Cursors in Collections:

Property

1. ***Purpose***
2. ***Legacy or not***
3. ***Applicable for which type of classes***
4. ***Universal cursor or not***
5. ***How to get the object***
6. ***How many methods***
7. ***Operations***
8. ***Cursor moment***
9. ***Class or interface***
10. ***Versions supports***

6) It contains two methods
hasMoreElements(): to check objects.
nextElement() : to retrieve the objects.

7) Only read operations.

8) Only forward direction.

Enumeration

1) *Used to retrieve the data from collection classes.*

2) *Introduced in 1.0 version it is legacy*

3) It is used to retrieve the data from only legacy classes like vector, Stack…etc

4) Not a universal cursor because it is applicable for only legacy classes.

5) Get the Enumeration Object by using elements() method.

```
Vector v =new Vector();
        v.add(10);
        v.add(20);
   Enumeration e = v.elements();
```

9) Interface

10) It supports both normal and generic version.

Iterator

1) *Used to retrieve the objects from collection classes.*

2) *Introduced in 1.2 version it is not a legacy*

3) It is used to retrieve the data from all collection classes.

4) It is a universal cursor because it is applicable for all collection classes.

5) Get the iterator Object by using iterator() method.

```
Vector v =new Vector();
        v.add(10);
        v.add(20);
   Enumeration e = v.iterator();
```

6) It contains two methods
hasNext(): to check the objects available or not.
Next() : to retrieve the objects.

7) read & remove operations are possible.

8) Only forward direction.

9) Interface

10) It supports both normal and generic version.

ListIterator

1) *Used to retrieve the data from collection classes.*

2) *Introduced in 1.2 version it is not a legacy*

3) It is used to retrieve the data from only List type of classes like ArrayList,LinkedList,Vector,Stack.

4) Not a universal cursor because it is applicable for only List interface classes.

5) Get the ListIterator Object by using listIterator() method.

```
Vector v =new Vector();
        v.add(10);
        v.add(20);
    Enumeration e = v.listIterator();
```

6) It contains 9 methods

7) Read, remove, add, and replace operations.

8) Bidirectional cursor direction.

9) Interface

10) It supports both normal and generic version.

ListIterator methods:-

```
public abstract boolean hasNext();
public abstract E next();
public abstract boolean hasPrevious();
public abstract E previous();
public abstract int nextIndex();
public abstract int previousIndex();
public abstract void remove();
public abstract void set(E);//replacement
public abstract void add(E);
```

Retrieving objects of collections classes:-

We are able to retrieve the objects from collection classes in 3-ways

1) **By using for-each loop.**
2) **By using get() method.**
3) **By using cursors.**

Example application:-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<String> al =new ArrayList<String>();
                al.add("ratan");
                al.add("anu");
                al.add("sravya");
                //1st appraoch to print Collection data
                for (String  a  :  al )
                {       System.out.println(a);
                }

                //2nd approach to print Collection data
                int size = al.size();
```

```
        for (int i=0;i<size;i++)
        {       System.out.println(al.get(i));
        }

        //3rd approach to print Collection data
        //normal version of Iterator(type casting required at the time of retrieving)
        Iterator itr1 = al.iterator();
        while (itr1.hasNext())
        {       String str =(String)itr1.next();
                System.out.println(str);
        }
        //generic  version of Iterator(type casting not required at the time of retrieving)
        Iterator<String> itr2 = al.iterator();
        while (itr2.hasNext())
        {       String str =itr2.next();
                System.out.println(str);
        }
    }
}
```

Example:-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<String> al =new ArrayList<String>();
                al.add("ratan");
                al.add("anu");
                al.add("sravya");
                ListIterator<String> lstr = al.listIterator();
                lstr.add("suneel");
                while(lstr.hasNext())
                {       if ((lstr.next()).equals("anu"))
                        {       lstr.set("Anushka");
                        }
                }
                lstr.add("aaa");
                for (String str:al)
                {       System.out.println(str);        }
        }
}
```

E:\>java Test

suneel
ratan
Anushka
sravya
aaa

if we want remove the data:-

```
import java.util.*;
class Test
{	public static void main(String[] args)
	{	ArrayList<String> al =new ArrayList<String>();
		al.add("ratan");
		al.add("anu");
		al.add("sravya");
		ListIterator<String> lstr = al.listIterator();
		while(lstr.hasNext())
		{	if ((lstr.next()).equals("ratan"))
			{	lstr.remove();
			}
		}
		for (String str:al)
		{	System.out.println(str);
		}
	}
}
```

E:\>java Test
anu
sravya

Example:-printing data in forward and backward directions.

```
import java.util.*;
class Test
{	public static void main(String[] args)
	{	ArrayList<String> al =new ArrayList<String>();
		al.add("ratan");
		al.add("anu");
		al.add("sravya");
		ListIterator<String> lstr = al.listIterator();
		System.out.println("printing data forward direction");
		while(lstr.hasNext())
		{	System.out.println(lstr.next());
		}
		System.out.println("printing data backward direction");
		while(lstr.hasPrevious())
		{	System.out.println(lstr.previous());
		}
	}

}
```