**Applets**

* **An applet is a Java program that can be embedded (insert or place or set) into a web page.**
* **It runs inside the web browser and works at client side.**
* **An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server.**
* Applets are used to make the website more dynamic and entertaining.
* An applet is a Java class that extends the java.applet.Applet class.
* A main() method is not invoked on an applet, and an applet class will not define main().

**Advantage of Applets:**

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.
* Execution of applets is easy in a Web browser and does not require any installation or deployment procedure in real time programming (where as servlets require).
* Writing and displaying (just opening in a browser) graphics and animations is easier than applications.

**Limitations of Applets:**

* Applets cannot [read from or write](http://way2java.com/io/semantics-of-file-copying/) to hard disk files.
* Applet methods cannot be native.
* Applets should not attempt to create [socket connections](http://way2java.com/networking/networking-introduction/).
* Applets cannot read system properties.
* Applets cannot use any software available on the system (except browser execution area).
* Cannot [create objects](http://way2java.com/oops-concepts/usning-local-and-instance-variables/) of applications available on the system by composition.
* Plugin is required at client browser to execute applet.

**Differences between Applets and Application Programs:**

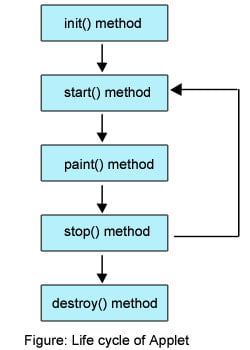
|  |  |
| --- | --- |
| **Applet programs** | **Application programs** |
| Small Program. | Large Program. |
| Used to run a program on client Browser. | Can be executed on standalone computer system. |
| Applet is portable and can be executed by any JAVA supported browser. | Need JDK, JRE, JVM installed on client machine. |
| Applet applications are executed in a Restricted Environment. | Application can access all the resources of the computer. |
| Applets are created by extending the java.applet.Applet | Applications are created by writing public static void main (String [] s) method. |
| Applet application has 5 methods which will be automatically invoked on occurrence of specific event. | Application has a single start point which is main method. |
| Example:  import java.awt.\*;  import java.applet.\*;  public class Myclass extends Applet  {  public void init() { }  public void start() { }  public void stop() {}  public void destroy() {}  public void paint(Graphics g) {}  } | public class MyClass  {  public static void main(String args[]) {}  } |

**Applet’s Life Cycle:**

* These methods are known as life cycle methods. These methods are defined in java.applet.Applet class except paint() method. The paint() method is defined in java.awt.Component class, an indirect super class of Applet.

**Description of Life Cycle methods:**

* Even though, the methods are called automatically by the browser, the programmer should know well when they are called and what he can do with the methods. Following is the schematic representation of the methods.

[](http://way2java.com/wp-content/uploads/2011/01/Life-cycle-of-Applet.jpg)

**Brief Description of Life Cycle Methods:**

* **init()** method is called at the time of starting the execution. This is called only once in the life cycle.
* **start()** method is called by the init() method. This method is called a number of times in the life cycle; whenever the applet is deiconifed , to make the applet active.
* **paint()** method is called by the start() method. This is called number of times in the execution.
* **stop()** method is called whenever the applet window is iconified to inactivate the applet. This method is called number of times in the execution.
* **destroy()** method is called when the applet is closed. This method is called only once in the life cycle.

**What is HTML?**

* **HTML stands for Hyper Text Markup Language.**
* **HTML is the standard markup language for creating Web pages.**
* **HTML describes the structure of a Web page.**
* HTML consists of a series of elements.
* HTML elements tell the browser how to display the content.
* HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

**The HTML APPLET Tag**

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. 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, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.

**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.

**Example 1 : (Applets can be run inside the applet window or Command prompt)**

// My Applet 1:

/\* <applet code="MyApplet1" width=300 height=100></applet> \*/

import java.awt.\*;

import java.applet.\*;

public class MyApplet1 extends Applet

{

    public void paint(Graphics g)

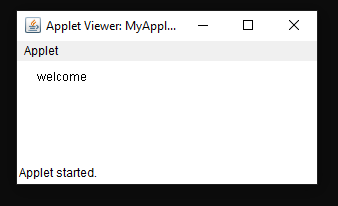
    {

        g.drawString("Hello World",100,100);

    }

}

**Output:**

****

**Example 2 : (Applets can be run inside the web browser window )**

**MyApplet1.java**

// My Applet 1:

import java.awt.\*;

import java.applet.\*;

/\* <applet code="MyApplet1" width=300 height=100></applet> \*/

public class MyApplet1 extends Applet

{

    public void paint(Graphics g)

    {

        g.drawString("Hello World",100,100);

    }

}

**MyApplet1.html**

<html>

    <head>

        <title>Hello World Applet</title>

    </head>

    <body>

        <h3>Applets can be run inside the web Page:</h3>

        <applet code="MyApplet1.class"

                width="400"

                height="150"

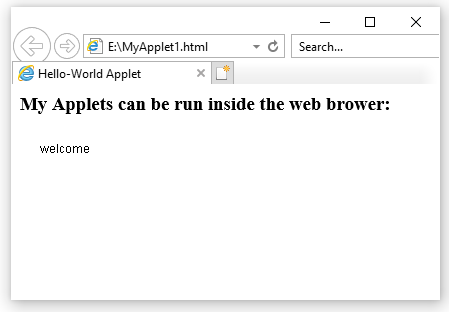
                alt="Error loading applet!">

        </applet>

    </body>

</html>

**Output:**

****

**Example 3:**

// My Applet 2:

/\* <applet code="MyApplet2" width=300 height=100></applet> \*/

import java.awt.\*;

import java.applet.\*;

public class MyApplet2 extends Applet

{

    public void init()

    {

        System.out.println("Applet Initialized");

    }

    public void start()

    {

        System.out.println("Applet Started");

    }

    public void stop()

    {

        System.out.println("Applet Stopped");

    }

    public void destroy()

    {

        System.out.println("Applet Destroyed");

    }

    public void paint(Graphics g)

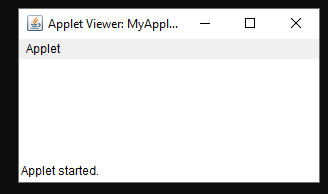
    {

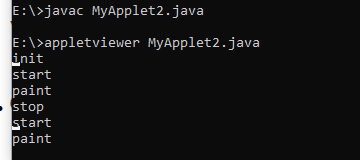
        System.out.println("Applet Painted");

    }

}

**Output:**

****

****

**Requesting Repainting:**

* **As a general rule, an applet writes to its window only when its update ( ) or paint ( ) method is called by the AWT. Whenever your applet needs to update the information displayed in its window, it simply calls repaint ( ).**
* The **repaint( )** method is defined by the AWT. It causes the AWT run-time system to execute a call to your applet’s **update( )** method, which, in its default implementation, calls **paint( )**. Thus, for another part of your applet to output to its window, simply store the output and then call **repaint( )**.
* Calling **repaint( )** is essentially a request that your applet be repainted sometime soon.

**Passing Parameters to Applets:**

* To pass the parameters to the Applet we need to use the **param** attribute of <applet> tag.
* To retrieve a parameter's value, we need to use the **getParameter()** method of Applet class.
* Thus, for numeric and **boolean** values, you will need to convert their string representations into their internal formats.
* Here is an example that demonstrates passing parameters:

// Passing Parameters to Applets:

import java.applet.\*;

import java.awt.\*;

import java.io.\*;

/\*

<applet code="ParamDemo" width=300  height=200 >

<param name="Name" value="Tayyab">

<param name="Designation" value="Data Engineer">

<param name="Company" value="Amazon">

<param name="Salary"  value=80000>

</applet>

\*/

public class ParamDemo extends Applet

{

    public void paint(Graphics g)

    {

        int d=0;

        g.drawString("Name : " + getParameter("Name"), 50, 50);

        g.drawString("Designation : " + getParameter("Designation"), 50, 70);

        g.drawString("Company : " + getParameter("Company"), 50, 90);

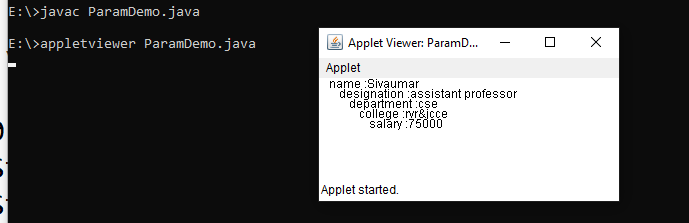
        d = Integer.parseInt(getParameter("Salary"));

        g.drawString("Salary : " + d, 50, 110);

    }

}

**Output:**

****

**Graphics class**

**The Graphics class defines a number of drawing functions.** Each shape can be drawn edge-only or filled. Objects are drawn and filled in the currently selected graphics color, which is black by default. When a graphics object is drawn that exceeds the dimensions of the window, output is automatically clipped.

Let’s take a look at several of the drawing methods.

**Drawing Lines:**

🡪Lines are drawn by means of the **drawLine( )** method, shown here:

void drawLine(int *startX*, int *startY*, int *endX*, int *endY*)

**drawLine( )** displays a line in the current drawing color that begins at *startX*,*startY* and ends at *endX*,*endY*.

The following applet draws several lines:

// Draw Lines:

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="Lines" width=300 height=300></applet>

\*/

public class Lines extends Applet

{

    public void paint(Graphics g)

    {

        g.setColor(Color.red);

        g.drawLine(0, 0, 100, 100);

        g.drawLine(0, 100, 100, 0);

        g.setColor(Color.green);

        g.drawLine(0, 100, 100, 200);

        g.drawLine(100, 100, 0, 200);

        g.setColor(Color.blue);

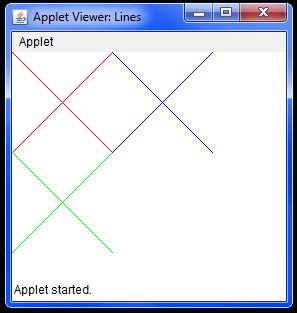
        g.drawLine(100, 0, 200, 100);

        g.drawLine(100, 100, 200, 0);

    }

}

Sample output from this program is shown here:



**Drawing Rectangles:**

🡪The **drawRect( )** and **fillRect( )** methods display an outlined and filled rectangle, respectively.They are shown here:

void drawRect(int *top*, int *left*, int *width*, int *height*)

void fillRect(int *top*, int *left*, int *width*, int *height*)

The upper-left corner of the rectangle is at *top*,*left.* The dimensions of the rectangle are specified by *width* and *height.*

🡪To draw a rounded rectangle, use **drawRoundRect( )** or **fillRoundRect( )**, both shown here:

void drawRoundRect(int *top*, int *left*, int *width*, int *height*,int *xDiam*, int *yDiam*)

void fillRoundRect(int *top*, int *left*, int *width*, int *height*,int *xDiam*, int *yDiam*)

A rounded rectangle has rounded corners. The upper-left corner of the rectangle is at *top*, *left.The* dimensions of the rectangle are specified by *width* and *height.* The diameter of the rounding arc along the X axis is specified by *xDiam.* The diameter of the rounding arc along the Y axis is specified by *yDiam.*

The following applet draws several rectangles:

// Draw Rectangles:

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="Rectangles" width=300 height=200></applet>

\*/

public class Rectangles extends Applet

{

    public void paint(Graphics g)

    {

        setBackground(Color.yellow);

        setForeground(Color.black);

        g.drawRect(10, 10, 60, 50);

        g.fillRect(100, 10, 60, 50);

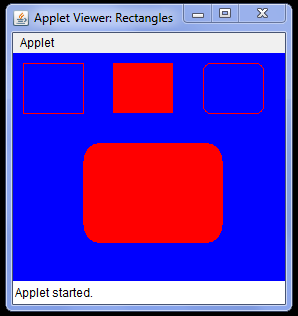
        g.drawRoundRect(190, 10, 60, 50, 15, 15);

        g.fillRoundRect(70, 90, 140, 100, 30, 40);

    }

}

Sample output from this program is shown here:



**Drawing Ellipses and Circles:**

🡪To draw an ellipse, use **drawOval( )**. To fill an ellipse, use **fillOval( )**. These methods are shown here:

void drawOval(int *top*, int *left*, int *width*, int *height*)

void fillOval(int *top*, int *left*, int *width*, int *height*)

The ellipse is drawn within a bounding rectangle whose upper-left corner is specified by *top*,*left* and whose width and height are specified by *width* and *height.* To draw a circle, specify a square as the bounding rectangle.

The following program draws several ellipses:

// Draw Ellipses

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="Ellipses" width=300 height=200></applet>

\*/

public class Ellipses extends Applet

{

    public void paint(Graphics g)

    {

        setBackground(Color.yellow);

        setForeground(Color.black);

        g.drawOval(10, 10, 50, 50);

        g.fillOval(100, 10, 75, 50);

        g.drawOval(190, 10, 90, 30);

        g.fillOval(70, 90, 140, 100);

    }

}

Sample output from this program is shown here:



**Drawing Arcs:**

🡪Arcs can be drawn with **drawArc( )** and **fillArc( )**, shown here:

void drawArc(int *top*, int *left*, int *width*, int *height*, int *startAngle*,int *sweepAngle*)

void fillArc(int *top*, int *left*, int *width*, int *height*, int *startAngle*,int *sweepAngle*)

The arc is bounded by the rectangle whose upper-left corner is specified by *top*,*left* and whose width and height are specified by *width* and *height.* The arc is drawn from *startAngle* through the angular distance specified by *sweepAngle.* Angles are specified in degrees. Zero degrees is on the horizontal, at the three o’clock position. The arc is drawn counterclockwise if *sweepAngle* is positive, and clockwise if *sweepAngle* is negative. Therefore, to draw an arc from twelve o’clock to six o’clock, the start angle would be 90 and the sweep angle 180.

The following applet draws several arcs:

// Draw Arcs:

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="Arcs" width=300 height=200></applet>

\*/

public class Arcs extends Applet

{

    public void paint(Graphics g)

    {

        setBackground(Color.yellow);

        setForeground(Color.black);

        g.drawArc(10, 40, 70, 70, 0, 75);

        g.fillArc(100, 40, 70, 70, 0, 75);

        g.drawArc(10, 100, 70, 80, 0, 175);

        g.fillArc(100, 100, 70, 90, 0, 270);

        g.drawArc(200, 80, 80, 80, 0, 180);

    }

}

Sample output from this program is shown here:



**Drawing Polygons:**

🡪It is possible to draw arbitrarily shaped figures using **drawPolygon( )** and **fillPolygon( )**,shown here:

void drawPolygon(int *x*[ ], int *y*[ ], int *numPoints*)

void fillPolygon(int *x*[ ], int *y*[ ], int *numPoints*)

The polygon’s endpoints are specified by the coordinate pairs contained within the *x* and *y* arrays. The number of points defined by *x* and *y* is specified by *numPoints.* There are alternativeforms of these methods in which the polygon is specified by a **Polygon** object.

The following applet draws an hourglass shape:

// Draw Polygons:

import java.awt.\*;

import java.applet.\*;

/\*<applet code="Polygons" width=300 height=300></applet>\*/

public class Polygons extends Applet

{

    public void paint(Graphics g)

    {

        int xpoints[] = {50, 200, 50, 200, 50};

        int ypoints[] = {50, 50, 200, 200, 50};

        int num = 5;

        setBackground(Color.yellow);

        setForeground(Color.black);

        g.drawString("Hour Glass: ", 50, 30);

        g.drawPolygon(xpoints, ypoints, num);

    }

}

Sample output from this program is shown here:



**Color class**

🡪**Color defines several constants to specify a number of common colors.**

* **You can also create your own colors, using one of the color constructors. Three commonly used forms are shown here:**

Color(int *red*, int *green*, int *blue*)

Color(int *rgbValue*)

Color(float *red*, float *green*, float *blue*)

* The first constructor takes three integers that specify the color as a mix of red, green, and blue. These values must be between 0 and 255, as in this example:

new Color(255, 100, 100); // light red

* The second color constructor takes a single integer that contains the mix of red, green, and blue packed into an integer. The integer is organized with red in bits 16 to 23, green in bits 8 to 15, and blue in bits 0 to 7. Here is an example of this constructor:

int newRed = (0xff000000 | (0xc0 << 16) | (0x00 << 8) | 0x00);

Color darkRed = new Color(newRed);

* The final constructor, **Color(float, float, float)**, takes three float values (between 0.0 and 1.0) that specify the relative mix of red, green, and blue.

🡪**Once you have created a color, you can use it to set the foreground and/or background color by using the setForeground( ) and setBackground( ) methods.**

* To set the background color, use **setBackground( )**.
* To set the foreground color, 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:

**Examples: The following example sets the background color to green and the text color to red:**

setBackground(Color.green);

setForeground(Color.red);

A good place to set the foreground and background colors is in the **init( )** method. Of course, you can change these colors as often as necessary during the execution of your applet.

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( )

Here is a very simple applet that sets the background color to cyan, the foreground color to red, and displays a message that illustrates the order in which the **init( )**, **start( )**, and **paint( )** methods are called when an applet starts up:

/\* 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=200></applet>

\*/

public class Sample extends Applet

{

    String msg;

    // set the foreground and background colors

    public void init()

    {

        setBackground(Color.yellow);

        setForeground(Color.black);

        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, 30, 100);

    }

}

You can also select it as the current drawing color.

🡪The **Color** class defines several methods that help manipulate colors. They are examined here. **Using Hue, Saturation, and Brightness**

* The *hue-saturation-brightness (HSB)* color model is an alternative to red-green-blue (RGB) for specifying particular colors. Figuratively, *hue* is a wheel of color. The hue is specified with a number between 0.0 and 1.0 (the colors are approximately red, orange, yellow, green, blue, indigo, and violet).
* *The Saturation* is another scale ranging from 0.0 to 1.0, representing light pastels to intense hues.
* *The Brightness* values also range from 0.0 to 1.0, where 1 is bright white and 0 is black.
* **Color** supplies two methods that let you convert between RGB and HSB. They are shown here:

static int HSBtoRGB(float *hue*, float *saturation*, float *brightness*)

static float[ ] RGBtoHSB(int *red*, int *green*, int *blue*, float *values*[ ])

* **HSBtoRGB( )** returns a packed RGB value compatible with the **Color(int)** constructor.
* **RGBtoHSB( )** returns a float array of HSB values corresponding to RGB integers. If *values* is not **null**, then this array is given the HSB values and returned. Otherwise, a new array iscreated and the HSB values are returned in it. In either case, the array contains the hue atindex 0, saturation at index 1, and brightness at index 2.

**🡪getRed( ), getGreen( ), getBlue( )**

You can obtain the red, green, and blue components of a color independently using **getRed( )**, **getGreen( )**, and **getBlue( )**, shown here:

int getRed( )

int getGreen( )

int getBlue( )

Each of these methods returns the RGB color component found in the invoking **Color** object in the lower 8 bits of an integer.

**🡪getRGB( )**

To obtain a packed, RGB representation of a color, use **getRGB( )**, shown here:

int getRGB( )

**Setting the Current Graphics Color**

By default, graphics objects are drawn in the current foreground color.

You can change this color by calling the **Graphics** method **setColor( )**:

void setColor(Color *newColor*)

Here, *newColor* specifies the new drawing color.

You can obtain the current color by calling **getColor( )**, shown here: getColor( )

**Example:**

The following applet constructs several colors and draws various objects using these colors:

// Demonstrate color:

import java.awt.\*;

import java.applet.\*;

/\* <applet code="ColorDemo" width=300 height=200></applet> \*/

public class ColorDemo extends Applet

{

    // draw lines

    public void paint(Graphics g)

    {

        Color c1 = new Color(255, 0, 0);

        Color c2 = new Color(0, 255, 0);

        Color c3 = new Color(0, 0, 255);

        setBackground(Color.white);

        g.setColor(c1);

        g.drawLine(0, 0, 100, 100);

        g.drawLine(0, 100, 100, 0);

        g.setColor(c2);

        g.drawLine(0, 100, 100, 200);

        g.drawLine(100, 100, 0, 200);

        g.setColor(c3);

        g.drawLine(100, 0, 200, 100);

        g.drawLine(100, 100, 200, 0);

        g.setColor(Color.red);

        g.drawOval(10, 10, 50, 50);

        g.fillOval(70, 90, 140, 100);

        g.setColor(Color.blue);

        g.drawOval(190, 10, 90, 30);

        g.drawRect(10, 10, 60, 50);

        g.setColor(Color.green);

        g.fillRect(100, 10, 60, 50);

        g.drawRoundRect(190, 10, 60, 50, 15, 15);

    }

}

Output:

