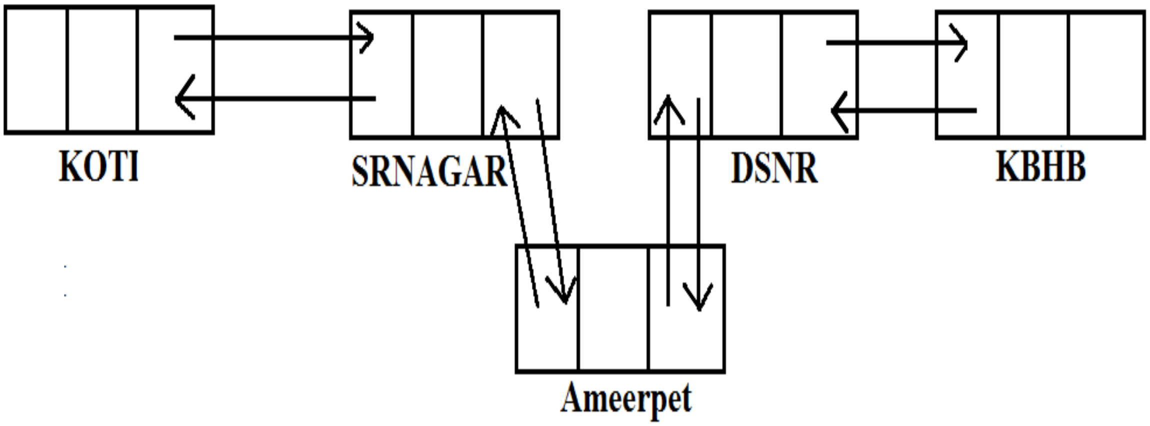
LinkedList:

1. The underlying data structure is double LinkedList
2. If our frequent operation is insertion (or) deletion in the middle then LinkedList is the best choice.
3. If our frequent operation is retrieval operation then LinkedList is worst choice.
4. Duplicate objects are allowed.
5. Insertion order is preserved.
6. Heterogeneous objects are allowed.
7. Null insertion is possible.
8. Implements Serializable and Cloneable interfaces but not RandomAccess.

Diagram:



Usually we can use LinkedList to implement Stacks and Queues.
To provide support for this requirement LinkedList class defines the following 6 specific methods.

1. void addFirst(Object o);
2. void addLast(Object o);
3. Object getFirst();
4. Object getLast();
5. Object removeFirst();
6. Object removeLast();

We can apply these methods only on LinkedList object.

Constructors:

1. LinkedList l=new LinkedList();
   Creates an empty LinkedList object.

2. **LinkedList l=new LinkedList(Collection c);**
   To create an equivalent LinkedList object for the given collection.

<u>Example:</u>

```
import java.util.*;
class LinkedListDemo
{
        public static void main(String[] args)
        {
                LinkedList l=new LinkedList();
                l.add("ashok");
                l.add(30);
                l.add(null);
                l.add("ashok");
                System.out.println(l);//[ashok, 30, null, ashok]
                l.set(0,"software");
                System.out.println(l);//[software, 30, null,
ashok]
                l.set(0,"venky");
                System.out.println(l);//[venky, 30, null, ashok]
                l.removeLast();
                System.out.println(l);//[venky, 30, null]
                l.addFirst("vvv");
                System.out.println(l);//[vvv, venky, 30, null]
        }
}
```

Vector:

1. **The underlying data structure is resizable array (or) growable array.**
2. **Duplicate objects are allowed.**
3. **Insertion order is preserved.**
4. **Heterogeneous objects are allowed.**
5. **Null insertion is possible.**
6. **Implements Serializable, Cloneable and RandomAccess interfaces.**

Every method present in Vector is synchronized and hence Vector is Thread safe.

<u>Vector specific methods:</u>

<u>To add objects:</u>

1. **add(Object o);-----Collection**
2. **add(int index,Object o);-----List**
3. **addElement(Object o);-----Vector**

<u>To remove elements:</u>

1. **remove(Object o);--------Collection**
2. **remove(int index);--------------List**
3. **removeElement(Object o);----Vector**
4. **removeElementAt(int index);-----Vector**
5. **removeAllElements();-----Vector**
6. **clear();-------Collection**

<u>To get objects:</u>

1. **Object get(int index);---------------List**
2. **Object elementAt(int index);-----Vector**
3. **Object firstElement();--------------Vector**
4. **Object lastElement();---------------Vector**

<u>Other methods:</u>

1. **Int size();//How many objects are added**
2. **Int capacity();//Total capacity**
3. **Enumeration elements();**

<u>Constructors:</u>

1. **Vector v=new Vector();**
   - **Creates an empty Vector object with default initial capacity 10.**
   - **Once Vector reaches its maximum capacity then a new Vector object will be created with double capacity. That is "newcapacity=currentcapacity*2".**
2. **Vector v=new Vector(int initialcapacity);**
3. **Vector v=new Vector(int initialcapacity, int incrementalcapacity);**
4. **Vector v=new Vector(Collection c);**

<u>Example:</u>

```
import java.util.*;
class VectorDemo
{
      public static void main(String[] args)
      {
            Vector v=new Vector();
            System.out.println(v.capacity());//10
            for(int i=1;i<=10;i++)
            {
                  v.addElement(i);
            }
      System.out.println(v.capacity());//10
      v.addElement("A");
      System.out.println(v.capacity());//20
      System.out.println(v);//[1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
A]
      }
```

}

Stack:

1. It is the child class of Vector.
2. Whenever last in first out(LIFO) order required then we should go for Stack.

Constructor:

It contains only one constructor.

Stack s= new Stack();

Methods:

1. Object push(Object o);
   To insert an object into the stack.
2. Object pop();
   To remove and return top of the stack.
3. Object peek();
   To return top of the stack without removal.
4. boolean empty();
   Returns true if Stack is empty.
5. Int search(Object o);
   Returns offset if the element is available otherwise returns "-1"

Example:

```
import java.util.*;
class StackDemo
{
	public static void main(String[] args)
	{
		Stack s=new Stack();
		s.push("A");
		s.push("B");
		s.push("C");
		System.out.println(s);//[A, B, C]
		System.out.println(s.pop());//C
		System.out.println(s);//[A, B]
		System.out.println(s.peek());//B
		System.out.println(s.search("A"));//2
		System.out.println(s.search("Z"));//-1
		System.out.println(s.empty());//false
	}
}
```

The 3 cursors of java:

If we want to get objects one by one from the collection then we should go for cursor. There are 3 types of cursors available in java. They are:

1. **Enumeration**
2. **Iterator**
3. **ListIterator**

Enumeration:

1. **We can use Enumeration to get objects one by one from the legacy collection objects.**
2. **We can create Enumeration object by using elements() method.**
   public Enumeration elements();
   Enumeration e=v.elements();
   using Vector Object

Enumeration interface defines the following two methods

1. **public boolean hasMoreElements();**
2. **public Object nextElement();**

Example:

```
import java.util.*;
class  EnumerationDemo
{
      public static void main(String[] args)
      {
            Vector v=new Vector();
            for(int i=0;i<=10;i++)
            {
                  v.addElement(i);
            }
            System.out.println(v);//[0, 1, 2, 3, 4, 5, 6, 7, 
8, 9, 10]
            Enumeration e=v.elements();
            while(e.hasMoreElements())
            {
                  Integer i=(Integer)e.nextElement();
                  if(i%2==0)
                        System.out.println(i);//0 2 4 6 8 10
            }
            System.out.print(v);//[0, 1, 2, 3, 4, 5, 6, 7, 8, 
9, 10]
      }
}
```

Limitations of Enumeration:

1. **We can apply Enumeration concept only for legacy classes and it is not a universal cursor.**
2. **By using Enumeration we can get only read access and we can't perform remove operations.**
3. **To overcome these limitations sun people introduced Iterator concept in 1.2v.**

Iterator:

1. **We can use Iterator to get objects one by one from any collection object.**
2. **We can apply Iterator concept for any collection object and it is a universal cursor.**
3. **While iterating the objects by Iterator we can perform both read and remove operations.**

We can get Iterator object by using iterator() method of Collection interface.
public Iterator iterator();
Iterator itr=c.iterator();

Iterator interface defines the following 3 methods.

1. **public boolean hasNext();**
2. **public object next();**
3. **public void remove();**

Example:

```
import java.util.*;
class IteratorDemo
{
      public static void main(String[] args)
      {
            ArrayList a=new ArrayList();
            for(int i=0;i<=10;i++)
            {
                  a.add(i);
            }
            System.out.println(a);//[0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10]
            Iterator itr=a.iterator();
            while(itr.hasNext())
            {
                  Integer i=(Integer)itr.next();
                  if(i%2==0)
                        System.out.println(i);//0, 2, 4, 6,
8, 10
                  else
                        itr.remove();
```

```
                }
                System.out.println(a);//[0, 2, 4, 6, 8, 10]
        }
}
```

<u>**Limitations of Iterator:**</u>

1. **Both enumeration and Iterator are single direction cursors only. That is we can always move only forward direction and we can't move to the backward direction.**
2. **While iterating by Iterator we can perform only read and remove operations and we can't perform replacement and addition of new objects.**
3. **To overcome these limitations sun people introduced listIterator concept.**

ListIterator:

1. **ListIterator is the child interface of Iterator.**
2. **By using listIterator we can move either to the forward direction (or) to the backward direction that is it is a bi-directional cursor.**
3. **While iterating by listIterator we can perform replacement and addition of new objects in addition to read and remove operations**

By using listIterator method we can create listIterator object.

public ListIterator listIterator();

ListIterator itr=l.listIterator();

(l is any List object)

<u>**ListIterator interface defines the following 9 methods.**</u>

1. **public boolean hasNext();**
2. **public Object next(); forward**
3. **public int nextIndex();**
4. **public boolean hasPrevious();**
5. **public Object previous(); backward**
6. **public int previousIndex();**
7. **public void remove();**
8. **public void set(Object new);**
9. **public void add(Object new);**

<u>**Example:**</u>

```
import java.util.*;
class  ListIteratorDemo
{
        public static void main(String[] args)
        {
            LinkedList l=new LinkedList();
            l.add("balakrishna");
            l.add("venki");
            l.add("chiru");
            l.add("nag");
            System.out.println(l);//[balakrishna, venki,
chiru, nag]
            ListIterator itr=l.listIterator();
            while(itr.hasNext())
            {
                  String s=(String)itr.next();
                  if(s.equals("venki"))
                  {
                        itr.remove();
                  }
            }
            System.out.println(l);//[balakrishna, chiru, nag]
      }
}
```