MODULE V

JAVA PROGRAMMING

EVENT AND LISTENER (JAVA EVENT HANDLING)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

Java Event Handling Code

We can put the event handling code into one of the following places:

1. Within class
2. Other class
3. Anonymous class

Java event handling by implementing ActionListener

```
import java.awt.*;

import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
```

```
TextField tf;
AEvent(){
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
//register listener
b.addActionListener(this);//passing current instance
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```

public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above positi

on of the component it may be button, textfield etc.

Java AWT Button

A button is basically a control component with a label that generates an event when pushed. The **Button** class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

When we press a button and release it, AWT sends an instance of **ActionEvent** to that button by calling **processEvent** on the button. The **processEvent** method of the button receives the all the events, then it passes an action event by calling its own method **processActionEvent**. This method passes the action event on to action listeners that are interested in the action events generated by the button.

To perform an action on a button being pressed and released, the **ActionListener** interface needs to be implemented. The registered new listener can receive events from the button by calling **addActionListener** method of the button. The Java application can use the button's action command as a messaging protocol.

AWT Button Class Declaration

1. **public class** Button **extends** Component **implements** Accessible

Button Class Constructors

Following table shows the types of Button class constructors

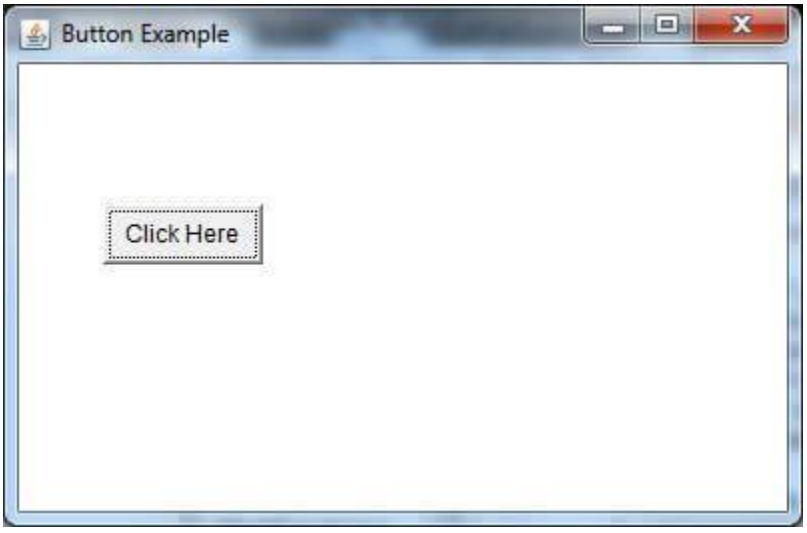
Sr. no.	Constructor	Description
1.	Button( )	It constructs a new button with an empty string i.e. it has no label.
2.	Button (String text)	It constructs a new button with given string as its label.

Sr. no.	Method	Description
1.	void setText (String text)	It sets the string message on the button
2.	String getText()	It fetches the String message on the button.
3.	void setLabel (String label)	It sets the label of button with the specified string.
4.	String getLabel()	It fetches the label of the button.
5.	void addNotify()	It creates the peer of the button.
6.	AccessibleContext getAccessibleContext()	It fetched the accessible context associated with the button.
7.	void addActionListener(ActionListener l)	It adds the specified action listener to get the action events from the button.
8.	String getActionCommand()	It returns the command name of the action event fired by the button.
9.	ActionListener[ ] getActionListeners()	It returns an array of all the action listeners registered on the button.
10.	T[ ] getListeners(Class listenerType)	It returns an array of all the objects currently

Java AWT Button Example
Example 1:

ButtonExample.java

```
import java.awt.*;
public class ButtonExample {
public static void main (String[] args) {
    // create instance of frame with the label
Frame f = new Frame("Button Example");
    // create instance of button with label
    Button b = new Button("Click Here");
    // set the position for the button in frame
    b.setBounds(50,100,80,30);
    // add button to the frame
    f.add(b);
    // set size, layout and visibility of frame
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```



Java AWT Label

The object of the Label class is a component for placing text in a container. It is used to display a single line of **read only text**. The text can be changed by a programmer but a user cannot edit it directly.

It is called a passive control as it does not create any event when it is accessed. To create a label, we need to create the object of **Label** class.

AWT Label Class Declaration

1. **public class** Label **extends** Component **implements** Accessible

AWT Label Fields

The java.awt.Component class has following fields:

1. **static int LEFT:** It specifies that the label should be left justified.
2. **static int RIGHT:** It specifies that the label should be right justified.
3. **static int CENTER:** It specifies that the label should be placed in center.

Sr. no.	Constructor	Description
1.	Label()	It constructs an empty label.
2.	Label(String text)	It constructs a label with the given string (left justified by default).
3.	Label(String text, int alignement)	It constructs a label with the specified string and the specified alignment.

Sr. no.	Method name	Description
1.	void setText(String text)	It sets the texts for label with the specified text.
2.	void setAlignment(int alignment)	It sets the alignment for label with the specified alignment.
3.	String getText()	It gets the text of the label
4.	int getAlignment()	It gets the current alignment of the label.
5.	void addNotify()	It creates the peer for the label.
6.	AccessibleContext getAccessibleContext()	It gets the Accessible Context associated with the label.
7.	protected String paramString()	It returns the string the state of the label.

LabelExample.java

```
import java.awt.*;

public class LabelExample {
public static void main(String args[]){

   // creating the object of Frame class and Label class
   Frame f = new Frame ("Label example");
   Label l1, l2;

   // initializing the labels
   l1 = new Label ("First Label.");
   l2 = new Label ("Second Label.");

   // set the location of label
   l1.setBounds(50, 100, 100, 30);
   l2.setBounds(50, 150, 100, 30);
```

```
        // adding labels to the frame
        f.add(l1);
        f.add(l2);

        // setting size, layout and visibility of frame
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    }
```

Java AWT TextField

The object of a **TextField** class is a text component that allows a user to enter a single line text and edit it. It inherits **TextComponent** class, which further inherits **Component** class.

When we enter a key in the text field (like key pressed, key released or key typed), the event is sent to **TextField**. Then the **KeyEvent** is passed to the registered **KeyListener**. It can also be done using **ActionEvent**; if the ActionEvent is enabled on the text field, then the ActionEvent may be fired by pressing return key. The event is handled by the **ActionListener** interface.

AWT TextField Class Declaration

1. **public class** TextField **extends** TextComponent

TextField Class constructors

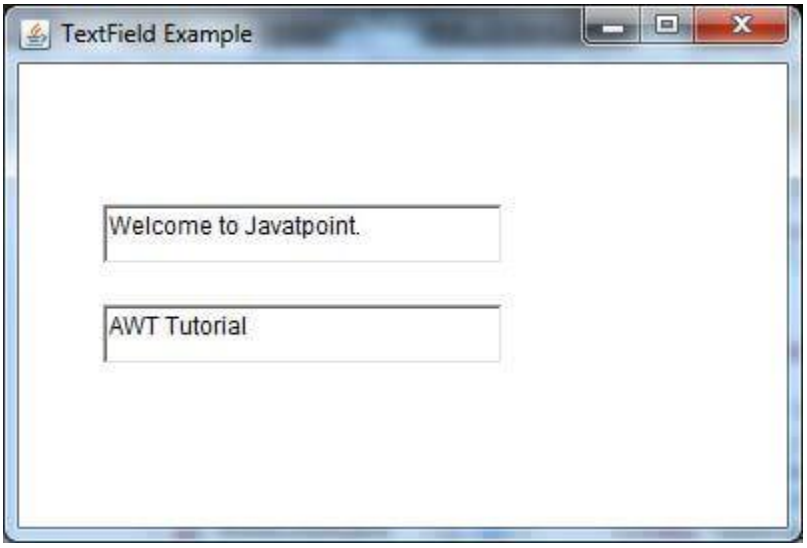
Sr. no.	Constructor	Description
1.	TextField()	It constructs a new text field component.
2.	TextField(String text)	It constructs a new text field initialized with the given string text to be displayed.
3.	TextField(int columns)	It constructs a new textfield (empty) with given number of columns.
4.	TextField(String text, int columns)	It constructs a new text field with the given text and given number of columns (width).

TextField Class Methods

Sr. no.	Method name	Description
1.	void addNotify()	It creates the peer of text field.
2.	boolean echoCharIsSet()	It tells whether text field has character set for echoing or not.
3.	void addActionListener(ActionListener l)	It adds the specified action listener to receive action events from the text field.
4.	ActionListener[] getActionListeners()	It returns array of all action listeners registered on text field.
5.	AccessibleContext getAccessibleContext()	It fetches the accessible context related to the text field.
6.	int getColumns()	It fetches the number of columns in text field.
7.	char getEchoChar()	It fetches the character that is used for echoing.
8.	Dimension getMinimumSize()	It fetches the minimum dimensions for the text

```
// importing AWT class
import java.awt.*;
public class TextFieldExample1 {
    // main method
    public static void main(String args[]) {
    // creating a frame
    Frame f = new Frame("TextField Example");
    // creating objects of textfield
    TextField t1, t2;
    // instantiating the textfield objects
    // setting the location of those objects in the frame
t1 = new TextField("Welcome to Javatpoint.");
```

```
        t1.setBounds(50, 100, 200, 30);
        t2 = new TextField("AWT Tutorial");
        t2.setBounds(50, 150, 200, 30);
        // adding the components to frame
        f.add(t1);
        f.add(t2);
        // setting size, layout and visibility of frame
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



AWT TextArea

The object of a TextArea class is a multiline region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

The text area allows us to type as much text as we want. When the text in the text area becomes larger than the viewable area, the scroll bar appears automatically which helps us to scroll the text up and down, or right and left.

AWT TextArea Class Declaration

1. **public class** TextArea **extends** TextComponent

Fields of TextArea Class

The fields of java.awt.TextArea class are as follows:

- **static int SCROLLBARS_BOTH** - It creates and displays both horizontal and vertical scrollbars.
- **static int SCROLLBARS_HORIZONTAL_ONLY** - It creates and displays only the horizontal scrollbar.
- **static int SCROLLBARS_VERTICAL_ONLY** - It creates and displays only the vertical scrollbar.
- **static int SCROLLBARS_NONE** - It doesn't create or display any scrollbar in the text area.

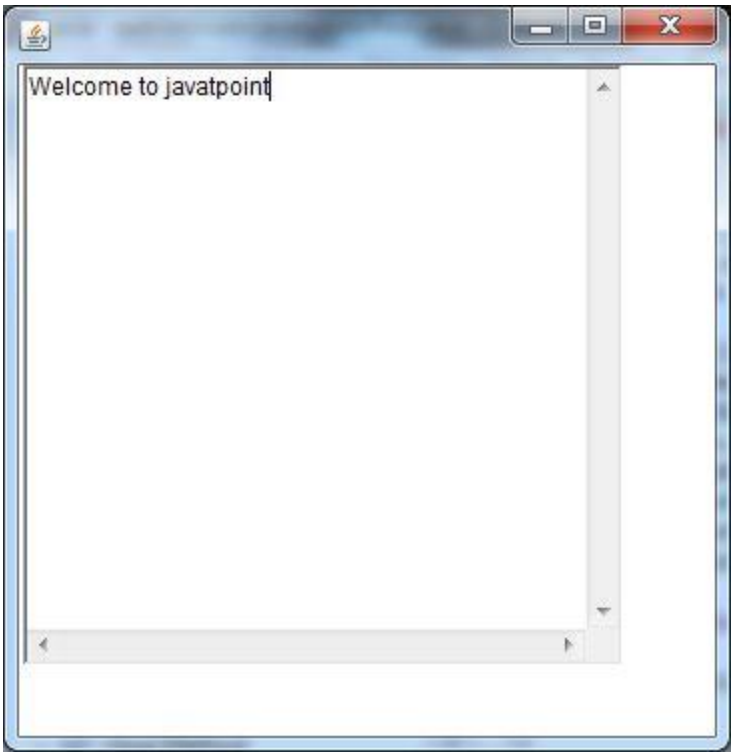
Class constructors:

Sr. no.	Constructor	Description
1.	TextArea()	It constructs a new and empty text area with no text in it.
2.	TextArea (int row, int column)	It constructs a new text area with specified number of rows and columns and empty string as text.
3.	TextArea (String text)	It constructs a new text area and displays the specified text in it.
4.	TextArea (String text, int row, int column)	It constructs a new text area with the specified text in the text area and specified number of rows and columns.
5.	TextArea (String text, int row, int column, int scrollbars)	It construcst a new text area with specified text in text area and specified number of rows and columns and visibility.

Sr. no.	Method name	Description
1.	void addNotify()	It creates a peer of text area.
2.	void append(String str)	It appends the specified text to the current text of text area.
3.	AccessibleContext getAccessibleContext()	It returns the accessible context related to the text area
4.	int getColumns()	It returns the number of columns of text area.
5.	Dimension getMinimumSize()	It determines the minimum size of a text area.
6.	Dimension getMinimumSize(int rows, int columns)	It determines the minimum size of a text area with the given number of rows and columns.
7.	Dimension getPreferredSize()	It determines the preferred size of a text area.
8.	Dimension preferredSize(int rows, int columns)	It determines the preferred size of a text area with given number of rows and columns.
9.	int getRows()	It returns the number of rows of text area.

```
//importing AWT class
import java.awt.*;
public class TextAreaExample
{
// constructor to initialize
    TextAreaExample() {
// creating a frame
        Frame f = new Frame();
// creating a text area
            TextArea area = new TextArea("Welcome to javatpoint");
// setting location of text area in frame
        area.setBounds(10, 30, 300, 300);
```

```
// adding text area to frame
        f.add(area);
// setting size, layout and visibility of frame
        f.setSize(400, 400);
        f.setLayout(null);
        f.setVisible(true);
     }
// main method
public static void main(String args[])
{
   new TextAreaExample();
}
}
```



Java AWT Checkbox

The Checkbox 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".

AWT Checkbox Class Declaration

1. **public class** Checkbox **extends** Component **implements** ItemSelectable, Accessible

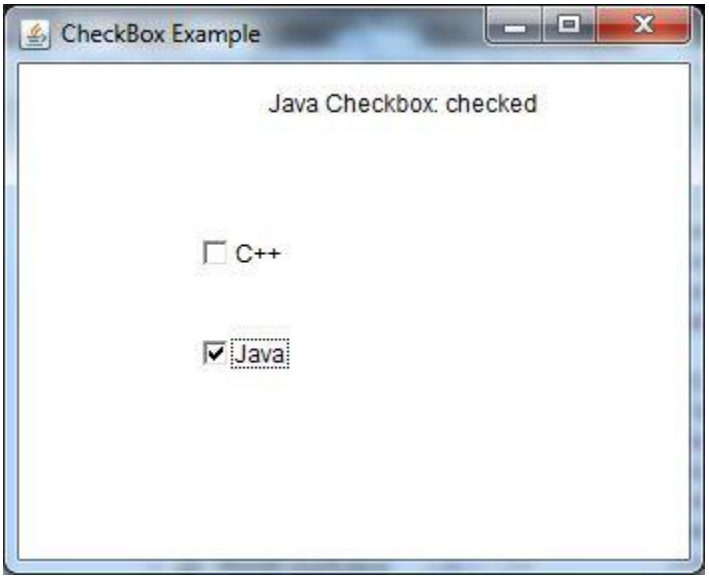
Checkbox Class Constructors

Sr. no.	Constructor	Description
1.	Checkbox()	It constructs a checkbox with no string as the label.
2.	Checkbox(String label)	It constructs a checkbox with the given label.
3.	Checkbox(String label, boolean state)	It constructs a checkbox with the given label and sets the given state.
4.	Checkbox(String label, boolean state, CheckboxGroup group)	It constructs a checkbox with the given label, set the given state in the specified checkbox group.
5.	Checkbox(String label, CheckboxGroup group, boolean state)	It constructs a checkbox with the given label, in the given checkbox group and set to the specified state.

```
// importing AWT class
import java.awt.*;
public class CheckboxExample1
{
// constructor to initialize
    CheckboxExample1() {
// creating the frame with the title
      Frame f = new Frame("Checkbox Example");
// creating the checkboxes
       Checkbox checkbox1 = new Checkbox("C++");
       checkbox1.setBounds(100, 100,  50, 50);
       Checkbox checkbox2 = new Checkbox("Java", true);
// setting location of checkbox in frame
checkbox2.setBounds(100, 150,  50, 50);
// adding checkboxes to frame
```

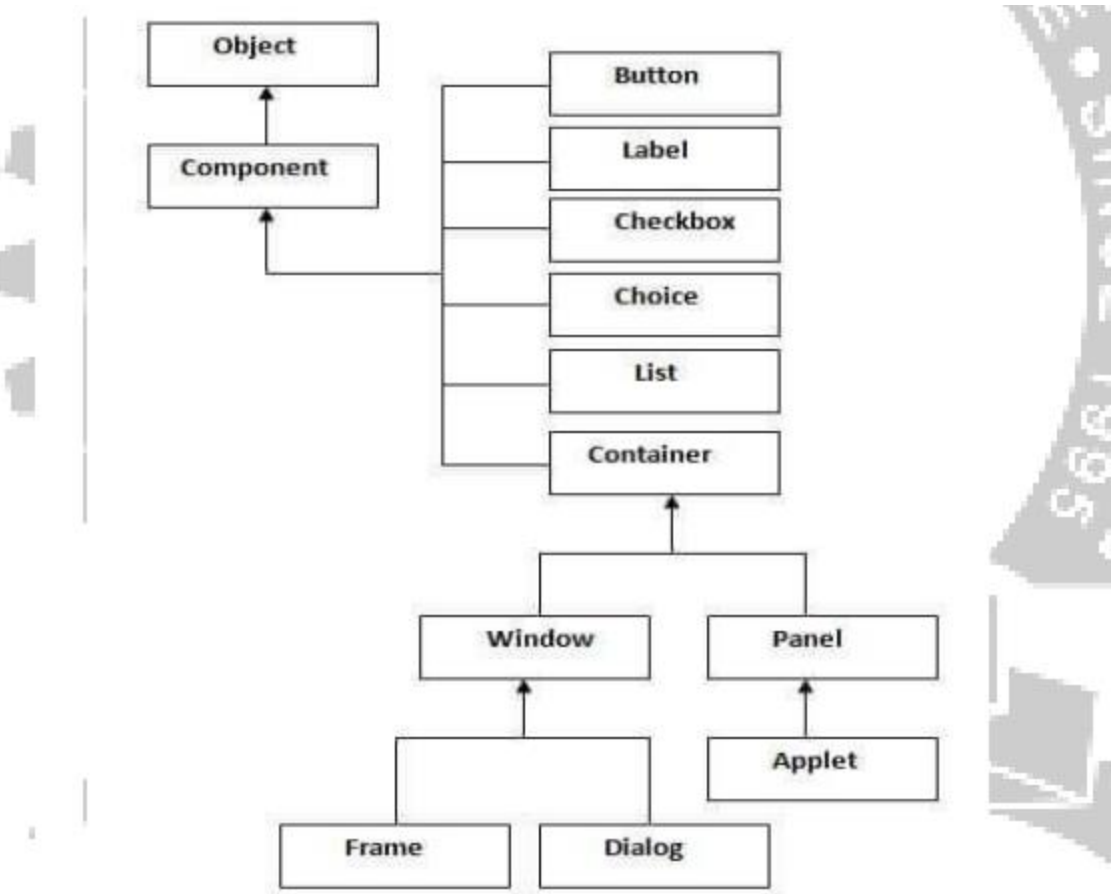
```
        f.add(checkbox1);
        f.add(checkbox2);

// setting size, layout and visibility of frame
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
// main method
public static void main (String args[])
{
   new CheckboxExample1();
}
}
```



STRUCTURE OF AWT

Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc. The hierarchy of Java AWT classes are given below.



Component: Component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. For examples buttons, checkboxes, list and scrollbars of a graphical user interface.

Container: The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window: The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel: The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame: The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc

layout managers.

Layout manager are used to arrange components within the container. It automatically places the control at a particular position within window. LayoutManager is an interface implemented by all the classes of layout managers. Following are the different classes of Layout manager in AWT

1. BorderLayout

2. CardLayout

3. FlowLayout

4. GridLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. It is used add method to add the component at specified region. Components of the BorderLayout Manager:

• BorderLayout.NORTH

• BorderLayout.SOUTH

• BorderLayout.EAST

• BorderLayout.WEST

• BorderLayout.CENTER

BorderLayout Constructor: - BorderLayout()It is used to create a new border layout with no gaps between components.

The CardLayout manages the components in form of stack and provides visibility to only one component at a time. It treats each component as a card that is why it is known as CardLayout. CardLayoutConstructors CardLayout(): creates a card layout with zero horizontal and vertical gap.

• CardLayout(int hgap, int vgap): creates a card layout with the given horizontal and vertical gap.

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel. Following are the possible values in FlowLayout manager. LEFT, RIGHT, CENTER, LEADING, TRAILING FlowLayout Constructors

• FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.

• FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.

• FlowLayout(int align, int hgap, int vgap): creates a flow layout with the given alignment and the given horizontal and vertical gap.

The GridLayout manager is used to arrange the components in the two-dimensional grid. Each component is displayed in a rectangle.

GridLayout Constructors

• GridLayout(): creates a grid layout with one column per component in a row.

• GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.

• GridLayout(int rows, int columns, int hgap, int vgap): creates a grid layout with the given rows and columns along with given horizontal and vertical gaps

Which are the graphics methods to draw a line and rectangle?

java.awt.Graphics class The Graphics class is the abstract super class for all graphics contexts which allow an application to draw onto components. Graphics methods for drawing line, rectangle etc., are shown below. For each drawing method that requires a width and height parameter, the width and height must be nonnegative values.

public void drawLine(int x1, int y1, int x2, int y2) Draws a line between the point (x1, y1) and the point (x2, y2).

public void drawRect(int x, int y, int width, int height) Draws a rectangle of the specified width and height. The top-left corner of the rectangle has the coordinates (x, y). Only the outline of the

rectangle is drawn using the Graphics object's colorthe body of the rectangle is not filled with this color.

public void fillRect(int x, int y, int width, int height) Draws a filled rectangle with the specified width and height. The top-left corner of the rectangle has the coordinate (x, y). The rectangle is filled with the Graphics object's color.

public void drawOval(int x, int y, int width, int height) Draws an oval in the current color with the specified width and height. The bounding rectangle's top-left corner is at the coordinates (x, y). The oval touches all four sides of the bounding rectangle at the center of each side . Only the outline of the shape is drawn.

public void fillOval(int x, int y, int width, int height) Draws a filled oval in the current color with the specified width and height. The bounding rectangle's top-left corner is at the coordinates (x, y). The oval touches all four sides of the bounding rectangle at the center of each side.

public void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight) Draws a rectangle with rounded corners in the current color with the specified width and height.

The arcWidth and arcHeight determine the rounding of the corners . Only the outline of the shape is drawn.

public void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight) Draws a filled rectangle with rounded corners in the current color with the specified width and height.

The arcWidth and arcHeight determine the rounding of the corners

18. Write a java program to drawoval, rectangle etc.

```
import java.applet.*;
import java.awt.*;
public class Shapes extends Applet
{
public void paint(Graphics g)
{
```

```
 g.drawLine(30,300,200,10);
 g.drawOval(150,50,100,100);
g.drawRect(400,50,200,100);
 }
}
```