



Object Oriented Development

ICT2123

Introduction to Graphic Programming

P.H.P. Nuwan Laksiri
Department of ICT
Faculty of Technology
University of Ruhuna

Lesson 05 – OOD Notes

What we discuss Today

- Graphic Programming
- GUI Toolkits in Java
 - AWT Components
 - Container, Window, Panel, Frame
 - AWT Controls
- Layout Managers
 - Border, Flow, Grid
- Custom drawings
 - Working with graphics
 - Working with colors
 - Working with fonts
- Few Examples
- Introduction to Graphic Coordinate System
- Introduction to Event handling
- Introduction to Java Applets

Graphic Programming

- Graphic programming include (such as drawing graphs, charts, drawings...etc.)
 - Use of standard GUI components
 - Use of custom drawing
- A graphics context provides the capabilities of drawing on the screen.
- The graphics context maintains states such as the color and font used in drawing, as well as interacting with the underlying operating system to perform the drawing.

Graphic Programming

- In Java, custom painting is done via the `java.awt.Graphics` class
- `java.awt.Graphics` Class
 - Manages a graphics context, and provides a set of device-independent methods for drawing texts, figures and images on the screen on different platforms

GUI Toolkits in Java

- Belongs to standard GUI components
- There are two sets of Java APIs for graphics programming:
 - AWT (Abstract Windowing Toolkit)
 - Swing
- We can reuse the graphics classes provided in JDK for constructing our own Graphical User Interface (GUI) application rather than re-invent the wheels.
- Java Graphics APIs - AWT and Swing - provide a huge set of reusable GUI components, such as
 - button, text field, label, choice, panel and frame for building GUI applications.

GUI Toolkits in Java

- Other than AWT/Swing Graphics APIs provided in JDK, others have also provided Graphics APIs that work with Java, such as
 - Eclipse's Standard Widget Toolkit (SWT),
 - Google Web Toolkit (GWT),

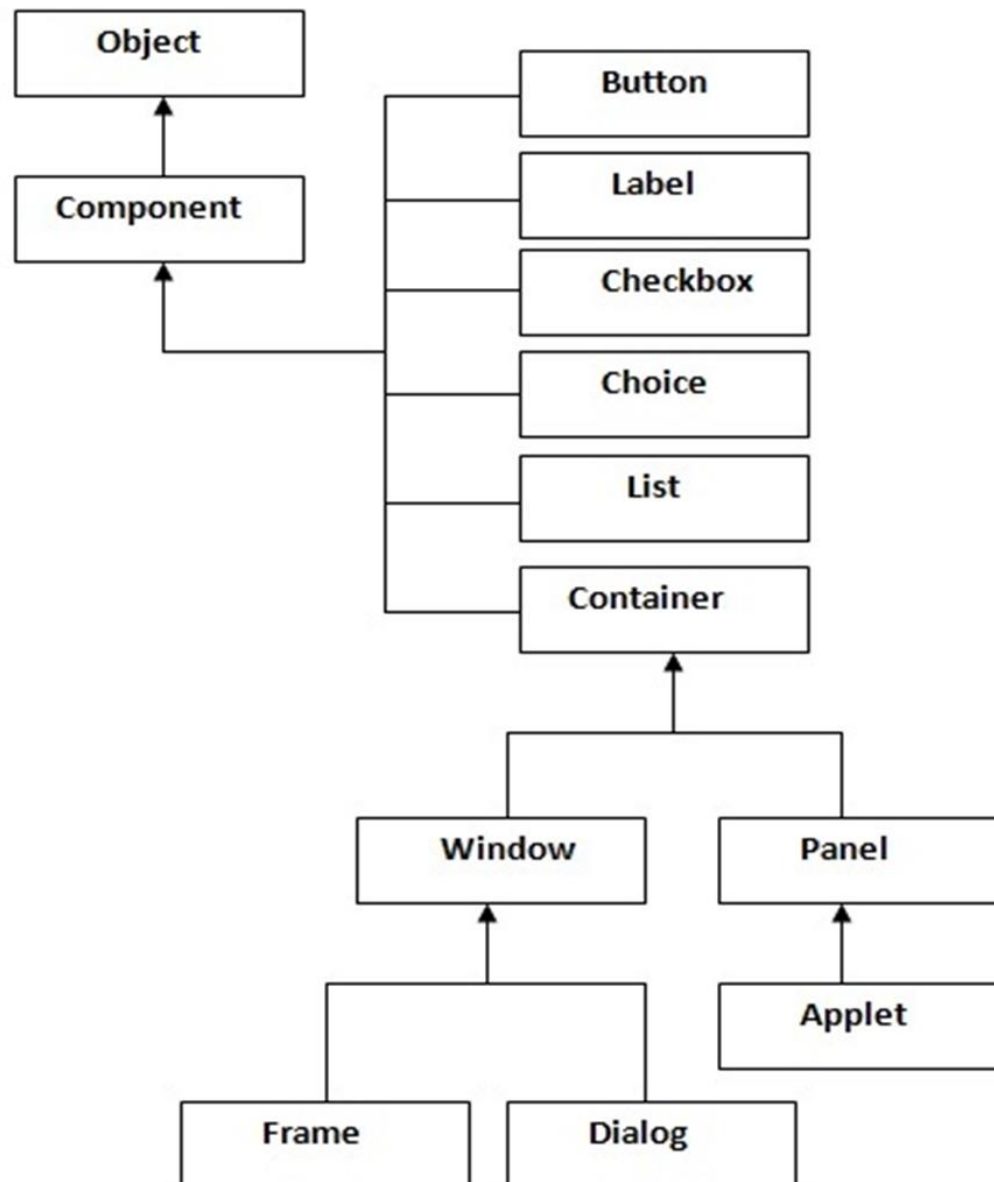
AWT

- Java AWT (Abstract Windowing Toolkit) is an API to develop GUI or window-based application in java.
- Java AWT components are platform-dependent
 - i.e. components are displayed according to the view of operating system.
- AWT is heavyweight
 - i.e. its components uses the resources of system.
- Provide a huge set of reusable GUI components.
- You can simply reuse these classes rather than re-invent the wheels.
- The java.awt package provides classes for AWT api such as
 - TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

AWT

- The java.awt package contains the core AWT graphics classes:
 - GUI Component classes (such as Button, TextField, and Label),
 - GUI Container classes (such as Frame, Panel, Dialog and ScrollPane),
 - Layout managers (such as FlowLayout, BorderLayout and GridLayout),
 - Custom graphics classes (such as Graphics, Color and Font).
- The java.awt.event package supports event handling:
 - Event classes (such as ActionEvent, MouseEvent, KeyEvent and WindowEvent),
 - Event Listener Interfaces (such as ActionListener, MouseListener, KeyListener and WindowListener),
 - Event Listener Adapter classes (such as MouseAdapter, KeyAdapter, and WindowAdapter).

AWT Hierarchy



Component

- Component is an abstract class in Java, so that you can only create objects belonging to its subclasses.
- Subclasses that represent basic GUI components are:
 - Button,
 - ScrollBar,
 - TextField,
 - Label, CheckBox,
 - Canvas, and so on.
- The Component class defines methods to manage events

Useful Methods of Component class

Method	Description
<code>public void add(Component c)</code>	inserts a component on this component.
<code>public void setSize(int width,int height)</code>	sets the size (width and height) of the component.
<code>public void setLayout(LayoutManager m)</code>	defines the layout manager for the component.
<code>public void setVisible(boolean status)</code>	changes the visibility of the component, by default false.

Container and Window

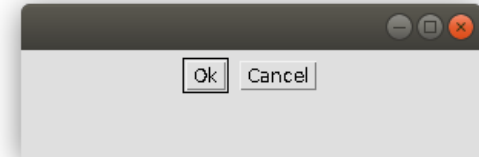
- Container
 - The classes that extends Container class are known as container such as Frame, Dialog and Panel.
- Window
 - The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panels

- A Panel is a rectangular Container whose sole purpose is to hold and manage components within a GUI.
- The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.
- Panel, which is only used for grouping components
- Panel is the Supper class of applet which simplify implementation of container.

Panel- Example

```
6 package paneldemo;
7 import java.awt.*;
8 /* @author nuwan */
9 public class MyPanel {
10
11     public MyPanel() {
12         Panel myPanel = new Panel();
13
14         myPanel.add(new Button("Ok"));
15         myPanel.add(new Button("Cancel"));
16
17         Frame myFrame = new Frame();
18         myFrame.setSize(300,100);
19         myFrame.setLocation(500, 500);
20         myFrame.add(myPanel);
21         myFrame.setVisible(true);
22     }
23 }
24
```



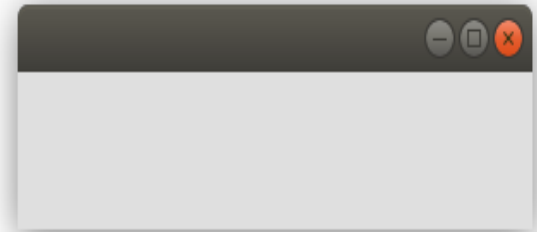
```
6 package paneldemo;
7 /* @author nuwan */
8 public class PanelDemo {
9     public static void main(String[] args) {
10         MyPanel myPanel = new MyPanel();
11     }
12 }
13
```

Frame

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

Frame - Example

```
6   package framedemo;
7
8   import java.awt.*;
9
10  /* @author nuwan */
11
12  public class FrameDemo {
13      public static void main(String[] args) {
14          Frame myFrame = new Frame();
15          myFrame.setSize(300,100);
16          myFrame.setLocation(500, 500);
17          myFrame.setVisible(true);
18      }
19
20  }
21
```



AWT Controls

- Labels

- This class is a Component which displays a single line of text.
- `Label aLabel = new Label("Enter password:");`

- Buttons

- This class represents a push-button which displays some specified text.
- When a button is pressed, it notifies its Listeners.
- To have a Listener for a button, an object must implement the ActionListener Interface.

AWT Controls

- Lists
 - Provide compact, multiple choice ,scrolling selection list
- Check box
 - This class represents a GUI checkbox with a textual label.
 - The Checkbox maintains a Boolean state indicating whether it is checked or not.
 - If a Checkbox is added to a CheckBoxGroup, it will behave like a radio button.
- Choice Lists
 - This class represents a dropdown list of Strings.
 - Only one item from the list can be selected at one time and the currently selected element is displayed

AWT Controls

- **TextField**
 - This class displays a single line of optionally editable text.
 - This class inherits several methods from `TextComponent`.
- **TextArea**
 - This class displays multiple lines of optionally editable text.
 - `TextArea` also provides the methods: `append()`, `insert()` and `replaceRange()`
 - Ex:-
 - `TextArea fullAddressTextArea = new TextArea(5, 80);`
`// 5 rows, 80 columns`
- **Scroll bars**
 - Used to select continuous values between specified minimum and maximum

Layout Managers

- The LayoutManagers are used to arrange components in a particular manner.
- LayoutManager is an interface that is implemented by all the classes of layout managers.

Layout Managers

- Since the Component class defines the setSize() and setLocation() methods, all Components can be sized and positioned with those methods.
- Problem:
 - The parameters provided to those methods are defined in terms of pixels. Pixel sizes may be different (depending on the platform) so the use of those methods tends to produce GUIs which will not display properly on all platforms

Layout Managers

- Solution:
 - Layout managers are assigned to Containers.
- When a Component is added to a Container, its Layout Manager is consulted in order to determine the size and placement of the Component.
- NOTE:
 - If you use a Layout Manager, you can no longer change the size and location of a Component through the `setSize` and `setLocation` methods.

Layout Managers

There are several different predefined LayoutManagers,

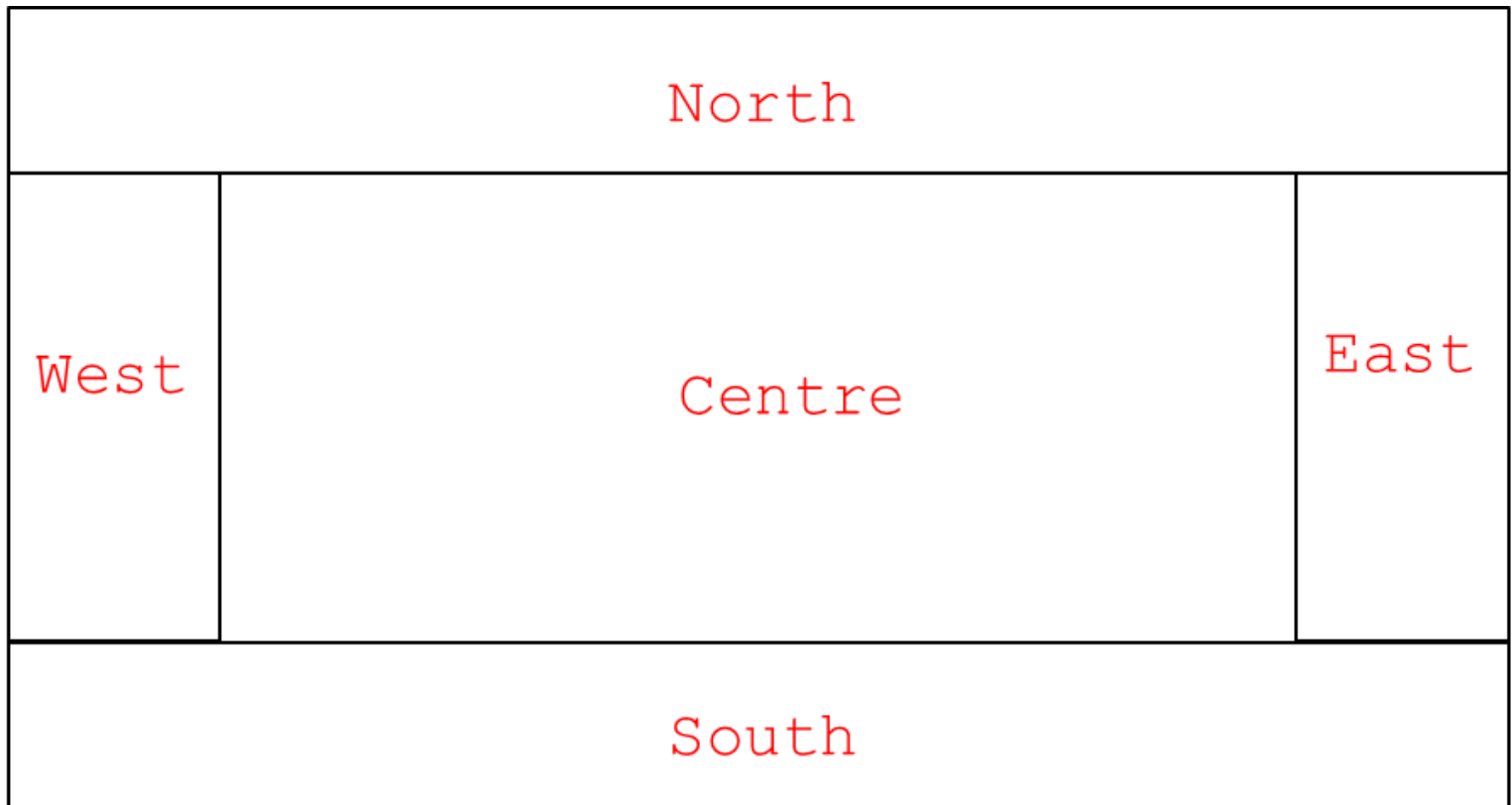
- `java.awt.BorderLayout`
- `java.awt.FlowLayout`
- `java.awt.GridLayout`
- `java.awt.CardLayout`
- `java.awt.GridBagLayout`
- `javax.swing.BoxLayout`
- `javax.swing.GroupLayout`
- `javax.swing.ScrollPaneLayout`
- `javax.swing.SpringLayout` etc

Border Layout

- 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.
- The BorderLayout provides five constants for each region:
 - `public static final int NORTH`
 - `public static final int SOUTH`
 - `public static final int EAST`
 - `public static final int WEST`
 - `public static final int CENTER`

Border Layout

The regions of the BorderLayout are defined as follows:

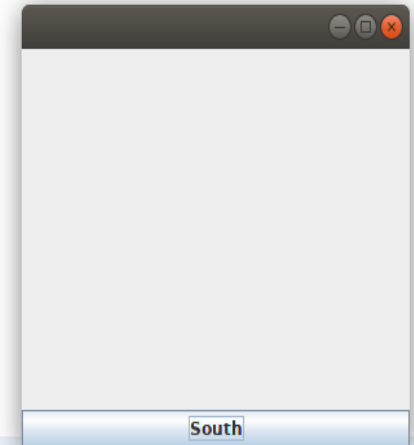


Constructors of BorderLayout class

- **BorderLayout():**
 - creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):**
 - creates a border layout with the given horizontal and vertical gaps between the components.

Border Layout - Example

```
6 package borderdemo;
7
8 import java.awt.*;
9 import javax.swing.*;
10
11 /* @author nuwan */
12 public class MyBorder {
13
14     public MyBorder() {
15         JFrame myJframe = new JFrame();
16         JButton myJbutton = new JButton("South");
17         myJframe.add(myJbutton, BorderLayout.SOUTH);
18         myJframe.setSize(300,300);
19         myJframe.setLocation(500, 500);
20         myJframe.setVisible(true);
21     }
22 }
23
```

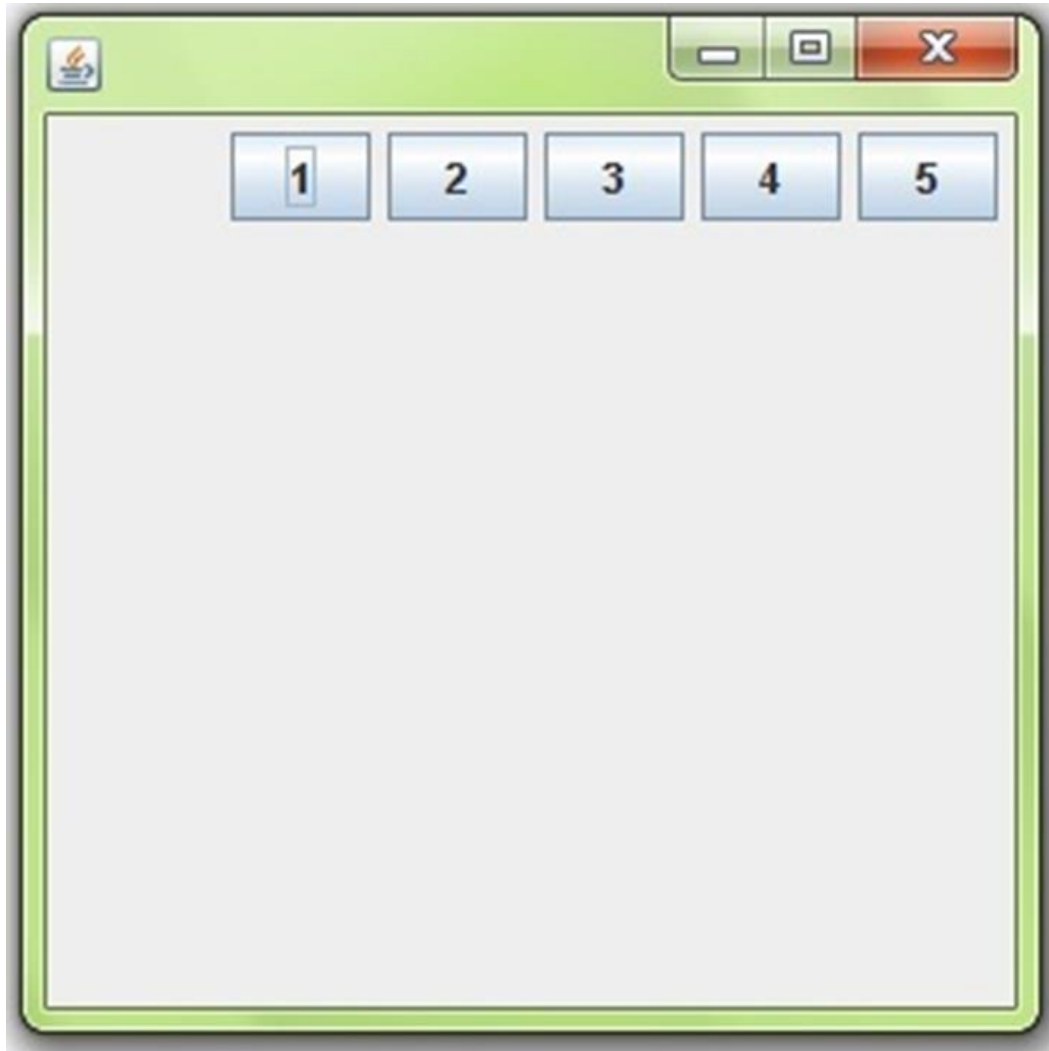


```
6 package borderdemo;
7
8 /* @author nuwan */
9 public class BorderDemo {
10
11     public static void main(String[] args) {
12         MyBorder myBorder = new MyBorder();
13     }
14
15 }
16
```

Flow Layout

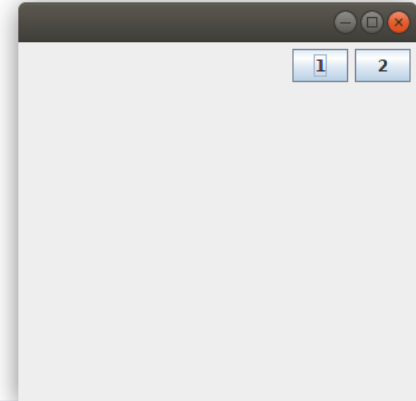
- The FlowLayout is used to arrange the components in a line, one after another (in a flow).
- It is the default layout of applet or panel.
- Fields of FlowLayout class
 - `public static final int LEFT`
 - `public static final int RIGHT`
 - `public static final int CENTER`
 - `public static final int LEADING`
 - `public static final int TRAILING`

Flow Layout



Flow Layout - Example

```
6 package flowlayoutdemo;
7
8 import java.awt.*;
9 import javax.swing.*;
10 /* @author nuwan */
11 public class MyFlowLayout {
12     public MyFlowLayout() {
13         JFrame myJframe = new JFrame();
14         JButton myJbutton1 = new JButton("1");
15         myJframe.add(myJbutton1);
16         JButton myJbutton2 = new JButton("2");
17         myJframe.add(myJbutton2);
18         myJframe.setLayout(new FlowLayout(FlowLayout.RIGHT));
19         myJframe.setSize(300,300);
20         myJframe.setLocation(500, 500);
21         myJframe.setVisible(true);
22     }
23 }
```



```
6 package flowlayoutdemo;
7
8 /* @author nuwan */
9 public class FlowLayoutDemo {
10
11     public static void main(String[] args) {
12         MyFlowLayout myFlowLayout = new MyFlowLayout();
13     }
14
15 }
16
```

Grid Layout

- The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle
- Constructors of GridLayout class:
 - 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 alongwith given horizontal and vertical gaps.

Grid Layout



Grid Layout - Example

```
6 package gridlayoutdemo;
7
8 import java.awt.*;
9 import javax.swing.*;
10 /* @author nuwan */
11 public class MyGridLayout {
12     public MyGridLayout(){
13         JFrame myJframe = new JFrame();
14         JButton myJbutton1 = new JButton("1");
15         myJframe.add(myJbutton1);
16         //Create 09 buttons
17
18         myJframe.setLayout(new GridLayout(3,3));
19
20         myJframe.setSize(300, 300);
21         myJframe.setVisible(true);
22     }
23 }
24
25 To change this template file, choose Tools | Templates
26 * and open the template in the editor.
27 */
28 package gridlayoutdemo;
29
30 /**
31  *
32  * @author nuwan
33  */
34 public class GridLayoutDemo {
35
36     /**
37      * @param args the command line arguments
38      */
39     public static void main(String[] args) {
40         // TODO code application logic here
41         MyGridLayout a = new MyGridLayout();
42     }
43 }
44 }
```

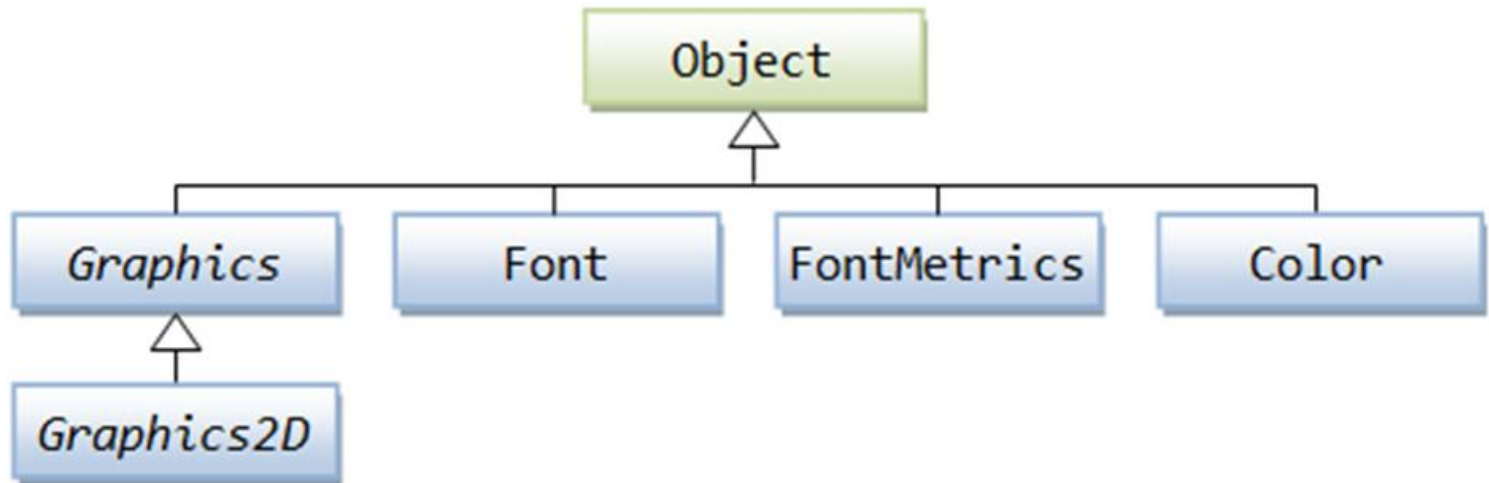


Working with Graphics

- Custom painting is provided via the Graphics class, which is part of the java.awt package.
- The Graphics class has methods for drawing lines, drawing and filling shapes and displaying text.
- It also has methods to set the colour, font etc.

java.awt.Graphics

- java.awt.Graphics is an abstract class



java.awt.Graphics

The Graphics class provides methods for drawing three types of graphical objects:

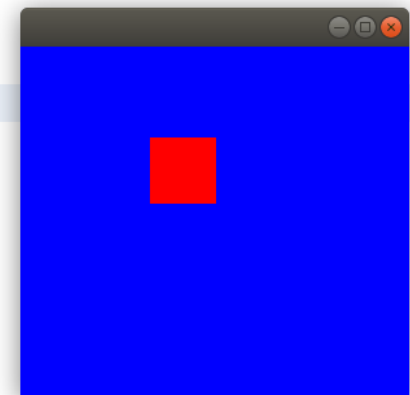
- Text strings:
 - via the drawString() method.
- Vector-graphic primitives and shapes:
 - via methods drawXxx() and fillXxx(), where Xxx could be Line, Rect, Oval, Arc, PolyLine, RoundRect, or 3DRect.
- Bitmap images:
 - via the drawImage() method.

java.awt.Graphics

- The graphic context maintains states (or attributes) such as
 - the current painting color,
 - the current font for drawing text strings, and
 - the current painting rectangular area (called clip)
- Graphics Class' Methods for Maintaining the Graphics Context
 - getColor(), setColor()
 - getFont(), setFont()
 - getClipBounds(), setClip()

java.awt.Graphics - Example

```
6   package graphicsdemo;
7
8   import java.awt.*;
9
10  /* @author nuwan */
11  public class MyDrawingGui extends Frame{
12      public MyDrawingGui(){
13          setBackground(Color.blue);
14      }
15
16      @Override
17      public void paint(Graphics g){
18          g.setColor(Color.red);
19          g.drawRect(100, 100, 50, 50);
20          g.fillRect(100, 100, 50, 50);
21      }
22  }
23
24  package graphicsdemo;
25
26  /* @author nuwan */
27  public class GraphicsDemo {
28
29      public static void main(String[] args) {
30          // TODO code application logic here
31          MyDrawingGui myDrawingGui = new MyDrawingGui();
32          myDrawingGui.setSize(300, 300);
33          myDrawingGui.setVisible(true);
34      }
35  }
```



Working with Colors

- Java support colors in portable device independent fashion
- Color class provides static fields that return a specific Color object:
 - BLACK, BLUE, GREEN, RED, CYAN, ORANGE, YELLOW.
- Create a custom color by passing red-green-blue (RGB) values to the Color class's constructor:
- To change a component's color, call the setForeground and setBackground methods of the component

java.awt.Color


- The class java.awt.Color provides 13 standard colors as named-constants.
- RED, GREEN, BLUE, MAGENTA, CYAN, YELLOW, BLACK, WHITE, GRAY, DARK_GRAY, LIGHT_GRAY, ORANGE, and PINK.
- Ex :-
 - g.setColor(Color.RED);
 - g.drawLine(10, 20, 30, 40); // in Color.RED
 - Color myColor = new Color(123, 111, 222);
 - g.setColor(myColor);
 - g.drawRect(10, 10, 40, 50); // in myColor

Working with Fonts

- The class `java.awt.Font` represents a specific font face, which can be used for rendering texts.
- constructor :
 - `public Font (java.lang.String name, int style, int size)`
 - `name` is the font name (such as "Verdana", "Arial", etc).
 - `style` argument takes an integer bitmask that may be `PLAIN` or a bitwise union of `BOLD` and/or `ITALIC`
 - `size` is the point size of the font.
- `setFont()` method in the component class to set the current font for the `Graphics` context `g` for rendering texts

Working with Fonts - Example

```
6 package fontsdemo;
7
8 import java.awt.*;
9
10 /* @author nuwan */
11 class MyFonts extends Frame{
12     public MyFonts(){
13         setBackground(Color.CYAN);
14     }
15
16     @Override
17     public void paint(Graphics g){
18         Font myFont = new Font("Courier",Font.PLAIN,20);
19         g.setFont(myFont);
20         g.drawString("Hello Fonts", 20, 50);
21     }
22 }
23
```



```
6 package fontsdemo;
7
8 /* @author nuwan */
9 public class FontsDemo {
10
11     public static void main(String[] args) {
12         // TODO code application logic here
13         MyFonts myFonts = new MyFonts();
14         myFonts.setSize(300, 300);
15         myFonts.setVisible(true);
16     }
17
18 }
19
```

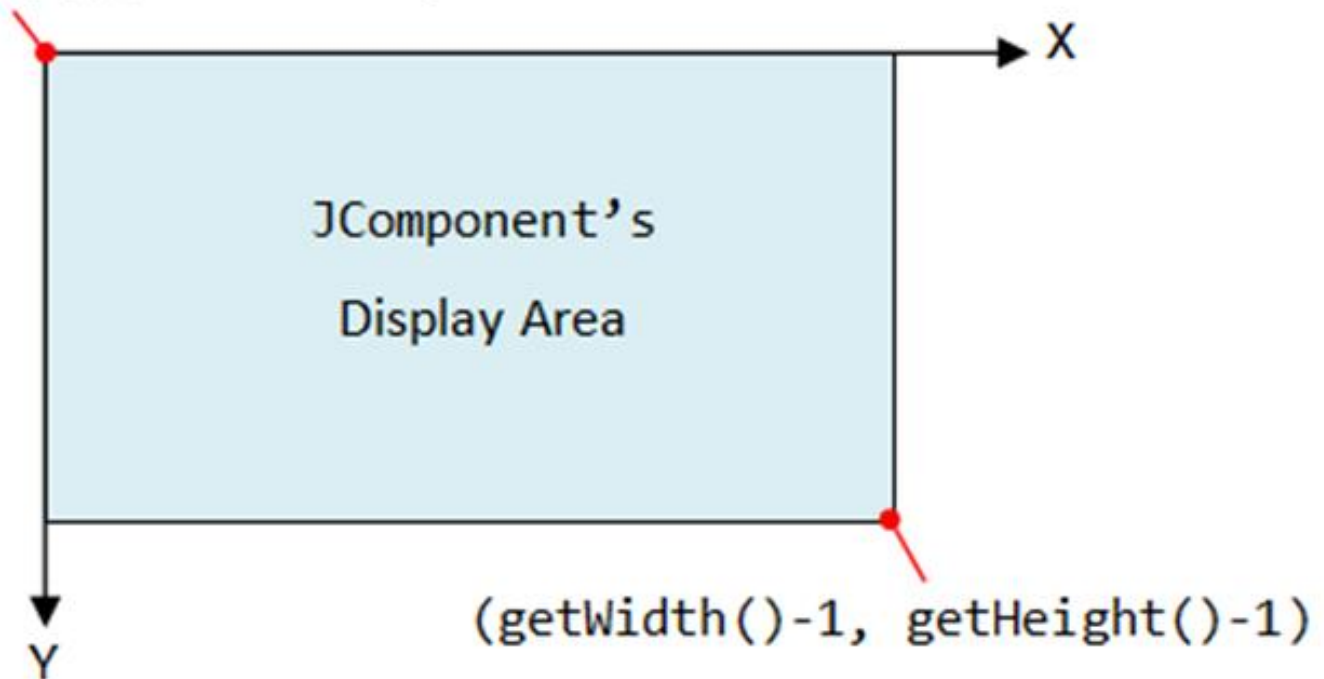
Graphic Coordinate System

- In Java Windowing Subsystem (like most of the 2D Graphics systems), the origin (0,0) is located at the top-left corner.
- EACH component/container has its own coordinate system, ranging for (0,0) to (width-1, height-1) as illustrated.
- You can use method getWidth() and getHeight() to retrieve the width and height of a component/container. You can use getX() or getY() to get the top-left corner (x,y) of this component's origin relative to its parent.

Graphic Coordinate System

Origin (0,0)

or (x,y) relative to parent



java.awt.Canvas

- This class is used to create an area in a frame to be used for displaying graphics.
- Canvas class methods:
 - void setSize(width, height)
 - Sets the size of the canvas
 - void setBackground(Color c)
 - Sets the background color of the canvas
 - void setForeground(Color c)
 - Sets the text color of the canvas
- public void paint(Graphics g)
 - methods which are used for displaying graphics
 - takes one attribute - an instance of the Graphics class

Canvas – Example I

```
1 import java.awt.*;
2
3 class GraphicsProgram extends Canvas{
4
5     public GraphicsProgram(){
6         setSize(200, 200);
7         setBackground(Color.white);
8     }
9
10    public static void main(String[] argS){
11
12        //GraphicsProgram class is now a type of canvas
13        //since it extends the Canvas class
14        //lets instantiate it
15        GraphicsProgram GP = new GraphicsProgram();
16
17        //create a new frame to which we will add a canvas
18        Frame aFrame = new Frame();
19        aFrame.setSize(300, 300);
20
21        //add the canvas
22        aFrame.add(GP);
23
24        aFrame.setVisible(true);
25    }
26
27    public void paint(Graphics g){
28
29        g.setColor(Color.blue);
30        g.drawLine(30, 30, 80, 80);
31        g.drawRect(20, 150, 100, 100);
32        g.fillRect(20, 150, 100, 100);
33        g.fillOval(150, 20, 100, 100);
34        Image img1 = Toolkit.getDefaultToolkit().getImage("sky.jpg");
35        g.drawImage(img1, 140, 140, this);
36    }
37 }
```

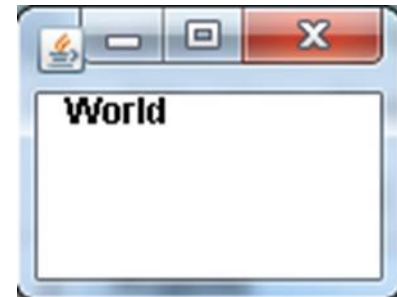
Canvas – Example II

```
package graphic;
import java.awt.*;
/**
 *
 * @author DELL
 */
public class Drawing extends Canvas {

    public Drawing(){
        Frame f=new Frame();
        f.add(this);
        f.setSize(100,100);
        f.setVisible(true);
    }

    public void paint(Graphics g){
        // g.drawString("hello",10,10);
        Font fn=new Font("Arial",1,12);
        g.setFont(fn);
        g.drawString("World",10,10);
    }

    public static void main(String args[]){
        Drawing dw=new Drawing();
    }
}
```



Event Handling

- An Event is something that the user does at the GUI.
- GUI components communicate with the rest of the applications through events.
- The source of an event is the component that causes that event to occur.
- The listener of an event is an object that receives the event and processes it appropriately.

Components of Event Handling

Event handling has three main components,

- Events :
 - An event is a change of state of an object.
- Events Source :
 - Event source is an object that generates an event.
 - The source is an object on which event occurs.
 - Source is responsible for providing information of the occurred event to it's handler.
 - Java provide as with classes for source object.
- Listeners :
 - A listener is an object that listens to the event.
 - A listener gets notified when an event occurs.
 - It is also known as event handler.
 - Listener is responsible for generating response to an event. Listener waits until it receives an event.
 - Once the event is received , the listener process the event an then returns.

Handling Events

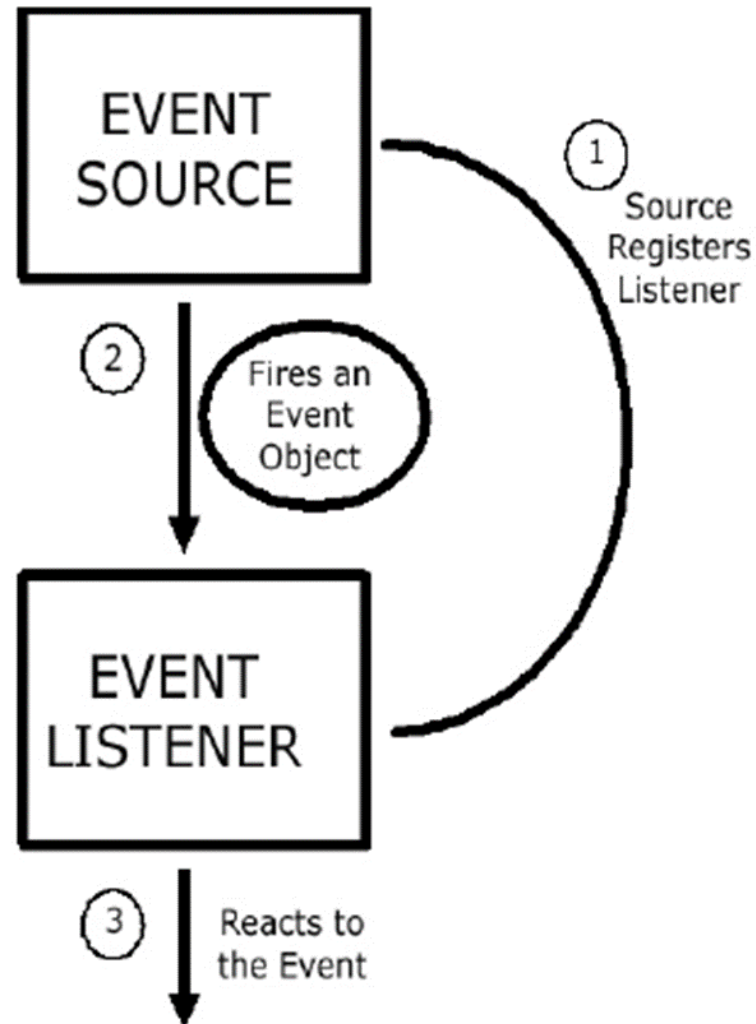
- Every time the user types a character or clicks the mouse, an event occurs.
- A source generates an Event and send it to one or more listeners registered with the source.
- Once event is received by the listener, they process the event and then return.
- Events are supported by a number of Java packages, like java.util, java.awt and java.awt.event



Steps involved in Event Handling Process

- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
- Event object is forwarded to the method of registered listener class.
- The method is now get executed and returns.

Delegation Event Model



Important Event Classes and Interface

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved, clicked, pressed or released also when the enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener

Important Event Classes and Interface

Event Classes	Description	Listener Interface
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseEvent	generated when mouse wheel is moved	MouseListener

Important Event Classes and Interface

Event Classes	Description	Listener Interface
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener

Important Event Classes and Interface

Event Classes	Description	Listener Interface
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

Registering Event Handlers and processing Events

- First, in the declaration of the event handler class,
 - `public class DemoClass implements ActionListener {`
- Second, registers an instance of the event handler class as a listener
 - `anyComponent.addActionListener(instanceOfDemoClass);`
- Third, the event handler must have a piece of code that implements the methods in the listener interface.
 - `public void actionPerformed(ActionEvent e) {
 ...//code that reacts to the action...
}`

Example

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

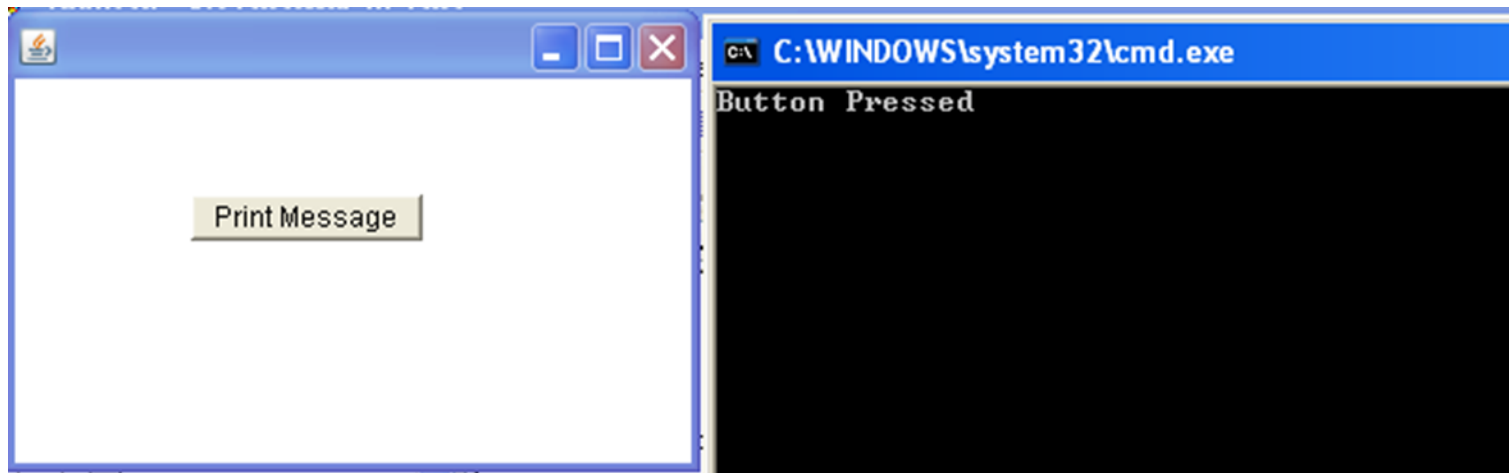
class ButtonController extends Frame implements ActionListener {
    Button myButton1;

    public ButtonController() {
        setLayout(null);
        myButton1 = new Button("Print Message");
        myButton1.setLocation(80,80);
        myButton1.setSize(100,20);
        add(myButton1);
        myButton1.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == myButton1) {
            System.out.println("Button Pressed");
        }
    }
}

public class MyGui {
    public static void main(String[] args) {
        ButtonController myController = new ButtonController();
        myController.setSize(300,200);
        myController.setVisible(true);
    }
}
```

Example - Output



Event Adapters

- Adapters are abstract classes for receiving various events.
- The methods in these classes are empty.
- These classes exist as convenience for creating listener objects

Event Adapters

<u>Adapter & Description</u>
FocusAdapter An abstract adapter class for receiving focus events.
KeyAdapter An abstract adapter class for receiving key events.
MouseAdapter An abstract adapter class for receiving mouse events.
MouseMotionAdapter An abstract adapter class for receiving mouse motion events.
WindowAdapter An abstract adapter class for receiving window events.

Applet in Java

- Applets are small Java applications that can be
 - accessed on an Internet server,
 - transported over Internet, and
 - can be automatically installed and run as apart of a web document.
- Any applet in Java is a class that extends the `java.applet.Applet` class.
- An Applet class does not have any `main()` method.
- It is viewed using JVM.
 - The JVM can use either a plug-in of the Web browser or a separate runtime environment to run an applet application.
- JVM creates an instance of the applet class and invokes `init()` method to initialize an Applet.
- Applets function like regular Java programs but with a few security restrictions.
 - such as applets can't read or write files on your computer

The Applet class

- The Applet class is used to create applets.
- This class is located in the `java.applet` package.
- Applets are not created by instantiating an Applet object, rather by creating a class which extends the Applet class.

Applet core structure

```
import java.applet.Applet;
```

```
public class AnApplet extends Applet  
{  
  
}
```


Applet class methods

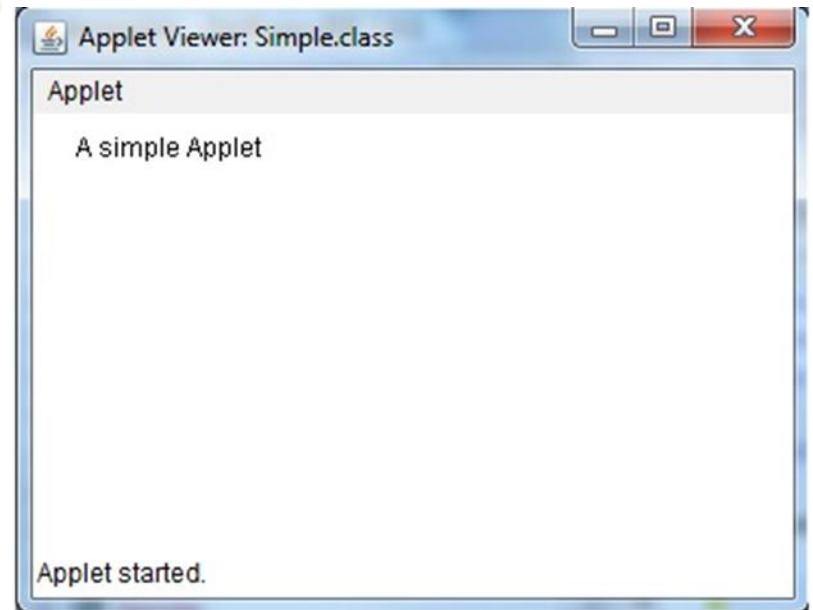
- `init()`
 - Initializes an applet for usage and informs the applet that it has been loaded
- `start()`
 - Informs an applet that it should start its execution
- `stop()`
 - Informs an applet that it should stop its execution
- `destroy()`
 - Informs an applet that it is being removed from memory and any resources it has allocated should be removed as well
- `resize(int width, int height)`
 - Resizes the applet to the specified width and height

Example - Applet class methods

```
import java.applet.Applet;
import java.awt.TextArea;
public class AnApplet extends Applet
{
    TextArea taI = new TextArea(12, 40);
    public void init(){
        add(taI);
        taI.append("Applet has been initialized");
    }
    public void start(){
        taI.append("\nApplet has been started");
    }
    public void stop(){
        taI.append("\nApplet has been stopped");
    }
    public void destroy(){
        taI.append("\nApplet has been destroyed");
    }
}
```

A Simple Applet

```
import java.awt.*;  
import java.applet.*;  
public class Simple extends Applet  
{  
    public void paint(Graphics g)  
    {  
        g.drawString("A simple Applet", 20, 20);  
    }  
}
```



Applet application

- Every Applet application must declare a `paint()` method.
- This method is defined by AWT class and must be overridden by the applet.
- `paint()` method is called each time an applet needed to redisplay its output.
- Another important thing to notice about applet application is that, execution of an applet does not begin at `main()` method.
- In fact an applet application does not have any `main()` method.

How to run an Applet application

- An Applet program is compiled in the same way as you have been compiling your console programs. However there are two ways to run an applet.
 - Executing the Applet within Java-compatible web browser.
 - Using an Applet viewer, such as the standard tool, applet viewer. An applet viewer executes your applet in a window

How to run an Applet application

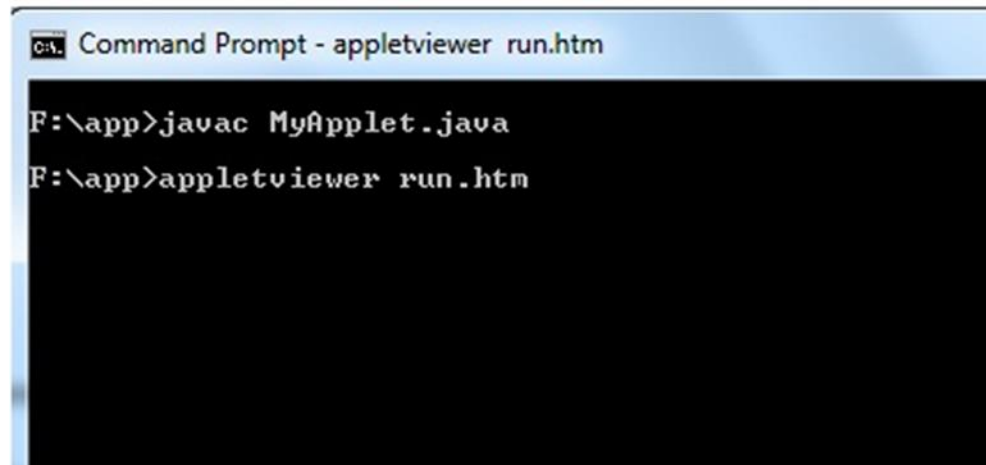
- For executing an Applet in an web browser, create short HTML file in the same directory. Inside body tag of the file, include the following code. (applet tag loads the Applet class)

```
< applet code = " Simple" width=400  
height=400 >  
< /applet >
```

- Simple is the name of the applet class

Running Applet using Applet Viewer

- To execute an Applet with an applet viewer, write short HTML file as discussed above. If name it as MyApplet.htm, then the following command will run your applet program.
- `f:/>appletviewer MyApplet.htm`



```
C:\> Command Prompt - appletviewer run.htm

F:\app>javac MyApplet.java
F:\app>appletviewer run.htm
```

References

- *The Swing Tutorial*
 - <https://docs.oracle.com/javase/tutorial/uiswing/index.html>
- *JavaFX*
 - <https://docs.oracle.com/javase/8/javase-clienttechnologies.htm>

Summary

- Graphic Programming
- GUI Toolkits in Java
 - AWT Components
 - Container, Window, Panel, Frame
 - AWT Controls
- Layout Managers
 - Border, Flow, Grid
- Custom drawings
 - Working with graphics
 - Working with colors
 - Working with fonts
- Examples
- Graphic Coordinate System
- AWT Event package
- GUI designing using java swing
- Applet and applet life cycle

References

- How To Program (Early Objects)
 - 10th Edition
 - By H .Deitel and P. Deitel
- Head First Java
 - 2nd Edition
 - By Kathy Sierra and Bert Bates

Questions ???





Thank You