9. APPLETS AND APPLICATIONS

❖ The Applet class

➢ What is an Applet?

- An applet is a Java program that embedded with web content(html) and runs in a Web browser. It runs inside the browser and works at client side.
- We can fetch image, audio clip, play audio clip, Print a status message in the browser, and Get the network location in applet.
- You have to import java.applet ,java.awt packages

➢ Advantages of Applet:

- It works at client side so less response time.
- More Secured
- Applets can work on all the version of Java Plugin**.**
- Applets are supported by most web browsers in many platforms like Windows, Mac OS and Linux platform**.**
- User can also have full access to the machine if user allows.

➢ Disadvantages of Java Applet:

- Java plug-in is required to run applet**.**
- Java applet requires JVM so first time it takes significant start up time
- Its difficult to design and build good user interface in applets compared to HTML technology

❖ Applets and HTML (The applet Tag)

- ➢ To start with an applet first of all we need to learn applet tag.
- ➢ There are two ways to run an applet
  (1) By html file.
  (2) By appletviewer tool (for testing purpose).
- ➢ Applets have the file extension "class". An example would be " FirstApplet.class". Some applets consist of more than just one class file.
- ➢ Before embedding an applet on your page you need to upload the required files to your server.
- ➢ Below is a short example showing how simple it is to embed an applet on a page

Attribute	**Explanation**	**Example**
Code	Name of class file	Code=" FirstApplet.class"
Width	Width of applet	Width=200
Height	Height of applet	Height=100
Alt"	Text that will be shown in browsers where the ability to show applets has been turned off.	alt="Menu Applet"
Name	Assigning a name to an applet can be used when applets should communicate with each other.	Name="starter"
Align=Left/Right/Top /Middle/Bottom	Justifies the applet according to the text and images surrounding it. A full explanation of the individual parameters is given here.	Align=Right
Vspace	Space over and under the applet.	Vspace=20
Hspace	Space to the left and right of applet.	Hspace=40

- The procedure to create and run an applet goes as follows for an applet that simply displays ***Welcome to Java*** in its graphical window and also prints it to the Java console.

- **1) run By HTML file**
- **Step 1:** Use an editor to enter the following code for **FirstApplet** applet:

```
import java.applet.*;
import java.awt.*;
public class FirstApplet extends Applet
{
      public void paint(Graphics g)
       {
          g.drawString("Welcome to Java",150,150);
       }
}
```

  - Save this code into a file called **FirstApplet.java** . (Note that the file name must match exactly with the class name FirstApplet.)
- **Step 2:** Then compile the application with **javac FirstApplet.java**
  - This creates the class file FirstApplet.class
- **Step 3:** Next you must create a web page file to hold the applet.
  - Put the following code into a HTML file give name **FirstApplet.html**:

```
<HTML>
<Head>
<Title> A Simple Program </Title>
<Body>
<Applet Code="FirstApplet .class" width="150" height="50">
</Applet>
</Body>
</HTML>
```

  - Then put this file into the same directory as the **FirstApplet.class** file.
- **Step 4:** We have included the applet in this web page. So opening the file FirstApplet.html in your browser will display the same as the window here. You can also see the **"Welcome to Java"** line in the Java Console of the browser.

- **2) By appletViewer tool (for testing purpose).**
  - An alternative way to run the applet is with appletviewer: **appletviewer FirstApplet.html**

```
import java.applet.*;
import java.awt.*;
public class FirstApplet extends Applet
{
      /*
      <applet CODE="FirstApplet .class" width="300" height="300">
      </applet>
      */
      public void paint(Graphics g)
      {
            g.drawString("Welcome to Java", 150, 150);
      }
}
```

  - **Compile : D:\java\javac FirstApplet.java**
  - **Run : D:\java\appletviewer FirstApplet.java**

❖ Life cycles of an applets(Init(), start, stop, destroy method).

- The life cycle of an applet involves the following five stages:
  - Initialization
  - Starting
  - Stopping
  - Destroying
  - Painting

- The following methods implement the life cycle of an Applet:
  - **init():** This method is called to initialized an applet.
  - **start():** This method is called after the initialization of the applet.
  - **stop():** This method can be called multiple times in the life cycle of an Applet.
  - **destroy():** This method is called only once in the life cycle of the applet when applet is destroyed.

- **Initialization(init() method)**
  - Initialization occurs when the applet is first loaded (or reloaded).
  - It is called after the param tags inside the applet tag have been processed.
  - **Syntax :**

```
public void init ( )
{
    // Original  signature over ridden
}
```

- **Starting(start() method)**
  - After an applet is initialized, it is started.
  - Starting can happen many different times during an applet's lifetime, whereas initialization happens only once.
  - Starting can also occur if the applet was previously stopped.
  - For example, an applet is stopped if the reader follows a link to a different page, and is started again when the reader returns back to the page.
  - **Syntax:**

```
public void start ( )
{
    // Original signature over ridden
}
```

- **Stopping(stop() method)**
  - Stopping occurs when the reader leaves the page that contains a currently running applet, or by calling the stop ( ) method.
  - By default, when the reader leaves a page, any threads the applet had started will continue running.
  - **Syntax:**

```
public void stop ( )
{
    // Original  signature  over ridden
}
```

- **Destroying(destroy() method)**
  - Destroying enables the applet to perform a cleanup job just before it is destroyed or the browser exits for example, or release any other resources.
  - Generally the destroy( ) method is overridden if there are specific objects that need to be released, for example threads that the applets has created .
  - **Syntax:**

```
public void destroy ( )
{
    // Original signature over ridden
}
```

- **Painting(paint() method)**
  - Called immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.
  - Painting is the way an applet actually draws something on the screen, be it text, a line, a colored background, or an image.
  - Painting may occur a number of times during an applets life cycle.
  - The Paint( ) method gets executed every time the applet is minimized or maximized, or when the applet is initialized and the browser is placed behind another window on the screen, and then brought forward again.
  - **Syntax:**

```
public void paint (Graphics g)
{
    // Original signature over ridden
}
```

❖ Graphics class

- The **AWT**(**Abstract Window Toolkit**) graphics class is an abstract class that is the basis for all graphical constructions in Java.
- You cannot implement this class directly.
- You use the paint and update methods to implement the features in the Graphics class in your programs.
- You use the following code line to declare the paint method:
  - ***public void paint(Graphics g)***
- To include the different primitives of the Graphics class, be sure to import it into your program:

```
import java.awt.Graphics;
```

- The example applet shown these ideas. The output from this applet is Figure

```
import java.awt.*;
import java.applet.*;
public class demo extends Applet
{
    public void paint(Graphics g)
    {
        /*
        <applet CODE="demo.class" width="100" height="100">
        </applet>
        */
        g.drawRect(10, 10, 100, 100)
    }
}
```

drawString():

- The drawSring() method draws the given string as the parameter. Here is the syntax of the drawString() method :
- **Syntax:**

  drawString(String string, int X, int Y);
  - X is the starting X- axis coordinate
  - Y is the starting Y- axis coordinate
- **Example:**

```
import java.applet.*;
import java.awt.*;
public class HelloWorld extends Applet
{
        /*
        <applet CODE="HelloWorld.class" width="100" height="100">
        </applet>
        */
        public void paint(Graphics g)
        {
                g.drawString("Hello world!", 50, 25);
        }
 }
```

drawRect() and fillRect():

- **drawRect()** method used to create a rectangle.
- **fillRect()** method to draw a color-filled rectangle.
  - **Syntax:-**

    g.drawRect (int X,int Y, int width, ino height);
    g.fillRect (int X,int Y, int width, ino height);
  - X is the starting X- axis coordinate, counting from the left edge of the screen
  - Y is the starting Y- axis coordinate, counting from top edge of the screen
  - width is the width of the rectangle
  - height is the height of the rectangle
- The Graphics class provides additional methods that you can use for creating different types of rectangles.
- **Example:**

```
import java.awt.*;
import java.applet.*;
public class Rect1 extends Applet
{
        public void paint(Graphics g)
        {
                /*
                <applet CODE="Rect1.class" width="100" height="100">
                </applet>
                */
                g.drawRect(10,50,10,20);
                g.fillRect(50,80,100,200);
        }
}
```

- **clearRect()**
  - The **clearRect()** method can be used to clear a rectangular area.
  - **Syntax:**
    g.clearRect (int X, int Y, int Width, int Height)
  - Height**:** The vertical size of the rectangle in pixels.
  - Width**:** The horizontal size of the rectangle in pixels.
  - X**:** Horizontal location of the upper left corner of the rectangle, in pixels.
  - Y: Vertical location of the upper left corner of the rectangle, in pixels.
  - **Example:-**

```
import java.applet.Applet;
import java.awt.*;
public class Clear1 extends Applet
{
        public void paint (Graphics g)
        {
                /*
                <applet CODE="Clear1.class" width="100" height="100">
                </applet>
                */
                g.fillOval (20, 20, 180, 180);
                g.clearRect (50, 70, 120, 80);
        }
}
```

- **drawLine()**
  - You use the **drawLine()** method to draw a line.
  - **Syntax:-**
    g.drawLine (int startX, into startY, into endX, into endY);
    - startX is the starting X- axis coordinate
    - startY is the starting Y- axis coordinate
    - endX is the ending X- axis coordinate
    - endY is the ending Y- axis coordinate
  - **Example:-**

```
import java.awt.*;
import java.applet.*;
public class line1 extends Applet
{
        public void paint(Graphics g)
        {
                /*
                <applet CODE="line2.class" width="100" height="100">
                </applet>
                */
                g.drawLine(10,20,30,40) ;
        }
}
```

- **fillRoundRect() And drawRoundRect()**
  - drawRoundRect ( ) method used to create a Rounded rectangle and fillRoundRect() method used to create filled Rounded rectangle
  - Used to draw to rectangle with rounded corners.

- **Syntax:-**

  g. drawRoundRect(int X, int Y, int Width, int Height, int arcWidth, int arcHeight);
  g.fillRoundRect(int X, int Y, int Width, int Height, int arcWidth, int arcHeight);

- X is the starting X- axis coordinate, counting from the left edge of the screen
- Y is the starting Y- axis coordinate, counting from top edge of the screen
- width is the widths of the rectangle
- height is the height of the rectangle
- The curvature of the corners is specified by the two arguments, arcWidth and arcHeight.
- You can also use the fillRoundRect() method to draw a color-filled of rounded rectangle.
- **Example:-**

```
import java.awt.*;
import java.applet.*;
public class RoundRect1 extends Applet
{
        public void paint(Graphics g)
        {
                /*
                <applet CODE="RoundRect1.class" width="100" height="100">
                </applet>
                */
                g.drawRoundRect(10,50,100,150,68,120) ;
                g.fillRoundRect(50,50,100,150,68,120) ;
        }
}
```

➢ drawOval() and fillOval()

- The **drawOval()** and **fillOval()** methods to draw circles and ellipses. The difference between drawing circles and ellipses lies in the coordinates specified in the **drawOval()** method.
  - **Syntax:**

    g. drawOval (int X, int Y, int Width , int Height);
    g. fillOval (int X, int Y, int Width , int Height);
  - X and Y represent the top-left screen coordinates
  - width and height represent the width and the height of the circle or ellipse
- In the case of a circle, the width and height should be same.
- **Example:**

```
import java.awt.*;
import java.applet.*;
public class Oval1 extends Applet
{
        public void paint(Graphics g)
        {
                /*
                <applet CODE="Oval1.class" width="100" height="100">
                </applet>
                */
                g.drawOval(50,50,100,150);
                g.fillOval (100,150,100,150);
        }
}
```

- **drawArc():**
  - **Description:**
    - Used to draw an arc, which is similar to an incomplete oval.
    - The first parameters are similar to those for drawing an oval.
    - The last two parameters specify the starting and ending angles of the arc.
    - **Example:**

```
import java.awt.*;
import java.applet.*;
public class Arc1 extends Applet
{
    public void paint(Graphics g)
    {
        /*
        <applet CODE="Arc1.class" width="100" height="100">
        </applet>
        */
        setBackground(Color.pink);
        g.setColor(Color.blue);
        g.drawArc(150,150,100,150,190,200);
    }
}
```

- **drawPolygon():**
  - **drawPolygon()** and **fillpolygon()** methods of the graphics class. In the case of a polygon, the coordinates are specified as a part of two array variables.
  - **Syntax:**

    g.drawpolygon (int x[] , int y[] , int points) ;
  - In the give syntax
    - x [] is an array of x-axis coordinates
    - y [] is an array of y-axis coordinates
    - points is the number of points defined by the (x, y) coordinates
  - **Example:**

```
import java.awt.*;
import java.applet.*;
public class Polygon1 extends Applet
{
    public void paint(Graphics g)
    {
        /*
        <applet CODE="Polygon1.class" width="100" height="100">
        </applet>
        */
        int x[]={20,100,150};
        int y[]={230,220,280};
        int n=3;
        g.drawPolygon(x,y,n);
        g.fillPolygon(x,y,n);
    }
}
```

❖ Painting the applet

➢ Update():

- This method is defined by the AWT and is called when your applet has requested that a portion of its window be redrawn.
- The problem is that the default version of update() first fills an applet with the default background color and then calls paint(), this gives rise to a flash of default color (usually gray) each time update is called.
- To avoid this you define your update() method such that it performs all necessary display activities
- The paint() in this case will simply call update().

```
public void update(Graphic g)
{
    //Redisplay your window here.
}

public void paint(Graphics g)
{
    update(g) // call to the update()method.
}
```

- You need to override update() only when needed.

➢ Paint():

- The paint() method is called each time your applet's output must be redrawn.
- For example, the window in which the applet is running may be overwritten by another window and then uncovered.
- Or the applet window was resized.
- paint() is also called when the applet begins execution.
- Whatever the cause, whenever the applet must redraw its output, paint() is called.
- The paint() method has one parameter of type Graphics.
- This is needed for the applet to know where the applet should paint its output.

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

- **Example:-**

```
import java.awt.*;
import java.applet.*;
public class Paint1 extends Applet
{
        public void paint(Graphics g)
        {
                /*
                <applet CODE="Paint1.class" width="100" height="100">
                </applet>
                */
                g.setColor(Color.blue);
                g.drawString("KAP",100,50);
        }
}
```

- **repaint():**
  - The repaint() method is defined by the AWT. It causes the AWT run time system to call to your applet's update() method, which in its default implementation, calls paint().
  - Again for example if a part of your applet needs to output a string, it can store this string in a variable and then call repaint()
  - Inside paint(), you can output the string using drawstring().
  - **The repaint method has two forms.**
    - **void repaint()**
    - **void repaint(int left, int top, int width, int height)**
  - **Let's look at each one –**
    - void repaint()
      - This causes the entire window to be repainted
    - void repaint(int left, int top, int width, int height)
      - This specifies a region that will be repainted. integers left, top, width and height are in pixels. You save time by specifying a region to repaint instead of the whole window.

❖ Passing parameters to applets getparameter() method

- Parameters are passed to applets in NAME=VALUE pairs in <PARAM> tags between the opening and closing APPLET tags.
- Inside the applet, you read the values passed through the PARAM tags with the getParameter() method of the java.applet class.
- The applet parameter "Message" is the string to be drawn.
- **Example:-**

```
import java.applet.*;
import java.awt.*;
public class Getparameter extends Applet
{

        private String a = "Hello!";
        public void paint(Graphics g)
        {
                /*
                <applet CODE="Getparameter.class" width="100" height="100">
                <PARAM name="Message" value="How Are You!">
                </applet>
                */
                String b = this.getParameter("Message");
                if (b == null)
                {
                        b = a;
                }
                g.drawString(b, 50, 25);
        }
}
```

- You also need an HTML file that references your applet. The following simple HTML file will do:

```
<html>
<body>
<applet CODE="Getparameter.class" width="100" height="100">
```

<PARAM name="Message" value="How Are You!">
</applet>
</body
</html>

- Of course you are free to change "How are you" to a "message" of your choice.
- You only need to change the HTML, not the Java source code. PARAMs let you customize applets without changing or recompiling the code.
- However rather than hard coding the message to be printed it's read into the variable **b** from a PARAM element in the HTML.
- You pass getParameter() a string that names the parameter you want.
- This string should match the name of a PARAM element in the HTML page.
- getParameter() returns the value of the parameter. All values are passed as strings. If you want to get another type like an integer, then you'll need to pass it as a string and convert it to the type you really want.
- The PARAM element is also straightforward. It occurs between <APPLET> and </APPLET>.
- It has two attributes of its own, NAME and VALUE.
- NAME identifies which PARAM this is.
- VALUE is the string value of the PARAM. Both should be enclosed in double quote marks if they contain white space.
- An applet is not limited to one PARAM. You can pass as many named PARAMs to an applet as you like.