Java Collection Framework

Group of objects is called collection.

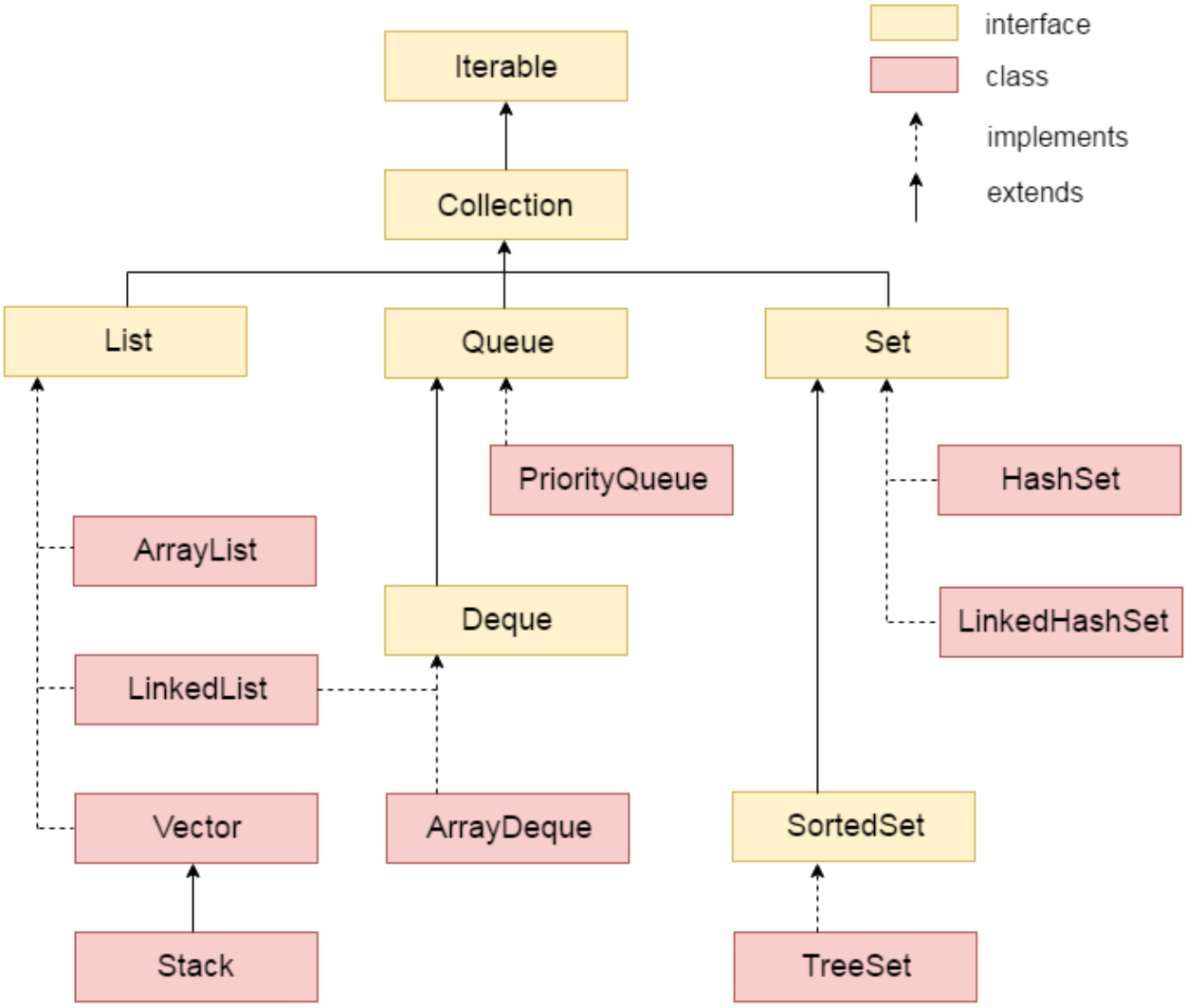
Standard techniques are called Framework(Guidelines).

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 collection framework. The **java.util** package contains all the classes and interfaces for Collection framework.



Some behavior of Collection:

1. All are resizable, No issue of size, flexible. Auto grow and shrink.
2. Allow multiple type of objects
3. Ordered/unordered
4. Sorted/Unsorted
5. Allow Unique/Duplicate
6. Allow null or not
7. Synchronous/Asynchronous
8. 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.

Iterator interface

Iterator interface provides the facility of iterating the elements in forward direction only.

Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

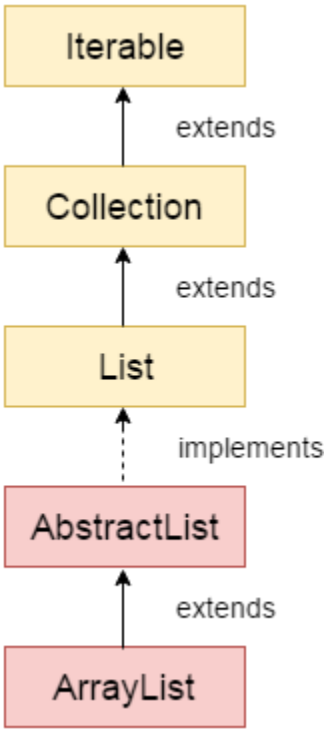
No.	Method	Description
1	public boolean hasNext()	It returns true if iterator has more elements.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.

3	public void remove()	It removes the last elements returned by the iterator. It is rarely used.

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 array works at the 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.

int lastIndexOf(Object o)	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.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
Object[] toArray(Object[] a)	It is used to return an array containing all of the elements in this list in the correct order.
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean addAll(int index, Collection c)	It is used to insert all of the elements in the specified collection into this list, starting at the specified position.
Object clone()	It is used to return a shallow copy of an ArrayList.
int indexOf(Object o)	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.
void trimToSize()	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 new generic collection allows you to have **only one type of object** in collection. Now it is type safe so typecasting is not required at run time.

In generic collection, we specify the type in **angular braces**. Now ArrayList is forced to have only specified type 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

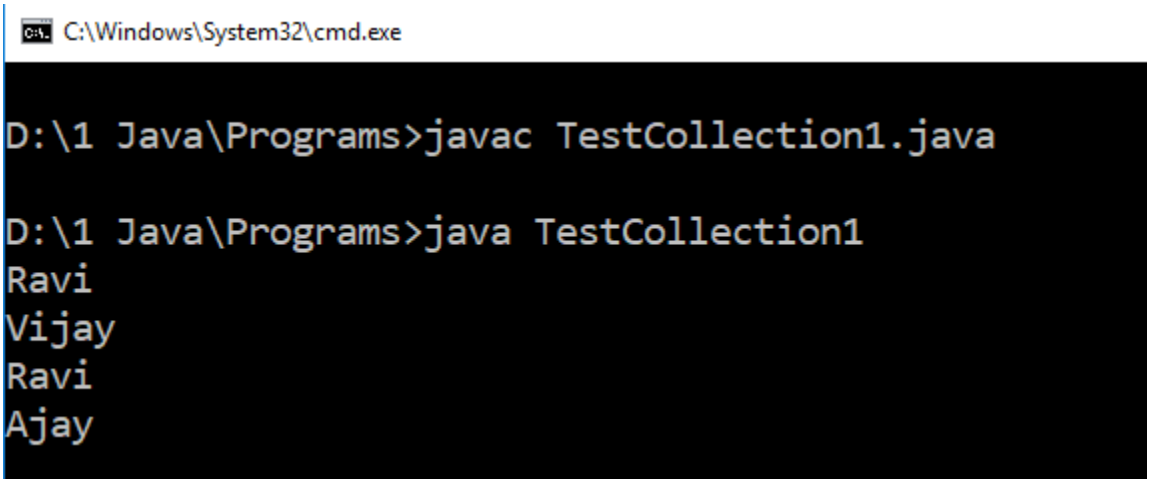
Let's see the new generic example of creating java collection.

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.
 }
 }
 }
```



//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  remove 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 removing 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.

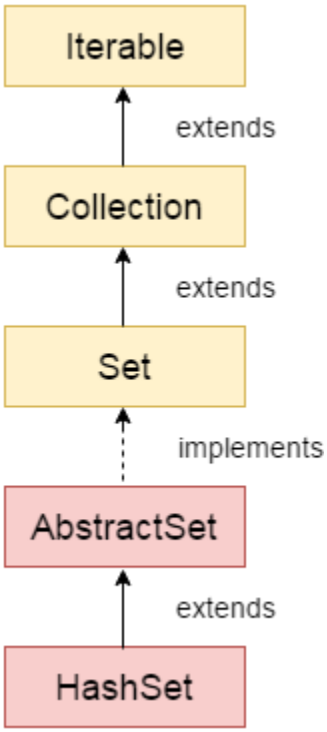
The important points about Java HashSet class are:

- 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:

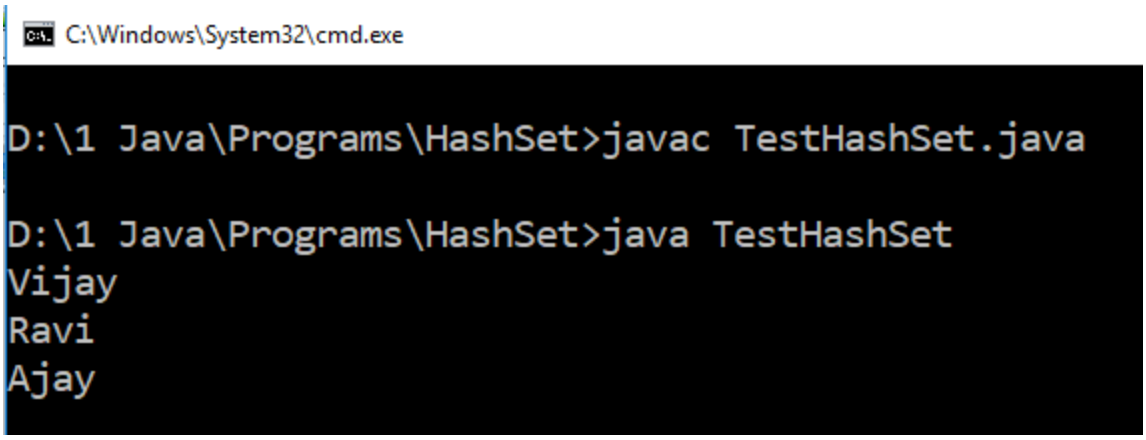
Constructor	Description
HashSet()	It is used to construct a default HashSet.
HashSet(Collection c)	It is used to initialize the hash set by using the elements of the collection c.
HashSet(int capacity)	It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity 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());
  }
 }
}
```



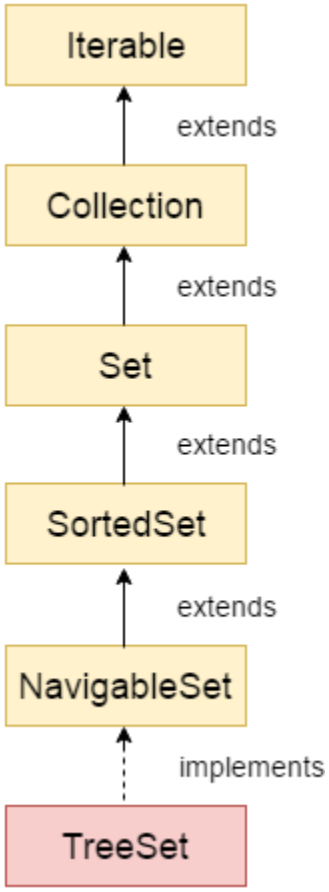
Java TreeSet class

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());
  }
 }
}
```