PEMROGRAMAN BERORIENTASI OBJEK (P)
PERTEMUAN IX

By: Aisyah Fitri Yuniasih, M. Si

THE MATERIAL – GUI

- Layout Manager
- Event Handling
- Menus

LAYOUT MANAGER (1)

- A layout manager is an object that determines the way components are arranged in a container

Layout Manager	Defined In
FlowLayout	AWT
BorderLayout	AWT
GridLayout	AWT
GridBagLayout	AWT
CardLayout	AWT
BoxLayout	Swing

LAYOUT MANAGER (2)

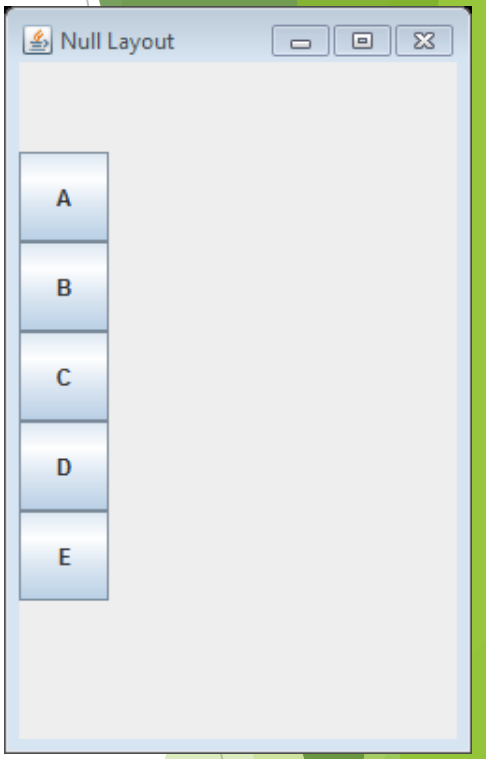
- Every container has a default layout manager
- The setLayout method is used to explicitly set the layout manager
- Each layout manager has its own particular rules governing how components are arranged

NULL LAYOUT MANAGER

- Rather than using a layout manager that controls the size and position of all components in a container, you can set the layout manager to null.
- Each component then controls its own position and size using its bounds → use setBounds(x, y, width, height) method

EXERCISE 1

- Buatlah sebuah frame dengan ukuran width 250 dan height 400 dengan judul Null Layout
- Atur sedemikian rupa sehingga Frame tersebut muncul di layar, ketika windows Frame tersebut ditutup program aplikasi juga ikut tertutup, dan component-component yang nantinya akan diletakkan di Frame tersebut posisinya di tengah Frame
- Buatlah sebuah Panel yang akan menampung component-component yang ada
- Buatlah 5 buah button dengan button text A, B, C, D, dan E
- Set null layout manager dan setBounds kelima button tersebut dengan syntax sbb:




```
39          panel1.setLayout(null);
40          btnA.setBounds(0,48,48,48);
41          btnB.setBounds(0,96,48,48);
42          btnC.setBounds(0,144,48,48);
43          btnD.setBounds(0,192,48,48);
44          btnE.setBounds(0,240,48,48);
```

FLOW LAYOUT

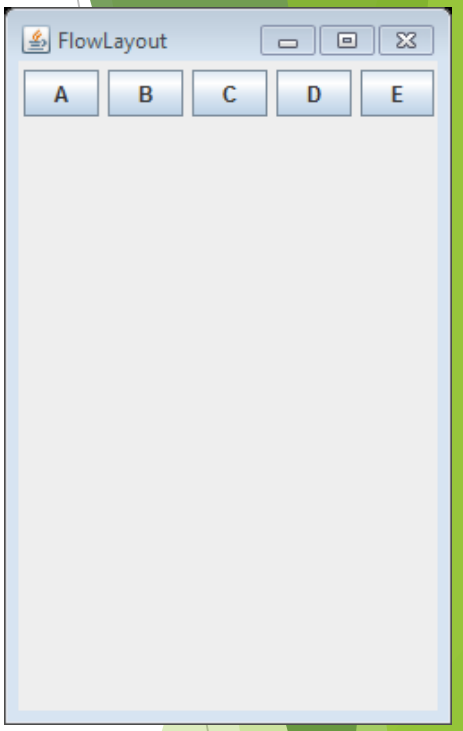
- Flow layout simply puts as many components as possible left to right on a row
- Rows are created as needed to accommodate all of the components, starting a new row if its container is not sufficiently wide
- Components are displayed in the order they are added
- Horizontal alignment and horizontal and vertical gaps can be explicitly set
- Flow layout is the default for a pane e.g. JPanel

FLOW LAYOUT

Constructor	Purpose
FlowLayout ()	Constructs a new FlowLayout object with a centered alignment and horizontal and vertical gaps with the default size of 5 pixels.
FlowLayout (int align)	Creates a new flow layout manager with the indicated alignment and horizontal and vertical gaps with the default size of 5 pixels. The alignment argument can be FlowLayout.LEFT, FlowLayout.LEADING, FlowLayout.CENTER, FlowLayout.RIGHT or FlowLayout.TRAILING.
FlowLayout (int align, int hgap, int vgap)	Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps. The hgap and vgap arguments specify the number of pixels to put between components.

EXERCISE 2

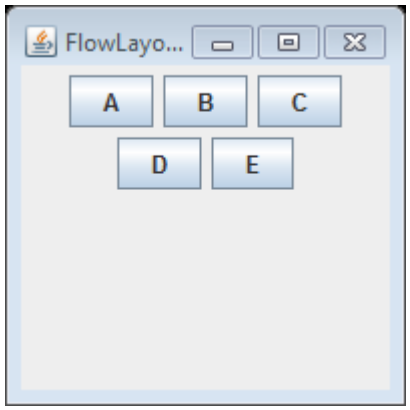
- Buatlah sebuah frame dengan ukuran width 250 dan height 400 dengan judul FLowLayout
- Atur sedemikian rupa sehingga Frame tersebut muncul di layar, ketika windows Frame tersebut ditutup program aplikasi juga ikut tertutup, dan component-component yang nantinya akan diletakkan di Frame tersebut posisinya di tengah Frame
- Buatlah sebuah Panel yang akan menampung component-component yang ada
- Buatlah 5 buah button dengan button text A, B, C, D, dan E
- Set FlowLayout layout manager dengan syntax sbb:



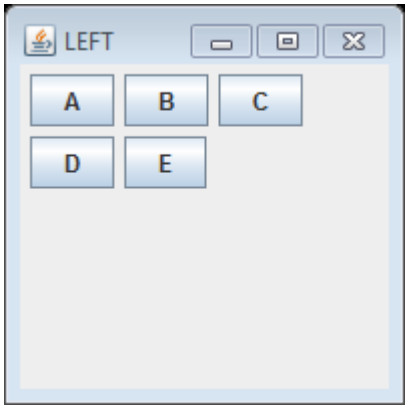

```
38          panel1.setLayout(new FlowLayout());
```

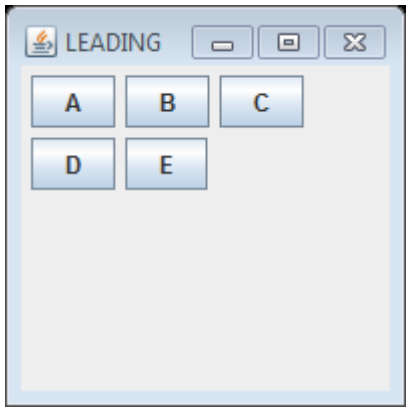
EXERCISE 2

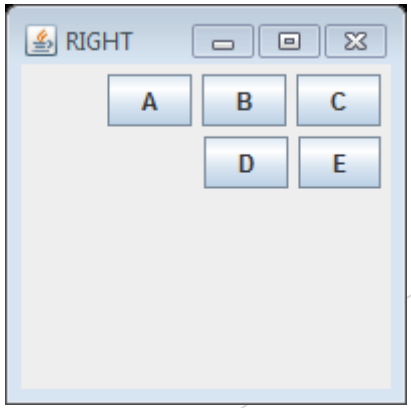
- Ganti ukuran frame dengan width 200 dan height 400

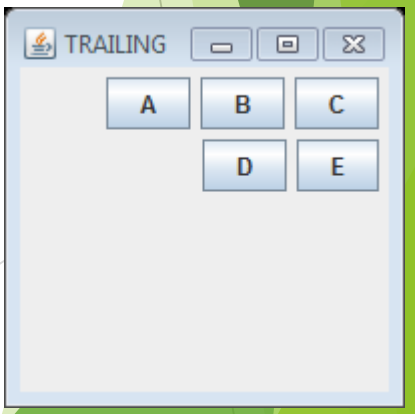


- Coba ubah-ubah alignmentnya menjadi FlowLayout.LEFT, FlowLayout.LEADING, FlowLayout.RIGHT dan FlowLayout.TRAILING.



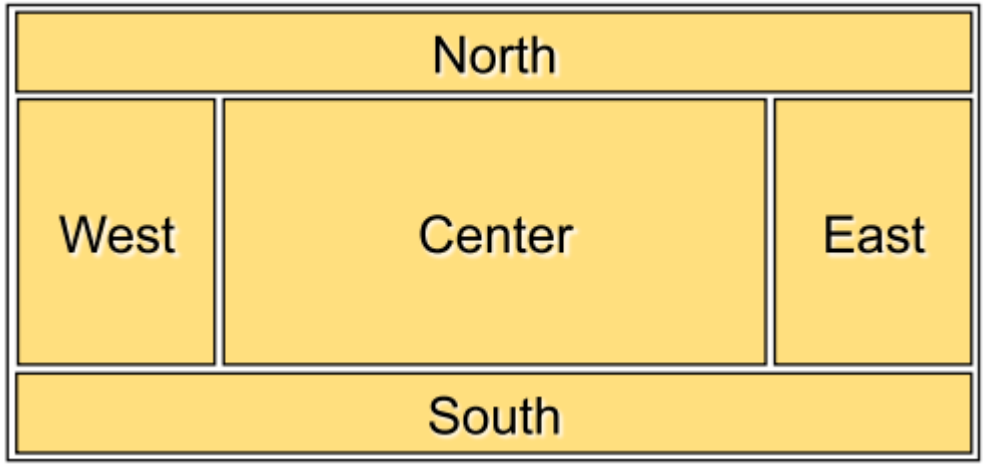






BORDER LAYOUT

- A border layout defines five areas:



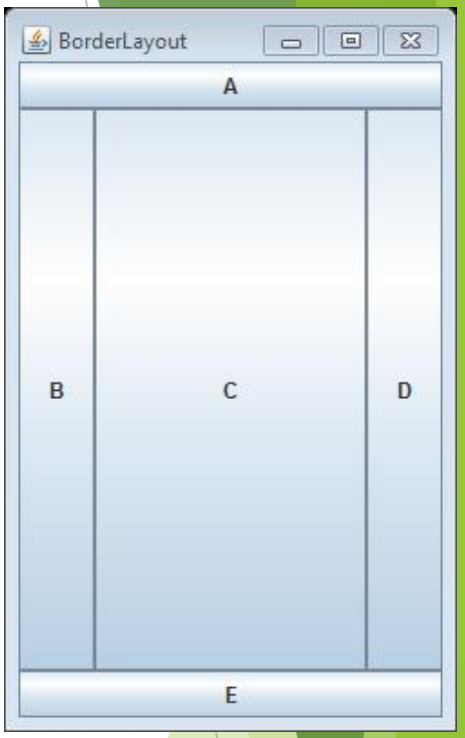
- A single component can be added to each area
- The areas expand or contract as needed to accommodate components or fill space
- All extra space is placed in the center area

BORDER LAYOUT

Constructor	Purpose
BorderLayout (int hgap, int vgap)	Defines a border layout with specified gaps between components
setHgap (int)	Sets the horizontal gap between components.
setVgap (int	Sets the vertical gap between components.

EXERCISE 3

- Buatlah sebuah frame dengan ukuran width 250 dan height 400 dengan judul BorderLayout
- Atur sedemikian rupa sehingga Frame tersebut muncul di layar, ketika windows Frame tersebut ditutup program aplikasi juga ikut tertutup, dan component-component yang nantinya akan diletakkan di Frame tersebut posisinya di tengah Frame
- Buatlah sebuah Panel yang akan menampung component-component yang ada
- Buatlah 5 buah button dengan button text A, B, C, D, dan E
- Set BorderLayout layout manager dan set btnA di North, btnB di West, btnC di Center, btnD di East, dan btnE di South dengan syntax sbb:




```
37          panel1.setLayout(new BorderLayout());
38          panel1.add(btnA, BorderLayout.NORTH);
39          panel1.add(btnB, BorderLayout.WEST);
40          panel1.add(btnC, BorderLayout.CENTER);
41          panel1.add(btnD, BorderLayout.EAST);
42          panel1.add(btnE, BorderLayout.SOUTH);
```

GRID LAYOUT

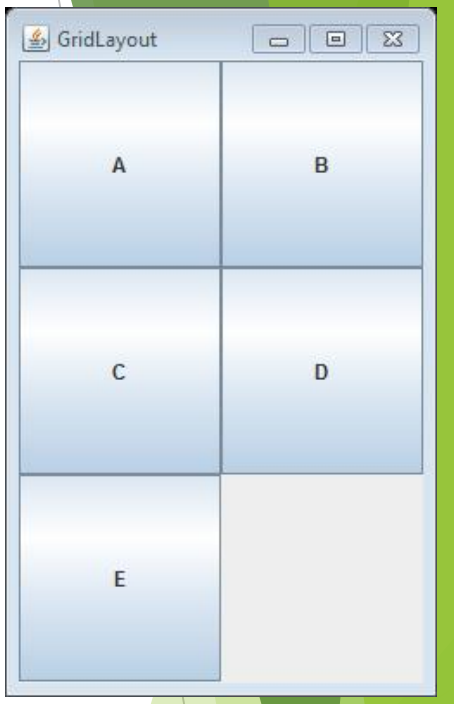
- A grid layout displays components in a rectangular grid of rows and columns (table-like grid) with equally sized cells
- One component per cell
- All cells have the same size
- As components are added, they fill the grid from left-to-right and top-to-bottom (by default)
- The size of each cell is determined by the overall size of the container

GRID LAYOUT

Constructor	Purpose
GridLayout (int rows, int cols)	Creates a grid layout with the specified number of rows and columns. All components in the layout are given equal size. One, but not both, of rows andcols can be zero, which means that any number of objects can be placed in a row or in a column.
GridLayout (int rows, int cols;int hgap, int vgap)	Creates a grid layout with the specified number of rows and columns. In addition, the horizontal and vertical gaps are set to the specified values. Horizontal gaps are places between each of columns. Vertical gaps are placed between each of the rows.

EXERCISE 4

- Buatlah sebuah frame dengan ukuran width 250 dan height 400 dengan judul GridLayout
- Atur sedemikian rupa sehingga Frame tersebut muncul di layar, ketika windows Frame tersebut ditutup program aplikasi juga ikut tertutup, dan component-component yang nantinya akan diletakkan di Frame tersebut posisinya di tengah Frame
- Buatlah sebuah Panel yang akan menampung component-component yang ada
- Buatlah 5 buah button dengan button text A, B, C, D, dan E
- Set GridLayout layout manager dengan syntax sbb:



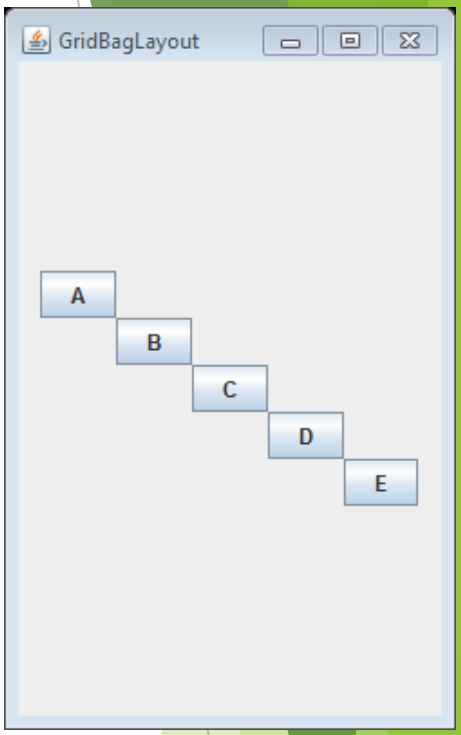

```
37          panel1.setLayout(new GridLayout(3,2));
```

GRIDBAG LAYOUT

- GridBagLayout is a sophisticated, flexible layout manager.
- It aligns components by placing them within a grid of cells, allowing components to span more than one cell.
- The rows in the grid can have different heights, and grid columns can have different widths.

EXERCISE 5

- Buatlah sebuah frame dengan ukuran width 250 dan height 400 dengan judul GridBagLayout
- Atur sedemikian rupa sehingga Frame tersebut muncul di layar, ketika windows Frame tersebut ditutup program aplikasi juga ikut tertutup, dan component-component yang nantinya akan diletakkan di Frame tersebut posisinya di tengah Frame
- Buatlah sebuah Panel yang akan menampung component-component yang ada
- Buatlah 5 buah button dengan button text A, B, C, D, dan E



EXERCISE 5

- Set GridbagLayout layout manager dengan syntax sbb:

```
31          panel1.setLayout(new GridBagLayout());
32
33          GridBagConstraints constraint1 = new GridBagConstraints();
34          GridBagConstraints constraint2 = new GridBagConstraints();
            GridBagConstraints constraint3 = new GridBagConstraints();
36          GridBagConstraints constraint4 = new GridBagConstraints();
37          GridBagConstraints constraint5 = new GridBagConstraints();
38
39          constraint1.gridx =0;
40          constraint1.gridy =0;
41          constraint2.gridx =1;
42          constraint2.gridy =1;
43          constraint3.gridx =2;
44          constraint3.gridy =2;
45          constraint4.gridx =3;
46          constraint4.gridy =3;
47          constraint5.gridx =4;
48          constraint5.gridy =4;
49
50          panel1.add(btnA, constraint1);
51          panel1.add(btnB, constraint2);
52          panel1.add(btnC, constraint3);
53          panel1.add(btnD, constraint4);
54          panel1.add(btnE, constraint5);
55
```

CARD LAYOUT

- The CardLayout class lets you implement an area that contains different components at diferent times.
- A CardLayout is often controlled by a combo box, with the state of the combo box determining which panel (group of components) the CardLayout displays. An alternative to using CardLayout is using a tabbed pane, which provides similar functionality but with a pre-defined GUI.

Basic GUI Programming Steps in Java

- declare a container and components
- add components to one or more containers using a layout manager
- register event listener(s) with the components which is source of the event
- create event listener method(s)

EVENT HANDLING

- GUI programming uses an event-driven model of programming.
- The program responds to events caused by the user interacting with a GUI component.
- For example, we might display some text fields, a few buttons, and a selectable list of items. Then our program will "sit back" and wait for the user to do something.
- When the user enters text into a text field, presses a button, or selects an item from the list, our program will respond, performing the function that the user has requested, then sit back again and wait for the user to do something else.

LISTENER INTERFACES

- A typical event handler class implements a listener interface.
- The listener interfaces, which inherit from the EventListener (XXXListener) interface, are supplied in the java.awt.event or javax.swing.event package.
- A listener interface specifies one or more abstract methods that an event handler class needs to override.
- The listener methods receive as a parameter an event object, which represents the event that was fired

EVENT OBJECTS

- Event classes are subclasses of the EventObject class and are in the java.awt.event and javax.swing.event packages.
- From the EventObject class, event classes inherit the getSource method

Return Value	Method Name and Argument List
Object	getSource () Returns the object reference of the component that fired the event

- With simple if .. else if statements, an event handler can identify which of its registered components fired the event.

EVENTS & LISTENERS (1)

JComponent	User Activity	Event Object	Listener Interface to implement
JTextField	Pressing Enter	ActionEvent	ActionListener
JTextArea	Pressing Enter	ActionEvent	ActionListener
JButton	Pressing the Button	ActionEvent	ActionListener
JRadioButton	Selecting a radio button	ItemEvent	ItemListener
JCheckBox	Selecting a checkbox	ItemEvent	ItemListener

EVENTS & LISTENERS (2)

JComponent	User Activity	Event Object	Listener Interface to implement
JList	Selecting an item	ListSelection Event	ListSelectionListener
JComboBox	Selecting an item	ItemEvent	ItemListener
Any component	Pressing or releasing mouse buttons	MouseEvent	MouseListener
Any component	Moving or dragging the mouse	MouseEvent	MouseMotionListener

REGISTERING A LISTENER

- In the constructor, we instantiate an object of our event handler class. Then we register that event handler on a component by calling an addXXXListener() method.

add XXXListener APIs
void addActionListener (ActionListener handler)
void addItemListener (ItemListener handler)
void addListSelectionListener (ListSelectionListener handler)
void addMouseListener (MouseListener handler)
void addMouseMotionListener (MouseMotionListener handler)

The ActionListener Interface

- An event handler that implements the ActionListener interface provides code in this method to respond to the ActionEvent fired by any registered components.

```
10  import java.awt.event.*;
11  /**
12   *
13   * @author Aisyah
14   */
15  public class ContohActionListener implements ActionListener{
16      @Override
17      public void actionPerformed(ActionEvent e){
18
19      }
20  }
21
```

Mouse Events

- For mouse events, there are two listeners:
  - the MouseListener interface specifies five methods to implement which are mousePressed, mouseReleased, mouseEntered, mouseExited, and mouseClicked
  - the MouseMotionListener interface specifies two methods to implement which are mouseDragged and mouseMoved
- If we want to use a MouseListener, but need to use only one of its five methods to process a MouseEvent , we still have to implement the other four methods as "do-nothing" methods with empty method bodies

Removing a Listener

- To remove a listener we use removeXXXListener method
- For example:

Return Value	Method Name and Argument List
void	removeMouseListener(MouseListener mh) removes the mouse listener mh so that it is no longer registered on this component.

Event Listener Interfaces in java.awt.event (1)

Event	Related to
ActionListener	Action events
AdjustmentListener	Adjustment events
AWTEventListener	Observe passively all events dispatched within AWT
ComponentListener	Component (move, size, hide, show) events
ContainerListener	Container (ad, remove component) events
FocusListener	Focus (gain, loss) events
HierarchyBoundsListener	Hierarchy (ancestor moved/resized) events
HierarchyListener	Hierarchy (visibility) events
InputMethodListener	Input method events (multilingual framework)
ItemListener	Item events
KeyListener	Keyboard events

Event Listener Interfaces in java.awt.event (2)

Event	Related to
MouseListener	Mouse buttons events
MouseMotionListener	Mouse motion events
MouseWheelListener	Mouse wheel events
TextListener	Text events
WindowFocusListener	Window focus events (new focus management framework)
WindowListener	Window events (non focus related)
WindowStateListener	Window state events

Event Listener Interfaces in java.swing.event (1)

Event	Related to
AncestorListener	Changes to location and visible state of a JComponent or its parents
CaretListener	Text cursor movement events
CellEditorListener	Cell editor events
ChangeListener	Change events
DocumentListener	Text document events
HyperlinkListener	Hyperlink events
InternalFrameListener	Internal frame events
ListDataListener	List data events
ListSelectionListener	List selection events
MenuDragMouseListener	Menu mouse movement events
MenuKeyListener	Menu keyboard events
MenuListener	Menu selection events

Event Listener Interfaces in java.swing.event (2)

Event	Related to
MouseInputListener	Aggregrated mouse and mouse motion events
PopupMenuListener	Popup menu events
TableColumnModelListener	Table column events
TableModelListener	Table model data events
TreeExpansionListener	Tree expand/collapse events
TreeModelListener	Tree model data events
TreeSelectionListener	Tree selection events
TreeWillExpandListener	Tree expand/collapse pending events
UndoableEditListener	Undo/Redo events

How to write Event Listener

E.g to write an Action Listener

- Way 1 Declare an event handler class (ussualy a Jframe class) and specify that the class either implements an ActionListener interface or extends a class that implements an ActionListener interface.
- Way 2 Use an inner class
- Way 3 Use an anonymous inner class

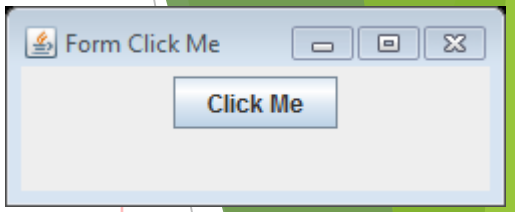
Way 1

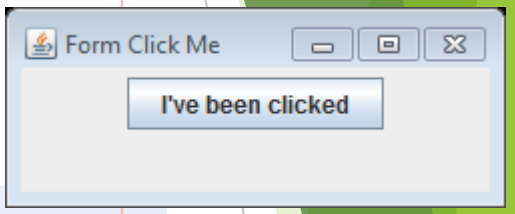
- 1. Declare an event handler class (ussualy a Jframe class) and specify that the class either implements an ActionListener interface or extends a class that implements an ActionListener interface.
- 2. Create component(s) which generate event
- 3. Register an the event handler class as a listener on one or more components.
- 4. Include code that implements the methods in listener interface.

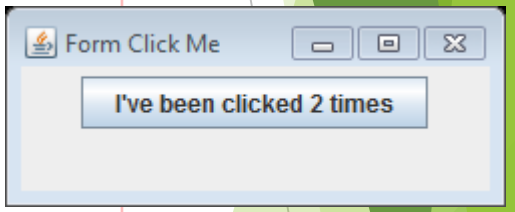
Way 1

```
16  public class FormClickMe extends JFrame implements ActionListener {
    private JButton btnClickMe;
18  private int clickCount = 0;
19  public FormClickMe(){
20      JPanel panel1 = new JPanel();
21      btnClickMe = new JButton("Click Me");
        btnClickMe.addActionListener(this);
23      panel1.add(btnClickMe);
24      this.add(panel1);
25      this.setSize(250,100);
26      this.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
27      this.setTitle("Form Click Me");
28      this.setVisible(true);
29      this.setLocationRelativeTo(null);
30  }
31  @Override
    public void actionPerformed(ActionEvent e) {
33      if(e.getSource() == btnClickMe){
34          clickCount++;
35          if(clickCount ==1)
36              btnClickMe.setText("I've been clicked");
37          else
38              btnClickMe.setText("I've been clicked " + clickCount + " times");
39      }
40  }
41  }
```

1 2 3 4







Way 1

- Actions for some buttons

```
if(e.getSource() == btn1){
    // method untuk btn1
} else if (e.getSource() == btn2){
    // method untuk btn2
} else if (e.getSource() == btn3){
    // method untuk btn3
```

- To make a max screen size

```
31          Dimension size=Toolkit.getDefaultToolkit().getScreenSize();
32          this.setSize(size);
```

Way 2

- 1. Create component(s) which generate event
- 2. Make an inner class which implements Listener
- 3. Instance a Listener from an inner class
- 4. Register an the event handler class as a listener on one or more components.

Way 2

```
17  public class FormClickMe2 extends JFrame{
        private JButton btnClickMe;
19      private int clickCount = 0;
20      public FormClickMe2(){
21          JPanel panel1 = new JPanel();
22          btnClickMe = new JButton("Click Me");
23          ClickListener cl = new ClickListener();
24          btnClickMe.addActionListener(cl);
25          panel1.add(btnClickMe);
26          this.add(panel1);
27          this.setSize(250,100);
28          this.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
29          this.setTitle("Form Click Me");
30          this.setVisible(true);
31          this.setLocationRelativeTo(null);
32      }
33      private class ClickListener implements ActionListener{
            public void actionPerformed(ActionEvent e){
35              if(e.getSource() == btnClickMe){
36              clickCount++;
37              if(clickCount ==1)
38                  btnClickMe.setText("I've been clicked");
39              else
40                  btnClickMe.setText("I've been clicked " + clickCount + " times");
41              }
42          }
43      }
44  }
```

1 2 3 4

Way 3

- You can create an inner class without specifying a name this is known as an *anonymous inner class*.
- Every time we make a component we directly make an anonymous inner class to handle event from that component
- While it might look strange at first glance, anonymous inner classes can make your code easier to read because the class is defined where it is referenced.
- However, you need to weigh the convenience against possible performance implications of increasing the number of classes → one component one class

Way 3

- 1. Create component(s) which generate event
- 2. Register a new listener on one components.
- 3. Make an anonymous inner class to handle event from that component

Way 3

```
21  public class FormClickMe3 extends JFrame{
22      private JButton btnClickMe;
23      private int clickCount = 0;
24      public FormClickMe3(){
25          JPanel panel1 = new JPanel();
26          btnClickMe = new JButton("Click Me");
27          btnClickMe.addActionListener(new ActionListener() {
28              @Override
29              public void actionPerformed(ActionEvent e) {
30                  if(e.getSource() == btnClickMe){
31                  clickCount++;
32                  if(clickCount ==1)
33                      btnClickMe.setText("I've been clicked");
34                  else
35                      btnClickMe.setText("I've been clicked " + clickCount + " times");
36                  }
37              }
38          });
39          panel1.add(btnClickMe);
40          this.add(panel1);
41          this.setSize(250,100);
42          this.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
43          this.setTitle("Form Click Me");
44          this.setVisible(true);
45          this.setLocationRelativeTo(null);
46      }
47  }
```

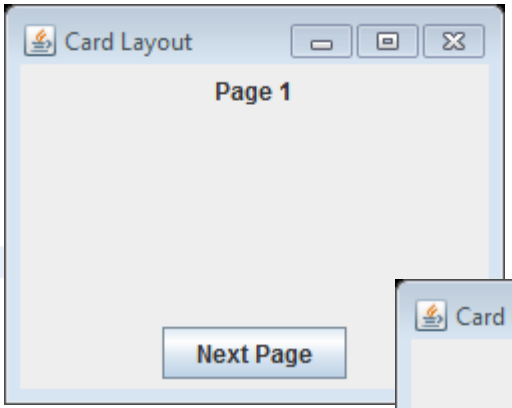
1 2 3

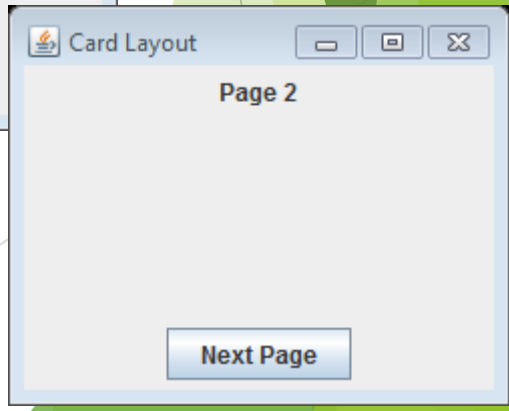
Way 3

```java
public class PanelCardLayout1 extends JPanel{
    public PanelCardLayout1(){
        JLabel lbl = new JLabel("Page 1");
        lbl.setSize ( 100,20);
        this.add(lbl);
    }
}
```

```java
public class PanelCardLayout2 extends JPanel{
    public PanelCardLayout2(){
        JLabel lbl = new JLabel("Page 2");
        lbl.setSize ( 100,20);
        this.add(lbl);
    }
}
```

```java
public class FormCardLayout extends JFrame {
    CardLayout cards = new CardLayout();
    JPanel panelUtama, panelButton;
    public FormCardLayout(){
        this.setSize(250,200);
        this.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        this.setTitle("Card Layout");
        this.setVisible(true);
        this.setLocationRelativeTo(null);
        panelUtama = new JPanel();
        panelButton = new JPanel();
        PanelCardLayout1 panel1 = new PanelCardLayout1();
        PanelCardLayout2 panel2 = new PanelCardLayout2();
        panelUtama.setLayout(cards);
        panelUtama.add(panel1, "First");
        panelUtama.add(panel2, "Second");
        JButton btnNext = new JButton("Next Page");
        panelButton.add(btnNext);
        this.setLayout(new BorderLayout ());
        this.add(panelUtama, BorderLayout.NORTH);
        this.add(panelButton, BorderLayout.SOUTH);
        btnNext.addActionListener(new ActionListener(){
            @Override
            public void actionPerformed(ActionEvent e){
                cards.next(panelUtama);
            }
        });
    }
}
```





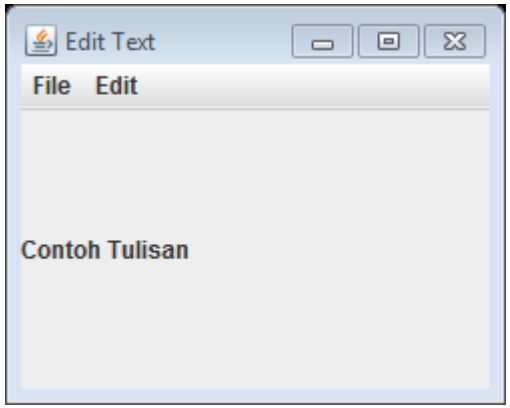
Menu

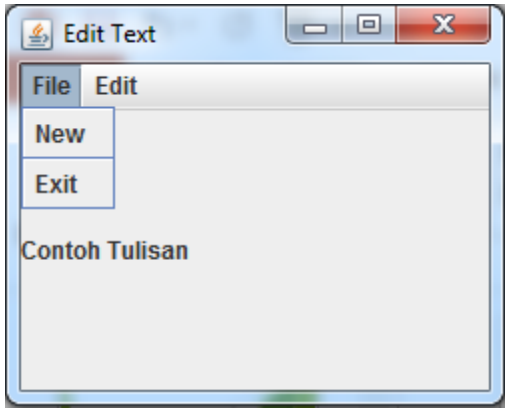
- A menu provides a space-saving way to let the user choose one of several options.
- Other components with which the user can make a one-of-many choice include combo boxes, lists, radio buttons, spinners, and tool bar

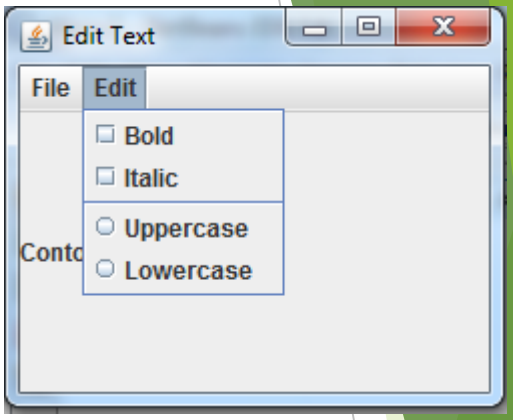
Menu

- Menus are unique in that, by convention, they aren't placed with the other components in the UI.
- Instead, a menu usually appears either in a *menu bar* or as a *popup menu*.
- A menu bar contains one or more menus and has a customary, platform-dependent location — usually along the top of a window.
- A popup menu is a menu that is invisible until the user makes a platform-specific mouse action, such as pressing the right mouse button, over a popup-enabled component. The popup menu then appears under the cursor.

Menu


Edit Text
File Edit
Contoh Tulisan


Edit Text
File Edit
New
Exit
Contoh Tulisan


Edit Text
File Edit
Bold
Italic
Uppercase
Lowercase

Menu

```
25  public class Frame extends JFrame{
26      public Frame (){
27          this.setSize(250,200);
28          this.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
29          this.setTitle("Edit Text");
30          this.setVisible(true);
31          this.setLocation(0,0);
32          JMenuBar menubar = new JMenuBar();
33          this.setJMenuBar(menubar);
34          JMenu menuFile = new JMenu ("File");
35          JMenu menuEdit = new JMenu ("Edit");
36          menubar.add(menuFile);
37          menubar.add(menuEdit);
38          JMenuItem menuitemNew = new JMenuItem("New");
39          JMenuItem menuitemExit = new JMenuItem("Exit");
40          menuFile.add(menuitemNew);
41          menuFile.addSeparator();
42          menuFile.add(menuitemExit);
43          JCheckBoxMenuItem chbmiBold = new JCheckBoxMenuItem("Bold");
44          JCheckBoxMenuItem chbmiItalic = new JCheckBoxMenuItem("Italic");
45          JRadioButtonMenuItem rbmiUppercase =new
46          JRadioButtonMenuItem("Uppercase");
47          JRadioButtonMenuItem rbmiLowercase =new
48          JRadioButtonMenuItem("Lowercase");
49          ButtonGroup bg1 =new ButtonGroup();
50          bg1.add(rbmiUppercase);
51          bg1.add(rbmiLowercase);
52          menuEdit.add(chbmiBold);
53          menuEdit.add(chbmiItalic);
54          menuEdit.addSeparator();
55          menuEdit.add(rbmiUppercase);
56          menuEdit.add(rbmiLowercase);
57          final JLabel teks = new JLabel();
58          teks.setText("Contoh Tulisan");
59          this.add(teks);
60      }
61  }
```

Aisyah Fitri Yuniasih, M. Si
aisyah.fy@stis.ac.id