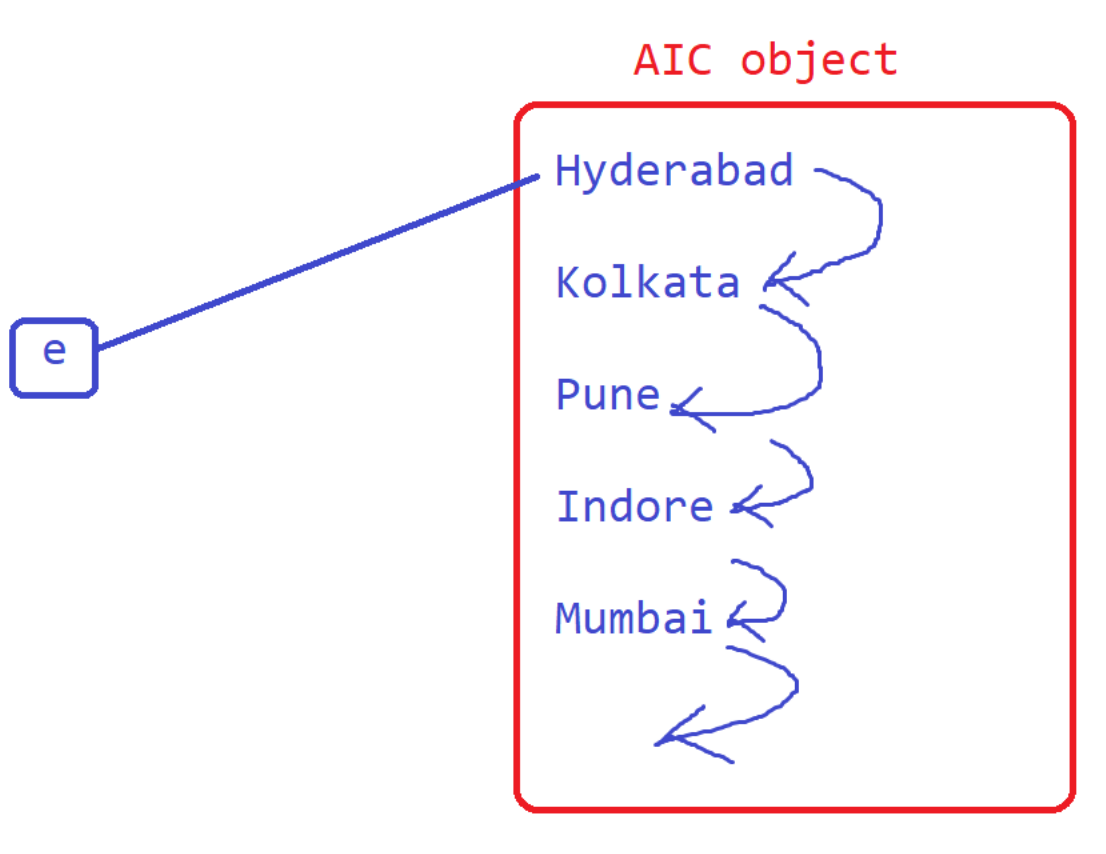
Enumeration<E> interface :
----
* It is a predefined legacy interface available in java.util package from JDK 1.0V.
* It is a cursor through which we can read collection object one by one.
* It works with legacy classes only.

```java
Enumeration<String> e = listOfCity.elements();

while(e.hasMoreElements())
{
    System.out.println(e.nextElement());
}
```



Methods :
----
public Enumeration elements()

public boolean hasMoreElements()

public E nextElement()

We can use Enumeration interface to fetch or retrieve the Objects one by one from the Collection because it is a cursor.

We can create Enumeration object by using elements() method of the legacy Collection class. Internally it uses anonymous inner class object.

public Enumeration elements();

Enumeration interface contains two methods :
----
1) public boolean hasMoreElements() :- It will return true if the collection object is available in the current position.

2) public E nextElement() :- It will return collection object so return type is Object(E) and move the cursor to the next line.

Note :- It will only work with legacy Collections classes.

Iterator<E> :

It is a predefined interface available in java.util package available from 1.2 version.

It is used to fetch/retrieve the elements from the Collection in forward direction only because it is also a cursor.

It is using private inner class i.e Itr class implements from Iterator<E>

public Iterator iterator();

Example :

Iterator itr = listOfCity.iterator();

Now, Iterator interface has provided two methods

public boolean hasNext() :-

It will verify, the element is available in the current position or not, if available it will return true otherwise it will return false.

public Object next() :- It will return the collection object and move the cursor to the element object.

Note : Iterator interafce has provided remove() method to solve the
issue of java.util.ConcurrentModificationException.

ListIterator<E> interface :

It is a predefined interface available in java.util package and it is the sub interface of Iterator available from JDK 1.2v.

It is used to retrieve the Collection object in both the direction i.e in forward direction as well as in backward direction. Here the inner class name is ListItr class extends from Itr class implements from ListIteartor interface

public ListIterator listIterator();

Example :

ListIterator lit = listOfCity.listIterator();

1) public boolean hasNext() :-
It will verify the element is available in the next position or not, if available it will return true otherwise it will return false.

2) public Object next() :- It will return the next position collection object.

3) public boolean hasPrevious() :-
It will verify the element is available in the previous position or not, if available it will return true otherwise it will return false.

4) public Object previous () :- It will return the previous position collection object.

Note : It works on the basis of index.

Note :- Apart from these 4 methods we have add(), set() and remove() method in ListIterartor interface.

Spliterator interface :

It is a predefined interface available in java.util package from java 1.8 version.

It is a cursor through which we can fetch the elements from the Collection [Collection, array, Stream]

***It is the combination of hasNext() and next() method.**

It is using forEachRemaining(Consumer<T> cons) method for fetching the elements.

Internally It uses do while loop

How forEach(Consumer<T>) method works internally ?
----
forEach() method is available from JDK 1.8V.

This forEach() method is used iterate the elements from the source.

The forEach(Consumer<T> cons) method is available in java.lang.Iterable interface

forEach(Consumer<T> cons) method of Iterable interface internally uses for each loop.

Case 1 :

```java
package com.ravi.introduction;

import java.util.Vector;
import java.util.function.Consumer;

public class ForEachInternalDemo1 {

    public static void main(String[] args)
    {
        Vector<String> listOfCity = new Vector<>();

            listOfCity.addElement("Hyderabad");
            listOfCity.add("Kolkata");
            listOfCity.add("Pune");
            listOfCity.add("Indore");
            listOfCity.add("Mumbai");

            //Anonymous inner class
            Consumer<String> cons = new Consumer<String>()
            {
                @Override
                public void accept(String city)
                {
                    System.out.println(city.toUpperCase());
                }
            };

            listOfCity.forEach(cons);

    }

}
```

Case 2 :

```java
package com.ravi.introduction;

import java.util.Vector;
import java.util.function.Consumer;

public class ForEachInternalDemo2 {

    public static void main(String[] args)
    {
        Vector<String> listOfCity = new Vector<>();

            listOfCity.addElement("Hyderabad");
            listOfCity.add("Kolkata");
            listOfCity.add("Pune");
            listOfCity.add("Indore");
            listOfCity.add("Mumbai");

            Consumer<String> cons = city -> System.out.println(city.toUpperCase());

            listOfCity.forEach(cons);
    }
}
```

Case 3 :

```java
package com.ravi.introduction;

import java.util.Vector;
import java.util.function.Consumer;

public class ForEachInternalDemo3
{
    public static void main(String[] args)
    {
        Vector<String> listOfCity = new Vector<>();

            listOfCity.addElement("Hyderabad");
            listOfCity.add("Kolkata");
            listOfCity.add("Pune");
            listOfCity.add("Indore");
            listOfCity.add("Mumbai");

            listOfCity.forEach(city -> System.out.println(city.toUpperCase()));

    }

}
```

stream() and parallelStream() method of Collection interface :
----
* In order to **process the collection object, Collection interface has provided the following two default methods from JDK 1.8**

1) default java.util.stream.Stream stream() : Used to convert the collection object into Stream for fast data processing with single thread application

2) default java.util.stream.Stream parallelStream() : Used to convert the collection object into Stream for fast data processing with multi-thread application

//Program that describes how to fetch Collection Object by using 11 ways :

```java
package com.ravi.introduction;

import java.util.Enumeration;
import java.util.Iterator;
import java.util.ListIterator;
import java.util.Spliterator;
import java.util.Vector;

public class RetrieveCollectionObject
{
    public static void main(String[] args)
    {
        Vector<String> listOfCity = new Vector<>();

           listOfCity.addElement("Hyderabad");
           listOfCity.add("Kolkata");
           listOfCity.add("Pune");
           listOfCity.add("Indore");
           listOfCity.add("Mumbai");

        System.out.println("1) By using toString() Method :");
        System.out.println(listOfCity.toString());

        System.out.println("______________________________");
        System.out.println("2) By using Ordinary for loop :");

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

        System.out.println("______________________________");
        System.out.println("3) By using  for each loop :");

        for(String city : listOfCity)
        {
              System.out.println(city);
        }

        System.out.println("______________________________");

        System.out.println("4) By using Enumeration interface :");

        Enumeration<String> elements = listOfCity.elements();

        while(elements.hasMoreElements())
        {
            System.out.println(elements.nextElement());
        }

        System.out.println("______________________________");
        System.out.println("5) By using Iterator interface :");

        Iterator<String> itr = listOfCity.iterator();
        itr.forEachRemaining(city -> System.out.println(city));

        System.out.println("______________________________");

        System.out.println("6) By using ListIterator interface :");


          ListIterator<String> listItr = listOfCity.listIterator();

          System.out.println("IN FORWARD DIRECTION");

          while(listItr.hasNext())
          {
              System.out.println(listItr.next());
          }

          System.out.println("IN BACKWARD DIRECTION");

          while(listItr.hasPrevious())
          {
              System.out.println(listItr.previous());
          }

            System.out.println("______________________________");

            System.out.println("7) By using Spliterator interface :");

            Spliterator<String> spliterator = listOfCity.spliterator();
            spliterator.forEachRemaining(city -> System.out.println(city));

             System.out.println("______________________________");

            System.out.println("8) By using For Each method :");
            listOfCity.forEach(city -> System.out.println(city));

            System.out.println("______________________________");
            System.out.println("9) By using Method Reference :");
            listOfCity.forEach(System.out::println);

            System.out.println("______________________________");
            System.out.println("10) By using stream() method of Collection :");
            listOfCity.stream().forEach(System.out::println);

            System.out.println("______________________________");
            System.out.println("11) By using parallelStream() method of Collection :");

            listOfCity.parallelStream().forEach(System.out::println);

    }

}
```