

UNIT I - JAVA FUNDAMENTALS

- Java Data types
- Class – Object
- I / O Streams
- File Handling concepts
- Threads
- **Applets**
- Swing Framework
- Reflection

Presented by,
B.Vijayalakshmi
Computer Centre
MIT Campus
Anna University

Java Applet

- An Applet is a **dynamic and interactive program that can run inside a web page** displayed by a java capable browsers
- **It runs inside the browser** and works at client side.
- In other words, we can say that 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.
- 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.
- **Note:** Java Applet is deprecated since Java 9. It means Applet API is no longer considered important.

Difference between Applets and Applications

Java Application	Java Applet
Java Applications are the stand-alone programs which can be executed independently	Java Applets are small Java programs which are designed to exist within HTML web document
Java Applications must have main() method for them to execute	Java Applets do not need main() for execution
Java Applications just needs the JRE	Java Applets cannot run independently and require API's
Java Applications do not need to extend any class unless required	Java Applets must extend <code>java.applet.Applet</code> class
Java Applications can execute codes from the local system	Java Applets Applications cannot do so
Java Applications has access to all the resources available in our system	Java Applets has access only to the browser-specific services

Restrictions to Applets

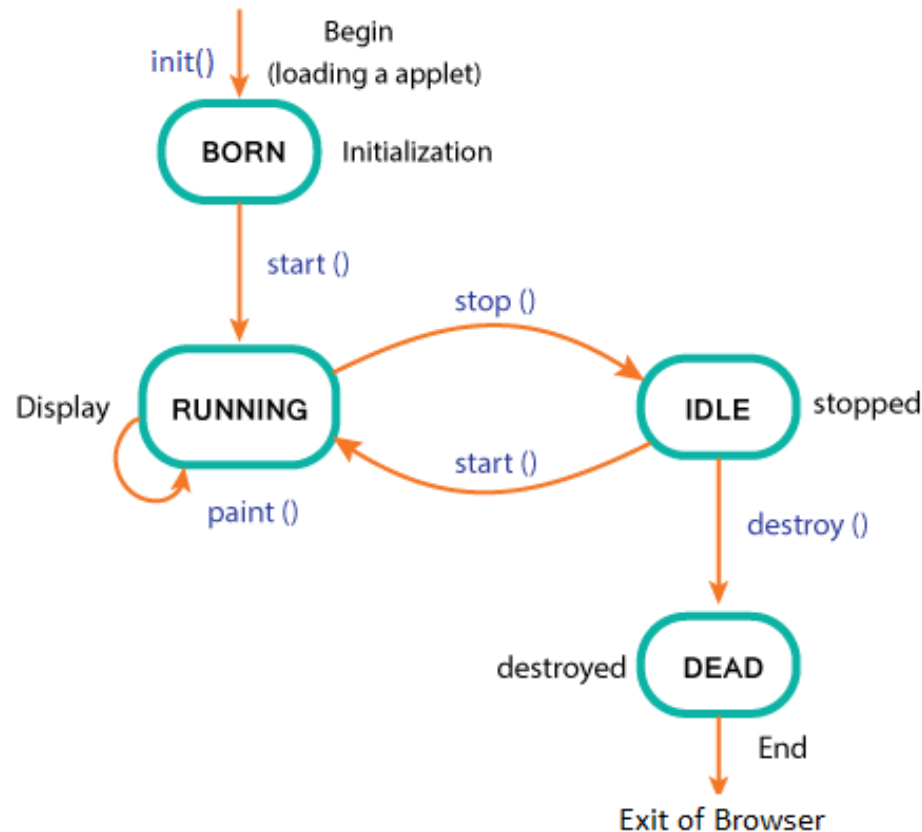
- To ensure security and to prevent them from being affected by viruses, some of the restriction are followed.
 - An applet **cannot load libraries** or define native methods.
 - An applet **cannot ordinarily read or write files** on the execution host.
 - An applet **cannot read certain system properties**.
 - An applet **cannot make network connections except to the host that it came from**.
 - An applet **cannot start any program on the host** that's executing it.

Applet Hierarchy in Java

- class java.lang.**Object**
 - class java.awt.**Component**
 - class java.awt.**Container**
 - class java.awt.**Panel**
 - class java.applet.**Applet**
- As displayed above,
 - The Java Applet class which is a class of applet package extends the Panel class of awt package.
 - The Panel class is a subclass of the Container class of the same package.
 - The Container class is an extension of Component class belonging to the same package.
 - The Component class is an abstract class and derives several useful classes for the components such as Checkbox, List, buttons, etc.

Applet Life Cycle

- Every **java Applet inherits** a set of default **behaviours from the Applet class**
- When an applet is loaded, it undergoes a series of changes as shown below,



Applet Life Cycle Cont'd

- **Initialization (or) Born state:**

- Applet enters this state when it is first loaded
- This is achieved by calling **init()** method
- At this stage, we may do the following
 - 1.Create objects needed by Applet
 - 2.Setting of initial values
 - 3.Load images and fonts
 - 4.Set up colours
- This occurs only once in the Applet life cycle.

- **Running State:**

- This happens automatically after the applet is initialized and when the system calls **start()** method
- It is invoked after init() method or browser is maximized. It is used to start the Applet
- Starting can also occur if the applet is already 'stopped'(idle)
- start() method may be called more than once
- We may override start method to create thread to control the applet

Applet Life Cycle Cont'd

- Idle (or) Stopped state:

- An Applet becomes idle when it is stopped from running
- Stopping occurs automatically when we leave the page
- Can also be stopped using **stop()** method
- If thread is used thread should be stopped

- Dead State:

- An applet is said to be dead when it is removed from memory
- This occurs automatically by invoking the `destroy()` method when we quit the browser
- It also **occurs only once**.

- Display state

- Applet moves to this state whenever it has to perform some output operations on the screen
- This happens immediately after the applet enters into the **running state**
- **paint()** method is used to accomplish the task and this method **helps in drawing, writing & creating a colored background** (or) an image onto the applet
- **It takes an argument, which is an instance of the Graphics class**

Lifecycle methods for Applet

- **Every Applet application must import two packages ,**
 1. java.awt
 2. java.applet
- **java.awt.***
 - It imports the **Abstract Window Toolkit (AWT)** classes.
 - Applets interact with the user (either directly or indirectly) through the AWT.
 - The AWT contains support for a window-based, graphical user interface.
- **java.applet.***
- It imports the applet package, which contains the class Applet.
- Every applet that we create must be a subclass of Applet class.
- The **java.applet.Applet** class provides 4 life cycle methods and **java.awt.Component** class provides 1 life cycle methods for an applet.

Lifecycle methods for Applet

- **java.applet.Applet class**

- For creating any applet **java.applet.Applet** class must be inherited. It provides 4 life cycle methods of applet.
- **public void init():** is used to initialize the Applet. It is invoked only once.
- **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
- **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
- **public void destroy():** is used to destroy the Applet. It is invoked only once.

- **java.awt.Component class**

- The Component class provides 1 life cycle method of applet.
- **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

- **Note:** The stop() method is always called before destroy() method.

Steps to create an Applet

1. Write a simple applet in Java
2. Compile the Java source code
3. Create an HTML page that references the applet
4. Run the Applet

There are **two** standard ways in which we can run an applet :

- Executing the applet within a **Java-compatible web browser**.
- Using an **applet viewer**, such as the standard tool, applet-viewer. An applet viewer executes our applet in a window. This is generally the fastest and easiest way to test our applet.

Procedure to write a simple applet in Java

- **import two packages** - java.awt and java.applet.
- The **class in the program must be declared as public**, because it will be accessed by code that is outside the program.
- **Every Applet application must declare a paint() method.**
- This method is defined by AWT class and must be overridden by the applet.
- The **paint()** method has one parameter of type Graphics.
public void paint(Graphics g)
where g is an object reference of class Graphic
- The paint() method is called each time when an applet needs 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.**

1. Write a simple applet in Java

Applet1.java

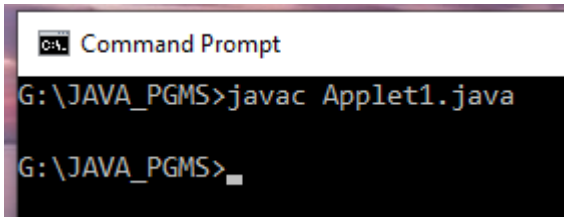
```
import java.applet.*;  
import java.awt.*;
```

```
public class Applet1 extends Applet  
{  
    public void paint(Graphics g)  
    {  
        g.drawString("welcome",50,100);  
    }  
}
```

This method outputs a string beginning at the specified X,Y location. It has the following general form:
`void drawString(String message, int x, int y)`

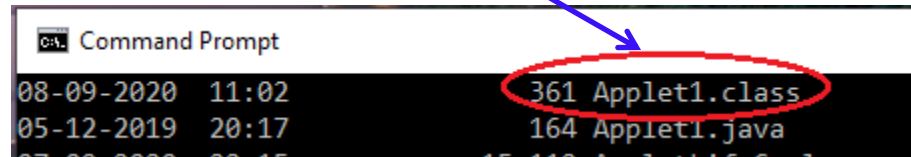
In a Java window, the upper-left corner is location 0,0.

2. Compile the Java source code



```
C:\> Command Prompt  
G:\JAVA_PGMS>javac Applet1.java  
G:\JAVA_PGMS>_
```

After compiling, we get **Applet1.class**



```
C:\> Command Prompt  
08-09-2020 11:02 361 Applet1.class  
05-12-2019 20:17 164 Applet1.java  
07-09-2020 22:15 15 113 Applet1.class
```

3. Create an HTML page that references the applet

Applet1.html

```
<html>
  <body>
    <applet code="Applet1.class"
            height=200
            width=200>
    </applet>
  </body>
</html>
```

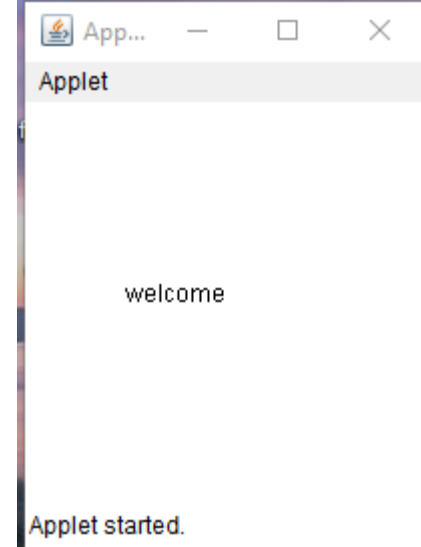
4. Open the HTML page in a browser

- **Now most of the browsers doest support Applets since they removed NPAPI (Netscape Plugin Application Programming Interface) support to improve speed, stability and security.**
- Use the supported browsers (or **AppletViewer**, a program bundled with JDK).
- **Coming with Java 9, Oracle reportedly deprecating the Applet plugin altogether**
- **Better, don't spend time learning about Applets based programming - its an obsolete technology.**

Using appletViewer to execute Applet

Command Prompt - appletviewer Applet1.html

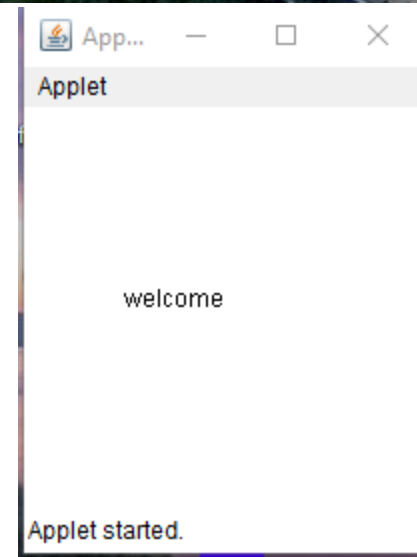
```
G:\JAVA_PGMS>appletviewer Applet1.html
```



To execute the applet by appletviewer tool, we create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer. Now Html file is not required but it is for testing purpose only.

```
import java.applet.*;
import java.awt.*;
/*
<applet code="WelcomeApplet" height=200 width=200>
</applet>
*/
public class WelcomeApplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("welcome",50,100);
    }
}
```

```
G:\JAVA_PGMS>javac WelcomeApplet.java
G:\JAVA_PGMS>appletviewer WelcomeApplet.java
```



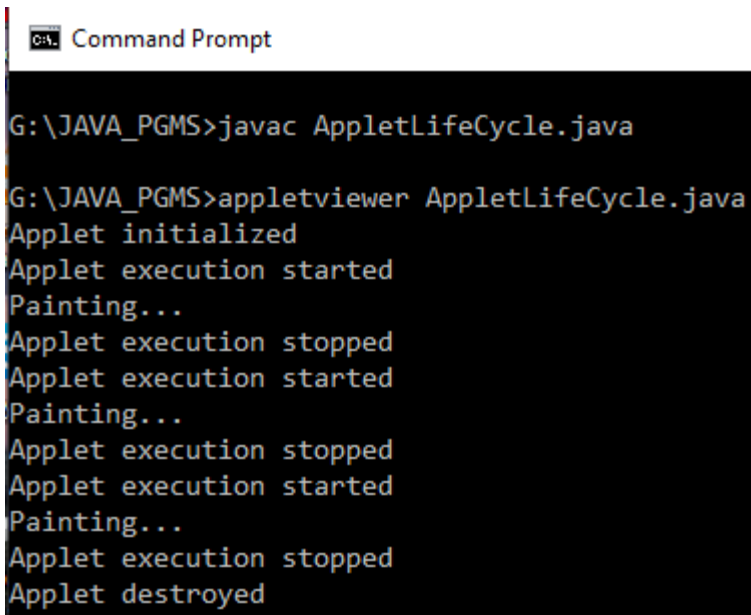
Applet Life Cycle Example

```
import java.awt.*;
import java.applet.*;
/*
<applet code="AppletLifeCycle" height=200 width=200>
</applet>
*/

public class AppletLifeCycle extends Applet
{
    public void init()
    {
        System.out.println("Applet initialized");
    }
    public void start()
    {
        System.out.println("Applet execution started");
    }
    public void stop()
    {
        System.out.println("Applet execution stopped");
    }
}
```

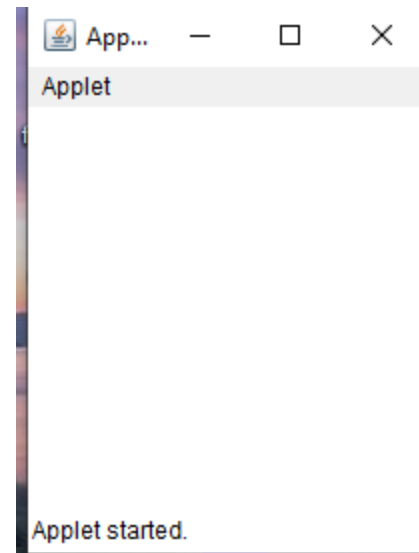


```
public void paint(Graphics g)
{
    System.out.println("Painting...");
}
public void destroy()
{
    System.out.println("Applet destroyed");
}
}
```



Command Prompt

```
G:\JAVA_PGMS>javac AppletLifeCycle.java
G:\JAVA_PGMS>appletviewer AppletLifeCycle.java
Applet initialized
Applet execution started
Painting...
Applet execution stopped
Applet execution started
Painting...
Applet execution stopped
Applet execution started
Painting...
Applet execution stopped
Applet destroyed
```

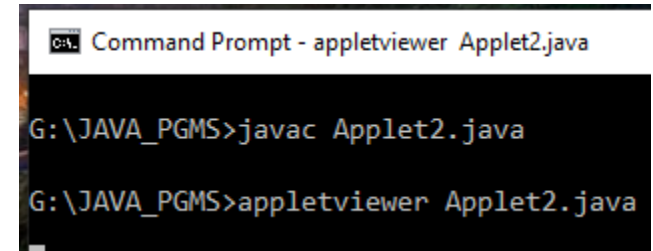


Passing parameters to an Applet

- To pass parameters to an applet, 2 things are required,
 1. A **special parameter tag <PARAM> in the HTML file**. it has two attributes namely NAME, VALUE
 2. **Code in Applet to parse these parameters**
 - The init() method of applet, contains a method called `getParameter()` .
 - `getParameter()` takes one argument, the String representing the name of the parameter being looked for and returns a String containing the corresponding value of that parameter
 - If this method is required to return types other than Strings, it has to be converted explicitly .
- In Applet, Parameters are passed on applet when it is loaded.

Passing parameters to an Applet Example

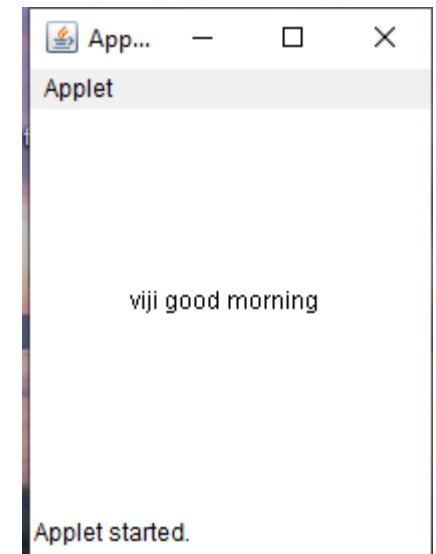
```
import java.applet.*;
import java.awt.*;
/*
<applet code="Applet2" height=200 width=200>
    <PARAM name=frName value="viji"> </PARAM>
</applet>
*/
public class Applet2 extends Applet
{
    String str;
    public void init()
    {
        str= getParameter("frName");
    }
    public void paint(Graphics g)
    {
        if(str==null)
            str="friend ";
        str=str+" good morning";
        g.drawString(str,50,100);
    }
}
```



```
Command Prompt - appletviewer Applet2.java

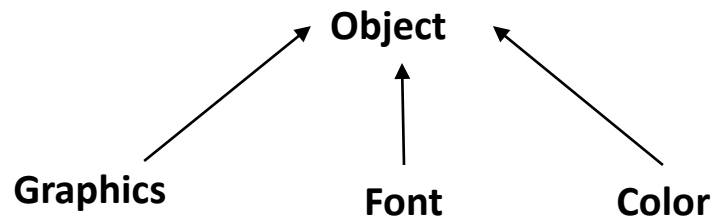
G:\JAVA_PGMS>javac Applet2.java

G:\JAVA_PGMS>appletviewer Applet2.java
```



Graphics, Color and Font Class

- In java, **all graphics operations can be performed using java.awt** package
- It contains various classes and methods for graphics and images



Graphics class

- This class contains methods for drawing strings, lines rectangles and other shapes defined in Graphics class

Methods of the Graphics class

Sr No.	Methods	Description
1	public abstract void drawString(String str, int x, int y)	Used to draw specified string.
2	public void drawRect(int x, int y, int width, int height)	Used to draw a rectangle of specified width and height.
3	public abstract void fillRect(int x, int y, int width, int height)	Used to draw a rectangle with a default colour of specified width and height.
4	public abstract void drawOval(int x, int y, int width, int height)	Used to draw oval of specified width and height.
5	public abstract void fillOval(int x, int y, int width, int height)	Used to draw oval with a default colour of specified width and height.

Methods of the Graphics class

Sr No.	Methods	Description
6	<code>public abstract void drawLine(int x1, int y1, int x2, int y2)</code>	Used for drawing lines between the point (x1, x1) and (x2, y2).
7	<code>public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)</code>	Used for drawing a specified image.
8	<code>public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code>	Used for drawing a circular arc.
9	<code>public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code>	Used for filling circular arc.
10	<code>public abstract void setColor(Color c)</code>	Used to set a colour to the object.
11	<code>public abstract void setFont(Font font)</code>	Used to set font.

Font Class

- We can select the font style of our choice by using **Font** class of **java.awt** package.
- Font class support the following constructor to create a new font -

public Font(String name, int style, int size)

- **name** specifies the font name. The font name can be a font face name or a font family name.
 - **style** can have one of the following three values Font.BOLD, Font.PLAIN, Font.ITALIC. We can also combine these styles. For example if we want bold and italic then we can combine Font.BOLD + Font.ITALIC.
 - **size** specifies the font size.
- After creating new font, we can use it by using **setFont()** method.

public abstract void setFont(Font fontObj)

- Where **fontObj** specifies the new font we have created.

Color class

- AWT provides **java.awt.Color** class that enables to specify and work with different colors.
- Color class defines some constant to specify a number of common colors (for example - Color.RED/Color.red, Color.GREEN/ Color.green, Color.PINK/Color.pink etc).
- **Note** - Color class supports both uppercase and lowercase color constants.
- Color class also have constructors to specify our own colors.

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

- where r, g, b indicates the red, green and blue colors whose values must be between 0 and 255.
- If r, g or b are outside of the range 0 to 255 it will throw an **IllegalArgumentException**.
- **Example** -
- Color c1 = new Color(0, 0, 0); // Black color
- Color c2 = new Color(255, 255, 255); //White Color
- Color c3 = new Color(255,0,0) //Red Color
- We can change the color of graphics we are drawing by using Graphics class **setColor()** method.

public abstract void setColor(Color c)

- Where c is new rendering color.

Graphics, Color and Font Class Example

```
import java.applet.*;
import java.awt.*;
/*
<applet code="Applet3"      height=500 width=500>
</applet>
*/
public class Applet3 extends Applet
{
    Font f1,f2;
    Color c1;
    public void init()
    {
        f1=new Font("Arial",Font.ITALIC,30);
        f2=new Font("TimesRoman",Font.BOLD,20);
        c1 = new Color(0, 200, 0);
    }
}
```

```

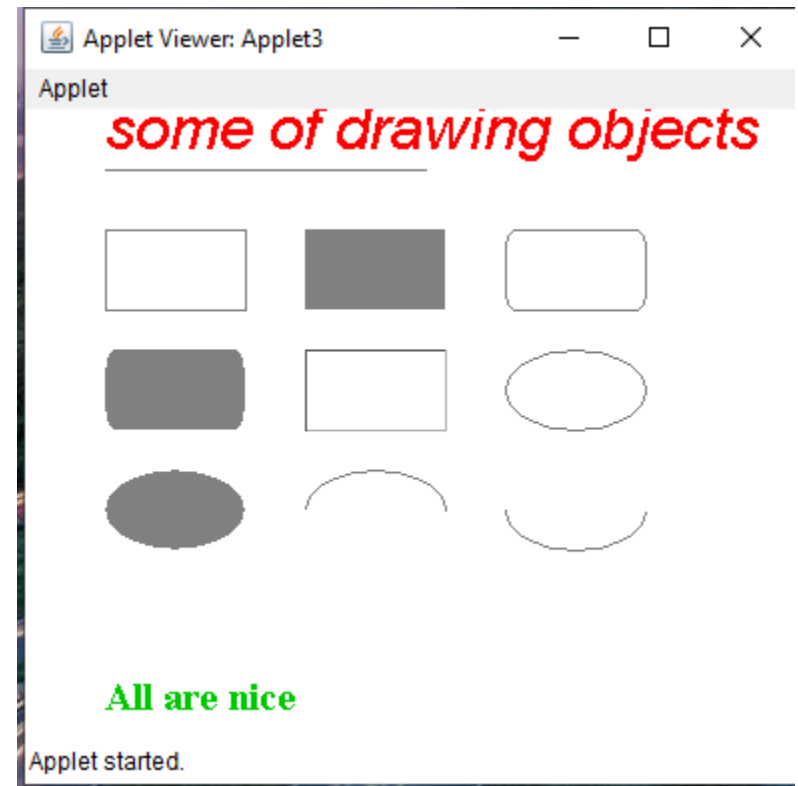
public void paint(Graphics g)
{
    g.setFont(f1);
    g.setColor(Color.red);
    g.drawString("some of drawing objects",40,20);
    g.setColor(Color.gray);
    g.drawLine(40,30,200,30);
    g.drawRect(40,60,70,40);
    g.fillRect(140,60,70,40);
    g.drawRoundRect(240,60,70,40,10,20);
    g.fillRoundRect(40,120,70,40,10,20);
    g.draw3DRect(140,120,70,40,false);
    g.drawOval(240,120,70,40);
    g.fillOval(40,180,70,40);
    g.drawArc(140,180,70,40,0,180);
    g.drawArc(240,180,70,40,0,-180);
    g.setFont(f2);
    g.setColor(c1);
    g.drawString("All are nice",40,300);
}
}

```

```

C:\ Command Prompt - appletviewer Applet3.java
G:\JAVA_PGMS>javac Applet3.java
G:\JAVA_PGMS>appletviewer Applet3.java

```



Images

- Image files (.gif, .jpeg, .bmp, etc) can be loaded into program and displayed
- Image files can be displayed with relative ease and image files can be edited (or) modified independent of the program file without altering the program.
- The java.applet.Applet class provides getImage() method that returns the object of Image that load image from a URL.

public Image getImage(URL u, String image)

- URL → specifies the location from which the image is to be loaded
 - String → Name of the image file
- The java.awt.Graphics class provides a method drawImage() to display the image at the given coordinates.

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

- ImageObserver → is usually the applet itself

Images Cont'd

- Other required methods of Applet class to display image:
- **public URL getDocumentBase()**
- **public URL getCodeBase()**
- The directory which contains the HTML file that started the applet (known as the **document base**).
- The other one is the directory from which the class file of the applet is loaded (known as the **code base**).
- These directories can be obtained as URL objects by using `getDocumentBase ()` and `getCodeBase ()` methods respectively
- `Image image1 = getImage(getCodeBase(), "image1.gif");`
`Image image2 = getImage(getDocumentBase(), "image1.jpeg");`

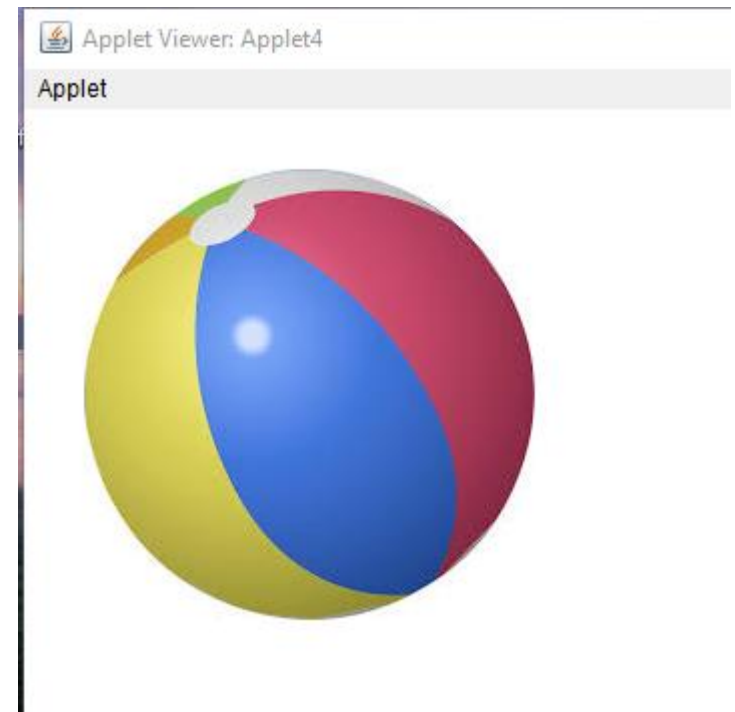
Displaying image in applet Example

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Applet4" height=500 width=500>
</applet>
*/
public class Applet4 extends Applet
{
    Image img;
    public void init()
    {
        img = getImage(getCodeBase(),"ball.jpg");
    }
    public void paint(Graphics g)
    {
        g.drawImage(img, 30,30, this);
    }
}
```

```
Command Prompt - appletviewer Applet4.java

G:\JAVA_PGMS>javac Applet4.java

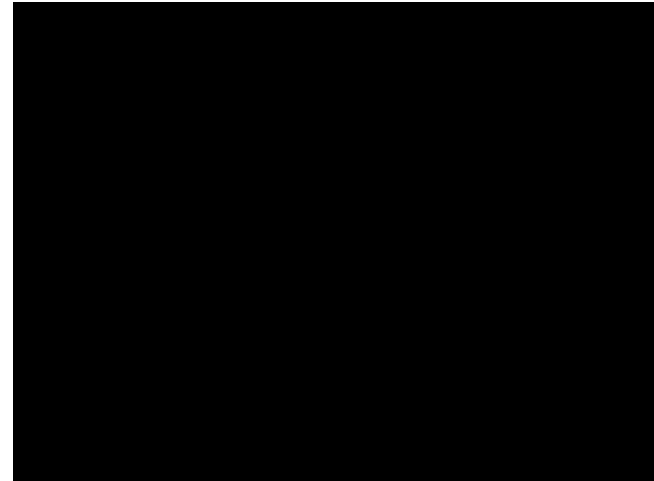
G:\JAVA_PGMS>appletviewer Applet4.java
```



Animation

- It is a **technique by which an object is moved on the screen**
- The object to be moved can be simple drawing (Or) a complex image loaded from an image file
- This object in animation jargon, is called a frame of animation
- Animation involves following steps,
 1. The object is drawn at a location on the screen
 2. The object's location is changed- the object at the old location is erased
 3. The object is repainted at the new location
- By repeating the above steps, an illusion of an object moving from one part of the screen to another can be created.
- Animation involves a loop in which the object gets painted, erased, and again repainted.
- As long as the loop lasts, the animation consumes all the processor time allocated to the program.

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Applet5" height=500 width=500>
</applet>
*/
public class Applet5 extends Applet
{
    Image img;
    public void init()
    {
        img =getImage(getDocumentBase(),"ball.jpg");
    }
    public void paint(Graphics g)
    {
        for(int i=0,j=0;i<500;i++,j=j+5)
        {
            g.drawImage(img, i,j, this);
            try{
                Thread.sleep(100);
            }catch(Exception e){}
        }
    }
}
```



Events

- **Changing the state of an object is known as an event.**
- In any interactive environment, the program should be able to respond to actions performed by the user.
- These actions can be mouse click, key press (or) selection of a menu item.
- Event handling has three main components,
 - **Events** : An event is a **change in state of an object**. For example when we press a button in our program or Android application the state of the button changes from 'Unclicked' to 'Clicked'. This change in the state of our button is called an Event.
 - **Events Source** : Event source is an object that generates an event. In Java, nearly everything is an object. The button we press is an object too. Source is the object which generates an event.
 - **Listeners** : A listener is an object that listens to the event. A listener gets notified when an event occurs. Listeners are also called as **event handlers** as they are the ones responsible to handle events occurring at the source. Listeners are interfaces and different types of listeners are used according to the event.
- The java.awt.event package provides many event classes and Listener interfaces for event handling.

Important Event Classes and Listener 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 and also when it enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener

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

How Events are handled?

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**

Mouse Events

- These are generated whenever a mouse is moved, clicked, pressed, release etc.
- It has methods to get coordinates at which mouse event has occurred.

```
public interface MouseListener
```

```
{
```

```
    public abstract void mouseClicked(MouseEvent e);
```

```
    public abstract void mouseEntered(MouseEvent e);
```

```
    public abstract void mouseExited(MouseEvent e);
```

```
    public abstract void mousePressed(MouseEvent e);
```

```
    public abstract void mouseReleased(MouseEvent e);
```

```
}
```

```
public interface MouseMotionListener
```

```
{
```

```
    public abstract void mouseDragged(MouseEvent e);
```

```
    public abstract void mouseMoved(MouseEvent e);
```

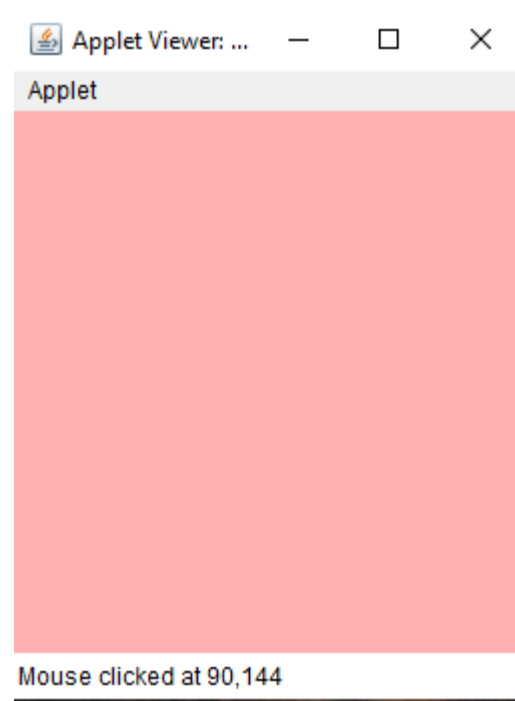
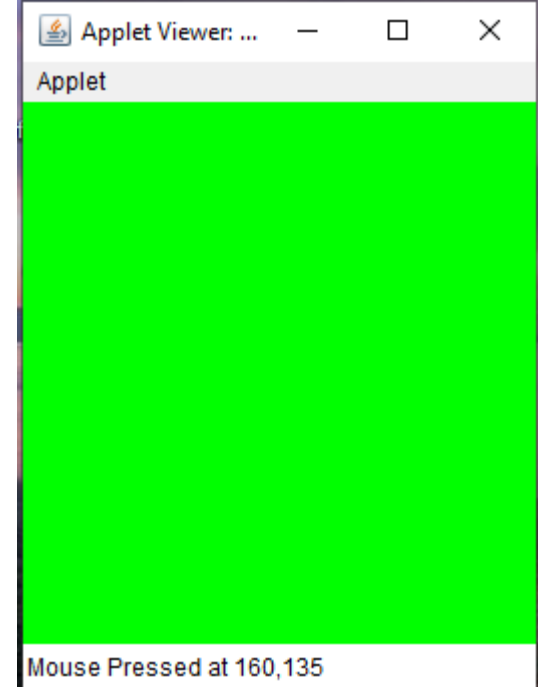
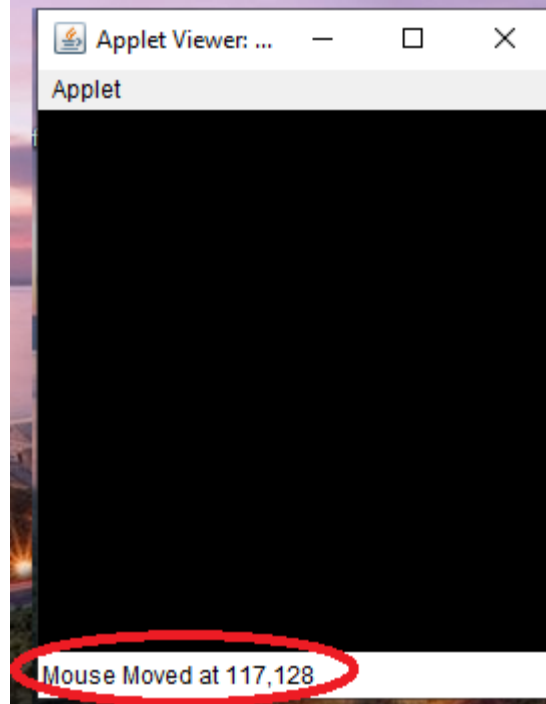
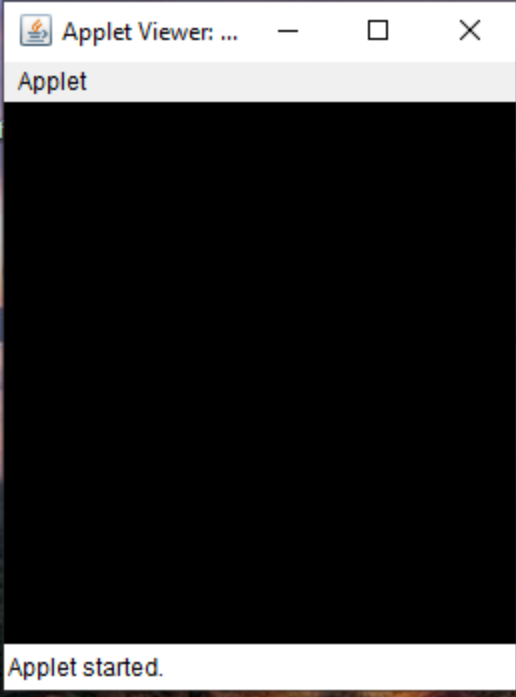
```
}
```

Mouse Event handling Example

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*
<applet code="MouseEventEx.class"    height=500 width=500>
</applet>
*/
public class MouseEventEx extends Applet implements MouseListener, MouseMotionListener
{
    public void init()
    {
        addMouseListener(this);
        addMouseMotionListener(this);
        setBackground(Color.black);
    }
    public void mouseClicked(MouseEvent e)
    {
        setBackground(Color.pink);
        showStatus("Mouse clicked at "+e.getX()+" "+e.getY());
    }
    public void mouseEntered(MouseEvent e)
    {
        showStatus("Mouse Entered at "+e.getX()+" "+e.getY());
    }
}
```

```
public void mouseExited(MouseEvent e)
{
    showStatus("Mouse Exited at "+e.getX()+" "+e.getY());
}
public void mousePressed(MouseEvent e)
{
    setBackground(Color.green);
    showStatus("Mouse Pressed at "+e.getX()+" "+e.getY());
}
public void mouseReleased(MouseEvent e)
{
    showStatus("Mouse Released at "+e.getX()+" "+e.getY());
}
public void mouseMoved(MouseEvent e)
{
    showStatus("Mouse Moved at "+e.getX()+" "+e.getY());
}
public void mouseDragged(MouseEvent e)
{
    showStatus("Mouse Dragged at "+e.getX()+" "+e.getY());
}
```

```
}
```



Keyboard Events

- These are generated on pressing (or) releasing a key

```
public void KeyListener
```

```
{
```

```
    public abstract void keyPressed(KeyEvent e);
```

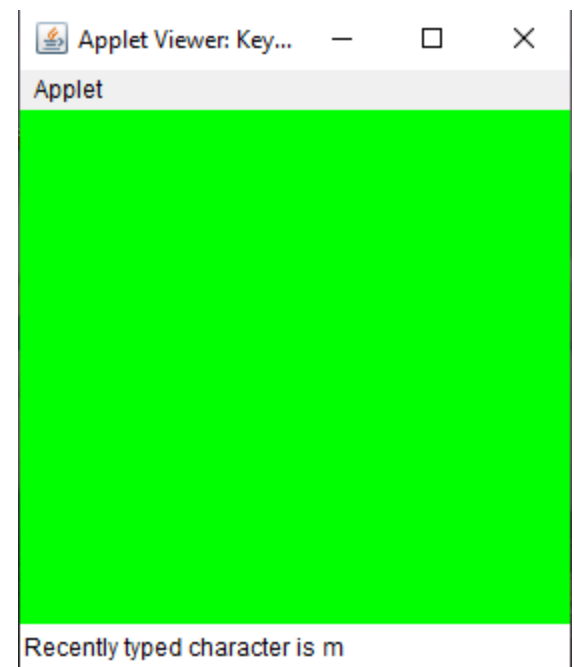
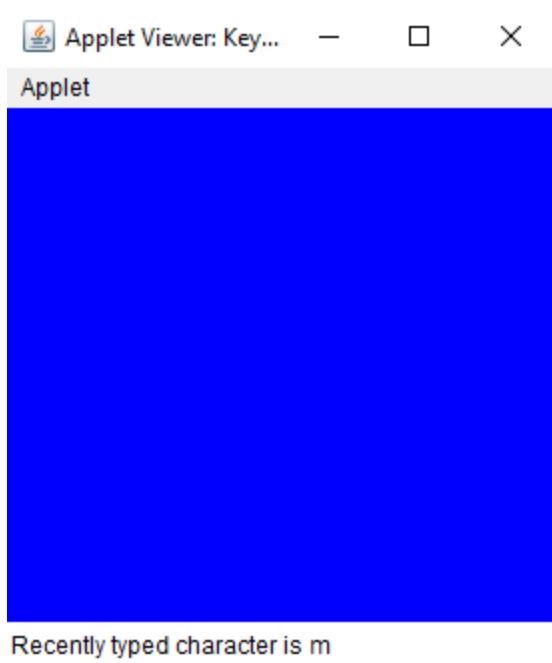
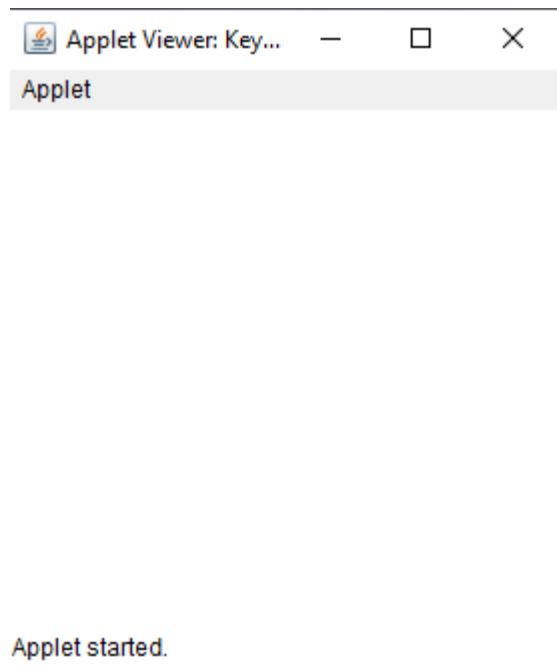
```
    public abstract void keyReleased(KeyEvent e);
```

```
    public abstract void keyTyped(KeyEvent e);
```

```
}
```

Keyboard Event handling Example

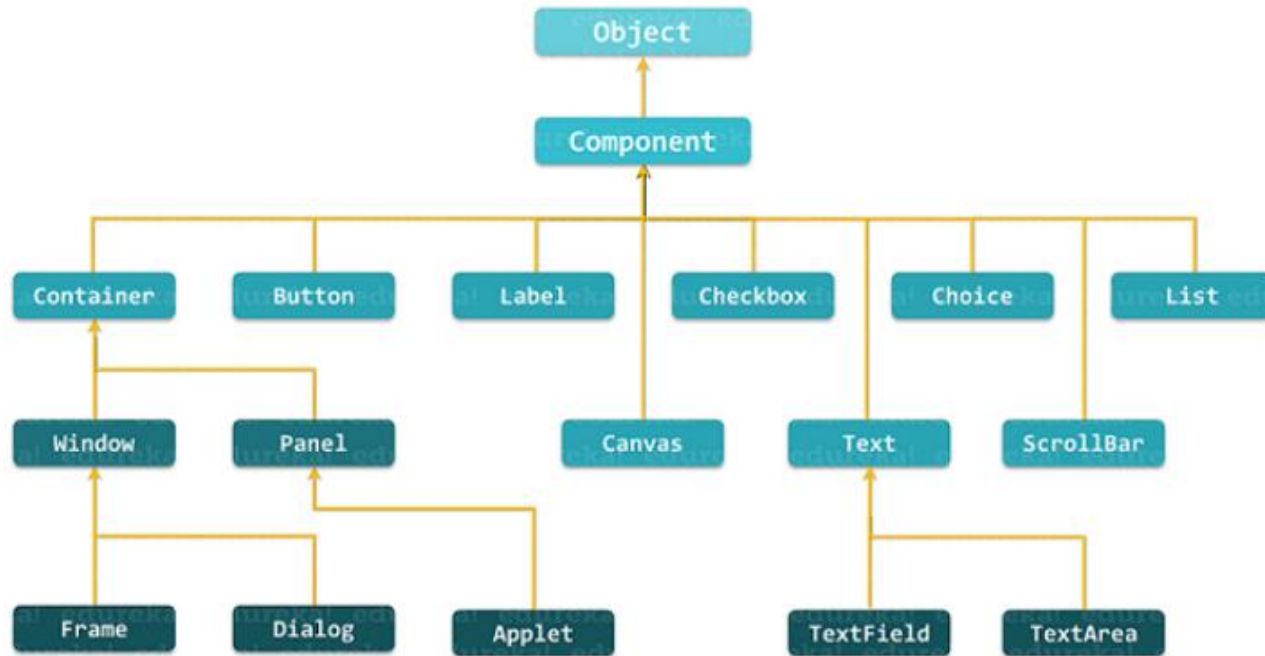
```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*<applet code="KeyboardEventEx.class" height=500 width=500></applet>*/
public class KeyboardEventEx extends Applet implements KeyListener
{
    public void init()
    {
        addKeyListener(this);
    }
    public void keyPressed(KeyEvent e)
    {
        setBackground(Color.blue);
    }
    public void keyReleased(KeyEvent e)
    {
        setBackground(Color.green);
    }
    public void keyTyped(KeyEvent e)
    {
        showStatus("Recently typed character is "+e.getKeyChar());
    }
}
```

Java AWT (Abstract Window Toolkit)

- It is an **API to develop GUI** or window-based applications in java
- It is a part of JFC (Java Foundation Classes) developed by Sun Microsystems
- The AWT API in Java primarily **consists of a comprehensive set of classes and methods that are required for creating and managing the Graphical User Interface(GUI)** in a simplified manner.
- **It was developed for providing a common set of tools for designing the cross-platform GUIs.**
- One of the important features of AWT is that it is **platform dependent**.
- Any UI will be made of three entities:
 - **UI elements:** These refers to the core visual elements which are visible to the user and used for interacting with the application. AWT in Java provides a comprehensive list of widely used and common elements.
 - **Layouts:** These define how UI elements will be organized on the screen and provide the final look and feel to the GUI.
 - **Behavior:** These define the events which should occur when a user interacts with UI elements.

Hierarchy Of AWT



- All the elements like buttons, text fields, scrollbars etc are known as components.
- In AWT we have classes for each component as shown in the above diagram.
- To have everything placed on a screen to a particular position, we have to add them to a container.
- A container is like a screen wherein we are placing components like buttons, text fields, checkbox etc.
- In short a container contains and controls the layout of components.
- A container itself is a component (shown in the above hierarchy diagram) thus we can add a container inside container.

Types of containers

- A **container** is a **place wherein we add components** like text field, button, checkbox etc.
- It is a **subclass of `java.awt.Component`** and is responsible for keeping a track of components being added.
- There are four types of containers provided by AWT in Java.
 1. **Window:** An instance of the Window class has no border and no title
 2. **Dialog:** Dialog class has border and title. An instance of the Dialog class cannot exist without an associated instance of the Frame class.
 3. **Panel:** Panel does not contain title bar, menu bar or border. It is a generic container for holding components. An instance of the Panel class provides a container to which to add components.
 4. **Frame:** A frame has title, border and menu bars. It can contain several components like buttons, text fields, scrollbars etc. This is most widely used container while developing an application in AWT. We can create a GUI using Frame in two ways:
 - 1) By extending Frame class
 - 2) By creating the instance of Frame class

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.

AWT Button Class

Test Button

- java.awt.Button class is used to create a labelled button.
- It is a GUI component that triggers a certain programmed action upon clicking it.
- The Button class has two constructors:
 1. `public Button(String btnLabel);` // Construct a Button with the given label
 2. `public Button();` // Construct a Button with empty label
- A few of the methods provided by this class have been listed below:

//Get the label of this Button instance

```
public String getLabel();
```

//Set the label of this Button instance

```
public void setLabel(String btnLabel);
```

//Enable or disable this Button. Disabled Button cannot be clicked

```
public void setEnable(boolean enable);
```

AWT Text Field class

Test TextField

- A java.awt.TextField class creates a single-line text box for users to enter texts.
- The TextField class has three constructors which are:
 - //Construct a TextField instance with the given initial text string with the number of columns.
`public TextField(String initialText, int columns);`
 - //Construct a TextField instance with the given initial text string.
`public TextField(String initialText);`
 - //Construct a TextField instance with the number of columns.
`public TextField(int columns);`
- A few of the methods provided by TextField class are:
 - // Get the current text on this TextField instance
`public String getText();`
 - // Set the display text on this TextField instance
`public void setText(String strText);`
 - //Set this TextField to editable (read/write) or non-editable (read-only)
`public void setEditable(boolean editable);`

AWT Label class

Test Label

- The java.awt.Label class provides a descriptive text string that is visible on GUI.
- An AWT Label object is a component for placing text in a container.
- Label class has three constructors which are:

// Construct a Label with the given text String, of the text alignment

public Label(String strLabel, int alignment);

//Construct a Label with the given text String

public Label(String strLabel);

//Construct an initially empty Label

public Label();

- This class also provides 3 constants which are:

public static final LEFT; // Label.LEFT

public static final RIGHT; // Label.RIGHT

public static final CENTER; // Label.CENTER

- public methods provided by this class:

public String getText();

public void setText(String strLabel);

public int getAlignment();

//Label.LEFT, Label.RIGHT, Label.CENTER

public void setAlignment(int alignment);

AWT Checkbox class



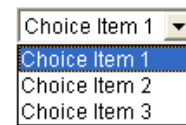
- Checkboxes have two states, on and off.
- The state of the button is returned as the Object argument, when a Checkbox event occurs.
 - **Checkbox()** → Constructs a Checkbox with an empty label. The check box starts in a false state and is not part of any check box group.
 - **Checkbox(String)** → Constructs a Checkbox with the specified label. The check box starts in a false state and is not part of any check box group.
 - **Checkbox(String, boolean)** → Constructs a Checkbox with the specified label. The check box starts in the specified state and is not part of any check box group.
 - **Checkbox(String, boolean, CheckboxGroup)** → Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.
 - **Checkbox(String, CheckboxGroup, boolean)** → Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.
 - **getState()** → returns a true or false value.

Radio Buttons AWT CheckboxGroup

☐ CB Item 1 ☐ CB Item 2 ☒ CB Item 3

- It Is a group of checkboxes, where only one of the items in the group can be selected at any one time.
- This class is used to create a multiple-exclusion scope for a set of Checkbox buttons.
- For example, creating a set of Checkbox buttons with the same CheckboxGroup object means that only one of those Checkbox buttons will be allowed to be "on" at a time.“
- **CheckboxGroup()** → Creates a new CheckboxGroup.

AWT Choice class



- The **Choice** object is more like what other IDEs might call a *list box* that operates like a *pull-down menu*.
- The selected choice is shown on the top of the given menu.
- **Choice()** → constructs a new choice
- **add(String)** → Adds an item to this Choice.
- **addItem(String)** → Adds an item to this Choice.
- **addItemListener(ItemListener)** → Adds the specified item listener to receive item events from this choice.
- **getItem(int)** → Returns the String at the specified index in the Choice.
- **getItemCount()** → Returns the number of items in this Choice.
- **getSelectedIndex()** → Returns the index of the currently selected item.
- **getSelectedItem()** → Returns a String representation of the current choice.
- **getSelectedObjects()** → Returns an array (length 1) containing the currently selected item.
- **insert(String, int)** → Inserts the item into this choice at the specified position.
- **remove(int)** → Removes an item from the choice menu.
- **remove(String)** → Remove the first occurrence of item from the choice menu.
- **removeAll()** → Removes all items from the choice menu.
- **removeItemListener(ItemListener)** → Removes the specified item listener so that it no longer receives item events from this choice.
- **select(int)** → Selects the item with the specified position.
- **select(String)** → Selects the item with the specified String.

```
import java.awt.*;
import java.awt.event.*;
/*
<applet code="SimpleAWT" height=500 width=500> </applet>
*/
public class SimpleAWT extends java.applet.Applet
implements ActionListener, ItemListener
{
    private Button button = new Button("Push Me!");
    private Checkbox checkbox = new Checkbox("Check Me!");
    private Choice choice = new Choice();
    private Label label = new Label("Pick something!");
    public void init()
    {
        button.addActionListener(this);
        checkbox.addItemListener(this);
        choice.addItemListener(this);
        // An Applet is a Container because it extends Panel.
        setLayout(new BorderLayout());
        choice.addItem("Red");
        choice.addItem("Green");
        choice.addItem("Blue");
    }
}
```

```

        Panel panel = new Panel();
        panel.add(button);
        panel.add(checkbox);
        panel.add(choice);
        add(label, "Center");
        add(panel, "South");
    }

    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == button)
        {
            label.setText("The Button was pushed.");
        }
    }

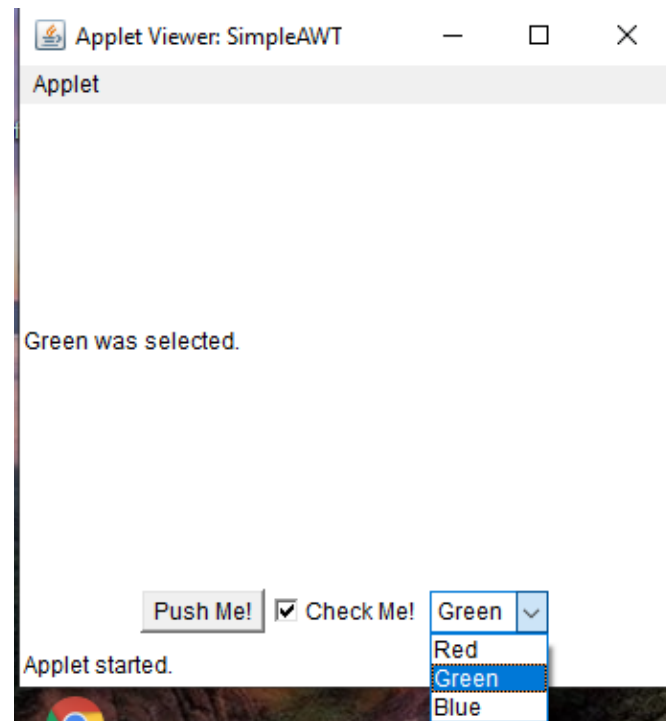
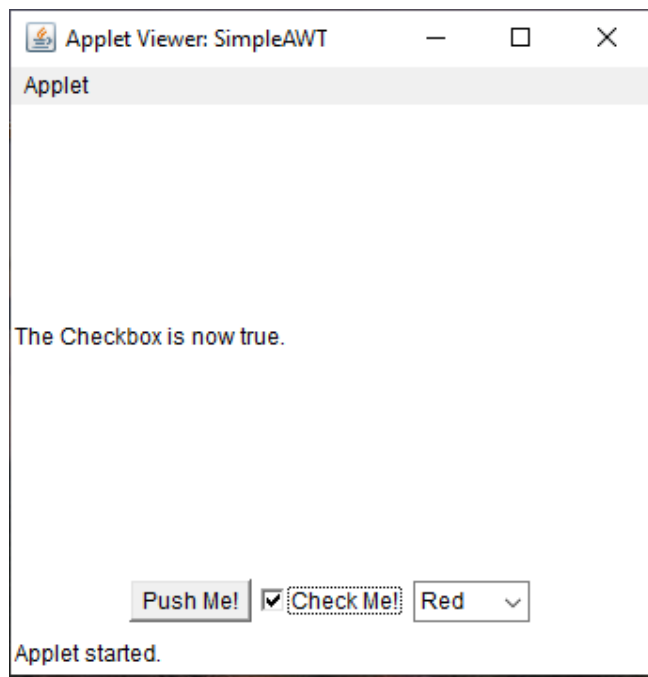
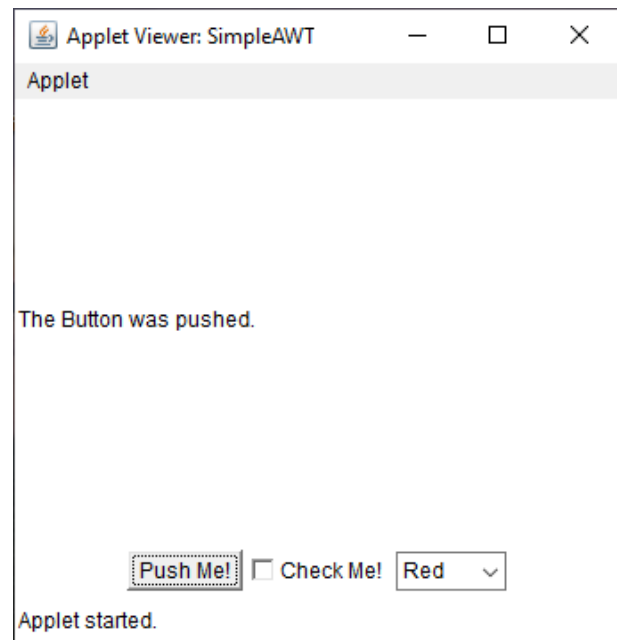
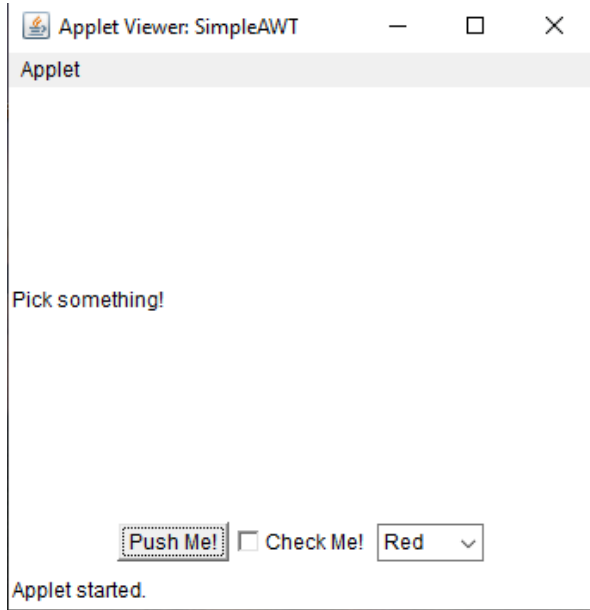
    public void itemStateChanged(ItemEvent e)
    {
        if (e.getSource() == checkbox)
        {
            label.setText("The Checkbox is now " + checkbox.getState() + ".");
        } else if (e.getSource() == choice)
        {
            label.setText(choice.getSelectedItem() + " was selected.");
        }
    }
}

```

```

}

```



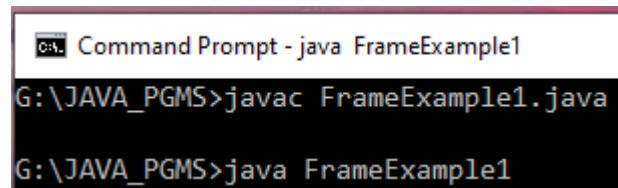
Creating GUI using Frame

- Two ways:
 - 1) By extending Frame class
 - 2) By creating the instance of Frame class

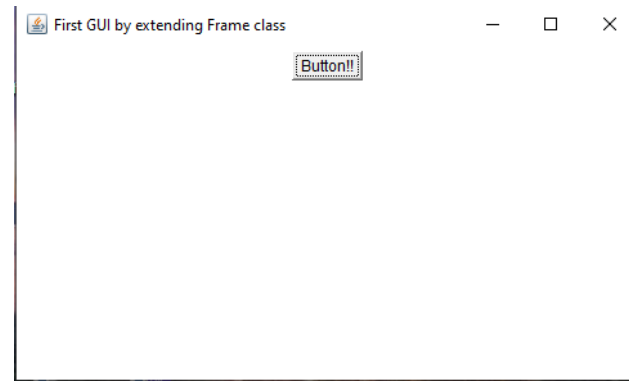
By extending Frame class

```
import java.awt.*;

public class FrameExample1 extends Frame
{
    FrameExample1()
    {
        Button b=new Button("Button!!");
        b.setBounds(50,50,50,50); // setting button position on screen
        add(b); //adding button into frame
        setSize(500,300); //Setting Frame width and height
        setTitle("First GUI by extending Frame class"); //Setting the title of Frame
        setLayout(new FlowLayout()); //Setting the layout for the Frame
        setVisible(true); /* By default frame is not visible so we are setting the visibility to true
                           to make it visible */
    }
    public static void main(String args[])
    {
        FrameExample1 fr=new FrameExample1(); //// Creating the instance of Frame
    }
}
```



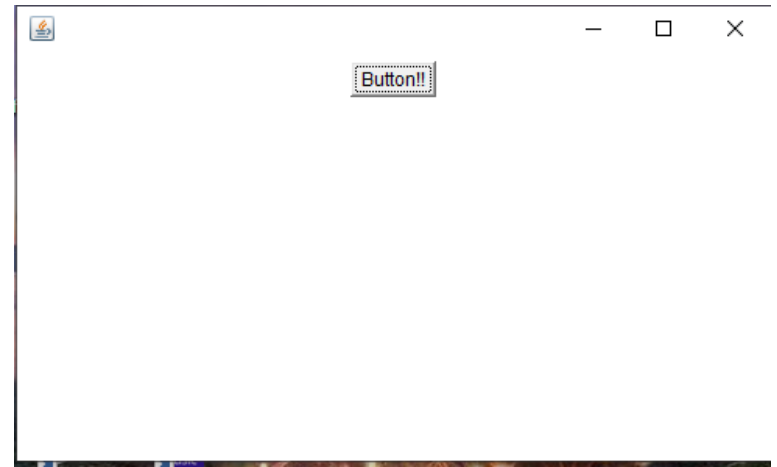
```
Command Prompt - java FrameExample1
G:\JAVA_PGMS>javac FrameExample1.java
G:\JAVA_PGMS>java FrameExample1
```



Creating Frame by creating instance of Frame class - Example

```
import java.awt.*;  
public class FrameExample2  
{  
    FrameExample2()  
    {  
        Frame fr=new Frame(); //Creating Frame  
        Button b=new Button("Button!!");  
        fr.add(b); //adding button to the frame  
        fr.setSize(500, 300); //setting frame size  
        fr.setLayout(new FlowLayout()); //Setting the layout for the Frame  
        fr.setVisible(true);  
    }  
    public static void main(String args[])  
    {  
        FrameExample2 ex = new FrameExample2();  
    }  
}
```

```
C:\> Command Prompt - java FrameExample2  
G:\JAVA_PGMS>javac FrameExample2.java  
G:\JAVA_PGMS>java FrameExample2
```



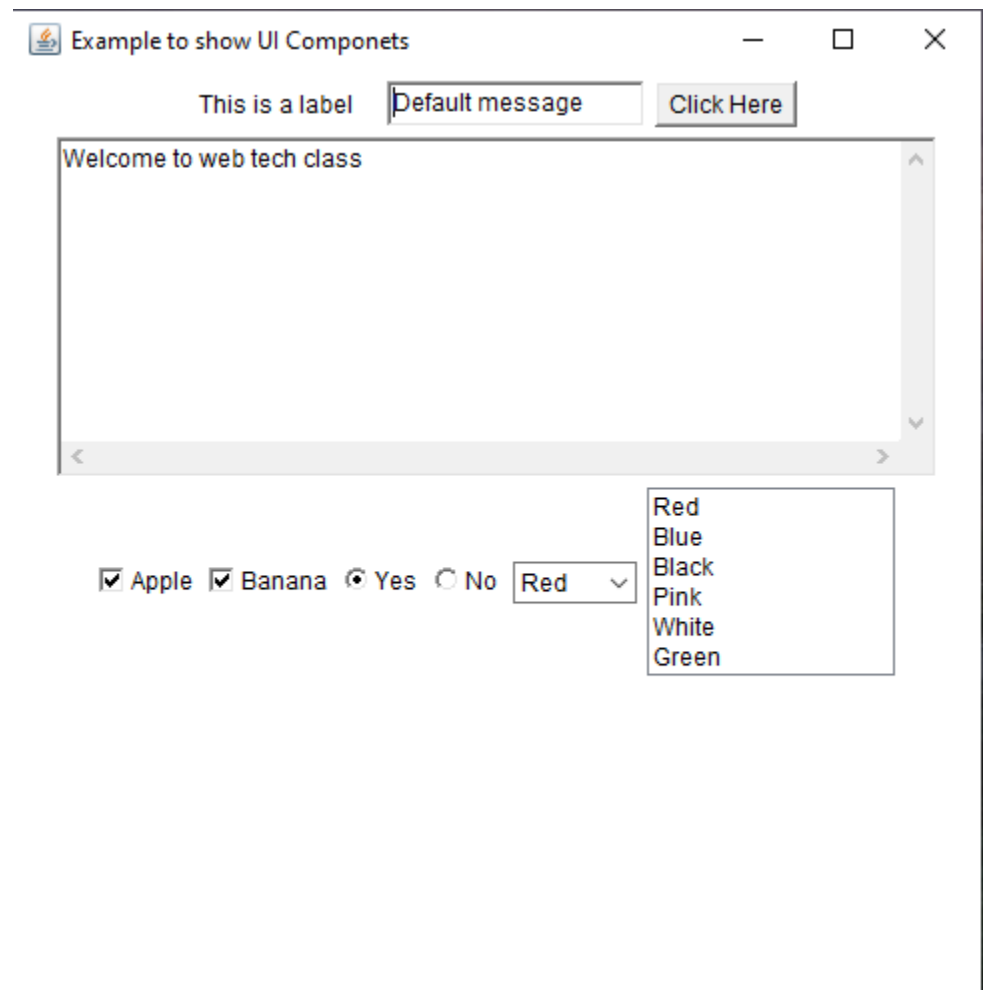
```
import java.awt.*;

public class FrameExample3 extends Frame
{
    FrameExample3()
    {
        setTitle("Example to show UI Componets");
        setSize(500,500);
        setLayout(new FlowLayout());
        Label lb=new Label("This is a label");
        add(lb);
        TextField tf=new TextField("Default message");
        add(tf);
        Button b=new Button("Click Here");
        add(b);
        TextArea ta=new TextArea("Welcome to web tech class");
        add(ta);
        Checkbox cb1=new Checkbox("Apple",true);
        Checkbox cb2=new Checkbox("Banana",true);
        add(cb1);
        add(cb2);
        CheckboxGroup obj = new CheckboxGroup();
        Checkbox r1 = new Checkbox("Yes", obj, true);
        Checkbox r2 = new Checkbox("No", obj, false);
        add(r1); add(r2);
    }
}
```

```

Choice ch=new Choice();
ch.add("Red");
ch.add("Blue");
ch.add("Black");
ch.add("Pink");
ch.add("White");
ch.add("Green");
add(ch);
List l=new List(6);
l.add("Red");
l.add("Blue");
l.add("Black");
l.add("Pink");
l.add("White");
l.add("Green");
add(l);
setVisible(true);
}
public static void main(String args[])
{
    FrameExample3 fr=new FrameExample3();
}
}

```



```

C:\> Command Prompt - java FrameExample3
G:\JAVA_PGMS>javac FrameExample3.java
G:\JAVA_PGMS>java FrameExample3

```

Simple Calculator Example

```
import java.awt.*;
import java.awt.event.*;
class Calculator extends Frame implements ActionListener
{
    Label lb1,lb2,lb3,msg;
    TextField txt1,txt2,txt3;
    Button btn1,btn2,btn3,btn4,btn5,btn6,btn7;

    public Calculator()
    {
        lb1 = new Label("Number 1");
        lb2 = new Label("Number 2");
        lb3 = new Label("Result");
        msg=new Label("");

        txt1 = new TextField(10);
        txt2 = new TextField(10);
        txt3 = new TextField(10);

        btn1 = new Button("ADD");
        btn2 = new Button("SUB");
        btn3 = new Button("MUL");
        btn4 = new Button("DIV");
```

```
btn5 = new Button("MOD");  
btn6 = new Button("RESET");  
btn7 = new Button("CLOSE");
```

```
txt1.setText("0");  
txt2.setText("0");  
txt3.setText("0");
```

```
add(lb1);  
add(txt1);  
add(lb2);  
add(txt2);  
add(lb3);  
add(txt3);  
add(btn1);  
add(btn2);  
add(btn3);  
add(btn4);  
add(btn5);  
add(btn6);  
add(btn7);
```

```
add(msg);
```

```
setSize(800,200);  
setTitle("Simple Calculator");  
setLayout(new FlowLayout());
```

```
btn1.addActionListener(this);  
btn2.addActionListener(this);  
btn3.addActionListener(this);  
btn4.addActionListener(this);  
btn5.addActionListener(this);  
btn6.addActionListener(this);  
btn7.addActionListener(this);
```

```
addWindowListener(new WindowAdapter()  
    {  
        public void windowClosing(WindowEvent e)  
        {  
            dispose();  
        }  
    });  
}
```

```
public void actionPerformed(ActionEvent ae)
{
    double n1=0,n2=0,n3=0;
    try
    {
        n1 = Double.parseDouble(txt1.getText());
    } catch (NumberFormatException e)
    {
        txt1.setText("Invalid input");
    }
    try
    {
        n2 = Double.parseDouble(txt2.getText());
    } catch (NumberFormatException e)
    {
        txt2.setText("Invalid input");
    }
    if(ae.getSource()==btn1)
    {
        n3=n1+n2;
        txt3.setText(String.valueOf(n3));
    }
}
```

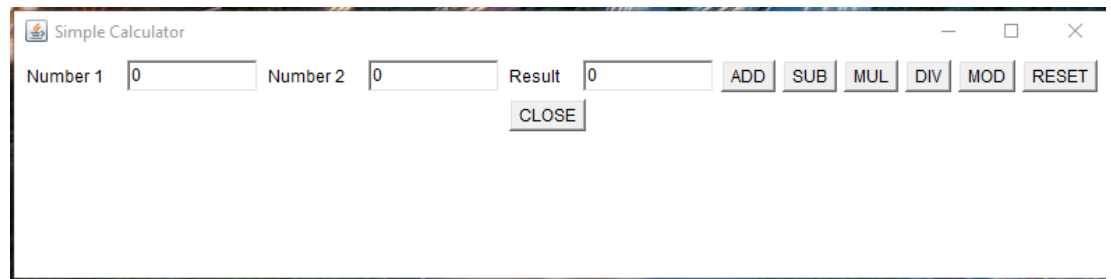
```
if(ae.getSource()==btn2)
{
    n3=n1-n2;
    txt3.setText(String.valueOf(n3));
}
if(ae.getSource()==btn3)
{
    n3=n1*n2;
    txt3.setText(String.valueOf(n3));
}
if(ae.getSource()==btn4)
{
    if (n2>0)
    {
        n3=n1/n2;
        txt3.setText(String.valueOf(n3));
    }
    else
        msg.setText("DENOMINATOR SHOULD NOT BE ZERO");
}
```



```

if(ae.getSource()==btn5)
{
    n3=n1%n2;
    txt3.setText(String.valueOf(n3));
}
if(ae.getSource()==btn6)
{
    txt1.setText("0");
    txt2.setText("0");
    txt3.setText("0");
}
if(ae.getSource()==btn7)
{
    System.exit(0);
}
}

```



```

public static void main(String[] args)
{
    Calculator calC = new Calculator();
    calC.setVisible(true);
    calC.setLocation(300,300);
}
}

```

Disadvantage of AWT

- AWT is **platform dependent** and **heavy-weight**
- **Platform dependent means,**
 - Java AWT calls **native platform (Operating systems) subroutine for creating components** such as textbox, checkbox, button etc.
 - For example an AWT GUI having a button would **have a different look and feel across platforms like windows, Mac OS & Unix**, this is because these platforms have different look and feel for their native buttons and AWT directly calls their native subroutine that creates the button.
 - In simple, an application build on AWT would look like a windows application when it runs on Windows, but the same application would look like a Mac application when runs on Mac OS.
- **heavy weight because ,**
 - they are being **generated by underlying operating system (OS)**.
 - For example if we are instantiating a text box in AWT that means we are actually asking OS to create a text box for us.

How to Solve this Problem?

Solution

Swing