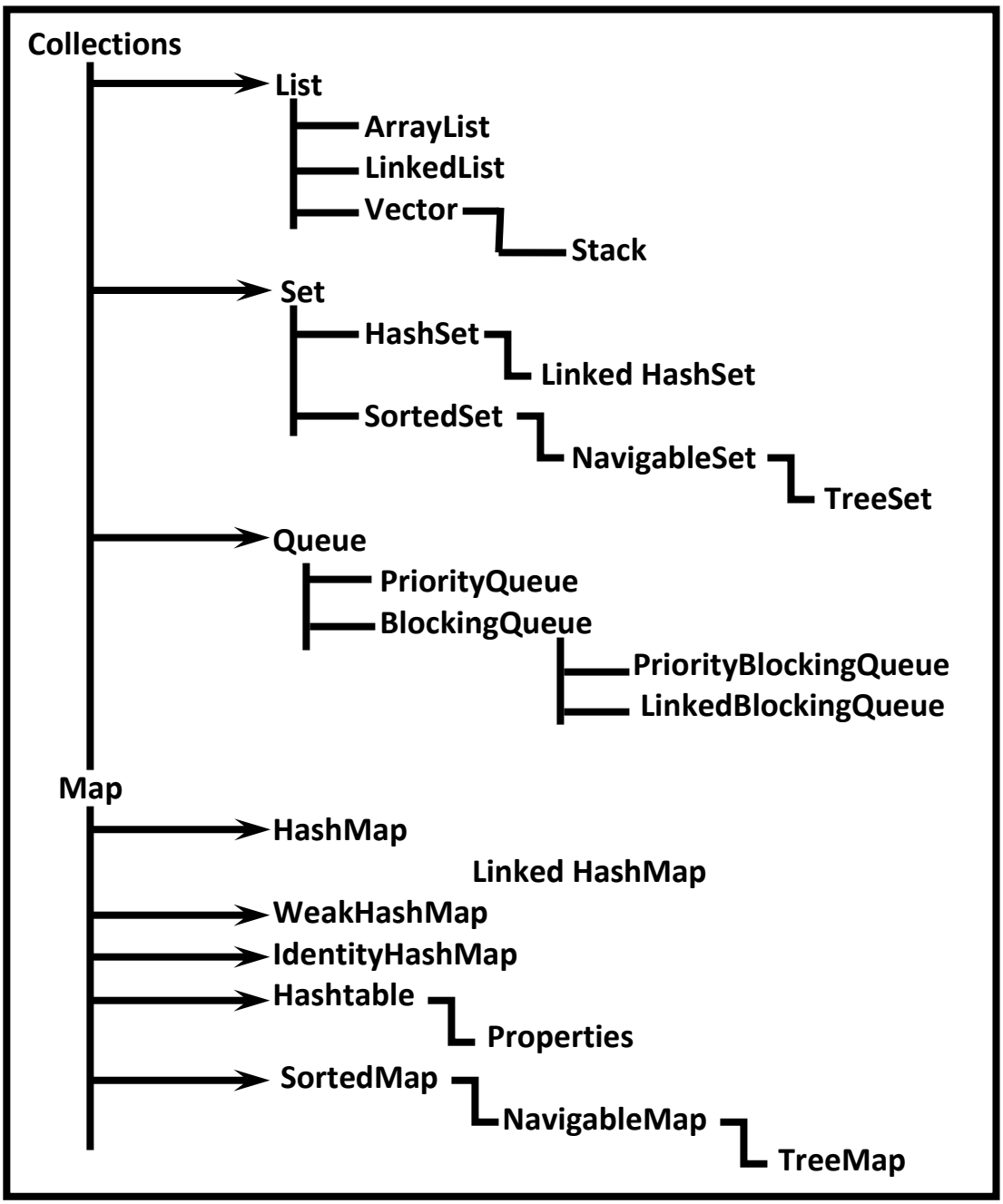
13. Collections Frame Work



Cursors

- Enumerations (I)
- Iterator (I)
- ListIterator (I)

Utility Classes

- Collections
- Arrays

Sorting

- Comparable (I)
- Comparator (I)

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

An Array is an Indexed Collection of Fixed Number of Homogeneous Data Elements. The Main Advantage of Arrays is we can Represent Multiple Values by using Single Variable so that Readability of the Code will be Improved.

Limitations Of Object Type Arrays:

1) Arrays are Fixed in Size that is Once we created an Array there is No Chance of Increasing OR Decreasing Size based on Our Requirement. Hence to Use Arrays Concept Compulsory we should Know the Size in Advance which May Not be Possible Always.

2) Arrays can Hold Only Homogeneous Data Type Elements.

Eg:

```
Student[] s = new Student[10000];
s[0] = new Student(); √

s[1]=new Customer(); ×
CE: incompatible types
     found: Costomer
     required: Student
```

We can Resolve this Problem by using Object Type Arrays.

Eg:

```
Object[] a = new Object[10000];
a[0] = new Student(); √
a[1] = new Customer(); √
```

3) Arrays Concept is Not implemented based on Some Standard Data Structure Hence Readymade Methods Support is Not Available. Hence for Every Requirement we have to write the Code Explicitly which Increases Complexity of the Programming.

To Overcome Above Problems of Arrays we should go for Collections.

Advantages Of Collections:

1) Collections are Growable in Nature. That is based on Our Requirement we can Increase OR Decrease the Size.
2) Collections can Hold Both Homogeneous and Heterogeneous Elements.
3) Every Collection Class is implemented based on Some Standard Data Structure. Hence for Every Requirement Readymade Method Support is Available. Being a Programmer we have to Use those Methods and we are Not Responsible to Provide Implementation.

Differences Between Arrays And Collections:

Arrays	*Collections*
Arrays are Fixed in Size.	Collections are Growable in Nature.
With Respect to Memory Arrays are Not Recommended to Use.	With Respect to Memory Collections are Recommended to Use.
With Respect to Performance Arrays are Recommended to Use.	With Respect to Performance Collections are Not Recommended to Use.
Arrays can Hold Only Homogeneous Data Elements.	Collections can Hold Both *Homogeneous* and *Heterogeneous* Elements.
Arrays can Hold Both Primitives and Objects.	Collections can Hold Only Objects but Not Primitives.
Arrays Concept is Not implemented based on Some Standard Data Structure. Hence Readymade Method Support is Not Available.	For every Collection class underlying Data Structure is Available Hence Readymade Method Support is Available for Every Requirement.

Collection:

If we want to Represent a Group of Individual Objects as a Single Entity then we should go for Collection.

Collection Frame Work:

It defines Several Classes and Interfaces which can be used to Represent a Group of Objects as a Single Entity.

JAVA	C++
Collection Collection Frame Work	Container Standard Template Library (STL)

9 Key Interfaces Of Collection Framework:

1) Collection (I)
2) List (I)
3) Set (I)
4) SortedSet (I)
5) NavigableSet (I)
6) Queue (I)
7) Map (I)
8) SortedMap (I)
9) NavigableMap (I)

1) Collection (I):

- If we want to Represent a Group of Individual Objects as a Single Entity then we should go for Collections.
- Collection Interface is considered as Root Interface of Collection Framework.
- Collection Interface defines the Most Common Methods which are Applicable for any Collection Object.

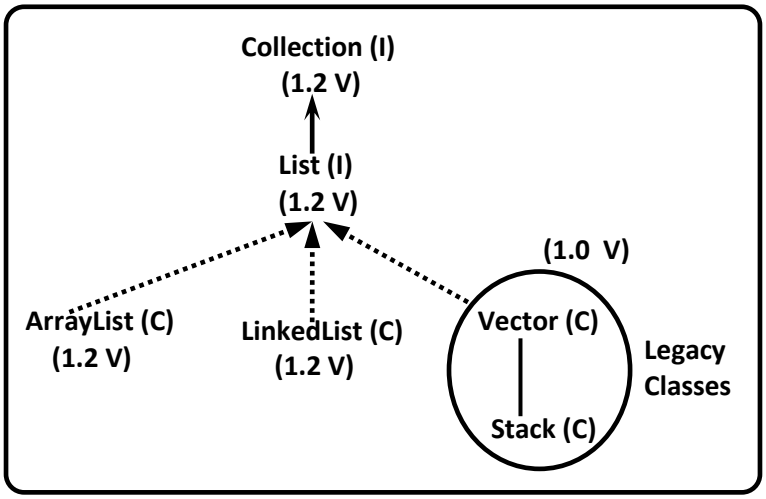
Difference Between Collection (I) and Collections (C):

- Collection is an Interface which can be used to Represent a Group of Individual Objects as a Single Entity.
- Whereas Collections is an Utility Class Present in *java.util* Package to Define Several Utility Methods for Collection Objects.

Note: There is No Concrete Class which implements Collection Interface Directly.

2) List (I):

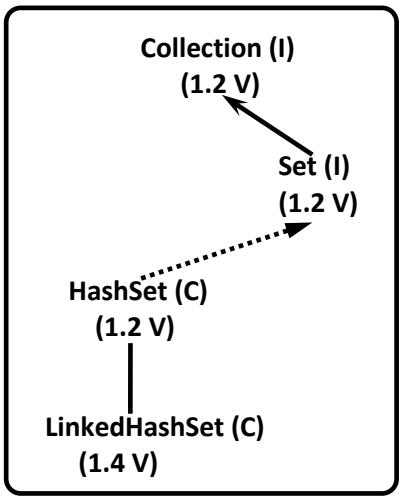
- It is the Child Interface of Collection.
- If we want to Represent a Group of Individual Objects as a Single Entity where Duplicates are allowed and Insertion Order Preserved. Then we should go for List.



Note: In 1.2 Version onwards *Vector* and *Stack* Classes are re-engineered to Implement List Interface.

3) Set (I):

- It is the Child Interface of the Collection.
- If we want to Represent a Group of Individual Objects as a Single Entity where Duplicates are Not allowed and Insertion Order won't be Preserved. Then we should go for Set Interface.

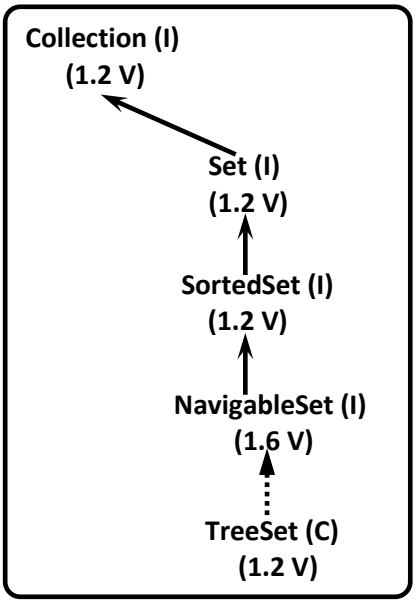


4) SortedSet (I):

- It is the Child Interface of Set.
- If we want to Represent a Group of Individual Objects Without Duplicates According to Some Sorting Order then we should go for SortedSet.

5) NavigableSet (I):

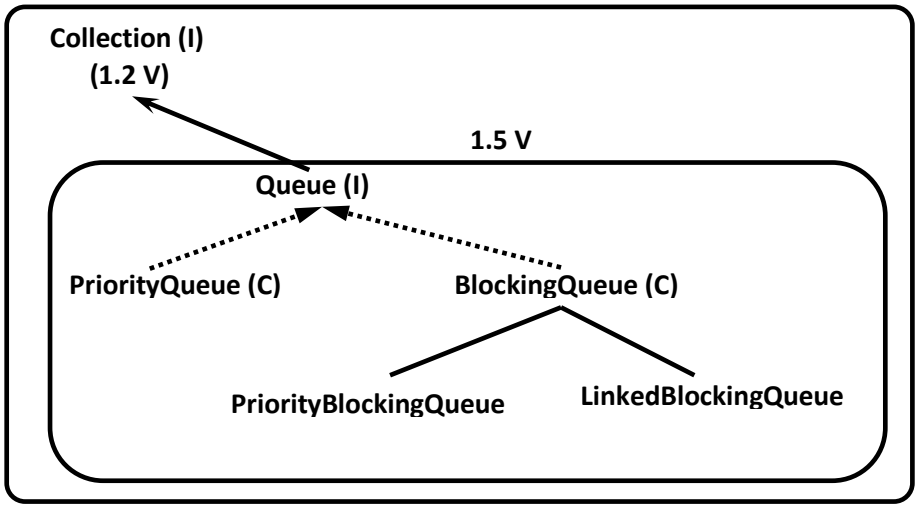
- It is the Child Interface of SortedSet.
- It defines Several Methods for Navigation Purposes.



6) Queue (I):

- It is the Child Interface of Collection.
- If we want to Represent a Group of Individual Objects Prior to Processing then we should go for Queue.

Eg: Before sending a Mail we have to Store All MailID's in Some Data Structure and in which Order we added MailID's in the Same Order Only Mails should be delivered (FIFO). For this Requirement Queue is Best Suitable.

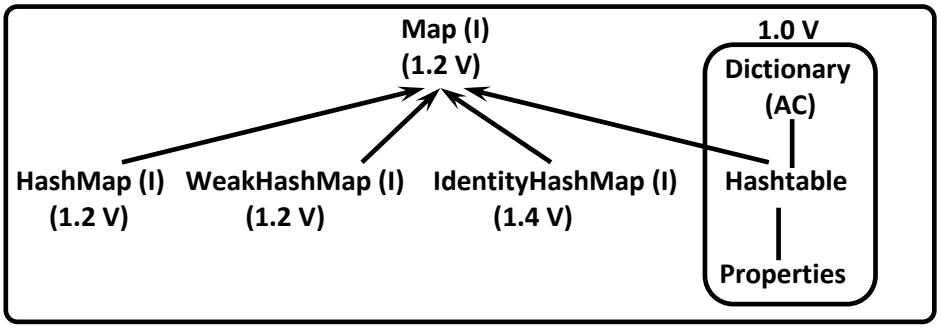


Note:

- All the Above Interfaces (Collection, List , Set, SortedSet, NavigableSet, and Queue) Meant for representing a Group of Individual Objects.
- If we want to Represent a Group of Key - Value Pairs then we should go for Map.

7) Map (I):

- Map is Not Child Interface of Collection.
- If we want to Represent a Group of Objects as Key - Value Pairs then we should go for Map Interface.
- Duplicate Keys are Not allowed but Values can be Duplicated.

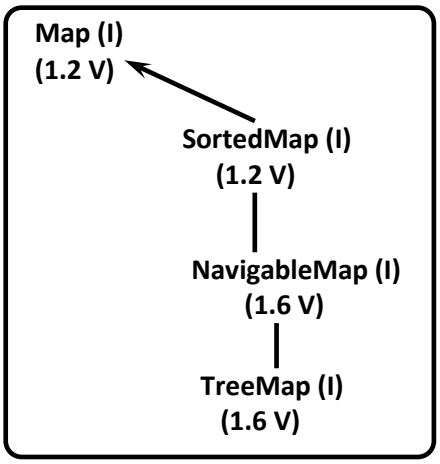


8) SortedMap (I):

- It is the Child Interface of Map.
- If we want to Represent a Group of Objects as Key - Value Pairs according to Some Sorting Order of Keys then we should go for SortedMap.
- Sorting should be Based on Key but Not Based on Value.

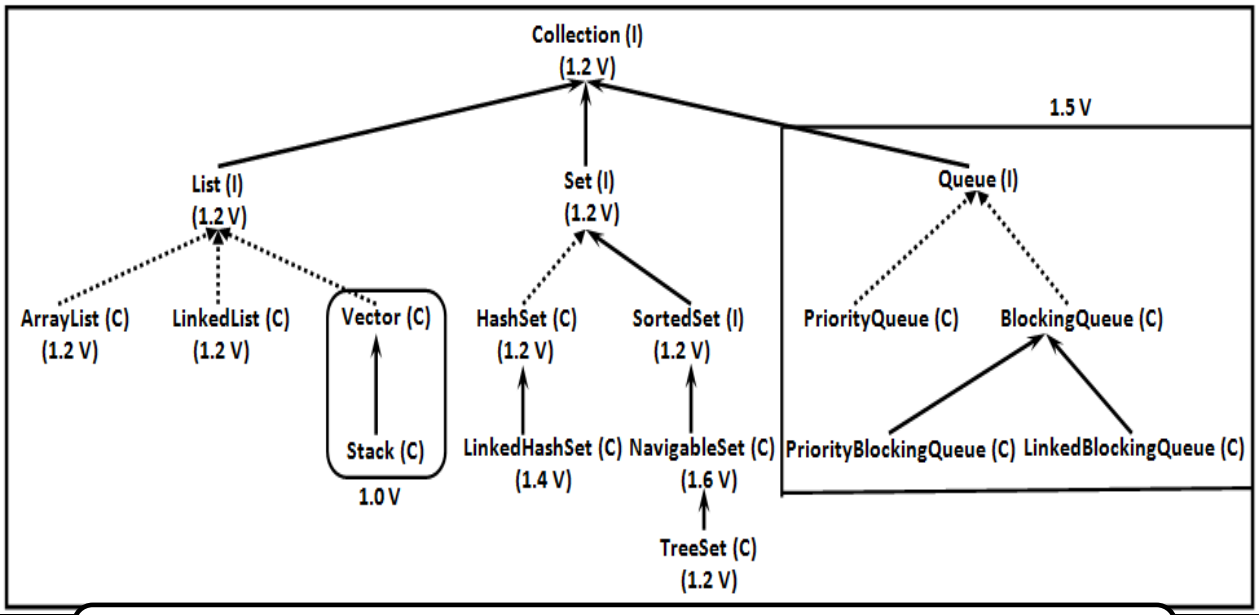
9) NavigableMap (I):

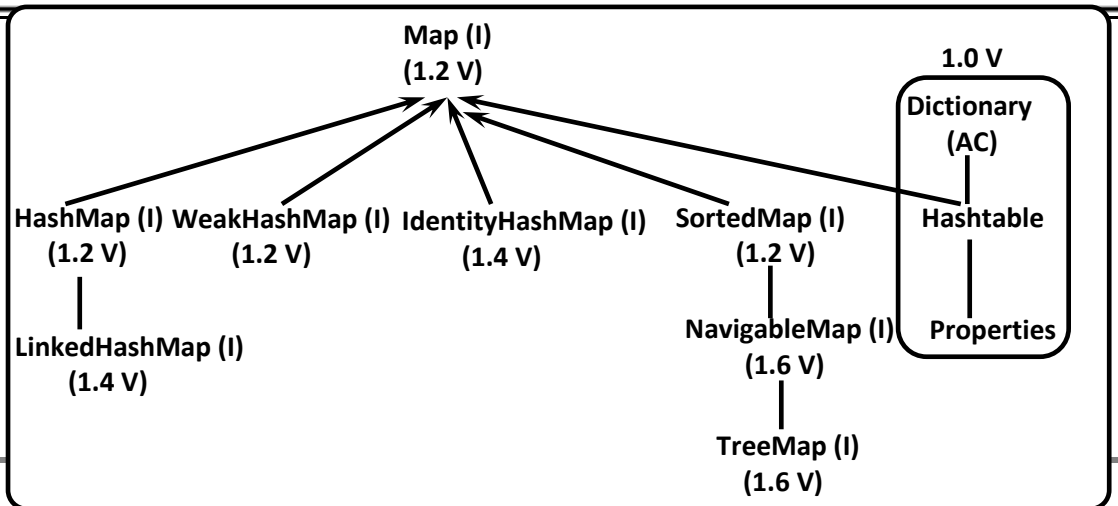
- It is the Child Interface of SortedMap.
- It Defines Several Methods for Navigation Purposes.



Note: In Collection Framework the following are Legacy Classes.

1) Enumeration (I)
2) Dictionary (Abstract Class)
3) Vector (Concrete Class)
4) Stack (Concrete Class)
5) Hashtable (Concrete Class)
6) Properties (Concrete Class)





Utility Classes

1) Collections
2) Arrays

Sorting

1) Comparable (I)
2) Comparator (I)

Cursors

1) Enumeration (I)
2) Iterator (I)
3) ListIterator (I)

1) Collection Interface:

If we want to Represent a Group of Individual Objects as a Single Entity then we should go for Collection Interface.

Methods:

- Collection Interface defines the Most Common Methods which are Applicable for any Collection Objects.
- The following is the List of the Methods Present Inside Collection Interface.

1) boolean add(Object o)

2) boolean addAll(Collection c)

3) boolean remove(Object o)

4) boolean removeAll(Collection c)

5) boolean retainAll(Collection c): To Remove All Objects Except those Present in c.

6) void clear()

7) boolean contains(Object o)

8) boolean containsAll(Collection c)

9) boolean isEmpty()

10) int size()

11) Object[] toArray()

12) Iterator iterator()

Note:

- There is No Concrete Class which implements Collection Interface Directly.
- There is No Direct Method in Collection Interface to get Objects.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

2) List:

- It is the Child Interface of Collection.
- If we want to Represent a Group of Individual Objects where Duplicates are allowed and Insertion Order Preserved. Then we should go for List.
- We can Preserve Insertion Order and we can Differentiate Duplicate Object by using Index. Hence Index will Play Very Important Role in List.

Methods: List Interface Defines the following Specific Methods.

1) void add(int index, Object o)

2) boolean addAll(int index, Collection c)

3) Object get(int index)

4) Object remove(int index)

5) Object set(int index, Object new): To Replace the Element Present at specified Index with provided Object and Returns Old Object.

6) int indexOf(Object o): Returns Index of 1st Occurrence of 'o'

7) int lastIndexOf(Object o)

8) ListIterator listIterator();

2.1) ArrayList:

- The Underlying Data Structure for ArrayList is Resizable Array OR Growable Array.
- Duplicate Objects are allowed.
- Insertion Order is Preserved.
- Heterogeneous Objects are allowed (Except *TreeSet* and *TreeMap* Everywhere Heterogeneous Objects are allowed).
- null Insertion is Possible.

Constructors:

1) ArrayList l = new ArrayList();
   - Creates an Empty ArrayList Object with Default Initial Capacity 10.
   - If ArrayList Reaches its Max Capacity then a New ArrayList Object will be Created with

     New Capacity = (Current Capacity * 3/2) + 1

2) ArrayList l = new ArrayList(int initialCapacity);
   Creates an Empty ArrayList Object with specified Initial Capacity.

3) ArrayList l = new ArrayList(Collection c);

- Creates an Equalent ArrayList Object for the given Collection Object.
- This Constructor Meant for Inter Conversion between Collection Objects.

```
import java.util.ArrayList;
class ArrayListDemo {
        public static void main(String[] args) {

                ArrayList l = new ArrayList();

                l.add("A");
                l.add(10);
                l.add("A");
                l.add(null);
                System.out.println(l); //[A, 10, A, null]

                l.remove(2);
                System.out.println(l); //[A, 10, null]

                l.add(2,"M");
                l.add("N");
                System.out.println(l); //[A, 10, M, null, N]

        }
}
```

- Usually we can Use Collections to Hold and Transfer Data (Objects) form One Location to Another Location.
- To Provide Support for this Requirement Every Collection Class Implements *Serializable* and *Cloneable* Interfaces.
- *ArrayList* and *Vector* Classes Implements *RandomAccess* Interface. So that we can Access any Random Element with the Same Speed.
- *RandomAccess* Interface Present in *java.util* Package and it doesn't contain any Methods. Hence it is a *Marker* Interface.
- Hence ArrayList is Best Suitable if Our Frequent Operation is Retrieval Operation.

```
ArrayList l1 = new ArrayList();
LinkedList l2 = new LinkedList();

System.out.println(l1 instanceof Serializable); //true
System.out.println(l2 instanceof Cloneable); //true
System.out.println(l1 instanceof RandomAccess); //true
System.out.println(l2 instanceof RandomAccess); //false
```

Differences between *ArrayList* and *Vector*:

ArrayList	Vector
Every Method Present Inside ArrayList is Non – Synchronized.	Every Method Present in Vector is Synchronized.
At a Time Multiple Threads are allow to Operate on ArrayList Simultaneously and Hence ArrayList Object is Not Thread Safe.	At a Time Only One Thread is allow to Operate on Vector Object and Hence Vector Object is Always Thread Safe.
Relatively Performance is High because Threads are Not required to Wait.	Relatively Performance is Low because Threads are required to Wait.
Introduced in 1.2 Version and it is Non – Legacy.	Introduced in 1.0 Version and it is Legacy.

How to get Synchronized Version of ArrayList Object?

By Default ArrayList Object is Non - Synchronized but we can get Synchronized Version ArrayList Object by using the following Method of Collections Class.

```
public static List synchronizedList(List l)
```

Eg:

```
ArrayList al = new ArrayList ();
List l = Collections.synchronizedList(al);
```

Synchronized Version

Non - Synchronized Version

Similarly we can get Synchronized Version of *Set* and *Map* Objects by using the following Methods of Collection Class.

```
public static Set synchronizedSet(Set s)

public static Map synchronizedMap(Map m)
```

- ArrayList is the Best Choice if we want to Perform Retrieval Operation Frequently.
- But ArrayList is Worst Choice if Our Frequent Operation is Insertion OR Deletion in the Middle. Because it required Several Shift Operations Internally.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Practice Questions for ArrayList

Q1. Given the code fragment:

```
1)  import java.util.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        List<String> l = new ArrayList<>();
7)        l.add("Robb");
8)        l.add("Bran");
9)        l.add("Rick");
10)       l.add("Bran");
11)       if(l.remove("Bran"))
12)       {
13)          l.remove("Jon");
14)       }
15)       System.out.println(l);
16)    }
17) }
```

What is the result?

A. [Robb, Rick, Bran]
B. [Robb, Rick]
C. [Robb, Bran, Rick, Bran]
D. An exception is thrown at runtime

Answer: A

Q2. Given the code fragment

```
1)  import java.util.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        ArrayList l = new ArrayList();
7)        String[] s;
8)        try
9)        {
10)          while(true)
11)          {
12)             l.add("MyString");
13)          }
14)       }
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

```
15)       catch (RuntimeException e)
16)       {
17)          System.out.println("RuntimeException caught");
18)       }
19)       catch (Exception e)
20)       {
21)          System.out.println("Exception caught");
22)       }
23)       System.out.println("Ready to use");
24)    }
25) }
```

What is the result?

A.
RuntimeException caught
Ready to use
B.
Exception caught
Ready to use
C. Compilation Fails
D. A runtime error thrown in the thread main

Answer: D

Q3) Given:

```
1)  import java.util.*;
2)  class Patient
3)  {
4)     String name;
5)     public Patient(String name)
6)     {
7)        this.name=name;
8)     }
9)  }
10) class Test
11) {
12)    public static void main(String[] args)
13)    {
14)       List l = new ArrayList();
15)       Patient p = new Patient("Mike");
16)       l.add(p);
17)       //insert code here==>Line-1
18)       if(f>=0)
19)       {
20)          System.out.println("Mike Found");
21)       }
22)    }
```

```
23) }
```

Which code inserted at Line-1 enable the code to print Mike Found.

A. int f=l.indexOf(p);
B. int f=l.indexOf(Patient("Mike"));
C. int f=l.indexOf(new Patient("Mike"));
D. Patient p1 = new Patient("Mike");
   int f=l.indexOf(p1);

Answer: A

Q4. Given the code fragment:

```
1)  import java.util.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        ArrayList<Integer> l = new ArrayList<>();
7)        l.add(1);
8)        l.add(2);
9)        l.add(3);
10)       l.add(4);
11)       l.add(null);
12)       l.remove(2);
13)       l.remove(null);
14)       System.out.println(l);
15)    }
16) }
```

What is the result?

A. [1, 2, 4]
B. NullPointerException is thrown at runtime
C. [1, 2, 4,null]
D. [1, 3, 4,null]
E. [1, 3, 4]
F. Compilation Fails

Answer: A

Q5. Given the following class declarations

public abstract class Animal
public interface Hunter
public class Cat extends Animal implements Hunter
public class Tiger extends Cat

Which one fails to compile?

A.
```
ArrayList<Animal> l = new ArrayList<>();
l.add(new Tiger());
```

B.
```
ArrayList<Hunter> l = new ArrayList<>();
l.add(new Cat());
```

C.
```
ArrayList<Hunter> l = new ArrayList<>();
l.add(new Tiger());
```

D.
```
ArrayList<Tiger> l = new ArrayList<>();
l.add(new Cat());
```

E.
```
ArrayList<Animal> l = new ArrayList<>();
l.add(new Cat());
```

Answer: D

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com