

Advanced Programming

CSE 201

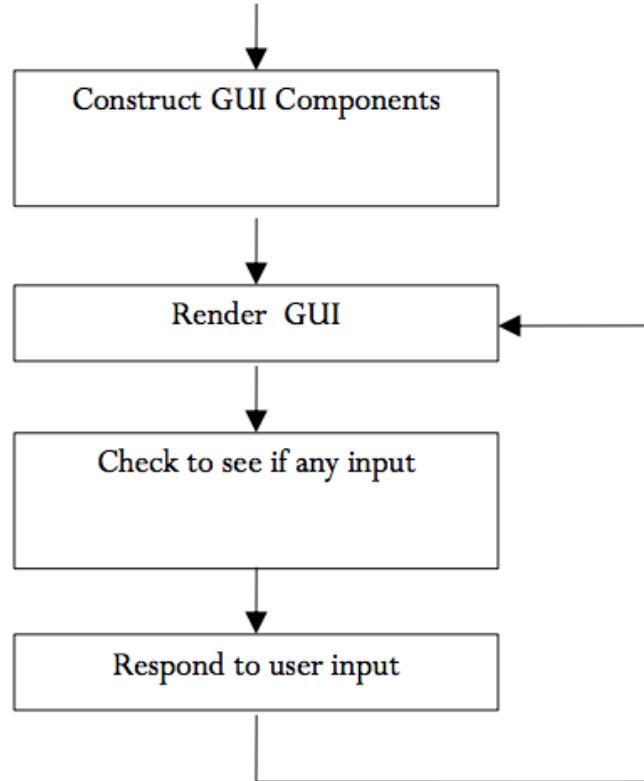
Instructor: Sambuddho

(Semester: Monsoon 2025)

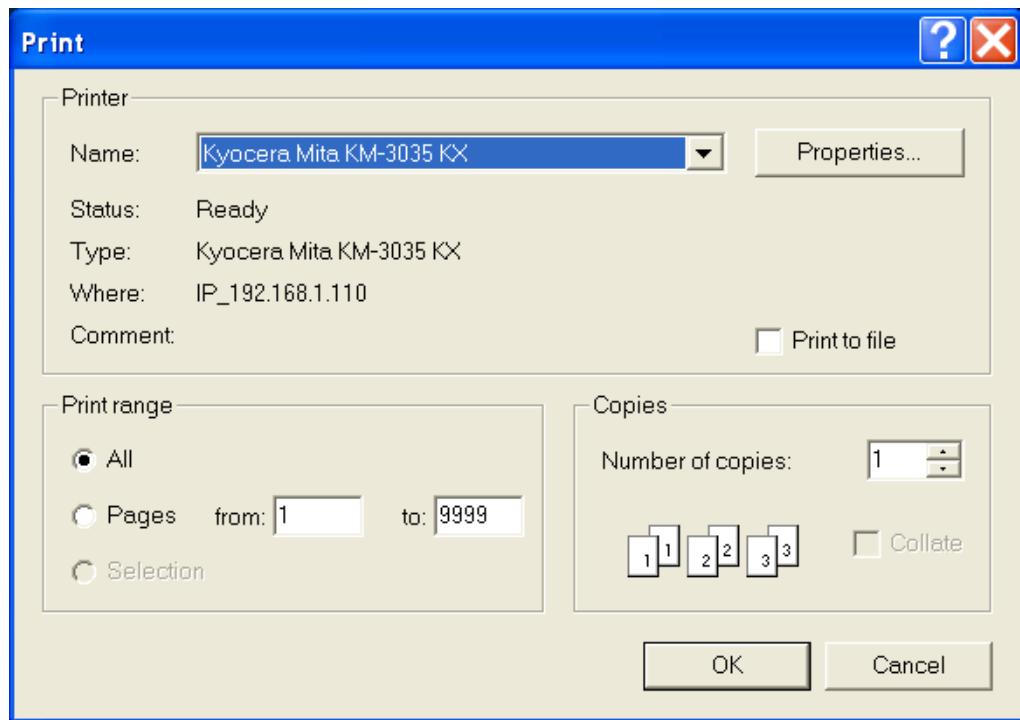
Week 8 - GUI/AWT

How do GUI Works?

- They loop and respond to events

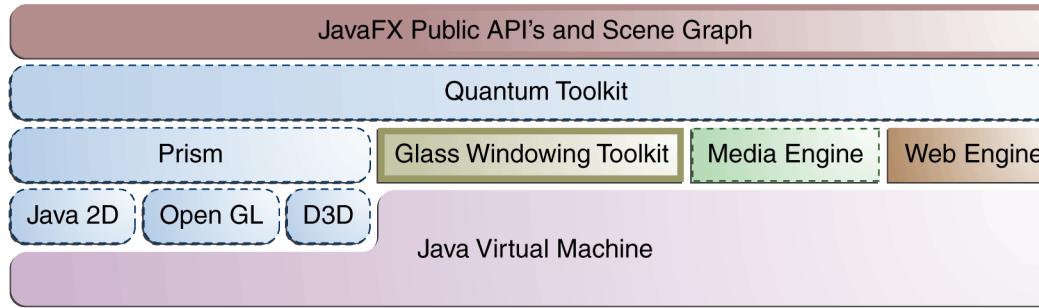


GUI Examples



Java Runtime High Level Architecture for GUIs

- You are here



Creating a Frame

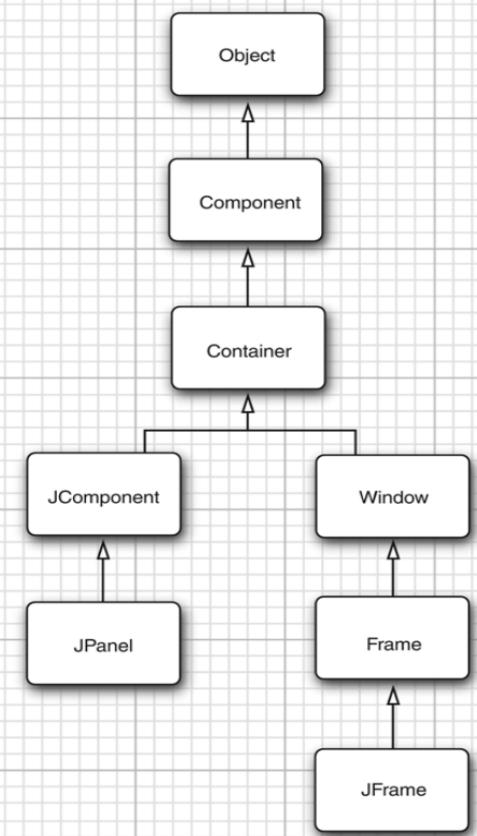
Swing JFrame

```
1 package simpleFrame;  
2  
3 import java.awt.*;  
4 import javax.swing.*;  
5  
6 /**  
7  * @version 1.34 2018-04-10  
8  * @author Cay Horstmann  
9  */  
10 public class SimpleFrameTest  
11 {  
12     public static void main(String[] args)  
13     {  
14         EventQueue.invokeLater(() ->  
15         {  
16             var frame = new SimpleFrame();  
17             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
18             frame.setVisible(true);  
19         });  
20     }  
21 }  
22 }
```



```
23 class SimpleFrame extends JFrame  
24 {  
25     private static final int DEFAULT_WIDTH = 300;  
26     private static final int DEFAULT_HEIGHT = 200;  
27  
28     public SimpleFrame()  
29     {  
30         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);  
31     }  
32 }
```

Frame Properties



java.awt.Component 1.0

- boolean isVisible()
- void setVisible(boolean b)

gets or sets the visible property. Components are initially visible, with the exception of top-level components such as JFrame.

- void setSize(int width, int height) 1.1
resizes the component to the specified width and height.

- void setLocation(int x, int y) 1.1

moves the component to a new location. The x and y coordinates use the coordinates of the container if the component is not a top-level component, or the coordinates of the screen if the component is top level (for example, a JFrame).

- void setBounds(int x, int y, int width, int height) 1.1
moves and resizes this component.

- Dimension getSize() 1.1
- void setSize(Dimension d) 1.1

gets or sets the size property of this component.

java.awt.Window 1.0

- void setLocationByPlatform(boolean b) 5

gets or sets the locationByPlatform property. When the property is set before this window is displayed, the platform picks a suitable location.

java.awt.Frame 1.0 (Continued)

- Image getIconImage()
- void setIconImage(Image image)

gets or sets the iconImage property that determines the icon for the frame. The windowing system may display the icon as part of the frame decoration or in other locations.

java.awt.Frame 1.0 (Continued)

- Image getIconImage()
- void setIconImage(Image image)

gets or sets the iconImage property that determines the icon for the frame. The windowing system may display the icon as part of the frame decoration or in other locations.

java.awt.Frame 1.0

- boolean isResizable()
- void setResizable(boolean b)

gets or sets the resizable property. When the property is set, the user can resize the frame.

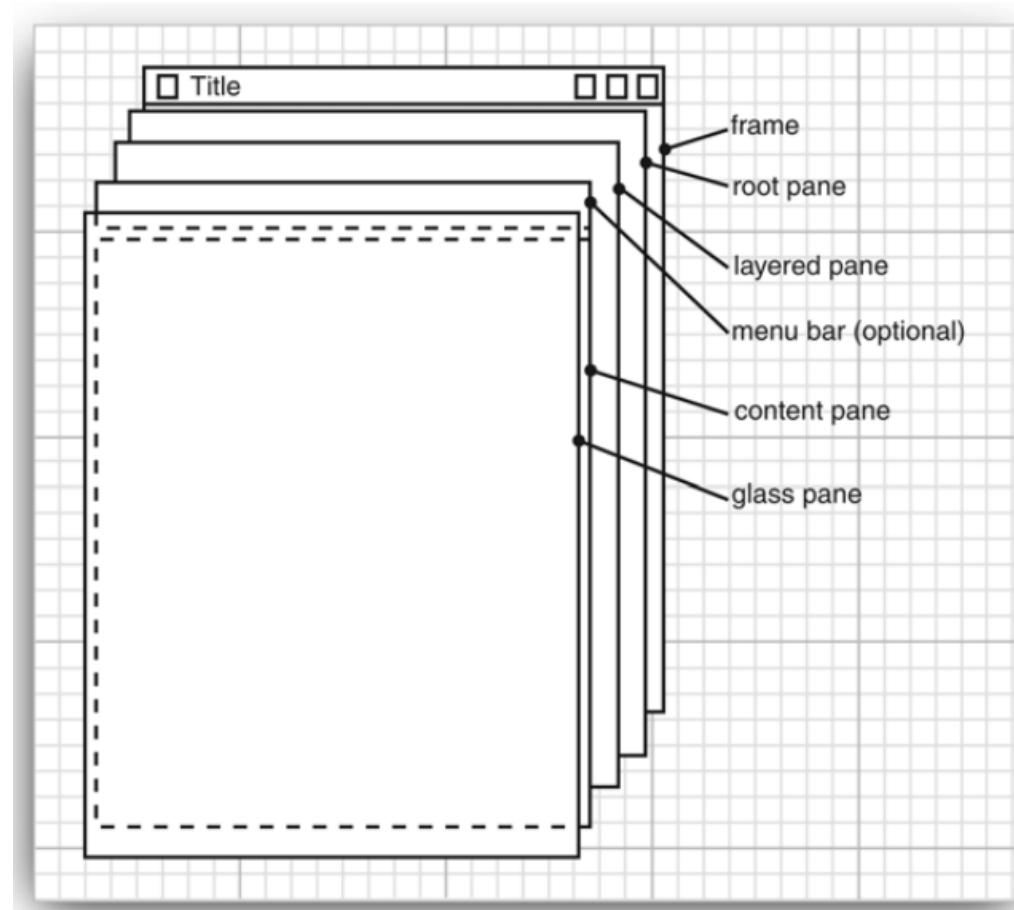
- String getTitle()
- void setTitle(String s)

gets or sets the title property that determines the text in the title bar for the frame.

Components

Content pane is the way to go -->

```
Component c = . . .;  
frame.add(c); // added to the content pane
```



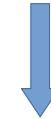
Components

```
class MyComponent extends JPanel
{
    public void paintComponent(Graphics g)
    {
        code for drawing
    }
}
```

```
public class NotHelloWorldComponent extends JPanel
{
    public static final int MESSAGE_X = 75;
    public static final int MESSAGE_Y = 100;

    public void paintComponent(Graphics g)
    {
        g.drawString("Not a Hello, World program", MESSAGE_X, MESSAGE_Y);
    }
    ...
}
```

```
public class NotHelloWorldComponent extends JPanel
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;
    ...
    public Dimension getPreferredSize()
    {
        return new Dimension(DEFAULT_WIDTH, DEFAULT_HEIGHT);
    }
}
```



```
class NotHelloWorldFrame extends JFrame
{
    public NotHelloWorldFrame()
    {
        add(new NotHelloWorldComponent());
        pack();
    }
}
```

A Basic GUI Program Using AWT

```
1 package notHelloWorld;
2
3 import javax.swing.*;
4 import java.awt.*;
5
6 /**
7  * @version 1.34 2018-04-10
8  * @author Cay Horstmann
9  */
10 public class NotHelloWorld
11 {
12     public static void main(String[] args)
13     {
14         EventQueue.invokeLater(() ->
15         {
16             var frame = new NotHelloWorldFrame();
17             frame.setTitle("NotHelloWorld");
18             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19             frame.setVisible(true);
20         });
21     }
22 }
23
24 /**
25  * A frame that contains a message panel.
26  */
27 class NotHelloWorldFrame extends JFrame
28 {
29     public NotHelloWorldFrame()
30     {
31         add(new NotHelloWorldComponent());
32         pack();
33     }
34 }
35
36 /**
37  * A component that displays a message.
38  */
39 class NotHelloWorldComponent extends JComponent
40 {
41     public static final int MESSAGE_X = 75;
42     public static final int MESSAGE_Y = 100;
43
44     private static final int DEFAULT_WIDTH = 300;
45     private static final int DEFAULT_HEIGHT = 200;
46
47     public void paintComponent(Graphics g)
48     {
49         g.drawString("Not a Hello, World program", MESSAGE_X, MESSAGE_Y);
50     }
51
52     public Dimension getPreferredSize()
53     {
54         return new Dimension(DEFAULT_WIDTH, DEFAULT_HEIGHT);
55     }
56 }
```

2D Objects

Line2D, Rectangle2D, Ellipse2D etc. ← all implementations of Shape interface.

To draw a shape:

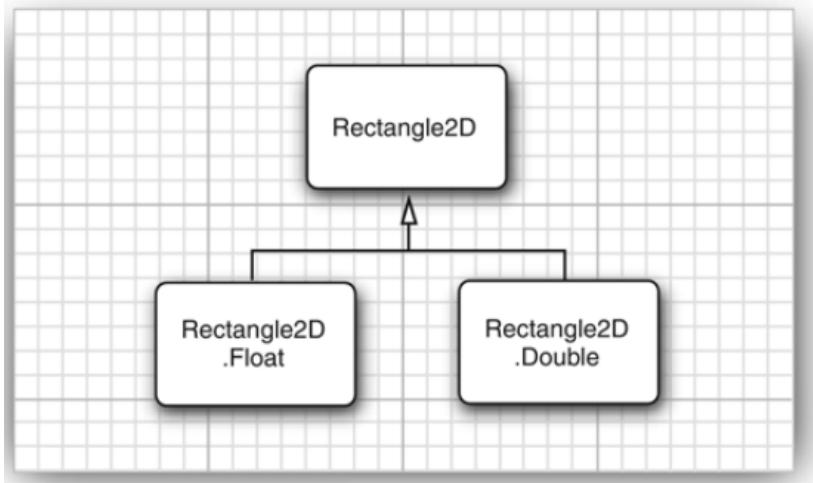
1. Create an object of the class that implements the Shape interface.
2. Then pass on the object to Graphics2D.draw().

```
public void paintComponent(Graphics g)
{
    Graphics2D g2 = (Graphics2D) g;
    Rectangle2D rect = . . . ;
    g2.draw(rect);

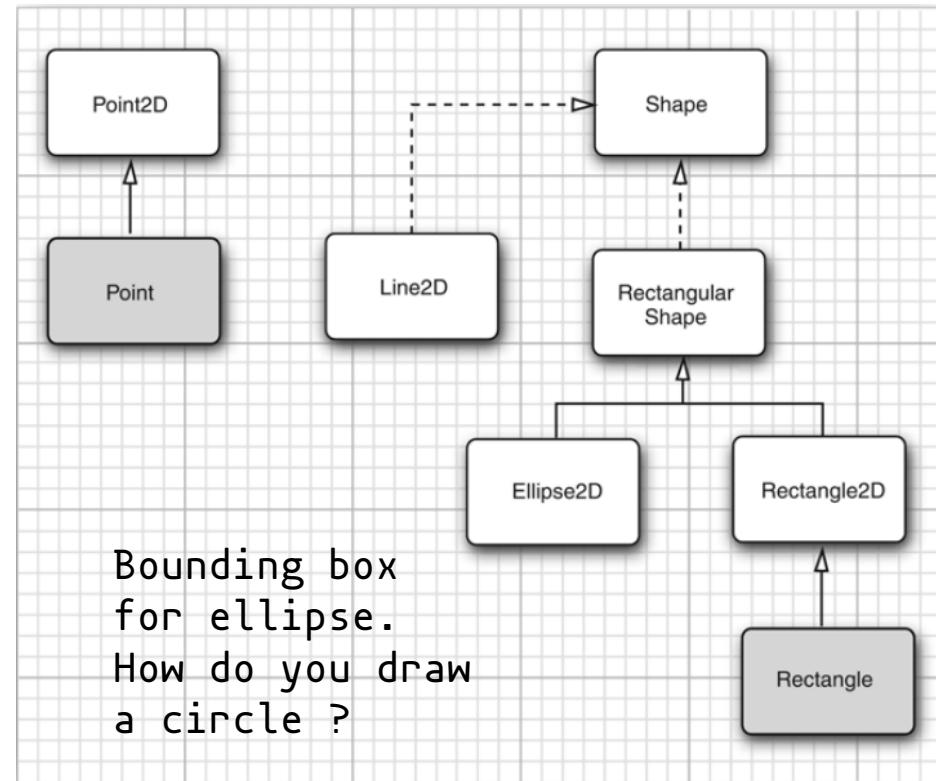
}
```

2D Objects

Shapes are of two types - based on floating point or double pixels.



```
var floatRect = new Rectangle2D.Float(10.0F, 25.0F, 22.5F, 20.0F);
var doubleRect = new Rectangle2D.Double(10.0, 25.0, 22.5, 20.0);
```



2D Objects

```
1 package draw;
2
3 import java.awt.*;
4 import java.awt.geom.*;
5 import javax.swing.*;
6
7 /**
8 * @version 1.34 2018-04-10
9 * @author Cay Horstmann
10 */
11 public class DrawTest
12 {
13     public static void main(String[] args)
14     {
15         EventQueue.invokeLater(() ->
16         {
17             var frame = new DrawFrame();
18             frame.setTitle("DrawTest");
19             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20             frame.setVisible(true);
21         });
22     }
23 }
24
25 /**
26 * A frame that contains a panel with drawings.
27 */
28 class DrawFrame extends JFrame
29 {
30     public DrawFrame()
31     {
32         add(new DrawComponent());
33         pack();
34     }
35 }
36
```

```
37     /**
38      * A component that displays rectangles and ellipses.
39      */
40     class DrawComponent extends JComponent
41     {
42         private static final int DEFAULT_WIDTH = 400;
43         private static final int DEFAULT_HEIGHT = 400;
44
45         public void paintComponent(Graphics g)
46         {
47             var g2 = (Graphics2D) g;
48
49             // draw a rectangle
50
51             double leftX = 100;
52             double topY = 100;
53             double width = 200;
54             double height = 150;
55
56             var rect = new Rectangle2D.Double(leftX, topY, width, height);
57             g2.draw(rect);
58
59             // draw the enclosed ellipse
60
61             var ellipse = new Ellipse2D.Double();
62             ellipse setFrame(rect);
63             g2.draw(ellipse);
64
65             // draw a diagonal line
66
67             g2.draw(new Line2D.Double(leftX, topY, leftX + width, topY + height));
68
69             // draw a circle with the same center
70
71             double centerX = rect.getCenterX();
72             double centerY = rect.getCenterY();
73             double radius = 150;
74
75             var circle = new Ellipse2D.Double();
76             circle.setFrameFromCenter(centerX, centerY, centerX + radius, centerY + radius);
77             g2.draw(circle);
78 }
```

2D Objects - Colors

```
Rectangle2D rect = . . .;  
g2.setPaint(Color.RED);  
g2.fill(rect); // fills rect with red
```

`java.awt.Color` - BLACK,
BLUE, CYAN, DARK_GRAY,
GREEN, GRAY, LIGHT_GRAY,
MAGENTA, ORANGE, PINK, RED,
WHITE, YELLOW

```
var component = new MyComponent();  
component.setBackground(Color.PINK);
```

java.awt.Color 1.0

- `Color(int r, int g, int b)`

creates a color object with the given red, green, and blue components between 0 and 255.

java.awt.Graphics2D 1.2

- `Paint getPaint()`
- `void setPaint(Paint p)`

gets or sets the paint property of this graphics context. The `Color` class implements the `Paint` interface. Therefore, you can use this method to set the paint attribute to a solid color.

- `void fill(Shape s)`
fills the shape with the current paint.

java.awt.Component 1.0

- `Color getForeground()`
- `Color getBackground()`
- `void setForeground(Color c)`
- `void setBackground(Color c)`

gets or sets the foreground or background color.

2D Objects - Fonts

Font face names available - SansSerif, Serif, Monospaced, Dialog, DialogInput

```
import java.awt.*;  
  
public class ListFonts  
{  
    public static void main(String[] args)  
    {  
        String[] fontNames = GraphicsEnvironment  
            .getLocalGraphicsEnvironment()  
            .getAvailableFontFamilyNames();  
        for (String fontName : fontNames)  
            System.out.println(fontName);  
    }  
}
```

Set font for a string



```
var sansbold14 = new Font("SansSerif", Font.BOLD, 14);  
g2.setFont(sansbold14);  
var message = "Hello, World!";  
g2.drawString(message, 75, 100);
```

2D Objects - Images

```
Image image = new ImageIcon(filename).getImage();  
  
public void paintComponent(Graphics g)  
{  
    . . .  
    g.drawImage(image, x, y, null);  
}
```

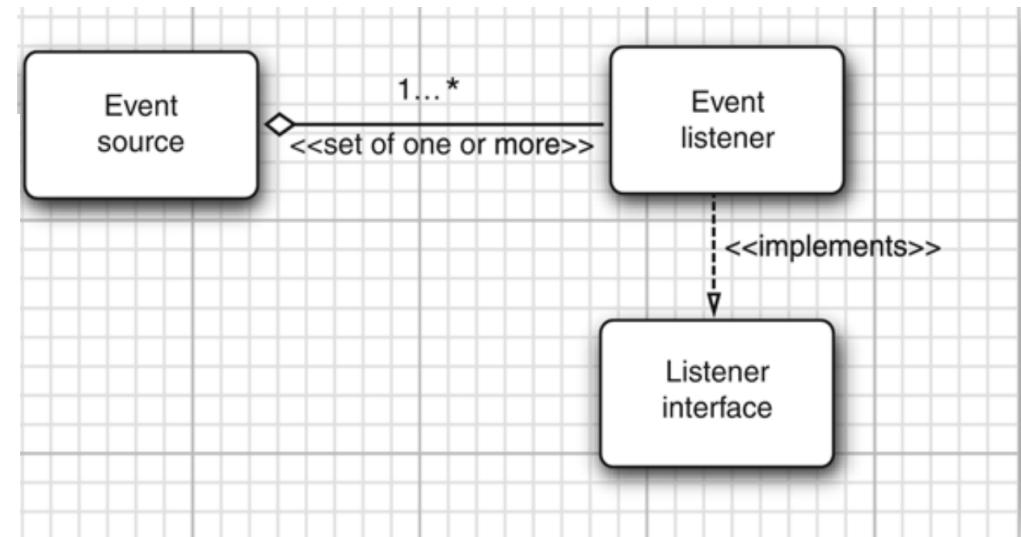
java.awt.Graphics 1.0

- boolean drawImage(Image img, int x, int y, ImageObserver observer)
- boolean drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)
draws an unscaled or scaled image. Note: This call may return before the image is drawn. The `imageObserver` object is notified of the rendering progress. This was a useful feature in the distant past. Nowadays, just pass a `null` observer.
- void copyArea(int x, int y, int width, int height, int dx, int dy)
copies an area of the screen. The `dx` and `dy` parameters are the distance from the source area to the target area.

Event Handling

- All events are converted to AWT event objects belonging to `java.util.EventObject` (subclasses e.g. `ActionEvent`, `WindowEvent` etc.).

```
class MyListener implements ActionListener
{
    ...
    public void actionPerformed(ActionEvent event)
    {
        // reaction to button click goes here
        ...
    }
}
```



- JButton object creates an ActionEvent and calls `listener.actionPerformed(event)`

Handling a Button Click Event

```
var yellowButton = new JButton("Yellow");
```

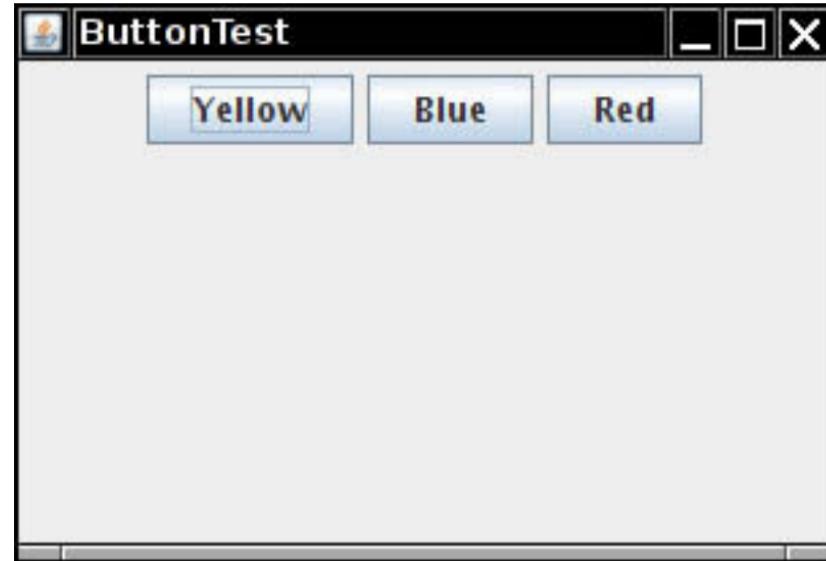
```
var blueButton = new JButton("Blue");
```

```
var redButton = new JButton("Red");
```

```
buttonPanel.add(yellowButton);
```

```
buttonPanel.add(blueButton);
```

```
buttonPanel.add(redButton);
```



Handling a Button Click Event

```
class ColorAction implements ActionListener  
{  
    private Color backgroundColor;  
  
    public ColorAction(Color c)  
    {  
        backgroundColor = c;  
    }  
  
    public void actionPerformed(ActionEvent event)  
    {  
        // set panel background color  
        . . .  
    }  
}
```

```
var yellowAction = new ColorAction(Color.YELLOW);  
var blueAction = new ColorAction(Color.BLUE);  
var redAction = new ColorAction(Color.RED);  
  
yellowButton.addActionListener(yellowAction);  
blueButton.addActionListener(blueAction);  
redButton.addActionListener(redAction);
```

Handling a Button Click Event

```
1 package button;
2
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 /**
8 * A frame with a button panel.
9 */
10 public class ButtonFrame extends JFrame
11 {
12     private JPanel buttonPanel;
13     private static final int DEFAULT_WIDTH = 300;
14     private static final int DEFAULT_HEIGHT = 200;
15
16     public ButtonFrame()
17     {
18         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
19
20         // create buttons
21         var yellowButton = new JButton("Yellow");
22         var blueButton = new JButton("Blue");
23         var redButton = new JButton("Red");
24
25         buttonPanel = new JPanel();
26
27         // add buttons to panel
28         buttonPanel.add(yellowButton);
29         buttonPanel.add(blueButton);
30         buttonPanel.add(redButton);
31
32         // add panel to frame
33         add(buttonPanel);
34
35         // add panel to frame
36         add(buttonPanel);
37
38         // create button actions
39         var yellowAction = new ColorAction(Color.YELLOW);
40         var blueAction = new ColorAction(Color.BLUE);
41         var redAction = new ColorAction(Color.RED);
42
43         // associate actions with buttons
44         yellowButton.addActionListener(yellowAction);
45         blueButton.addActionListener(blueAction);
46         redButton.addActionListener(redAction);
47
48     }
49
50     /**
51      * An action listener that sets the panel's background color.
52      */
53     private class ColorAction implements ActionListener
54     {
55         private Color backgroundColor;
56
57         public ColorAction(Color c)
58         {
59             backgroundColor = c;
60         }
61
62         public void actionPerformed(ActionEvent event)
63         {
64             buttonPanel.setBackground(backgroundColor);
65         }
66     }
67 }
```

Handling a Button Click Event - Single Event Handler for All Types

```
public void makeButton(String name, Color backgroundColor)
{
    var button = new JButton(name);
    buttonPanel.add(button);
    button.addActionListener(event ->
        buttonPanel.setBackground(backgroundColor));
}
```

```
makeButton("yellow", Color.YELLOW);
makeButton("blue", Color.BLUE);
makeButton("red", Color.RED);
```

Adapter Classes

- Used to monitor window open/close events.
- An appropriate listener must be required to catch such events.

```
public interface WindowListener
{
    void windowOpened(WindowEvent e);
    void windowClosing(WindowEvent e);
    void windowClosed(WindowEvent e);
    void windowIconified(WindowEvent e);
    void windowDeiconified(WindowEvent e);
    void windowActivated(WindowEvent e);
    void windowDeactivated(WindowEvent e);
}
```

Requires implementing all the methods.

Adapter Classes

- Alternative - extend and use *adapter* classes
- Adapter classes have dummy do-nothing methods for all the rest of the operations.

```
class Terminator extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        if (user agrees)
            System.exit(0);
    }
}
```

```
var listener = new Terminator();
frame.addWindowListener(listener);
```

Mouse Events

- Three main listener methods - `mousePressed`, `mouseClicked` and `mouseReleased`.
- `MouseEvent` object sent to the event handler class (adapter or listener implementation).

`java.awt.event.MouseEvent 1.1`

- `int getX()`
- `int getY()`
- `Point getPoint()`

returns the *x* (horizontal) and *y* (vertical) coordinates of the point where the event happened, measured from the top left corner of the component that is the event source.

- `int getClickCount()`

returns the number of consecutive mouse clicks associated with this event. (The time interval for what constitutes “consecutive” is system-dependent.)

```
private class MouseHandler extends MouseAdapter
{
    public void mousePressed(MouseEvent event)
    {
        // add a new square if the cursor isn't inside a square
        current = find(event.getPoint());
        if (current == null) add(event.getPoint());
    }

    public void mouseClicked(MouseEvent event)
    {
        // remove the current square if double clicked
        current = find(event.getPoint());
        if (current != null && event.getClickCount() >= 2) remove(current);
    }
}
```

Mouse Event

```
package mouse;

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

/**
 * A component with mouse operations for adding and removing squares.
 */
public class MouseComponent extends JComponent
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;

    private static final int SIDELENGTH = 10;
    private ArrayList<Rectangle2D> squares;
    private Rectangle2D current; // the square containing the mouse cursor

    public MouseComponent()
    {
        squares = new ArrayList<>();
        current = null;
    }

    addMouseListener(new MouseHandler());
    addMouseMotionListener(new MouseMotionHandler());
}

public Dimension getPreferredSize()
{
    return new Dimension(DEFAULT_WIDTH, DEFAULT_HEIGHT);
}

public void paintComponent(Graphics g)
{
    var g2 = (Graphics2D) g;

    // draw all squares
    for (Rectangle2D r : squares)
        g2.draw(r);
}

/**
 * Finds the first square containing a point.
 * @param p a point
 * @return the first square that contains p
 */
public Rectangle2D find(Point2D p)
{
    for (Rectangle2D r : squares)
    {
        if (r.contains(p)) return r;
    }
    return null;
}

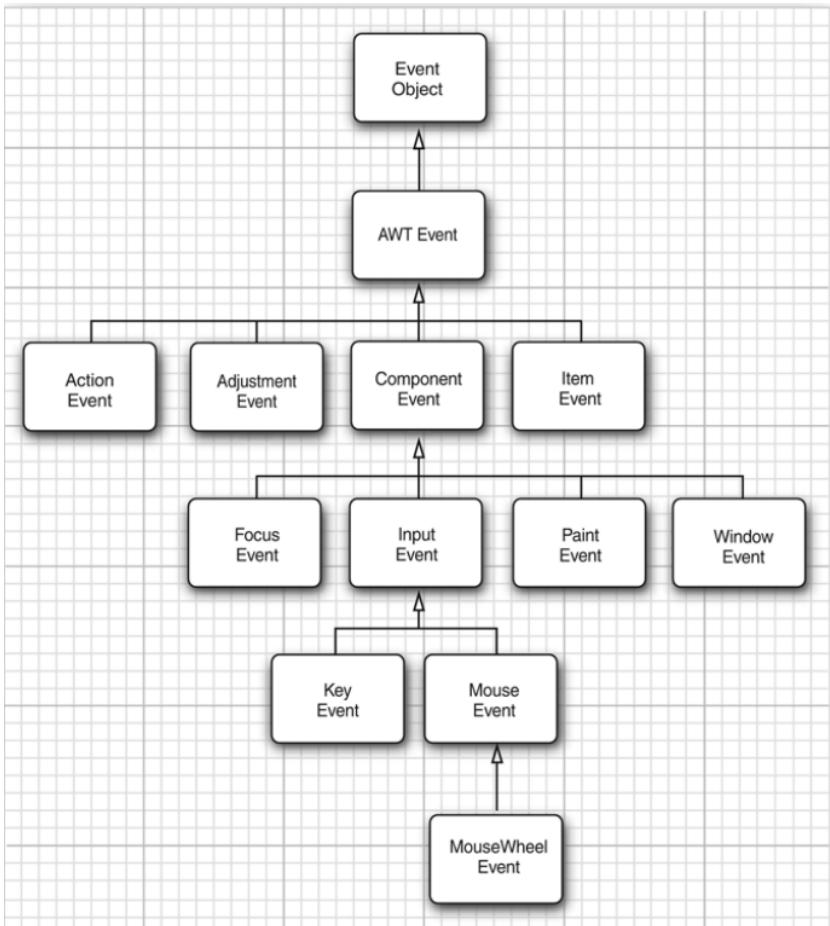
/**
 * Adds a square to the collection.
 * @param p the center of the square
 */
public void add(Point2D p)
{
    double x = p.getX();
    double y = p.getY();

    current = new Rectangle2D.Double(x - SIDELENGTH / 2, y - SIDELENGTH / 2,
                                    SIDELENGTH, SIDELENGTH);
    squares.add(current);
    repaint();
}
```

Mouse Event

```
/**  
 * Removes a square from the collection.  
 * @param s the square to remove  
 */  
public void remove(Rectangle2D s)  
{  
    if (s == null) return;  
    if (s == current) current = null;  
    squares.remove(s);  
    repaint();  
}  
  
private class MouseHandler extends MouseAdapter  
{  
    public void mousePressed(MouseEvent event)  
    {  
        // add a new square if the cursor isn't inside a square  
        current = find(event.getPoint());  
        if (current == null) add(event.getPoint());  
    }  
  
    public void mouseClicked(MouseEvent event)  
    {  
        // remove the current square if double clicked  
        current = find(event.getPoint());  
        if (current != null && event.getClickCount() >= 2) remove(current);  
    }  
}  
  
private class MouseMotionHandler implements MouseMotionListener  
{  
    public void mouseMoved(MouseEvent event)  
    {  
        // set the mouse cursor to cross hairs if it is inside a rectangle  
  
        if (find(event.getPoint()) == null) setCursor(Cursor.getDefaultCursor());  
        else setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));  
    }  
  
    public void mouseDragged(MouseEvent event)  
    {  
        if (current != null)  
        {  
            int x = event.getX();  
            int y = event.getY();  
  
            // drag the current rectangle to center it at (x, y)  
            current setFrame(x - SIDELENGTH / 2, y - SIDELENGTH / 2, SIDELENGTH, SIDELENGTH);  
            repaint();  
        }  
    }  
}
```

AWT Hierarchy



Interface	Methods	Parameter/Accessors	Events Generated By
ActionListener	actionPerformed	ActionEvent <ul style="list-style-type: none">• getActionCommand• getModifiers	AbstractButton JComboBox JTextField Timer
AdjustmentListener	adjustmentValueChanged	AdjustmentEvent <ul style="list-style-type: none">• getAdjustable• getAdjustmentType	JScrollbar
Interface	Methods	Parameter/Accessors	Events Generated By
ItemListener	itemStateChanged	ItemEvent <ul style="list-style-type: none">• getItem• getItemSelectable• getStateChange	AbstractButton JComboBox
FocusListener	focusGained focusLost	FocusEvent <ul style="list-style-type: none">• isTemporary	Component
KeyListener	keyPressed keyReleased keyTyped	KeyEvent <ul style="list-style-type: none">• getKeyChar• getKeyCode• getKeyModifiersText• getKeyText• isActionKey	Component
MouseListener	mousePressed mouseReleased mouseEntered mouseExited mouseClicked	MouseEvent <ul style="list-style-type: none">• getClickCount• getX• getY• getPoint• translatePoint	Component
MouseMotionListener	mouseDragged mouseMoved	MouseEvent	Component

AWT Hierarchy

MouseWheelListener	mouseWheelMoved	MouseWheelEvent	Component
		<ul style="list-style-type: none">• getWheelRotation• getScrollAmount	
WindowListener	windowClosing windowOpened windowIconified windowDeiconified windowClosed windowActivated windowDeactivated	WindowEvent	Window
		<ul style="list-style-type: none">• getWindow	

Interface	Methods	Parameter/Accessors	Events Generated By
WindowFocusListener	windowGainedFocus windowLostFocus	WindowEvent	Window
		<ul style="list-style-type: none">• getOppositeWindow	
WindowStateListener	windowStateChanged	WindowEvent	Window
		<ul style="list-style-type: none">• getOldState• getNewState	