```
package: java.util.*;
                         |-> refers to all classes/interfaces/enum present in the current package(util package)

Inside any of Collection What kind of data is been stored?
      A. primitive(no)
                  Even if the programmer gives primitive data, internally from JDK1.5Version JVM will use
                  wrapper class concepts to convert primitive type of data to object and it would be stored.
                              eg:    al.add(8);
                                                 |------>al.add( Integer.valueOf(8));
      B. Object(yes)

Collection(I) ----> it is the root interface for all type of collection
List(I)
      a. ArrayList
      b. LinkedList
                  To utilise the scattered/dispursed memory in efficient way we use LinkedList.
                  a. SinglyLinkedList
                  b. DoublyLinkedList
                        Note: All the Node creation and address maintainence is totally managed by JVM since
                                 java does not support the concept of pointers for programmer, this only is the
                                reson to say java is "abstract high level language".

      c. Vector  -> All the methods of Vector is synchronized(Thread safe, slow in execution)
                        this class also implements Random Access interface so it is best suited for "Retreival operation",
                        but it is not suited for insertion and deletion at the middle.
                              methods
                                    a. addElement(Object o)
                                    b. removeElement(Object o)
                                    c. removeAllElements()
                                    d. Object elelementAt(int idnex)
                                    e. Object firstElement()
                                    f.  Object lastElement()
            a. Stack
                  Constructors
                              a. Stack s =new Stack();
                        methods
                              a.  Object push(Object o)
                              b.  Object pop() -> removes and returns the top element of  the stack
                              c.  boolean empty()
                              d. Object peek() -> It will give the top element of the stack without removal
                              e. search(Object) -> it returns the offset if the element is present,otherwise it returns -1.


There are 3 cursor in java
   -> Inside collection data would stored as Objects.
```

=> After storing the data as Objects, it is comon requirement to take the Object one by one from Collection

=> To do this we have cursors in java

a. Enumeration(I) -> It is applicable for legacy classes only.

```
public interface Enumeration{
  public abstract boolean hasMoreElements();//will check in the collection whether elements are there are not
  public abstract E nextElement(); // this method will get the current cursor data and makes the cursor to point to next collection object
}
```

eg#1.

```
import java.util.*;

class Test
{
	public static void main(String[] args)
	{
		Vector v = new Vector();
		for (int i=1;i<=10 ; i++)
		{
			v.addElement(i);
		}
		System.out.println(v);//internally v.toString() is called.

		Enumeration e = v.elements();// to get the cursor
		System.out.println("Reading elements one by one from collection");
		while (e.hasMoreElements())
		{
			   Integer data=(Integer)e.nextElement();
			   System.out.println(data);
			   if (data%2==0){
				 System.out.println(data + ":  is an even number");
			   }
		}


	}
}
```

Limitation
----

1. It is applicable only on legacy classes
2. using this cursor we can perform only read operation and we can't perform remove operation
3. To resovle this problem only we use "iteator".

b. Iterator(I) ----> Universal Cursor(applied on any type of Collection Object)

```
public interface Iterator {
  public abstract boolean hasNext();//check whether the collection has next element or not
  public abstract E next(); //retreive the element and takes the cursor to the next element
  public void remove(); //to remove the object from collection.
  public void forEachRemaining(java.util.function.Consumer<? super E>);//Stream api's
}
import java.util.*;
```

```
class Test
{
      public static void main(String[] args)
      {
            ArrayList al = new ArrayList();
            for (int i=1;i<=10 ; i++)
            {
                  al.add(i);
            }
            System.out.println(al);//internally al.toString() is called.

            Iterator itr = al.iterator();
            System.out.println("Reading elements one by one from collection");
            while (itr.hasNext())
            {
                     Integer data=(Integer)itr.next();
                     System.out.println(data);
                     if (data%2==0)
                           System.out.println(data +" : is an even number");
                     else
                           itr.remove();


            }
            System.out.println(al);


      }
}
```

Limitation

1. Using this cursor we can move only in forward direction, not in backward direction so we say the cursor is "UniDirectional cursor".
2. Using this cursor we can perform only remove operation, operations like adding the object,replacing the object is not possible.
3. To overcome this limitation we need to use ListIterator.

c. ListIterator(I)

```
public interface java.util.ListIterator<E> extends java.util.Iterator<E> {

//for forward traversing
  public abstract boolean hasNext();
  public abstract E next();
  public abstract int nextIndex();

// for backward traversing
  public abstract boolean hasPrevious();
  public abstract E previous();
  public abstract int previousIndex();

//for operations like add,remove and update
  public abstract void set(E);
  public abstract void add(E);
  public abstract void remove();
}
```

Limitation

a. Eventhough it is a powerful cursor it can be applied only on List(I) implementation object,but not on all Collections.

Revise and get back for doubts

Set(I)

SortedSet(I)

NavigableSet(I)

Queue(I)

Concurrent Collections(java.util.concurrent.*)

failfast -> while one thread is trying to perform iteration on collection Object and if another thread is trying to do some structural modification to the same collection object, then immedieatly iterator would fail, by resulting in an exception called ConcurrentModificationException,such type of iterators are called as "FailFastIterator".

eg:

```
import java.util.*;

class Test
{
	public static void main(String[] args)
	{
		ArrayList al = new ArrayList();
		al.add("A");
		al.add("B");
		al.add("C");

		Iterator itr = al.iterator(); // fail fast iterator
		while (itr.hasNext())
		{
			String data = (String) itr.next();
			System.out.println(data);

		//Assume one more thread is doing up modification on ArrayList
			al.add("D");//Trying to change the structure of an ArrayList
		}
	}
}
```

If we don't want the exception to occur even during mulithreading events then prefers using "Concurrent Collections" which supports concurrent modifcations.

fail safe: while one thread is trying to perform iteration on collection Object and if another thread is trying to do some structural modification to the same collection object, then also iteration won't fail becoz the iterator is "fail safe iterator". here exception wont occur becoz every update operation will be performed on seperate cloned copy.

eg#1.

```
import java.util.concurrent.*;
import java.util.*;

class Test
{
	public static void main(String[] args)
	{
		CopyOnWriteArrayList al = new CopyOnWriteArrayList();
```

```
            al.add("A");
            al.add("B");
            al.add("C");

            Iterator itr = al.iterator(); //fail safe iterator
            while (itr.hasNext())
            {
                  String data = (String) itr.next();
                  System.out.println(data);

            //Assume one more thread is doing up modification on ArrayList
                  al.add("D");//Trying to change the structure of an ArrayList
            }
            System.out.println(al);
      }
}
```

It will be discussed in tommo session

Map(I)
NavigableMap(I)
SortedMap(I)