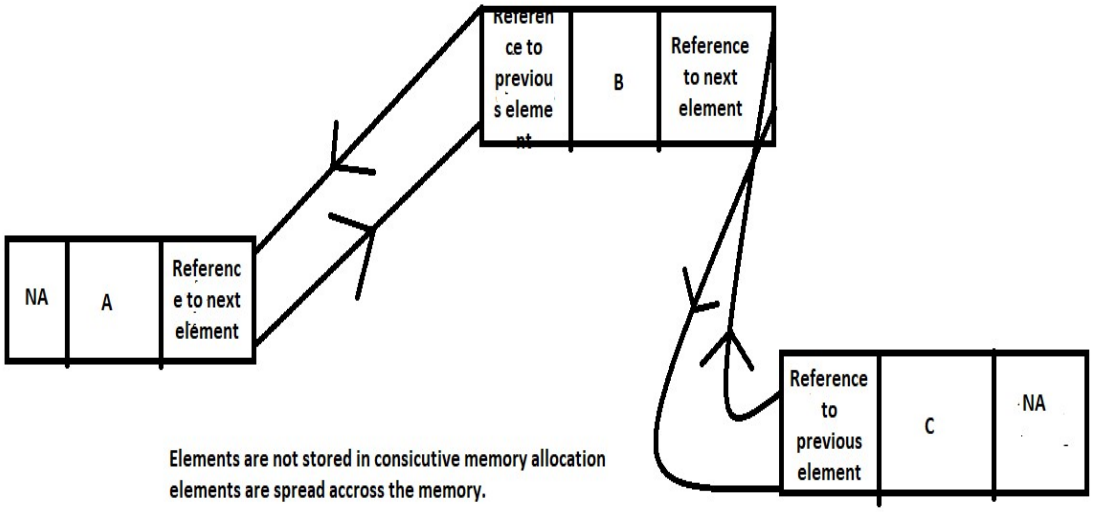
Linked List:

- Child class of List Interface.
- Underlying data structure is doublyLinkedList.
- Memory allocation is not consecutive objects present in linked list are stored across the memory and they are linked with each other with the help of references.

e.g



- Insertion order is preserved and duplicate objects are allowed.
- Best suitable if our frequent operation is insertion or deletion in the middle.
- Not recommended if our frequent operation is retrival operation.
- Heteroginious objects are allowed.
- Null insertion is possible.
- LinkedList implements serialaizable and clonable but not Random access interface.
- Capacity concept is not available for the LinkedList.

Constructors:

- LinkedList <String> l=new LinkedList<String>();//Creates an empty LinkedList
- LinkedList <String> l=new LinkedList<String>(Collection C);//creates an equivalent LinkedList object for the given collection.

LinkedList class Specific methods:

- Usually we can use LinkedList to develope stacks and queues so to provide support linkedList class defines following methods.
- void addFirst(Object o).
- void addLast(Object o).

Vector:

- Child class of List interface.
- Underlying Data Structure is Resizable array.
- Duplicates are allowed.
- Insertion order is preserved.
- Hterogeneous objects are allowed.
- It implements serialaizable,clonable,randomaccess interface.
- Every method present in vector is synchronized and hence vector object is thread safe.
- Vector is also known as Legacy class because introduced in java version 1.0.

Constructors in Vector:

- Vector <Object> v=new Vector<Object>();
- Here vector v will create a new vector object of default capacity 10 and once the vector reaches the max capacity a new vector will be created of by doubling the size(newcapacity=previouscapacity*2) and all the elements will be added in the new vector.
- Vector v=new Vector(int initialCapacity)
- Above constructor will create a new vector object with specified capacity.

- Vector v=new Vector(int initialcapacity, int incrementalcapacity);
- Vector v=new Vector(Collection c)//Creates an equivalent vector for given collection.
- This collection is meant for interconversion between collection objects.

Stack:

- It is the child class of vector.
- It is a specially designed class for LIFO(Last in First out)

Constructors specifically for stack:

- Stack s=new Stack()//Will create a new empty Stack.

Stack Specific Methods:

- Object push(Object o)//Insert an object into the stack
- Object pop()//Remove and return top of the stack
- Object peek()//Return the top of the stack without removing
- Boolean empty()//Return true if the stack is empty
- Int search(Object o)//Return the offset if the object is present otherwise return -1.

E.g:

```
Stack<String> s=new Stack<String>();
          s.add("Tusar");//offset 4
          s.add("Sangram");//offset 3
          s.add("Marc");//offset 2
          s.add("Starc");//offset 1
          System.out.println(s);
          System.out.println(s.search("Tusar"));//Return 4
          System.out.println(s.search("Mom"));//Return -1
          System.out.println(s.peek());//Return Starc from
the top here the top is Starc and bottom is Tusar
          s.pop();//Will remove the object from the top of
the stack
          System.out.println(s);//[Tusar, Sangram, Marc]
```

The 3 Cursors of JAVA:

- There are 3 types of cursors present in java which helps us to iterate over a collection to get the objects one by one.

1. **Enumuration:**

- We can use enumuraton to get objects one by one from legacy collection object.
- We can create enumeration object by using elements method of vector class.
  Public Enumuration e=v.elements()

Methods:

- Public Boolean hasMoreElements();
- Public Object nextElement();

Limitations:

- We can apply enumeration concept only for legacy classes and it is not a universal cursor.
- By using enumeration we can get only read access and we can't perform remove operation.
- To overcome above limitations we should go for iterator.

Iterator:

- We can apply Iterator concept for any collection object and hence it is universal cursor.
- By using Iterator we can perform read and remove operation.

Methods of Iterator:

- We can create iterator object by using iterator method of collection interface.

  Public Iterator iterator();
  Eg:
  Iterator itr=c.Iterator();//Here c is any collection object

- public Boolean hasnext();
- public object next();
- public void remove();

E.g:

```
ArrayList<Integer> al=new ArrayList<Integer>();
          for(int i=0;i<10;i++) {
               al.add(i);
          }
          System.out.println(al);//[0, 1, 2, 3, 4, 5, 6, 7,
8, 9]
          Iterator itr=al.iterator();
          while(itr.hasNext()) {
               Integer i=(Integer) itr.next();//it will
give the vale one by one
               System.out.println(i);//0, 1, 2, 3, 4, 5, 6,
7, 8, 9
               if(i%2==0) {
               System.out.println(i);//0,2,4,6,8
               }else {
                    itr.remove();//it will remove the
value.
               }
          }

          System.out.println(al);
     }
```

Limitations of Iterator:

- By using enumeration and iterator we can always move towards forward direction and we can't move towards backward direction.
- These are single directional coursor but not bidirectional.
- By using iterator we can perform only read and remove operation and we can't perform replace or addition of new objects.
- To overcome the above limitations we have listIterator cursor.

ListIterator:

- It is a bidirectional cursor means by using ListIterator we can move both forward and backword direction.
- We can add or replace objects from the middle with the help of ListIterator.
- We can create ListIterator by using listIterator method of List interface.

  Public ListIterator=listIterator();

  Eg:

  ListIterator ltr=l.listIterator();//here l is any list object.
- ListIterator is the child interface of Iterator, Hence all the methods present in Iterator are bydeafult available to ListIterator.

Methods Present in ListIterator:

- Public boolean hasNext();
- Public object next();
- Public int nextIndex();
- Above methods are for forward movement.
- Public Boolean hasPrevious();
- Public object previous();
- Public int previousIndex();
- Above methods are meant for backward movement.
- Public void remove();
- Public void add(object o)
- Public void set(object o)

e.g:

```
LinkedList<String>ll=new LinkedList<>();
ll.add("Tusar");
ll.add("Rocky");
ll.add("Lal");
ll.add("Master");
//System.out.println(ll);
ListIterator<String> l1=ll.listIterator();
```

```
while(l1.hasNext()) {
     String s=(String) l1.next();
     if(s.equals("Tusar")) {
          l1.remove();
          //System.out.println(ll);
     }
     else if(s.equals("Lal")) {
          l1.add("Tusar");
          System.out.println(l1);
```

- The most powerful cursor is listiterator but its limitation is it is only applicable for list object.