# 1.00 Lecture 19

**More on Events**

**Inner Classes**

**Layout Managers**

**Reading for next time: 18.3**

## Java Event Model: Recap

- How do GUIs *interact* with users? How do applications recognize when the user has done something?
- In Java this depends on 3 related concepts:
  - Events: objects that represent a user action with the system
  - Event sources: in Swing, these are components that can recognize user action, like a button or an editable text field
  - Event listeners: objects that can respond when an event occurs

# Events

- **Events are instances of simple classes (objects) that supply information about what happened.**
  - **Instances of `ActionEvent` have `getSource()` methods to return the object that fired the event**
  - **Instances of `MouseEvent` have `getX()` and `getY()` methods that will tell you where the mouse event (e.g., mouse press) occurred. And so on.**
- **The event object is delivered to the event listener by the operating system and Java Virtual Machine**
  - **Listener methods are invoked when they receive an event object from the OS or JVM**
  - **Your Java code does not explicitly create event objects**
  - **Your Java code does not call event listeners explicitly**

# Event Sources

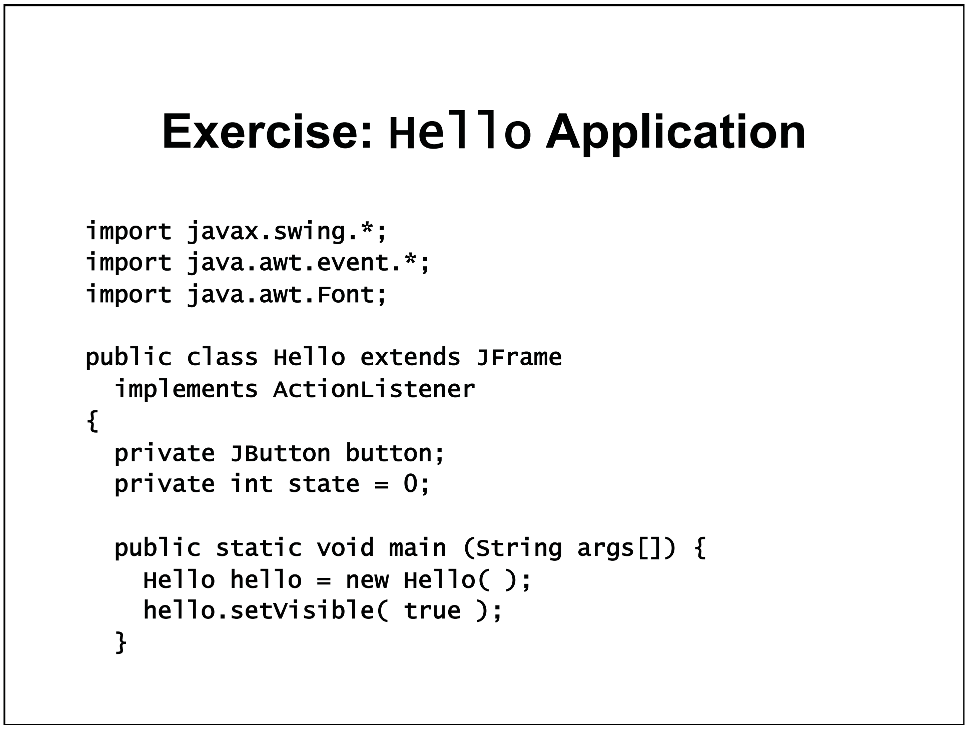
- **Event sources generate events**
- **The ones you will be most interested in are subclasses of `JComponent` like `JButton` and `JComboBox`**
- **You will use already-written classes as your event sources**
- **Or inherit from them (e.g. `SwitchButton` inherits from `JButton`)**
  - **There is a class `EventSource` that you can inherit from if you want to create a new source type**

# Event Listeners

- Event listeners
  - An object becomes an event listener when its class implements an event listener interface
  - The event listener gets called when the event occurs if we register the event listener with the event source
  - All event listener methods take an event as an argument
- You may select any object, as long as it implements ActionListener (or XXXListener), to be the event listener. You have three options:
  - Use an existing GUI element
    - Make the containing panel listen to its buttons, etc., as in both examples in class so far. Simple but not ideal.
  - Create instance (object) of new class as listener
  - Create inner class object as listener (covered next)

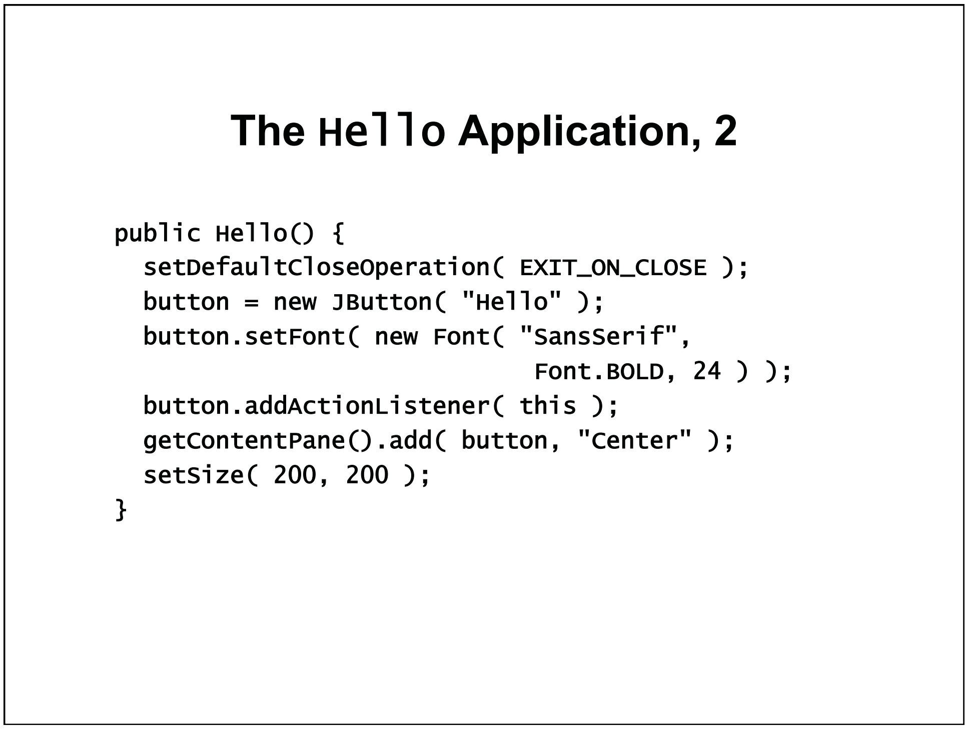
# Exercise

- There are 5 steps to handling an event.
- Mark up the next three slides:
  - Circle and label where steps 1, 2, 3, 4 and 5 occur:
    - Step 1: Identify type and source of event
    - Step 2: Identify object to handle event
    - Step 3: Select appropriate listener interface
    - Step 4: Write listener method required by interface
    - Step 5: Register listener with event source



## Exercise: `Hello` Application

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

public class Hello extends JFrame
  implements ActionListener
{
  private JButton button;
  private int state = 0;

  public static void main (String args[]) {
    Hello hello = new Hello( );
    hello.setVisible( true );
  }
```



## The `Hello` Application, 2

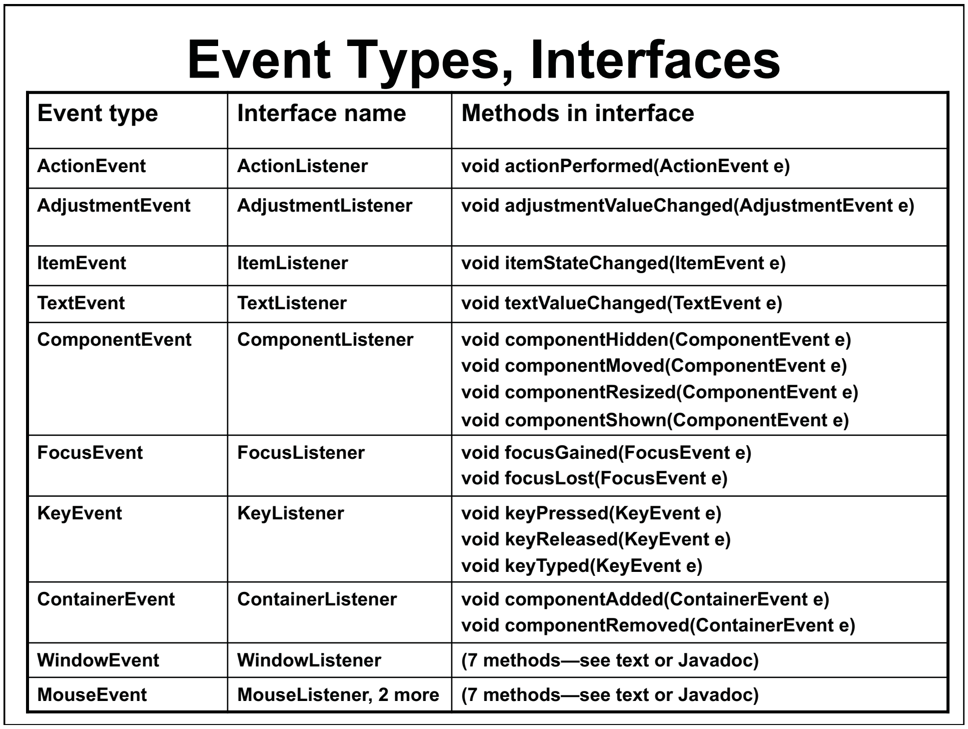
```
  public Hello() {
    setDefaultCloseOperation( EXIT_ON_CLOSE );
    button = new JButton( "Hello" );
    button.setFont( new Font( "SansSerif",
                               Font.BOLD, 24 ) );
    button.addActionListener( this );
    getContentPane().add( button, "Center" );
    setSize( 200, 200 );
  }
```

## The `Hello` Application, 3

```
   public void actionPerformed( ActionEvent e ) {
     if ( state == 0 ) {
       button.setText( "Goodbye" );
       state++;
     } else {
       System.exit( 0 );
     }
   }
 }
```

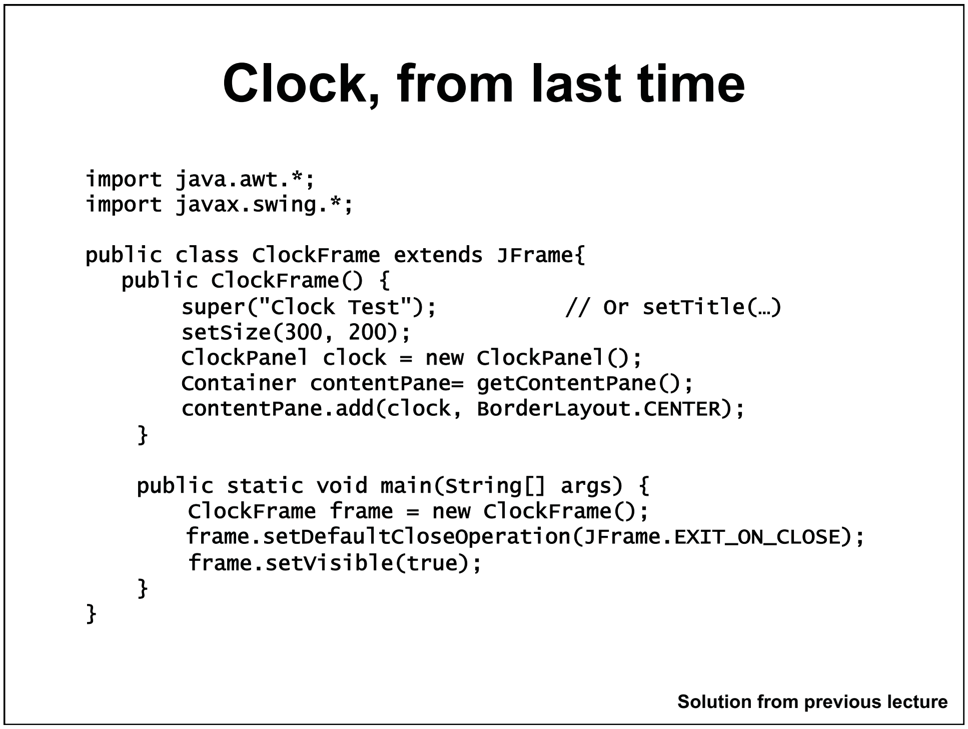
## Event Types

- Semantic events vs low-level events
  - <u>Semantic events</u> are a meaningful group of low-level events
    - ActionEvent: user action on object (button click, etc.)
    - AdjustmentEvent: value adjusted (scroll bar, etc.)
    - ItemEvent: selectable item changed (combo box)
    - TextEvent: value of text changed
  - You can often just use ActionEvent, especially if a button is present to initiate program operation
    - In actionPerformed(), you can then get the values of all other Swing components.
  - <u>Low level events</u>:
    - Mouse press, mouse move, key release, etc.
    - There are many of these



# Event Types, Interfaces

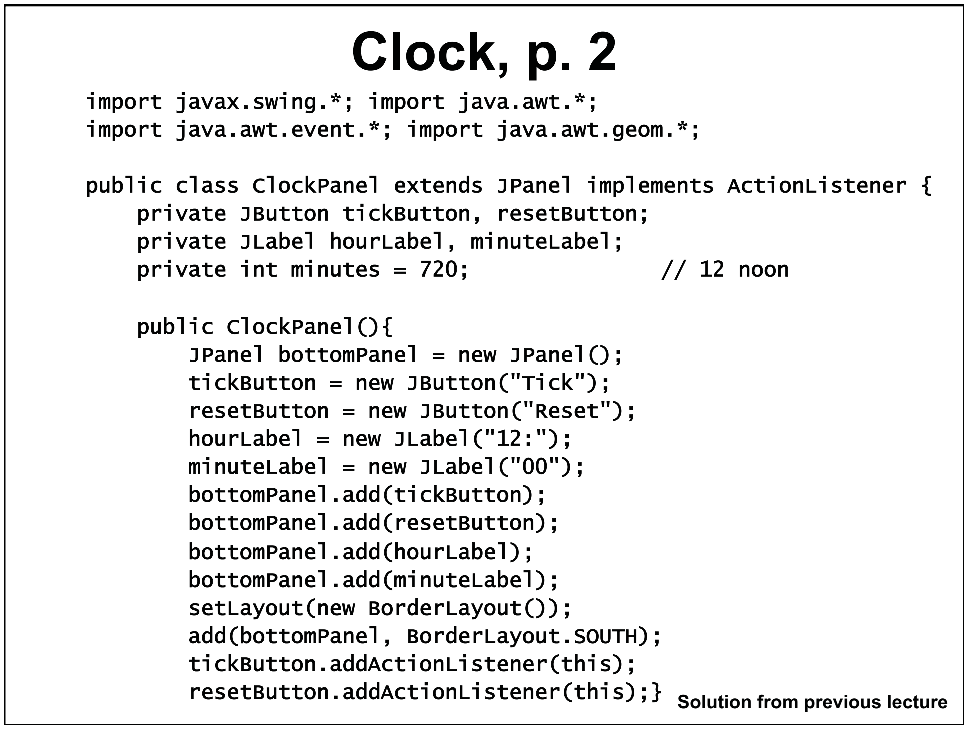
| Event type | Interface name | Methods in interface |
|---|---|---|
| ActionEvent | ActionListener | void actionPerformed(ActionEvent e) |
| AdjustmentEvent | AdjustmentListener | void adjustmentValueChanged(AdjustmentEvent e) |
| ItemEvent | ItemListener | void itemStateChanged(ItemEvent e) |
| TextEvent | TextListener | void textValueChanged(TextEvent e) |
| ComponentEvent | ComponentListener | void componentHidden(ComponentEvent e)<br>void componentMoved(ComponentEvent e)<br>void componentResized(ComponentEvent e)<br>void componentShown(ComponentEvent e) |
| FocusEvent | FocusListener | void focusGained(FocusEvent e)<br>void focusLost(FocusEvent e) |
| KeyEvent | KeyListener | void keyPressed(KeyEvent e)<br>void keyReleased(KeyEvent e)<br>void keyTyped(KeyEvent e) |
| ContainerEvent | ContainerListener | void componentAdded(ContainerEvent e)<br>void componentRemoved(ContainerEvent e) |
| WindowEvent | WindowListener | (7 methods—see text or Javadoc) |
| MouseEvent | MouseListener, 2 more | (7 methods—see text or Javadoc) |



# Clock, from last time

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

public class ClockFrame extends JFrame{
   public ClockFrame() {
        super("Clock Test");          // Or setTitle(…)
        setSize(300, 200);
        ClockPanel clock = new ClockPanel();
        Container contentPane= getContentPane();
        contentPane.add(clock, BorderLayout.CENTER);
    }

    public static void main(String[] args) {
        ClockFrame frame = new ClockFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

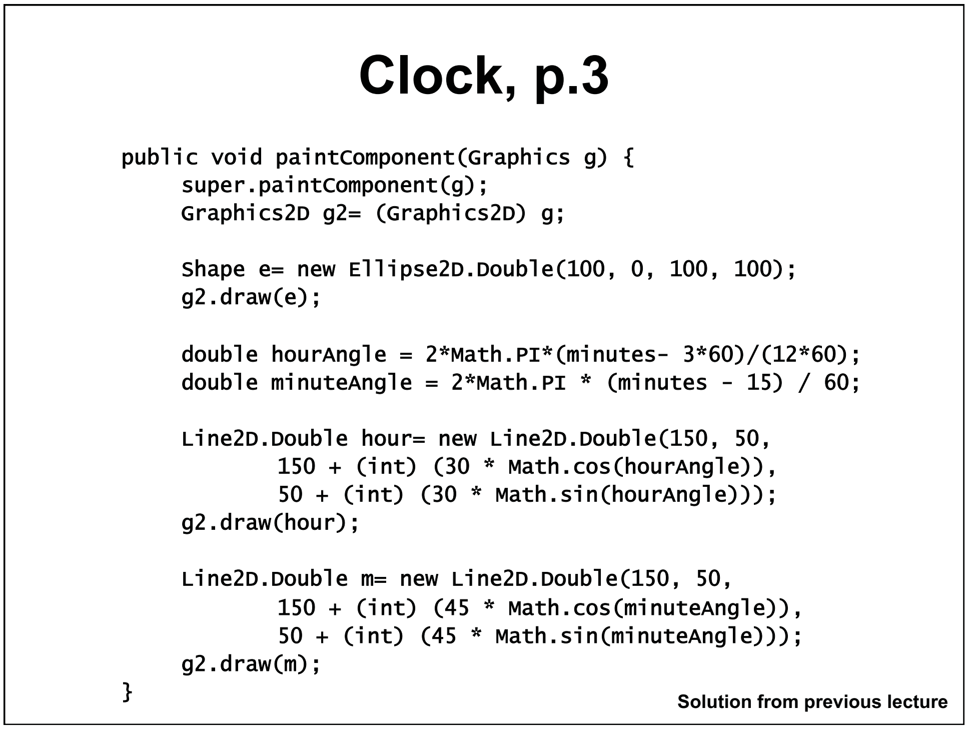
Solution from previous lecture



# Clock, p. 2

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

public class ClockPanel extends JPanel implements ActionListener {
    private JButton tickButton, resetButton;
    private JLabel hourLabel, minuteLabel;
    private int minutes = 720;              // 12 noon

    public ClockPanel(){
        JPanel bottomPanel = new JPanel();
        tickButton = new JButton("Tick");
        resetButton = new JButton("Reset");
        hourLabel = new JLabel("12:");
        minuteLabel = new JLabel("00");
        bottomPanel.add(tickButton);
        bottomPanel.add(resetButton);
        bottomPanel.add(hourLabel);
        bottomPanel.add(minuteLabel);
        setLayout(new BorderLayout());
        add(bottomPanel, BorderLayout.SOUTH);
        tickButton.addActionListener(this);
        resetButton.addActionListener(this);}
```

Solution from previous lecture



# Clock, p.3

```
public void paintComponent(Graphics g) {
     super.paintComponent(g);
     Graphics2D g2= (Graphics2D) g;

     Shape e= new Ellipse2D.Double(100, 0, 100, 100);
     g2.draw(e);

     double hourAngle = 2*Math.PI*(minutes- 3*60)/(12*60);
     double minuteAngle = 2*Math.PI * (minutes - 15) / 60;

     Line2D.Double hour= new Line2D.Double(150, 50,
            150 + (int) (30 * Math.cos(hourAngle)),
            50 + (int) (30 * Math.sin(hourAngle)));
     g2.draw(hour);

     Line2D.Double m= new Line2D.Double(150, 50,
            150 + (int) (45 * Math.cos(minuteAngle)),
            50 + (int) (45 * Math.sin(minuteAngle)));
     g2.draw(m);
}
```

Solution from previous lecture

# Clock, p.4

```
  public void setLabels(){           // Doesn't handle midnight
        int hours = minutes/60;
        int min = minutes - hours*60;
        hourLabel.setText(hours+ ":");
        if (min < 10)                // Minutes should be two digits
            minuteLabel.setText("0" + min);
        else
            minuteLabel.setText("" + min);
  }

  public void actionPerformed(ActionEvent e) {
       if(e.getSource().equals(tickButton))
               minutes++;
       else   // Reset button
               minutes= 720;
       repaint();     // Repaint redraws circle and lines
       setLabels();   // setLabels resets hour, minute text
  }
}
```

**Solution from previous lecture**

# Inner Classes

**You can define an *inner class* inside another class:**

```
public class EnclosingClass {
    public class InnerClass1 { ... }
    private class InnerClass2 { ... }
}
```

- **Inner class name is the outer class name qualified with the inner class name: e.g., `EnclosingClass.InnerClass1`**
  - **You already saw `Rectangle2D.Double` (it's static, a slight variation)**
- **An inner class is considered to be part of the enclosing class:**
  - **Make it `public` if you want methods in other classes to use it**
  - **Make it `private` if you only use it in the enclosing class**
- **The inner class has access to instance data and methods of the enclosing class**
- **The enclosing class has access to instance data and methods of the inner class, even if it is `private`**

# Exercise 1: Inner classes

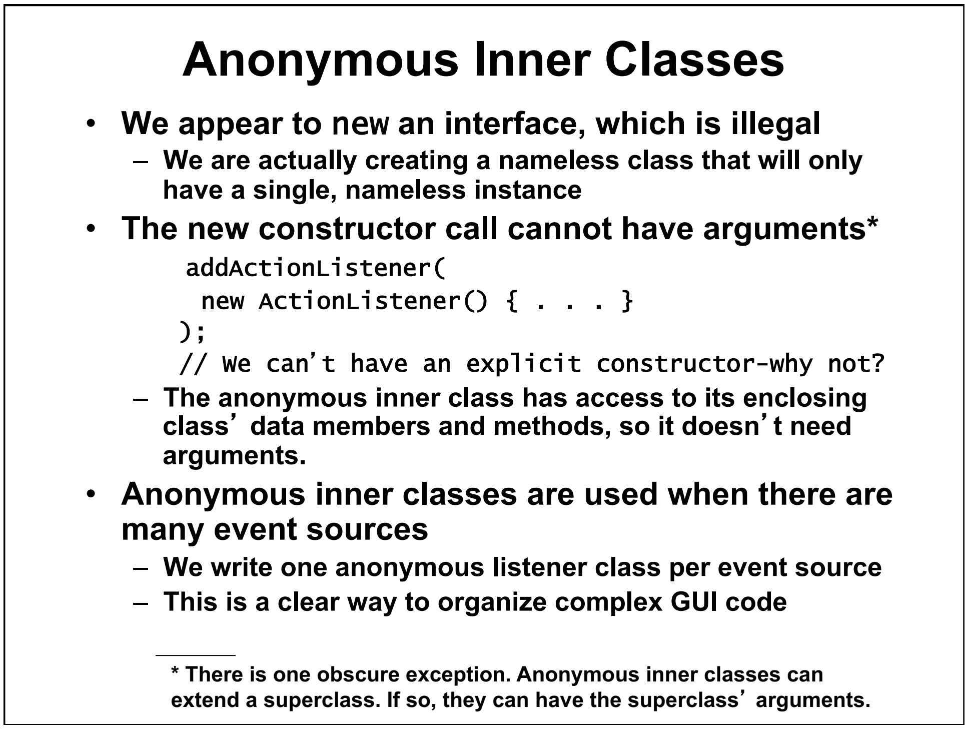
- Create a TickButtonListener inner class inside ClockPanel. Put it after the data members.
  - Same syntax as any other class, but defined inside a class
  - Must implement ActionListener interface
  - Must have actionPerformed() method to increment minutes
  - No constructor or data members needed in inner class
- Create ResetButtonListener inner class inside ClockPanel in same way.
  - Its actionPerformed() method sets minutes=720.
- Create instances (new) of the inner classes and register them as the listeners for the tick and reset buttons
  - Can do it all in one line, in addActionListener(). Use new …
- ClockPanel no longer implements ActionListener or has actionPeformed()
  - Remove actionPerformed() method from ClockPanel

# Anonymous Inner Classes

- Shortcut way to define inner classes
  - Used for small, simple classes such as listeners
  - Separates listener from source in a simple way
- There is no public class declaration
  - The class is defined and the object is created (new) within the argument to addActionListener()

```
// Code fragment for the button within a JPanel
public class SomePanel extends JPanel {
   private JButton someButton;
   public SomePanel() {
     JButton someButton=  new JButton(“Some button”);
     // Other code…
     someButton.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent ae) {
        // Body of method executed when button pressed
     } });
   }
}
```

Creates anonymous object of anonymous inner class that implements ActionListener interface:
- Class has no name/reference
- Object has no name/reference



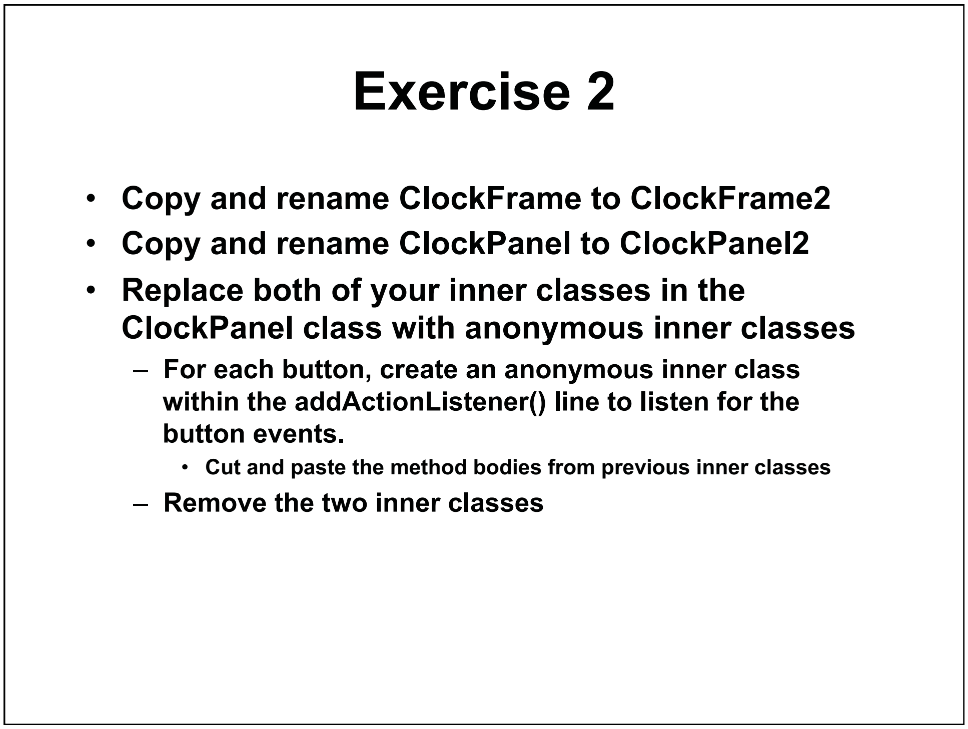
# Anonymous Inner Classes

- We appear to new an interface, which is illegal
  - We are actually creating a nameless class that will only have a single, nameless instance
- The new constructor call cannot have arguments*

```
addActionListener(
  new ActionListener() { . . . }
 );
 // We can't have an explicit constructor-why not?
```

  - The anonymous inner class has access to its enclosing class' data members and methods, so it doesn't need arguments.
- Anonymous inner classes are used when there are many event sources
  - We write one anonymous listener class per event source
  - This is a clear way to organize complex GUI code

* There is one obscure exception. Anonymous inner classes can extend a superclass. If so, they can have the superclass' arguments.



# Exercise 2

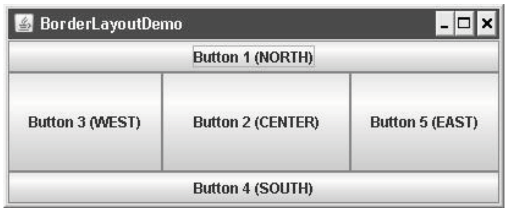
- Copy and rename ClockFrame to ClockFrame2
- Copy and rename ClockPanel to ClockPanel2
- Replace both of your inner classes in the ClockPanel class with anonymous inner classes
  - For each button, create an anonymous inner class within the addActionListener() line to listen for the button events.
    - Cut and paste the method bodies from previous inner classes
  - Remove the two inner classes

## Layout Management

- Layout management is the process of determining the size and location of a container's components.
  - Java containers do not handle their own layout. They delegate that task to their layout manager, an instance of another class.
  - Content panes and panels need layout (and a few others)
- Each layout manager enforces a different *layout policy*.
  - Layout proceeds bottom-up: it finds the size of individual elements, then sizes their containers until the frame or panel is sized

## `BorderLayout`

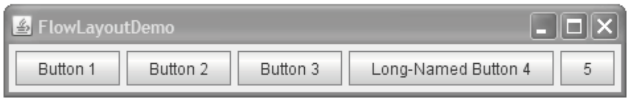
"A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. <u>Each region may contain no more than one component</u>, and is identified by a corresponding constant." - javadoc



`BorderLayout` is the default layout manager for `contentPane on JFrame`

# FlowLayout

"A flow layout arranges components in a left-to-right flow, much like lines of text in a paragraph. Flow layouts are typically used to arrange buttons in a panel. It will arrange buttons left to right until no more buttons fit on the same line. Each line is centered." - javadoc



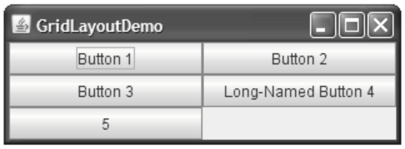
`FlowLayout` is the default layout manager for `JPanel`

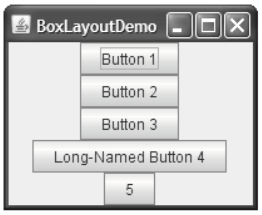
# Layout Management

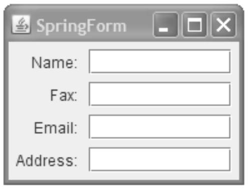
- If you do not like a container's default layout manager, you can change it.

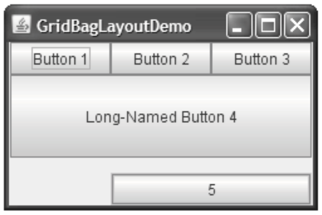
```
// Content pane has BorderLayout as default
Container contentPane = getContentPane();
contentPane.setLayout( new FlowLayout() );
// Flow Layout uses a 1 argument add() method
panel.add(button);    // Order matters
panel.add(label);

// JPanel has FlowLayout as default
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout( ));
// Border Layout uses a 2 argument add() method
// Can only add one component to each sector
panel.add(button, BorderLayout.NORTH);
panel.add(label, BorderLayout.SOUTH);
// If you want more than one component in a sector, put
// a panel on the sector and place components on it
```

# Other Layout Managers







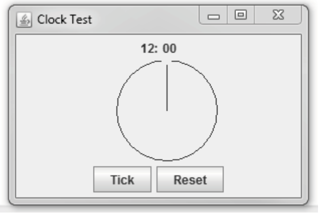


# Using Other Layout Managers

- To display a component in as much space as it can get
  - BorderLayout
- To display a few components in a row at their natural size
  - FlowLayout or BoxLayout
- To display a few components of same size in rows and columns
  - GridLayout
- To display a few components in row or column with varying amounts of space between them
  - BoxLayout
- To display aligned columns in a form with column of labels used to describe text fields in adjacent column
  - SpringLayout
- To display a complex GUI
  - GridBagLayout

# Exercise 3: Layout and Components

- Copy your previous solution to new classes
- Change the layout of the clock:



  - Create a new JPanel and place it at BorderLayout.NORTH
  - Add the hour and minute labels to the top panel
  - Change the y coordinates of the clock drawing in paintComponent() to allow room for the top panel

1.00 / 1.001 / 1.002 Introduction to Computers and Engineering Problem Solving
Spring 2012