Linked List

- A linked list is a sequence of data structures, which are connected together via links.
- Linked List is a sequence of links which contains items.
- Each link contains a connection to another link.
- Linked List is a second most-used data structure after array.
- Following are the important terms to understand the concept of linked list.
    - Info – Each info of a linked list can store a data called an element.
    - Link – Each link of a linked list contains a link to the next link called next.

Types of Linked List

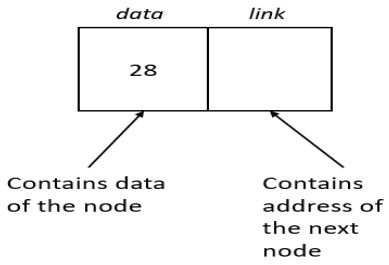
- Following are the various types of linked list.
    - Singly Linked List – Item navigation is forward only.
    - Doubly Linked List – Items can be navigated forward and backward.
    - Singly Circular Linked List – Last item contains link of the first element as next.

Basic Operations

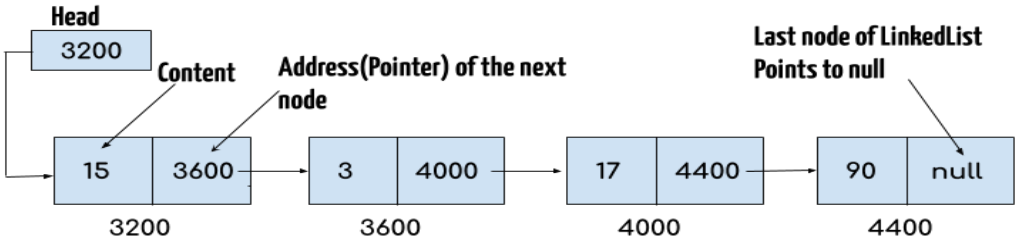
- Following are the basic operations supported by a list.
    - Insertion at Beginning – Adds an element at the beginning of the list.
    - Insertion at Last – Adds and element at the end of the list.
    - Deletion at Beginning – Deletes and element at the beginning of the list.
    - Deletion at Last – Deletes and element at the end of the list.
    - Display – Displays the complete list.

Singly Linked List Representation

- Singly Linked list can be visualized as a chain of nodes, where every node points to the next node.



-



- As the above figure, following are important points to be considered.
  - Linked List contains a node which contains data and link part.
  - Each node carries a data field(s)(info) and a link field called next.
  - HEAD is the element which contains the link of first element of the linked list.
  - Each link is linked with its next element using its next link.
  - Last node of SLL carries a link as null to mark the end of the list.

Creating a Node for Singly Linked List

Here, NewNode=node to be linked, having data and address of the next node.

HEAD=Pointer pointing to the first node.

CurrPtr=Pointer pointing to current processing node.

Step 1: Allocate Memory to NewNode (Using new)

Step 2: Input data for the newnode

NewNode->data=X

NewNode->next=NULL

Inserting node at the beginning

- Step 1: Create NewNode
- Step 2: if HEAD==NULL then

```
        HEAD=NewNode
    Else
        NewNode->Next=HEAD
        HEAD=NewNode
    End if
```

Inserting node at the end

- Step 1: Create NewNode
- Step 2: if HEAD==NULL then

```
        HEAD=NewNode
    Else
        CurrPtr=HEAD;
        While(CurrPtr->Next!=NULL)
            CurrPtr=CurrPtr->Next
        End While
        CurrPtr->Next=NewNode
    End if
```

Inserting node at a specific position, POS

- Step 1: Create NewNode
- Step 2: if HEAD==NULL then

```
        HEAD=NewNode
    Else
        CurrPtr=HEAD;
        for(int i=1;i<p-1 && currentptr!=null;i++)
```

```
        {
            CurrPtr=CurrPtr->Next
        }

        if(CurrPtr==NULL)then
            print "position is not available"
        else
            NewNode->Next=CurrPtr->Next;
            CurrPtr->Next=NewNode;
        End if
    End if
```

Deleting node at the beginning

- Step 1: CurrPtr =HEAD
- Step 2: if CurrPtr ==NULL then

```
            Print "LinkedList is empty"
    Else
            HEAD=CurrPtr->Next;
            delete(CurrPtr)
    End if
```

Deleting node at the end

- Step 1: CurrPtr =HEAD,PrevPtr=NULL;
- Step 2: if CurrPtr ==NULL then

```
            Print "LinkedList is empty"
    Else
            While(CurrPtr->Next!=NULL)
```

```
                PrevPtr=CurrPtr;
                CurrPtr=CurrPtr->Next
            End while
            PrevPtr->Next=NULL
            delete(CurrPtr);
        End if
```

Deleting node at a specific position

- Step 1: CurrPtr =HEAD,PrevPtr=NULL, POS
- Step 2: if CurrPtr ==NULL then

```
            Print “LinkedList is empty”
        Else
            CurrPtr=HEAD;
            While(CurrPtr!=NULL && count<position)
                PrevPtr=CurrPtr
                CurrPtr=CurrPtr->Next
                count++;
            End while
            if(CurrPtr==NULL)then
                print “position is not available”
            else
                PrevPtr->Next=CurrPtr->Next
                delete (CurrPtr)
            End if
        End if
```

Traverse and Display Linked List

- To display the linked list, you have to traverse a link list, node by node from the first node to the last node. Traverse involves following steps:
- Step 1: CurrPtr =HEAD
- Step 2: Repeat step 3 and 4 until CurrPtr !=NULL
- Step 3: Display CurrPtr ->Data
- Step 4: CurrPtr=CurrPtr->Next

Write a menu driven program for Singly Linked List in JAVA.

```
import java.util.Scanner;

import org.omg.PortableServer.Current;

class linkedlist
{
    private int data;
    private linkedlist next;
    static linkedlist head,currentptr,prevptr;
    linkedlist()
    {
        data=0;
        next=null;
    }
    linkedlist getnode()
    {
        linkedlist temp=new linkedlist();
        if(temp==null)
        {
            return null;
        }
        else
        {
            return temp;
        }
    }
    public void insertatfirst(int ele)
    {
        linkedlist newnode=getnode();
        newnode.data=ele;
        newnode.next=null;
        if(head==null)
```

```
        {
            head=newnode;
        }
        else
        {
            newnode.next=head;
            head=newnode;
        }
    }
    public void insertatlast(int ele)
    {
        linkedlist newnode=getnode();
        newnode.data=ele;
        newnode.next=null;
        if(head==null)
        {
            head=newnode;
        }
        else{
            currentptr=head;
            while(currentptr.next!=null)
            {
                currentptr=currentptr.next;
            }
            currentptr.next=newnode;
        }
    }

   public void insertatposition(int ele,int p)
    {
        linkedlist newnode=getnode();
        newnode.data=ele;
        newnode.next=null;

        currentptr=head;
        for(int i=1;i<p-1 && currentptr!=null;i++)
        {
            currentptr=currentptr.next;
        }
        if(currentptr==null)
        {
            System.out.println("Your Entered Position is wrong.");
        }
        else
        {
            newnode.next=currentptr.next;
            currentptr.next=newnode;
        }
```

```
    }

    public void deleteatfirst()
    {
        if(head==null)
        {
            System.out.println("Singly Linked List is Empty.");
        }
        else
        {
            currentptr=head;
            head=currentptr.next;
            System.out.println("Deleted Element is
"+currentptr.data);
            currentptr=null;
            //memory automatically removed as it is java
        }
    }
    public void deleteatlast()
    {
        if(head==null)
        {
            System.out.println("Singly Linked List is Empty.");
        }
        else
        {
            currentptr=head;
            while(currentptr.next!=null)
            {
                prevptr=currentptr;
                currentptr=currentptr.next;
            }
            prevptr.next=null;
            System.out.println("Deleted Element is
"+currentptr.data);
            currentptr=null;
            //memory automatically removed as it is java
        }
    }
     public void deletefromposition(int p)
        {
            if(head==null)
            {
                System.out.println("Singly Linked List is
Empty.");
            }
            else
            {
```

```
                currentptr=head;
                for(int i=0;i<p-1 && currentptr!=null;i++)
                {
                    prevptr=currentptr;
                    currentptr=currentptr.next;

                }
                prevptr.next=currentptr.next;
                System.out.println("Deleted Element is
"+currentptr.data);
                currentptr=null;
                //memory automatically removed as it is java
            }
        }

    void display(){
        System.out.print("The linkedlist Elements are:");
        currentptr=head;
        while(currentptr!=null)
        {
            System.out.print(currentptr.data+", ");
            currentptr=currentptr.next;
        }
    }
}

public class linkedlistdemo
{
    public static void main(String[] args)
    {
        int ch,ele,p;
        linkedlist l=new linkedlist();
        Scanner sc=new Scanner(System.in);
        do{
            System.out.println("\n1.Insert at First");
            System.out.println("2. Insert at end");
            System.out.println("3. Insert at given position");
            System.out.println("4. Delete at first");
            System.out.println("5. Delete at last");
            System.out.println("6. Delete at given position");
            System.out.println("7. Display");
            System.out.println("0. Display");
            System.out.println("Enter your choice:");
            ch=sc.nextInt();
            switch(ch)
            {
                case 1:
                        System.out.println("Enter the element: ");
```

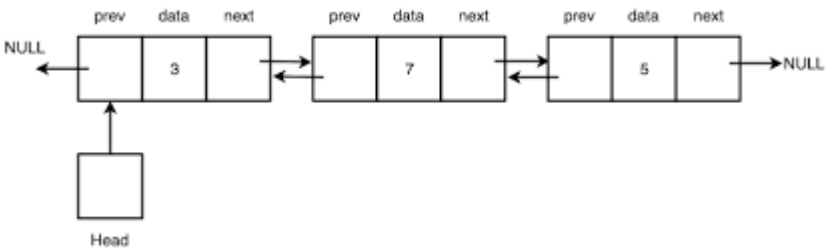
```
                        ele=sc.nextInt();
                        l.insertatfirst(ele);
                        break;
                case 2:
                        System.out.println("Enter the element: ");
                        ele=sc.nextInt();
                        l.insertatlast(ele);
                        break;
                case 3:
                        System.out.println("Enter the element: ");
                        ele=sc.nextInt();
                        System.out.println("Enter the position at
which you want to insert element:");
                        p=sc.nextInt();
                        l.insertatposition(ele, p);
                        break;
                case 4:
                        l.deleteatfirst();
                        break;
                case 5:
                        l.deleteatlast();
                        break;
                case 6:
                    System.out.println("Enter the position at
which you want to delete element:");
                    p=sc.nextInt();
                    l.deletefromposition(p);
                case 7:
                        l.display();
                        break;
                case 0:
                        System.out.println("Program End...");
                        break;
                default:
                    System.out.println("Enter Proper Choice");

            }
        }while(ch!=0);
    }
}
```

Doubly Linked List

- A doubly linked list is a two-way list in which all nodes will have two links.
- This helps in accessing both successor node and predecessor node from the given node position.
- It Provides bi-directional traversing. Each node contains three fields:
    - Left Link (Previous)
    - Data (element)
    - Right Link (Next)

- Node
- The left link points to the previous node and the right link points to the next node.
- The data field stores the required data.
- The Basic operations in doubly linked list are:
    - Creation
    - Insertion
    - Deletion
    - Traversing
- A doubly linked list is show in following figure.

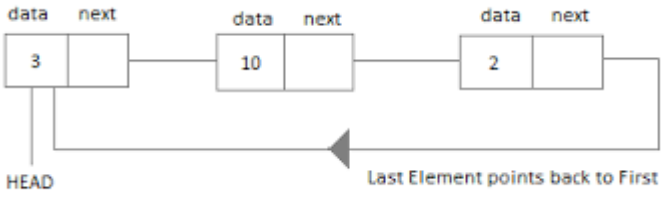


- As the above figure, following are important points to be considered.

- Doubly Linked List contains a node which contains data and two link part.
- Prev contains the pointer to previous node, data contains the information and next contain the pointer to next node.
- Head is the element which contains the link of first element of the doubly linked list.
- In doubly linked list, first node contains NULL in Prev pointer and Last node contains NULL in the Next pointer.

Circular Linked List

- It is just a single linked list in which the link field of last node points back to the address of the first node.
- A circular linked list has no beginning and no end.
- HEAD is pointing to the first node of the circular linked list.
- Circular linked list are frequently used instead of ordinary linked list as many operations are much easier to implement.
- In circular linked list no null pointer are used, hence all pointers contain valid address.



-

- The basic operations in a circular single linked list are:
  - Creation
  - Insertion
  - Deletion
  - Traversing

Algorithm

Creating a Node for Circular Linked List

Here, Node=node to be linked, having data and address of the next node.

Head=Pointer pointing to the first node.

CurrPtr=Pointer pointing to current processing node.

Step 1: Allocate Memory to Node (Using new)

Step 2: Input data for the node

node->data=X

node->next=NULL

Inserting node at the beginning

- Step 1: Create NewNode
- Step 2: if HEAD==NULL then

```
        HEAD=NewNode
        HEAD->Next=HEAD
    Else
        CurrPtr=HEAD;
        While(CurrPtr->Next!=HEAD)
            CurrPtr=CurrPtr->Next
        End while
        NewNode->Next=HEAD
        HEAD=NewNode
        CurrPtr->Next=HEAD
    End if
```

Inserting node at the end

- Step 1: Create NewNode
- Step 2: if HEAD==NULL then

```
        HEAD=NewNode
    Else
        CurrPtr=HEAD;
        While(CurrPtr->Next!=HEAD)
```

```
                    CurrPtr=CurrPtr->Next
                End while
                CurrPtr->Next=NewNode
                NewNode->Next=HEAD
        End if
```

Inserting node at a specific position,POS

- Step 1: Create NewNode
- Step 2: if HEAD==NULL then

```
                HEAD=NewNode
        Else
                CurrPtr=HEAD;
                count=1;
                While(CurrPtr->Next!=HEAD && count<position)
                        PrevPtr=CurrPtr
                        CurrPtr=CurrPtr->Next
                        count++;
                End While
                if(CurrPtr==HEAD)then
                        print “position is not available”
                else
                        NewNode->Next=CurrPtr
                        PrevPtr->Next=NewNode
                End if
        End if
```

Deleting node at the beginning

- Step 1: CurrPtr=HEAD,PrevPtr=NULL;
- Step 2: if CurrPtr==NULL then

```
                Print “LinkedList is empty”
        Else
                HEAD=CurrPtr->Next
                DO
                        PrevPtr=CurrPtr;
                        CurrPtr=CurrPtr->Next
```

```
            While(CurrPtr->Next!=HEAD);
            PrevPtr->Next=HEAD
            If(CurrPtr->next==CurrPtr)
                HEAD=NULL
            End if
            delete(CurrPtr)
        End if
```

Deleting node at the end

- Step 1: CurrPtr =HEAD,PrevPtr=NULL;
- Step 2: if CurrPtr ==NULL then

```
            Print "LinkedList is empty"
        Else
            While(CurrPtr->Next!=HEAD)
                PrevPtr=CurrPtr;
                CurrPtr=CurrPtr->Next
            End While
            PrevPtr->Next=HEAD;
            If(CurrPtr->next==CurrPtr)
                HEAD=NULL
            End if
            delete(CurrPtr);
        End if
```

Deleting node at a specific position

- Step 1: CurrPtr =HEAD,PrevPtr=NULL, POS
- Step 2: if CurrPtr ==NULL then

```
            Print "LinkedList is empty"
        Else
            CurrPtr=HEAD;
            While(CurrPtr->Next!=HEAD && count<position)
                PrevPtr=CurrPtr
                CurrPtr=CurrPtr->Next
                count++;
            End While
```

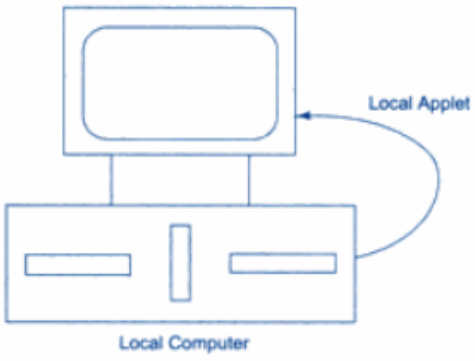
```
            if(CurrPtr==HEAD)then
                print "position is not available"
            else
                PrevPtr->Next=CurrPtr->Next
                delete(CurrPtr)
            end if
        End if
```

Traverse and Display Linked List

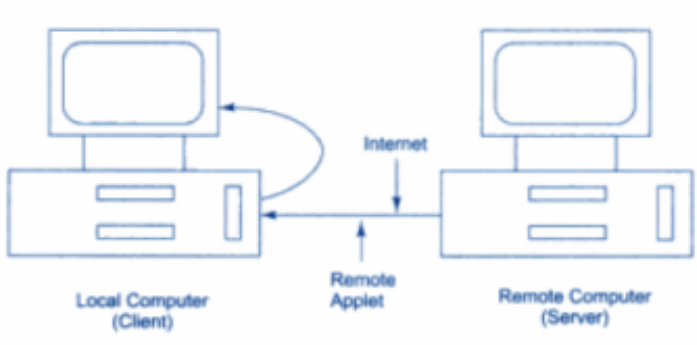
- To display the linked list, you have to traverse a link list, node by node from the first node to the last node. Traverse involves following steps:
- Step 1: CurrPtr=HEAD
- Step 2: Repeat step 3 and 4 until CurrPtr !=HEAD
- Step 3: Display CurrPtr->Data
- Step 4: CurrPtr=CurrPtr->Next

What is Applet? Explain types of Applets.

- Applets are small java programs that are embedded into a web page. It runs inside the web browser and works at client side.
- Applet can perform arithmetic operations, display graphics, play sounds, accept user input, create animation and play interactive games.
- We can embed applet into web pages in two ways.
    - Local applets: 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 to use the 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.



    - Remote applets: A remote applet is that which is developed by someone else and stored on a remote computer connected to the internet. If our system is connected to the internet, we can download the remote applet onto our system via at the internet and run it.

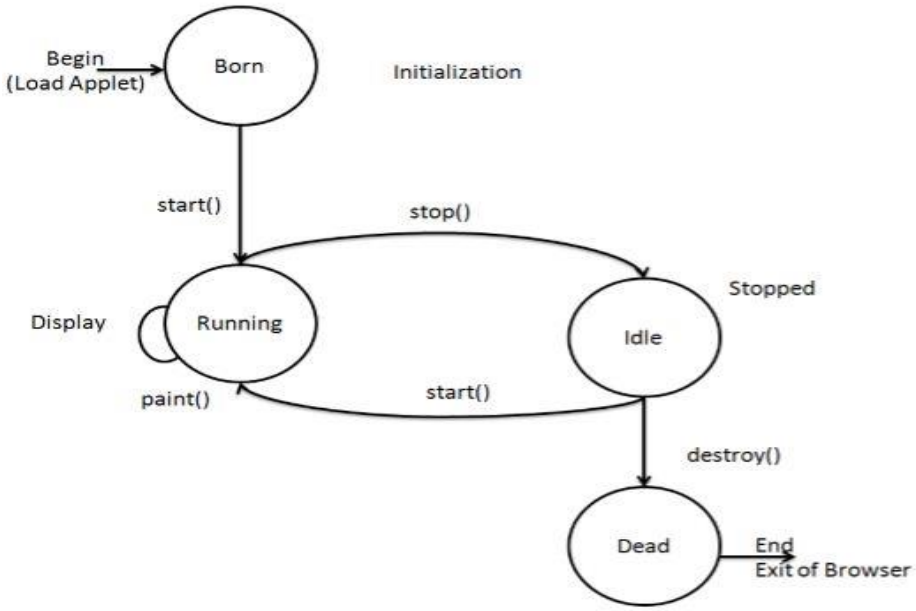


Differentiate application and applet

BASIS FOR COMPARISON	APPLET	APPLICATION
Basic	It is small program uses another application program for its execution.	An application is the programs executed on the computer independently.
main() method	Do not use the main method	Uses the main method for execution
Execution	Cannot run independently require API's (Ex. Web API).	Can run alone but require JRE.
Installation	Prior installation is not needed	Requires prior explicit installation on the local computer.
Read and write operation	The files cannot be read and write on the local computer through applet.	Applications are capable of performing those operations to the files on the local computer.
Communication with other servers	Cannot communicate with other servers.	Communication with other servers is probably possible.
Restrictions	Applets cannot access files residing on the local computer.	Can access any data or file available on the system.
Security	Requires security for the system as they are untrusted.	No security concerns are there.

Explain life cycle of the applet. OR Applet skeleton

- The applet life cycle include following states :
    - Born or initialization state
    - Running state
    - Idle state
    - Dead or destroy state



Initialization state:

- Applet enters the initialization state when it is first loaded. This is achieved by calling the init() method of Applet class.
- At this stage we may do following things :
  - Create objects needed by the applet
  - Set up initial values
  - Load images or fonts
  - Set up colors
- The initialization occurs only once in the applet's life cycle.

```
public void init()
{
      //Action to be performed
}
```

Running State:

- Applet enters the running state when the system calls the start() method of Applet class. This occurs automatically after the applet is initialized. Starting can also occur if the applet is already in "stopped" (idle) state.

- For example, we may leave the web page containing the applet temporarily to another page and return back to the page.

```
public void start()
{
    //Action to be performed
}
```

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 the stop() method explicitly.

```
public void stop()
{
       //Action to be performed
}
```

Dead State

- An applet is in dead state when it has been removed from the memory. This can be done by using destroy() method. It's general form is:

```
public void destroyed()
{
      //Action to be performed
}
```

Display state

- Applet moves to the display state whenever it has to perform some output operations on the screen. This happens immediately after the applet enters into the running state.
- The paint method is called to accomplish this task.

```
public void paint(Graphics g)
{
     //Action to be performed
}
```

What is applet? How to pass parameters to the applet? Explain with example.

- We can supply user-defined parameters to an applet using <PARAM> tag. Each <PARAM> tag has a name attribute such as color and a value attribute such as red.
- Inside the applet code, the applet can refer to that parameter by name to find its value.
- For example, we can change the color of the text displayed to red by an applet by using a <PARAM> tag as follows:

```
<APPLET>
<PARAM = color VALUE = "red"/>
</APPLET>
```

- Passing parameter to an applet code using <PARAM> tag is something similar to passing parameters to the main() method using command line arguments.
- To set up and handle parameters, we need to do two things :
    1. Include appropriate <PARAM> tags in HTML document.
    2. Provide code in the applet to parse these parameters.
- Parameters are passed on an applet when it is loaded. We can define the init() method in the applet tag to get hold of the parameters defined in the <PARAM> tags.

- This is done using the getParameter() method, which takes one string argument representing the name of the parameter and returns a string containing the value of that parameter.
- Example :

```
<APPLET CODE = "ParamApplet.class" WIDTH = 300 HEIGHT = 100>
<PARAMNAME = "str" VALUE = "SYBCA">
</APPLET>
import java.applet*;
import java.awt.*;

public class ParamApplet extends Applet
{
          public void paint (Graphics g)
          {
              String str=getParameter ("str");
              g.drawString (str, 50, 50);
          }
}
```

HTML Applet Tag

- The APPLET tag of HTML is used to start an applet either from a web browser or an applet viewer. The HTML tag allows a Java applet to be embedded in an HTML document.
- The complete syntax of the applet tag is given below:

```
<applet
code = "appletFile" -or- [object = "serializedApplet"]
width = "pixels"
height = "pixels" [codebase = "codebaseURL"]
[archive = "archiveList"]
[alt = "alternateText"]
[name = "appletlnstanceName"]
[align = "alignment"]
[vspace = "pixels"]
[hspace = "pixels"]
>
[<param name = "appletAttribute1" value = "value">]
[<param name = "appletAttribute2" value = "value">]
….
….
[alternateHTML]
</applet>
```

Attributes in the applet tag.

- The attributes code, width and height are mandatory and all other attributes are options, which are shown in brackets ([]). The attributes need not follow the order that was specified in the above syntax. The description of each attribute is given below:

- **code**The name of the Java applet. Class file, which contains the compiled applet code. This name must be relative to the base URL of the Applet and cannot be an absolute URL. The extension in the file name is optional.

- **width** This refers to the width of the applet panel specified in pixels in the browser window.

- **height** This refers to the height of the applet panel specified in pixels in the browser window. The width and height attributes specify the applet's, display area. This applet area does not include any windows or dialogue boxes that the applet shows.

- **Codebase** This refers to the base URL of the applet. That is, the URL of the directory that contains the applet class file. If this attribute is not specified then the HTML document's URL directory is taken as the CODEBASE.

- **archive** There are one or more archive files containing Java classes and other resources that will be preloaded. The classes are loaded using an instance of the AppletClassLoader class with the given codebase. The archives in archivelist are separated by a comma (,).

- **alt** This refers to text to be displayed if the browser understands the applet tag but cannot run Java applets.

- **name** This refers to a name for the applet instance which makes it possible for applets to communicate with each other on the same HTML page. The getApplet () method, which is defined in the AppletContext interface, can be used to get the applet instance using name.

- **align** This refers to the alignment of the apple!. The possible attribute values are LEFT, RlGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE and ABSBOTTOM.

- **vspace** This refers to the space, specified in number of pixels, used as a margin above and below the applet.

- **hspace** This refers to the space, specified in number of pixels, used as a margin to the left and right of the applet.

- **param** The param tag, specifies an applet parameter as a name-value pair. The name is the parameters name, value is. its value. The values of these parameters can be obtained using the getParameter () method.

- **alternateHTML** This is ordinary HTML to be displayed in the absence of Java or if the browser does not understand the applet tag. This is different from the alt attribute which understands the· applet tag but cannot run, whereas the alternateHTML tag is provided when a browser does not support applets.

What is applet? List out various methods of graphics class used with applet.

drawString(): it is used to draw the specified string.

- Syntax:

```
void drawString(String message, int x, int y)
```

- Here, message is the string to be output beginning at x,y. In a java window, the upper-left corner is location 0,0.

drawLine(): it is used to draw the line between two given points.

- Synrax:

```
void drawString(int startX,startY,endX,endY)
```

- This method draw line that begins at startX, startY and ends at endX,endY.

drawRect(): it draw a rectangle with specified width and height.

- Syntax:

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

- Here the upper-left corner of the rectangle is at top,left. The dimensions of the rectangle are specified by width and height.
- If width and height are equal then it become square.

fillRect(): it is used to draw rectangle with specified color.

- Syntax:

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

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

drawRoundRect(): it is used to draw rounded rectangle.

- Syntax:

```
void drawRoundRect(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 eidth and height,
- The diameter of the rounding arc along the X axis is specified by xDim. The diameter of the rounding arc along the Y axis is specified by yDiam.

fillRoundRect(): it is used to draw rounded rectangle and that is filled with specified color.

- Syntax:

```
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 eidth and height,
- The diameter of the rounding arc along the X axis is specified by xDim. The diameter of the rounding arc along the Y axis is specified by yDiam.

drawOval(): it is used to draw oval/ellipse with specified width and height.

- Syntax:

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

- The ellipse is drawn within a bounding rectangle whose upper-left corner is specified bt top,left ans whose width and height are specified by width and height.
- If width and height are equal then it become circle.

fillOval(): it is used to draw oval/ellipse with specified width and height and fill with specified color.

- Syntax:

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

- The ellipse is drawn within a bounding rectangle whose upper-left corner is specified bt top,left ans whose width and height are specified by width and height.

drawArc(): it is used to draw arc.

- Syntax:

```
   void drawArc(int top,int left, int width, int height,
intstartAngle, int sweepAngle)
```

- The arc is bounded by the rectangle whoe 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 seepAngle.
- Angles are specified in degrees. Zero degree 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.

```
<applet code="GraphicsDemo.class" width="300" height="300">
</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);
      }
}
```