1. **LinkedList-**
    1. It is child class of collection.
    2. It is present in Java.util package.
    3. Insertion order is preserved.
    4. Underlying data structure is double linked list.
    5. Duplicates are allowed.
    6. Heterogeneous objects are allowed.
    7. Null insertion is possible.
    8. LinkedList will implements serializable and clonable interface but not random access interface.
    9. LinkedList is best choice if our frequent operation is insertion and deletion in middle.
    10. LinkedList is worst choice if our frequent operation is retrival.

Methods

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

Constructor

1. LinkedList l= new LinkedList();
   Create the empty linkedlist object.

2. LinkedList ll= new LinkedList(Collection c);
   Create the object for given collection.

Example-

```
import java.util.ArrayList;

import java.util.Collections;

import java.util.LinkedList;

import java.util.List;


public class LinkedListDemo {
```

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

                        LinkedList al = new LinkedList();
                        al.add(50);
                        al.add("Ajay");
                        al.add(10);
                        al.add(null);
                        al.addFirst("Pune");
                        al.add(2, "Software");
                        System.out.println(al);
                    }
}
package com.linkedlist;

import java.util.Iterator;
import java.util.LinkedList;

public class LinkedListDemo {

                    public static void main(String[] args) {

                        LinkedList<Integer> linkedList= new
LinkedList<Integer>();
                        linkedList.add(10);
                        linkedList.add(40);
                        linkedList.add(30);
                        linkedList.add(20);
                        linkedList.addFirst(70);
```

```
				//linkedList.addLast(80);
				
				int a=linkedList.get(3);
				System.out.println("4th index is="+a);
				
//				System.out.println("first way="+linkedList);
				
				Iterator<Integer> itr= linkedList.iterator();
				while(itr.hasNext()) {
					System.out.println(""+itr.next());
				}
				
				for(Integer t : linkedList) {
					System.out.println("using for each 
loop="+t);
				}
				
				
			}
}
```

Difference between ArrayList and LinkedList.

ArrayList	LinkedList
It is best choice if our frequent operation is retrieval.	It is the best choice if our frequent operation is insertion and deletion.
It is worst choice if our frequent operation is insertion and deletion.	It is the worst choice if our frequent operation is retrieval.
Underlying data structure for arraylist is resizable or growable array.	Underlying data structure is double linked list.

3. Vector-

1. The underlying data structure for vector is growable or resizable array.
2. Duplicates objects are allowed.
3. Insertion order is preserved.
4. Null insertion is possible.
5. Heterogenous objects are allowed.
6. Vector class implements serializable, clonable and randmoaccess interface.
7. Most of methods are present in vector are synchronized.
8. Hence vector object is thread safe.
9. Best choice if frequent operation is retrieval.

Methods-

1) To Add Elements:
- add(Object o) -> Collection
- add(int index, Object o) ->List
- addElement(Object o) ->Vector

2) To Remove Elements:
- remove(Object o) ->Collection
- removeElement(Object o) ->Vector
- remove(int index) ->List
- removeElementAt(int index) ->Vector
- clear()->Collection
- removeAllElements()->Vector

3) To Retrive Elements:
- Object get(int index) ->List
- Object elementAt(int index) ->Vector
- Object firstElement()->Vector
- Object lastElement()->Vector

4) Some Other Methods:
- int size()
- int capacity()
- Enumeration element()

Constructor-

1. Vector v= new Vector();
   Create the empty object with default initial capacity 10, once the vector reaches its max capacity a new vector object will be created with

   New capacity= 2*current capacity.

2. Vector v= new Vector(int initialcapacity);
   Create the empty vector object with specified initial capacity.

3. Vector v= new Vector(int initialcapacity, int incrementalcapacity)

4. Vector v= new Vector(Collection c);

   Create the equivalent vector object for given collection.

Example-

```
import java.util.Vector;

public class Test {

	public static void main(String args[]) {

		Vector v = new Vector();
		System.out.println(v.capacity());
		for (int i = 1; i <= 10; i++) {
			v.addElement(i);
		}
		System.out.println(v.capacity());
		v.addElement("J");
		System.out.println(v.capacity());
		System.out.println(v);
	}

}
```

Output-

```
10
10
20
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, J]
```

```
}
```

```
package com.linkedlist;

import java.util.Vector;

public class VectorDemo {

        public static void main(String[] args) {

                Vector<String> vector= new Vector<String>();
                vector.add("Ajay");
                vector.add("velocity");
                vector.add("Pune");
                vector.add("Pune");

                while(vector.contains("Pune")) {
                        vector.remove("Pune");
                }

                System.out.println("new vector is="+vector);

                /*System.out.println("vector size="+vector.size());
                System.out.println("vector capacity="+vector.capacity());
                System.out.println("index="+vector.get(2));
        */


        }
}

package com.linkedlist;

import java.util.*;

public class VectorDemo2 {

        public static void main(String[] args) {

                Vector v= new Vector();

                for(int i=1;i<=10;i++) {

                        v.addElement(i);
                }

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

ArrayList	Vector
Every method present in arraylist is non synchronized.	Every method present in vector is synchronized.
At a time multiple threads are allowed to operate on arraylist object, hence arraylist is not thread safe.	At a time only one threads is allowed to operate on vector object, hence vector is thread safe.
Threads are not required to wait to operate on arraylist, hence performance is high.	Threads are required to wait to operate on vector, hence performance is low.
Introducted in 1.2 version, it is non-legacy class	Introducted in 1.0 version, it is legacy class

Note- Legacy class means which comes from old generation called as Legacy,

Stack and vector are legacy classes.

4. Stack-
   1. It is the child class of vector.
   2. It is specially design the class for Last In First Out (LIFO or FILO)

      Constructor

      Stack s= new Stack();

      Methods-

      1. Object push(Object obj);
         For inserting an object to stack.

      2. Object pop();
         To remove the return top of stack.

      3. Object peak();
         To return the top of stack without removal of object.

      4. int Search(Object obj);

if specified object is available it returns its offset from top of stack. If object is not available then it return -1.

Example-

```
import java.util.Stack;
import java.util.Vector;

public class StackDemo {

	public static void main(String args[]) {

		Stack stack = new Stack();
		stack.push("J");
		stack.push("M");
		stack.push("K");

		System.out.println(stack);
		System.out.println(stack.search("X"));//if element not found then 
return -1
	}

}
```

Output-

```
[J, M, K]
3
-1
```