
        add(b6);add(b7);add(b8);add(b9);
        setLayout(new GridLayout(3,3))
        setSize(300,300);
        setVisible(true);
    }

    public static void main(String [] arg
s){
        new Example();
    }
}
```



⇒ GridBagLayout

- Most flexible layout manager class.
- It manages component in grid of rows and columns.
- Allow components to occupy multiple cells accross row or columns

→ Code for Grid Bag Layout

```
import javax.swing.*;
import java.awt.*;

public class GridBagLayoutExample extends JFrame {
```

```
    public GridBagLayoutExample() {
        // Set the layout manager
        setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();

        // Create components
        JButton button1 = new JButton("Button 1");
        JButton button2 = new JButton("Button 2");
        JButton button3 = new JButton("Button 3");
        JButton button4 = new JButton("Button 4");
        JButton button5 = new JButton("Button 5");
        JLabel label = new JLabel("Label");
        JTextField textField = new JTextField("Text Field");

        // Add components with GridBagConstraints

        // Button 1
        gbc.gridx = 0; // Column 0
        gbc.gridy = 0; // Row 0
        gbc.gridwidth = 1; // Span 1 column
        gbc.gridheight = 1; // Span 1 row
        gbc.weightx = 0.5; // Weight for resizing horizontally
        gbc.weighty = 0.5; // Weight for resizing vertically
        gbc.fill = GridBagConstraints.BOTH; // Fill both horizonta
lly and vertically
        add(button1, gbc);

        // Button 2
        gbc.gridx = 1; // Column 1
        gbc.gridy = 0; // Row 0
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        add(button2, gbc);

        // Button 3
        gbc.gridx = 0;
        gbc.gridy = 1; // Row 1
        gbc.gridwidth = 2; // Span 2 columns
        add(button3, gbc);

        // Button 4
        gbc.gridx = 0;
```

```
        gbc.gridy = 2; // Row 2
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        add(button4, gbc);

        // Button 5
        gbc.gridx = 1;
        gbc.gridy = 2; // Row 2
        add(button5, gbc);

        // Label
        gbc.gridx = 0;
        gbc.gridy = 3; // Row 3
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        add(label, gbc);

        // TextField
        gbc.gridx = 1;
        gbc.gridy = 3; // Row 3
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.fill = GridBagConstraints.HORIZONTAL; // Fill horizont
ally
        add(textField, gbc);

        // Set frame properties
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Ensure
the application exits on close
        setVisible(true);
    }

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

→***Explanation:***

1. **Set the Layout Manager**:
    - `setLayout(new GridBagLayout())`:
      Sets the layout manager of the

`JFrame` to `GridBagLayout`.

2. **GridBagConstraints**:
   - `GridBagConstraints gbc = new GridBagConstraints();`: Creates an instance of `GridBagConstraints`, which is used to specify the constraints for each component.
   - For each component, you specify constraints such as (7)
     - `gridwidth`
     - `gridheight`
     - `gridx`
     - `gridy`
     - `weightx`
     - `weighty`
     - `fill`
3. **Component Addition**:
   - For each component, set the constraints and add the component to the `JFrame`:

```
gbc.gridx = 0; // Column index
 gbc.gridy = 0; // Row index
 gbc.gridwidth = 1; // Number of columns to span
 gbc.gridheight = 1; // Number of rows to span
 gbc.weightx = 0.5; // Horizontal resizing weight
 gbc.weighty = 0.5; // Vertical resizing weight
 gbc.fill = GridBagConstraints.BOTH; // Resize both horizont
 ally and vertically
 add(button1, gbc);
```

This example places five buttons, a label, and a text field in a grid layout with specific constraints.

⇒ Group Layout

- It groups components and places them in container hierarchical.
- Group is abstract class and two concrete classes implemets this group class.
- **SequentialGroup**

    - set position of child sequentially one after another
- **ParallelGroup**
    - set position of child on top of each other

→ Methods

- `createParallelGroup()`
- `createSequentialGroup()`

> ***Note:***
>
> Group layout treats each axis independently.
>
> There is horizontal axis and vertical axis.
> Each component mut exit in horizontal and vertical group else IllegalStateException is thorwn.

→ Code

GroupLayout is a layout manager that allows for the explicit positioning of components, making it easier to design complex layouts. Here is a simple example demonstrating how to use `GroupLayout` in a `JFrame`.

```
import javax.swing.*;
import java.awt.*;

public class GroupLayoutExample extends JFrame {

    public GroupLayoutExample() {
        // Create components
        JLabel label1 = new JLabel("Label 1:");
        JTextField textField1 = new JTextField(10);
        JLabel label2 = new JLabel("Label 2:");
        JTextField textField2 = new JTextField(10);
        JButton button = new JButton("Submit");

        // Create GroupLayout and set it to the frame
        GroupLayout layout = new GroupLayout(getContentPane());
```

```
        getContentPane().setLayout(layout);

        // Automatically create gaps between components
        layout.setAutoCreateGaps(true);
        layout.setAutoCreateContainerGaps(true);

        // Set the horizontal group
        layout.setHorizontalGroup(
            layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(GroupLayout.A
lignment.LEADING)
                    .addComponent(label1)
                    .addComponent(label2))
                .addGroup(layout.createParallelGroup(GroupLayout.A
lignment.LEADING)
                    .addComponent(textField1)
                    .addComponent(textField2)
                    .addComponent(button))
        );

        // Set the vertical group
        layout.setVerticalGroup(
            layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(GroupLayout.A
lignment.BASELINE)
                    .addComponent(label1)
                    .addComponent(textField1))
                .addGroup(layout.createParallelGroup(GroupLayout.A
lignment.BASELINE)
                    .addComponent(label2)
                    .addComponent(textField2))
                .addComponent(button)
        );

        // Set frame properties
        pack();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null); // Center the frame
        setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(GroupLayoutExample::new);
```

```
    }
}
```

Explanation

1. **Create Components**:
   - Components such as `JLabel`, `JTextField`, and `JButton` are created.
2. **Create and Set GroupLayout**:
   - A `GroupLayout` instance is created and set as the layout manager for the frame's content pane:

```
GroupLayout layout = new GroupLayout(getContentPane());
getContentPane().setLayout(layout);
```

3. **Set Auto Gaps**:
   - Gaps between components and container edges are automatically created:

```
layout.setAutoCreateGaps(true);
layout.setAutoCreateContainerGaps(true);
```

4. **Define the Horizontal Group**:
   - The horizontal arrangement of components is defined using a sequential group that contains parallel groups:

```
layout.setHorizontalGroup(
    layout.createSequentialGroup()
        .addGroup(layout.createParallelGroup(GroupLayout.Al
ignment.LEADING)
            .addComponent(label1)
            .addComponent(label2))
        .addGroup(layout.createParallelGroup(GroupLayout.Al
ignment.LEADING)
            .addComponent(textField1)
            .addComponent(textField2)
            .addComponent(button))
);
```

5. **Define the Vertical Group**:
   - The vertical arrangement of components is defined using a sequential group that contains parallel groups:

```
layout.setVerticalGroup(
    layout.createSequentialGroup()
        .addGroup(layout.createParallelGroup(GroupLayout.Al
ignment.BASELINE)
            .addComponent(label1)
            .addComponent(textField1))
        .addGroup(layout.createParallelGroup(GroupLayout.Al
ignment.BASELINE)
            .addComponent(label2)
            .addComponent(textField2))
        .addComponent(button)
);
```

6. **Set Frame Properties**:
    - `pack()`: Sizes the frame so that all its contents are at or above their preferred sizes.
    - `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`: Ensures that the application exits when the frame is closed.
    - `setLocationRelativeTo(null)`: Centers the frame on the screen.
    - `setVisible(true)`: Makes the frame visible.

This example creates a simple form with two labels, two text fields, and a submit button using `GroupLayout`. The components are aligned and spaced correctly according to the defined layout groups.

2D shape

- **Graphics class that allow us to call basic 2D shapes that can be drawn to a window.**
- **Graphics is an abstract class provided by Java AWT which is used to draw or paint on components.**
- JFrame class that allows for us to create a regular window to draw shapes to screen
- **Canvas class controls and represents blank rectangular are where application can draw or trap input events from user.**

⇒Graphics context

- Enalbes drawing on screen

- Graphics object manages graphics context.

⇒ Method

```
public void paint (Graphics g)
```

- Class componet’s method that takes Graphics object

⇒ Color Class

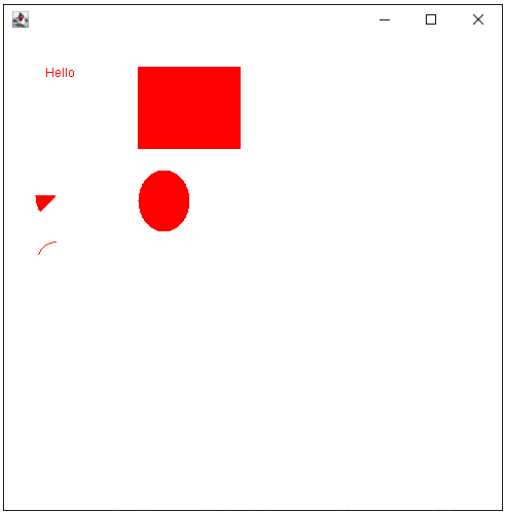
- To set color and use color

```
Color.RED
```

→ Code for 2d shapes

- Fill will cover whole
- Draw will just do line

```
package com.example.demo;

import javax.swing.*;
import java.awt.*;

class Example extends Canvas {

    Example() {
        // Create a JFrame instanc
e and set properties
        JFrame f = new JFrame();
        f.setSize(500, 500);
        f.setLayout(null);

        // Add the Canvas (Exampl
e) to the JFrame
        this.setBounds(0, 0, 500,
500);
        f.add(this);
```



- **Canvas**
  - setBackground
  - setForeground
- **paint (Graphics g)**

```
        f.setVisible(true);

        // Set background and fore
ground colors
        setBackground(Color.WHIT
E);
        setForeground(Color.RED);
    }

    @Override
    public void paint(Graphics g)
{
        // Painting logic
        g.drawString("Hello", 40,
40);
        g.fillRect(130, 30, 100, 8
0);
        g.fillOval(130, 130, 50, 6
0);
        g.drawArc(30, 200, 40, 50,
90, 60);
        g.fillArc(30, 130, 40, 50,
180, 40);
    }

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

- drawString
- drawArc
- fillRect
- fillOval
- fillArc

→ code for Font adding

```
class Example extends canvas{
    Example(){
        JFrame f = new JFrame();
        this.setBounds(0,0,500,500);
        f.add(this);
        f.setSize(500,500);
        f.setLayout(null);
        f.setVisible(true);
```

```
    }

    public void paint(Graphics g){
        Font font = new Font(Font.SANS_SERIF,Font.BOLD,20);
        g.setFont(font);
        g.setColor(Color.GRAY);
        g.drawString("Hello",150,150);
    }


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

Displaying Image

- We can use method drawImage() of Graphics class. Its like (i.e drawOval, drawRect…etc)
- ImageIo class provie read() to load image from file, URL or InputStream
- Once we have Image object then we can draw into graphics context with drawImage() method of Graphics class.

```
public abstract boolean drawImage(Image img, int x, int y, ImageOb
server observer)
```

⇒ URL image load

```
URL url = getClass().getResource("/folder/something.png");
Image image = ImageIO.read(url);
```

⇒ File image load

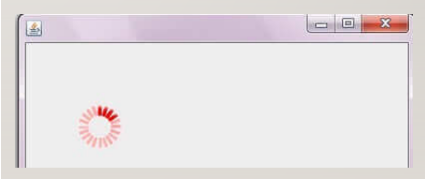
```
File file = new File("E://folder/something.png");
Image image = ImageIO.read(file);
```

⇒ Preloading images

```
ImageIcon icon = new ImageIcon("something.png");
Image image = icon.getImage();
```

→ **Code**

```
class Example extends Canvas{

    Example(){
        JFrame f = new JFrame();
        //add canvas to frame
        this.setBounds(0,0,500,500);
        f.add(this);

        f.setSize(500,500);
        f.setLayout(null);
        f.setVisible(true);

    }


    public void paint(Graphics g){
        ImageIcon ic= new ImageIcon("i
mage.png");
        Image image = ic.getImage();
        g.drawImage(image,120,120,thi
s); //Image , x , y , ImageObserver

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



MVC Architecture

- Model-View-Controller a design pattern in web development field.
- It specifies that a application shall consist of data model, presentaion and control information.

⇒ Model

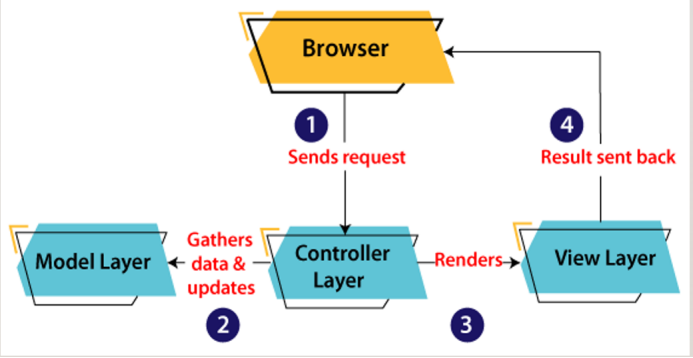
- Represent data and business logic of application.
- Manages data
- Response to request for information from view
- Response to information for change state of data from controller

⇒ View

- Represent presentation layer of application
- Display data from model to user
- Send user input to controller

⇒ Controller

- Bridge for model and view
- Manages flow of application
- Receive input from view
- Process input and update model and view accordingly.



Steps:

1. Client (browser) send request to controller for page
2. Controller then calls model. Receive that requested data
3. Controller transfer data to view layer
4. From view to browser

→ Advantage / Benefits

1. **Separation of Concerns**:
   By dividing the application into three interconnected components, each part of the application can be developed, tested, and maintained independently.
2. **Reusability**:
   Components can be reused across different parts of the application or even across different projects.
3. **Manageability**:
   Easier to manage and update the application since the business logic, user interface, and control are separated.
4. **Scalability**:
   Makes it easier to scale the application as it grows, as each component can be modified or replaced independently.

→ Disadvantage

1. **Increase complexity**
2. **Increase number of files**
3. **Tight coupling:** controller and view
4. **Testing complexity**
5. **Difficult to coordinate multiple view**
