

Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs using the **Applet Viewer** or inside the any **web browser** that supports Java. An applet, like any application program, can do many things for us. It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation and play video.

Local and Remote Applets

We can embed applets into Web pages in two ways, One, we can write our applets and embed them into web pages, Second, we can download an applet from a remote computer system and then embed it into a web page.

An applet developed locally and stored in a local system is known as a *local applet*. When a web page is trying to find a local applet, it does not need internet and therefore the local system does not require the internet connection. It simply searches the directories in the local system and locates and loads the specified applet.

A remote applet is that which is developed by someone else and stored on a web server connected to the internet. We can download the remote applet onto our system via at the internet and run it.

Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured

It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

Restrictions of Applets

1. Applets are required separate compilation before opening in a browser.
2. In real-time environment, the bytecode of applet is to be downloaded from the server to the client machine.
3. Applets are treated as **untrusted** (as they were developed by unknown people and placed on unknown servers whose trustworthiness is not guaranteed) and for this reason they are not allowed, as a security measure, to access any system resources like file system etc. available on the client system.
4. Applets are not permitted to use any system resources like file system as they are untrusted and can inject virus into the system.
5. Applets cannot read from or write to hard disk files.
6. Applet methods cannot be native.
7. Applets should not attempt to create socket connections.(Cannot contact to other server) .
8. Applets cannot read system properties.
9. Applets cannot use any software available on the system (except browser execution area)

How Applets Differ From Applications

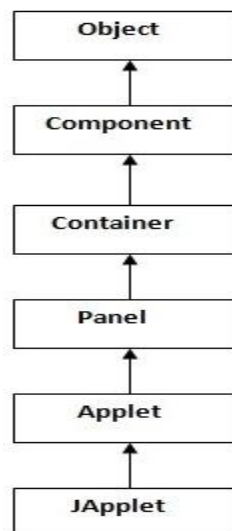
Applet is a Java program executed by a browser. The position of applets in software world is they occupy the client-side position in Web communication. On the server-side, you guess, another Java program comes, **Servlets**. Applets on client-side and servlets on server-side makes Java a truly "**Internet-based language**". To execute applets, the browsers come with JRE (Java Runtime Environment). The browsers with Java Runtime Environment (or to say, JVM) loaded are known as **Java enabled browsers**.

Note: Browser do not have a Java compiler as a compiled applet file (.class file) is given to browser to execute.

You have seen two main differences between applications and applets. Let us summarize them. -

Feature	Application	Applet
main() method	Present	Not present
Execution	Requires JRE	Requires a browser like Chrome
Nature	Called as stand-alone application as application can be executed from command prompt	Requires some third party tool help like a browser to execute
Restrictions	Can access any data or software available on the system	cannot access any thing on the system except browser's services
Security	Does not require any security	Requires highest security for the system as they are untrusted

Hierarchy of Applet



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

Lifecycle of an Applet:

For creating any applet **java.applet.Applet** class must be inherited. It provides 4 life cycle methods of applet.

Initialized State:

Applet enters the initialization state when it is first loaded. This is achieved by calling the **init()** method of Applet class. In this method, we may do the following, if required.

- Create objects needed by the applet
- Declaring & initialized variable.
- Load images.
- setting background and foreground colors in GUI etc.

It will be occur in only once in the applet life cycle.

```
public void init( )  
{  
    .....  
}
```

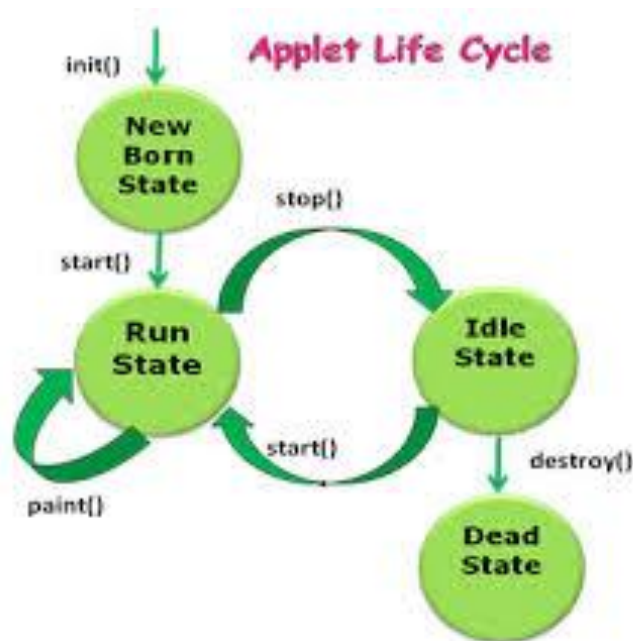


Figure 10.1 An applet state transition Diagram

Running State

Applet enters the running state when the system call the **start()** method of **Applet** class. This occurs automatically after the applet is initialized. Starting can also occur if the applet is already

“Stopped” (idle) state. For example, we may leave the web page containing the applet temporarily to another page and return back to the page. This again starts the applet running. Note that, unlike **init()** method, the **start()** method may be called more than once.

```
public void start( )
{
.....
}
```

Idle or Stopped State

An applet becomes idle when it is stopped from running. Stopping occurs automatically when we leave the page containing the currently running applet. We can also do so by calling **stop()** method explicitly.

```
public void stop( )
{
.....
}
```

Dead State

An applet is said to be dead state when it is removed from memory. This occurs automatically by invoking the **destroy()** method when we quit from the browser. Like initialization, destroying stage occurs only once in the applet life cycle. If an applet has created any resources, like threads, we may override the **destroy()** method to clean up the resources.

```
public void destroy( )
{
.....
}
```

Display State

Applet moves to the display state whenever it has to perform some output operation on the screen. This happens immediately after the applet enters into the running state. The **paint()** method is called to accomplish this task. Almost every applet will have a **paint()** method. It is used to display anything on the screen.

This method is defined by the AWT and must be overridden by the applet. **paint()** is called each time that the applet must redisplay its output.

It is to be noted that the display state is not considered as a part of the applet life cycle. In fact, the **paint()** method is defined in the **Applet** class. It is inherited from the **Component** class, a super class of **Applet**.

```
public void paint(Graphics g )
{
.....
}
```

Overriding update()

In some situations, your applet may need to override another method defined by the AWT, called **update()**. This method is called when your applet has requested that a portion of its window be redrawn. The default version of **update()** simply calls **paint()**. However, you can override the **update()** method so that it performs more subtle repainting. In general, overriding **update()** is a specialized technique that is not applicable to all applets.

The 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.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("welcome",150,150);
    }
}
```

Note:

class must be public because its object is created by Java Plugin software (Java Compatible Browser) that resides on the browser.

Every applet that you create must be a subclass of Applet.

myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

Simple example of Applet by appletviewer tool:

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

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("welcome to applet",150,150);
    }
}

/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```




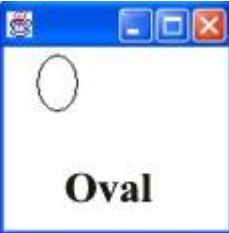

To execute the applet by appletviewer tool, write in command prompt:


```
c:\>javac First.java
c:\>appletviewer First.java
```

Displaying Graphics in Applet

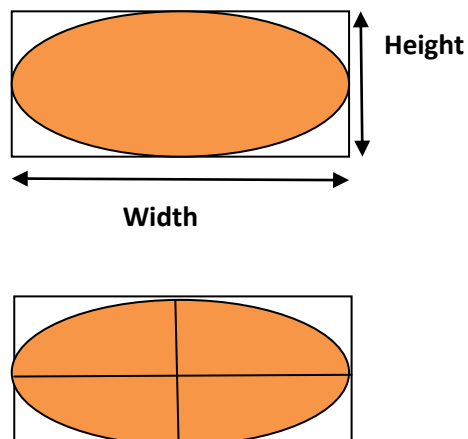
java.awt.Graphics class provides many methods for graphics programming.

Basic Graphing Methods:

Line	g.drawLine(35, 45, 75, 95); drawLine(int x1, int y1, int x2, int y2) Used to draw a straight line from point (x1,y1) to (x2,y2).	
Rectangle	g.drawRect(35, 45, 25, 35); drawRect(int x, int y, int width, int length) Used to draw a rectangle with the upper left corner at (x,y) and with the specified width and length.	
Round Edge Rectangle	g.drawRoundRect(35,45,25,35,10,10); drawRoundRect(int x, int y, int width, int length, int arcWidth, int arcHeight) Used to draw a rounded edged rectangle. The amount of rounding is controlled by arcWidth and arcHeight.	
Oval / Circle	g.drawOval(25, 35, 25, 35); g.drawOval(25, 35, 25, 25); → circle drawOval(int x, int y, int width, int length) Used to draw an oval inside an imaginary rectangle whose upper left corner is at (x,y). To draw a circle keep the width and length the same.	
Arc	g.drawArc(35, 45, 75, 95, 0, 90); drawArc(int x, int y, int width, int length, int startAngle, int arcAngle) Used to draw an arc inside an imaginary rectangle whose upper left corner is at (x,y). The arc is drawn from the startAngle to startAngle + arcAngle and is measured in degrees. A startAngle of 0° points horizontally to the right (like the unit circle in math).	

	Positive is a counterclockwise rotation starting at 0°.	
String (text)	g.drawString("Java is cool!", 40, 70); drawString(String str, int x, int y); Draws a string starting at the point indicated by (x,y). Be sure you leave enough room from the top of the screen for the size of the font.	

drawOval(20,20,200,120)



Example of Graphics in applet:

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

public class GraphicsDemo extends Applet
{

    public void paint(Graphics g)
    {
        g.setColor(Color.red);
        g.drawString("Welcome",50, 50);
        g.drawLine(20,30,20,300);
        g.drawRect(70,100,30,30);
        g.fillRect(170,100,30,30);
        g.drawOval(70,200,30,30);
    }
}
```



```
g.setColor(Color.pink);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);

}
}
```

myapplet.html

1. <html>
2. <body>
3. <applet code="GraphicsDemo.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>

To set the background color of an applet's window, use **setBackground()**. To set the foreground color (the color in which text is shown, for example), use **setForeground()**. These methods are defined by **Component**, and they have the following general forms:

```
void setBackground(Color newColor)
void setForeground(Color newColor)
```

Here, *newColor* specifies the new color. The class **Color** defines the constants shown here that can be used to specify colors:

Color.black	Color.magenta
Color.blue	Color.orange
Color.cyan	Color.pink
Color.darkGray	Color.red

A good place to set the foreground and background colors is in the **init()** method.

You can obtain the current settings for the background and foreground colors by calling **getBackground()** and **getForeground()**, respectively. They are also defined by **Component** and are shown here:

```
Color getBackground( )
Color getForeground( )
```

Example of Applet to Set foreground and background color of an Applet

```
/* A simple applet that sets the foreground and
background colors and outputs a string. */
import java.awt.*;
import java.applet.*;
/*
<applet code="Sample" width=300 height=50>
</applet>
*/
public class Sample extends Applet
{
    String msg;
    // set the foreground and background colors.
    public void init()
    {
        setBackground(Color.cyan);
        setForeground(Color.red);
        msg = "Inside init( ) --";
    }
    // Initialize the string to be displayed.
    public void start()
    {
        msg += " Inside start( ) --";
    }
    // Display msg in applet window.
    public void paint(Graphics g) {
        msg += " Inside paint( ).";
        g.drawString(msg, 10, 30);
    }
}
```

Using the Status Window

In addition to displaying information in its window, an applet can also output a message to the status window of the browser or applet viewer on which it is running. To do so, call **showStatus()** with the string that you want displayed. The status window is a good place to give the user feedback about what is occurring in the applet, suggest options, or possibly report some types of errors.

The following applet demonstrates **showStatus()**:

```
// Using the Status Window.
import java.awt.*;
import java.applet.*;
/*
<applet code="StatusWindow" width=300 height=50>
</applet>
```

```
*/  
public class StatusWindow extends Applet  
{  
    public void init()  
    {  
        setBackground(Color.cyan);  
    }  
    // Display msg in applet window.  
    public void paint(Graphics g)  
    {  
        g.drawString("This is in the applet window.", 10, 20);  
        showStatus("This is shown in the status window.");  
    }  
}
```

The HTML APPLET Tag

As mentioned earlier, Sun currently recommends that the APPLET tag be used to start an applet from both an HTML document and from an applet viewer. An applet viewer will execute each APPLET tag that it finds in a separate window, while web browsers will allow many applets on a single page. So far, we have been using only a simplified form of the APPLET tag. Now it is time to take a closer look at it.

The syntax for a fuller form of the APPLET tag is shown here. Bracketed items are optional.

```
< APPLET  
[CODEBASE = codebaseURL]  
CODE = appletFile  
[ALT = alternateText]  
[NAME = appletInstanceName]  
WIDTH = pixels HEIGHT = pixels  
[ALIGN = alignment]  
[VSPACE = pixels] [HSPACE = pixels]  
>  
[< PARAM NAME = AttributeName VALUE = AttributeValue>]  
[< PARAM NAME = AttributeName2 VALUE = AttributeValue>]  
...  
[HTML Displayed in the absence of Java]  
</APPLET>
```

Let's take a look at each part now.

CODEBASE: CODEBASE is an optional attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class file (specified by the CODE tag). The HTML document's URL directory is used as the CODEBASE if this

attribute is not specified. The CODEBASE does not have to be on the host from which the HTML document was read.

CODE CODE is a required attribute that gives the name of the file containing your applet's compiled **.class** file. This file is relative to the code base URL of the applet, which is the directory that the HTML file was in or the directory indicated by CODEBASE if set.

ALT The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser recognizes the APPLET tag but can't currently run Java applets. This is distinct from the alternate HTML you provide for browsers that don't support applets.

NAME NAME is an optional attribute used to specify a name for the applet instance. Applets must be named in order for other applets on the same page to find them by name and communicate with them. To obtain an applet by name, use **getApplet()**, which is defined by the **AppletContext** interface.

WIDTH and HEIGHT WIDTH and HEIGHT are required attributes that give the size (in pixels) of the applet display area.

ALIGN ALIGN is an optional attribute that specifies the alignment of the applet. This attribute is treated the same as the HTML IMG tag with these possible values: LEFT, RIGHT, TOP, BOTTOM and MIDDLE.

VSPACE and HSPACE These attributes are optional. VSPACE specifies the space, in pixels, above and below the applet. HSPACE specifies the space, in pixels, on each side of the applet. They're treated the same as the IMG tag's VSPACE and HSPACE attributes.

PARAM NAME and VALUE The PARAM tag allows you to specify applet-specific arguments in an HTML page. Applets access their attributes with the **getParameter()** method. Other valid APPLET attributes include ARCHIVE, which lets you specify one or more archive files, and OBJECT, which specifies a saved version of the applet.

In general, an APPLET tag should include only a CODE or an OBJECT attribute, but not both.

Displaying Image in Applet

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The **java.awt.Graphics** class provide a method **drawImage()** to display the image.

Syntax of drawImage() method:

1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.

How to get the object of Image:

The **java.applet.Applet** class provides **getImage()** method that returns the object of Image.

Syntax:

1. `public Image getImage(URL u, String image){ }`

Other required methods of Applet class to display image:

1. **public URL getDocumentBase():** is used to return the URL of the document in which applet is embedded.
2. **public URL getCodeBase():** is used to return the base URL.

Example of displaying image in applet:

```
import java.awt.*;
import java.applet.*;
public class DisplayImage extends Applet
{
    Image picture;
    public void init()
    {
        picture = getImage(getDocumentBase(),"myganesha.jpg");
    }
    public void paint(Graphics g)
    {
        g.drawImage(picture, 30,30, this);
    }
}
```

In the above example, `drawImage()` method of **Graphics** class is used to display the image. The 4th argument of `drawImage()` method of is **ImageObserver** object. The **Component** class implements **ImageObserver** interface. So current class object would also be treated as **ImageObserver** because **Applet** class indirectly extends the **Component** class.

drawImage

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

Draws the specified image at the specified coordinate (x, y). If the image is incomplete the image observer will be notified later.

Parameters:

img - the specified image to be drawn

x - the x coordinate

y - the y coordinate

observer - notifies if the image is complete or not

myapplet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

Animation in Applet

Applet is mostly used in games and animation. For this purpose image is required to be moved.

Example:

```
import java.awt.*;
import java.applet.*;
public class AnimationExample extends Applet
{
    Image picture1;
    Image picture2;

    public void init()
    {
        picture1 =getImage(getDocumentBase(),"firstcar.jpg");
        picture2 =getImage(getDocumentBase(),"secondcar.jpg");
    }
    public void paint(Graphics g)
    {
        for(int i=0;i<50;i++)
        {
            if(i%2 ==0)
                g.drawImage(picture1, 30,30, this);
            else
                g.drawImage(picture2, 30,30, this);
            try
            {
                Thread.sleep(500);
            }
            catch(Exception e){}
        }
    }
}
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

MyApplet.HTML

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

Passing Parameters to Applets

As just discussed, the APPLET tag in HTML allows you to pass parameters to your applet. To retrieve a parameter, use the **getParameter()** method of Applet class. It returns the value of the specified parameter in the form of a **String** object. Here is an example that demonstrates passing parameters:

public String getParameter(String parameterName)

```
import java.applet.Applet;
import java.awt.Graphics;

public class UseParam extends Applet
{
    String str;
    public void init()
    {
        str=getParameter("msg");
        if(str == null)
            str = "GEHU";

        str = "Hello " + str;
    }
    public void paint(Graphics g)
    {
        g.drawString(str,50, 50);
    }
}
```

myapplet.html

```
<html>
<body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
</html>
```