What are Data Structures in Java?

A data structure is a specialized format for organizing, processing, retrieving and storing data. There are several basic and advanced types of data structures, all designed to arrange data to suit a specific purpose.

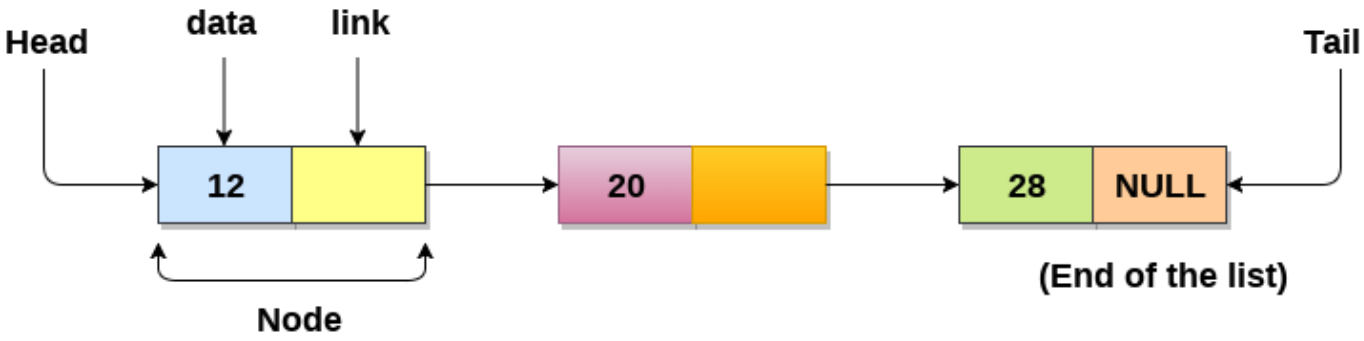
Types of Data Structures in Java

1. Arrays
2. ArrayList
3. LinkedList
4. Stack
5. Queue
6. HashMap
7. HashSet
8. TreeSet
9. TreeMap
10. Graph
11. Tree

Linked List

Linked List is a linear data structure, in which elements are not stored at a contiguous location, Each node in a linked list can be allocated independently, allowing for dynamic memory allocation and efficient insertion and deletion operations.

Linked List forms a series of connected nodes, where each node stores the data and the address of the next node.



The linked list is accessed through the head node, which points to the first node in the list.

The last node in the list points to null, indicating the end of the list. This node is known as the tail node.

Advantages:

1. **Dynamic Size:** LinkedList can grow or shrink dynamically, making it suitable for varying or unknown data sizes.
2. **Efficient Insertion and Deletion:** Inserting or deleting elements within a LinkedList is efficient, as it does not require shifting elements.
3. **No Contiguous Memory Requirement:** LinkedList does not need contiguous memory allocation, making it flexible and suitable for unpredictable memory situations.
4. **Easy Modification:** LinkedList allows easy modification of elements by changing reference pointers, enabling efficient manipulation.

Types of Linked list

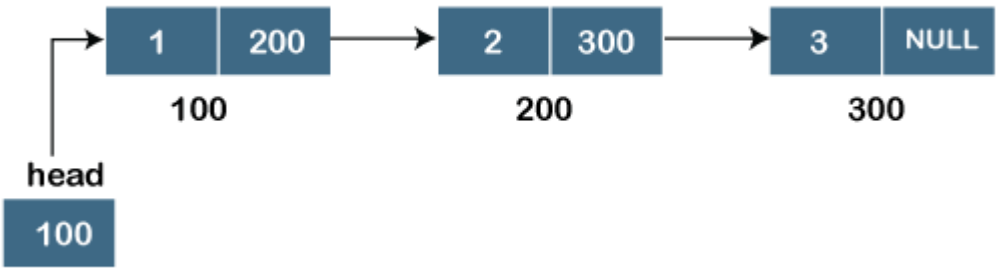
The following are the types of linked list:

- Singly Linked list
- Circular Linked list

Singly Linked list

It is the commonly used linked list in programs. The singly linked list is a data structure that contains two parts, one is the data part, and the other one is the address part, which contains the address of the next node. Traversing a singly linked list is done in a forward direction.

Suppose we have three nodes, and the addresses of these three nodes are 100, 200 and 300 respectively. The representation of three nodes as a linked list is shown in the below figure:



The linked list, which is shown in the above diagram, is known as a singly linked list as it contains only a single link. In this list, only forward traversal is possible; we cannot traverse in the backward direction as it has only one link in the list.

Program :

```
import java.util.Scanner;

public class linkedlist {
   class Node {
      int data;
      Node next;
```

```
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

Node head = null;

public void display() {
    if (head == null) {
        System.out.println("List is empty..");
        return;
    }
    Node n = head;
    while (n != null) {
        System.out.println(n.data + "-->");
        n = n.next;
    }
}

public void insertFirst(int data) {
    Node newnode = new Node(data);
    if (head == null) {
        head = newnode;
        return;
    }
    newnode.next = head;
    head = newnode;
}

public void insertLast(int data) {
    Node newnNode = new Node(data);
    if (head == null) {
        head = newnNode;
        return;
    }
    Node n = head;
    while (n.next != null) {
        n = n.next;
    }
    n.next = newnNode;
}

public void insertAt(int data, int index) {
    Node newnNode = new Node(data);
    if (index == 0) {
        insertFirst(data);
        return;
    }
    Node n = head;
    for (int i = 0; i < index - 1; i++) {
```

```
            n = n.next;
        }
        newnNode.next = n.next;
        n.next = newnNode;
    }

    public void deleteFirst() {
        if (head == null) {
            System.out.println("List is already empty..");
            return;
        }
        if (head.next == null) {
            head = null;
            return;
        }
        head = head.next;
        System.out.println("First element deleted");
    }

    public void deletelast() {
        if (head == null) {
            System.out.println("List is already empty..");
            return;
        }
        if (head.next == null) {
            head = null;
            return;
        }
        Node n = head;
        while (n.next.next != null) {
            n = n.next;
        }
        n.next = null;
    }

    public void deleteAt(int index) {
        if(head==null){
            System.out.println("List is empty..");
            return;
        }
        if (index == 0) {
            deleteFirst();
            return;
        }
        Node n = head;
        for (int i = 0; i < index - 1; i++) {
            n = n.next;
        }
        n.next = n.next.next;
    }

    public static void main(String[] args) {
```

```
linkedlist list = new linkedlist();
int ch, n;
Scanner sc = new Scanner(System.in);
while (true) {
   System.out.println("1. Add first");
   System.out.println("2. Add last");
   System.out.println("3. Add between");
   System.out.println("4. Display list");
   System.out.println("5. Delete first");
   System.out.println("6. Delete last");
   System.out.println("7. Delete between");
   System.out.println("8. Exit");
   System.out.print("ENTER YOUR CHOICE : ");
   ch = sc.nextInt();

   switch (ch) {
      case 1:
         System.out.print("Enter a number you want to insert at first :");
         n = sc.nextInt();
         list.insertFirst(n);
         break;
      case 2:
         System.out.print("Enter a number you want to insert at last :");
         n = sc.nextInt();
         list.insertLast(n);
         break;
      case 3:
         System.out.print("Enter position :");
         int pos = sc.nextInt();
         System.out.print("Enter value : ");
         n = sc.nextInt();
         list.insertAt(n, pos);
         break;
      case 4:
         list.display();
         break;
      case 5:
         list.deleteFirst();
         break;
      case 6:
         list.deletelast();
         break;
      case 7:
         System.out.print("Enter position :");
         int pos1 = sc.nextInt();
         list.deleteAt(pos1);
         break;
      case 8:
         System.exit(0);
      default:
         System.out.println("Invalid choice..");
   }
```

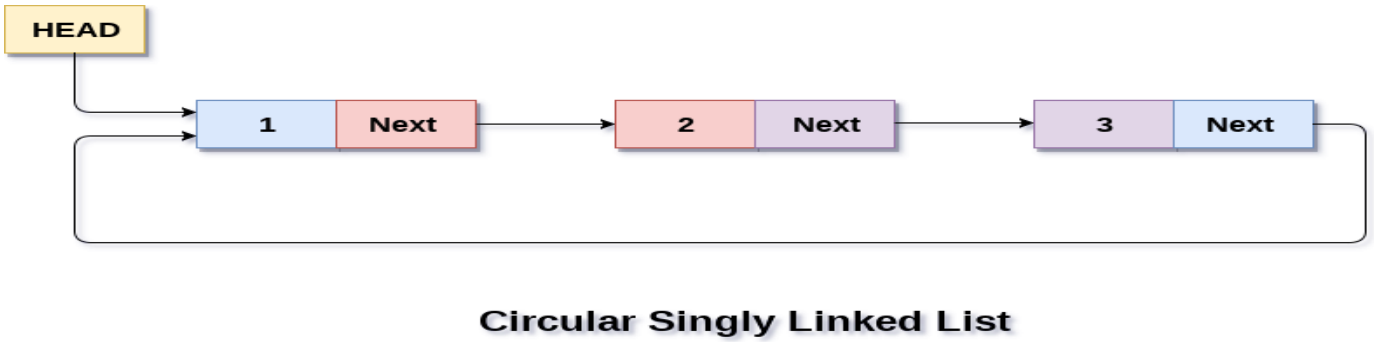
```
    }
  }
}
```

Circular linked list

A circular linked list is a variation of a singly linked list.

the circular linked list is a list in which the last node connects to the first node, so the link part of the last node holds the first node's address. The circular linked list has no starting and ending node.

The circular linked list is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.



Circular Singly Linked List

```
import java.util.Scanner;

public class CircularLL {
   class Node {
      int data;
      Node next;

      Node(int data) {
         this.data = data;
         this.next = null;
      }
   }

   Node head = null;
   Node tail = null;
```

```
public void display() {
   if (head == null) {
      System.out.println("List is empty");
      return;
   }
   Node n = head;
   do {
      System.out.println(n.data + "-->");
      n = n.next;
   } while (n != head);
}

public void insertFirst(int data) {
   Node newnode = new Node(data);
   if (head == null) {
      head = newnode;
      tail = newnode;
      newnode.next = head;
      return;
   }
   newnode.next = head;
   head = newnode;
   tail.next = head;
}

public void insertLast(int data) {
   Node newnode = new Node(data);
   if (head == null) {
      head = newnode;
      tail = newnode;
      newnode.next = head;
      return;
   }
   tail.next = newnode;
   tail = newnode;
   tail.next = head;
}

public void insertAt(int data, int index) {
   Node newnNode = new Node(data);
   if (index == 0) {
      insertFirst(data);
      return;
   }
```

```
    Node n = head;
    for (int i = 0; i < index - 1; i++) {
        n = n.next;
    }
    newnNode.next = n.next;
    n.next = newnNode;
}

public void deleteFirst(){
    if(head==null){
        System.out.println("List is empty..");
        return;
    }
    if(head.next==head){
        head=null;
        tail=null;
        return;
    }
    head=head.next;
    tail.next=head;
}
public void deleteLast(){
    if(head==null){
        System.out.println("List is empty..");
        return;
    }
    if(head.next==head){
        head=null;
        tail=null;
        return;
    }
    Node n=head;
    do{
        n=n.next;
    }while(n.next.next!=head);
    tail=n;
    tail.next=head;
}
public void deleteAt(int index) {
    if(head==null){
        System.out.println("List is empty..");
        return;
    }
    if (index == 0) {
```

```
            deleteFirst();
            return;
        }
        Node n = head;
        for (int i = 0; i < index - 1; i++) {
            n = n.next;
        }
        n.next = n.next.next;
    }

    public static void main(String[] args) {
        CircularLL list = new CircularLL();
        int ch, n;
        Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.println("1. Add first");
            System.out.println("2. Add last");
            System.out.println("3. Add between");
            System.out.println("4. Display list");
            System.out.println("5. Delete first");
            System.out.println("6. Delete last");
            System.out.println("7. Delete between");
            System.out.println("8. Exit");
            System.out.print("ENTER YOUR CHOICE : ");
            ch = sc.nextInt();

            switch (ch) {
                case 1:
                    System.out.print("Enter a number you want to insert at first :");
                    n = sc.nextInt();
                    list.insertFirst(n);
                    break;
                case 2:
                    System.out.print("Enter a number you want to insert at last :");
                    n = sc.nextInt();
                    list.insertLast(n);
                    break;
                case 3:
                    System.out.print("Enter position :");
                    int pos = sc.nextInt();
                    System.out.print("Enter value : ");
                    n = sc.nextInt();
                    list.insertAt(n, pos);
                    break;
```

```
            case 4:
                list.display();
                break;
            case 5:
                list.deleteFirst();
                break;
            case 6:
                list.deleteLast();
                break;
            case 7:
                System.out.print("Enter position :");
                int pos1 = sc.nextInt();
                list.deleteAt(pos1);
                break;
            case 8:
                System.exit(0);
            default:
                System.out.println("Invalid choice..");
        }
    }
  }
}
```

Java Applet

Applet is a special type of program that is **embedded** in the webpage **to generate** the dynamic content. It runs inside the browser and works at client side.

An Applet execution is **not independent**. It is dependent on Java Enabled web browser. They can be run using the AppletViewer or any Web browser that supports java.

It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation.

Applet codes uses the services of two classes namely **Applet** and **Graphics** from the java class library.

Important points :

1. All applets are sub-classes of *java.applet.Applet* class.
2. In general, execution of an applet does not begin at main() method.

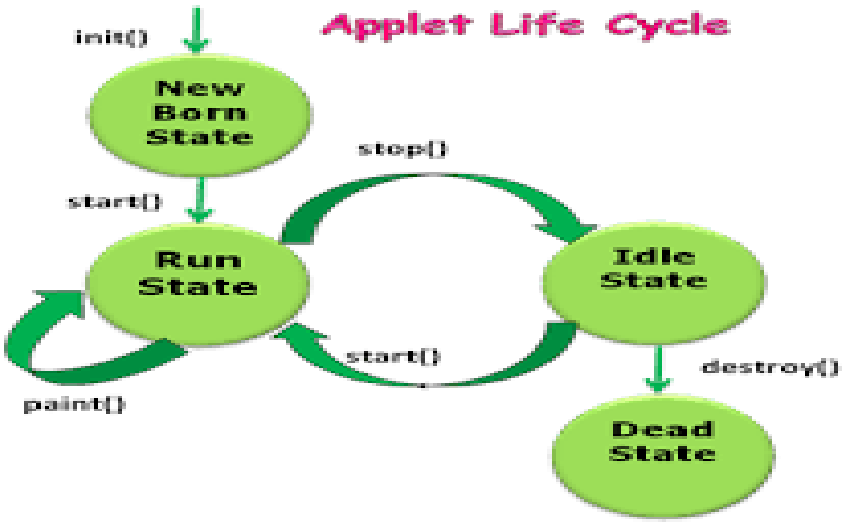
Applet Life Cycle in Java

The applet life cycle can be defined as the process of how the object is created, started, stopped, and destroyed during the entire execution of its applet. It basically has five core methods namely init(), start(), stop(), paint() and destroy().

Applet Life Cycle Working

- The **Java plug-in software** is responsible for managing the life cycle of an applet.
- An applet is a Java application executed in any web browser and works on the client-side. It doesn't have the main() method because it runs in the browser.
- The init(), start(), stop() and destroy() methods belongs to the **applet.Applet** class.
- The paint() method belongs to the **awt.Component** class.
- In Java, if we want to make a class an Applet class, we need to extend the **Applet**

1. Born or initialization state
2. Running state
3. Idle State
4. Dead or destroyed state



- **Initialization state :**

  When an applet is loaded, it enters the initialization state. This happens by calling the init() method of Applet class. It can be invoked only once at the time of initialization. At this stage, the following actions may be taken:

  1. Create object as required by the Applet
  2. Set up initial values
  3. Load images or fonts
  4. Set up colors

  To go with any of the above points we need to override init() method:

```
public void init(){
        //code of initializing the applet
}
```

- **Running state :**

  Applet enters the running state when the system calls the start() method that contains the actual code of the applet and starts the applet. It is invoked immediately after the init() method is invoked. Every time the browser is loaded or refreshed, the start() method is invoked. It is also invoked whenever the applet is maximized, restored, or moving from one tab to another in the browser. It is in an inactive state until the init() method is invoked.

  the paint() method belongs to the Graphics class in Java. It is used to draw shapes like circle, square etc. in the applet. It is executed after the start() method and when the browser or applet windows are resized.

- **Idle state :**

  An applet becomes idle when it is stopped from running. The stop() method stops the execution of the applet. whenever the applet is stopped, minimized, or moving from one tab to another in the browser, the stop() method is invoked. When we go back to that page, the start() method is invoked again.

- **Dead state :**

  The destroy() method destroys the applet after its work is done. It is invoked when the applet window is closed or when the tab containing the webpage is closed. It removes the applet object from memory and is executed only once. We cannot start the applet once it is destroyed.

Sequence of method execution when an applet is executed:

1. init()
2. start()
3. paint()
4. stop()
5. destroy()

Who is responsible to manage the life cycle of an applet?

Java Plug-in software.

Applet Architecture

Applets are event driven. An applet waits until an event occurs. The AWT notifies the applet about an event by calling an event handler that has been provided by the Applet. Once this happens, the applet must take the appropriate action and then quickly return control to the AWT.

The user interacts with the applet as he/she wants .The interactions are sent to the applet as events to which the applet must response. i.e. when the user clicks a mouse, a mouse-clicked event is generated.

The java AWT makes applet architecture simple

How to run an Applet? (inside architecture)

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 that resides on the browser.

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.

Syntax

```
<applet code="URL" height="200" width="100">.....</applet
```

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

What is <applet> tag? How to pass parameter in it?

HTML <applet> tag was used to embed the Java applet in an HTML document.

The use of Java applet is deprecated, and most browsers do not support the use of plugins.

Note: The <applet> tag is deprecated in HTML4.0 and not supported in HTML5. So you can use <object> tag or <embed> tag instead of <applet>.

Syntax

```
<applet code="URL" height="200" width="100">.....</applet>
```

Attribute name	Value	Description
code	URL	It specifies the URL of Java applet class file.
width	pixels	It specifies the display width of the applet panel.
height	pixels	It specifies the display height of applet panel

align	○ left ○ right ○ top ○ middle ○ bottom	It specifies the position of applet application relative to surrounding content.
alt	text	It is used to display alternative text in case browser does not support Java.

Parameter in Applet

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named **getParameter().**

Parameters specify extra information that can be passed to an applet from the HTML page. Parameters are specified using the HTML's *<param>* tag.

The <param> tag is a sub tag of the <applet> tag. The <param> tag contains two attributes: *name* and *value* which are used to specify the name of the parameter and the value of the parameter respectively.

Syntax: **public** String getParameter(String parameterName)

Example of using parameter in Applet:

```
import java.applet.Applet;
import java.awt.Graphics;
 /* <applet code="UseParam.class" width="300" height="300">
        <param name="msg" value="Welcome to applet">
</applet>  */
public class UseParam extends Applet{

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

Graphics in Applet

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

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public abstract void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color with specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color with specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of Graphics in applet:

```
import java.applet.Applet;
import java.awt.*;
 /* <applet code="GraphicsDemo.class" width="300" height="300">
</applet> */

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);

}
}
```

Applet viewer

Applet viewer is a command-line program to run Java applets. It is included in the SDK. It helps you to test an applet before you run it in a browser. An applet is a special type of application that's included as a part of an HTML page and can be stored in a web page and run within a web browser. The applet's code gets transferred to the system and then the Java Virtual Machine (JVM) of the browser executes that code and displays the output. So for running the applet, the browser should be Java enabled. To create an applet, we need to define a class that inherits the Applet. We generally use web browsers to run applets. Its not always mandatory to open a Web browser for running an applet.

Java Application Vs. Java Applet

Parameters	Java Application	Java Applet
Meaning	A Java Application also known as application program is a type of program that independently executes on the computer.	The Java applet works on the client side, and runs on the browser and makes use of another application program so that we can execute it.
Requirement of main( ) method	Its execution starts with the main( ) method only. The use of the main( ) is mandatory.	It does not require the use of any main() method. Java applet initializes through init( ) method.
Execution	It cannot run independently, but requires JRE to run.	It cannot start independently but requires APIs for use (Example. APIs like Web API).
Installation	We need to install the Java application first and obviously on the local computer.	Java applet does not need to be pre-installed.
Connectivity with server	It is possible to establish connections with other servers.	It cannot establish connection to other servers.

Operation	It performs read and write tasks on a variety of files located on a local computer.	It cannot run the applications on any local computer.
File access	It can easily access a file or data available on a computer system or device.	It cannot access the file or data found on any local system or computer.
Security	Java applications are pretty trusted, and thus, come with no security concerns.	Java applets are less reliable. So, they need to be safe.