Graphics Programming

B.Tech. (IT), Sem-5,
Core Java Technology (CJT)

Dharmsinh Desai University
Prof. (Dr.) H B Prajapati

Introduction to GUI

- Desktop Applications in Java can be of two types:
  1. Command-line Interface (CLI) based
  2. Graphical User Interface (GUI) based
- CLI based applications allow text based input and output (Using keyboard)
- GUI based applications allow input using keyboard and mouse and output in graphics (picture) form.

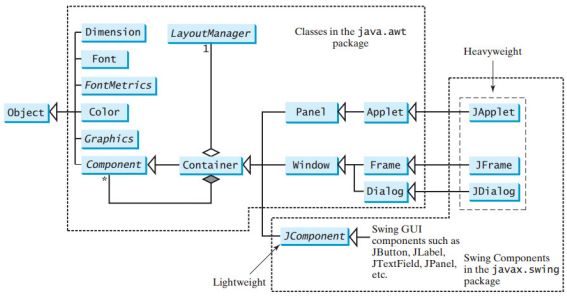
GUI applications in Java

- Java supports two libraries for GUI based applications
  1. Abstract Window Toolkit (AWT)
  2. Swing
- AWT (java.awt package) provides platform-independent interfaces/classes to render platform specific GUI
  - AWT relies on the underlying operating system on a specific platform to represent its GUI components (i.e components in AWT are called Heavyweight)
  - Same program will provide different GUI on different platforms
- Swing (javax.swing package) provides same GUI on all platforms
  - Swing components are lightweight and are not dependent on the underlying Windowing system

Components and Containers

- A building block of a graphical user interface is a GUI component. Can be graphical elements like buttons, scrollbars, lists, textfields, etc.
  - In AWT, these components are instances of the respective Component classes.
- Components cannot exist alone; they are found within containers.
  - Container is a special component that can hold another component or container.
  - In AWT, all containers are objects of class Container or one of its subtypes.

AWT and Class Hierarchy



AWT and Class Hierarchy

- AWT classes use concepts of classes, inheritance, and interfaces to get maximum code reusability.
- Main classes:
- Component:
  - It is a superclass of all AWT UI classes. It is an abstract class.
- Container:
  - It is used to group components.
  - It is derived from Component
  - Container has one layout manager to position various components
  - Window,Panel,Applet,Frame, and
  - Dialogare the container classes for AWT components

AWT and Class Hierarchy

- Window
  - It is used to create a top-level window of a GUI application
  - Window's subclasses such as Frame and Dialog are used.
- Frame
  - Starting point of GUI
  - It contain other UI components
- Dialog
  - Pop-up window for temporary operation
  - Used to get additional information from the user

7

AWT and Class Hierarchy

- Panel
  - It is invisible container that can hold UI components
  - A Panel can be placed in a frame or applet
- Applet
  - It is a subclass of Panel
- Which class to use Frame or Applet?
  - Frame is used when writing desktop application
  - Applet is used when writing web browser based application
- Graphics
  - It is an abstract class that provides the methods for drawing text, lines, and simple shapes.

8

AWT and Class Hierarchy

- Color
  - It encapsulates color (RGB) information.
  - Generally used for setting foreground or background color for GUI.
- Font
  - It is used when we write(draw) text on GUI.
  - We can specify
    - Font type (e.g., Times New Roman),
    - Style (Bold or Italic)
    - Size (12 or 16)

9

AWT and Class Hierarchy

- FontMetrics
  - It is an abstract class
  - It is used to get properties (metrics) of font.

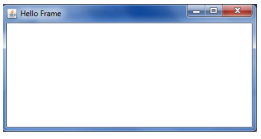
10

Smallest GUI program

```
import java.awt.Frame;
class MyFrame{
   public static void main(String[] args){
        Frame f=new Frame("Hello Frame");
        f.setSize(400,200);
        f.setVisible(true);
   }
}
```

11

Smallest GUI program



- How to close this window?
  - Press Control+C on Windows and Control+Z on Linux

12

Frame class

- Constructors:
  - Frame()
    - Frame without title
  - Frame(String title)
    - Frame with title
- Methods:
  - setSize(int width, int height)
  - setVisible(boolean visible)

13

Concept of Event Driven Programming

- CLI based program executed in a procedural order.
- We can use loop and decision constructs to control the flow of execution.
- But, the program dictates the flow of execution

- GUI based program executes based on events.
- Program code is executed upon activation of events.

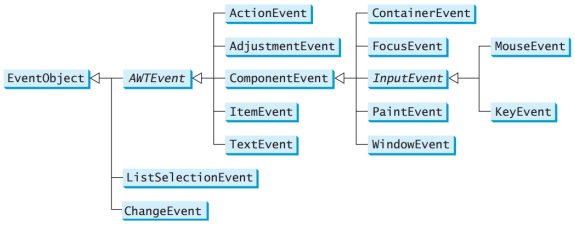
14

Event and Event Source

- Event
  - A signal to the program that something has happened.
  - Events are triggered by
    - By external user actions, such as mouse movements, button clicks, and keystrokes, or
    - By internal program activities, such as a timer.
  - The program can choose to respond to or ignore an event.
- Event Source
  - The component that creates an event and fires it is called the source object or source component. E.g., Button

15

Class hierarchies of Event classes



- The root class of the event classes is java.util.EventObject.

16

Members of Event objects

- An event object contains properties related to the event.
- We can get the source object (generator of the event) using getSource() method of EventObject class.

17

User Action, Source, and Event

- User action, Source Object, and Event type

User Action	Source Object	Event type generated
Clicked on a button	Button	ActionEvent
Changed text	TextComponent	TextEvent
Pressed return on a text field	TextField	ActionEvent
Double-clicked on a list item	List	ActionEvent
Selected or deselected an item with a single click	List	ItemEvent
Selected or deselected an item	Choice	ItemEvent
Selected or deselected an item	Choicebox	ItemEvent

18

User Action, Source, and Event

- User action, Source Object, and Event type

User Action	Source Object	Event type generated
Selected a menu item	MenuItem	ActionEvent
Moved the scroll bar	Scrollbar	AdjustmentEvent
Window opened, closed, iconified, deiconified, or closing	Window	WindowEvent
Added or removed from the container	Container	ContainerEvent
Component moved, resized, hidden, or shown	Component	ComponentEvent

19

User Action, Source, and Event

- User action, Source Object, and Event type

User Action	Source Object	Event type generated
Component gained or lost focus	Component	FocusEvent
Key released or pressed	Component	KeyEvent
Mouse pressed, released, clicked, entered, or exited	Component	MouseEvent
Mouse moved or dragged	Component	MouseEvent

20

Event registration, listening, and handling: Java's Event Delegation Model

- Java uses a delegation-based model for event handling
  - Source: A source object fires an event
  - Event handler: The object, responsible for performing the task when an event occurs is called the event handler.
  - Event information: Contains status of component and information related to user action.
- Working
  - A source object fires an event.
  - An object interested in the event handles it.
  - There may be more than one event handlers for one single event generated.
  - the source object passes event information to the listener in event object.

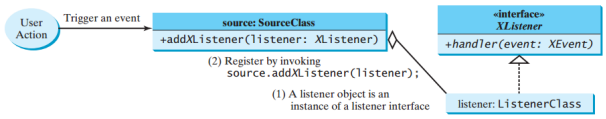
21

Event registration, listening, and handling: Java's Event Delegation Model

- But, how does the source object knows which listener(s) are interested in the events?
  - The listener object must be registered by the source object.
  - Analogy: We register/subscribe for news paper. When an event occurs (daily newspaper is printed), all registered customers are automatically given the newspaper (event information)
  - For example
    - Owner of Gujarat Samachar publication is Source.
    - Gujarat Samachar is Event Information.
    - Customers are listener.

22

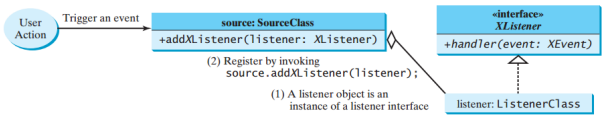
Event registration, listening, and handling: Java's Event Delegation Model



- Step 1:
- The listener object must be an instance of the corresponding event-listener interface and has to implement the methods for processing the event.
- The listener interface is usually named XListener for XEvent.
- For example, listener interface for ActionEvent is ActionListener.
- Exception: for MouseMotionListener event is MouseEvent.

23

Event registration, listening, and handling: Java's Event Delegation Model



- Step 2:
- The listener object must be registered by the source object.
- In general, the method is named addXListenerfor XEvent.
- For ActionEvent, the method is addActionListener.

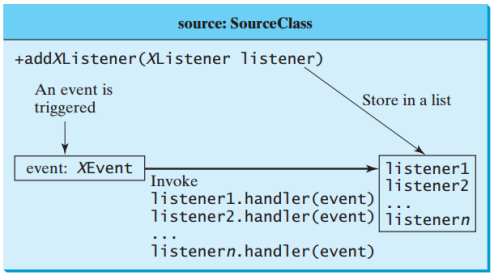
24

Event registration, listening, and handling: Java's Event Delegation Model

- How does source handle multiple listeners?
  - Source object maintains a list of all registered listeners.
  - Notifies all the registered listeners when the event occurs.
- One source may generate different types of events depending upon user action.
  - A source object may fire several types of events.
  - For each supported event, the source maintains a list of registered listeners
  - Notifies all registered listeners by invoking the handler method of the listener object to respond to the event

25

Event registration, listening, and handling: Java's Event Delegation Model



26

Event registration, listening, and handling: Java's Event Delegation Model

- How to use event delegation model?

```
//source
Button btClose=new Button("Close");
//listener object
ActionListener closeListener = new CloseListener();
// source register listener
btClose.addActionListener(closeListener);
```

27

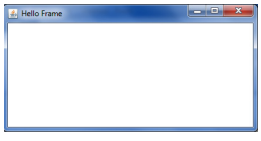
Event registration, listening, and handling: Java's Event Delegation Model

- We can implement the listener interface in the same class that contains the source object.
- If our MyGUIApplication class contains Button (source) object and MyGUIApplication implements ActionListener interface, then
- How to use event delegation model?

```
//source
Button btClose=new Button("Close");
// source register listener
btClose.addActionListener(this);
```

28

Example: Closing our application window



- We closed our application using Ctrl+C.
- We want to close our application, when we click on X (close) button.

29

Example: Closing our application window

```
import java.awt.Frame;
class MyGUIApplication extends Frame {
    public MyGUIApplication(String title){
        super(title);
        setVisible(true);
        setSize(600,400);
        addWindowListener(new MyWindowListener());
    }
    public static void main(String[] args){
        new MyGUIApplication("My Frame");
        System.out.println("End of main");
    }
}
```

30

Example: Closing our application window

```
import java.awt.event.WindowListener;
import java.awt.event.WindowEvent;
class MyWindowListener implements WindowListener{
    public void windowClosing(WindowEvent e){
        System.out.println("Window Closing");
        System.exit(0);
    }
    public void windowClosed(WindowEvent e){
        System.out.println("Window Closed");
    }
```

31

Example: Closing our application window

```
    public void windowOpened(WindowEvent e){
        System.out.println("Window Opened");
    }
    public void windowIconified(WindowEvent e){
        System.out.println("Window Iconfied");
    }
    public void windowDeiconified(WindowEvent e){
        System.out.println("Window Deiconfied");
    }
```

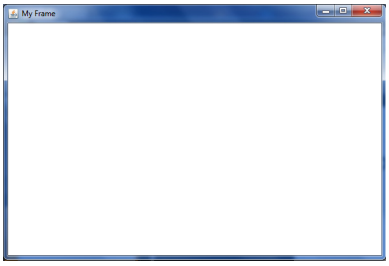
32

Example: Closing our application window

```
    public void windowActivated(WindowEvent e){
        System.out.println("Window Activated");
    }
    public void windowDeactivated(WindowEvent e){
        System.out.println("Window Deactivated");
    }
}
```

33

Example: Closing our application window

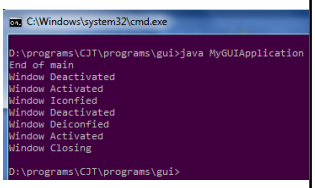


34

Example: Closing our application window

Steps:

1. Run application
2. Activate command prompt
3. Activate our application
4. Iconify (minimize) our application
5. De-iconify our application
6. Click on X (close) button



35

Example: Closing our application window – another way

```
import java.awt.*;
import java.awt.event.*;
public class MyFrameWithExitHandling extends Frame
   implements WindowListener{
    public MyFrameWithExitHandling(String title){
        super(title);
        setVisible(true);
        setSize(400,400);
        addWindowListener(this);
    }
```

36

Example: Closing our application window – another way

```
public static void main(String[] args){
        new MyFrameWithExitHandling("My Frame");
        System.out.println("End of main");
    }
    public void windowClosing(WindowEvent e){
        System.out.println("Window Closing");
        dispose();
    }
    public void windowClosed(WindowEvent e){
        System.out.println("Window Closed");
        System.exit(0);
    }
```

37

Example: Closing our application window – another way

```
public void windowOpened(WindowEvent e){
        System.out.println("Window Opened");
    }
    public void windowIconified(WindowEvent e){
        System.out.println("Window Iconfied");
    }
    public void windowDeiconified(WindowEvent e){
        System.out.println("Window Deiconfied");
    }
```

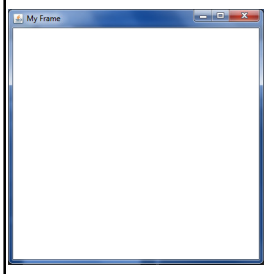
38

Example: Closing our application window – another way

```
    public void windowActivated(WindowEvent e){
        System.out.println("Window Activated");
    }
    public void windowDeactivated(WindowEvent e){
        System.out.println("Window Deactivated");
    }
}
```

39

Example: Closing our application window – another way



- The dispose() method disposes the frame object when the object is no longer needed.
- It causes a call to windowClosed() event handler.
- How to handle termination of our application?
  - Call dispose() on frame object inside windowClosing() event handler
  - Then, call System.exit(0) in windowClosed() event handler.

40

Example: Closing our application window – another way

Steps:

1. Run application
2. Activate command prompt
3. Activate our application
4. Click on X (close) button

```
D:\programs\CJT\programs\gui>java MyFrameWithExitHandling
End of main
Window Deactivated
Window Activated
Window Closing
Window Deactivated
Window Closed

D:\programs\CJT\programs\gui>
```

41

Events, Event Listeners, and Listener Methods

Event Class	Listener Interface	Listener methods
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
MouseEvent	MouseListener	mousePressed(MouseEvent)
		mouseReleased(MouseEvent)
		mouseEntered(MouseEvent)
		mouseExited(MouseEvent)
		mouseClicked(MouseEvent)
MouseEvent	MouseMotionListener	mouseDragged(MouseEvent)
		mouseMoved(MouseEvent)

42

Events, Event Listeners, and Listener Methods

Event Class	Listener Interface	Listener methods
KeyEvent	KeyListener	keyPressed(KeyEvent)
		keyReleased(KeyEvent)
		keyTyped(KeyEvent)
WindowEvent	WindowListener	windowClosing(WindowEvent)
		windowOpened(WindowEvent)
		windowIconified(WindowEvent)
		windowDeiconified(WindowEvent)
		windowClosed(WindowEvent)
		windowActivated(WindowEvent)
		windowDeactivated(WindowEvent)

43

Events, Event Listeners, and Listener Methods

Event Class	Listener Interface	Listener methods
ContainerEvent	ContainerListener	componentAdded(ContainerEvent)
		componentRemoved(ContainerEvent)
ComponentEvent	ComponentListener	componentMoved(ComponentEvent)
		componentHidden(ComponentEvent)
		componentResized(ComponentEvent)
		componentShown(ComponentEvent)
FocusEvent	FocusListener	focusGained(FocusEvent)
		focusLost(FocusEvent)

44

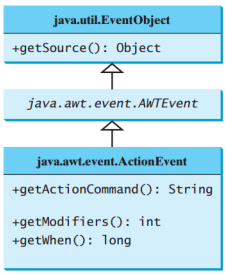
Events, Event Listeners, and Listener Methods

Event Class	Listener Interface	Listener methods
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)
ChangeEvent	ChangeListener	stateChanged(ChangeEvent)
ListSelectionEvent	ListSelectionListener	valueChanged(ListSelectionEvent)

45

Event object

- The event object contains information related to the event.
- getActionCommand() returns command string (e.g., for button, it is button label.
- getSource() method provides which source generated the event.
- getModifiers() returns the modifier keys held pressed during the event.
- getWhen() provides when (time) event occurred. The time is in terms of number of milliseconds since January 1, 1970, 00:00:00 GMT.



46

Anonymous class listeners

- A listener class is written to create a listener object for a GUI component (e.g., a button).
- The listener class will not be shared by other applications and therefore is appropriate to be defined inside the frame class as an inner class.
- An anonymous inner class is an inner class without a name. (If used only once, why to give it a name?)

47

Use of inner class

```
public class MyFrameWithExitHandling extends Frame{
    public MyFrameWithExitHandling(String title){
        addWindowListener(new MyWindowListener());
    }
    class MyWindowListener implements WindowListener{
        ...
        // event handlers
    }
}
```

48

Use of anonymous inner class

```
public class MyFrameWithExitHandling extends Frame{
    public MyFrameWithExitHandling(String title){
        addWindowListener(new WindowListener(){
                ...
                // event handlers
        });
    }
}
```

49

Use of anonymous inner class

- An anonymous inner class must always extend a superclass or implement an interface, but it cannot have an explicit extends or implements clause.
- An anonymous inner class must implement all the abstract methods present in the superclass or in the interface.
- An anonymous inner class always uses the no-argument constructor from its superclass to create an instance. If an anonymous inner class implements an interface, the constructor is Object().
- An anonymous inner class is compiled into a class named OuterClassName$n.class.
- For example, if the outer class Test has two anonymous inner classes, they are compiled intoTest$1.classandTest$2.class.

50

Can we minimize the event handler class?

- But, I am not interested in all event handler methods of a particular interface.
- E.g., the WindowListener interface has seven event handler methods.
- I want to handle only windowClosing(), why should I write other six methods?
- Is there any solution?

- We can use adapter classes.

51

Adapters for Listener Interfaces

- Java provides support classes, called convenience adapters, which provide default implementations for all the methods present in the listener interface.
- The default implementation has method definitions with empty body.
- Java provides adapters for those listener interfaces that have more than one event handler.
- A listener adapter is named XAdapter for XListener.
- For example, WindowAdapteris a listener adapter for WindowListener.

52

Adapters for Listener Interfaces

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

53

Example: closing window using WindowAdapter

```
import java.awt.*;
import java.awt.event.*;
public class MyFrameWithWindowAdapter extends Frame{
    public MyFrameWithWindowAdapter(String title){
        super(title);
        setVisible(true);
        setSize(400,400);
```

54

Example: closing window using WindowAdapter

```
addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
                System.out.println("Window Closing");
                dispose();
        }
        public void windowClosed(WindowEvent e){
                System.out.println("Window Closed");
                System.exit(0);
        }
});
}
```

55

Example: closing window using WindowAdapter

```
public static void main(String[] args){
        new MyFrameWithWindowAdapter("My Frame");
        System.out.println("End of main");
}
}
```

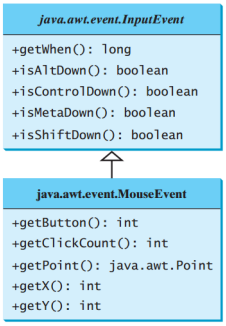
56

Mouse Events

- A mouse event is fired whenever a mouse is pressed, released, clicked, moved, or dragged on a component.
- The MouseEvent object captures the event, such as the number of clicks associated with it or the location (x- and y-coordinates) of the mouse when event fired.
- Since the MouseEvent class inherits InputEvent, you can use the methods defined in the InputEvent class on a MouseEvent object.

57

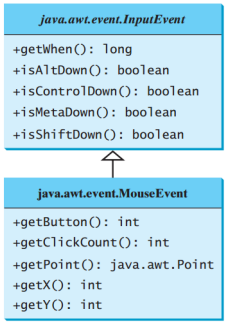
Mouse Events



- getWhen() returns the timestamp when the event occurred.
- isAltDown() returns true if the Alt key is pressed while the event occurred.
- isControlDown() returns true if the Ctrl key is pressed while the event occurred.
- isMetaDown() returns true if the Meta mouse button is pressed while the event occurred.
- isShiftDown() returns true if the Shift key is pressed while the event occurred.

58

Mouse Events



- getButton() indicates which button of the mouse has been clicked.
- getClickCount() returns the number of mouse clicks associated with this event.
- getPoint() Returns a java.awt.Point object containing the x- and y-coordinates.
- getX() returns the x-coordinate of the mouse point.
- getY() returns the y-coordinate of the mouse point.

59

Mouse Events

- java.awt.event.MouseListener
  - mousePressed(): Invoked after the mouse button has been pressed on the source component.
  - mouseReleased(): Invoked after the mouse button has been released on the source component.
  - mouseClicked(): Invoked after the mouse button has been clicked (pressed and released) on the source component.
  - mouseEntered(): Invoked after the mouse enters the source component.
  - mouseExited(): Invoked after the mouse exits the source component.

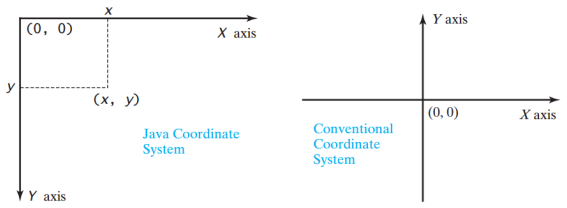
60

Mouse Events

- java.awt.event.MouseMotionListener
  - mouseDragged(): Invoked after a mouse button is moved with a button pressed.
  - mouseMoved(): Invoked after a mouse button is moved without a button pressed.

61

Graphical Coordinate System



62

Methods related to paint operation

- If we want that window should be refreshed, we need to call repaint() method:

  public void repaint();
- We should not override repaint() method, as it internally calls update() method.

  public void update(Graphics g)
- The update() method clears the drawing area of our application/container and then calls paint() method.

  public void paint(Graphics g)

63

When paint() method is called?

- There are two ways in which paint() method gets called
  1. Implicitly by Java when the area needs to be redrawn
  2. Explicitly by calling repaint() method

64

How to write/draw on client area of the window

- We perform writing text or drawing shapes, lines, or objects on the client area of the window using paint() method.
- The paint(Graphics g) method is invoked whenever the client area is to be drawn.
- Inside paint() method, we can write text or draw shapes using Graphics object g.
- How to re-draw screen after changes have occurred in the state of the application?
  - We need to call repaint() method.
  - The repaint() method wipes/clears out the screen and then calls paint() method.

65

Graphics object

- How to get Graphics object?
- We can get access to Graphics object inside paint() method.
- We can also get access to Graphics object by invoking getGraphics() method on Frame object.

66

Example: Show mouse coordinates as the mouse is moved

```
import java.awt.*;
import java.awt.event.*;
class MouseLocationDemo extends Frame implements
   MouseMotionListener{
   int cx=0,cy=0;
   public MouseLocationDemo(){
        super("Mouse location demo");
        addMouseMotionListener(this);
        setSize(300,400);
        setVisible(true);
   }
```

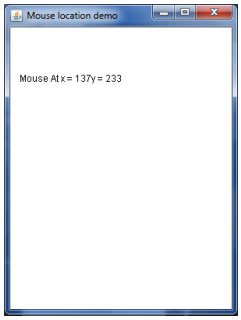
67

Example: Show mouse coordinates as the mouse is moved

```
   public void mouseDragged(MouseEvent e){ }
   public void mouseMoved(MouseEvent e){
        cx=e.getX();
        cy=e.getY();
        repaint();
   }
   public void paint(Graphics g){
        g.drawString("Mouse At x = "+cx+"\t\n\n\ty = "+cy,20,100);
   }
   public static void main(String [] args){
        new MouseLocationDemo();
   }
}
```

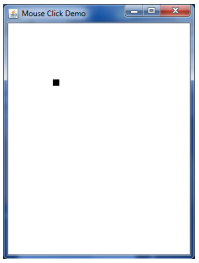
68

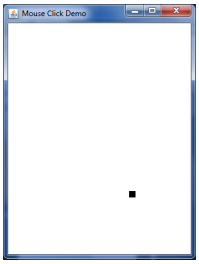
Example: Show mouse coordinates as the mouse is moved



69

Example: draw a square where mouse is clicked





70

Example: draw a square where mouse is clicked

```
import java.awt.*;
import java.awt.event.*;
class MouseClickDemo extends Frame implements
   MouseListener{
   int cx=0,cy=0;
   public MouseClickDemo(String title){
        super(title);
        addMouseListener(this);
        setSize(300,400);
        setVisible(true);
   }
```

71

Example: draw a square where mouse is clicked

```
   public void mousePressed(MouseEvent e){
        cx=e.getX();
        cy=e.getY();
        repaint();
   }
   public void mouseClicked(MouseEvent e){ }
   public void mouseReleased(MouseEvent e){}
   public void mouseEntered(MouseEvent e){}
   public void mouseExited(MouseEvent e){}
```

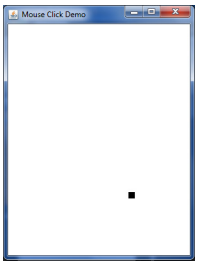
72

Example: draw a square where mouse is clicked

```
    public void paint(Graphics g){
        g.fillRect(cx-5,cy-5,10,10);
    }
    public static void main(String [] args){
        new MouseClickDemo("Mouse Click Demo");
    }
}
```

73

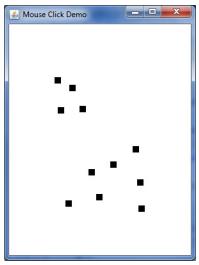
Example: draw a square where mouse is clicked



74

Example:

- Now, we want to preserve previously drawn squares



75

Example: preserve previously drawn squares

```
import java.awt.*;
import java.awt.event.*;
class PreserveMouseClickDemo extends Frame implements
    MouseListener{
    int cx=0,cy=0;
    public PreserveMouseClickDemo(String title){
        super(title);
        addMouseListener(this);
        setSize(300,400);
        setVisible(true);
    }
```

76

Example: preserve previously drawn squares

```
    public void mousePressed(MouseEvent e){
        cx=e.getX();
        cy=e.getY();
        Graphics g=getGraphics();
        g.fillRect(cx-5,cy-5,10,10);
    }
    public void mouseClicked(MouseEvent e){
    }
    public void mouseReleased(MouseEvent e){
    }
```

77

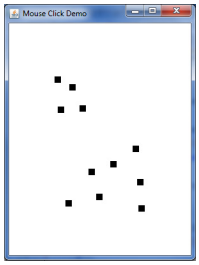
Example: preserve previously drawn squares

```
    public void mouseEntered(MouseEvent e){
    }
    public void mouseExited(MouseEvent e){
    }
    public static void main(String [] args){
        new PreserveMouseClickDemo("Mouse Click
    Demo");
    }
}
```

78

Example: preserve previously drawn squares

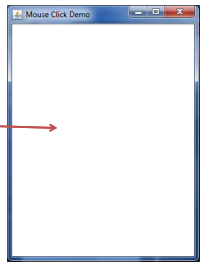
- But, if the client area is redrawn?
- Test
  - Iconify the window
  - Then, De-iconify the window
  - Observe what you see



79

Example: preserve previously drawn squares

- But, if the client area is redrawn?
- Test
  - Iconify the window
  - Then, De-iconify the window
  - Observe what you see
  - All drawn squares disappeared



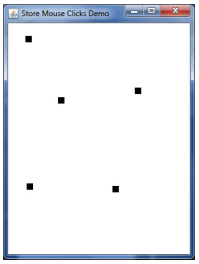
80

Example: Store previously drawn squares

- In previous application, if we minimize the window, and then restore the window, all drawn squares disappear.
- We need to store each point that we click on.
- We need to render all the points inside paint() method.
- The paint() method is called every time the client area is to be drawn.

81

Example: Store previously drawn squares



- Now even if we minimize and restore the window, squares will not disappear.

82

Example: Store previously drawn squares

```
import java.awt.*;
import java.awt.event.*;
class StoreMouseClickDemo extends Frame implements
    MouseListener{
    Point[] points=new Point[5]; // We can store max 5 clicks
    int clickCount=0;
    public StoreMouseClickDemo(String title){
          super(title);
          addMouseListener(this);
          setSize(300,400);
          setVisible(true);
    }
```

83

Example: Store previously drawn squares

```
public void mousePressed(MouseEvent e){ }
public void mouseClicked(MouseEvent e){
      int x,y;
      x=e.getX();              y=e.getY();
      if(clickCount<points.length){
            clickCount++;
            points[clickCount-1]=new Point(x,y);
      }
      repaint();
}
public void mouseReleased(MouseEvent e){}
```

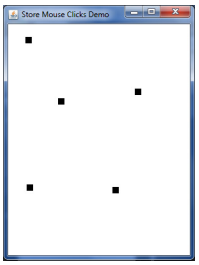
84

Example: Store previously drawn squares

```
public void mouseEntered(MouseEvent e){ }
public void mouseExited(MouseEvent e){ }
public void paint(Graphics g){
    for(int i=0;i<clickCount;i++)
        g.fillRect((int)points[i].getX()-5,
(int)points[i].getY()-5,10,10);
}
public static void main(String [] args){
    new StoreMouseClickDemo("Store Mouse Clicks Demo");
}
}
```

85

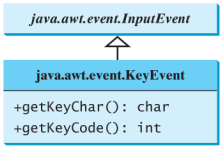
Example: Store previously drawn squares



86

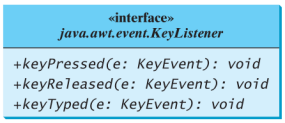
Key Events

- A key event is fired whenever a key is pressed, released, or typed on a component.
- The KeyEvent object describes the type of the event (a key has been pressed, released, or typed) and the value of the key



- The getKeyChar() method returns the character associated with the key
- The getKeyCode() returns integer Key code associated with the key

KeyListener



- The keyPressed() is invoked after a key is pressed on the source component.
- The keyReleased() is invoked after a key is released on the source component.
- The keyTyped() is invoked after a key is pressed and then released on the source component.

Key Constants

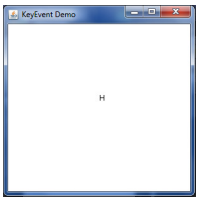
Constant	Description	Constant	Description
VK_HOME	The Home key	VK_CONTROL	The Control key
VK_END	The End key	VK_SHIFT	The Shift key
VK_PGUP	The Page Up key	VK_BACK_SPACE	The Backspace key
VK_PGDN	The Page Down key	VK_CAPS_LOCK	The Caps Lock key
VK_UP	The up-arrow key	VK_NUM_LOCK	The Num Lock key
VK_DOWN	The down-arrow key	VK_ENTER	The Enter key
VK_LEFT	The left-arrow key	VK_UNDEFINED	The keyCode unknown
VK_RIGHT	The right-arrow key	VK_F1 to VK_F12	The function keys from F1 to F12
VK_ESCAPE	The Esc key	VK_0 to VK_9	The number keys
VK_TAB	The Tab key	VK_A to VK_Z	The letter keys

Key code or key char?

- For the key-pressed and key-released events, getKeyCode()returns the value as defined earlier
- For the key-typed event
  - getKeyCode()returns VK_UNDEFINED, while
  - getKeyChar() returns the character entered.

Example: KeyEvent

- Using arrow keys, we can move the character
- We can also change the character



91

Example: KeyEvent

```
import java.awt.*;
import java.awt.event.*;
class KeyEventDemo  extends Frame implements KeyListener{
    private int x=150;
    private int y=150;
    private char keyChar = 'H';
    public KeyEventDemo(String title){
        super(title);
        addKeyListener(this);
        setSize(300,300);
        setVisible(true);
    }
```

92

Example: KeyEvent

```
    public static void main(String[] args){
        Frame f=new KeyEventDemo("KeyEvent Demo");
    }
    public void keyReleased(KeyEvent ke){
    }
    public void keyTyped(KeyEvent ke){
    }
```

93

Example: KeyEvent

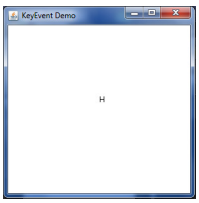
```
    public void keyPressed(KeyEvent ke){
        switch(ke.getKeyCode()){
            case KeyEvent.VK_DOWN: y +=10; break;
            case KeyEvent.VK_UP: y -=10; break;
            case KeyEvent.VK_LEFT: x -=10; break;
            case KeyEvent.VK_RIGHT: x +=10; break;
            default: keyChar = ke.getKeyChar();

        }
        repaint();
    }
```

94

Example: KeyEvent

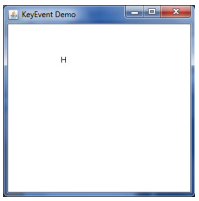
```
    public void paint(Graphics g){
        g.drawString(String.valueOf(keyChar),x,y);
    }
}
```

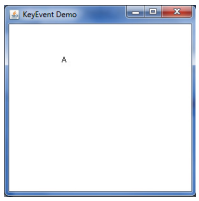


95

Example: KeyEvent

After moving the character 'H', we typed character 'A'.





96

97

Concept of Layout Managers

- In many GUI programming, the user interface components are arranged by hard-coded pixel measurements.
- Using AWT, the window might be displayed on many windowing systems on many screens.
- Java's layout manager provides abstraction to automatically map our user interface on all windowing system.

98

Layout Managers

- Layout manager is responsible for controlling size of components and arranging layout of components inside a container object.
- Every container has a layout manager.
- The container's setLayout method can be used to set a layout manager.
- Layout managers provided by AWT
  - FlowLayout
  - BorderLayout
  - CardLayout
  - GridLayout
  - GridBagLayout

How to use layout manager?

- Layout managers are defined by implementing LayoutManager interface.
- LayoutManager interface defines common methods
  - add(): to add a component in a container
  - remove() : to remove a component from a container
- For a container c, we can set a specific layout using setLayout() method on the container c.
  - c.setLayout(new specificLayout());
- Where specificLayout could be any of five mentioned in the previous slide.

100

Default Layout Managers

- Certain types of containers have default layout managers.
  - Frame has BorderLayout as default layout
  - Panel and Applet have FlowLayout as default layout

Layout Managers: FlowLayout

- FlowLayout is the simplest layout manager.
- The components are arranged in the container from left to right in the order in which they are added.
- It places the components in rows according to the width of the container and the number and size of the components.
- When one row becomes filled, a new row is started.
- On resizing the window, components may flow from one row to the previous.

Layout Managers: FlowLayout

- We can specify the way the components are aligned by using one of three constants
  - FlowLayout.RIGHT, FlowLayout.LEFT, FlowLayout.CENTER
- We can also specify gap (horizontal and vertical) between components in pixels.
- Constructors
  - FlowLayout(int align, int hGap, int vGap)
  - FlowLayout(int align)
    - default gap of 5 pixels
  - FlowLayout()
    - default gap of 5 pixels and default center alignment

Example: FlowLayout

```
import java.awt.*;
class FlowLayoutDemo extends Frame{
   void initComponents(){
        for(int i=0;i<30;i++)
                add(new Button("Button "+i));
   }
```

104

Example: FlowLayout

```
FlowLayoutDemo(String title){
     setTitle(title);
     setLayout(new FlowLayout());
     initComponents();
     setSize(400,300);
     setVisible(true);
}
```

105

Example: FlowLayout

```
   FlowLayoutDemo(String title, int alignment){
        setTitle(title);
        setLayout(new FlowLayout(alignment));
        initComponents();
        setSize(400,300);
        setVisible(true);
   }
```

106

Example: FlowLayout

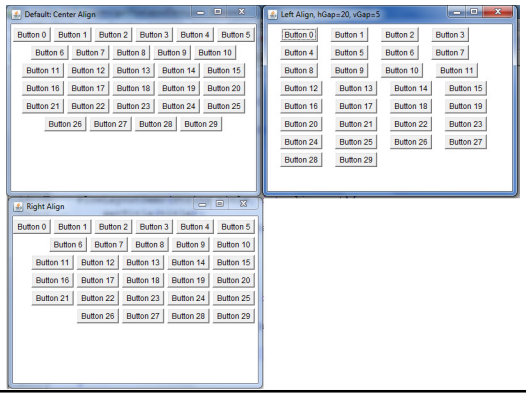
```
FlowLayoutDemo(String title, int alignment, int hGap, int
vGap){
     setTitle(title);
     setLayout(new FlowLayout(alignment, hGap, vGap));
     initComponents();
     setSize(400,300);
     setVisible(true);
}
```

107

Example: FlowLayout

```
   public static void main(String[] args){
        new FlowLayoutDemo("Default: Center Align");
        new FlowLayoutDemo("Right Align",FlowLayout.RIGHT);
        new FlowLayoutDemo("Left Align, hGap=20,
   vGap=5",FlowLayout.LEFT, 20,5);
   }
}
```

108



109

Example: ActionEvent demo

```
import java.awt.*;
import java.awt.event.*;
public class ActionEventDemo extends Frame implements
    ActionListener {
    private Button closeBtn;
    public ActionEventDemo(String title){
        setTitle(title);
        setLayout(new FlowLayout());
        closeBtn=new Button("Close");
        add(closeBtn);
        closeBtn.addActionListener(this);
```

110

Example: ActionEvent demo

```
        setVisible(true);
        setSize(400,400);
    }
    public void actionPerformed(ActionEvent ae){
        String cmd=ae.getActionCommand();
        if(ae.getSource() instanceof Button)
            if(cmd.equals("Close"))
                System.exit(0);
    }
```

111

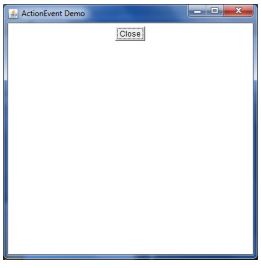
Example: ActionEvent demo

```
    public static void main(String[] args){
        new ActionEventDemo("ActionEvent Demo");

    }
}
```

112

Example: ActionEvent demo



113

Layout Managers: BorderLayout

- BorderLayout consists of five fixed areas: North, South, East, West, and Center.
- If any or all of the North, South, East, or West areas are left out, the Central area takes space of the missing area or areas.
- However, if the Central area is left out, the North, South, East, or West areas do not change.
- Note that when using the BorderLayout, you must add components with the add(String, Component), where first argument could be one of the following:
  - "North", "South", "East", "West", "Center"

Layout Managers: BorderLayout

- Constructors:
- public BorderLayout()
  - Constructs a new border layout without horizontal or vertical gaps
- public BorderLayout(int hGap, int vGap)
  - Constructs a new border layout with the specified horizontal and vertical gaps between the components.

Customizing a component

- We can customize existing Java AWT component by extending from it and overriding the component's paint(Graphics) method
- E.g.,

```
class MyButton extends Button{
   public void paint(Graphics g){
        …
   }
}
```

116

Example: BorderLayout

```
import java.awt.*;
class BorderLayoutDemo extends Frame{
   BorderLayoutDemo(String title){
        setTitle(title);
        Button red=new MyButton("Red color button");
        add("Center",new Button("Center"));
        add(red);
```

117

Example: BorderLayout

```
        add("North",new Button("North"));
        add("South",new Button("South"));
        add("East",new Button("East"));
        add("West",new Button("West"));
        setSize(400,400);
        setVisible(true);
   }
   public static void main(String[] args){
        new BorderLayoutDemo("Custom Button in
BorderLayout");
   }
}
```

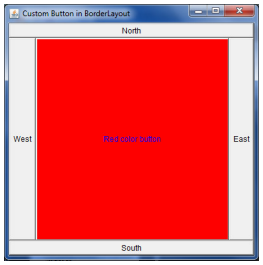
118

Example: BorderLayout

```
class MyButton extends Button{
   public MyButton(String label){
        super(label);
   }
  public void paint(Graphics g){
        setBackground(Color.red);
        setForeground(Color.blue);
   }
}
```

119

Example: BorderLayout



120

Layout Managers: GridLayout

- The Grid layout arranges components into a grid of rows and columns, specified by the constructor.
- The components are placed in the grid from left to right starting with the first row, then second, and so on.
- Components are placed in the order they are added.
- The cells in the grid are equal size based on the largest component in the grid.
- We can resize the window to see how the components are resized to fit cells as they get larger and smaller.

Layout Managers: GridLayout

- We can specify
  - the number of rows and columns, or
  - the number of rows only and let the layout manager determine the number of columns,
  - or the number of columns only and let the layout manager determine the number of rows.
- Widely used Constructors:
  - public GridLayout(int rows, int columns)
  - public GridLayout(int rows, int columns, int hGap, int vGap)

Example: GridLayout

```
import java.awt.*;
class GridLayoutDemo extends Frame{
    GridLayoutDemo(String title){
        setTitle(title);
        setLayout(new GridLayout(4,3));
        for(int i=0;i<15;i++)
                add(new Button(" "+i));
        setSize(400,400);
        setVisible(true);
    }
```

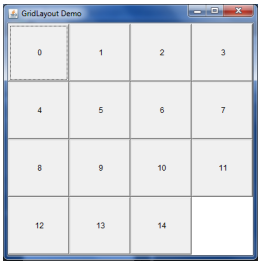
123

Example: GridLayout

```
    public static void main(String[] args){
        new GridLayoutDemo("GridLayout Demo");
    }
}
```

124

Example: GridLayout



125

Panels

- When certain layout of components is not possible using a single container, we can use Panel as a container.
- We can add components in a Panel having desired layout manager.
- Then, we can add the Panel in the main container having an appropriate layout manager.

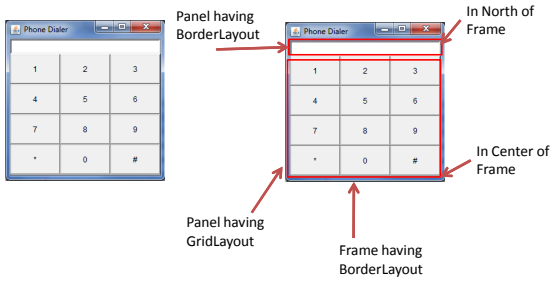
126

Panel

- Call constructor
  - Panel p=new Panel();
- Add components in the panel
  - p.add(new Button(“+”));
  - p.add(new Button(“-”));
  - p.add(new Button(“*”));
  - p.add(new Button(“/”));
- By default, Panels have FlowLayout.

127

Example: Panel



128

Example: Panel

```
import java.awt.*;
class PhoneDialerGUI extends Frame{
   PhoneDialerGUI(String title){
        setTitle(title);
        Panel numberPanel=new Panel();
        numberPanel.setLayout(new BorderLayout());
        numberPanel.add(new TextField());
```

129

Example: Panel

```
Panel dialPanel=new Panel();
dialPanel.setLayout(new GridLayout(4,3));
for(int i=1;i<=9;i++)
        dialPanel.add(new Button(""+i));
dialPanel.add(new Button("*"));
dialPanel.add(new Button("0"));
dialPanel.add(new Button("#"));
```

130

Example: Panel

```
        setLayout(new BorderLayout());
        add("North",numberPanel);
        add("Center",dialPanel);
        setVisible(true);
        setSize(250,250);
   }
   public static void main(String[] args){
        new PhoneDialerGUI("Phone Dialer");
   }
}
```

131

Example: Panel

132

Canvases

- Canvas is a UI component that can be used to draw graphics
- It can also enable user interaction.
- When we create a Canvas object, it appears as a blank space inside the container.
- We can perform the following with Canvas:
  - Setting the color
  - Setting size
  - Getting events

133

Canvas

- How to use Canvas

```
Canvas c=new Canvas();
c.setSize(50, 50);
c.setBackground(Color.blue);
add(c);
```

134

Example: Canvas

```
class MyCanvas extends Canvas{
    int x=10;
    int y=20;
    String message="Your message here...";
    public MyCanvas(String message){
        this.message=message;
        repaint();
    }
    public void paint(Graphics g){
        g.drawString(message,x,y);
    }
}
```

135

Example: Canvas

```
import java.awt.*;
class CanvasDemo extends Frame{
    CanvasDemo(String title){
        setTitle(title);
        setVisible(true);
        setSize(300,300);

        Canvas c=new Canvas();
        c.setSize(300,100);
        c.setBackground(Color.yellow);
        add("South", c);
```

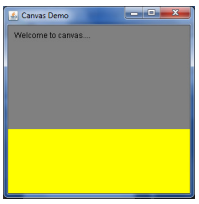
136

Example: Canvas

```
        MyCanvas myCanvas=new MyCanvas("Welcome to
    canvas....");
        myCanvas.setSize(300,100);
        myCanvas.setBackground(Color.gray);
        add("Center", myCanvas);
    }
    public static void main(String[] args){
        new CanvasDemo("Canvas Demo");
    }
}
```

137

Example: Canvas



138

Graphics class and its methods

Graphics class

- The Graphics class provides the methods for drawing strings, lines, rectangles, ovals, arcs, polygons, and polylines.
- We can think of
  - GUI component as a piece of paper
  - Graphics object as a pencil or paintbrush.
- We can apply the methods in the Graphics class to draw things on a GUI component.
- The Graphics class is an abstract class and it provides a device-independent graphics interface for displaying figures and images on the screen on different platforms.

Methods of Graphics class

- setColor(Color color):
  - Sets a new color for subsequent drawings
- setFont(Font font)
  - Sets a new font for subsequent drawings

Methods of Graphics class

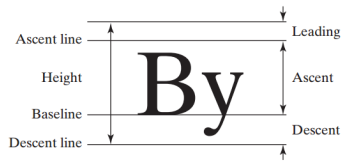
- drawString(String s, int x, int y)
  - Draws a string starting at point (x, y).

Font and FontMetrics classes

- We can set a particular font for drawing using Font object.
  Font myFont=new Font(name, style, size);
- We specify font's name as String, e.g.,
  - "TimesRoman", "Courier", "Arial", "Helvetica"
- We can choose font style using the following:
  - Font.PLAIN, Font.BOLD, Font.ITALIC
- How to create font object?
  Font f1=new Font("TimesRoman", Font.BOLD, 16);
  Font f2=new Font("Courier", Font.BOLD+Font.ITALIC, 12);
- Set font using Graphics object
  - g.setFont(f2);

Font and FontMetrics classes

- We can use FontMetrics class to get measurement (length and width) information of a drawn string using particular Font.



Font and FontMetrics classes

- Leading (pronounced as ledding): amount of space between line of text.
- Ascent: It is height of a character, from the baseline to the top.
- Descent: It is the distance from the baseline to the bottom of a descending character, such as j, y, and g.
- Height: It is the sum of leading, ascent, and descent.

145

Font and FontMetrics classes

- FontMetrics is an abstract class.
- To get FontMetrics object, use getFontMetrics() methods defined in the Graphics class.
  public FontMetrics getFontMetrics(Font font)
  public FontMetrics getFontMetrics()
- Methods to get Font information:
  - public int getAscent();
  - public int getDescent();
  - public int getLeading();
  - public int getHeight();
  - public int stringWidth(String str);

146

Example: Font and FontMetrics classes

```
import java.awt.*;
import java.awt.event.*;
class FontDemo extends Frame{
    String msg="Hello world";
    public FontDemo(String title){
        setTitle(title);
        addComponentListener(new ComponentAdapter(){
            public void componentResized(ComponentEvent ce){
                repaint();
            }
        });
```

147

Example: Font and FontMetrics classes

```
        setSize(400,200);
        setVisible(true);
    }
    public static void main(String[] args){
        FontDemo fd=new FontDemo("String at Center");
    }
```

148

Example: Font and FontMetrics classes

```
    public void paint(Graphics g){
        int x,y;
        int h,w;
        Font f=new Font("Arial",Font.BOLD,16);
        g.setFont(f);
        FontMetrics fm=g.getFontMetrics(f);
        h=fm.getAscent();
        w=fm.stringWidth(msg);
        x=(getSize().width-w)/2;
        y=(getSize().height+h)/2;
        g.drawString(msg,x,y);
    }
}
```

149

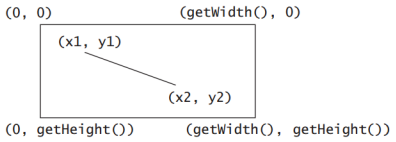
Example: Font and FontMetrics classes

150

Drawing shapes using Graphics class

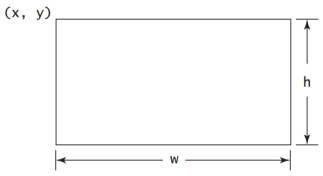
Methods of Graphics class

- drawLine(int x1, int y1, int x2, int y2)
  - Draws a line from (x1, y1)to (x2, y2)



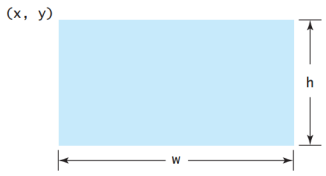
Methods of Graphics class

- drawRect(int x, int y, int w, int h)
  - Draws a rectangle with specified upper-left corner point at (x,y) and width w and height h



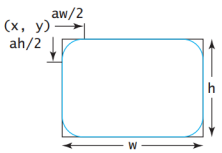
Methods of Graphics class

- fillRect(int x, int y, int w, int h)
  - Draws a filled rectangle with specified upper-left corner point at (x, y)and width w and height h



Methods of Graphics class

- drawRoundRect(int x, int y, int w, int h, int aw, int ah)
  - Draws a round-cornered rectangle with specified arc width aw and arc height ah



Methods of Graphics class

- fillRoundRect(int x, int y, int w, int h, int aw, int ah)
  - Draws a filled round-cornered rectangle with specified arc width aw and arc height ah

Example: shapes demo

```
import java.awt.*;
import java.awt.event.*;
class ShapesDemo extends Frame{
   String msg="Hello world";
   public ShapesDemo(String title){
        setTitle(title);
        addComponentListener(new ComponentAdapter(){
                public void
   componentResized(ComponentEvent ce){
                        repaint();
                }
        });
```

157

Example: shapes demo

```
        setSize(400,200);
        setVisible(true);
}
public static void main(String[] args){
        ShapesDemo sd=new ShapesDemo("Shapes...");
}
```

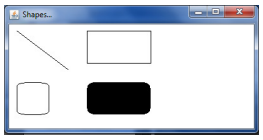
158

Example: shapes demo

```
   public void paint(Graphics g){
        g.drawLine(20,40,100,100);
        g.drawRect(130,40,100,50);
        g.drawRoundRect(20,120,50,50,40,10);
        g.fillRoundRect(130,120,100,50,20,20);
   }
}
```

159

Example: shapes demo



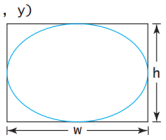
160

Methods of Graphics class

- draw3DRect(int x, int y, int w, int h, boolean raised)
  - Draws a 3-D rectangle raised above the surface or sunk into the surface
- fill3DRect(int x, int y, int w, int h, boolean raised)
  - Draws a filled 3-D rectangle raised above the surface or sunk into the surface.

Methods of Graphics class

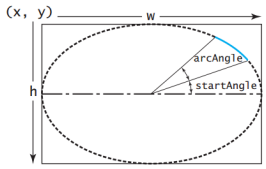
- drawOval(int x, int y, int w, int h)
  - Draws an oval bounded by the rectangle specified by the param



- fillOval(int x, int y, int w, int h)
  - Draws a filled oval bounded by the rectangle specified by the parameters x, y, w, and h.

Methods of Graphics class

- drawArc(int x, int y, int w, int h, int startAngle, int arcAngle)
  - Draws an arc conceived as part of an oval bounded by the rectangle specified by the parameters x, y, w, and h.



Methods of Graphics class

- fillArc(int x, int y, int w, int h, int startAngle, int arcAngle)
  - Draws a filled arc conceived as part of an oval bounded by the rectangle specified by the parameters x, y, w, and h.

Example: Shapes demo

```
import java.awt.*;
import java.awt.event.*;
class ShapesDemo2 extends Frame{
   String msg="Hello world";
   public ShapesDemo2(String title){
        setTitle(title);
        addComponentListener(new ComponentAdapter(){
                public void
   componentResized(ComponentEvent ce){
                        repaint();
                }
        });
```

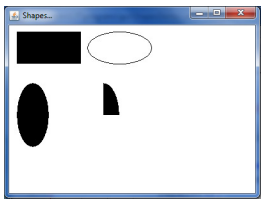

Example: Shapes demo

```
        setSize(400,300);
        setVisible(true);
}
public static void main(String[] args){
        ShapesDemo2 sd=new ShapesDemo2("Shapes...");
}
```


Example: Shapes demo

```
   public void paint(Graphics g){
        g.fill3DRect(20,40,100,50,true);
        g.drawOval(130,40,100,50);
        g.fillOval(20,120,50,100);
        g.fillArc(130,120,50,100,0,90);
   }
}
```


Example: Shapes demo

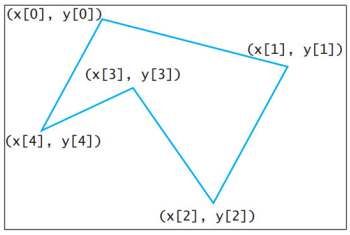


Methods of Graphics class

- drawPolygon(Polygon g)
  - Draws a closed polygon defined by a Polygon object
- fillPolygon(Polygon g)
  - Draws a filled polygon defined by a Polygon object.

Methods of Graphics class

- drawPolygon(int[] xPoints, int[] yPoints, int nPoints)
  - Draws a closed polygon defined by arrays of x- and y-coordinates. Each pair of (x[i], y[i])-coordinates is a point

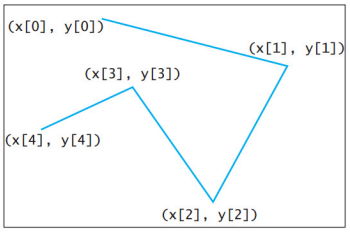


Methods of Graphics class

- fillPolygon(int[] xPoints, int[] yPoints, int nPoints)
  - Draws a filled polygon defined by arrays of x- and y-coordinates. Each pair of (x[i], y[i])-coordinates is a point.

Methods of Graphics class

- drawPolyline(int[] xPoints, int[] yPoints, int nPoints)
  - Draws a polyline defined by arrays of x- and y-coordinates. Each pair of (x[i], y[i])-coordinates is a point



Example: polygon

```
import java.awt.*;
import java.awt.event.*;
class ShapesDemo3 extends Frame{
   String msg="Hello world";
   public ShapesDemo3(String title){
        setTitle(title);
        addComponentListener(new ComponentAdapter(){
                public void
   componentResized(ComponentEvent ce){
                        repaint();
                }
        });
```

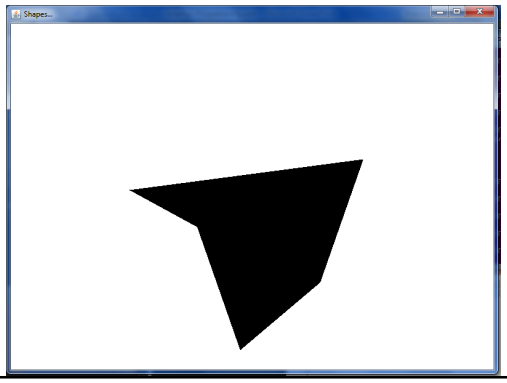
173

Example: polygon

```
        setSize(800,600);
        setVisible(true);
   }
   public static void main(String[] args){
        ShapesDemo3 sd=new ShapesDemo3("Shapes...");
   }
   public void paint(Graphics g){
        int x[] = {200,310,380,510,580};
        int y[] = {300,360,560,450,250};
        g.fillPolygon(x,y,x.length);
   }
}
```

174

Example: polygon



175