Collections in java

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

What is Collection in java

Collection represents a single unit of objects i.e. a group.

What is framework in java

- provides readymade architecture.
- represents set of classes and interface.
- is optional.

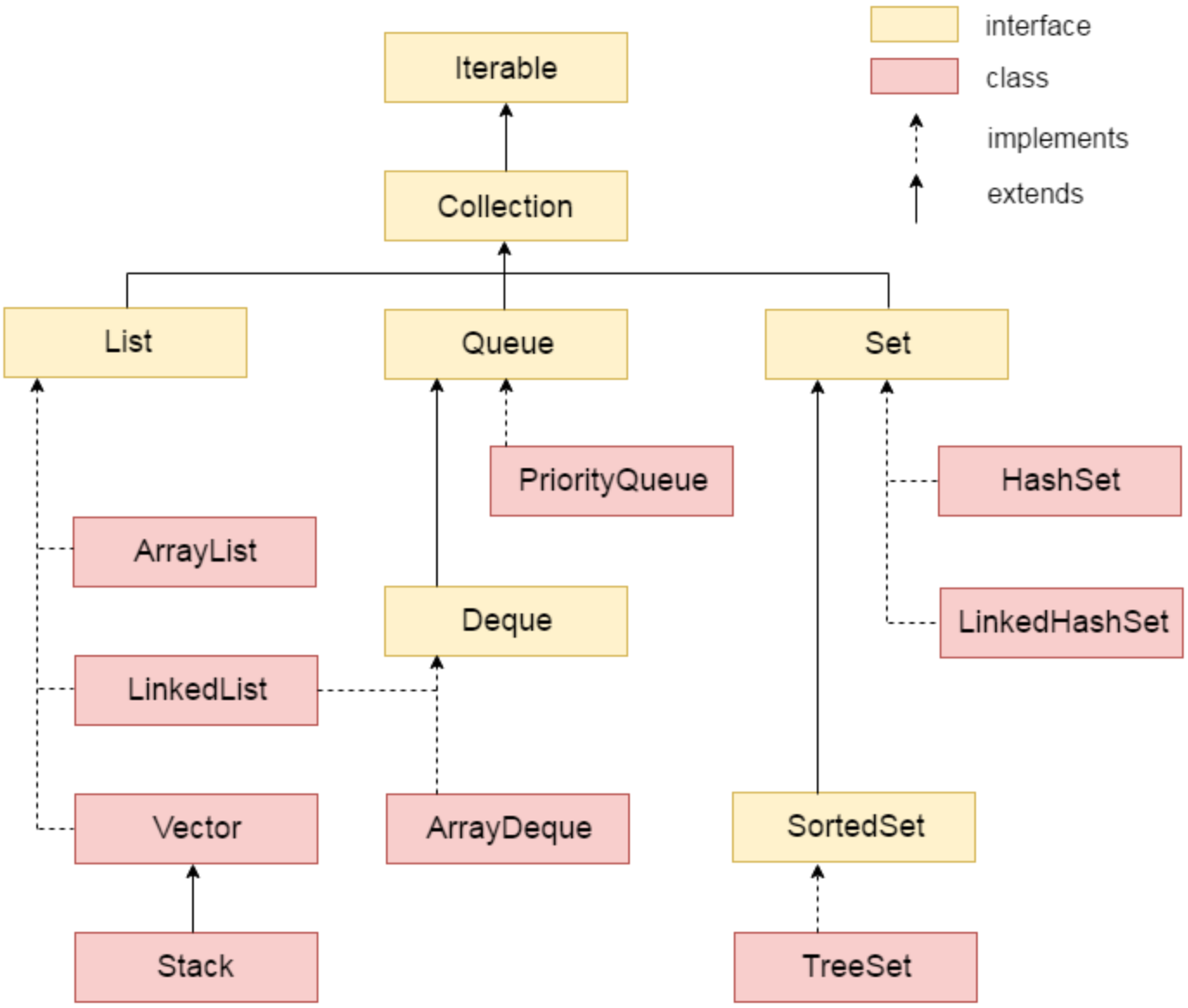
What is Collection framework

Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

1. Interfaces and its implementations i.e. classes
2. Algorithm

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.



Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

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:

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 List Interface

List Interface is the subinterface of Collection.It contains methods to insert and delete elements in index basis.It is a factory of ListIterator interface.

List Interface declaration

1. public interface List<E> extends Collection<E>

Methods of Java List Interface

Method	Description
void add(int index,Object element)	It is used to insert element into the invoking list at the index passed in the index.
boolean addAll(int index,Collection c)	It is used to insert all elements of c into the invoking list at the index passed in the index.
object get(int index)	It is used to return the object stored at the specified index within the invoking collection.
object set(int index,Object element)	It is used to assign element to the location specified by index within the invoking list.
object remove(int index)	It is used to remove the element at position index from the invoking list and return the deleted element.
ListIterator listIterator()	It is used to return an iterator to the start of the invoking list.

ListIterator listIterator(int index)	It is used to return an iterator to the invoking list that begins at the specified index.

Java List Example

```
1. import java.util.*;
2. public class ListExample{
3. public static void main(String args[]){
4. ArrayList<String> al=new ArrayList<String>();
5. al.add("Amit");
6. al.add("Vijay");
7. al.add("Kumar");
8. al.add(1,"Sachin");
9. System.out.println("Element at 2nd position: "+al.get(2));
10. for(String s:al){
11.  System.out.println(s);
12. }
13. }
14. }
```

Output:

```
Element at 2nd position: Vijay
Amit
Sachin
Vijay
Kumar
```

Java ListIterator Interface

ListIterator Interface is used to traverse the element in backward and forward direction.

ListIterator Interface declaration

```
1. public interface ListIterator<E> extends Iterator<E>
```

Methods of Java ListIterator Interface:

Method	Description
boolean hasNext()	This method return true if the list iterator has more elements when traversing the list in the forward direction.
Object next()	This method return the next element in the list and advances the cursor position.

boolean hasPrevious()	This method return true if this list iterator has more elements when traversing the list in the reverse direction.
Object previous()	This method return the previous element in the list and moves the cursor position backwards.

Example of ListIterator Interface

```
1. import java.util.*;
2. public class TestCollection8{
3. public static void main(String args[]){
4. ArrayList<String> al=new ArrayList<String>();
5. al.add("Amit");
6. al.add("Vijay");
7. al.add("Kumar");
8. al.add(1,"Sachin");
9. System.out.println("element at 2nd position: "+al.get(2));
10. ListIterator<String> itr=al.listIterator();
11. System.out.println("traversing elements in forward direction...");
12. while(itr.hasNext()){
13. System.out.println(itr.next());
14. }
15. System.out.println("traversing elements in backward direction...");
16. while(itr.hasPrevious()){
17. System.out.println(itr.previous());
18. }
19. }
20. }
```

Test it Now

Output:

```
element at 2nd position: Vijay
traversing elements in forward direction...
Amit
Sachin
Vijay
Kumar
traversing elements in backward direction...
Kumar
Vijay
Sachin
Amit
```

Example of ListIterator Interface: Book

```
1. import java.util.*;
```

```
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {
7.     this.id = id;
8.     this.name = name;
9.     this.author = author;
10.    this.publisher = publisher;
11.    this.quantity = quantity;
12. }
13. }
14. public class ListExample {
15. public static void main(String[] args) {
16.    //Creating list of Books
17.    List<Book> list=new ArrayList<Book>();
18.    //Creating Books
19.    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
20.    Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

21.    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22.    //Adding Books to list
23.    list.add(b1);
24.    list.add(b2);
25.    list.add(b3);
26.    //Traversing list
27.    for(Book b:list){
28.    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
29.    }
30. }
31. }
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

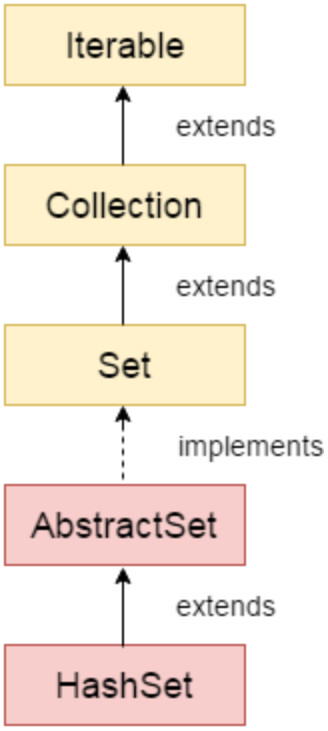
A Set is a Collection that cannot contain duplicate elements. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

The methods declared by Set are summarized in the following table −

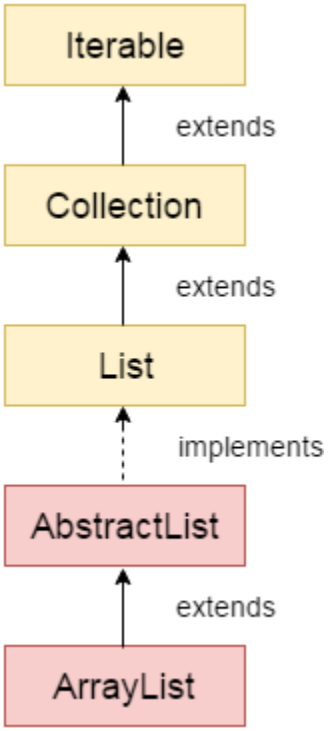
Sr.No.	Method & Description
1	**add( )** Adds an object to the collection.
2	**clear( )** Removes all objects from the collection.
3	**contains( )** Returns true if a specified object is an element within the collection.
4	**isEmpty( )** Returns true if the collection has no elements.
5	**iterator( )** Returns an Iterator object for the collection, which may be used to retrieve an object.
6	**remove( )** Removes a specified object from the collection.
7	**size( )** Returns the number of elements in the collection.

Difference between List and Set

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



Java ArrayList class



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

The important points about Java ArrayList class are:

- 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.
- In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Hierarchy of ArrayList class

As shown in above diagram, Java ArrayList class extends AbstractList class which implements List interface. The List interface extends Collection and Iterable interfaces in hierarchical order.

ArrayList class declaration

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

```
1. public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable
   , Serializable
```

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 ArrayList Example

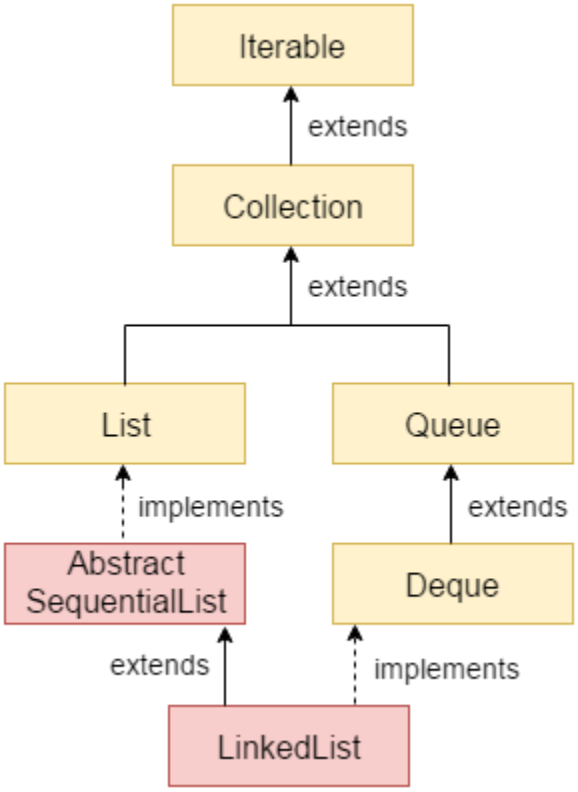
```
1.  import java.util.*;
2.  class TestCollection1{
3.   public static void main(String args[]){
4.    ArrayList<String> list=new ArrayList<String>();//Creating arraylist
5.    list.add("Ravi");//Adding object in arraylist
6.    list.add("Vijay");
7.    list.add("Ravi");
8.    list.add("Ajay");
9.    //Traversing list through Iterator
10.   Iterator itr=list.iterator();
11.   while(itr.hasNext()){
12.    System.out.println(itr.next());
13.   }
14.  }
```

15. }

output

```
Ravi
Vijay
Ravi
Ajay
```

Java LinkedList class



Java LinkedList class uses doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
- Java LinkedList class can be used as list, stack or queue.

Hierarchy of LinkedList class

As shown in above diagram, Java LinkedList class extends AbstractSequentialList class and implements List and Deque interfaces.

Doubly Linked List

In case of doubly linked list, we can add or remove elements from both side.

fig- doubly linked list

LinkedList class declaration

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

1. public class LinkedList<E> extends AbstractSequentialList<E> implements List<E>, Deque<E>, Cloneable, Serializable

Constructors of Java LinkedList

Constructor	Description
LinkedList()	It is used to construct an empty list.
LinkedList(Collection c)	It is used to construct a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Methods of Java LinkedList

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
void addFirst(Object o)	It is used to insert the given element at the beginning of a list.
void addLast(Object o)	It is used to append the given element to the end of a list.
int size()	It is used to return the number of elements in a list

boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean contains(Object o)	It is used to return true if the list contains a specified element.
boolean remove(Object o)	It is used to remove the first occurence of the specified element in a list.
Object getFirst()	It is used to return the first element in a list.
Object getLast()	It is used to return the last element in a list.
int indexOf(Object o)	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
int lastIndexOf(Object o)	It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element.

Java LinkedList Example

```
1.  import java.util.*;
2.  public class TestCollection7{
3.   public static void main(String args[]){
4.
5.    LinkedList<String> al=new LinkedList<String>();
6.    al.add("Ravi");
7.    al.add("Vijay");
8.    al.add("Ravi");
9.    al.add("Ajay");
10.
11.  Iterator<String> itr=al.iterator();
12.  while(itr.hasNext()){
13.   System.out.println(itr.next());
14.  }
15. }
16. }
```

Test it Now

```
Output:Ravi
       Vijay
       Ravi
       Ajay
```

Java LinkedList Example: Book

```
1.  import java.util.*;
```

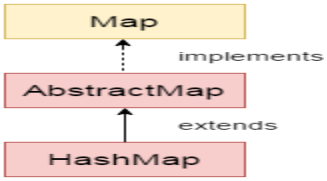
```
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {
7.     this.id = id;
8.     this.name = name;
9.     this.author = author;
10.    this.publisher = publisher;
11.    this.quantity = quantity;
12. }
13. }
14. public class LinkedListExample {
15. public static void main(String[] args) {
16.    //Creating list of Books
17.    List<Book> list=new LinkedList<Book>();
18.    //Creating Books
19.    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
20.    Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

21.    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22.    //Adding Books to list
23.    list.add(b1);
24.    list.add(b2);
25.    list.add(b3);
26.    //Traversing list
27.    for(Book b:list){
28.    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
29.    }
30. }
31. }
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

Java HashMap class



Java HashMap class implements the map interface by using a hashtable. It inherits AbstractMap class and implements Map interface.

The important points about Java HashMap class are:

- A HashMap contains values based on the key.
- It contains only unique elements.
- It may have one null key and multiple null values.
- It maintains no order.

Hierarchy of HashMap class

As shown in the above figure, HashMap class extends AbstractMap class and implements Map interface.

HashMap class declaration

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

```
1. public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Seriali
   zable
```

HashMap class Parameters

Let's see the Parameters for java.util.HashMap class.

- **K**: It is the type of keys maintained by this map.
- **V**: It is the type of mapped values.

Constructors of Java HashMap class

Constructor	Description
HashMap()	It is used to construct a default HashMap.
HashMap(Map m)	It is used to initializes the hash map by using the elements of the given Map object m.
HashMap(int capacity)	It is used to initializes the capacity of the hash map to the given integer value, capacity.
HashMap(int capacity, float fillRatio)	It is used to initialize both the capacity and fill ratio of the hash map by using its arguments.

Methods of Java HashMap class

Method	Description
void clear()	It is used to remove all of the mappings from this map.
boolean containsKey(Object key)	It is used to return true if this map contains a mapping for the specified key.
boolean containsValue(Object value)	It is used to return true if this map maps one or more keys to the specified value.
boolean isEmpty()	It is used to return true if this map contains no key-value mappings.
Object clone()	It is used to return a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
Set entrySet()	It is used to return a collection view of the mappings contained in this map.
Set keySet()	It is used to return a set view of the keys contained in this map.
Object put(Object key, Object value)	It is used to associate the specified value with the specified key in this map.
int size()	It is used to return the number of key-value mappings in this map.
Collection values()	It is used to return a collection view of the values contained in this map.

Java HashMap Example

```
1.  import java.util.*;
2.  class TestCollection13{
3.   public static void main(String args[]){
4.    HashMap<Integer,String> hm=new HashMap<Integer,String>();
5.    hm.put(100,"Amit");
6.    hm.put(101,"Vijay");
7.    hm.put(102,"Rahul");
8.    for(Map.Entry m:hm.entrySet()){
9.     System.out.println(m.getKey()+" "+m.getValue());
10.   }
11.  }
12. }
```

Test it Now

```
Output:102 Rahul
```

```
100 Amit
101 Vijay
```

Java HashMap Example: remove()

```
1. import java.util.*;
2. public class HashMapExample {
3.    public static void main(String args[]) {
4.    // create and populate hash map
5.    HashMap<Integer, String> map = new HashMap<Integer, String>();
6.    map.put(101,"Let us C");
7.    map.put(102, "Operating System");
8.    map.put(103, "Data Communication and Networking");
9.    System.out.println("Values before remove: "+ map);
10.   // Remove value for key 102
11.   map.remove(102);
12.   System.out.println("Values after remove: "+ map);
13.   }
14. }
```

Output:

```
Values before remove: {102=Operating System, 103=Data Communication and
Networking, 101=Let us C}
Values after remove: {103=Data Communication and Networking, 101=Let us C}
```

Difference between HashSet and HashMap

HashSet contains only values whereas HashMap contains entry(key and value).

Java HashMap Example: Book

```
1. import java.util.*;
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {
7.    this.id = id;
8.    this.name = name;
9.    this.author = author;
10.   this.publisher = publisher;
11.   this.quantity = quantity;
12. }
13. }
14. public class MapExample {
15. public static void main(String[] args) {
```

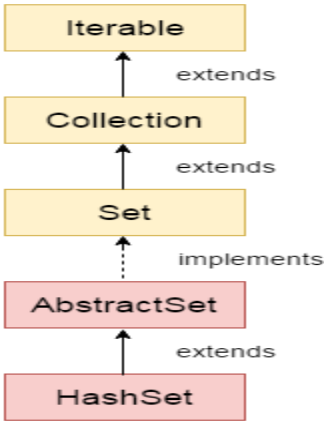
```
16.    //Creating map of Books
17.    Map<Integer,Book> map=new HashMap<Integer,Book>();
18.    //Creating Books
19.    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
20.    Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

21.    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22.    //Adding Books to map
23.    map.put(1,b1);
24.    map.put(2,b2);
25.    map.put(3,b3);
26.
27.    //Traversing map
28.    for(Map.Entry<Integer, Book> entry:map.entrySet()){
29.       int key=entry.getKey();
30.       Book b=entry.getValue();
31.       System.out.println(key+" Details:");
32.       System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
33.    }
34. }
35. }
```

Output:

```
1 Details:
101 Let us C Yashwant Kanetkar BPB 8
2 Details:
102 Data Communications & Networking Forouzan Mc Graw Hill 4
3 Details:
103 Operating System Galvin Wiley 6
```

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

The HashSet class extends AbstractSet class which implements Set interface. The Set interface inherits Collection and Iterable interfaces in hierarchical order.

HashSet class declaration

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

```
1. public class HashSet<E> extends AbstractSet<E> implements Set<E>, Cloneable, Serializable
```

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.

Java HashSet Example

```
1.  import java.util.*;
2.  class TestCollection9{
3.   public static void main(String args[]){
4.    //Creating HashSet and adding elements
5.    HashSet<String> set=new HashSet<String>();
6.    set.add("Ravi");
7.    set.add("Vijay");
8.    set.add("Ravi");
9.    set.add("Ajay");
10.   //Traversing elements
11.   Iterator<String> itr=set.iterator();
12.   while(itr.hasNext()){
13.    System.out.println(itr.next());
14.   }
15.  }
16. }
```

Test it Now

```
Ajay
Vijay
Ravi
```

Java HashSet Example: Book

Let's see a HashSet example where we are adding books to set and printing all the books.

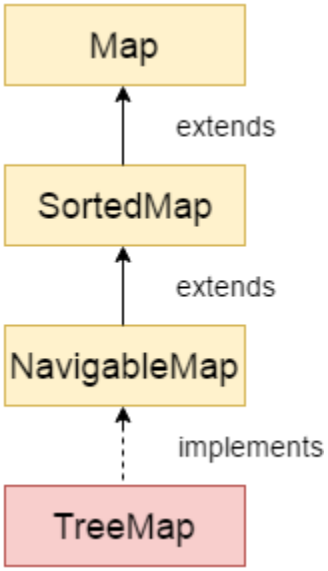
```
1. import java.util.*;
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {
7.     this.id = id;
8.     this.name = name;
9.     this.author = author;
10.    this.publisher = publisher;
11.    this.quantity = quantity;
12. }
13. }
14. public class HashSetExample {
15. public static void main(String[] args) {
16.    HashSet<Book> set=new HashSet<Book>();
17.    //Creating Books
18.    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
19.    Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

20.    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
21.    //Adding Books to HashSet
22.    set.add(b1);
23.    set.add(b2);
24.    set.add(b3);
25.    //Traversing HashSet
26.    for(Book b:set){
27.    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
28.    }
29. }
30. }
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

Java TreeMap class



Java TreeMap class implements the Map interface by using a tree. It provides an efficient means of storing key/value pairs in sorted order.

The important points about Java TreeMap class are:

- A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- It contains only unique elements.
- It cannot have null key but can have multiple null values.
- It is same as HashMap instead maintains ascending order.

TreeMap class declaration

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

```
1. public class TreeMap<K,V> extends AbstractMap<K,V> implements NavigableMap<K,V>, Clonea
   ble, Serializable
```

TreeMap class Parameters

Let's see the Parameters for java.util.TreeMap class.

- **K**: It is the type of keys maintained by this map.
- **V**: It is the type of mapped values.

Constructors of Java TreeMap class

Constructor	Description

TreeMap()	It is used to construct an empty tree map that will be sorted using the natural order of its key.
TreeMap(Comparator comp)	It is used to construct an empty tree-based map that will be sorted using the comparator comp.
TreeMap(Map m)	It is used to initialize a tree map with the entries from **m**, which will be sorted using the natural order of the keys.
TreeMap(SortedMap sm)	It is used to initialize a tree map with the entries from the SortedMap **sm**, which will be sorted in the same order as **sm.**

Methods of Java TreeMap class

Method	Description
boolean containsKey(Object key)	It is used to return true if this map contains a mapping for the specified key.
boolean containsValue(Object value)	It is used to return true if this map maps one or more keys to the specified value.
Object firstKey()	It is used to return the first (lowest) key currently in this sorted map.
Object get(Object key)	It is used to return the value to which this map maps the specified key.
Object lastKey()	It is used to return the last (highest) key currently in this sorted map.
Object remove(Object key)	It is used to remove the mapping for this key from this TreeMap if present.
void putAll(Map map)	It is used to copy all of the mappings from the specified map to this map.
Set entrySet()	It is used to return a set view of the mappings contained in this map.
int size()	It is used to return the number of key-value mappings in this map.
Collection values()	It is used to return a collection view of the values contained in this map.

Java TreeMap Example:

```
1.  import java.util.*;
2.  class TestCollection15{
3.   public static void main(String args[]){
4.    TreeMap<Integer,String> hm=new TreeMap<Integer,String>();
5.    hm.put(100,"Amit");
6.    hm.put(102,"Ravi");
7.    hm.put(101,"Vijay");
8.    hm.put(103,"Rahul");
9.    for(Map.Entry m:hm.entrySet()){
10.    System.out.println(m.getKey()+" "+m.getValue());
11.   }
12.  }
13. }
```

Test it Now

```
Output:100 Amit
       101 Vijay
       102 Ravi
       103 Rahul
```

Java TreeMap Example: remove()

```
1.  import java.util.*;
2.  public class TreeMapExample {
3.     public static void main(String args[]) {
4.     // Create and populate tree map
5.     Map<Integer, String> map = new TreeMap<Integer, String>();
6.     map.put(102,"Let us C");
7.     map.put(103, "Operating System");
8.     map.put(101, "Data Communication and Networking");
9.     System.out.println("Values before remove: "+ map);
10.    // Remove value for key 102
11.    map.remove(102);
12.    System.out.println("Values after remove: "+ map);
13.    }
14. }
```

Output:

```
Values before remove: {101=Data Communication and Networking, 102=Let us C,
103=Operating System}
Values after remove: {101=Data Communication and Networking, 103=Operating
System}
```

What is difference between HashMap and TreeMap?

HashMap	TreeMap
1) HashMap can contain one null key.	TreeMap can not contain any null key.
2) HashMap maintains no order.	TreeMap maintains ascending order.

Java TreeMap Example: Book

```
1. import java.util.*;
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {
7.     this.id = id;
8.     this.name = name;
9.     this.author = author;
10.    this.publisher = publisher;
11.    this.quantity = quantity;
12. }
13. }
14. public class MapExample {
15. public static void main(String[] args) {
16.    //Creating map of Books
17.    Map<Integer,Book> map=new TreeMap<Integer,Book>();
18.    //Creating Books
19.    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
20.    Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

21.    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22.    //Adding Books to map
23.    map.put(2,b2);
24.    map.put(1,b1);
25.    map.put(3,b3);
26.
27.    //Traversing map
28.    for(Map.Entry<Integer, Book> entry:map.entrySet()){
29.       int key=entry.getKey();
30.       Book b=entry.getValue();
31.       System.out.println(key+" Details:");
32.       System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
33.    }
34. }
35. }
```

Output:

1 Details:
101 Let us C Yashwant Kanetkar BPB 8
2 Details:
102 Data Communications & Networking Forouzan Mc Graw Hill 4
3 Details:
103 Operating System Galvin Wiley 6