PROGRAMMING III
OOP. JAVA LANGUAGE

COURSE 7

PREVIOUS COURSE CONTENT

- Exceptions
- Database access

COURSE CONTENT

- **Graphical User Interfaces**

- **Abstract Windows Toolkit**
  - Components
  - Containers
  - Layout Managers
  - Action Management
  - Drawing Components

GRAFICAL USER INTERFACE

- ❑ **What are Grafical User Interfaces (GUI)?**

GRAFICAL USER INTERFACE

- ❑ **What are Grafical User Interfaces (GUI)?**

  - ❑ Is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation

JAVA GUI IMPLEMETATIONS

- **Grafical User Interfaces**
  - Abstract Windows Toolkit (AWT)
  - Swing
  - Java FX

IMPLEMENTING GUI IN JAVA

- **The Java Foundation Classes (JFC) are a set of packages encompassing the following APIs**
  - Abstract Window Toolkit (AWT)
    - Native GUI components
  - Swing
    - Lightweight GUI components
  - 2D
    - Rendering two-dimensional shapes, text, and images
  - Accessibility
    - Allowing compatibility with, for example, screen readers and screen magnifiers

AWT

- **First Java API used for GUI applicaions buildig**

- **Provides basic UI components**
  - Buttons, lists, menus, textfields, etc
  - Event handling mechanism
  - Clipboard and data transfer
  - Image manipulation
  - Font manipulation
  - Graphics

- **Platform independence is achieved through peers, or native GUI components**

AWT

- **Creation of a graphical application inculdes**
  - Design definition
    - Creation of a displaying surface (eg. window) on wich the the components (buttons, text fields/area, lists, ..) used for communication with user will lay
    - Creation and positioning the grafical components on the created surface
  - Adding functionality
    - Defing of some actions that have to be executed when the user intercats with application graphical components
    - Adding listeners to components in order to link the user actions with the desired behavior for that components

COURSE CONTENT

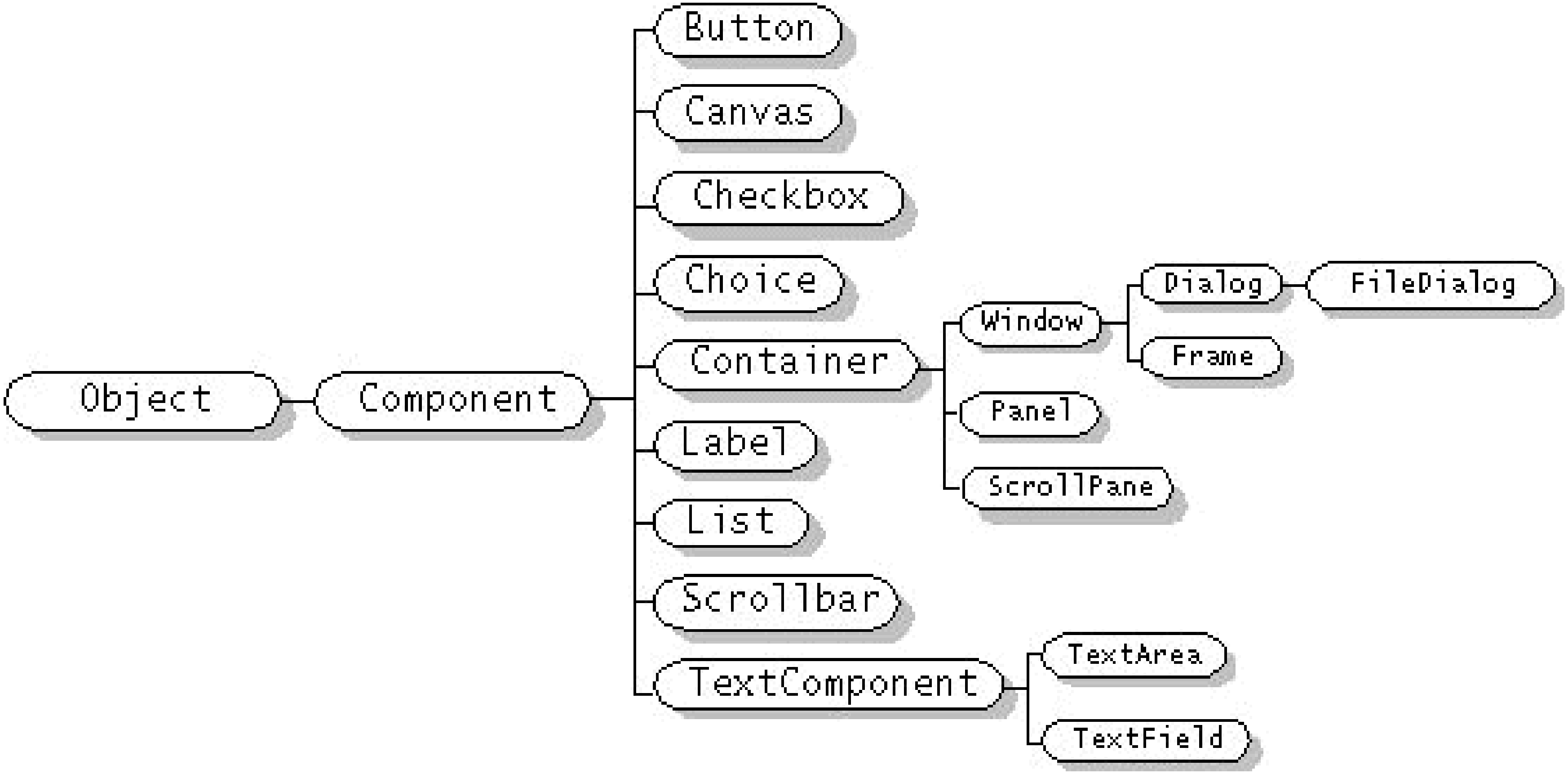
- **Graphical User Interfaces**

- **Abstract Windows Toolkit**
  - Components
  - Containers
  - Layout Managers
  - Action Management
  - Drawing Components

AWT. COMPONENTS

- **Components**
  - Graphical elements that allow the user to interact with the program and provide the user with visual feedback about the state of the program
  - Examples: buttons, scrollbars, text fields, ...
  - Component class - superclas of all components
  - Componets are gruped into containers
- **Containers**
  - Contain and control the layout of components
  - Are components, and can thus be placed inside other containers

AWT. COMPONENTS



AWT. CONTAINERS TYPE

- **Window**
  - A top-level display surface (a window).
  - An instance of the Window class is not attached to nor embedded within another container.
  - An instance of the Window class has no border and no title.
- **Frame**
  - A top-level display surface (a window) with a border and title.
  - An instance of the Frame class may have a menu bar. It is otherwise very much like an instance of the Window class.
- **Dialog**
  - A top-level display surface (a window) with a border and title.
  - An instance of the Dialog class cannot exist without an associated instance of the Frame class.
- **Panel**
  - A generic container for holding components.
  - An instance of the Panel class provides a container to which to add components.

AWT. CONTAINER CREATION

- **BULDING APPLICATION**
  - first create an instance of class Window or class Frame
- **APPLET**
  - a frame (the browser window) already exists

```
public class Example1
{
  public static void main(String [] args)
  {
    Frame f = new Frame("Example 1");
    f.show();
  }
```

```
public class Example1a extends Panel
{
  public static void main(String [] args)
  {
    Frame f = new Frame("Example 1a");
    Example1a ex = new Example1a();
    f.add("Center", ex);
    f.pack();
    f.show();
  }
}
```

AWT. ADDING COMPONENTS

- **a user interface must consist of more than just a container**
  - Components are added to containers via a container's add() method
    - There are three basic forms of the add() method.
    - The method to use depends on the container's layout manager

```
public class Example3 extends
java.applet.Applet {
  public void init()   {
    add(new Button("One"));
    add(new Button("Two"));
  }
  public Dimension preferredSize()
  {
    return new Dimension(200, 100);
  }
```

```
  public static void main(String [] args)
  {
    Frame f = new Frame("Example 3");
    Example3 ex = new Example3();
    ex.init();
    f.add("Center", ex);
    f.pack();
    f.show();
  }
}
```

COURSE CONTENT

- **Garaphical User Interfaces**

- **Abstract Windows Toolkit**
  - Components
  - Containers
  - <u>Layout Managers</u>
  - Action Managment
  - Drawing Components

AWT. COMPONENT LAYOUT

- **Layout manager**
  - Makes all of the component placement decisions
  - Layout manager classes implement the LayoutManager interface
  - Types of managers
    - FlowLayout
    - BorderLayout
    - CardLayout
    - GridLayout
    - GridBagLayout

AWT. COMPONENT LAYOUT

- **Every container has a default layout manager, but we can explicitly set the layout manager as well**
  - JPanel default = FlowLayout
  - JFrame default = BorderLayout
- **Each layout manager has its own particular rules governing how the components will be arranged**
- **Some layout managers pay attention to a component's preferred size or alignment, while others do not**
- **A layout manager attempts to adjust the layout as components are added and as containers are resized**

AWT. COMPONENT LAYOUT

- **We can use the setLayout method of a container to change its layout manager**

  - General syntax
    - container.setLayout(new LayoutMan());

  - Examples:

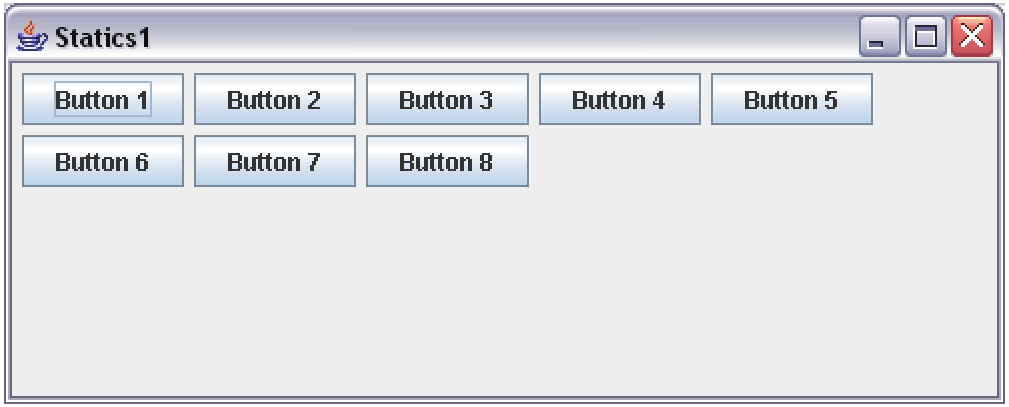
    - Panel p1 = new Panel(new BorderLayout());

    - Panel p2 = new Panel();
    - p2.setLayout(new BorderLayout());

AWT. COMPONENT LAYOUT

- **Flow Layout**
  - Puts as many components as possible on a row, then moves to the next row
  - Rows are created as needed to accommodate all of the components
  - Components are displayed in the order they are added to the container
  - Each row of components is centered horizontally in the window by default, but could also be aligned left or right
  - Also, the horizontal and vertical gaps between the components can be explicitly set

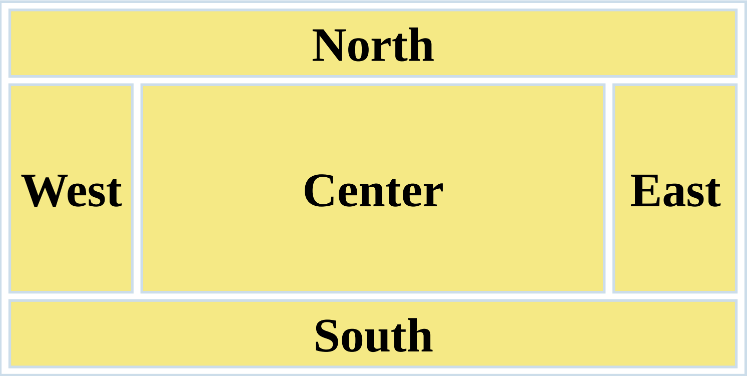
AWT. COMPONENT LAYOUT



- Flow Layout - example

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

public class Statics1 {
   public static void main(String[] args) {
      new S1GUI();
   }
}

class S1GUI {
   private JFrame f;

   public S1GUI() {
      f = new JFrame("Statics1");
      f.setSize(500, 200);
      f.setLayout(new FlowLayout(FlowLayout.LEFT));
      for (int b = 1; b < 9; b++)
         f.add(new JButton("Button " + b));
      f.setVisible(true);
   }
}
```

AWT. COMPONENT LAYOUT



North

West

Center

East

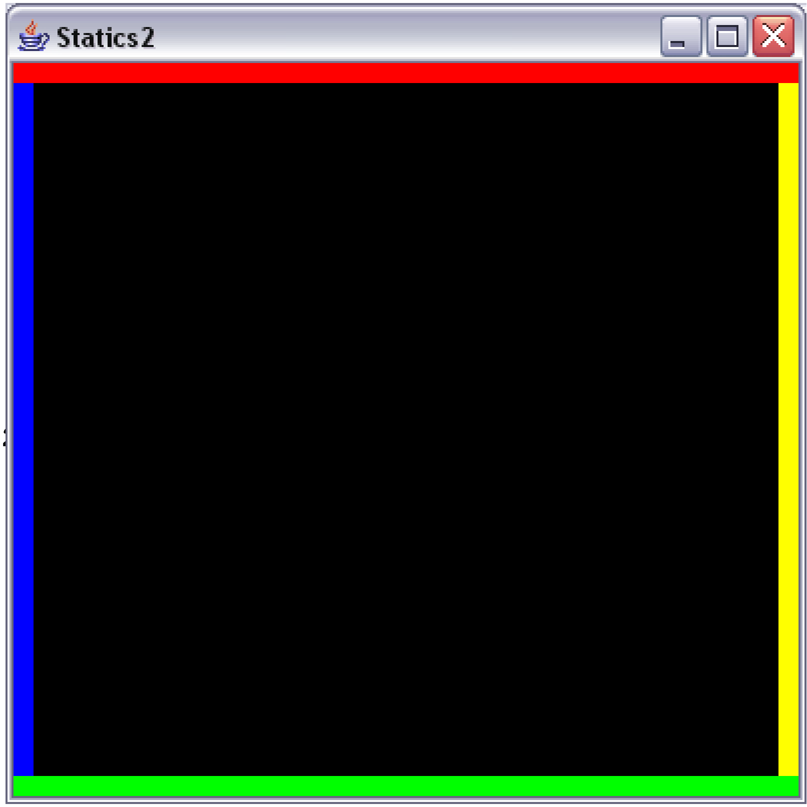
South

- ❑ **Border layout**
  - ❑ A *border layout* defines five areas into which components can be added
  - ❑ Each area displays one component (which could be a container such as a `JPanel`)
  - ❑ Each of the four outer areas enlarges as needed to accommodate the component added to it
  - ❑ If nothing is added to the outer areas, they take up no space and other areas expand to fill the void
  - ❑ The center area expands to fill space as needed

AWT. COMPONENT LAYOUT

- Border layout - example

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

public class Statics2 {
   public static void main(String[] args) { new S
}

class ColoredJPanel extends Panel {
   Color color;
   ColoredJPanel(Color color) {
      this.color = color;
   }
   public void paint(Graphics g) {
      g.setColor(color);
      g.fillRect(0, 0, 400, 400);
   }
}

class S2GUI extends Frame {
   public S2GUI() {
      setTitle("Statics2");
      addWindowListener(new WindowAdapter() {
           @Override
           public void windowClosing(WindowEvent we) {
                System.exit(0);
           }
     });
      setSize(400, 400);
      add(new ColoredJPanel(Color.RED), BorderLayout.NORTH);
      add(new ColoredJPanel(Color.GREEN), BorderLayout.SOUTH);
      add(new ColoredJPanel(Color.BLUE), BorderLayout.WEST);
      add(new ColoredJPanel(Color.YELLOW), BorderLayout.EAST);
      add(new ColoredJPanel(Color.BLACK), BorderLayout.CENTER);
      setVisible(true);
   }
}
```



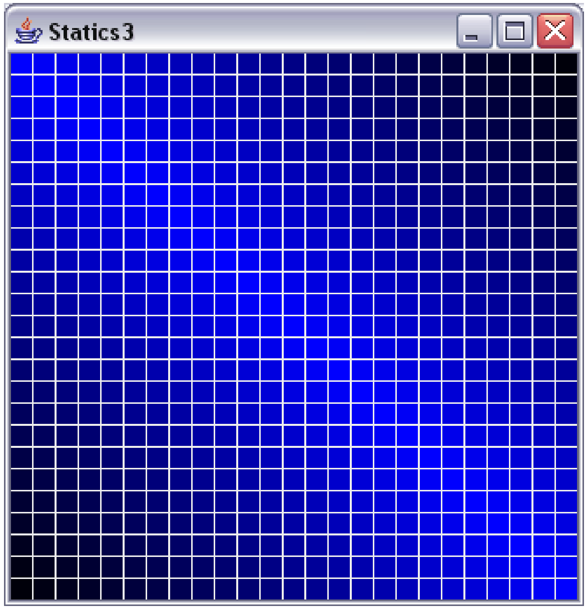
AWT. COMPONENT LAYOUT

- **GridLayout**
  - A *grid layout* presents a container’s components in a rectangular grid of rows and columns
  - One component is placed in each cell of the grid, and all cells have the same size
  - As components are added to the container, 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

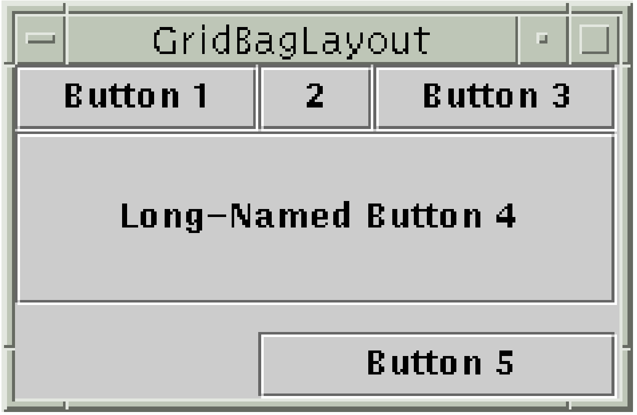
AWT. COMPONENT LAYOUT

- **GridLayout - example**

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

public class Statics3 {
   public static void main(String[] args) { new S3GUI(); }
}

class S3GUI extends Frame {
   static final int DIM = 25;
   static final int SIZE = 12;
   static final int GAP = 1;

   public S3GUI() {
      setTitle("Statics3");
      addWindowListener(new WindowAdapter() {
           @Override
           public void windowClosing(WindowEvent we) {
                System.exit(0);
           }
     });
      setLayout(new GridLayout(DIM, DIM, GAP, GAP));
      for (int i = 0; i < DIM * DIM; i++) add(new MyPanel());
      pack();
      setVisible(true);
   }

   class MyPanel extends Panel {
      MyPanel() { setPreferredSize(new Dimension(SIZE, SIZE)); }
      public void paint(Graphics g) {
         float gradient =
            1f - ((float)Math.abs(getX() - getY()))/(float)((SIZE + GAP) * DIM);
         g.setColor(new Color(0f, 0f, gradient));
         g.fillRect(0, 0, getWidth(), getHeight());
      }
   }
}
```



AWT. COMPONENT LAYOUT



- **GridBagLayout**
  - Divides the window into grids, without requiring the components to be the same size
  - More flexible than the other standard layout managers, but harder to use
  - Each component managed by a grid bag layout is associated with an instance of *GridBagConstraints*
  - The GridBagConstraints specifies:
    - How the component is laid out in the display area
    - In which cell the component starts and ends
    - How the component stretches when extra room is available
    - Alignment in cells

AWT. COMPONENT LAYOUT

- **GridBagLayout - steps to use**
  - Set the layout, saving a reference to it

```
          GridBagLayout layout = new GridBagLayout();
          setLayout(layout);
```

  - Allocate a `GridBagConstraints` object

```
          GridBagConstraints constraints = new
GridBagConstraints();
```

  - Set up the `GridBagConstraints` for component 1

```
          constraints.gridx = x1;
          constraints.gridy = y1;
          constraints.gridwidth = width1;
          constraints.gridheight = height1;
```

  - Add component 1 to the window, including constraints

```
          add(component1, constraints);
```

  - Repeat the last two steps for each remaining component

AWT. COMPONENT LAYOUT

- **GridBagConstraints - Properties**
  - gridx, gridy
    - Specifies the top-left corner of the component
    - Upper left of grid is located at (gridx, gridy)=(0,0)
    - Set to GridBagConstraints.RELATIVE to auto-increment row/column

```
GridBagConstraints constraints = new GridBagConstraints();
constraints.gridx = GridBagConstraints.RELATIVE;
container.add(new Button("one"), constraints);
container.add(new Button("two"), constraints);
```

AWT. COMPONENT LAYOUT

- **GridBagConstraints - Properties**
  - gridwidth, gridheight
    - Specifies the number of columns and rows the Component occupies

      constraints.gridwidth = 3;
    - GridBagConstraints.REMAINDER lets the component take up the remainder of the row/column
  - weightx, weighty
    - Specifies how much the cell will stretch in the x or y direction if space is left over

      constraints.weightx = 3.0;
    - Constraint affects the cell, not the component (use fill)
    - Use a value of 0.0 for no expansion in a direction
    - Values are relative, not absolute

AWT. COMPONENT LAYOUT

- **GridBagConstraints - Properties**
  - fill
    - Specifies what to do to an element that is smaller than the cell size
      constraints.fill = GridBagConstraints.VERTICAL;
    - The size of row/column is determined by the widest/tallest element in it
    - Can be NONE, HORIZONTAL, VERTICAL, or BOTH
  - anchor
    - If the fill is set to GridBagConstraints.NONE, then the anchor field determines where the component is placed
    - constraints.anchor = GridBagConstraints.NORTHEAST;
    - Can be NORTH, EAST, SOUTH, WEST, NORTHEAST, NORTHWEST, SOUTHEAST, or SOUTHWEST

AWT. COMPONENT LAYOUT

- GridBagLayout - example

```
public class Statics4 {public static void
main(String[] args) { new S4GUI(); }}

class S4GUI extends JFrame {

    public S4GUI() {

      setTitle("Statics4");

      setDefaultCloseOperation(EXIT_ON_CLOSE);


      JButton button;

      Container contentPane = getContentPane();

      GridBagLayout gridbag = new GridBagLayout();

      GridBagConstraints c = new
GridBagConstraints();

      contentPane.setLayout(gridbag);

      c.fill = GridBagConstraints.HORIZONTAL;


      button = new JButton("Button 1");

      c.weightx = 0.5;

      c.gridx = 0;

      c.gridy = 0;

      gridbag.setConstraints(button, c);

      contentPane.add(button);
```

```
button = new JButton("2");

c.gridx = 1;

c.gridy = 0;

gridbag.setConstraints(button, c);

contentPane.add(button);


button = new JButton("Button 3");

c.gridx = 2;

c.gridy = 0;

gridbag.setConstraints(button, c);

contentPane.add(button);


button = new JButton("Long-Named Button 4");

c.ipady = 40;      //make this component tall

c.weightx = 0.0;

c.gridwidth = 3;

c.gridx = 0;

c.gridy = 1;

gridbag.setConstraints(button, c);

contentPane.add(button);
```

```
button = new JButton("Button 5");

      c.ipady = 0;       //reset to default

      c.weighty = 1.0;

//request any extra vertical space

      c.anchor = GridBagConstraints.SOUTH;
//bottom of space

      c.insets = new Insets(10,0,0,0);

//top padding

      c.gridx = 1;       //aligned with button 2

      c.gridwidth = 2;   //2 columns wide

      c.gridy = 2;       //third row

      gridbag.setConstraints(button, c);

      contentPane.add(button);


      pack();

      setVisible(true);

   }



}
```

AWT. COMPONENT LAYOUT

CARD LAYOUT

- CardLayout
  - Stacks components on top of each other, displaying the top one
  - Associates a name with each component in window

```
Panel cardPanel;
CardLayout layout new CardLayout();
Panel.setLayout(layout);
...
cardPanel.add("Card 1", component1);
cardPanel.add("Card 2", component2);
                ...
layout.show(cardPanel, "Card 1");
layout.first(cardPanel);
layout.next(cardPanel);
```

NULL LAYOUT

- NullLayout
  - Manually sets relative position of the components

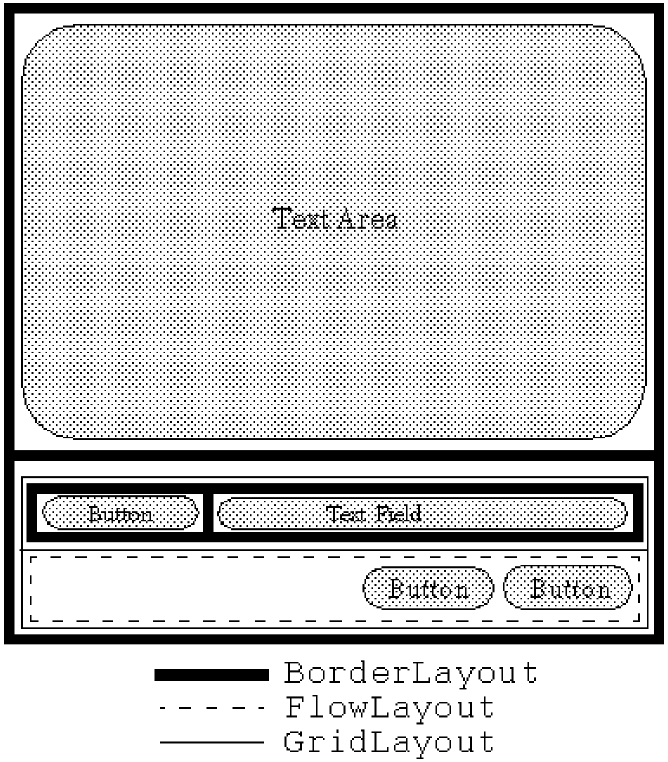
```
setLayout(null);
Button b1 = new Button("Button 1");
Button b2 = new Button("Button 2");
    ...
b1.setBounds(0, 0, 150, 50);
b2.setBounds(150, 0, 75, 50);
    ...
add(b1);
add(b2);
    ...
```

AWT. COMPONENT LAYOUT

- **Use nested containers**
  - Rather than struggling to fit your design in a single layout, try dividing the design into sections
  - Let each section be a panel with its own layout manager
- **Turn off the layout manager for some containers**
- **Adjust the empty space around components**
  - Change the space allocated by the layout manager
  - Override insets in the Container
  - Use a Canvas or a Box as an invisible spacer

AWT. COMPONENT LAYOUT

```
    JPanel subPanel1 = new JPanel();
    JPanel subPanel2 = new JPanel();
    subPanel1.setLayout(new BorderLayout());
    subPanel2.setLayout(new
FlowLayout(FlowLayout.RIGHT,2,2));

    subPanel1.add(bSaveAs,BorderLayout.WEST)
subPanel1.add(fileField,BorderLayout.CENTER)
    subPanel2.add(bOk);
    subPanel2.add(bExit);

    bottomPanel.add(subPanel1);
    bottomPanel.add(subPanel2);

    add(bottomPanel,BorderLayout.SOUTH);
```



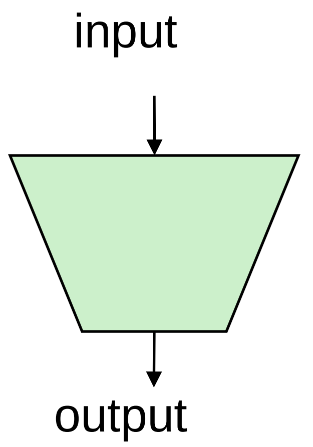
COURSE CONTENT

- **Graphical User Interfaces**

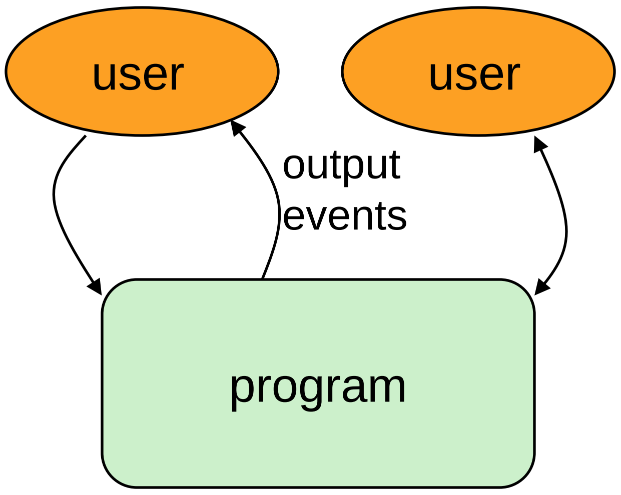
- **Abstract Windows Toolkit**
  - Components
  - Containers
  - Layout Managers
  - Action Management
  - Drawing Components

INTERACTIVE PROGRAMS

- “Classic” view of computer programs: Transform inputs to outputs, stop



- Event-driven programs: interactive, long-running
  - Servers interact with clients
  - Applications interact with user(s)



EVENT-DRIVEN PROGRAMMING

- Reactive
- Program's execution is indeterminate
- On-screen components cause *events* to occur when they are clicked / interacted with
- Events can be handled, causing the program to respond, *driving* the execution thru events (an "event-driven" program)
- Typically uses a GUI (Graphical User Interface)

JAVA EVENT HIERARCHY

```
java.lang.Object
  +--java.util.EventObject
        +--java.awt.AWTEvent
              +--java.awt.event.ActionEvent
              +--java.awt.event.TextEvent
              +--java.awt.event.ComponentEvent
                    +--java.awt.event.FocusEvent
                    +--java.awt.event.WindowEvent
                    +--java.awt.event.InputEvent
                          +--java.awt.event.KeyEvent
                          +--java.awt.event.MouseEvent
```

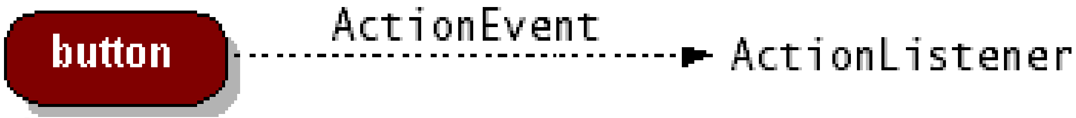
```
import java.awt.event.*;
```

EVENT HANDELING STRATEGY

- **Determine what type of listener is of interest**
  - 11 standard AWT listener types.
    - ActionListener, AdjustmentListener, ComponentListener, ContainerListener, FocusListener, ItemListener, KeyListener, MouseListener, MouseMotionListener, TextListener, WindowListener
- **Define a class of that type**
  - Implement interface (KeyListener, MouseListener, etc.)
  - Extend class (KeyAdapter, MouseAdapter, etc.)
- **Register an object of your listener class with the window**
  - w.addXxxListener(new MyListenerClass());
  - E.g., addKeyListener, addMouseListener

EVENT HANDELING STRATEGY

- Example



  - Adding actions to a button
    - Create an action listener

```
public class MyActionListener
                    implements ActionListener {
   public void actionPerformed(ActionEvent event){
      System.out.println("Event occurred!");
   }
}
```

```
public interface ActionListener {
  void actionPerformed(ActionEvent event);
}
```

    - Add action lister to the button

```
Button button = new JButton("button 1");
ActionListener listener = new MyActionListener();
button.addActionListener(listener);
```

ACTION LISTERNERS

- **ActionEvent class**
  - `public Object getSource()`
    Returns object that caused this event to occur.
  - `public String getActionCommand()`
    Returns a string that represents this event.
    (for example, text on button that was clicked)
- **How to implemet action listeners?**

ACTION LISTERNERS

- **How to implemet action listeners?**
  - Handling events with separate listeners
  - Handling events by main class
  - Handling events with named inner classes
  - Handling events with anonymous inner classes

ACTION LISTERNERS

- **How to implemet action listeners?**
  - Handling events with separate listeners
    - Create a separate class to handele the event

```
public class MyActionListener
           implements ActionListener {
  public void actionPerformed(ActionEvent event){
    System.out.println("Event occurred!");
  }
}
```

    - Add action lister to the button

```
Button button = new Button("button 1");
ActionListener listener = new MyActionListener();
button.addActionListener(listener);
```

  - Handling events by main class
  - Handling events with named inner classes
  - Handling events with anonymous inner classes

ACTION LISTERNERS

- ❑ **How to implemet action listeners?**
  - ❑ Handling events with separate listeners
  - ❑ Handling events by main class

```
public class MyApplication extends Frame implements
ActionListener {
  void initComonents(){
   Button button = new Button("button 1");
   button.addActionListener(this);
   ...
  }
 ...
 public void actionPerformed(ActionEvent event){
   System.out.println("Event occurred!");
 }
}
```

  - ❑ Handling events with named inner classes
  - ❑ Handling events with anonymous inner classes

ACTION LISTERNERS

- **How to implemet action listeners?**
  - Handling events with separate listeners
  - Handling events by main class
  - Handling events with named inner classes

```
public class MyApplication extends Frame {
  void initComonents(){
   Button button = new Button("button 1");
   MyAction  action = new MyAction ()
   button.addActionListener(action);
   ...
  }
 ...
  public class MyAction implements ActionListener
{
     public void actionPerformed(ActionEvent
event){
        System.out.println("Event occurred!");
     }
   }
}
```

ACTION LISTERNERS

- **How to implemet action listeners?**
  - Handling events with separate listeners
  - Handling events by main class
  - Handling events with named inner classes
  - Handling events with anonymous inner classes

```
public class MyApplication extends Frame {
  void initComonents(){
   Button button = new Button("button 1");
   MyAction  action = new MyAction ()
   button.addActionListener(new ActionListner(){
      public void actionPerformed(ActionEvent event){
        System.out.println("Event occurred!");
      }
   });
   ...
  }
 }
```

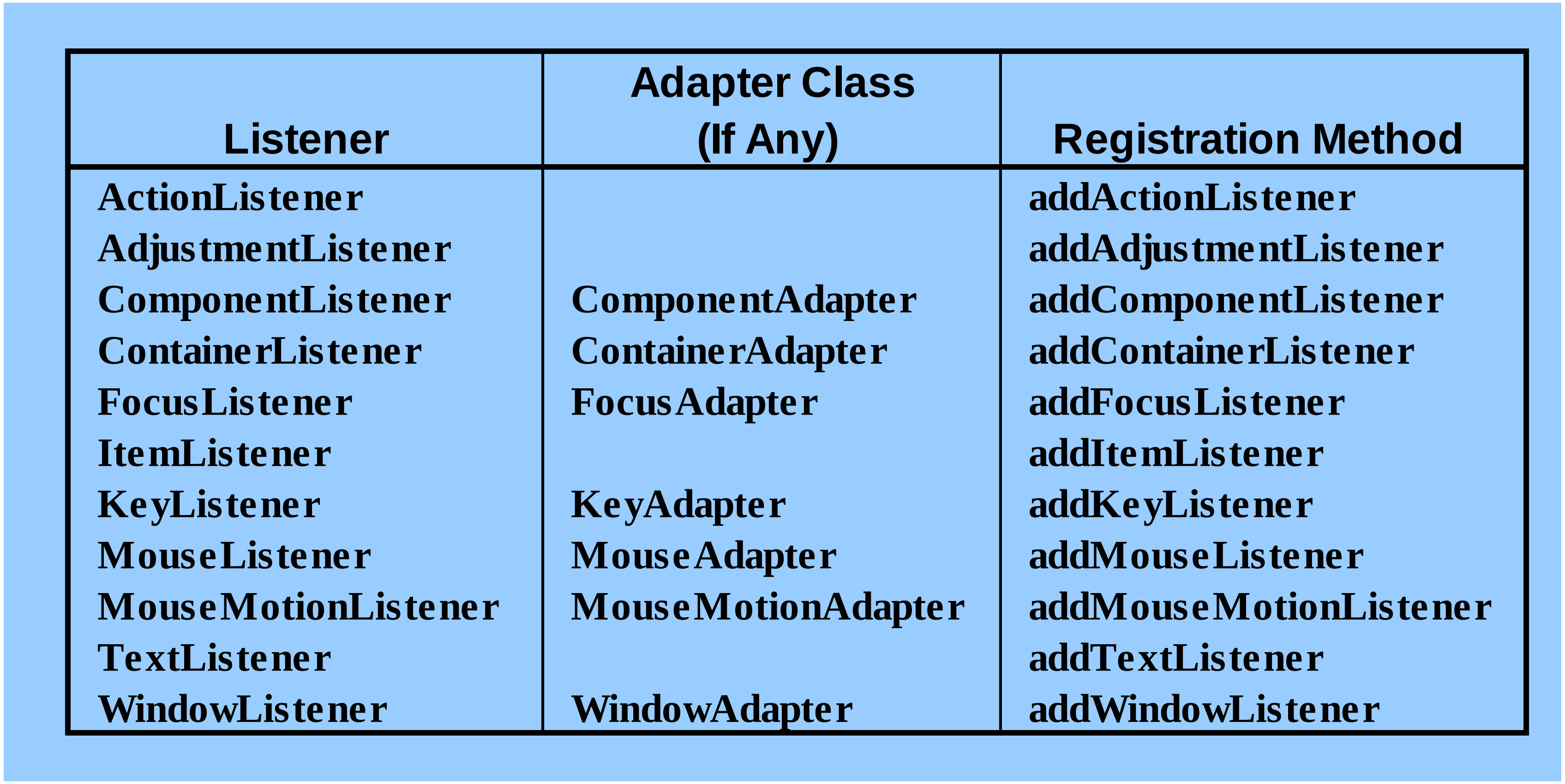
EVENT HANDLING STRATEGIES: PROS AND CONS

- **Separate Listener**
  - Advantages
    - Can extend adapter and thus ignore unused methods
    - Separate class easier to manage
  - Disadvantage
    - Need extra step to call methods in main window
- **Main window that implements interface**
  - Advantage
    - No extra steps needed to call methods in main window
  - Disadvantage
    - Must implement methods you might not care about

EVENT HANDLING STRATEGIES: PROS AND CONS

- **Named inner class**
  - Advantages
    - Can extend adapter and thus ignore unused methods
    - No extra steps needed to call methods in main window
  - Disadvantage
    - A bit harder to understand
- **Anonymous inner class**
  - Advantages
    - Same as named inner classes
    - Even shorter
  - Disadvantage
    - Much harder to understand

STANDARD AWT EVENT LISTENERS



Listener	Adapter Class (If Any)	Registration Method
ActionListener		addActionListener
AdjustmentListener		addAdjustmentListener
ComponentListener	ComponentAdapter	addComponentListener
ContainerListener	ContainerAdapter	addContainerListener
FocusListener	FocusAdapter	addFocusListener
ItemListener		addItemListener
KeyListener	KeyAdapter	addKeyListener
MouseListener	MouseAdapter	addMouseListener
MouseMotionListener	MouseMotionAdapter	addMouseMotionListener
TextListener		addTextListener
WindowListener	WindowAdapter	addWindowListener

STANDARD AWT EVENT LISTENERS

- **ActionListener**
  - Handles buttons and a few other actions
  - actionPerformed(ActionEvent event)
- **AdjustmentListener**
  - Applies to scrolling
  - adjustmentValueChanged(AdjustmentEvent event)
- **ComponentListener**
  - Handles moving/resizing/hiding GUI objects
  - componentResized(ComponentEvent event)
  - componentMoved (ComponentEvent event)
  - componentShown(ComponentEvent event)
  - componentHidden(ComponentEvent event)

STANDARD AWT EVENT LISTENERS

- **ContainerListener**
  - Triggered when window adds/removes GUI controls
  - componentAdded(ContainerEvent event)
  - componentRemoved(ContainerEvent event)

- **FocusListener**
  - Detects when controls get/lose keyboard focus
  - focusGained(FocusEvent event)
  - focusLost(FocusEvent event)

STANDARD AWT EVENT LISTENERS

- **ItemListener**
  - Handles selections in lists, checkboxes, etc.
  - itemStateChanged(ItemEvent event)

- **KeyListener**
  - Detects keyboard events
  - keyPressed(KeyEvent event) -- any key pressed down
  - keyReleased(KeyEvent event) -- any key released
  - keyTyped(KeyEvent event) -- key for printable char released

STANDARD AWT EVENT LISTENERS

- **MouseListener**
  - Applies to basic mouse events
  - mouseEntered(MouseEvent event)
  - mouseExited(MouseEvent event)
  - mousePressed(MouseEvent event)
  - mouseReleased(MouseEvent event)
  - mouseClicked(MouseEvent event) -- Release without drag
    - Applies on release if no movement since press
- **MouseMotionListener**
  - Handles mouse movement
  - mouseMoved(MouseEvent event)
  - mouseDragged(MouseEvent event)

COURSE CONTENT

- **Graphical User Interfaces**

- **Abstract Windows Toolkit**
  - Components
  - Containers
  - Layout Managers
  - Action Management
  - Drawing Components

CANVAS

- **Canvas**
  - Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
- **AWT**

```
public void paint(Graphics g) {
        ...
}
```

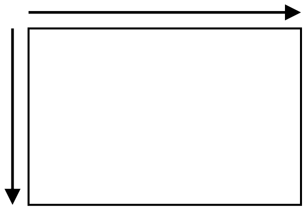
  - no default double buffering
- **Swing**

```
public void paintComponent(Graphics g) {
        super.paintComponent(g);
        ...
}
```

  - default double buffering

COORDINATE SYSTEM

- Each (x, y) position is a pixel ("picture element").
- (0, 0) is at the window's top-left corner.
  - x increases rightward and the y increases downward



DRAWING METHODS

Method name	Description
`g.drawLine(`**x1**, **y1**, **x2**, **y2**`);`	line between points (*x1*, *y1*), (*x2*, *y2*)
`g.drawOval(`**x**, **y**, **width**, **height**`);`	outline largest oval that fits in a box of size *width* * *height* with top-left at (*x*, *y*)
`g.drawRect(`**x**, **y**, **width**, **height**`);`	outline of rectangle of size *width* * *height* with top-left at (*x*, *y*)
`g.drawString(`**text**, **x**, **y**`);`	text with bottom-left at *(x, y)*
`g.fillOval(`**x**, **y**, **width**, **height**`);`	fill largest oval that fits in a box of size *width* * *height* with top-left at (*x*, *y*)
`g.fillRect(`**x**, **y**, **width**, **height**`);`	fill rectangle of size *width* * *height* with top-left at (*x*, *y*)
`g.setColor(`**Color**`);`	set `Graphics` to paint any following shapes in the given color

COLOR

Create one using Red-Green-Blue (RGB) values from 0-255

```
Color name = new Color(red, green, blue);
```

- Example:
  ```
  Color brown = new Color(192, 128, 64);
  ```

Or use a predefined `Color` class constant (more common)

```
Color.CONSTANT_NAME
```

where CONSTANT_NAME is one of:

- `BLACK`, `BLUE`, `CYAN`, `DARK_GRAY`, `GRAY`, `GREEN`, `LIGHT_GRAY`, `MAGENTA`, `ORANGE`, `PINK`, `RED`, `WHITE`, or `YELLOW`

EXAMPLE

```
public class ExPaint {
    public static void main(String[] args) {
        JFrame f = new JFrame("Swing Paint Demo");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(new MyPanel());  f.pack();  f.setVisible(true);
    }
}

class MyPanel extends JPanel {
    public MyPanel() {  setBorder(BorderFactory.createLineBorder(Color.blac
    public Dimension getPreferredSize() { return new Dimension(250, 200);}
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.red);
        for (int i = 0; i < 6; i++) {
            g.drawRect(11 + 20 * i, 150 - 20 * i, 20, 20);
        }
    }
}
```