Java : Collection API

SUPPORTED BY JAVA.UTIL PACKAGE

Collection API

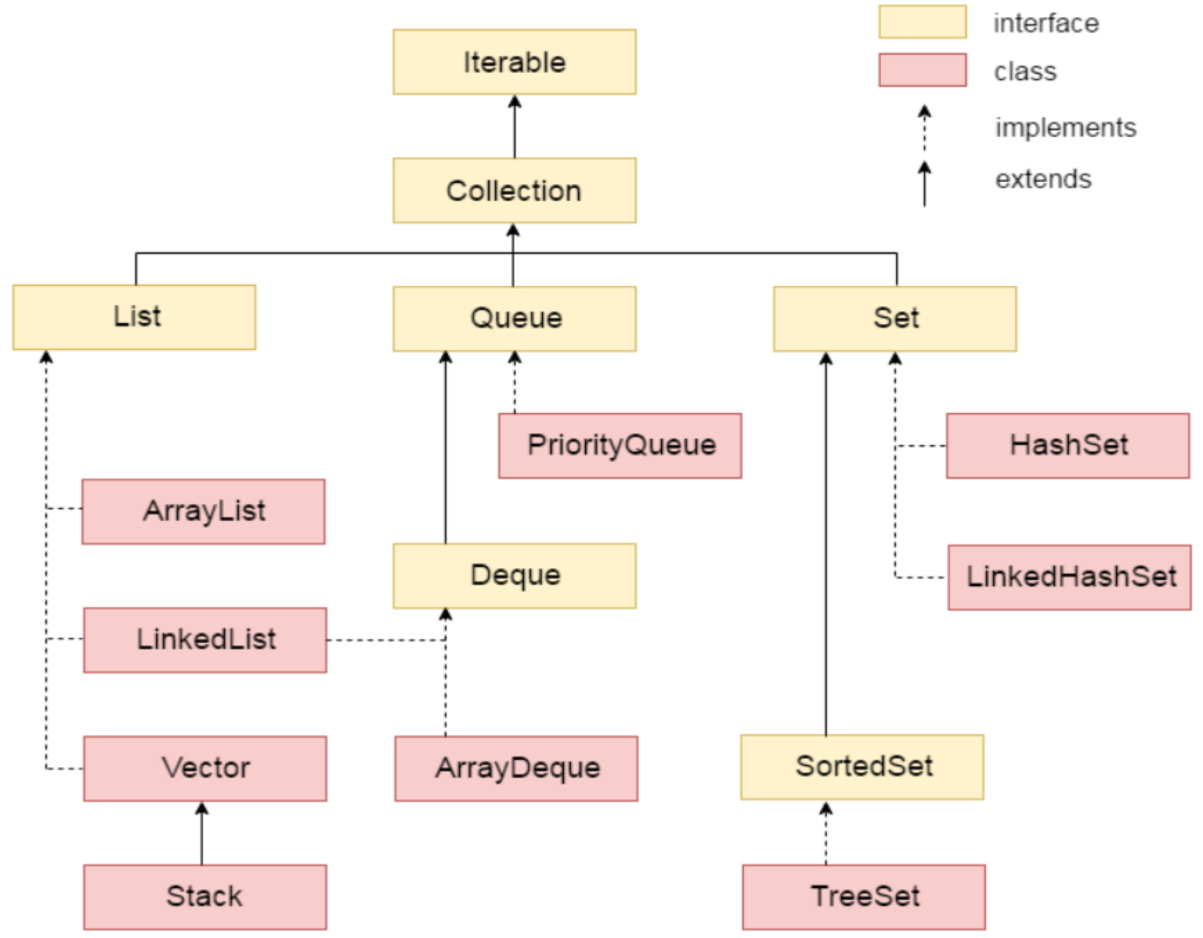
- Collections Framework provides a well-designed set of interfaces and classes for storing and manipulating groups of data as a single unit, a collection.

- It provides a convenient API to many of the ADTs like maps, sets, lists, trees, arrays, hash tables, and other collections.

- All the classes and interfaces of collection API are available in a **java.util** package.

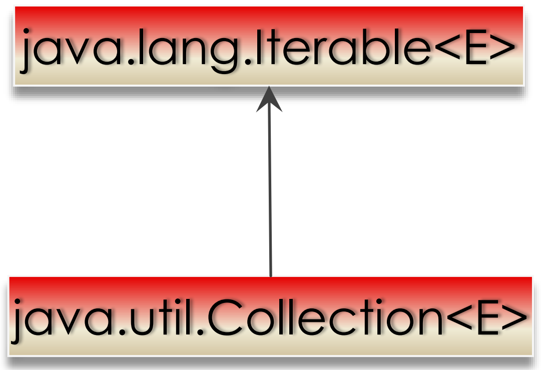
Collection Framework Hierarchy:

- The **java.util** package contains one of Java's most powerful subsystems: collections.
- Collections were added by the initial release of Java 2



java.util.Collection<E>

- Super Interface for all classes that define a collection.
- Defines the most fundamental behavior of every collection object.
- Because Collection extends Iterable all collections can be cycled through using for-each loop.



```
public interface java.lang.Iterable{
   public abstract java.util.Iterator iterator();
}

public interface java.util.Iterator{
   public abstract boolean hasNext();
   public abstract java.lang.Object next();
   public abstract void remove();
}
```

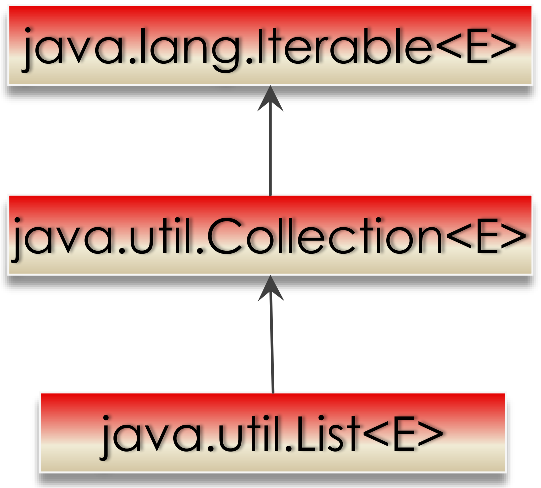
Collection Interface:

Abtsrtact Methods: defined in its implemented classes.

Declaration	Description
boolean add(Object obj)	Adds obj to the invoking collection. Returns true if obj was added to the collection. Returns false otherwise.
boolean addAll(Collection c)	Adds all the elements of c to the invoking collection. Returns true if the operation succeeded. Otherwise, returns false.
void clear( )	Removes all elements from the invoking collection.
boolean contains(Object obj)	Returns true if obj is an element of the invoking collection. Otherwise, returns false.
boolean containsAll(Collection c)	Returns true if the invoking collection contains all elements of c. Otherwise, returns false.
boolean equals(Object obj)	Returns true if the invoking collection and obj are equal. Otherwise, returns false.
boolean isEmpty( )	Returns true if the invoking collection is empty. Otherwise, returns false.
Iterator iterator( )	Returns an iterator for the invoking collection.
boolean remove(Object obj)	Removes one instance of obj from the invoking collection. Returns true if the element was removed. Otherwise, returns false.
boolean removeAll(Collection c)	Removes all elements of c from the invoking collection. Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false.
boolean retainAll(Collection c)	Removes all elements from the invoking collection except those in c. Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false.
int size( )	Returns the number of elements held in the invoking collection.
Object[ ] toArray( )	Returns an array that contains all the elements stored in the invoking collection.
Object[ ] toArray(Object array[ ])	Returns an array containing only those elements whose type matches that of array.
int hashCode( )	Returns the hash code for the invoking collection.

java.util.List<E>

- Declares the behavior of a collection that stores a sequence of elements.
- Elements can be inserted or accessed by their position in the list.
- It may have duplicate elements & null also.



void add(int,E)
boolean addAll(int, Collection<? extends E>)
E get(int)
int indexOf(Object)
int lastIndexOf(Object)

returns -1 if the element isn't in the list

java.util.List<E>

- ListIterator<E> listIterator()
- ListIterator<E> listIterator(int index)

  Iterator starting from index.
- E remove(int)
- E set(int,E)
- List<E> subList(int start,int end)

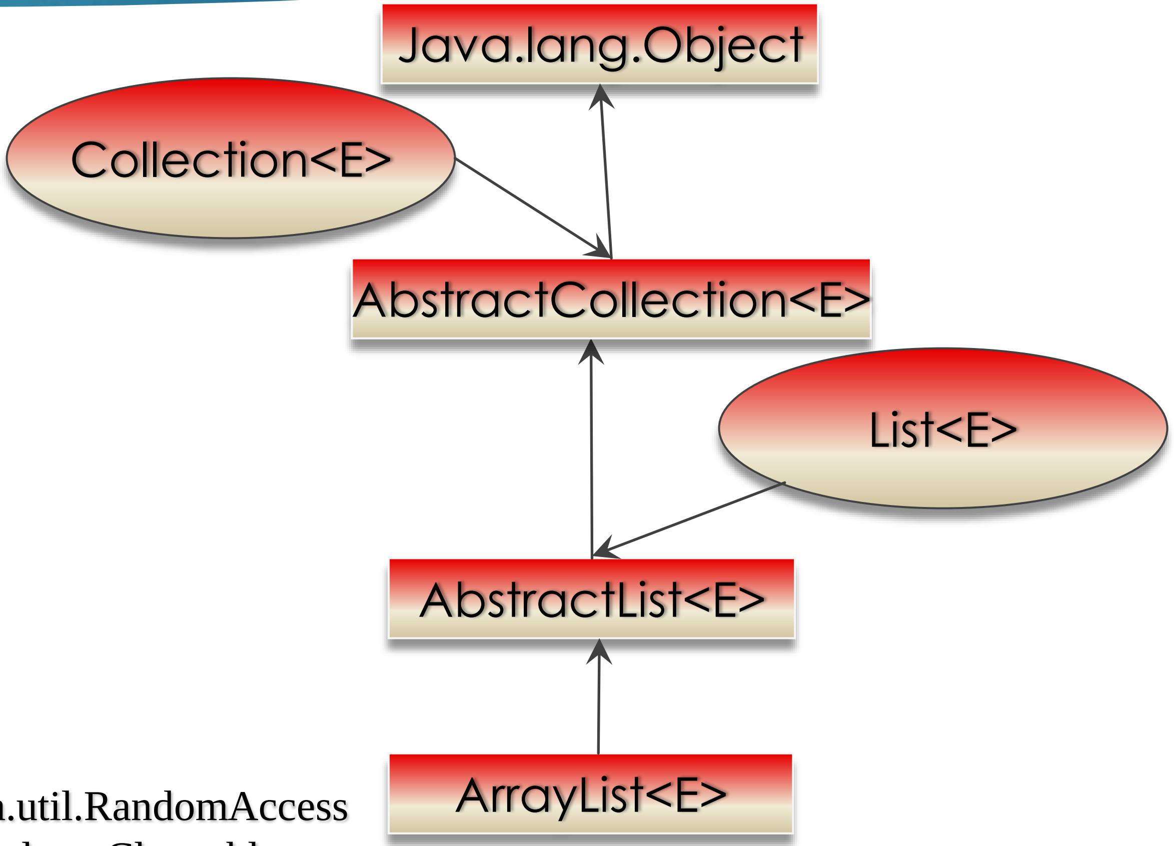
java.util.ListIterator<E>

java.util.ListIterator<E> extends java.util.Iterator<E>

- 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);
- public abstract void add(E);

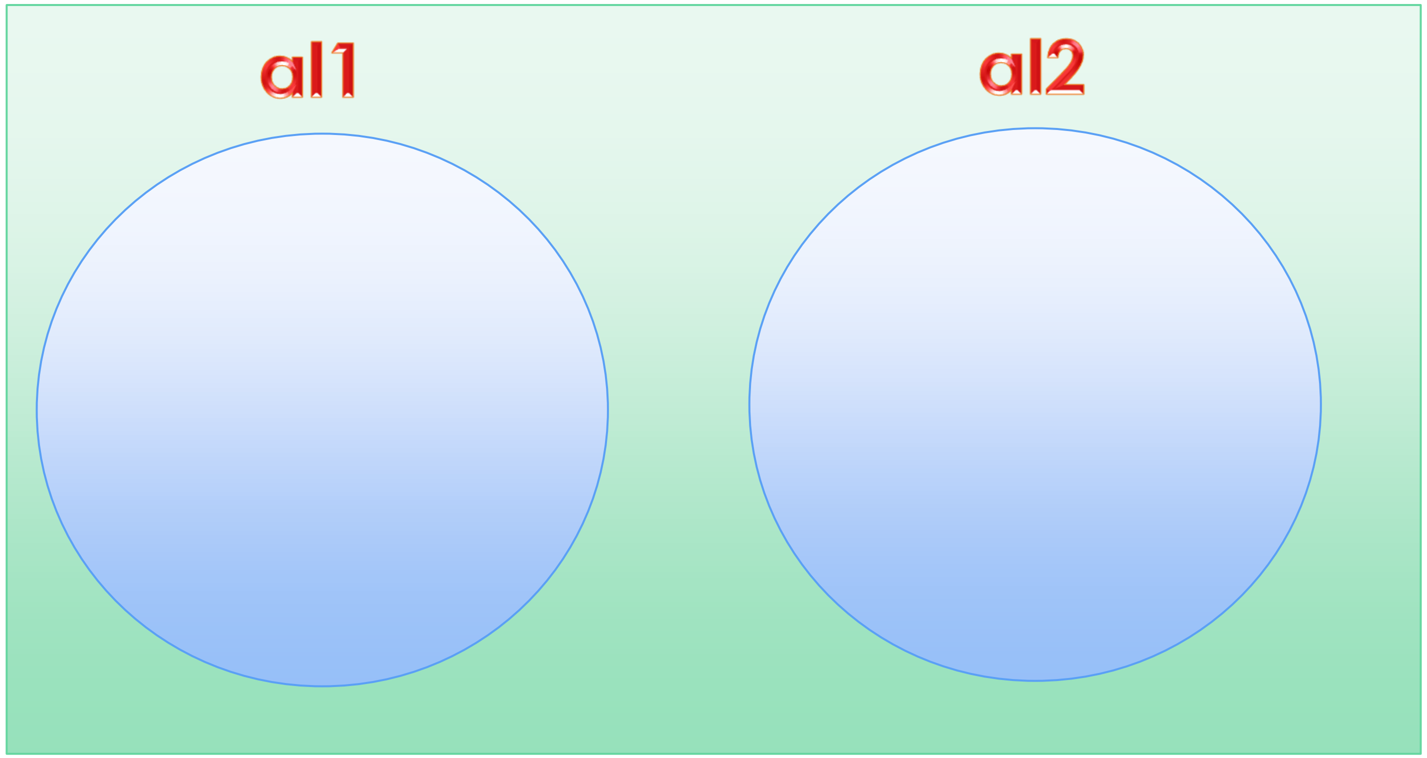
java.util.ArrayList<E>

- Java ArrayList class uses a dynamic array for storing the elements.
- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because array works at the index basis.



Other Interfaces Implemented by ArrayList : java.util.RandomAccess
java.lang.Cloneable
java.io.Serializable

Collection Methods:



CODE :

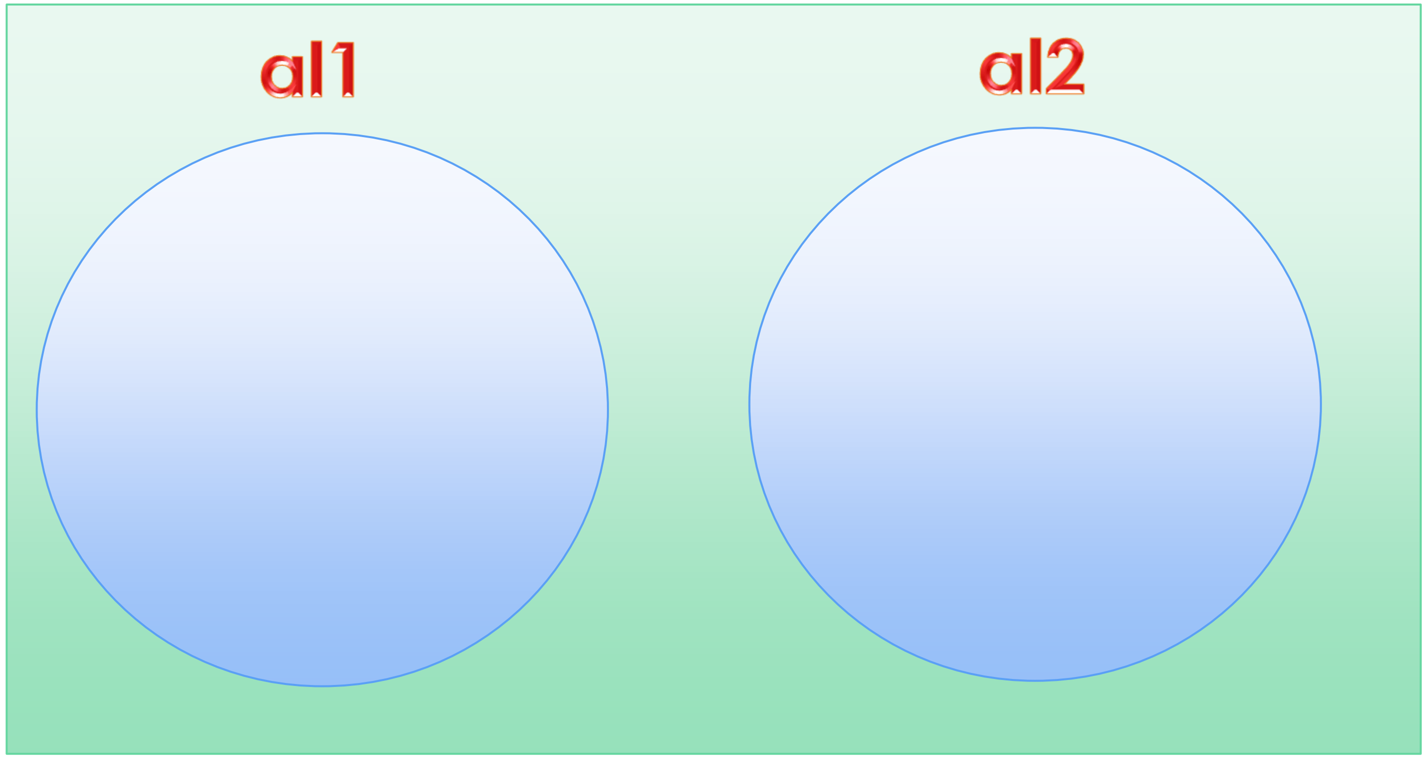
```
List al1 = new ArrayList( );
List al2 = new ArrayList( );

    al1.isEmpty( )
```

OUTPUT :

```
    true
```

Collection Methods:



CODE :

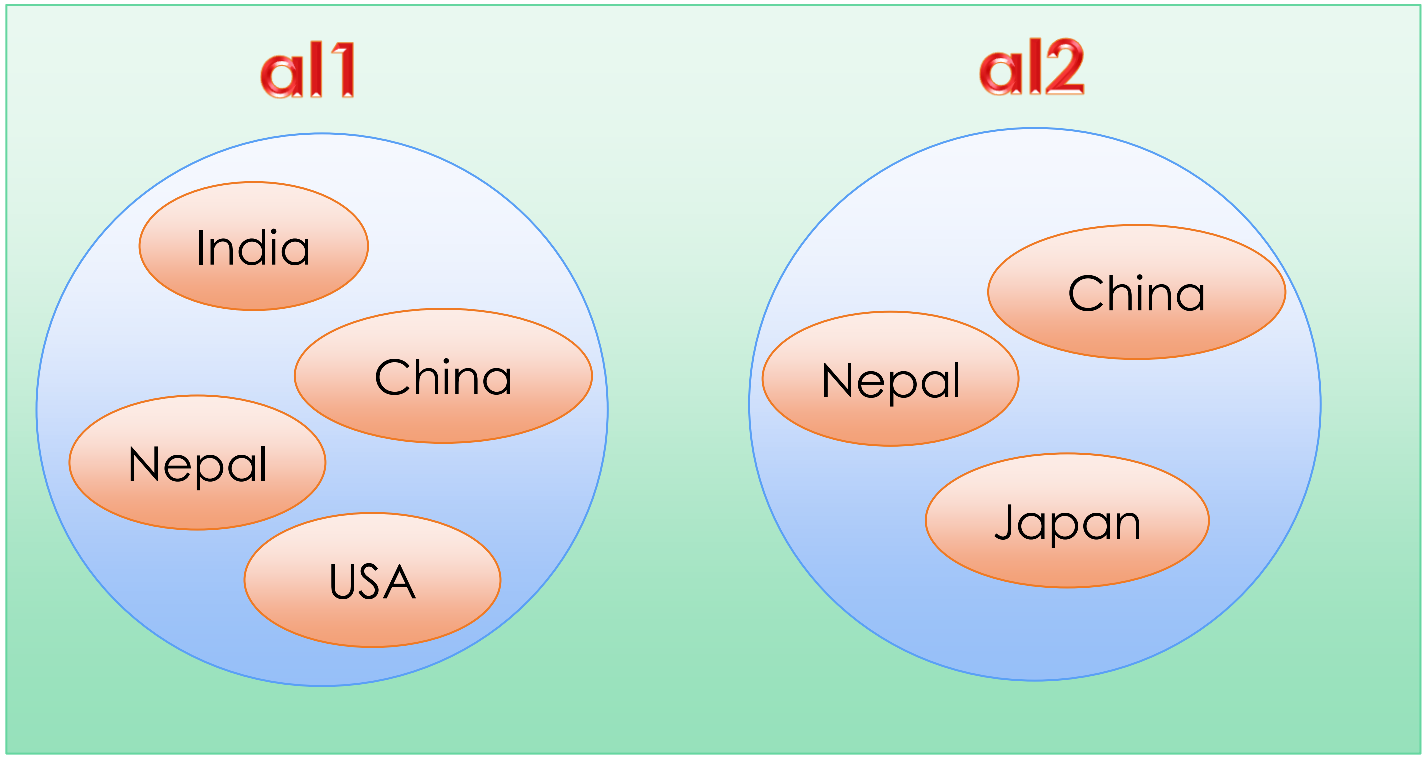
```
List al1 = new ArrayList( );
List al2 = new ArrayList( );

    al1.size( )
```

OUTPUT :

```
    0
```

Collection Methods:



CODE :

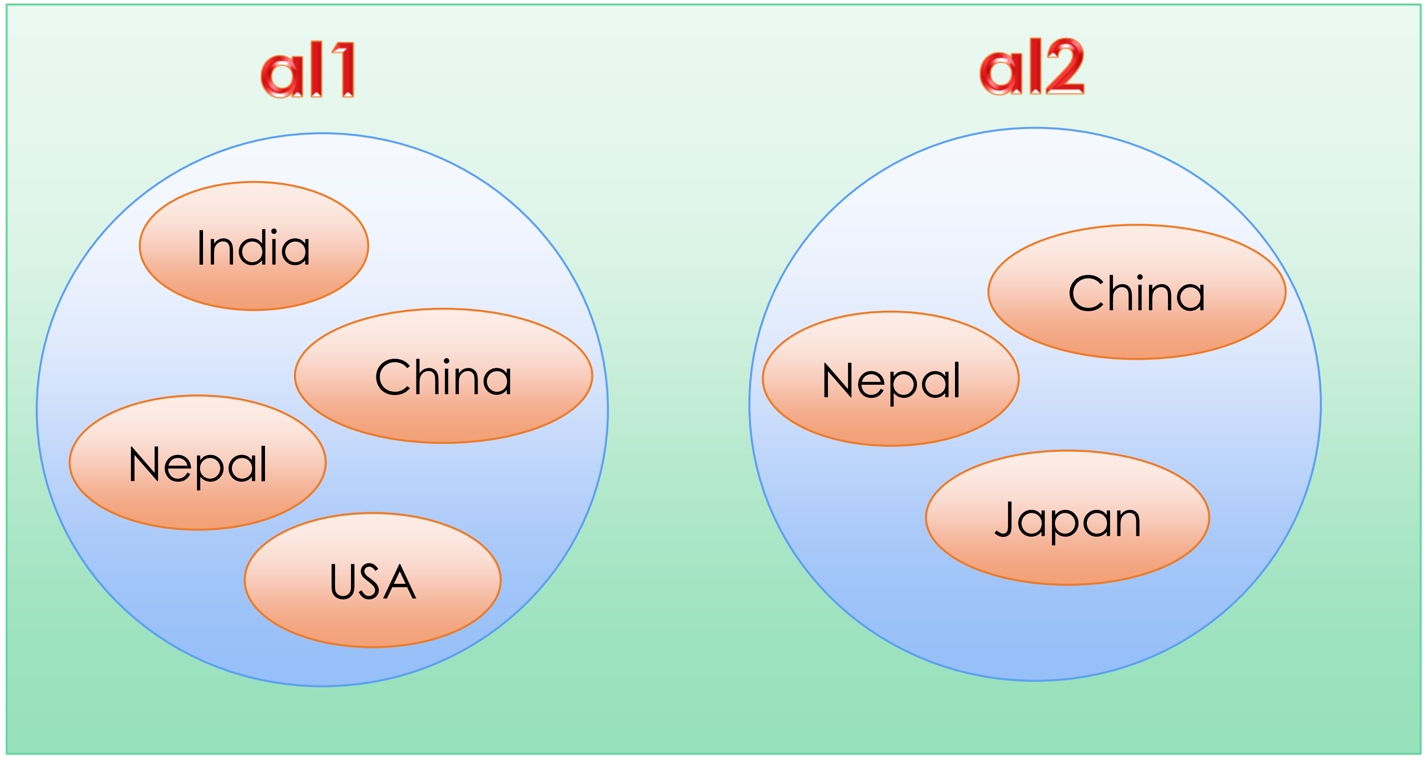
```
List al1 = new ArrayList( );
List al2 = new ArrayList( );

    al1.add("India" )
```

RETURN VALUE :

```
true
```

Collection Methods:



CODE :

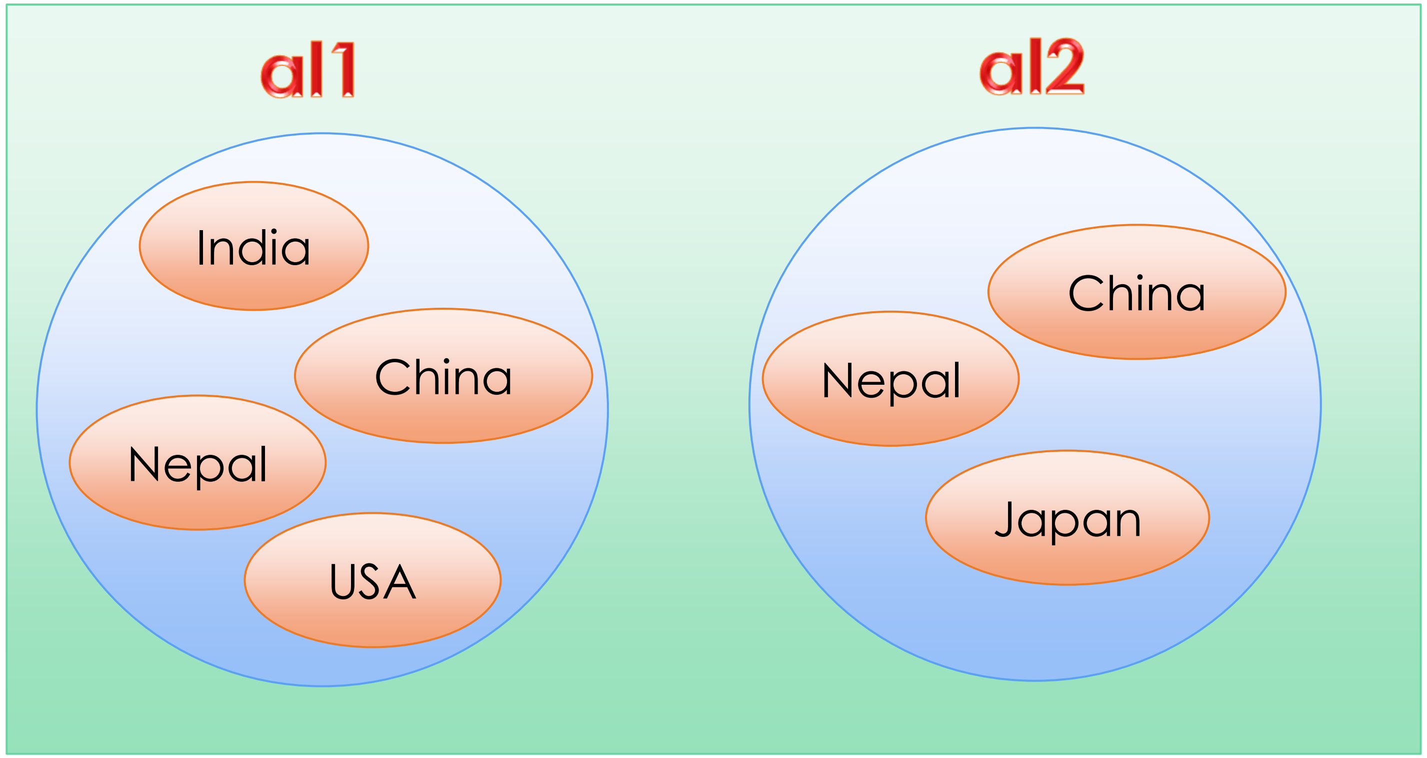
List al1 = new ArrayList( );
List al2 = new ArrayList( );

al1.remove("Japan")

RETURN VALUE :

false

Collection Methods:



CODE :

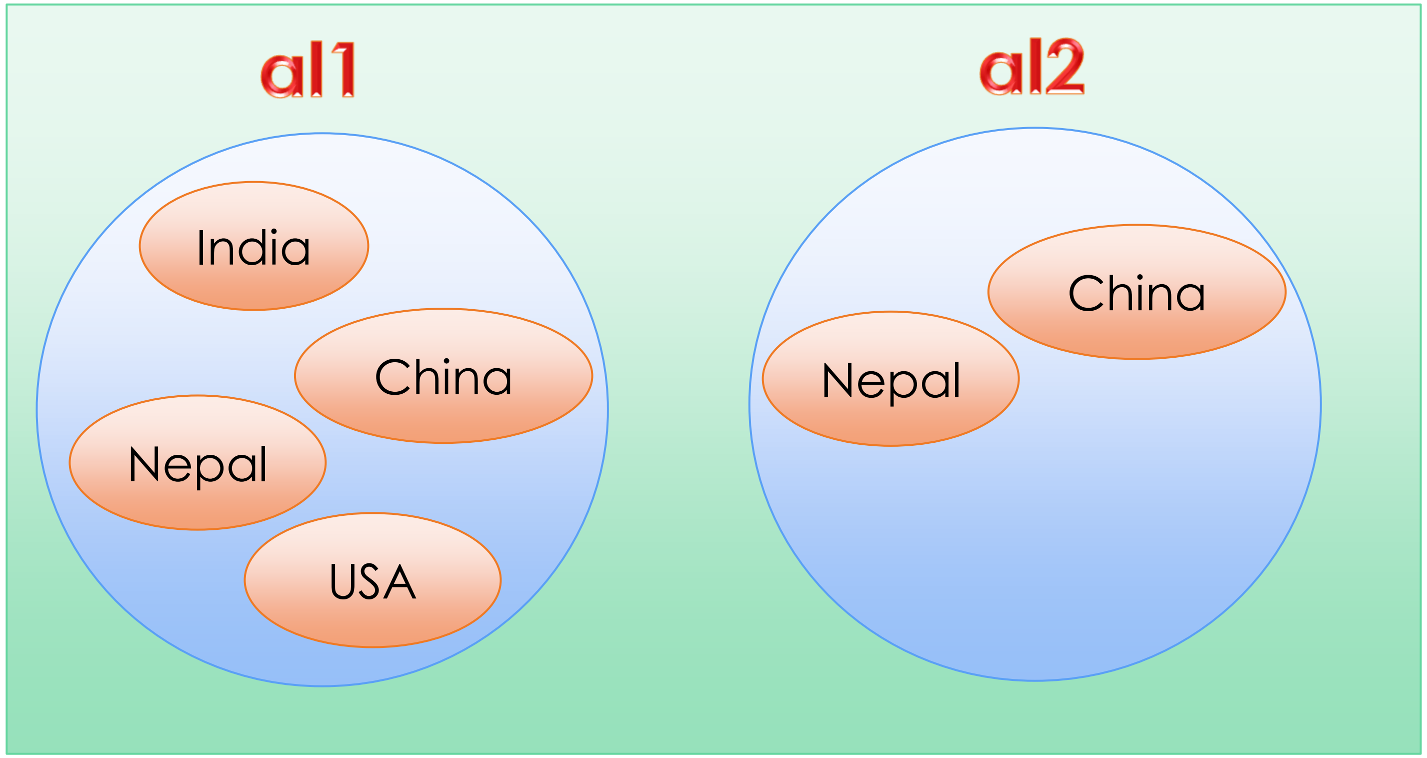
List al1 = new ArrayList( );
List al2 = new ArrayList( );

al2.remove("Japan")

RETURN VALUE :

true

Collection Methods:



CODE :

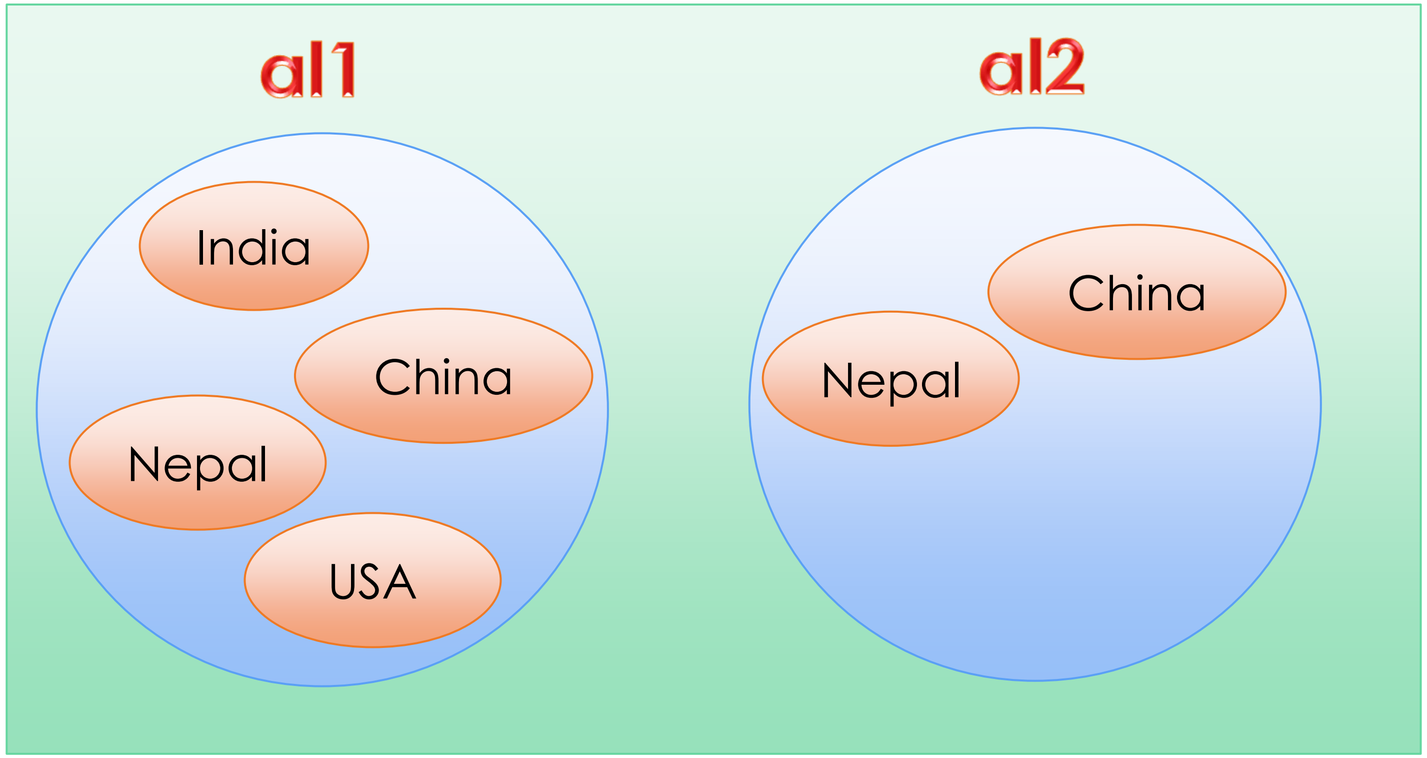
```
List al1 = new ArrayList( );
List al2 = new ArrayList( );

  al1.contains("India" )
  al2.contains("India" )
```

RETURN VALUE :

true
false

Collection Methods: Bulk Operation



CODE :

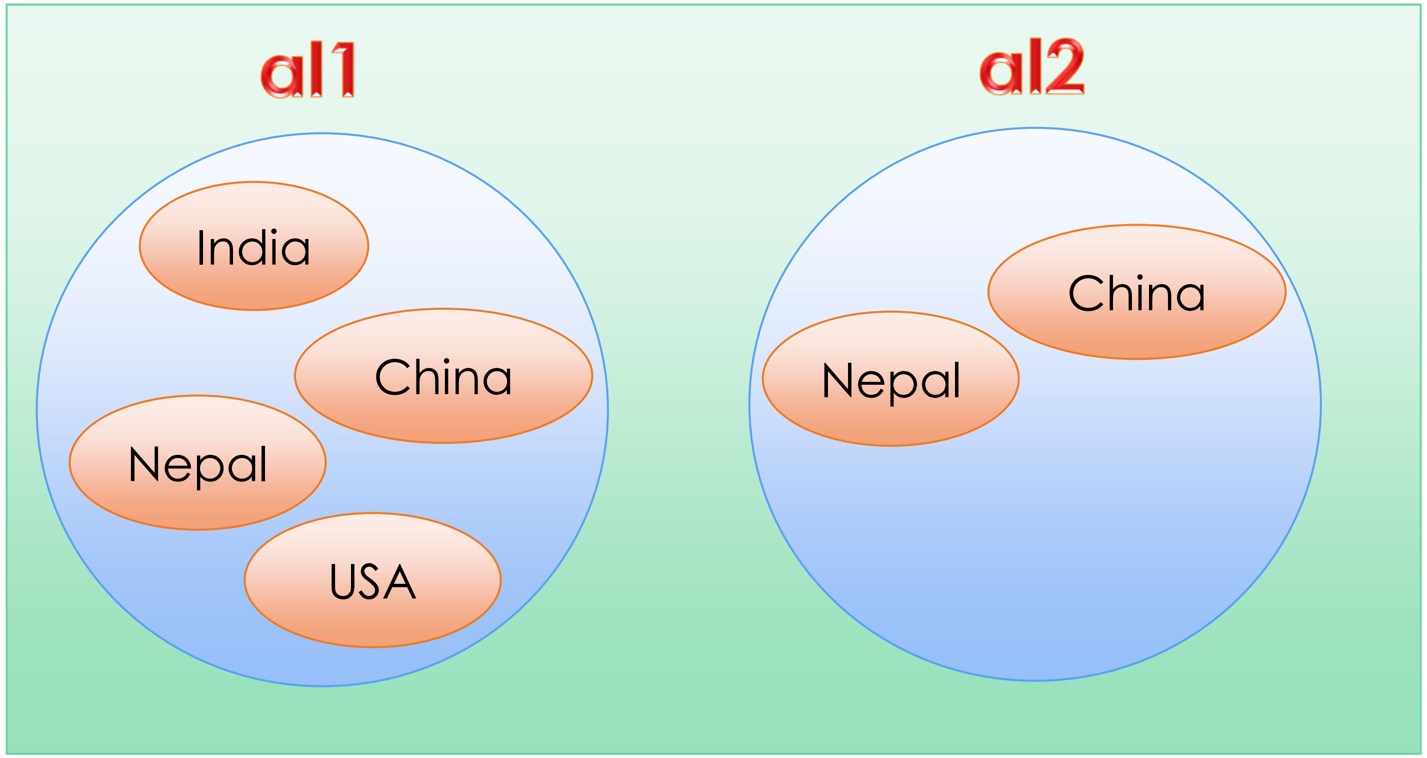
```
List al1 = new ArrayList( );
List al2 = new ArrayList( );

  al1.containsAll( al2 )
```

RETURN VALUE :

true

Collection Methods: Bulk Operation



CODE :

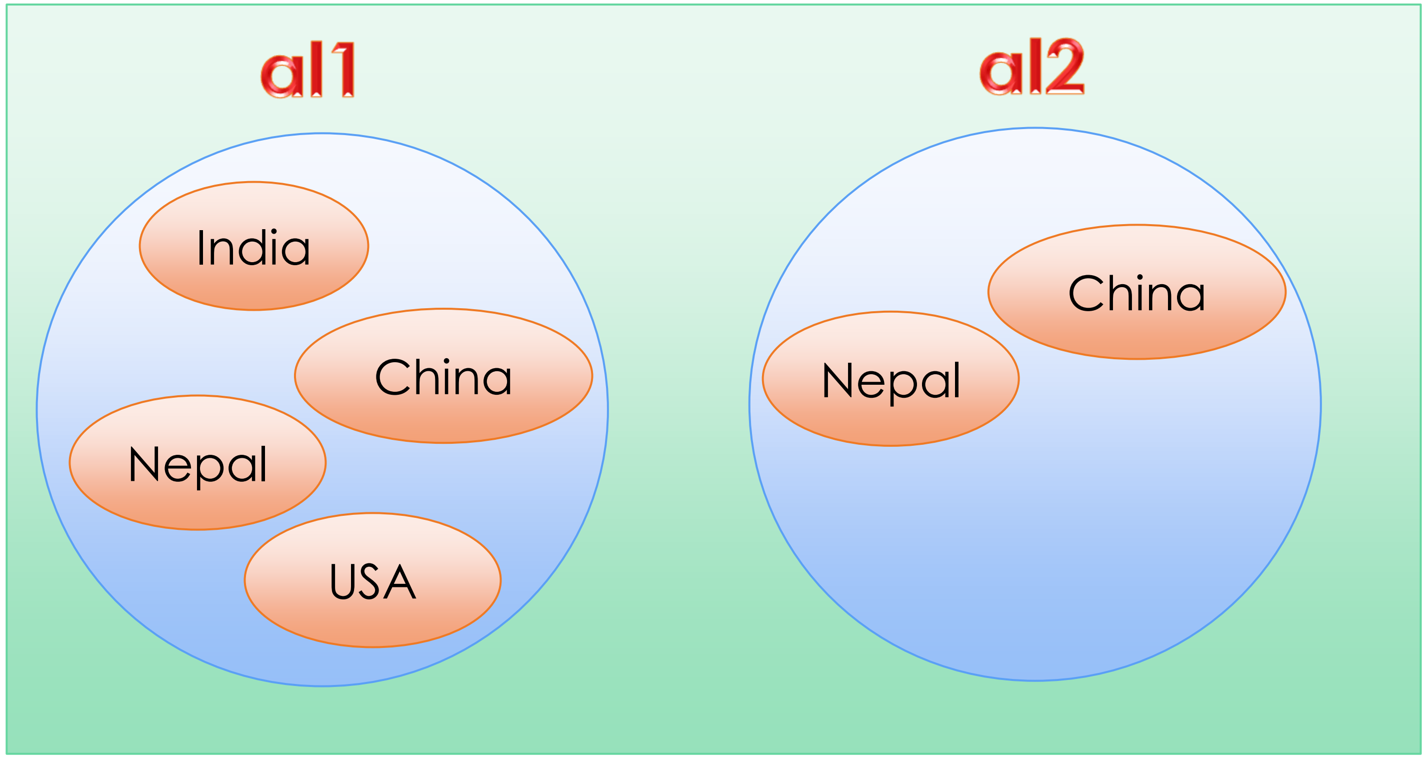
```
List al1 = new ArrayList( );
List al2 = new ArrayList( );

  al1.removeAll( al2 )
```

RETURN VALUE :

true

Collection Methods: Bulk Operation



CODE :

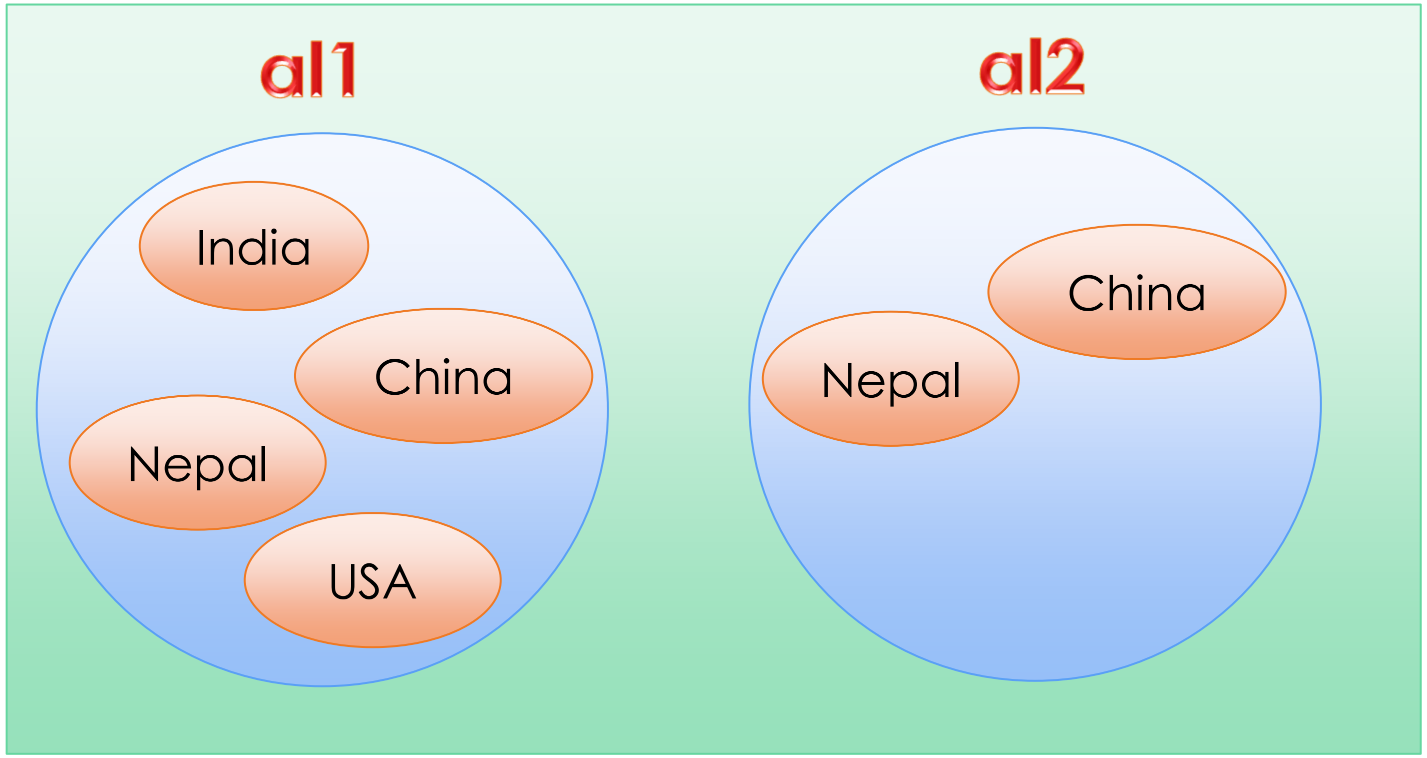
```
List al1 = new ArrayList( );
List al2 = new ArrayList( );

  al1.retainAll( al2 )
```

RETURN VALUE :

true

Collection Methods: Bulk Operation



CODE :

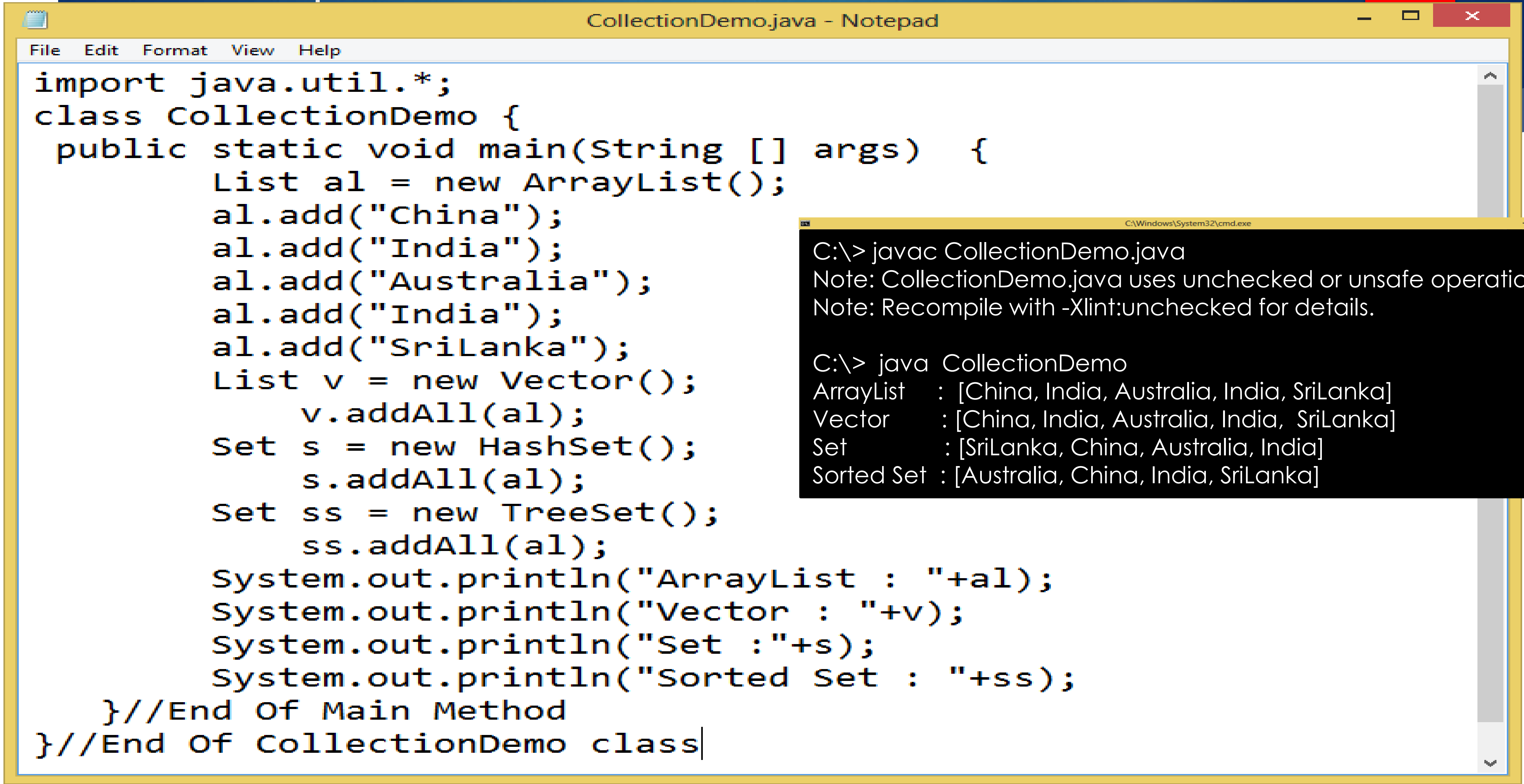
```
List al1 = new ArrayList( );
List al2 = new ArrayList( );

  al1.clear( );
  al2.clear( );
```

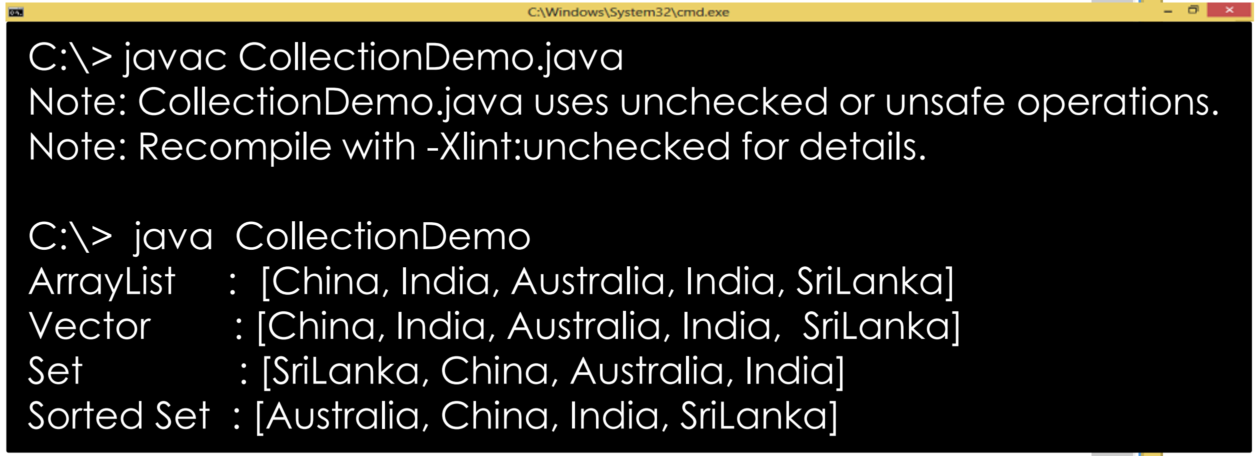
RETURN VALUE :

```
    true
    true
```

Example:


CollectionDemo.java - Notepad

```java
import java.util.*;
class CollectionDemo {
 public static void main(String [] args)   {
        List al = new ArrayList();
        al.add("China");
        al.add("India");
        al.add("Australia");
        al.add("India");
        al.add("SriLanka");
        List v = new Vector();
            v.addAll(al);
        Set s = new HashSet();
            s.addAll(al);
        Set ss = new TreeSet();
            ss.addAll(al);
        System.out.println("ArrayList : "+al);
        System.out.println("Vector : "+v);
        System.out.println("Set :"+s);
        System.out.println("Sorted Set : "+ss);
   }//End Of Main Method
}//End Of CollectionDemo class
```

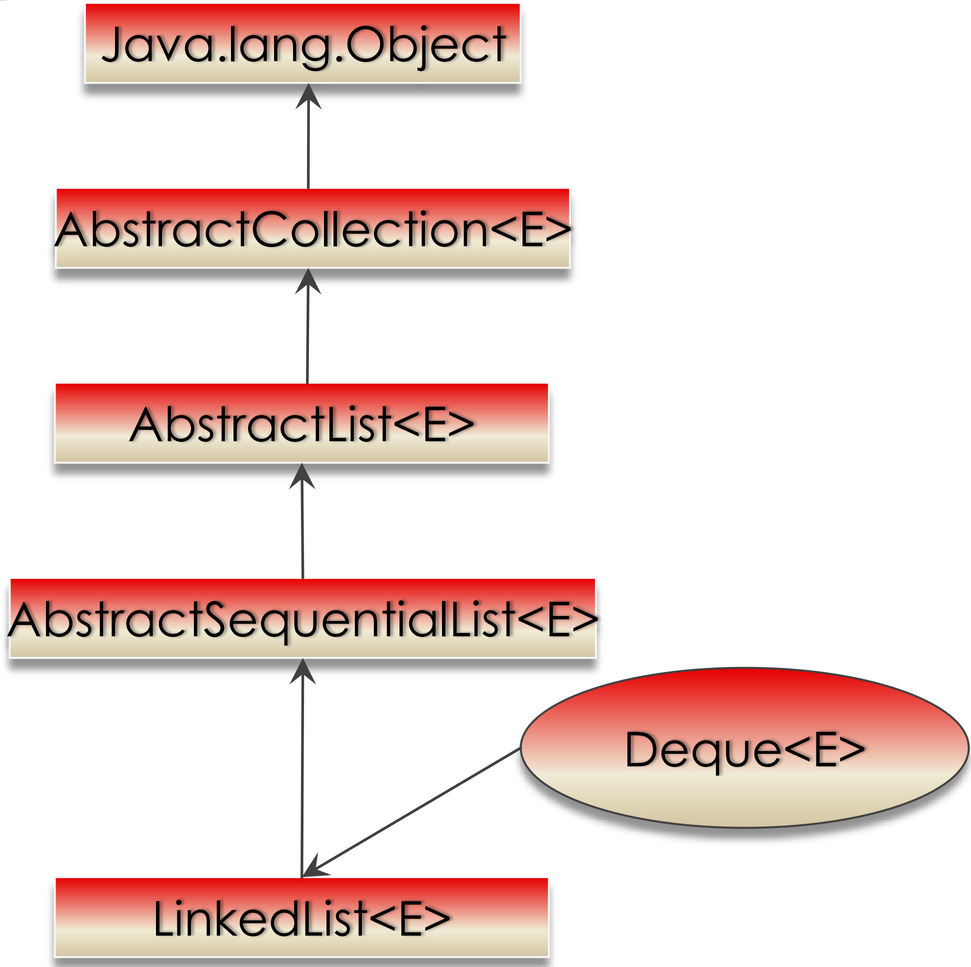

C:\Windows\System32\cmd.exe

```
C:\> javac CollectionDemo.java
Note: CollectionDemo.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\>  java  CollectionDemo
ArrayList    :  [China, India, Australia, India, SriLanka]
Vector        : [China, India, Australia, India,  SriLanka]
Set             : [SriLanka, China, Australia, India]
Sorted Set  : [Australia, China, India, SriLanka]
```

java.util.LinkedList<E>

- Java LinkedList class uses doubly linked list to store the elements. It provides a linked-list data structure.
- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- Java LinkedList class can be used as list, stack or queue.



Queue<E> extends Collection<E>

Declares the behavior of a FIFO List.

Methods declared by Queue<E>:-

- E element( )

  Returns the head of the queue. Element isn't removed, throws NoSuchElementException if the queue is empty.

- boolean offer(E obj)

  Adds obj to the queue.

- E peek( )

  Works same as element. But doesn't throw exception if the list is empty. It returns null in that case.

- E poll( )

  Returns & removes the head of the queue. Returns null if the queue is empty.

- E remove( )

  Returns & removes the head of the queue. If the queue is empty, throws NoSuchElementException.

Deque<E> extends Queue<E>

Declares the behavior of a double-ended queue.

Can be used to implement both FIFO queue & LIFO stack at the same time.

A Deque implementation can be capacity-restricted.

- addFirst(E)

  Adds to the head of the deque. Throws IllegalStateException if deque capacity is reached.

- addLast(E)

  Adds to the head of the deque. Throws IllegalStateException if deque capacity is reached.

- Iterator<E> descendingIterator()

- E getFirst()

  Returns but doesn't remove the first element. NoSuchElementException is thrown if deque is empty.

- E getLast()

  Returns but doesn't remove the last element. NoSuchElementException is thrown if deque is empty.

Deque<E> extends Queue<E> contd.

- E pop()

  Returns & remove the head of the deque. Throws NoSuchElementException if deque is empty.
- boolean offerFirst(E)

  Adds to the head of the queue. Returns false if the capacity of deque is reached.
- boolean offerLast(E)

  Adds to the tail of the queue. Returns false if the capacity of deque is reached.
- E peekFirst( )

  Returns but doesn't remove the first element of the deque. Returns null if deque is empty.
- E peekLast( )
- E pollFirst( )

  Returns & remove the first element of the deque. Returns null if deque is empty.

Deque<E> extends Queue<E> contd.

- E pollLast( )
- void push(E)

  Adds to the head of the queue. Throws IllegalStateException if the capacity of deque is reached.
- E removeFirst()

  Returns & removes the head. NoSuchElementException is thrown if deque is empty.
- E removeLast()
- boolean removeFirstOccurrence(Object)

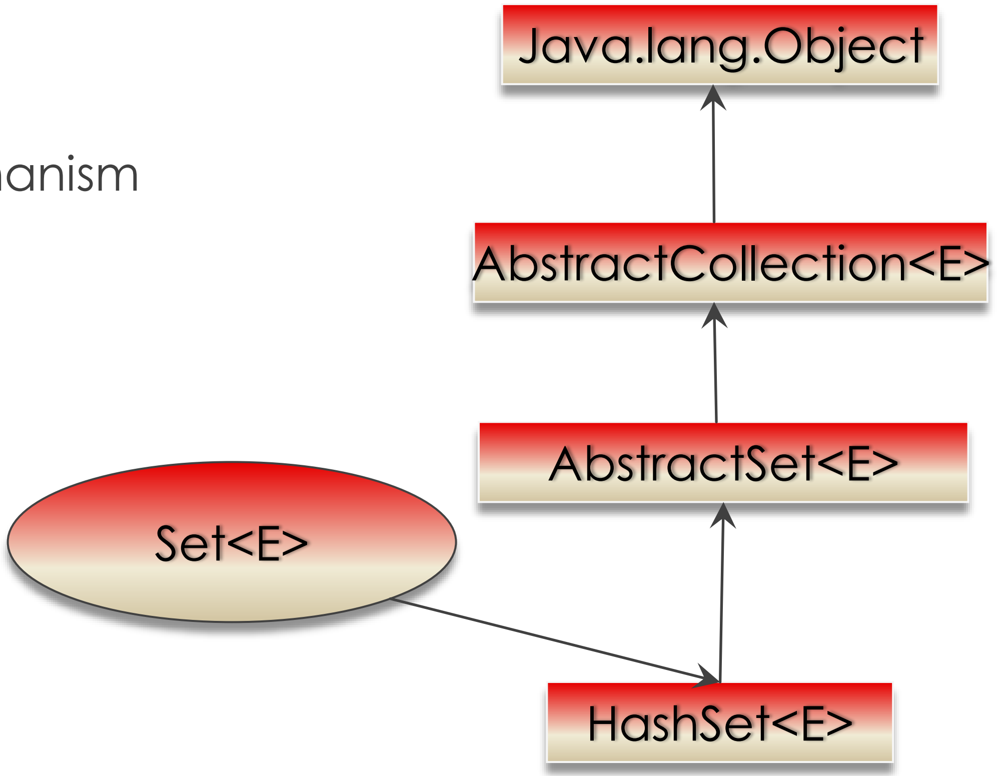
  Removes the first occurrence of the given object. Returns false if it's not a member of the deque.
- boolean removeLastOccurrence(Object)

java.util.Set<E>

- Declares the behavior of a Set. i.e., a Collection that doesn’t allow duplicate elements.
- add() returns false if we try to insert duplicate elements to the set.
- Doesn’t declare any method of its own.

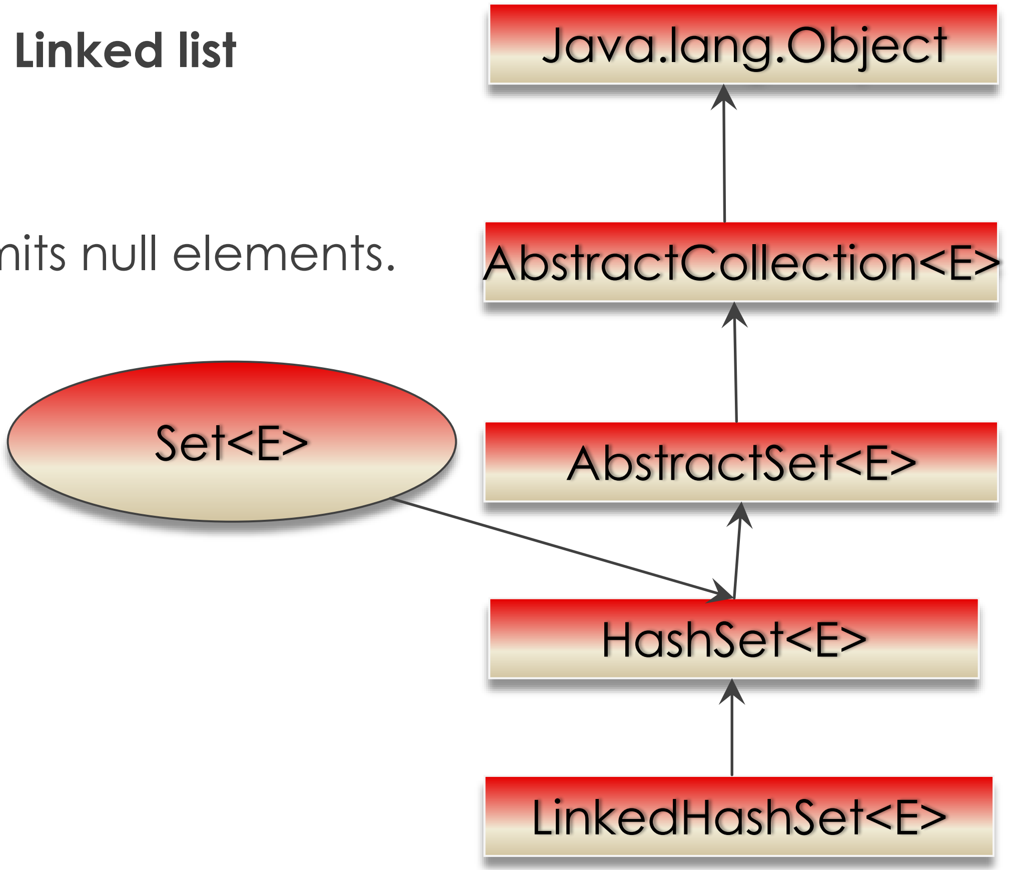
java.util.HashSet<E>

- Java HashSet class is used to create a collection that uses a hash table for storage.
- HashSet stores the elements by using a mechanism called **hashing.**
- HashSet contains unique elements only.



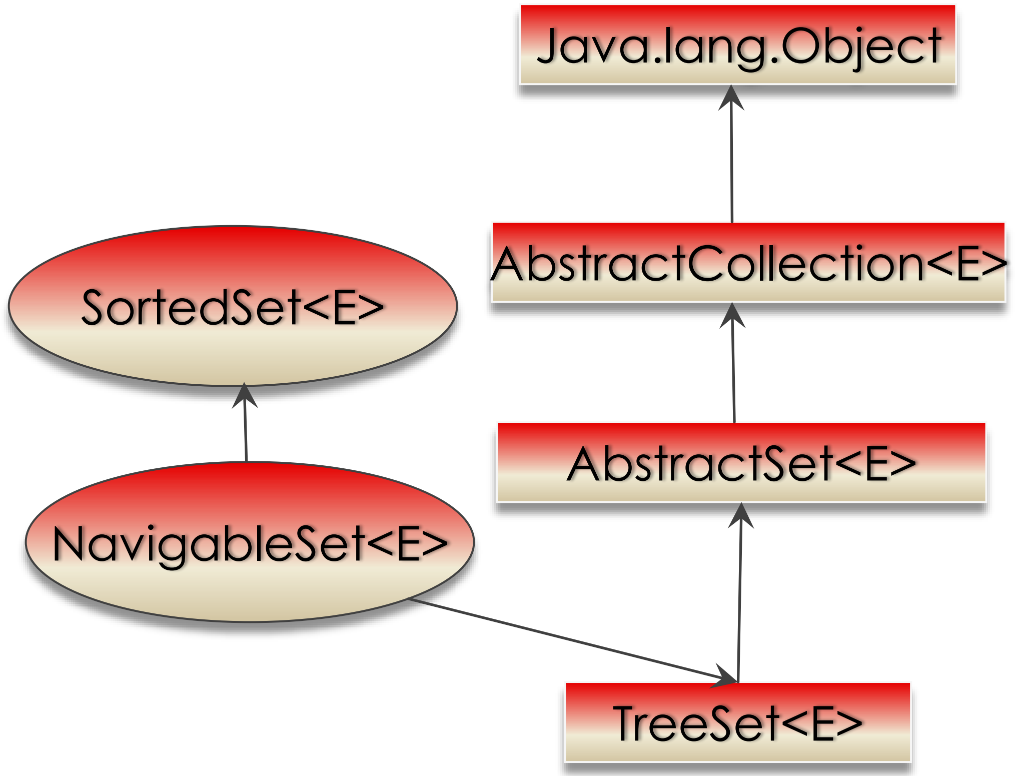
java.util.LinkedHashSet<E>

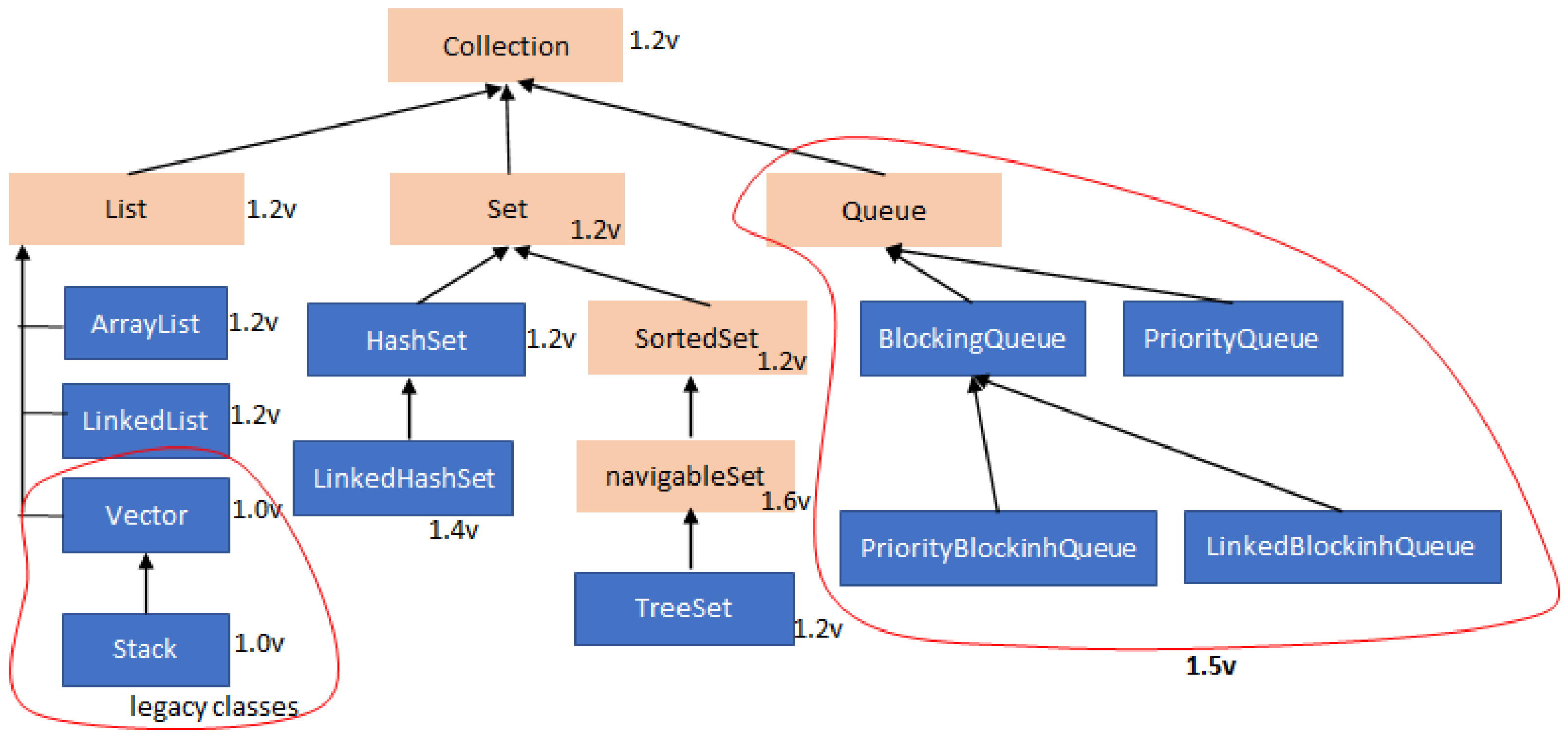
- Java LinkedHashSet class is a **Hash table and Linked list** implementation of the set interface.
- Provides all optional set operations, and permits null elements.
- Maintains insertion order.



java.util.TreeSet<E>

- Java TreeSet class implements the Set interface that uses a tree for storage.
- Contains unique elements only like HashSet.
- Access and retrieval times are quiet fast.
- Maintains ascending order.





Collection Framework

THANK YOU!!