

## **Java Collection Framework**

- A Group of objects is called a collection.
- Standard techniques or guidelines are called Frameworks.

It provides:

### **1. Organization of Objects (Data Structures): Arrangement**

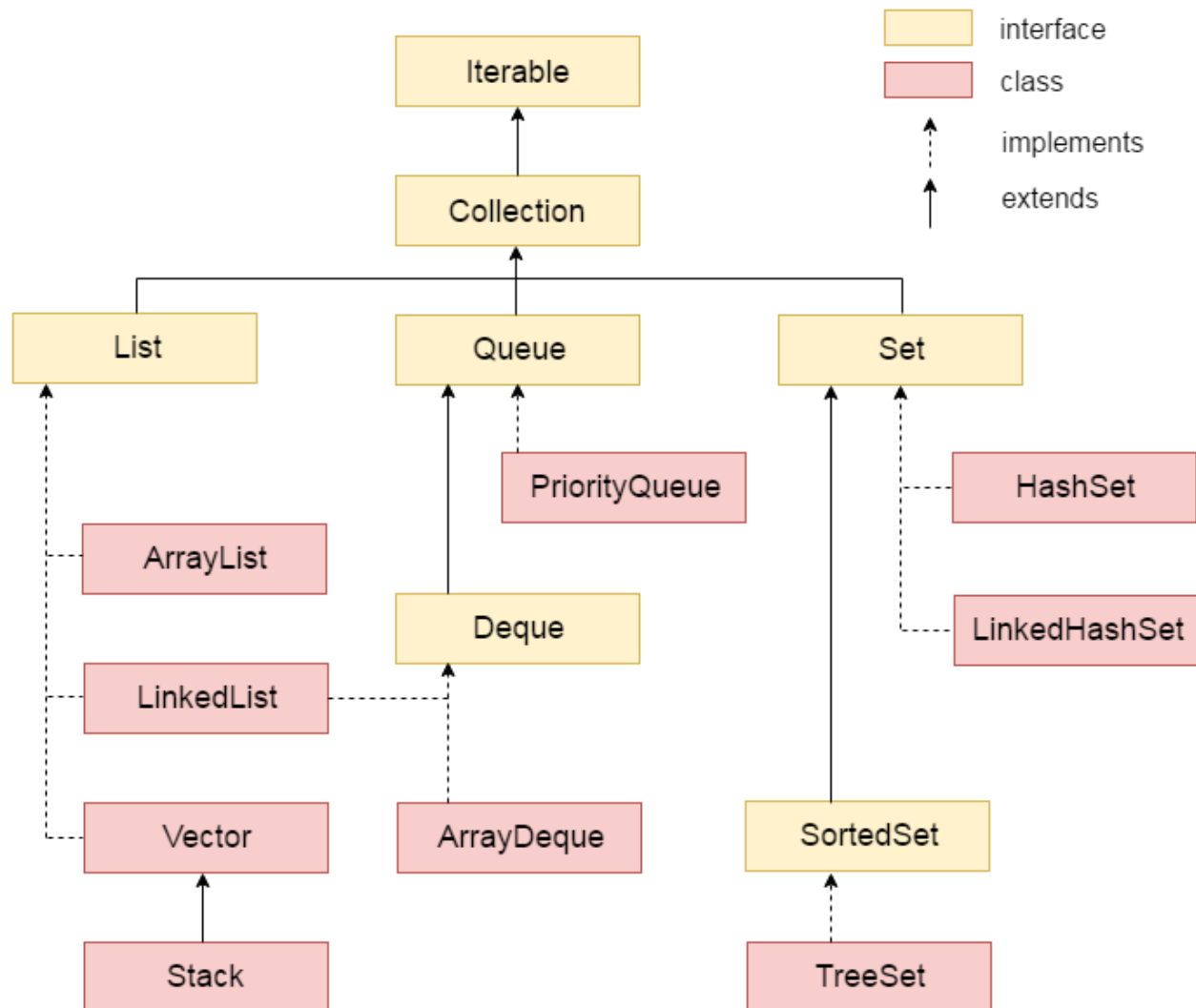
- Array
- Stack
- LinkedList
- Map(Key-Value)
- Tree etc...

### **2. Operations on data structures:**

- Insert
- Delete
- Traverse
- Add
- Edit
- Find
- Duplicate
- Shuffle
- Max
- Min
- Sort
- Search
- Merge etc.

# Hierarchy of Collection Framework

Let us see the hierarchy of the collection framework. The **java.util** package contains all the classes and interfaces for the Collection framework.



### Some behavior of Collection:

1. All are resizable, have No issue with size, and are flexible.
2. Auto grows and shrinks.
3. Allow multi types of objects
4. Ordered/unordered
5. Sorted/Unsorted
6. Allow Unique/Duplicate
7. Allow null or not
8. Synchronous/Asynchronous
9. Key Value(K-V) pair (in only map)

### Methods of Collection interface:

No.	Method	Description
1	public boolean add(Object element)	is used to insert an element in this collection.
2	public boolean addAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.

8	public boolean contains(Object element)	is used to search an element.
9	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public boolean isEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public int hashCode()	returns the hashcode number for collection.

### **Bubble sort (Not in Syllabus but discussing for the problem requirement)**

- Compare adjacent elements in the array.
- Swap them if they are in the wrong order.
- Repeat the process for all elements until the array is sorted.
- Largest elements "bubble up" to their correct positions in each pass.
- Stop when no more swaps are needed.

### **Time Complexity:**

- Worst/Average Case:  $O(n^2)$
- Best Case (Already Sorted):  $O(n)$

### **Properties:**

- Fixed Size: Arrays require a predefined size.
- Manual Sorting: We need to write custom sorting logic.
- Adding/Removing Elements: Requires shifting elements manually.
- Bubble Sort is one of the simplest sorting algorithms.
- It uses basic loops and swapping, making it easy to implement and understand.
- Bubble Sort shows each swap step clearly, making it a good choice for demonstrating sorting logic.

- Bubble Sort is inefficient for large datasets ( $O(n^2)$  complexity), but its performance is acceptable for small arrays like 5 integers or 3 students.

### Questions:

- Write a Java program that stores five integer values in an array, sorts them in ascending order without using the Java Collection Framework, and then prints the sorted numbers. Implement the sorting logic manually using Bubble Sort.
- Write a Java program to store student details (name and age) in a fixed-size array. Implement a sorting mechanism to arrange the students in ascending order based on their age without using the Java Collection Framework. Use Bubble Sort to perform the sorting and then display the sorted list of students.

### Program without Using Collections (Using Arrays)

```

WithoutCollections.java
1  import java.util.Arrays;
2  class WithoutCollections {
3      public static void main(String[] args) {
4          int[] numbers = new int[5]; // Fixed size array
5          numbers[0] = 10;
6          numbers[1] = 50;
7          numbers[2] = 30;
8          numbers[3] = 40;
9          numbers[4] = 20;
10         // Sorting manually (Bubble Sort)
11         for (int i = 0; i < numbers.length - 1; i++) {
12             for (int j = 0; j < numbers.length - i - 1; j++) {
13                 if (numbers[j] > numbers[j + 1]) {
14                     // Swap elements
15                     int temp = numbers[j];
16                     numbers[j] = numbers[j + 1];
17                     numbers[j + 1] = temp;
18                 }
19             }
20         }
21         // Printing sorted numbers
22         for (int num : numbers) {
23             System.out.print(num + " ");
24         }
25     }
26 }

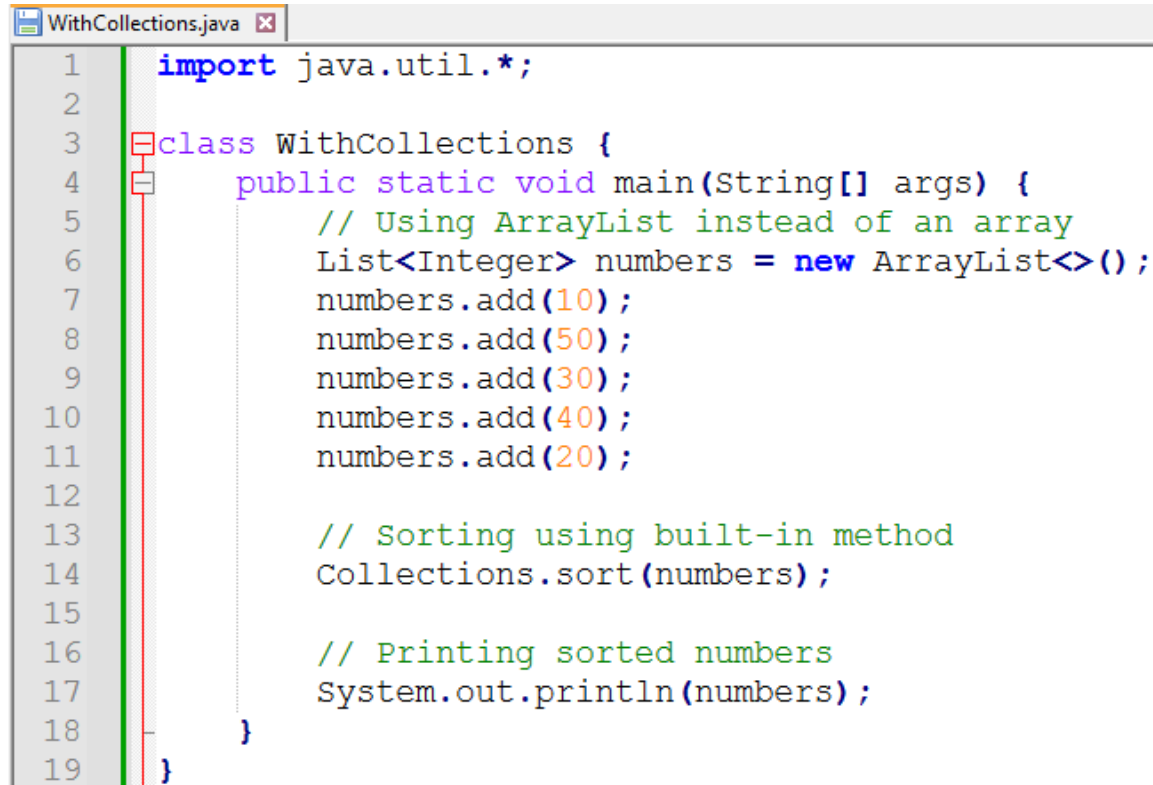
```

Output:

```
D:\Java Code\collection>javac WithoutCollections.java

D:\Java Code\collection>java WithoutCollections
10 20 30 40 50
```

### Program code Using Collections (ArrayList)



```
1  import java.util.*;
2
3  class WithCollections {
4      public static void main(String[] args) {
5          // Using ArrayList instead of an array
6          List<Integer> numbers = new ArrayList<>();
7          numbers.add(10);
8          numbers.add(50);
9          numbers.add(30);
10         numbers.add(40);
11         numbers.add(20);
12
13         // Sorting using built-in method
14         Collections.sort(numbers);
15
16         // Printing sorted numbers
17         System.out.println(numbers);
18     }
19 }
```

```
D:\Java Code\collection>javac WithCollections.java

D:\Java Code\collection>java WithCollections
[10, 20, 30, 40, 50]
```

### Comparison Table

Feature	Without Collections (Array)	With Collections (ArrayList)
Size	Fixed	Dynamic
Sorting	Manual Bubble Sort	Collections.sort()
Insertion/Deletion	Complex, manual shifting required	Simple add() / remove() methods
Flexibility	Requires manual management	Built-in operations

Write a Java program to store student details (name and age) in a fixed-size array. Implement a sorting mechanism to arrange the students in ascending order based on their age without using the Java Collection Framework. Use Bubble Sort to perform the sorting and then display the sorted list of students.

### Without Using Collections (Using Arrays)

```
WithoutCollections1.java
1  import java.util.Arrays;
2  class Student {
3      String name;
4      int age;
5      Student(String name, int age) {
6          this.name = name;
7          this.age = age; }
8  class WithoutCollections1 {
9      public static void main(String[] args) {
10         // Fixed size array
11         Student[] students = new Student[3];
12         students[0] = new Student("Abhay", 22);
13         students[1] = new Student("Balveer", 20);
14         students[2] = new Student("Charitra", 25);
15         // Sorting students manually by age using Bubble Sort
16         for (int i = 0; i < students.length - 1; i++) {
17             for (int j = 0; j < students.length - i - 1; j++) {
18                 if (students[j].age > students[j + 1].age) {
19                     // Swap students
20                     Student temp = students[j];
21                     students[j] = students[j + 1];
22                     students[j + 1] = temp; }
23             // Display students
24             for (Student s : students) {
25                 System.out.println(s.name + " - " + s.age); }
26         }
```

```
D:\Java Code\collection>javac WithoutCollections1.java
```

```
D:\Java Code\collection>java WithoutCollections1
```

```
Balveer - 20
```

```
Abhay - 22
```

```
Charitra - 25
```

## Using Collection framework

```
WithCollections1.java x
1  import java.util.*;
2  class Student {
3      String name;
4      int age;
5      Student(String name, int age) {
6          this.name = name;
7          this.age = age;
8      }}
9
10 class WithCollections1 {
11     public static void main(String[] args) {
12         // Using ArrayList to store students
13         List<Student> students = new ArrayList<>();
14         students.add(new Student("Abhay", 22));
15         students.add(new Student("Balveer", 20));
16         students.add(new Student("Charitra", 25));
17
18         // Sorting students by age using Collections.sort()
19         students.sort(Comparator.comparingInt(s -> s.age));
20
21         // Display students
22         for (Student s : students) {
23             System.out.println(s.name + " - " + s.age);
24         }}
```

s -> s.age is a lambda expression.

```
D:\Java Code\collection>java WithCollections1
Balveer - 20
Abhay - 22
Charitra - 25
```

## Comparison Table

Feature	Without Collections (Array)	With Collections (ArrayList)
Size Management	Fixed, hardcoded size	Dynamic, can grow/shrink
Sorting	Manual Bubble Sort	Collections.sort() with Comparator
Adding/Removing	Complex, requires shifting	Simple add() and remove()
Code Complexity	More lines, error-prone	Cleaner, easier to maintain



## Iterable and Iterator Interface

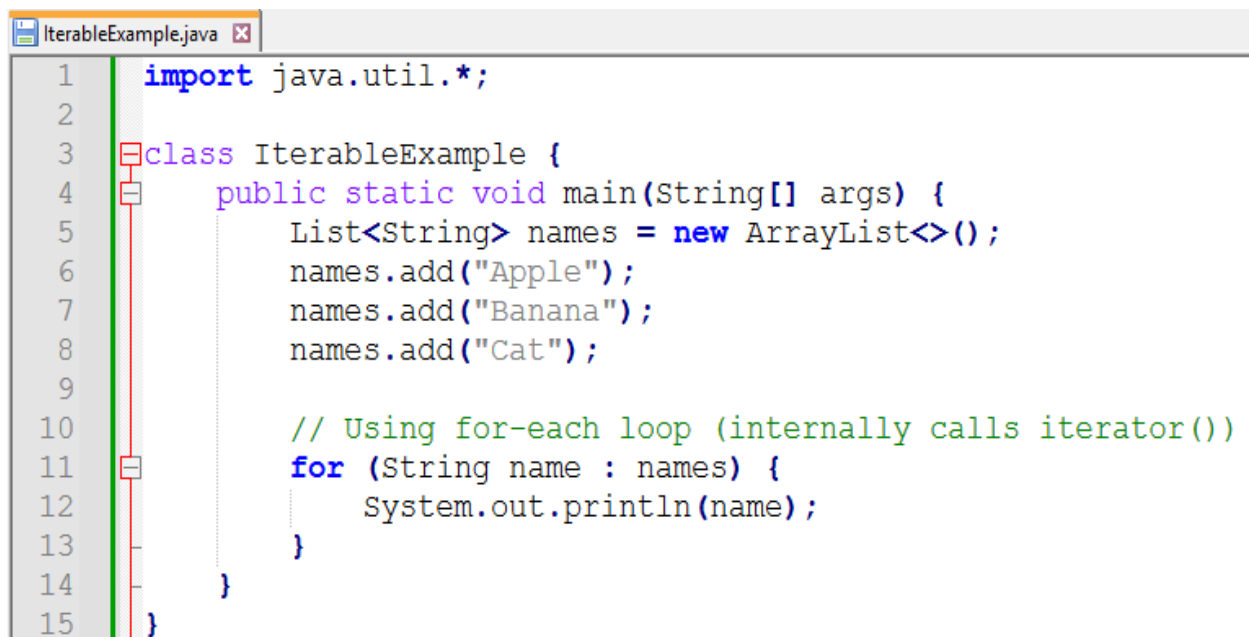
- The Iterable and Iterator interfaces are part of the Java Collections Framework, providing a way to traverse or iterate over elements in a collection.
- While they are related, they have distinct purposes and roles.

### Iterable Interface

The Iterable interface is present in java.lang package and is the root interface for all collection classes. It allows an object to be the target of the for-each loop.

#### Key Points:

- It has only one method: `iterator()`, which returns an `Iterator`.
- Implementing `Iterable<T>` allows the collection to be iterated using an enhanced for loop.
- Common collections like `ArrayList`, `HashSet`, and `LinkedList` implement `Iterable`.



```
1  import java.util.*;
2
3  class IterableExample {
4      public static void main(String[] args) {
5          List<String> names = new ArrayList<>();
6          names.add("Apple");
7          names.add("Banana");
8          names.add("Cat");
9
10         // Using for-each loop (internally calls iterator())
11         for (String name : names) {
12             System.out.println(name);
13         }
14     }
15 }
```



```
Apple
Banana
Cat
```

## Iterator Interface

- The Iterator interface is present in the Java.util package and provides methods to iterate over a collection.
- The iterator interface allows iterating the elements in the forward direction only.

### Key Methods:

- hasNext(): Returns true if more elements exist.
- next(): Returns the next element.
- remove(): Removes the last element returned by next().

```
IteratorExample.java
1  import java.util.*;
2
3  class IteratorExample {
4      public static void main(String[] args) {
5          List<Integer> numbers = new ArrayList<>();
6          numbers.add(10);
7          numbers.add(20);
8          numbers.add(30);
9
10         Iterator<Integer> itr = numbers.iterator();
11
12         while (itr.hasNext()) {
13             System.out.println(itr.next());
14         }
15     }
16 }
```

```
10
20
30
```

```

1  import java.util.*;
2  class IteratorMethodsExample {
3      public static void main(String[] args) {
4          // Creating a list of integers
5          List<Integer> numbers = new ArrayList<>();
6          numbers.add(10);
7          numbers.add(20);
8          numbers.add(30);
9          numbers.add(40);
10         numbers.add(50);
11         // Getting an iterator for the list
12         Iterator<Integer> itr = numbers.iterator();
13         // Using hasNext() to check if elements are available
14         while (itr.hasNext()) {
15             // Using next() to get the next element
16             int num = itr.next();
17             System.out.println("Current Element: " + num);
18
19             // Using remove() to remove elements that meet a condition
20             if (num == 30) {
21                 itr.remove(); // Removes the last returned element (30)
22                 System.out.println("Element 30 removed");
23             }
24         }
25         System.out.println("Final List: " + numbers);
26     }
}

```

```

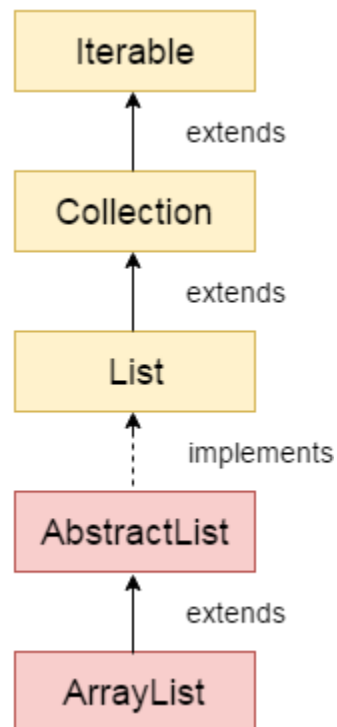
Current Element: 10
Current Element: 20
Current Element: 30
Element 30 removed
Current Element: 40
Current Element: 50
Final List: [10, 20, 40, 50]

```

## Java ArrayList class

Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

- 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 the array works on an index basis.
- ArrayList is Duplicate, Ordered, Unsorted, Asynchronous, Allow Null



### ArrayList class declaration:

```
public class ArrayList<E> extends AbstractList<E> implements List<E>
```

### Constructors of Java ArrayList

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

### Methods of Java ArrayList

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
boolean addAll(Collection c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
void clear()	It is used to remove all of the elements from this list.

<code>int lastIndexOf(Object o)</code>	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
<code>Object[] toArray()</code>	It is used to return an array containing all of the elements in this list in the correct order.
<code>Object[] toArray(Object[] a)</code>	It is used to return an array containing all of the elements in this list in the correct order.
<code>boolean add(Object o)</code>	It is used to append the specified element to the end of a list.
<code>boolean addAll(int index, Collection c)</code>	It is used to insert all of the elements in the specified collection into this list, starting at the specified position.
<code>Object clone()</code>	It is used to return a shallow copy of an ArrayList.
<code>int indexOf(Object o)</code>	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
<code>void trimToSize()</code>	It is used to trim the capacity of this ArrayList instance to be the list's current size.

## Java Non-generic Vs. Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

Java's new generic collection allows you to have **only one type of object** in the collection. Now it is type-safe, so typecasting is not required at run time.

In the generic collection, we specify the type in **angular braces**. Now ArrayList is forced to have only specified types of objects in it. If you try to add another type of object, it gives *compile time error*.

1. ArrayList al=**new** ArrayList ( ) ;//creating old non-generic arraylist

Let's see the new generic example of creating java collection.

1. ArrayList<String> al=**new** ArrayList<String>();//creating new generic arraylist

### ArrayList Program:

```
import java.util.*;

class TestCollection1 {

    public static void main(String args[]){

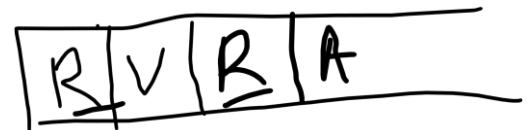
        ArrayList<String> list=new ArrayList<String>();//Creating array list

        list.add("Ravi");//Adding object in array list

        list.add("Vijay");

        list.add("Ravi");

        list.add("Ajay");
```



### //Traversing list through Iterator

```
Iterator itr=list.iterator();

while(itr.hasNext())// It returns true if iterator has more elements {

    System.out.println(itr.next()); //fetch item.

}

}
```

```
}
```

C:\Windows\System32\cmd.exe

```
D:\1 Java\Programs>javac TestCollection1.java
```

```
D:\1 Java\Programs>java TestCollection1
```

```
Ravi
```

```
Vijay
```

```
Ravi
```

```
Ajay
```

### //USER DEFINED ARRAYLIST

```
class Student{
    int rollno;
    String name;
    int age;
    Student(int rollno,String name,int age){
        this.rollno=rollno;
        this.name=name;
        this.age=age;
    }
}
```

```
import java.util.*;
class UserAL{
    public static void main(String args[]){
        //Creating user-defined class objects
        Student s1=new Student(101,"Saurabh",23);
        Student s2=new Student(102,"Ravi",21);
        Student s3=new Student(103,"Vineet",25);
        //creating arraylist
        ArrayList<Student> al=new ArrayList<Student>();
        al.add(s1);//adding Student class object
        al.add(s2);
        al.add(s3);
        //Getting Iterator
        Iterator itr=al.iterator();
        //traversing elements of ArrayList object
        while(itr.hasNext()){
```



```
        Student st=(Student)itr.next(); //type casting
        System.out.println(st.rollno+" "+st.name+" "+st.age);
    }
}
```

```
D:\1 Java\Programs\ArrayList>javac UserAL.java
```

```
D:\1 Java\Programs\ArrayList>java UserAL
```

```
101 Saurabh 23
```

```
102 Ravi 21
```

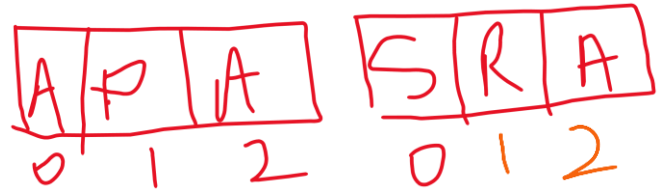
```
103 Vineet 25
```

### //addAll() method

```
import java.util.*;
class AddAll{
    public static void main(String args[]){
        ArrayList<String> al=new ArrayList<String>();
        al.add("Anurag");
        al.add("Praksah");
        al.add("Ajay");

        ArrayList<String> al2=new ArrayList<String>();
        al2.add("Saurabh");
        al2.add("Rohit");
        al2.add("Ajay");

        al.addAll(al2); //adding second list al2 in first list al
        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```



```
D:\1 Java\Programs\ArrayList>javac AddAll.java

D:\1 Java\Programs\ArrayList>java AddAll
Anurag
Praksah
Ajay
Saurabh
Rohit
Ajay
```

### //Example of removeAll() method

```
import java.util.*;
class RemoveAll{
    public static void main(String args[]){
        ArrayList<String> al=new ArrayList<String>();
        al.add("Anurag");
        al.add("Praksah");
        al.add("Ajay");

        ArrayList<String> al2=new ArrayList<String>();
        al2.add("Saurabh");
        al2.add("Rohit");
        al2.add("Ajay");
        al.removeAll(al2); //it removes duplicate element "Ajay" from al.
        System.out.println("iterating the elements after removing the elements of al2...");
        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }

    }
}
```

```
D:\1 Java\Programs\ArrayList>javac RemoveAll.java

D:\1 Java\Programs\ArrayList>java RemoveAll
iterating the elements after removing the elements of al2...
Anurag
Praksah
```

### //Example of retainAll() method

```
import java.util.*;
class RetainAll{
    public static void main(String args[]){
        ArrayList<String> al=new ArrayList<String>();
        al.add("Anurag");
        al.add("Praksah");
        al.add("Ajay");

        ArrayList<String> al2=new ArrayList<String>();
        al2.add("Saurabh");
        al2.add("Rohit");
        al2.add("Ajay");
        al.retainAll(al2);//It will retain all duplicate elements only.
        System.out.println("iterating the elements after retain the elements of al2...");
        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

```
D:\1 Java\Programs\ArrayList>java RetainAll
iterating the elements after removing the elements of al2...
Ajay
```

## Java HashSet class

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

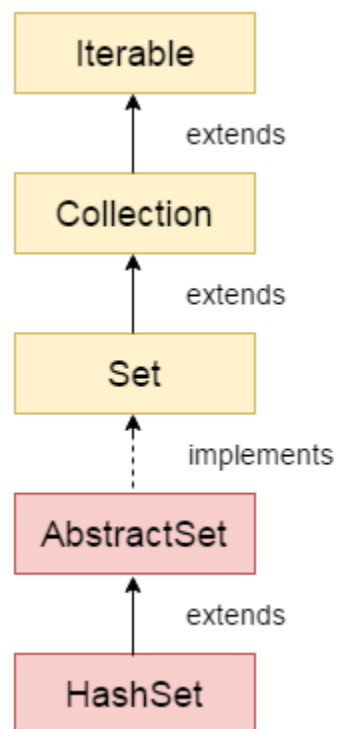
The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing**.
- HashSet contains unique elements only.

## Difference between List and Set

List can contain duplicate elements whereas Set contains unique elements only.

Hierarchy of HashSet class



## HashSet class declaration

Let's see the declaration for java.util.HashSet class.

1. **public class** HashSet<E> **extends** AbstractSet<E> **implements** Set<E>

Constructors of Java HashSet class:

Constructor	Description
HashSet()	It is used to construct a default HashSet.
HashSet(Collection c)	It is used to initialize the hash set by using the elements of the collection c.
HashSet(int capacity)	It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.

Methods of Java HashSet class:

Method	Description
void clear()	It is used to remove all of the elements from this set.
boolean contains(Object o)	It is used to return true if this set contains the specified element.
boolean add(Object o)	It is used to adds the specified element to this set if it is not already present.
boolean isEmpty()	It is used to return true if this set contains no elements.

boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
Object clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
Iterator iterator()	It is used to return an iterator over the elements in this set.
int size()	It is used to return the number of elements in this set.

```

import java.util.*;
class TestHashSet{
public static void main(String args[]){
//Creating HashSet and adding elements
HashSet<String> set=new HashSet<String>();
set.add("Ravi");
set.add("Vijay");
set.add("Ravi");
set.add("Ajay");
//Traversing elements
Iterator<String> itr=set.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}

```

```
C:\Windows\System32\cmd.exe

D:\1 Java\Programs\HashSet>javac TestHashSet.java

D:\1 Java\Programs\HashSet>java TestHashSet
Vijay
Ravi
Ajay
```

### Java TreeSet class

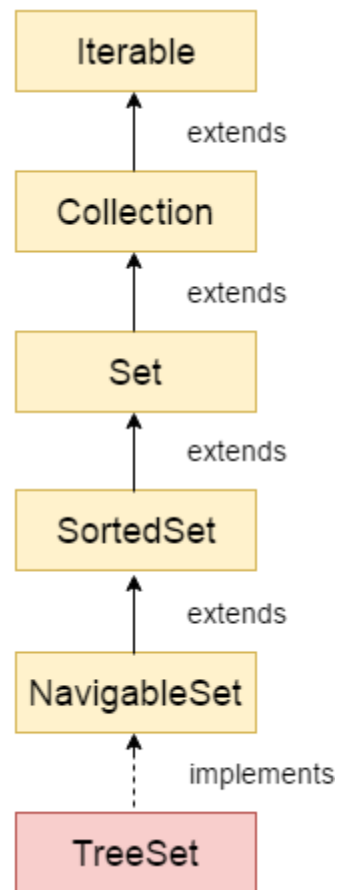
Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements NavigableSet interface. The objects of TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

- Contains unique elements only like HashSet.
- Access and retrieval times are quiet fast.
- Maintains ascending order.

Hierarchy of TreeSet class





### TreeSet class declaration

public class **TreeSet**<E> extends **AbstractSet**<E> implements **NavigableSet**<E>

Constructors of Java TreeSet class

Constructor	Description
TreeSet()	It is used to construct an empty tree set that will be sorted in an ascending order according to the natural order of the tree set.
TreeSet(Collection c)	It is used to build a new tree set that contains the elements of the collection c.
TreeSet(Comparator comp)	It is used to construct an empty tree set that will be sorted according to given comparator.

TreeSet(SortedSet ss)	It is used to build a TreeSet that contains the elements of the given SortedSet.
-----------------------	--

#### Methods of Java TreeSet class

Method	Description
boolean addAll(Collection c)	It is used to add all of the elements in the specified collection to this set.
boolean contains(Object o)	It is used to return true if this set contains the specified element.
boolean isEmpty()	It is used to return true if this set contains no elements.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
void add(Object o)	It is used to add the specified element to this set if it is not already present.
void clear()	It is used to remove all of the elements from this set.
Object clone()	It is used to return a shallow copy of this TreeSet instance.
Object first()	It is used to return the first (lowest) element currently in this sorted set.
Object last()	It is used to return the last (highest) element currently in this sorted set.
int size()	It is used to return the number of elements in this set.

#### //TreeSet Example

```
import java.util.*;
class TestTreeSet{
public static void main(String args[]){
//Creating and adding elements
TreeSet<String> al=new TreeSet<String>();
```

```
al.add("Ravi");
al.add("Vijay");
al.add("Ravi");
al.add("Ajay");
//Traversing elements
Iterator<String> itr=al.iterator();
while(itr.hasNext()){
    System.out.println(itr.next());
}
}
}
```

C:\Windows\System32\cmd.exe

```
D:\1 Java\Programs\TreeSet>javac TestTreeSet.java

D:\1 Java\Programs\TreeSet>java TestTreeSet
Ajay
Ravi
Vijay
```