

UNIT V EVENT DRIVEN PROGRAMMING

Graphics programming - Frame – Components - working with 2D shapes - Using color, fonts, and images - Basics of event handling - event handlers - adapter classes - actions - mouse events - AWT event hierarchy - Introduction to Swing – layout management - Swing Components – Text Fields , Text Areas – Buttons- Check Boxes – Radio Buttons – Lists- choices- Scrollbars – Windows –Menus – Dialog Boxes

5.1 Graphics Programming

When Java 1.0 was introduced, it contained a class library, which Sun called the Abstract Window Toolkit (AWT), for basic GUI programming.

The basic AWT library deals with user interface elements by delegating their creation and behavior to the native GUI toolkit on each target platform (Windows, Solaris, Macintosh, and so on).

The Class hierarchy for Panel and Frame

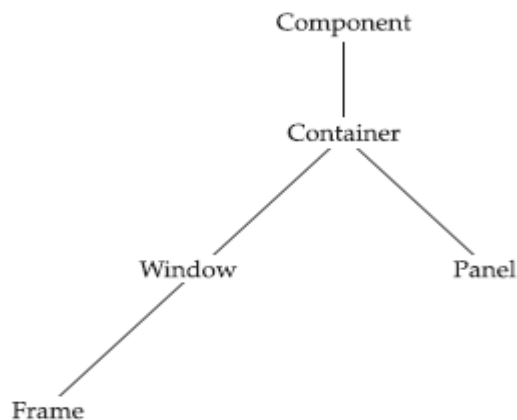


Fig 5.1 The Class hierarchy for Panel and Frame

Component

A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Components are the buttons, checkboxes, and scrollbars of a typical graphical user interface.

Container

The **Container** class is a subclass of **Component**. **Component** objects to be nested within it. Other **Container** objects can be stored inside of a **Container**. A container is responsible for laying out (that is, positioning) any components that it contains.

Panel

The **Panel** class is a concrete subclass of **Container**. A **Panel** may be thought of as a recursively nestable, concrete screen component. **Panel** is the superclass for **Applet**. When screen output is directed to an applet, it is drawn on the surface of a **Panel** object.

In essence, a **Panel** is a window that does not contain a title bar, menu bar, or border. Other components can be added to a **Panel** object by its **add()** method. Once these components have been added, you can position and resize them manually using the **setLocation()**, **setSize()**, **setPreferredSize()**, or **setBounds()** methods defined by **Component**.

Window

The **Window** class creates a top-level window. A *top-level window* is not contained within any other object; it sits directly on the desktop.

Frame

Frame encapsulates what is commonly thought of as a “window.” It is a subclass of **Window** and has a title bar, menu bar, borders, and resizing corners.

5.2 Frame

A top-level window (that is, a window that is not contained inside another window) is called a **frame** in Java.

The AWT library has a class, called **Frame**, for this top level.

The Swing version of this class is called **JFrame** and extends the **Frame** class.

The **JFrame** is one of the few Swing components that is not painted on a canvas.

Thus, the decorations (buttons, title bar, icons, and so on) are drawn by the user's windowing system, not by Swing.

5.2.1 Frame Methods

1. **String getTitle():** Gets the title of the frame.
2. **void setTitle(String title):** Sets the title for this frame to the specified string.
3. **void setDefaultCloseOperation(int operation):** Sets the operation that will happen by default when the user initiates a "close" on this frame.
4. **void setIconImage():** Sets the image to be displayed in the minimized icon for this frame.
5. **void setResizable():** Sets whether this frame is resizable by the user.

Example Program:

```
import javax.swing.*;
public class SimpleFrameTest
{
    public static void main(String[] args)
    {
        SimpleFrame frame = new SimpleFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
class SimpleFrame extends JFrame
{
    public SimpleFrame()
    {
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
    }
    public static final int DEFAULT_WIDTH = 300;
    public static final int DEFAULT_HEIGHT = 200;
}
```

Output:



5.3 Component

A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Components are the buttons, checkboxes, and scrollbars of a typical graphical user interface.

Window

The **Window** class creates a top-level window. A *top-level window* is not contained within any other object; it sits directly on the desktop.

java.awt.Window contains the following methods

- **void toFront()**
shows this window on top of any other windows.
- **void toBack()**
moves this window to the back of the stack of windows on the desktop and rearranges all other visible windows accordingly.
- **boolean isLocationByPlatform()**
- **void setLocationByPlatform(boolean b)**
gets or sets the locationByPlatform property. When the property is set before this window is displayed, the platform picks a suitable location.

java.awt.Component contains the following methods

- **void setLocation(x,y)** : The top-left corner is located x pixels across and y pixels down.

- **void setBounds(x,y,width,height):** Resize and Relocate a component.
- **boolean isVisible():** Checks if this component is set to be visible.
- **void setVisible(boolean b):** Shows or hides the component depending on whether b is true or false.
- **boolean isShowing():** Checks if this component is showing on the screen.
- **boolean isEnabled():** Checks if this component is enabled.
- **void setEnabled(boolean b):** Enables or disables a component.
- **Point getLocation():** returns the location of the top-left corner of this component.
- **Point getLocationOnScreen():** returns the location of the top-left corner of component,using the screen's coordinates.
- **void setBounds(int x,int y,int width,int height):** Moves and resizes this component.The location of the top-left corner is given by x and y and the new size is given by the width and height parameters.
- **Dimension getSize():** Gets the current size of this component.

When designing a frame, you add components into the content pane, using code such as the following:

```
Container contentPane = frame.getContentPane();  
Component c = . . .;  
contentPane.add(c);
```

In our case, we want to add a single component to the frame onto which we will draw our message.

To draw on a component, you define a class that extends JComponent and override the paintComponent method in that class.

The `paintComponent` method takes one parameter of type `Graphics`. A `Graphics` object remembers a collection of settings for drawing images and text, such as the font you set or the current color. All drawing in Java must go through a `Graphics` object.

It has methods that draw patterns, images, and text. Here's how to make a component onto which you can draw:

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

Each time a window needs to be redrawn, the event handler notifies the component. This causes the `paintComponent` methods of all components to be executed.

Displaying text is considered a special kind of drawing. The `Graphics` class has a `drawString` method that has the following syntax:

```
g.drawString(text, x, y)
```

Example Program:

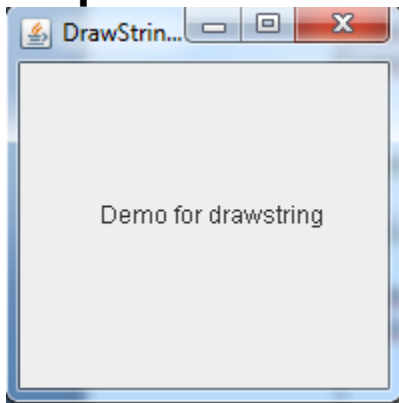
```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
  
public class DrawStringDemo  
{  
  public static void main(String args[])  
  {  
    JFrame frame=new JFrame("DrawStringDemo");  
    Container contentPane=frame.getContentPane();  
    contentPane.add(new JPanel());  
    frame.setSize(200,200);  
    frame.setLocation(100,200);  
  }  
}
```

```

frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.show();
}
}
class NewPanel extends JPanel
{
public void paintComponent(Graphics g)
{
super.paintComponent(g);
g.drawString("Demo for drawstring",40,80);
}
}

```

Output:



5.3.1 Frame Positioning and Centering

Toolkit class in Abstract Window Kit is used to get system-dependent information.

Methods:

- static Toolkit getDefaultToolkit(): Gets the default toolkit
- Dimension getScreenSize(): Gets the size of the user's screen
- Image getImage(String filename): Loads an image from the file with name filename

Example Program:

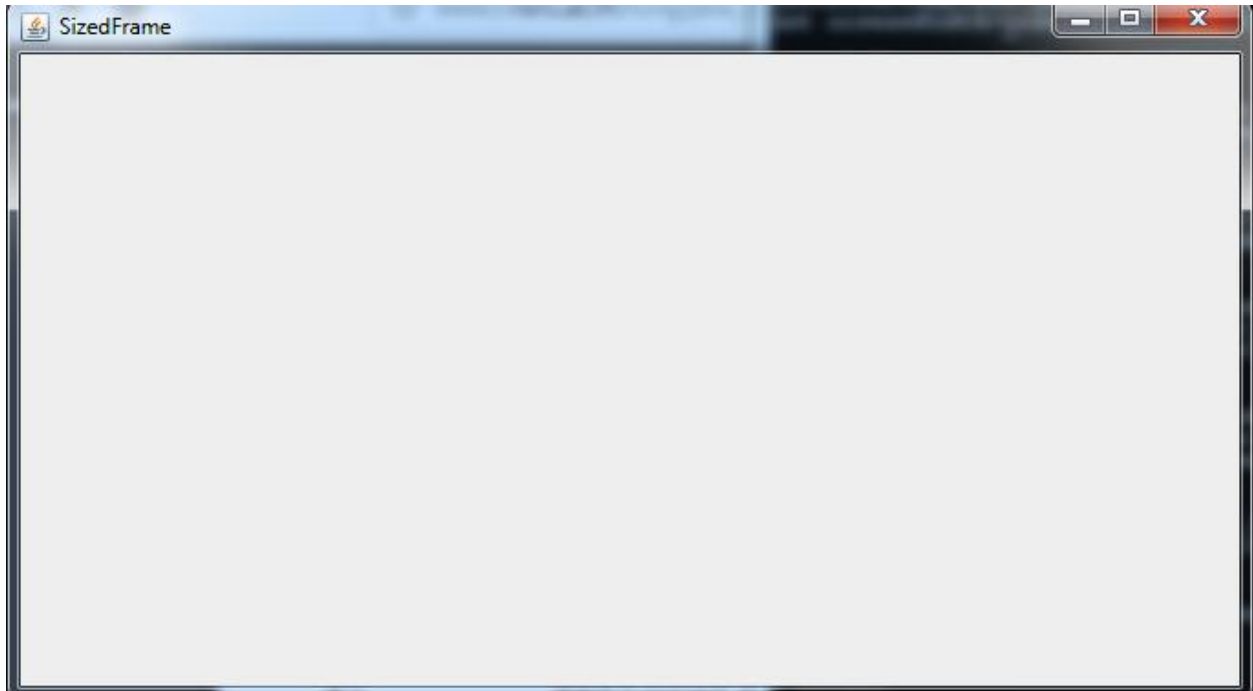
```
import java.awt.*;
```

```

import javax.swing.*;
public class SizedFrameTest
{
    public static void main(String[] args)
    {
        SizedFrame frame = new SizedFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
class SizedFrame extends JFrame
{
    public SizedFrame()
    {
        // get screen dimensions
        Toolkit kit = Toolkit.getDefaultToolkit();
        Dimension screenSize = kit.getScreenSize();
        int screenHeight = screenSize.height;
        int screenWidth = screenSize.width;
        // set frame width, height and let platform pick screen location
        setSize(screenWidth / 2, screenHeight / 2);
        setLocationByPlatform(true);
        // set frame icon and title
        Image img = kit.getImage("icon.gif");
        setIconImage(img);
        setTitle("SizedFrame");
    }
}

```

Output:



5.4 Working with 2D Shapes

To draw shapes in Java 2D library, an object of the Graphics2D class is obtained. All shapes are drawn using paintComponent with Graphics object.

Syntax:

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

Constructor:

Graphics2D : Constructs a new Graphics2D object.

5.4.1 Methods:

- **void draw(Shape s):** draws the shape using the settings of the current Graphics2D context.
- **void fill(Shape s):** Fills the interior of a shape using the settings of the current Graphics2D context.

- **void setPaint(Paint paint):** Sets the paint attribute for the Graphics2D context.
- **void drawString(String s,float x,float y):** draws the string in the x,y coordinates specified in the drawstring method using Graphics2D context.
- **FontRenderContext getFontRenderContext():** Gets the rendering context of the font with this Graphics2D context.

The Java 2D library organizes geometric shapes in an object-oriented fashion. In particular, there are classes to represent lines, rectangles, and ellipses:

Line2D

Rectangle2D

Ellipse2D

These classes all implement the Shape interface.

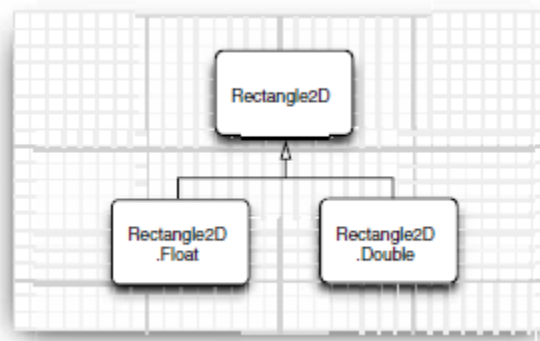
To draw a shape, you first create an object of a class that implements the Shape interface and then call the draw method of the Graphics2D class. For example:

```
Rectangle2D rect = . . . ;  
g2.draw(rect);
```

Consider the Rectangle2D class. This is an abstract class with two concrete subclasses, which are also

Rectangle2D.Float

Rectangle2D.Double



Methods:

java.awt.geom.RectangularShape

- **double getCenterX()**
- **double getCenterY()**
- **double getMinX()**
- **double getMinY()**
- **double getMaxX()**
- **double getMaxY()**

returns the center, minimum, or maximum x- or y-value of the enclosing rectangle.

- **double getWidth()**
- **double getHeight()**

returns the width or height of the enclosing rectangle.

- **double getX()**
- **double getY()**

returns the x- or y-coordinate of the top-left corner of the enclosing rectangle.

java.awt.geom.Rectangle2D.Double

Rectangle2D.Double(double x, double y, double w, double h)

constructs a rectangle with the given top-left corner, width, and height.

java.awt.geom.Rectangle2D.Float

Rectangle2D.Float(float x, float y, float w, float h)

constructs a rectangle with the given top-left corner, width, and height

java.awt.geom.Ellipse2D.Double

Ellipse2D.Double(double x, double y, double w, double h)

constructs an ellipse whose bounding rectangle has the given top-left corner, width, and height.

java.awt.geom.Point2D.Double

Point2D.Double(double x, double y)

constructs a point with the given coordinates.

java.awt.geom.Line2D.Double
Line2D.Double(Point2D start, Point2D end)
Line2D.Double(double startX, double startY, double endX, double endY)

constructs a line with the given start and end points.

Example Program:

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
public class DrawTest
{
    public static void main(String[] args)
    {
        DrawFrame frame = new DrawFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
/**
 * A frame that contains a panel with drawings
 */
class DrawFrame extends JFrame
{
    public DrawFrame()
    {
        setTitle("DrawTest");
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
        // add panel to frame
        DrawComponent component = new DrawComponent();
        add(component);
    }
    public static final int DEFAULT_WIDTH = 400;
    public static final int DEFAULT_HEIGHT = 400;
}
/**
 * A component that displays rectangles and ellipses.
```

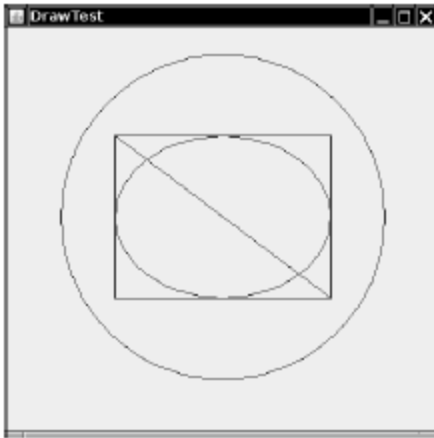
```

*/
class DrawComponent extends JComponent
{
public void paintComponent(Graphics g)
{
Graphics2D g2 = (Graphics2D) g;
// draw a rectangle
double leftX = 100;
double topY = 100;
double width = 200;
double height = 150;
Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width,
height);
g2.draw(rect);

// draw the enclosed ellipse
Ellipse2D ellipse = new Ellipse2D.Double();
ellipse setFrame(rect);
g2.draw(ellipse);
// draw a diagonal line
g2.draw(new Line2D.Double(leftX, topY, leftX + width, topY +
height));
// draw a circle with the same center
double centerX = rect.getCenterX();
double centerY = rect.getCenterY();
double radius = 150;
Ellipse2D circle = new Ellipse2D.Double();
circle.setFrameFromCenter(centerX, centerY, centerX + radius,
centerY + radius);
g2.draw(circle);
}
}

```

Output:



5.5 Using color, fonts, and images

5.5.1 Color

The `setPaint` method of the `Graphics2D` class lets you select a color that is used for all subsequent drawing operations on the graphics context. For example:

```
g2.setPaint(Color.RED);  
g2.drawString("Warning!", 100, 100);
```

You can fill the interiors of closed shapes (such as rectangles or ellipses) with a color. Simply call `fill` instead of `draw`:

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

To draw in multiple colors, you select a color, draw or fill, then select another color, and draw or fill again.

You define colors with the `Color` class. The **`java.awt.Color`** class offers predefined constants for the following 13 standard colors:

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

You can specify a custom color by creating a `Color` object by its red, green, and blue components.

Using a scale of 0–255 (that is, one byte) for the redness, blueness, and greenness, call the `Color` constructor like this:

```
Color(int redness, int greenness, int blueness)
```

To set the *background color*, you use the setBackground method of the Component class, an ancestor of JComponent.

```
MyComponent p = new MyComponent();  
p.setBackground(Color.PINK);
```

5.5.1.1 Methods:

java.awt.Color

Color(int r, int g, int b)

creates a color object.

Color getColor()

void setColor(Color c)

gets or sets the current color. All subsequent graphics operations will use the newcolor.

java.awt.Color

Parameters: r The red value (0–255)

g The green value (0–255)

b The blue value (0–255)

java.awt.Graphics

Color getColor()

void setColor(Color c)

gets or sets the current color. All subsequent graphics operations will use the new color.

Parameters: c The new color

java.awt.Graphics2D

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

Color getBackground()

void setBackground(Color c)

gets or sets the background color.

Color getForeground()

void setForeground(Color c)

gets or sets the foreground color.

5.5.2 Fonts

The java.awt.Font class is used to create Font objects to set the font.

Constructor:

Font(String name,int style,int size):Creates a new font with the specified name,style and point size.

Example:

```
Font sansbold14 = new Font("SansSerif", Font.BOLD, 14);
```

Type setting terms:

The **getStringBounds** method creates a rectangle that contains the string.

The **baseline** is the imaginary line where, for example, the bottom of a character like "e" rests.

The **ascent** is the distance from the baseline to the top of an *ascender*, which is the upper part of a letter like "b" or "k," or an uppercase character.

The **descent** is the distance from the baseline to a *descender*, which is the lower portion of a letter like "p" or "g."

Leading is the space between the descent of one line and the ascent of the next line. (The term has its origin from the strips of lead that typesetters used to separate lines.)

The **height** of a font is the distance between successive baselines, which is the same as descent + leading + ascent.

The **width** of the rectangle is the horizontal extent of the string returned using getStringBounds() method.

The **height** of the **rectangle** is the sum of ascent, descent, and leading. The rectangle has its origin at the baseline of the string. The top y-coordinate of the rectangle is negative.

5.5.2.1 Methods:

java.awt.Font

1. **Font(String name, int style, int size)**
creates a new font object.

Parameters:

name	The font name. This is either a font face name (such as "Helvetica Bold") or a logical font name (such as "Serif", "SansSerif")
style	The style (Font.PLAIN, Font.BOLD, Font.ITALIC, or Font.BOLD +Font.ITALIC)
size	The point size

String getFontName()

gets the font face name (such as "Helvetica Bold").

String getFamily()

gets the font family name (such as "Helvetica").

String getName()

gets the logical name (such as "SansSerif") if the font was created with a logical font name; otherwise, gets the font face name.

Rectangle2D getStringBounds(String s, FontRenderContext context)

returns a rectangle that encloses the string. The origin of the rectangle falls on the baseline. The negative of the ascent equals to the top y-coordinate of the rectangle. The height of the rectangle is the sum of ascent, descent, and leading.

LineMetrics getLineMetrics(String s, FontRenderContext context)

returns a line metrics object to determine the extent of the string.

Font deriveFont(int style)

Font deriveFont(float size)

Font deriveFont(int style, float size)

returns a new font that equals this font, except that it has the given size and style.

java.awt.font.LineMetrics

float getAscent()

gets the font ascent—the distance from the baseline to the tops of uppercase characters.

float getDescent()

gets the font descent—the distance from the baseline to the bottoms of descenders.

float getLeading()

gets the font leading—the space between the bottom of one line of text and the top of the next line.

float getHeight()

gets the total height of the font—the distance between the two baselines of text (descent + leading + ascent).

java.awt.Graphics

Font getFont()

void setFont(Font font)

gets or sets the current font. That font will be used for subsequent text-drawing operations.

Parameters: font A font

void drawString(String str, int x, int y)

draws a string in the current font and color.

Parameters: str The string to be drawn

x The x-coordinate of the start of the string

y The y-coordinate of the baseline of the string

java.awt.Graphics2D

FontRenderContext getFontRenderContext()

gets a font render context that specifies font characteristics in this graphics context.

void drawString(String str, float x, float y)

draws a string in the current font and color.

Parameters: str The string to be drawn

x The x-coordinate of the start of the string

y The y-coordinate of the baseline of the string

javax.swing.JComponent

FontMetrics getFontMetrics(Font f)

gets the font metrics for the given font. The FontMetrics class is a precursor to the LineMetrics class.

java.awt.FontMetrics

FontRenderContext getFontRenderContext()

gets a font render context for the font

Example Program:

```
import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;
import javax.swing.*;
public class FontTest
{
    public static void main(String[] args)
    {
        FontFrame frame = new FontFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
class FontFrame extends JFrame
{
    public FontFrame()
    {
        setTitle("FontTest");
    }
}
```

```

setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
// add component to frame
FontComponent component = new FontComponent();
add(component);
}
public static final int DEFAULT_WIDTH = 300;
public static final int DEFAULT_HEIGHT = 200;
}
/**
 * A component that shows a centered message in a box.
 */
class FontComponent extends JComponent
{
public void paintComponent(Graphics g)
{
Graphics2D g2 = (Graphics2D) g;
String message = "Hello, World!";
Font f = new Font("Serif", Font.BOLD, 36);
g2.setFont(f);
// measure the size of the message
FontRenderContext context = g2.getFontRenderContext();
Rectangle2D bounds = f.getStringBounds(message, context);
// set (x,y) = top-left corner of text
double x = (getWidth() - bounds.getWidth()) / 2;
double y = (getHeight() - bounds.getHeight()) / 2;
// add ascent to y to reach the baseline
double ascent = -bounds.getY();
double baseY = y + ascent;
// draw the message
g2.drawString(message, (int) x, (int) baseY);
g2.setPaint(Color.LIGHT_GRAY);
// draw the baseline
g2.draw(new Line2D.Double(x, baseY, x + bounds.getWidth(),
baseY));
// draw the enclosing rectangle
Rectangle2D rect = new Rectangle2D.Double(x, y,
bounds.getWidth(), bounds.getHeight());
g2.draw(rect);

```

```
}  
}  
}
```

Output:



5.5.3 Images

Once images are stored in local files or someplace on the Internet, you can read them into a Java application and display them on Graphics objects.

```
String filename = "...";  
Image image = ImageIO.read(new File(filename));
```

Now the variable image contains a reference to an object that encapsulates the image data.
You can display the image with the drawImage() method of the Graphics class.

```
public void paintComponent(Graphics g)  
{  
    ...  
    g.drawImage(image, x, y, null);  
}
```

5.5.3.1 Methods:

javax.imageio.ImageIO

staticBufferedImage read(File f)

staticBufferedImage read(URL u)

reads an image from the given file or URL.

java.awt.Graphics

boolean drawImage(Image img, int x, int y, ImageObserver observer)

draws an unscaled image. Note: This call may return before the image is drawn.

Parameters: **img** The image to be drawn

x The x-coordinate of the top-left corner

y The y-coordinate of the top-left corner

width The desired width of image

height The desired height of image

observer The object to notify of the progress of the rendering process (may be null)

void copyArea(int x, int y, int width, int height, int dx, int dy)

copies an area of the screen.

Parameters: **x** The x-coordinate of the top-left corner of the source area

y The y-coordinate of the top-left corner of the source area

width The width of the source area

height The height of the source area

dx The horizontal distance from the source area to the target area

dy The vertical distance from the source area to the target area

Example Program:

```
import java.awt.*;
import java.io.*;
import javax.imageio.*;
import javax.swing.*;
public class ImageTest
{
    public static void main(String[] args)
    {
        ImageFrame frame = new ImageFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

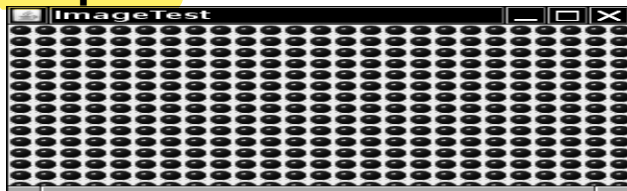
```

}
/**
 * A frame with an image component
 */
class ImageFrame extends JFrame
{
    public ImageFrame()
    {
        setTitle("ImageTest");
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
        // add component to frame
        ImageComponent component = new ImageComponent();
        add(component);
    }
    public static final int DEFAULT_WIDTH = 300;
    public static final int DEFAULT_HEIGHT = 200;
}
class ImageComponent extends JComponent
{
    public ImageComponent()
    {
        // acquire the image
        try
        {
            image = ImageIO.read(new File("blue-ball.gif"));
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
    public void paintComponent(Graphics g)
    {
        if (image == null) return;
        int imageWidth = image.getWidth(this);
        int imageHeight = image.getHeight(this);
        // draw the image in the top-left corner
        g.drawImage(image, 0, 0, null);
    }
}

```

```
// tile the image across the component
for (int i = 0; i * imageWidth <= getWidth(); i++)
for (int j = 0; j * imageHeight <= getHeight(); j++)
if (i + j > 0) g.copyArea(0, 0, imageWidth, imageHeight, i *
imageWidth, j * imageHeight);
}
private Image image;
}
```

Output:



5.6 Basics of event handling

5.6.1 Event Handlers

The operating system supports mouse & keyboard events to the pgm that are running. In event handling programming we need to write the code that constantly checks the event queue for what operating system reporting.

- * The information about the events is encapsulated in the event object.

- * All event object will be inherited from **java.util.EventObject**.

- * There are subclasses for each event type such as

- Action Event
- Window Event

- * Different event sources can produce different kind of event.

For eg,

A button can send action event object where as a window can send window event object.

The three participants in the event delegation model:

1. **event source**: class that broad cast event
2. **event listeners**: classes that receive event notification
3. **event objects**: class that describe the event.

5.6.2 HOW EVENT HANDLING WORKS IN awt CONTROL?

1. A listener object is an instance of a class that implements a special interface called a listener interface.
2. An event source is an object that can register listener object & send them to event object.
3. When event occurs, the event source sends out event objects to all register listeners.
4. The listener object will then use the information in the event object to determine their reaction to the event.

To register the listener objects with the source object, we can use `addEventListener()` method.

Example:

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

The listener object is notified whenever an "action event" occurs in the button. For button, an action event is a button click.

5.6.3 RELATIONSHIP B/W EVENT SOURCE & LISTENER:

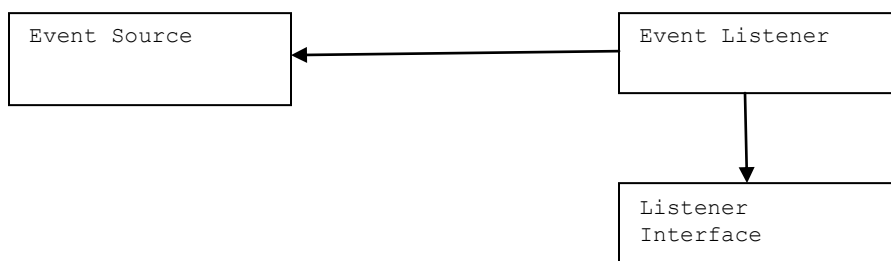


Fig 5.2 Event Handling

5.6.4 Methods

```
javax.swing.JButton  
JButton(String label)
```

JButton(Icon icon)

JButton(String label, Icon icon)

constructs a button.

java.awt.Container

Component add(Component c)

adds the component c to this container.

javax.swing.ImageIcon

ImageIcon(String filename)

constructs an icon whose image is stored in a file.

Example Program:

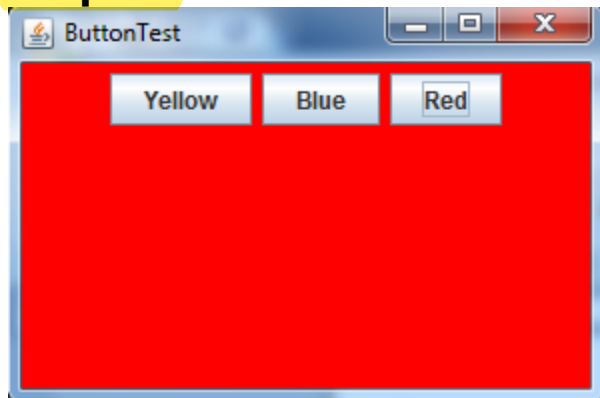
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ButtonTest
{
    public static void main(String[] args)
    {
        ButtonFrame frame = new ButtonFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
class ButtonFrame extends JFrame
{
    public ButtonFrame()
    {
        setTitle("ButtonTest");
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
        JButton yellowButton = new JButton("Yellow");
        JButton blueButton = new JButton("Blue");
        JButton redButton = new JButton("Red");
        JPanel buttonPanel = new JPanel();
        // add buttons to panel
        buttonPanel.add(yellowButton);
        buttonPanel.add(blueButton);
        buttonPanel.add(redButton);
        add(buttonPanel);
    }
}
```

```

ColorAction yellowAction = new ColorAction(Color.YELLOW);
ColorAction blueAction = new ColorAction(Color.BLUE);
ColorAction redAction = new ColorAction(Color.RED);
yellowButton.addActionListener(yellowAction);
blueButton.addActionListener(blueAction);
redButton.addActionListener(redAction);
}
private class ColorAction implements ActionListener
{
    public ColorAction(Color c)
    {
        backgroundColor = c;
    }
    public void actionPerformed(ActionEvent event)
    {
        buttonPanel.setBackground(backgroundColor);
    }
    private Color backgroundColor;
}
private JPanel buttonPanel;
public static final int DEFAULT_WIDTH = 300;
public static final int DEFAULT_HEIGHT = 200;
}

```

Output:



5.7 Adapter Classes

When the program user tries to close a frame window, the JFrame object is the source of a WindowEvent. If you want to catch that event, you must have an appropriate listener object and add it to the frame's list of window listeners.

```
WindowListener listener = . . .;  
frame.addWindowListener(listener);
```

The window listener must be an object of a class that implements the WindowListener interface.

There are actually seven methods in the WindowListener interface. The frame calls them as the responses to seven distinct events that could happen to a window. The names are self-explanatory, except that "iconified" is usually called "minimized" under Windows. Here is the complete WindowListener interface:

```
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);  
}
```

5.7.1 Methods

java.awt.event.WindowListener

void windowOpened(WindowEvent e)

is called after the window has been opened.

void windowClosing(WindowEvent e)

is called when the user has issued to close the window. Note that the window will close only if its hide or dispose method is called.

void windowClosed(WindowEvent e)

is called after the window has closed.

void windowIconified(WindowEvent e)

is called after the window has been iconified.

void windowActivated(WindowEvent e)

is called after the window has become active. Only a frame or dialog can be active. Typically, the window manager decorates the active window, for example, by highlighting the title bar.

void windowDeactivated(WindowEvent e)

is called after the window has become deactivated.

java.awt.event.WindowStateListener

void windowStateChanged(WindowEvent event)

is called after the window has been maximized, iconified, or restored to normal size.

java.awt.event.WindowEvent

int getNewState()

int getOldState()

returns the new and old state of a window in a window state change event. The

returned integer is one of the following values:

Frame.NORMAL

Frame.ICONIFIED

Frame.MAXIMIZED_HORIZ

Frame.MAXIMIZED_VERT

Frame.MAXIMIZED_BOTH

5.7.2 WindowListener Interface

we can define a class that implements the interface, add a call to `System.exit(0)` in the `windowClosing` method, and write do-nothing functions for the other six methods:

class Terminator implements WindowListener

{

public void windowClosing(WindowEvent e)

{

if (user agrees)

System.exit(0);

}

public void windowOpened(WindowEvent e) {}

public void windowClosed(WindowEvent e) {}

public void windowIconified(WindowEvent e) {}

public void windowDeiconified(WindowEvent e) {}

public void windowActivated(WindowEvent e) {}

```
public void windowDeactivated(WindowEvent e) {}
}
```

5.7.3 WindowAdapter Class

Terminator class extends the properties of WindowAdapter class to override windowClosing() method.

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

Explanation:

- Defines a class without a name that extends the WindowAdapter class
- Adds a windowClosing method to that anonymous class (as before, this method exits the program)
- Inherits the remaining six do-nothing methods from WindowAdapter
- Creates an object of this class; that object does not have a name, either
- Passes that object to the addWindowListener method

5.7.4 Creating and Registering an object

Now you can register an object of type Terminator as the event listener:

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

Whenever the frame generates a window event, it passes it to the listener object by calling one of its seven

methodstheWindowClosing method calls System.exit(0), terminating the application.

5.8 Actions

To encapsulate commands and to attach the commands to multiple event sources, we can use the swing package with some interfaces: the Action interface. An action is an object that encapsulates

- A description of the command (as a text string and an optional icon); and
- Parameters that are necessary to carry out the command (such as the requested color in our example).

5.8.1 Predefined Action Table Names

Name	Value
NAME	The name of the action;to be displayed on buttons and menu items
SMALL_ ICON	A place to store a small icon;for displaying in a button,menu item,or toolbar
SHORT_DESCRIPTION	A short description of the icon;for displaying in a tooltip
LONG_DESCRIPTION	A long description of the icon
MNEMONIC_KEY	A mnemonic abbreviation;for displaying in menu items
ACCELERATOR_KEY	A place to store an accelerator keystroke.No swing components uses this value
ACTION_COMMAND_KEY	Used in the registerKeyboardAction method
DEFAULT	Useful for catch-all property.

5.8.2 Input Maps

Flag	Invoke Action
WHEN_FOCUSED	Used to find when this

	component has keyboard focus
WHEN_ANCESTOR_OF_FOCUSED_COMPONENT	Used to find when this component contains the component that has keyboard focus
WHEN_IN_FOCUSED_WINDOW	When this component is contained in the same window as the component that has keyboard focus

5.8.3 Steps for performing actions on a button, menu item or keystroke

1. Implement a class that extends the **AbstractAction** class. You may be able to use the same class for multiple related actions.
2. Construct an **object** of the **action** class.
3. Construct a button or menu item from the action object. The constructor will read the label text and icon from the action object.
4. For actions that can be triggered by keystrokes, you have to carry out additional steps. First locate the top-level component of the window, such as a panel that contains all other components.
5. Then get the WHEN_ANCESTOR_OF_FOCUSED_COMPONENT input map of the top-level component. Make a KeyStroke object for the desired keystroke. Make an action key object, such as a string that describes your action. Add the pair (keystroke, action key) into the input map.
6. Finally, get the action map of the top-level component. Add the pair (action key, action object) into the map.

5.8.4 Methods

javax.swing.Action

boolean isEnabled()

void setEnabled(boolean b)

gets or sets the enabled property of this action.

void putValue(String key, Object value)

places a name/value pair inside the action object.

Parameters: key The name of the feature to store with the action object. This can be any string, but several names have predefined meanings.

value The object associated with the name.

Object getValue(String key)

returns the value of a stored name/value pair.

javax.swing.KeyStroke

static KeyStroke getKeyStroke(String description)

constructs a keystroke from a humanly readable description (a sequence of whitespace-delimited strings). The description starts with zero or more modifiers shift control ctrl meta alt altGraph and ends with either the string typed, followed by a one-character string (for example, "typed a"), or an optional event specifier (pressed—the default—or released), followed by a key code. The key code, when prefixed with VK_, should correspond to a KeyEvent constant; for example, "INSERT" corresponds to KeyEvent.VK_INSERT.

javax.swing.JComponent

ActionMap getActionMap() returns the map that associates action map keys (which can be arbitrary objects) with Action objects.

InputMap getInputMap(int flag)

gets the input map that maps key strokes to action map keys.

Parameters: flag A condition on the keyboard focus to trigger the action

Example Program:

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ActionTest
{
    public static void main(String[] args)
    {
        ActionFrame frame = new ActionFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
class ActionFrame extends JFrame
{
    public ActionFrame()
    {
        setTitle("ActionTest");
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
        buttonPanel = new JPanel();
        Action yellowAction = new ColorAction("Yellow", new
        ImageIcon("yellow-ball.gif"), Color.YELLOW);
        Action blueAction = new ColorAction("Blue", new
        ImageIcon("blue-ball.gif"), Color.BLUE);
        Action redAction = new ColorAction("Red", new ImageIcon("red-
        ball.gif"), Color.RED);
        buttonPanel.add(new JButton(yellowAction));
        buttonPanel.add(new JButton(blueAction));
        buttonPanel.add(new JButton(redAction));
        add(buttonPanel);
        InputMap imap =
        buttonPanel.getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);
        imap.put(KeyStroke.getKeyStroke("ctrl Y"), "panel.yellow");
        imap.put(KeyStroke.getKeyStroke("ctrl B"), "panel.blue");
        imap.put(KeyStroke.getKeyStroke("ctrl R"), "panel.red");
        ActionMap amap = buttonPanel.getActionMap();
        amap.put("panel.yellow", yellowAction);
        amap.put("panel.blue", blueAction);
    }
}

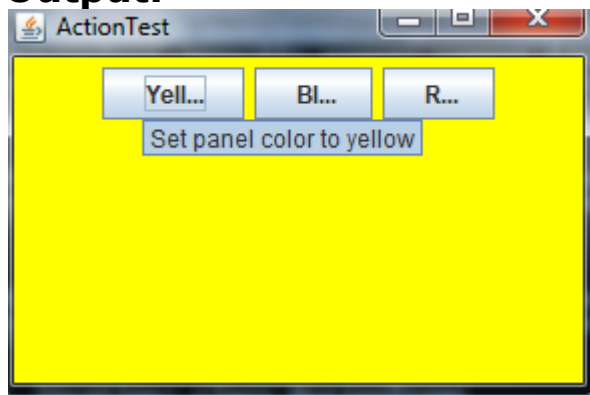
```

```

amap.put("panel.red", redAction);
}
public class ColorAction extends AbstractAction
{
public ColorAction(String name, Icon icon, Color c)
{
putValue(Action.NAME, name);
putValue(Action.SMALL_ICON, icon);
putValue(Action.SHORT_DESCRIPTION, "Set panel color to " +
name.toLowerCase());
putValue("color", c);
}
public void actionPerformed(ActionEvent event)
{
Color c = (Color) getValue("color");
buttonPanel.setBackground(c);
}
}
private JPanel buttonPanel;
public static final int DEFAULT_WIDTH = 300;
public static final int DEFAULT_HEIGHT = 200;
}

```

Output:



5.9 Mouse Events

When the user clicks a mouse button, three listener methods are called:

1. **mousePressed** - when the mouse is first pressed
2. **mouseReleased**- when the mouse is released
3. **mouseClicked**- When the mouse is clicked

As the mouse moves over a window, the window receives a steady stream of mouse movement events. There are two interfaces to handle it

- `MouseListener`
- `MouseMotionListener`

5.9.1 Sample Cursor Shapes















Icon	Constant	Icon	Constant
	<code>DEFAULT_CURSOR</code>		<code>NE_RESIZE_CURSOR</code>
	<code>CROSSHAIR_CURSOR</code>		<code>E_RESIZE_CURSOR</code>
	<code>HAND_CURSOR</code>		<code>SE_RESIZE_CURSOR</code>
	<code>MOVE_CURSOR</code>		<code>S_RESIZE_CURSOR</code>
	<code>TEXT_CURSOR</code>		<code>SW_RESIZE_CURSOR</code>
	<code>WAIT_CURSOR</code>		<code>W_RESIZE_CURSOR</code>
	<code>N_RESIZE_CURSOR</code>		<code>NW_RESIZE_CURSOR</code>

Fig 5.3 Cursor Shapes

5.9.2 Methods

java.awt.event.MouseEvent

int getX()

int getY()

Point getPoint()

returns the x- (horizontal) and y- (vertical) coordinate, or 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.)

java.awt.event.InputEvent

int getModifiersEx()

returns the extended or "down" modifiers for this event. The returned value is tested by the following mask

BUTTON1_DOWN_MASK
BUTTON2_DOWN_MASK
BUTTON3_DOWN_MASK
SHIFT_DOWN_MASK
CTRL_DOWN_MASK
ALT_DOWN_MASK
ALT_GRAPH_DOWN_MASK
META_DOWN_MASK

static String getModifiersExText(int modifiers)

returns a string such as "Shift+Button1" describing the extended or "down" modifiers in the given flag set.

Java.awt.Toolkit

**public Cursor createCustomCursor(Image image, Point
37eom.37t, String name)**

creates a new custom cursor object.

Java.awt.Component

public void setCursor(Cursor cursor)

sets the cursor image to the specified cursor.

Example Program:

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

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

```

{
public MouseFrame()
{
setTitle("MouseTest");
setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
MouseComponent component = new MouseComponent();
add(component);
}
public static final int DEFAULT_WIDTH = 300;
public static final int DEFAULT_HEIGHT = 200;
}
class MouseComponent extends Jcomponent
{
public MouseComponent()
{
squares = new ArrayList<Rectangle2D>();
current = null;
addMouseListener(new MouseHandler());
addMouseMotionListener(new MouseMotionHandler());
}
public void paintComponent(Graphics g)
{
Graphics2D g2 = (Graphics2D) g;
for (Rectangle2D r : squares)
g2.draw(r);
}
public Rectangle2D find(Point2D p)
{
for (Rectangle2D r : squares)
{
if (r.contains(p)) return r;
}
return null;
}
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();
}
public void remove(Rectangle2D s)
{
if (s == null) return;
if (s == current) current = null;
squares.remove(s);
repaint();
}
private static final int SIDELENGTH = 10;
private ArrayList<Rectangle2D> squares;
private Rectangle2D current;
private class MouseHandler extends MouseAdapter
{
public void mousePressed(MouseEvent event)
{
current = find(event.getPoint());
if (current == null) add(event.getPoint());
}
public void mouseClicked(MouseEvent event)
{
current = find(event.getPoint());
if (current != null && event.getClickCount() >= 2)
remove(current);
}
}
private class MouseMotionHandler implements
MouseMotionListener
{
public void mouseMoved(MouseEvent event)
{
if (find(event.getPoint()) == null)
setCursor(Cursor.getDefaultCursor());
}
}

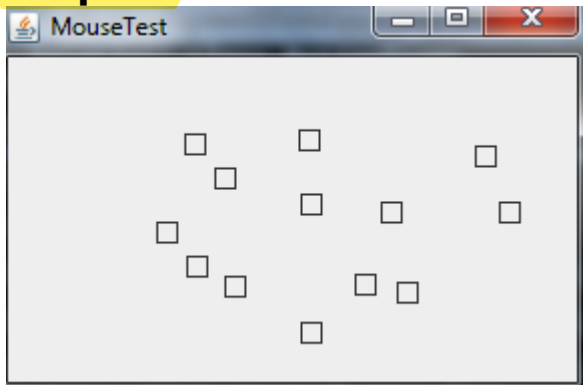
```

```

public void mouseDragged(MouseEvent event)
{
    if (current != null)
    {
        int x = event.getX();
        int y = event.getY();
        current.setFrame(x - SIDELENGTH / 2, y - SIDELENGTH / 2,
            SIDELENGTH, SIDELENGTH);
        repaint();
    }
}
}
}
}

```

Output:



5.10 AWT Event Hierarchy

The `EventObject` class has a subclass `AWTEvent`, which is the parent of all AWT event classes.

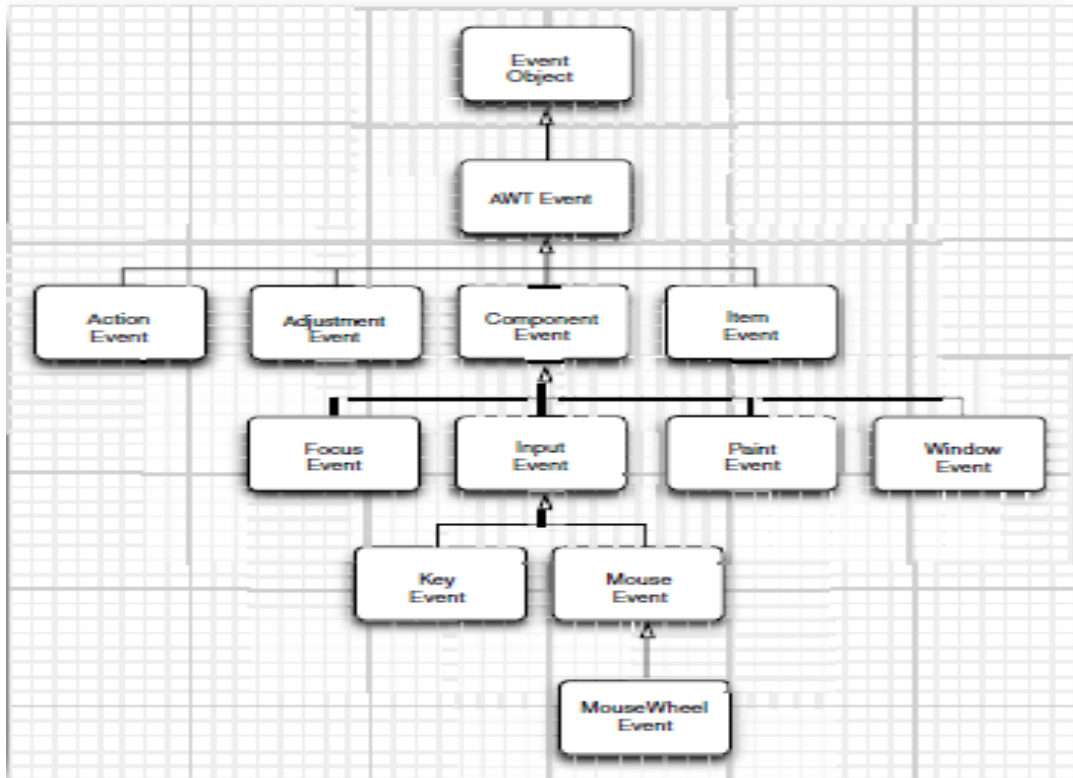


Fig 5.4 AWT Event Hierarchy

5.10.1 Semantic and Low-Level Events

Semantic event describes what the user is doing, such as “clicking that button”; Eg: Action Event is a semantic event.

Lower-level events are the events, if the user selects the button with the TAB key and activates it with the space bar. Eg: Dragging the mouse is a low-level event

5.10.2 Semantic Event Classes

ActionEvent (Class is used for a menu selection, , selecting a list item, a button click, or ENTER typed in a text field)

AdjustmentEvent (the user adjusted a scrollbar)

ItemEvent (the user made a selection from a set of checkbox or list items)

5.10.3 Lower-Level Event Classes

KeyEvent (a key was pressed or released)

MouseEvent (the mouse button was pressed, released, moved, or dragged)

MouseWheelEvent (the mouse wheel was rotated)

FocusEvent (a component got focus or lost focus)

WindowEvent (the window state changed)

5.10.4 Interfaces listen to these events

ActionListener, AdjustmentListener, FocusListener, ItemListener, KeyListener, MouseListener, MouseMotionListener, MouseWheelListener, WindowListener, WindowFocusListener, WindowStateListener

5.11 Introduction to Swing-Layout Management

The process of determining the size and position of components. Layout management can be done using absolute positioning

- Size and position of every component within the container must be specified.
- Does not adjust well when the top-level container is resized.
- Does not adjust well to differences between users and systems, such as font size.

Layout management is often performed using layout managers. Components can provide size and position hints to layout managers, but layout managers have the final say on the size and position of those components. Layout Manager is used to specify to a container how components within that container should be arranged.

5.11.1 Tasks of Layout Managers

1. A layout manager is a kind of “strategy” for placing components on the content pane or another component
2. In java, the content pane and any GUI component is a container.
3. A layout is chosen by calling the container’s setLayout method
4. Layouts are used to achieve some degree of platform independence and scalability.

5.11.2 Setting the Layout Manager

Method	Purpose
void setLayout(LayoutManager m)	Set the component's layout manager
LayoutManager getLayout()	Get the component's layout manager

Table 5.1 Setting the Layout Manager

The only containers whose layout managers you need to worry about are JPanel and content panes

5.11.3 JPanel Layout Manager

Each JPanel object is initialized to use FlowLayout, unless you specify differently when creating it. You can set a panel's layout manager using the JPanel constructor.

JPanel panel=new JPanel(new BorderLayout());

5.11.4 Content Pane Layout Manager

The default Layout manager for the content panes is BorderLayout. The default layout manager that a panel or content pane uses can be changed using the setLayout() method only after a container has been created.

Container contentPane=frame.getContentPane();
contentPane.setLayout(new FlowLayout());

5.11.5 Absolute Layout Manager

The container with no layout manager can be done by setting a container's layout property to null

setLayout(null);

5.11.6 Adding Components to a Container

When the components to a panel or content pane is added. The arguments you specify to the add method depend on the layout manager that the panel or content pane is using

Method	Purpose
Component add(Component comp)	Appends the specified component to the end of this container
void add(Component comp, Object constraints)	Adds the specified component to the end of this container

Table 5.2 Adding components to a container

5.11.7 Providing Size and Alignment

Method	Purpose
void setPreferredSize(Dimension) void setMaximumSize(Dimension) void setMinimumSize(Dimension)	Set the component's preferred, maximum, or minimum size, measured in pixels
Dimension getPreferredSize() Dimension getMaximumSize() Dimension getMinimumSize()	Get the preferred, maximum, or minimum size of the component, measured in pixels
void setAlignmentX(float) void setAlignmentY(float)	Set the alignment along the x or y axis
float getAlignmentX() float getAlignmentY()	Get the alignment of the component along the x or y axis

Table 5.3 Providing Size and Alignment

5.11.8 Putting space between components

Three factors influence the amount of space between visible components in a container

- 1.The layout manager
- 2.Invisible components
- 3.Empty borders

5.11.9 Setting the container's orientation

The property component-Orientation provides a way of indicating that a particular component should use

Method	Purpose
void applyComponentOrientation(ComponentOrientation)	Apply the component orientation to all containers
void setComponentOrientation(ComponentOrientation)	To set a container's orientation property

Table 5.4 Setting the container's orientation

The standard layout managers that support component orientation are FlowLayout, BorderLayout, BoxLayout, GridBagLayout and GridLayout.

5.11.10 Types of Layout Manager

The following classes implement the **java.awt.LayoutManager** interface

The AWT layout managers are

1. FlowLayout
2. BorderLayout
3. GridLayout
4. CardLayout and
5. GridBagLayout

Each component has a default layout manager, which remains in effect until the components setLayout method is called. Some of the defaults are

- 1.Content Pane-BorderLayout
- 2.JPanel-FlowLayout

3.Box-BoxLayout

5.11.10.1 FlowLayout Manager

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout manager for every panel. The components can be from the right to the left and right to the left. The manager allows to align the components.

Constructor

Constructor	Purpose
FlowLayout(int align)	Constructs a new FlowLayout with the specified alignment
FlowLayout(int align,int hgap,int vgap)	Creates a new flow layout with the given alignment and the specified horizontal and vertical gaps

Table 5.5 FlowLayout Constructors

Method1: (Using JPanel)

The default layout manager for JPanel is FlowLayout. So there is no need of setting the layout manager using setLayout() method

Example Program:

```
import java.awt.*;
import javax.swing.*;
public class FlowLayoutPanel extends JFrame
{
    public FlowLayoutPanel()
    {
        setTitle("FlowLayout For Panel");
        JPanel panel=new JPanel();
        JButton ok=new JButton("OK");
        JButton close=new JButton("Close");
        JButton reset=new JButton("Reset");
```

```

panel.add(ok);
panel.add(close);
panel.add(reset);
getContentPane().add(panel);
setSize(300,250);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);
setVisible(true);
}
public static void main(String[] args)
{
new FlowLayoutPanel();
}
}

```

Method2:

The default layout manager for content pane is BorderLayout. Creating a container with a layout manager for content pane involves the following steps. Set the container's layout manager to FlowLayout by

```

Container cont=getContentPane();
Cont.setLayout(new FlowLayout(FlowLayout.LEFT));

```

Example Program:

```

import java.awt.*;
import javax.swing.*;
public class FlowLayoutFrame extends JFrame
{
public FlowLayoutFrame()
{
setTitle("FlowLayout For Panel");
JPanel panel=new JPanel();
JButton ok=new JButton("OK");
JButton close=new JButton("Close");
JButton reset=new JButton("Reset");
Container cont=getContentPane();
cont.setLayout(new FlowLayout(FlowLayout.LEFT));
cont.add(ok);

```

```

cont.add(close);
cont.add(reset);
setSize(300,250);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);
setVisible(true);
}
public static void main(String[] args)
{
new FlowLayoutFrame();
}
}

```



5.11.10.2 BorderLayout Manager

BorderLayout is the default layout manager for the content pane of a

- JFrame
- JWindow
- JDialog
- JInternalFrame
- JApplet

Regions

It divides the container into five regions- North, West, South, East and Centre. Each region can have only one component. If more components are needed to put into a region, put a panel there with a manager of our choice.

The components in N, W, S, E regions get their preferred size. The component in the centre takes up the whole space left.

NORTH		
WEST	CENTER	EAST
SOUTH		

Constructor

Constructor	Description
BorderLayout()	Constructs a new BorderLayout
BorderLayout(int hgap,int vgap)	Constructs a new BorderLayout with the specified horizontal an vertical gaps between components

Table 5.6 BorderLayout Constructors

Methods

Methods	Purpose
setHgap(int)	Sets the horizontal gap between components
setVgap(int)	Sets the vertical gap between components

Table 5.7 BorderLayout Methods

Method1:

To set the layout,the setLayout() method.

Example Program:

```
import java.awt.*;
import javax.swing.*;
class BorderTest extends JFrame
{
BorderTest()
{
```

```

setTitle("FlowLayout For Panel");
JPanel panel=new JPanel();
JButton north=new JButton("North");
JButton east=new JButton("East");
JButton west=new JButton("West");
JButton south=new JButton("South");
JButton center=new JButton("Center");
JPanel content=new JPanel();
content.setLayout(new BorderLayout());
content.add(north,BorderLayout.NORTH);
content.add(south,BorderLayout.SOUTH);
content.add(east,BorderLayout.EAST);
content.add(west,BorderLayout.WEST);
content.add(center,BorderLayout.CENTER);
setContentPane(content);
setSize(300,250);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);
setVisible(true);
}
public static void main(String[] args)
{
new BorderTest();
}
}

```

Output:



5.11.10.3 GridLayout

A GridLayout places components in a rectangular grid whose cells all have the same size. Each component is sized to fill the cell. All components in the layout are given equal size. Any number of objects can be placed in a row or in a column.

Constructor

Constructor	Description
GridLayout(int rows,int cols)	Creates a grid layout with the specified number of rows and columns
GridLayout(int rows,int cols,int hgap,int vgap)	Creates a grid layout with the given number of rows and columns as well as the horizontal and vertical gaps

Table 5.8 GridLayout Constructors

Example Program

```
import java.awt.*;
import javax.swing.*;
class GridTest extends JFrame
{
    GridTest()
    {
        setTitle("FlowLayout For Panel");
        JPanel panel=new JPanel();
        JButton north=new JButton("North");
        JButton east=new JButton("East");
        JButton west=new JButton("West");
        JButton south=new JButton("South");
        JButton center=new JButton("Center");
        Container content=getContentPane();
        content.setLayout(new GridLayout(3,2));
        content.add(north);
        content.add(south);
        content.add(east);
        content.add(west);
        content.add(center);
    }
}
```

```

setSize(300,250);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);
setVisible(true);
}
public static void main(String[] args)
{
new GridTest();
}
}

```

Output:



5.12 Swing Components

Java Swing is a part of Java Foundation Classes (**JFC**) that is *used to create window-based applications*. It is built on the top of AWT (**Abstract Windowing Toolkit**) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The Java swing API classes are available in javax.swing package. It contains the following classes

1. JButton
2. JTextField
3. JTextArea
4. JColorChooser

5. JMenu
6. JCheckbox
7. JRadioButton etc

5.12.1 Difference between AWT and Swing

AWT	Swing
AWT components are platform-dependent .	Java swing components are platform-independent .
AWT components are heavyweight .	Swing components are lightweight .
AWT doesn't support pluggable look and feel .	Swing supports pluggable look and feel .
AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
AWT doesn't support the concept of MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC .

5.12.1 Text Fields

The JTextField object is a text component that allows to type of a single line text. It is inherited from JTextComponent class.

Constructors:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.

JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Table 5.9 JTextField Constructors

Methods:

Method	Description
void addActionListener(ActionListener l)	This method is used to add action listener to receive the actions given by the Text Field.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	This method is used to remove action listener when the action is no longer needed.

Table 5.10 JTextField Methods

Example Program:

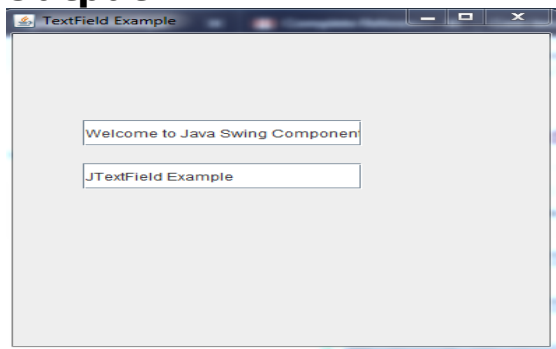
```
import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
{
JFrame f= new JFrame("TextField Example");
JTextField t1,t2;
t1=new JTextField("Welcome to Javatpoint.");
t1.setBounds(50,100, 200,30);
```

```

t2=new JTextField("AWT Tutorial");
t2.setBounds(50,150, 200,30);
f.add(t1);
f.add(t2);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}

```

Output:



5.12.2 Text Areas

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

Constructors

Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.
JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays

	no text initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

Table 5.11 JTextArea Constructors

Methods

Method	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

Table 5.12 JTextArea Methods

Example Program:

```
import javax.swing.*;
class TextFieldExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("TextField Example");
        JTextField t1,t2;
        t1=new JTextField("Welcome to Javatpoint.");
        t1.setBounds(50,100, 200,30);
        t2=new JTextField("AWT Tutorial");
        t2.setBounds(50,150, 200,30);
        f.add(t1); f.add(t2);
        f.setSize(400,400);
    }
}
```



```
f.setLayout(null);
f.setVisible(true);
}
}
```

Output:



5.12.3 Buttons

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

Constructors

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Table 5.13 JButton Constructor

Methods

Method	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text

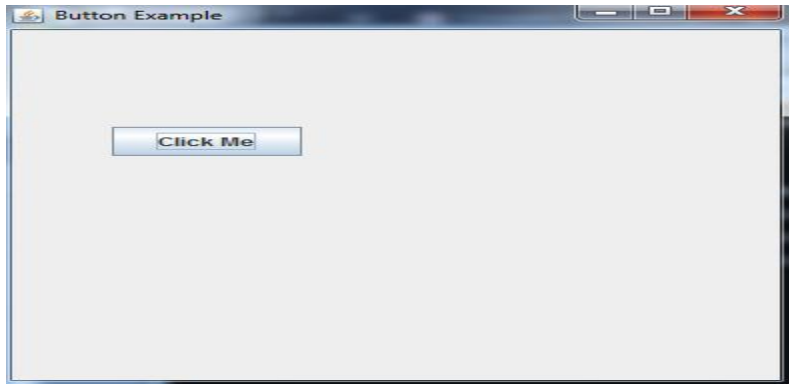
	of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Table 5.14 JButton Methods

Example Program:

```
import javax.swing.*;
public class ButtonExample
{
    public static void main(String[] args)
    {
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton("Click Me");
        b.setBounds(50,100,95,30);
        f.add(b);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output:



5.12.4 Check Boxes

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

Constructors

Constructor	Description
JCheckBox()	Creates an initially unselected check box button with no text, no icon.
JCheckBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

Table 5.15 JCheckBox Constructors

Methods

Method	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.

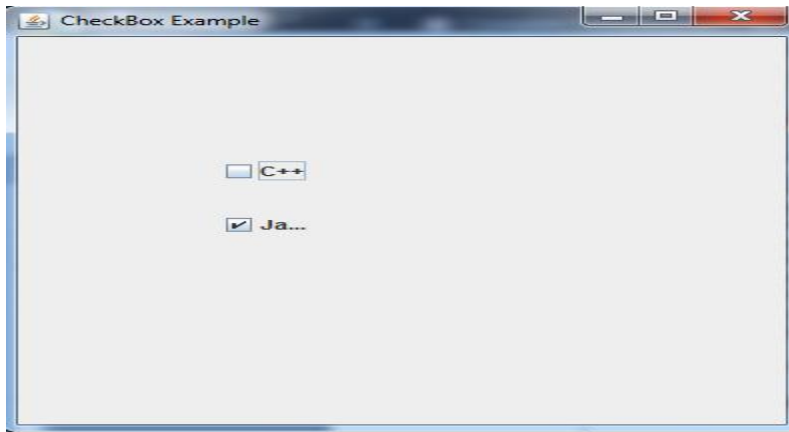
protected paramString()	String	It returns a string representation of this JCheckBox.
----------------------------	--------	---

Table 5.16 JCheckBox Methods

Example Program

```
import javax.swing.*;
public class CheckBoxExample
{
    CheckBoxExample()
    {
        JFrame f= new JFrame("CheckBox Example");
        JCheckBox checkBox1 = new JCheckBox("C++");
        checkBox1.setBounds(100,100, 50,50);
        JCheckBox checkBox2 = new JCheckBox("Java", true);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1);
        f.add(checkBox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new CheckBoxExample();
    }
}
```

Output:



5.12.5 Radio Buttons

The `JRadioButton` class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

Constructors

Constructor	Description
<code>JRadioButton()</code>	Creates an unselected radio button with no text.
<code>JRadioButton(String s)</code>	Creates an unselected radio button with specified text.
<code>JRadioButton(String s, boolean selected)</code>	Creates a radio button with the specified text and selected status.

Table 5.17 JRadioButton Constructors

Methods

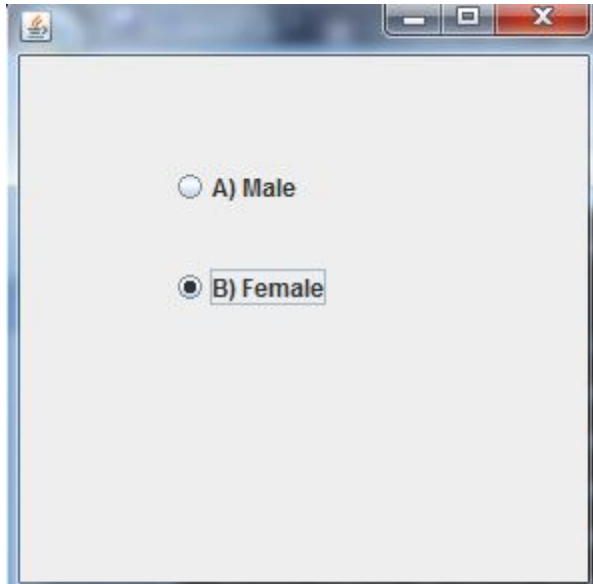
Method	Description
<code>void setText(String s)</code>	It is used to set specified text on button.
<code>String getText()</code>	It is used to return the text of the button.
<code>void setEnabled(boolean b)</code>	It is used to enable or disable the button.
<code>void setIcon(Icon b)</code>	It is used to set the specified Icon on the button.

Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Table 5.18 JRadioButton Methods

Example Program

```
import javax.swing.*;
public class RadioButtonExample
{
    JFrame f;
    RadioButtonExample()
    {
        f=new JFrame();
        JRadioButton r1=new JRadioButton("A) Male");
        JRadioButton r2=new JRadioButton("B) Female");
        r1.setBounds(75,50,100,30);
        r2.setBounds(75,100,100,30);
        ButtonGroup bg=new ButtonGroup();
        bg.add(r1);
        bg.add(r2);
        f.add(r1);
        f.add(r2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
        new RadioButtonExample();
    }
}
```



5.12.6 Lists

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

Constructors

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Table 5.19 JList Constructors

Methods

Method	Description
--------	-------------

Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a list of items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

Table 5.20 JRadioButton Methods

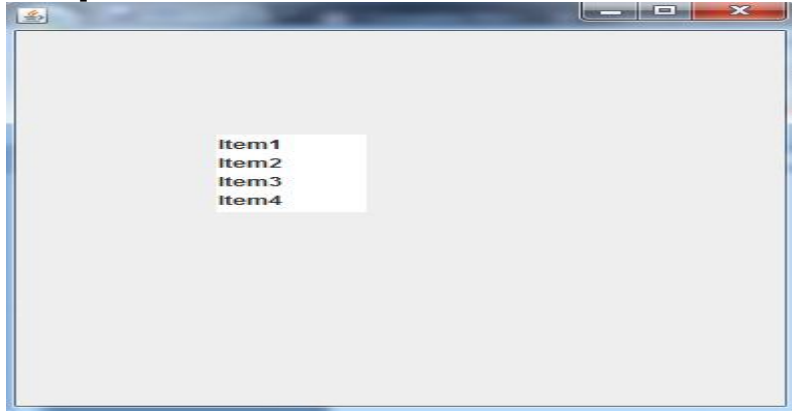
Example Program

```
import javax.swing.*;
public class ListExample
{
    ListExample()
    {
        JFrame f= new JFrame();
        DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("Item1");
        l1.addElement("Item2");
        l1.addElement("Item3");
        l1.addElement("Item4");
        JList<String> list = new JList<>(l1);
        list.setBounds(100,100, 75,75);
        f.add(list);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new ListExample();
    }
}
```



```
}  
}
```

Output:



5.12.7 choices

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

Choice Class Declaration

public class Choice extends Component implements ItemSelectable, Accessible

Example Program

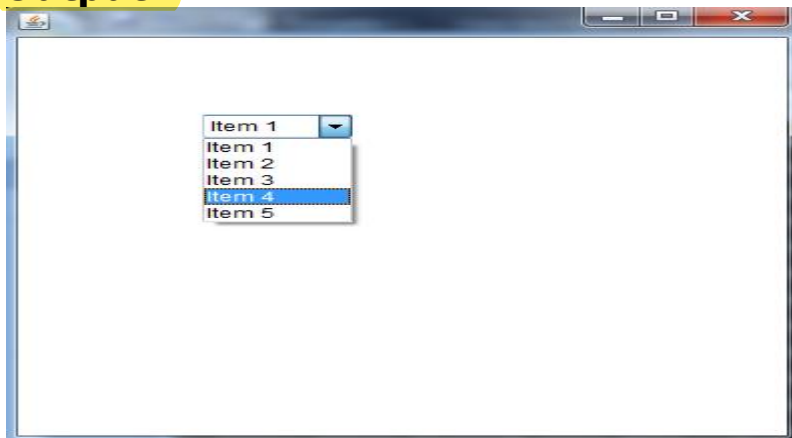
```
import java.awt.*;  
public class ChoiceExample  
{  
    ChoiceExample()  
    {  
        Frame f= new Frame();  
        Choice c=new Choice();  
        c.setBounds(100,100, 75,75);  
        c.add("Item 1");  
        c.add("Item 2");  
        c.add("Item 3");  
        c.add("Item 4");  
        c.add("Item 5");  
    }  
}
```

```

f.add(c);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[])
{
new ChoiceExample();
}
}

```

Output:



5.12.8 Scrollbars

The object of JScrollbar class is used to add horizontal and vertical scrollbar. It is an implementation of a scrollbar. It inherits JComponent class.

Constructors

Constructor	Description
JScrollbar()	Creates a vertical scrollbar with the initial values.
JScrollbar(int orientation)	Creates a scrollbar with the specified orientation and the initial values.
JScrollbar(int	Creates a scrollbar with the

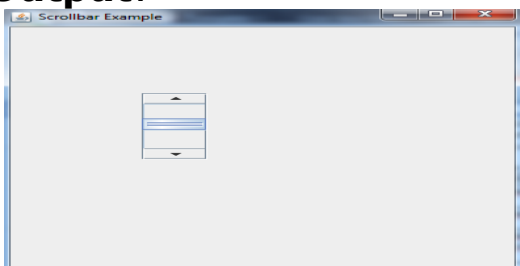
orientation, int value, int extent, int min, int max)	specified orientation, value, extent, minimum, and maximum.
---	---

Table 5.21 JScrollBar Constructors

Example Program

```
import javax.swing.*;
class ScrollBarExample
{
    ScrollBarExample()
    {
        JFrame f= new JFrame("Scrollbar Example");
        JScrollBar s=new JScrollBar();
        s.setBounds(100,100, 50,100);
        f.add(s);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new ScrollBarExample();
    }
}
```

Output:



5.12.9 Windows

The class **JWindow** is a container that can be displayed but does not have the title bar or window-management buttons.

Constructors

Constructor	Description
<code>JWindow()</code>	Creates a window with no specified owner.
<code>JWindow(Frame owner)</code>	Creates a window with the specified owner frame.
<code>JWindow(GraphicsConfiguration gc)</code>	Creates a window with the specified GraphicsConfiguration of a screen device.
<code>JWindow(Window owner)</code>	Creates a window with the specified owner window.
<code>JWindow(Window owner, GraphicsConfiguration gc)</code>	Creates a window with the specified owner window and GraphicsConfiguration of a screen device.

Table 5.22 JWindow Constructors

Methods

Method	Description
<code>protected void windowInit()</code>	Called by the constructors to init the JWindow properly.
<code>void repaint(long time, int x, int y, int width, int height)</code>	Repaints the specified rectangle of this component within time milliseconds.
<code>void setLayout(LayoutManager manager)</code>	Sets the LayoutManager.
<code>void update(Graphics g)</code>	Calls paint(g).

Table 5.23 JWindow Methods

Example Program:

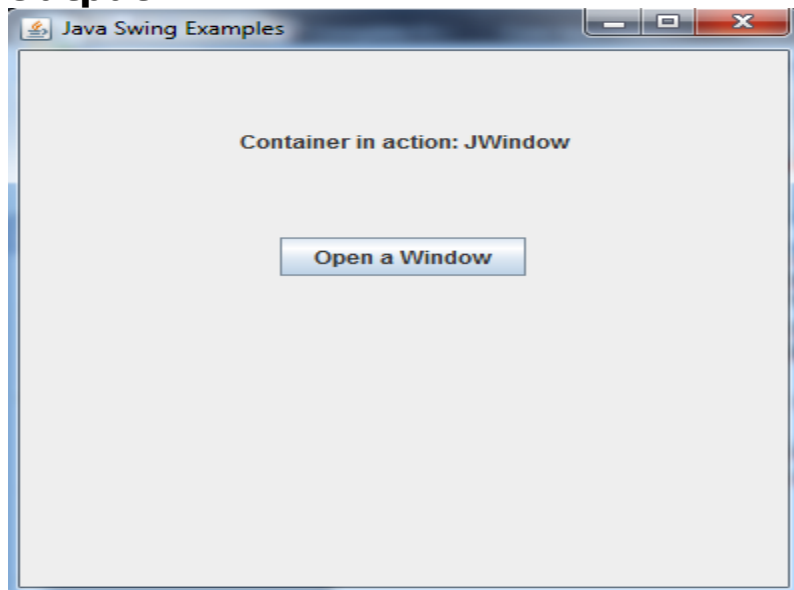
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SwingContainerDemo
{
    private JFrame mainFrame;
    private JLabel headerLabel;
    private JLabel statusLabel;
    private JPanel controlPanel;
    private JLabel msglabel;
    public SwingContainerDemo()
    {
        prepareGUI();
    }
    public static void main(String[] args)
    {
        SwingContainerDemo swingContainerDemo = new
        SwingContainerDemo();
        swingContainerDemo.showJWindowDemo();
    }
    private void prepareGUI()
    {
        mainFrame = new JFrame("Java Swing Examples");
        mainFrame.setSize(400,400);
        mainFrame.setLayout(new GridLayout(3, 1));
        mainFrame.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent windowEvent)
            {
                System.exit(0);
            }
        });
        headerLabel = new JLabel("", JLabel.CENTER);
        statusLabel = new JLabel("",JLabel.CENTER);
        statusLabel.setSize(350,100);
        msglabel = new JLabel("Welcome to TutorialsPoint SWING
        Tutorial.", JLabel.CENTER);
```

```

controlPanel = new JPanel();
controlPanel.setLayout(new FlowLayout());
mainFrame.add(headerLabel);
mainFrame.add(controlPanel);
mainFrame.add(statusLabel);
mainFrame.setVisible(true);
}
private void showJWindowDemo()
{
headerLabel.setText("Container in action: JWindow");
final MessageWindow window = new MessageWindow(
mainFrame, "Welcome to Tutorialspoint SWING Tutorial.");
JButton okButton = new JButton("Open a Window");
okButton.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
window.setVisible(true);
statusLabel.setText("A Window shown to the user.");
}
}
}

```

Output:



5.12.10 Menus

To create the menus in the window or frame, we can use JMenu class.

The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

JMenuBar class declaration

```
public class JMenuBar extends JComponent implements MenuElement, Accessible
```

JMenu class declaration

```
public class JMenu extends JMenuItem implements MenuElement, Accessible
```

JMenuItem class declaration

```
public class JMenuItem extends AbstractButton implements Accessible, MenuElement
```

Example Program

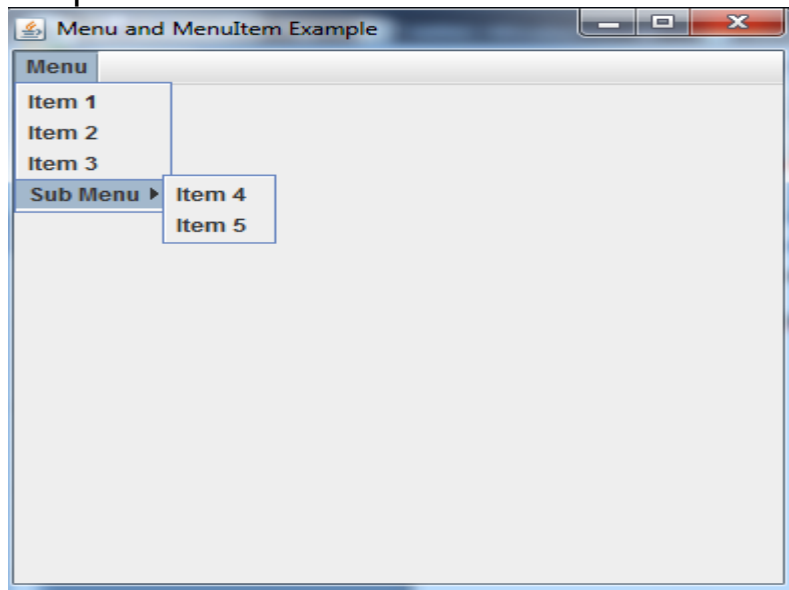
```
import javax.swing.*.*;
class MenuExample
{
    JMenu menu, submenu;
    JMenuItem i1, i2, i3, i4, i5;
    MenuExample(){
        JFrame f= new JFrame("Menu and MenuItem Example");
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");
        submenu=new JMenu("Sub Menu");
        i1=new JMenuItem("Item 1");
        i2=new JMenuItem("Item 2");
        i3=new JMenuItem("Item 3");
        i4=new JMenuItem("Item 4");
        i5=new JMenuItem("Item 5");
        menu.add(i1);
        menu.add(i2);
```

```

menu.add(i3);
submenu.add(i4);
submenu.add(i5);
menu.add(submenu);
mb.add(menu);
f.setJMenuBar(mb);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}
}

```

Output:



5.12.11 Dialog Boxes

The JDialog control is looking like a window with a border and a title. It is used to take the input from the user. It inherits the Dialog class.

Constructors

Constructor	Description
JDialog()	Creates a modeless dialog without a title and without a specified Frame owner.
JDialog(Frame owner)	creates a modeless dialog with given frame as its owner and an empty title.
JDialog(Frame owner, String title, boolean modal)	Modal Dialog is created with the specified title, owner Frame and modality.

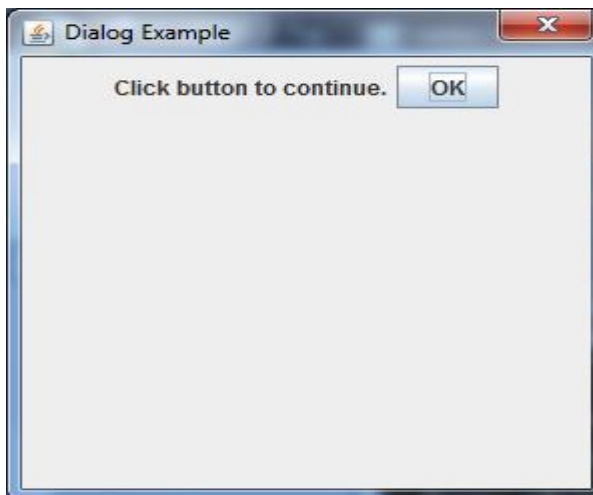
Table 5.24 JDialog Constructors

Example Program

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DialogExample
{
    private static JDialog d;
    DialogExample()
    {
        JFrame f= new JFrame();
        d = new JDialog(f , "Dialog Example", true);
        d.setLayout( new FlowLayout() );
        JButton b = new JButton ("OK");
        b.addActionListener ( new ActionListener()
        {
            public void actionPerformed((ActionEvent e )
            {
                DialogExample.d.setVisible(false);
            }
        });
        d.add( new JLabel ("Click button to continue."));
        d.add(b);
        d.setSize(300,300);
        d.setVisible(true);
    }
}
```

```
public static void main(String args[])
{
new DialogExample();
}
}
```

Output:



2 Marks Questions and Answers

1.How are frames created in java?

- A top-level window (that is, a window that is not contained inside another window) is called a frame in Java.
- The AWT library has a class, called Frame, for this top level.
- The Swing version of this class is called JFrame and extends the Frame class.

2. What is adapter class?

- When the program user tries to close a frame window, the JFrame object is the source of a

- WindowEvent. If you want to catch that event, you must have an appropriate listener object
- and add it to the frame's list of window listeners.
WindowListener listener = . . .;
frame.addWindowListener(listener);
- ComponentAdapter, FocusAdapter, KeyAdapter, MouseAdapter, ContainerAdapter and MouseMotionAdapter, WindowAdapter are the adapter classes.
- When the user closes the frame window, the JFrame object is the source of the window event class.

3. Give the value for the following predefined actions.

- SMALL ICON
- MNEMONIC KEY

a.SMALL ICON:

Public abstract java.lang.string smallicon returns small icons for this filter if present.
 Default:" "

b.MNEMONIC KEY:

Static functional mnemonic_key

The key used for storing an integer that corresponds to one of the key event keycodes.The value is commonly used to specify mnemonics.

4.State the functions of action in event handler.

An action can be used to separate functionally & state from a component.An action object is an action listener that provides not only action-event handling, but also centralized handling of the state of action-event-fixing components such as tool bar buttons,menu items,common buttons&text fields.

5.What are the methods under WindowListener Interface?

There are seven methods under WindowListener. A frame calls the seven

methods and window will react for those methods.

```
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);
}
```

6. What is reflection?

- Reflection is the process of collecting details about both pre-defined and user-defined classes.
- This can be done by importing the package of `java.lang.reflect.*`;
- It can analyse the capabilities of class at run-time. It can return constructor, methods, variables of a particular class at run-time. It can give the details of object at run-time.

7. Explain toString method of object class?

`String toString()` - defines the string representation of an object.

Where, `String` is the return type for the method `toString()`.

8. Give the syntax of drawImage() and copyArea() methods.

`drawImage()`

Syntax:

1. `void drawImage(Image img, int x, int y, ImageObserver onserver)`

Definition:

`drawImage()` method is used to Place the image on the screen.

Parameters:

x- top x co-ordinate of image.

y-top y co-ordinate of image.

Observer- to note the progress of the image.

Img- image to be drawn.

```
2.void drawImage(Image img, int x, int y,int width, int height, ImageObserver observer)
```

Parameter:

Img- image to be drawn

x- top x co-ordinate of image.

y- top y co-ordinate of image.

Observer-note the progress of image.

width-adjust the width of the image.

height-adjust the height of the image.

copyArea():

syntax:

```
void copyArea(int x, int y, int z, int height, int dx, int dy)
```

9. What are the methods under reflection used to analyse the capabilities of classes?

To get the details about the classes there are two methods.

They are,

1.class getClass()

2.class getSuperClass()

getClass()- This method is used to return the name of the class.

getSuperClass()- This method is used to return the name of super class.

Field[] getFields()-This method is used to return the fields of the class

Method[] getMethods()-This method is used to return the methods of the class

Constructor[] getConstructors()-This method is used to return the constructors of the class

10.What is Class.forName() does and how it is useful?

It loads the class into the ClassLoader. It returns the Class. Using that you can get the instance (—class-instance.newInstance()).

11. What are classes in Graphics Programming?

JCanvas. The usual drawing operations can be directly done on a JCanvas.

JBox. A container for other Swing components. A JBox works almost like a Box object in Swing, but offers easier control over the layout of components in the box. This means that a JBox can be used as an alternative to a number of different layout managers.

JEventQueue. An event-handler for the usual events in Swing programs. The events are placed in a queue and the program can then extract them. Using a JEventQueue is an alternative to writing programs in an event-driven style.

12.Explain the types of events occur in a graphic based Program?

- Events from pressing a button.
- Events from state based components.
- Events from text component.
- Events from a canvas.
- Timers.
- Menus.
- Window events.

13. Explain Component with example.

A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Components are the buttons, checkboxes, and scrollbars of a typical graphical user interface.

14. Explain about graphics in the Java 2 platform.

```
public void paintComponent(Graphics g) {  
    // Clear background if opaque  
    super.paintComponent(g);  
    // Cast Graphics to Graphics2D  
    Graphics2D g2d = (Graphics2D)g;  
    // Set pen parameters  
    g2d.setPaint(fillColorOrPattern);  
}
```

```

g2d.setStroke(penThicknessOrPattern);
// Allocate a shape
SomeShape s = new SomeShape(...);
// Draw shape
g2d.draw(s); // outline
g2d.fill(s); // solid
}

```

15. What is an Event Listener?

An event listener is an object to which a component has delegated the task of handling a particular kind of event. In other words, if you want an object in your program to respond to another object which is generating events, you must register your object with the event generating object, and then handle the generated events when they come.

16. What is an adapter?

It converts the existing interfaces to a new interface to achieve compatibility and reusability of the unrelated classes in one application. Also known as Wrapper pattern.

17. What is an action in Java?

To encapsulate commands and to attach the commands to multiple event sources, we can use the swing package with some interfaces: the Action interface. An action is an object that encapsulates

- A description of the command (as a text string and an optional icon); and
- Parameters that are necessary to carry out the command (such as the requested color in our example).

18. Difference between Awt and Swing Package.

AWT	Swing
AWT components are platform-dependent .	Java swing components are platform-independent .
AWT components are	Swing components are

heavyweight.	lightweight.
AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
AWT doesn't support the concept of MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC.

19. What is the use of Mouse Event?

When the user uses the mouse (or similar input device) to interact with a component then the mouse event occurs. The cursor enters or exits a component's on screen area if the user presses or releases one of the mouse buttons.

20.What is ActionEvent?

ActionEvent class extends the properties from the **AWTEvent** class. It indicates the component-defined events occurred. The generated event is passed to every **EventListener** objects that receives such types of events using the **addActionListener()** method of the object.

21. What is AdjustmentEvent?

AdjustmentEvent class extends the properties from the **AWTEvent** class. When the **Adjustable Value** is changed then the event is generated.

22. What is ComponentEvent?

ComponentEvent class also extends the properties from the **AWTEvent** class. when the object moved, changed,it creates the

low-level event to indicate its states . This class only performs the notification about the state of the object.

23. What is ContainerEvent?

The **ContainerEvent** class extends the properties from the **ComponentEvent** class. This is a low-level event when the container's changes the contents because of adding or removing the components.

24. What is FocusEvent?

The **FocusEvent** class also extends the properties from the **ComponentEvent** class. This class indicates about the focus where the focus has gained or lost by the object. The `addFocusListener()` method is used, when the focus events occurs.

25. What is Input Event?

The **Input Event** class also extends the properties from the **Component Event** class. This event class handles all the component-level input events. This class acts as a root class for all component-level input events.

26. What is Item Event?

The **Item Event** class extends the properties from the **AWT Event** class. The **Item Event** class handles all the indication about the selection of the object i.e. whether selected or not.

27. What is KeyEvent?

KeyEvent class extends the properties from the **InputEvent** class. The **KeyEvent** class handles all the key operations in the application if you press any key for any purposes of the object then the generated event gives the information about the pressed key. This type of events check whether the pressed key left key or right key, 'A' or 'a' etc.

28. What is Swing?

Swing is basically a set of customizable graphical components whose *look-and-feel* can be dictated at runtime.

Pre-built *look-and-feel*: Windows, Macintosh, Metal on a *any* platform

Personal *look-and-feel* also definable

29. What are the Features of Swing?

- Do not depend on native peers to render themselves.
- Simplified graphics to paint on screen
- Similar behaviour across all platforms
- Portable look and feel
- Only a few top level containers not lightweight.

30. Explain about the Swing Components?

- **JApplet**-simple extension of java.applet.Applet class for use when swing programs are designed to be used in web browser or appletviewer
- **JDialog** -replacement for java.awt.Dialog provides information and simple user prompts. JOptionPane class is a new Swing alternative for easy creation of simple dialogs.
- **JFrame** -Most common container
- **JWindow** A window container

31. Explain about the Button.

The Button class creates pushbuttons with text labels. Java doesn't have a built-in class that supports buttons with images, we can also do that by extending the Canvas class. The most common usage is to simply create one with a specified label, drop it in a window, then watch for action events to see if it has been pressed. For instance, to create a button and add it to the current window, you'd do something like the following:

```
Button button = new Button("...");  
add(button);
```

32. Explain about the Layout Management?

Some containers, such as JTabbedPane and JSplitPane, define a particular arrangement for their children. Other containers such as JPanel (and JFrame, JDialog, and other top-level containers

that use JPanel as their default content pane) do not define any particular arrangement. When working with containers of this type, you must specify a LayoutManager object to arrange the children within the container.

13 Marks Questions

1. Explain the AWT Event handling in detail.
2. Give the methods available in graphics for COLOR.
3. List the methods available to draw shapes.
4. Explain frame components with examples.
5. Elaborately discuss the components of graphics programming. Illustrate the basic of event handling.
6. Describes the types of layout management and swing components briefly.
7. Explain in detail about Graphic Programming.
8. Write a Java program to illustrate Mouse Events.
9. What is swing? Write a program for a simple calculator using swing.
10. Discuss the adapter classes using example.
11. Explain about the concepts of creating and positioning of frame.
12. Define Event handling. Write a program to handle a button event.

15 Marks Questions

1. What are called as event listeners? Create an Applet for "Patient Registration System" with Grid Layout and do the action events for mouse and key events.
2. Explain in detail about AWT classes with syntax and examples.
3. Write a program using various AWT controls to design a form which accepts the details of a user.
4. Explain the Action Event and the Action Listener class of Event package.

