**UNIT-V: Java Collections Framework**

* Collections overview
* collection classes: ArrayList, LinkedList, HashSet, LinkedHashSet, TreeSet,HashMap.
* Accessing a Collection: Iterator and for-each.

**Collections in Java**

The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

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

**What is a framework in Java**

* It provides readymade architecture.
* It represents a set of classes and interfaces.

**What is Collection framework**

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

* Interfaces and its implementations, i.e., classes
* Algorithm

**Hierarchy of Collection Framework**

The java.util package contains all the classes and interfaces for the 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(E e) | It is used to insert an element in this collection. |
| 2 | Public boolean addAll(Collection<? extends E> c) | It is used to insert the specified collection elements in the invoking collection. |
| 3 | public boolean remove(Object element) | It is used to delete an element from the collection. |
| 4 | public boolean removeAll(Collection<?> c) | It is used to delete all the elements of the specified collection from the invoking collection. |
| 7 | public int size() | It returns the total number of elements in the collection. |
| 8 | public void clear() | It removes the total number of elements from the collection. |
| 9 | public boolean contains(Object element) | It is used to search an element. |
| 10 | public boolean containsAll(Collection<?> c) | It is used to search the specified collection in the collection. |
| 11 | public Iterator iterator() | It returns an iterator. |
| 12 | public Object[] toArray() | It converts collection into array. |
| 14 | public boolean isEmpty() | It checks if collection is empty. |
| 18 | public boolean equals(Object element) | It matches two collections. |
| 19 | public int hashCode() | It returns the hash code number of the collection. |

**Iterator interface**

|  |
| --- |
| Iterator interface provides the facility of iterating the elements in a 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 the iterator has more elements otherwise it returns false. |
| 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 less used. |

**Iterable Interface**

The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e.,

1. Iterator<T> iterator()

It returns the iterator over the elements of type T.

**Collection Interface**

The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.

Some of the methods of Collection interface are Boolean add ( Object obj), Boolean addAll ( Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.

**List Interface**

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

1. List <data-type> list1= **new** ArrayList();
2. List <data-type> list2 = **new** LinkedList();
3. List <data-type> list3 = **new** Vector();
4. List <data-type> list4 = **new** Stack();

There are various methods in List interface that can be used to insert, delete, and access the elements from the list.

**ArrayList**

Java ArrayList class uses a dynamic array for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the java.util package. It is like the Vector in C++.

The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of the List interface here. The ArrayList maintains the insertion order internally.It inherits the 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](https://www.javatpoint.com/synchronization-in-java).
* Java ArrayList allows random access because the array works on an index basis.
* In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
* We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases. For example:

1. ArrayList<**int**> al = ArrayList<**int**>(); // does not work
2. ArrayList<Integer> al = **new** ArrayList<Integer>(); // works fine

* Java ArrayList gets initialized by the size. The size is dynamic in the array list, which varies according to the elements getting added or removed from the list.

Hierarchy of ArrayList class

As shown in the above diagram, the Java ArrayList class extends AbstractList class which implements the List interface. The List interface extends the [Collection](https://www.javatpoint.com/collections-in-java) 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 ArrayList

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| ArrayList() | It is used to build an empty array list. |
| ArrayList(Collection<? extends E> 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 ArrayList

|  |  |
| --- | --- |
| **Method** | **Description** |
| void [add](https://www.javatpoint.com/java-arraylist-add-method)(int index, E element) | It is used to insert the specified element at the specified position in a list. |
| boolean [add](https://www.javatpoint.com/java-arraylist-add-method)(E e) | It is used to append the specified element at the end of a list. |
| boolean [addAll](https://www.javatpoint.com/java-arraylist-addall-method)(Collection<? extends E> 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. |
| boolean [addAll](https://www.javatpoint.com/java-arraylist-addall-method)(int index, Collection<? extends E> c) | It is used to append all the elements in the specified collection, starting at the specified position of the list. |
| void [clear](https://www.javatpoint.com/java-arraylist-clear-method)() | It is used to remove all of the elements from this list. |
| E get(int index) | It is used to fetch the element from the particular position of the list. |
| boolean isEmpty() | It returns true if the list is empty, otherwise false. |
| 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. |
| <T> T[] toArray(T[] a) | It is used to return an array containing all of the elements in this list in the correct order. |
| Object clone() | It is used to return a shallow copy of an ArrayList. |
| boolean contains(Object o) | It returns true if the list contains the specified element. |
| 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. |
| E remove(int index) | It is used to remove the element present at the specified position in the list. |
| boolean [remove](https://www.javatpoint.com/java-arraylist-remove-method)(Object o) | It is used to remove the first occurrence of the specified element. |
| boolean [removeAll](https://www.javatpoint.com/java-arraylist-removeall-method)(Collection<?> c) | It is used to remove all the elements from the list. |
| boolean removeIf(Predicate<? super E> filter) | It is used to remove all the elements from the list that satisfies the given predicate. |
| protected void [removeRange](https://www.javatpoint.com/java-arraylist-removerange-method)(int fromIndex, int toIndex) | It is used to remove all the elements lies within the given range. |
| void replaceAll(UnaryOperator<E> operator) | It is used to replace all the elements from the list with the specified element. |
| E set(int index, E element) | It is used to replace the specified element in the list, present at the specified position. |
| void sort(Comparator<? super E> c) | It is used to sort the elements of the list on the basis of the specified comparator. |
| List<E> subList(int fromIndex, int toIndex) | It is used to fetch all the elements that lies within the given range. |
| int size() | It is used to return the number of elements present in the list. |

Java ArrayList Example

**import** java.util.\*;

**class** ArrayListExample1{

**public** **static** **void** main(String args[]){

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

      list.add("Mango");//Adding object in arraylist

      list.add("Apple");

      list.add("Banana");

      list.add("Grapes");

      //Printing the arraylist object

      System.out.println(list);

 }

}

**Iterating ArrayList using Iterator**

**import** java.util.\*;

**class** ArrayListExample2{

**public** **static** **void** main(String args[]){

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

  list.add("Mango");//Adding object in arraylist

  list.add("Apple");

  list.add("Banana");

  list.add("Grapes");

  //Traversing list through Iterator

  Iterator itr=list.iterator();//getting the Iterator

**while**(itr.hasNext()){//check if iterator has the elements

   System.out.println(itr.next());//printing the element and move to next

  }

 }

}

**Iterating ArrayList using For-each loop**

**import** java.util.\*;

**public** **class** ArrayListExample3{

**public** **static** **void** main(String args[]){

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

  list.add("Mango");//Adding object in arraylist

  list.add("Apple");

  list.add("Banana");

  list.add("Grapes");

  //Traversing list through for-each loop

**for**(String fruit:list)

    System.out.println(fruit);

 }

}

**Get and Set ArrayList**

The *get() method* returns the element at the specified index, whereas the *set() method* changes the element.

**import** java.util.\*;

**class** ArrayListExample4{

**public** **static** **void** main(String args[]){

  ArrayList<String> al=**new** ArrayList<String>();

  al.add("Mango");

  al.add("Apple");

  al.add("Banana");

  al.add("Grapes");

  //accessing the element

  System.out.println("Returning element: "+al.get(1));//it will return the 2nd element, because index starts from 0

  //changing the element

  al.set(1,"Dates");

  //Traversing list

**for**(String fruit:al)

    System.out.println(fruit);

 }

}

How to Sort ArrayList

The *java.util* package provides a utility class **Collections**, which has the static method sort(). Using the **Collections.sort()** method, we can easily sort the ArrayList.

**import** java.util.\*;

**class** SortArrayList{

**public** **static** **void** main(String args[]){

  //Creating a list of fruits

  List<String> list1=**new** ArrayList<String>();

  list1.add("Mango");

  list1.add("Apple");

  list1.add("Banana");

  list1.add("Grapes");

  //Sorting the list

  Collections.sort(list1);

   //Traversing list through the for-each loop

**for**(String fruit:list1)

    System.out.println(fruit);

 System.out.println("Sorting numbers...");

  //Creating a list of numbers

  List<Integer> list2=**new** ArrayList<Integer>();

  list2.add(21);

  list2.add(11);

  list2.add(51);

  list2.add(1);

  //Sorting the list

  Collections.sort(list2);

   //Traversing list through the for-each loop

**for**(Integer number:list2)

    System.out.println(number);

 }

}

**User-defined class objects in Java 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** ArrayList5{

**public** **static** **void** main(String args[]){

  //Creating user-defined class objects

  Student s1=**new** Student(101,"Sonoo",23);

  Student s2=**new** Student(102,"Ravi",21);

  Student s2=**new** Student(103,"Hanumat",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();

    System.out.println(st.rollno+" "+st.name+" "+st.age);

  }

 }

}

**LinkedList**

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

* 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 occur.
* Java LinkedList class can be used as a list, stack or queue.

Hierarchy of LinkedList class

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

Doubly Linked List

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



LinkedList class declaration

**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<? extends E> 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** |
| boolean add(E e) | It is used to append the specified element to the end of a list. |
| void add(int index, E element) | It is used to insert the specified element at the specified position index in a list. |
| boolean addAll(Collection<? extends E> 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. |
| boolean addAll(Collection<? extends E> 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. |
| boolean addAll(int index, Collection<? extends E> c) | It is used to append all the elements in the specified collection, starting at the specified position of the list. |
| void addFirst(E e) | It is used to insert the given element at the beginning of a list. |
| void addLast(E e) | It is used to append the given element to the end of a list. |
| void clear() | It is used to remove all the elements from a list. |
| Object clone() | It is used to return a shallow copy of an ArrayList. |
| boolean contains(Object o) | It is used to return true if a list contains a specified element. |
| Iterator<E> descendingIterator() | It is used to return an iterator over the elements in a deque in reverse sequential order. |
| E element() | It is used to retrieve the first element of a list. |
| E get(int index) | It is used to return the element at the specified position in a list. |
| E getFirst() | It is used to return the first element in a list. |
| E 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. |
| ListIterator<E> listIterator(int index) | It is used to return a list-iterator of the elements in proper sequence, starting at the specified position in the list. |
| boolean offer(E e) | It adds the specified element as the last element of a list. |
| boolean offerFirst(E e) | It inserts the specified element at the front of a list. |
| boolean offerLast(E e) | It inserts the specified element at the end of a list. |
| E peek() | It retrieves the first element of a list |
| E peekFirst() | It retrieves the first element of a list or returns null if a list is empty. |
| E peekLast() | It retrieves the last element of a list or returns null if a list is empty. |
| E poll() | It retrieves and removes the first element of a list. |
| E pollFirst() | It retrieves and removes the first element of a list, or returns null if a list is empty. |
| E pollLast() | It retrieves and removes the last element of a list, or returns null if a list is empty. |
| E pop() | It pops an element from the stack represented by a list. |
| void push(E e) | It pushes an element onto the stack represented by a list. |
| E remove() | It is used to retrieve and removes the first element of a list. |
| E remove(int index) | It is used to remove the element at the specified position in a list. |
| boolean remove(Object o) | It is used to remove the first occurrence of the specified element in a list. |
| E removeFirst() | It removes and returns the first element from a list. |
| boolean removeFirstOccurrence(Object o) | It is used to remove the first occurrence of the specified element in a list (when traversing the list from head to tail). |
| E removeLast() | It removes and returns the last element from a list. |
| boolean removeLastOccurrence(Object o) | It removes the last occurrence of the specified element in a list (when traversing the list from head to tail). |
| E set(int index, E element) | It replaces the element at the specified position in a list with the specified element. |
| Object[] toArray() | It is used to return an array containing all the elements in a list in proper sequence (from first to the last element). |
| <T> T[] toArray(T[] a) | It returns an array containing all the elements in the proper sequence (from first to the last element); the runtime type of the returned array is that of the specified array. |
| int size() | It is used to return the number of elements in a list. |

Java LinkedList Example

**import** java.util.\*;

**class** LinkedList1{

**public** **static** **void** main(String args[]){

  LinkedList<String> al=**new** LinkedList<String>();

  al.add("Ravi");

  al.add("Vijay");

  al.add("Ravi");

  al.add("Ajay");

  Iterator<String> itr=al.iterator();

**while**(itr.hasNext()){

   System.out.println(itr.next());

  }

 }

}

Java LinkedList example to add elements

Here, we see different ways to add elements.

**import** java.util.\*;

**class** LinkedList2{

**public** **static** **void** main(String args[]){

 LinkedList<String> ll=**new** LinkedList<String>();

           System.out.println("Initial list of elements: "+ll);

           ll.add("Ravi");

           ll.add("Vijay");

           ll.add("Ajay");

           System.out.println("After invoking add(E e) method: "+ll);

           //Adding an element at the specific position

           ll.add(1, "Gaurav");

           System.out.println("After invoking add(int index, E element) method: "+ll);

           LinkedList<String> ll2=**new** LinkedList<String>();

           ll2.add("Sonoo");

           ll2.add("Hanumat");

           //Adding second list elements to the first list

           ll.addAll(ll2);

           System.out.println("After invoking addAll(Collection<? extends E> c) method: "+ll);

           LinkedList<String> ll3=**new** LinkedList<String>();

           ll3.add("John");

           ll3.add("Rahul");

           //Adding second list elements to the first list at specific position

           ll.addAll(1, ll3);

           System.out.println("After invoking addAll(int index, Collection<? extends E> c) method: "+ll);

           //Adding an element at the first position

           ll.addFirst("Lokesh");

           System.out.println("After invoking addFirst(E e) method: "+ll);

           //Adding an element at the last position

           ll.addLast("Harsh");

           System.out.println("After invoking addLast(E e) method: "+ll);

 }

}

Java LinkedList example to remove elements

**import** java.util.\*;

**class** LinkedList3 {

**public** **static** **void** main(String [] args)

        {

           LinkedList<String> ll=**new** LinkedList<String>();

           ll.add("Ravi");

           ll.add("Vijay");

           ll.add("Ajay");

           ll.add("Anuj");

           ll.add("Gaurav");

           ll.add("Harsh");

           ll.add("Virat");

           ll.add("Gaurav");

           ll.add("Harsh");

           ll.add("Amit");

           System.out.println("Initial list of elements: "+ll);

         //Removing specific element from arraylist

              ll.remove("Vijay");

              System.out.println("After invoking remove(object) method: "+ll);

         //Removing element on the basis of specific position

              ll.remove(0);

              System.out.println("After invoking remove(index) method: "+ll);

              LinkedList<String> ll2=**new** LinkedList<String>();

              ll2.add("Ravi");

              ll2.add("Hanumat");

         // Adding new elements to arraylist

              ll.addAll(ll2);

              System.out.println("Updated list : "+ll);

         //Removing all the new elements from arraylist

              ll.removeAll(ll2);

              System.out.println("After invoking removeAll() method: "+ll);

         //Removing first element from the list

              ll.removeFirst();

              System.out.println("After invoking removeFirst() method: "+ll);

          //Removing first element from the list

              ll.removeLast();

              System.out.println("After invoking removeLast() method: "+ll);

          //Removing first occurrence of element from the list

              ll.removeFirstOccurrence("Gaurav");

              System.out.println("After invoking removeFirstOccurrence() method: "+ll);

          //Removing last occurrence of element from the list

              ll.removeLastOccurrence("Harsh");

              System.out.println("After invoking removeLastOccurrence() method: "+ll);

              //Removing all the elements available in the list

              ll.clear();

              System.out.println("After invoking clear() method: "+ll);

       }

    }

Java LinkedList Example to reverse a list of elements

**import** java.util.\*;

**class** LinkedList4{

**public** **static** **void** main(String args[]){

  LinkedList<String> ll=**new** LinkedList<String>();

           ll.add("Ravi");

           ll.add("Vijay");

           ll.add("Ajay");

           //Traversing the list of elements in reverse order

           Iterator i=ll.descendingIterator();

**while**(i.hasNext())

           {

               System.out.println(i.next());

           }

 }

}

**Java LinkedList Example: Book**

**import** java.util.\*;

**class** Book {

**int** id;

String name,author,publisher;

**int** quantity;

**public** Book(**int** id, String name, String author, String publisher, **int** quantity) {

**this**.id = id;

**this**.name = name;

**this**.author = author;

**this**.publisher = publisher;

**this**.quantity = quantity;

}

}

**class** LinkedListExample {

**public** **static** **void** main(String[] args) {

    //Creating list of Books

    List<Book> list=**new** LinkedList<Book>();

    //Creating Books

    Book b1=**new** Book(101,"Let us C","Yashwant Kanetkar","BPB",8);

    Book b2=**new** Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

    Book b3=**new** Book(103,"Operating System","Galvin","Wiley",6);

    //Adding Books to list

    list.add(b1);

    list.add(b2);

    list.add(b3);

    //Traversing list

**for**(Book b:list){

    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

    }

}

}

**HashSet**

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.

* HashSet stores the elements by using a mechanism called **hashing.**
* HashSet contains unique elements only.
* HashSet allows null value.
* HashSet class is non synchronized.
* HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
* HashSet is the best approach for search operations.
* The initial default capacity of HashSet is 16, and the load factor is 0.75.

Difference between List and Set

A 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**

**public** **class** HashSet<E> **extends** AbstractSet<E> **implements** Set<E>, Cloneable, Serializable

Constructors of Java HashSet class

|  |  |  |
| --- | --- | --- |
| **SN** | **Constructor** | **Description** |
| 1) | HashSet() | It is used to construct a default HashSet. |
| 2) | 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. |
| 3) | HashSet(int capacity, float loadFactor) | It is used to initialize the capacity of the hash set to the given integer value capacity and the specified load factor. |
| 4) | HashSet(Collection<? extends E> c) | It is used to initialize the hash set by using the elements of the collection c. |

**Methods of Java HashSet class**

|  |  |  |  |
| --- | --- | --- | --- |
| **SN** | **Modifier & Type** | **Method** | **Description** |
| 1) | boolean | [add(E e)](https://www.javatpoint.com/java-hashset-add-method) | It is used to add the specified element to this set if it is not already present. |
| 2) | void | [clear()](https://www.javatpoint.com/java-hashset-clear-method) | It is used to remove all of the elements from the set. |
| 3) | object | [clone()](https://www.javatpoint.com/java-hashset-clone-method) | It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned. |
| 4) | boolean | [contains(Object o)](https://www.javatpoint.com/java-hashset-contains-method) | It is used to return true if this set contains the specified element. |
| 5) | boolean | [isEmpty()](https://www.javatpoint.com/java-hashset-isempty-method) | It is used to return true if this set contains no elements. |
| 6) | Iterator<E> | [iterator()](https://www.javatpoint.com/java-hashset-iterator-method) | It is used to return an iterator over the elements in this set. |
| 7) | boolean | [remove(Object o)](https://www.javatpoint.com/java-hashset-remove-method) | It is used to remove the specified element from this set if it is present. |
| 8) | int | [size()](https://www.javatpoint.com/java-hashset-size-method) | It is used to return the number of elements in the set. |

**Java HashSet Example**

Let's see a simple example of HashSet. Notice, the elements iterate in an unordered collection.

**import** java.util.\*;

**class** HashSet1{

**public** **static** **void** main(String args[]){

  //Creating HashSet and adding elements

    HashSet<String> set=**new** HashSet();

           set.add("One");

           set.add("Two");

           set.add("Three");

           set.add("Four");

           set.add("Five");

           Iterator<String> i=set.iterator();

**while**(i.hasNext())

           {

           System.out.println(i.next());

           }

 }

}

**Java HashSet example ignoring duplicate elements**

In this example, we see that HashSet doesn't allow duplicate elements.

**import** java.util.\*;

**class** HashSet2{

**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 HashSet example to remove elements**

Here, we see different ways to remove an element.

**import** java.util.\*;

**class** HashSet3{

**public** **static** **void** main(String args[]){

  HashSet<String> set=**new** HashSet<String>();

           set.add("Ravi");

           set.add("Vijay");

           set.add("Arun");

           set.add("Sumit");

           System.out.println("An initial list of elements: "+set);

           //Removing specific element from HashSet

           set.remove("Ravi");

           System.out.println("After invoking remove(object) method: "+set);

           HashSet<String> set1=**new** HashSet<String>();

           set1.add("Ajay");

           set1.add("Gaurav");

           set.addAll(set1);

           System.out.println("Updated List: "+set);

           //Removing all the new elements from HashSet

           set.removeAll(set1);

           System.out.println("After invoking removeAll() method: "+set);

           //Removing elements on the basis of specified condition

           set.removeIf(str->str.contains("Vijay"));

           System.out.println("After invoking removeIf() method: "+set);

           //Removing all the elements available in the set

           set.clear();

           System.out.println("After invoking clear() method: "+set);

 }

}

**Java HashSet from another Collection**

**import** java.util.\*;

**class** HashSet4{

**public** **static** **void** main(String args[]){

   ArrayList<String> list=**new** ArrayList<String>();

           list.add("Ravi");

           list.add("Vijay");

           list.add("Ajay");

           HashSet<String> set=**new** HashSet(list);

           set.add("Gaurav");

           Iterator<String> i=set.iterator();

**while**(i.hasNext())

           {

           System.out.println(i.next());

           }

 }

}

**Java HashSet Example: Book**

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

**import** java.util.\*;

**class** Book {

**int** id;

String name,author,publisher;

**int** quantity;

**public** Book(**int** id, String name, String author, String publisher, **int** quantity) {

**this**.id = id;

**this**.name = name;

**this**.author = author;

**this**.publisher = publisher;

**this**.quantity = quantity;

}

}

**class** HashSetExample {

**public** **static** **void** main(String[] args) {

    HashSet<Book> set=**new** HashSet<Book>();

    //Creating Books

    Book b1=**new** Book(101,"Let us C","Yashwant Kanetkar","BPB",8);

    Book b2=**new** Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

    Book b3=**new** Book(103,"Operating System","Galvin","Wiley",6);

    //Adding Books to HashSet

    set.add(b1);

    set.add(b2);

    set.add(b3);

    //Traversing HashSet

**for**(Book b:set){

    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

    }

}

}

**LinkedHashSet**

Java LinkedHashSet class is a Hashtable and Linked list implementation of the Set interface. It inherits the HashSet class and implements the Set interface.

* Java LinkedHashSet class contains unique elements only like HashSet.
* Java LinkedHashSet class provides all optional set operations and permits null elements.
* Java LinkedHashSet class is non-synchronized.
* Java LinkedHashSet class maintains insertion order.

Hierarchy of LinkedHashSet class

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

LinkedHashSet Class Declaration

**public** **class** LinkedHashSet<E> **extends** HashSet<E> **implements** Set<E>, Cloneable, Serializable

Constructors of Java LinkedHashSet 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. |
| LinkedHashSet(int capacity) | It is used to initialize the capacity of the linked hash set to the given integer value capacity. |
| LinkedHashSet(int capacity, float fillRatio) | It is used to initialize both the capacity and the fill ratio (also called load capacity) of the hash set from its argument. |

Java LinkedHashSet Example

**import** java.util.\*;

**class** LinkedHashSet1{

**public** **static** **void** main(String args[]){

 //Creating HashSet and adding elements

        LinkedHashSet<String> set=**new** LinkedHashSet();

               set.add("One");

               set.add("Two");

               set.add("Three");

               set.add("Four");

               set.add("Five");

               Iterator<String> i=set.iterator();

**while**(i.hasNext())

               {

               System.out.println(i.next());

               }

 }

}

Java LinkedHashSet example ignoring duplicate Elements

**import** java.util.\*;

**class** LinkedHashSet2{

**public** **static** **void** main(String args[]){

  LinkedHashSet<String> al=**new** LinkedHashSet<String>();

  al.add("Ravi");

  al.add("Vijay");

  al.add("Ravi");

  al.add("Ajay");

  Iterator<String> itr=al.iterator();

**while**(itr.hasNext()){

   System.out.println(itr.next());

  }

 }

}

Remove Elements Using LinkeHashSet Class

**import** java.util.\*;

**public** **class** LinkedHashSet3

{

**public** **static** **void** main(String argvs[])

{

LinkedHashSet<String> lhs = **new** LinkedHashSet<String>();

lhs.add("Java");

lhs.add("T");

lhs.add("Point");

lhs.add("Good");

lhs.add("Website");

System.out.println("The hash set is: " + lhs);

System.out.println(lhs.remove("Good"));

System.out.println("After removing the element, the hash set is: " + lhs);

System.out.println(lhs.remove("For"));

}

}

Java LinkedHashSet Example: Book

**import** java.util.\*;

**class** Book {

**int** id;

String name,author,publisher;

**int** quantity;

**public** Book(**int** id, String name, String author, String publisher, **int** quantity) {

**this**.id = id;

**this**.name = name;

**this**.author = author;

**this**.publisher = publisher;

**this**.quantity = quantity;

}

}

**class** LinkedHashSetExample {

**public** **static** **void** main(String[] args) {

    LinkedHashSet<Book> hs=**new** LinkedHashSet<Book>();

    //Creating Books

    Book b1=**new** Book(101,"Let us C","Yashwant Kanetkar","BPB",8);

    Book b2=**new** Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

    Book b3=**new** Book(103,"Operating System","Galvin","Wiley",6);

    //Adding Books to hash table

    hs.add(b1);

    hs.add(b2);

    hs.add(b3);

    //Traversing hash table

**for**(Book b:hs){

    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

    }

}

}

**TreeSet**

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

* Java TreeSet class contains unique elements only like HashSet.
* Java TreeSet class access and retrieval times are quite fast.
* Java TreeSet class doesn't allow null elements.
* Java TreeSet class is non-synchronized.
* Java TreeSet class maintains ascending order.
* The TreeSet can only allow those generic types that are comparable. For example The Comparable interface is being implemented by the StringBuffer class.

Hierarchy of TreeSet class

As shown in the above diagram, the Java TreeSet class implements the NavigableSet interface. The NavigableSet interface extends SortedSet, Set, Collection and Iterable interfaces in hierarchical order.

TreeSet Class Declaration

**public** **class** TreeSet<E> **extends** AbstractSet<E> **implements** NavigableSet<E>, Cloneable, Serializable

Constructors of Java TreeSet Class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| TreeSet() | It is used to construct an empty tree set that will be sorted in ascending order according to the natural order of the tree set. |
| TreeSet(Collection<? extends E> c) | It is used to build a new tree set that contains the elements of the collection c. |
| TreeSet(Comparator<? super E> comparator) | It is used to construct an empty tree set that will be sorted according to given comparator. |
| TreeSet(SortedSet<E> s) | It is used to build a TreeSet that contains the elements of the given SortedSet. |

Methods of Java TreeSet Class

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(E e) | It is used to add the specified element to this set if it is not already present. |
| boolean addAll(Collection<? extends E> c) | It is used to add all of the elements in the specified collection to this set. |
| E ceiling(E e) | It returns the equal or closest greatest element of the specified element from the set, or null there is no such element. |
| Comparator<? super E> comparator() | It returns a comparator that arranges elements in order. |
| Iterator descendingIterator() | It is used to iterate the elements in descending order. |
| NavigableSet descendingSet() | It returns the elements in reverse order. |
| E floor(E e) | It returns the equal or closest least element of the specified element from the set, or null there is no such element. |
| SortedSet headSet(E toElement) | It returns the group of elements that are less than the specified element. |
| NavigableSet headSet(E toElement, boolean inclusive) | It returns the group of elements that are less than or equal to(if, inclusive is true) the specified element. |
| E higher(E e) | It returns the closest greatest element of the specified element from the set, or null there is no such element. |
| Iterator iterator() | It is used to iterate the elements in ascending order. |
| E lower(E e) | It returns the closest least element of the specified element from the set, or null there is no such element. |
| E pollFirst() | It is used to retrieve and remove the lowest(first) element. |
| E pollLast() | It is used to retrieve and remove the highest(last) element. |
| SortedSet subSet(E fromElement, E toElement)) | It returns a set of elements that lie between the given range which includes fromElement and excludes toElement. |
| SortedSet tailSet(E fromElement) | It returns a set of elements that are greater than or equal to the specified element. |
| boolean contains(Object o) | It returns true if this set contains the specified element. |
| boolean isEmpty() | It returns 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 clear() | It is used to remove all of the elements from this set. |
| Object clone() | It returns a shallow copy of this TreeSet instance. |
| E first() | It returns the first (lowest) element currently in this sorted set. |
| E last() | It returns the last (highest) element currently in this sorted set. |
| int size() | It returns the number of elements in this set. |

**Java TreeSet Examples**

**import** java.util.\*;

**class** TreeSet1{

**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());

  }

 }

}

Java TreeSet Example 2:

**import** java.util.\*;

**class** TreeSet2{

**public** **static** **void** main(String args[]){

 TreeSet<String> set=**new** TreeSet<String>();

         set.add("Ravi");

         set.add("Vijay");

         set.add("Ajay");

         System.out.println("Traversing element through Iterator in descending order");

         Iterator i=set.descendingIterator();

**while**(i.hasNext())

         {

             System.out.println(i.next());

         }

 }

}

Java TreeSet Example 3:

**import** java.util.\*;

**class** TreeSet3{

**public** **static** **void** main(String args[]){

 TreeSet<Integer> set=**new** TreeSet<Integer>();

         set.add(24);

         set.add(66);

         set.add(12);

         set.add(15);

         System.out.println("Lowest Value: "+set.pollFirst());

         System.out.println("Highest Value: "+set.pollLast());

 }

}

Java TreeSet Example 4:

**import** java.util.\*;

**class** TreeSet4{

**public** **static** **void** main(String args[]){

  TreeSet<String> set=**new** TreeSet<String>();

         set.add("A");

         set.add("B");

         set.add("C");

         set.add("D");

         set.add("E");

         System.out.println("Initial Set: "+set);

         System.out.println("Reverse Set: "+set.descendingSet());

         System.out.println("Head Set: "+set.headSet("C", **true**));

         System.out.println("SubSet: "+set.subSet("A", **false**, "E", **true**));

         System.out.println("TailSet: "+set.tailSet("C", **false**));

 }

}

Java TreeSet Example 5:

**import** java.util.\*;

**class** TreeSet5{

**public** **static** **void** main(String args[]){

  TreeSet<String> set=**new** TreeSet<String>();

         set.add("A");

         set.add("B");

         set.add("C");

         set.add("D");

         set.add("E");

         System.out.println("Intial Set: "+set);

         System.out.println("Head Set: "+set.headSet("C"));

         System.out.println("SubSet: "+set.subSet("A", "E"));

         System.out.println("TailSet: "+set.tailSet("C"));

 }

}

Java TreeSet Example: Book

**import** java.util.\*;

**class** Book **implements** Comparable<Book>{

**int** id;

String name,author,publisher;

**int** quantity;

**public** Book(**int** id, String name, String author, String publisher, **int** quantity) {

**this**.id = id;

**this**.name = name;

**this**.author = author;

**this**.publisher = publisher;

**this**.quantity = quantity;

}

// implementing the abstract method

**public** **int** compareTo(Book b) {

**if**(id>b.id){

**return** 1;

    }**else** **if**(id<b.id){

**return** -1;

    }**else**{

**return** 0;

    }

}

}

**public** **class** TreeSetExample {

**public** **static** **void** main(String[] args) {

    Set<Book> set=**new** TreeSet<Book>();

    //Creating Books

    Book b1=**new** Book(121,"Let us C","Yashwant Kanetkar","BPB",8);

    Book b2=**new** Book(233,"Operating System","Galvin","Wiley",6);

    Book b3=**new** Book(101,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

    //Adding Books to TreeSet

    set.add(b1);

    set.add(b2);

    set.add(b3);

    //Traversing TreeSet

**for**(Book b:set){

    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

    }

}

}

ClassCast Exception in TreeSet

If we add an object of the class that is not implementing the Comparable interface, the ClassCast Exception is raised. Observe the following program.

// important import statement

**import** java.util.\*;

**class** Employee

{

**int** empId;

String name;

// getting the name of the employee

String getName()

{

**return** **this**.name;

}

// setting the name of the employee

**void** setName(String name)

{

**this**.name = name;

}

// setting the employee id

// of the employee

**void** setId(**int** a)

{

**this**.empId = a;

}

// retrieving the employee id of

// the employee

**int** getId()

{

**return** **this**.empId;

}

}

**public** **class** ClassCastExceptionTreeSet

{

// main method

**public** **static** **void** main(String[] argvs)

{

// creating objects of the class Employee

Employee obj1 = **new** Employee();

Employee obj2 = **new** Employee();

TreeSet<Employee> ts =  **new** TreeSet<Employee>();

// adding the employee objects to

// the TreeSet class

ts.add(obj1);

ts.add(obj2);

System.out.println("The program has been executed successfully.");

}

}

When we compile the above program, we get the ClassCastException. **Explanation:** In the above program, it is required to implement a Comparable interface. It is because the TreeSet maintains the sorting order, and for doing the sorting the comparison of different objects that are being inserted in the TreeSet is must, which is accomplished by implementing the Comparable interface.

**HashMap**

Java HashMap class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform operations using the key index like updation, deletion, etc. HashMap class is found in the java.util package.

* Java HashMap contains values based on the key.
* Java HashMap contains only unique keys.
* Java HashMap may have one null key and multiple null values.
* Java HashMap is non synchronized.
* Java HashMap maintains no order.
* The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

**Hierarchy of HashMap class**

HashMap class extends AbstractMap class and implements Map interface.

**HashMap class declaration**

**public** **class** HashMap<K,V> **extends** AbstractMap<K,V> **implements** Map<K,V>, Cloneable, Serializable

**HashMap class Parameters**

* **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<? extends K,? extends V> m) | It is used to initialize 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 loadFactor) | It is used to initialize both the capacity and load factor 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 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. |
| V put(Object key, Object value) | It is used to insert an entry in the map. |
| void putAll(Map map) | It is used to insert the specified map in the map. |
| V putIfAbsent(K key, V value) | It inserts the specified value with the specified key in the map only if it is not already specified. |
| V remove(Object key) | It is used to delete an entry for the specified key. |
| boolean remove(Object key, Object value) | It removes the specified values with the associated specified keys from the map. |
| boolean containsValue(Object value) | This method returns true if some value equal to the value exists within the map, else return false. |
| boolean containsKey(Object key) | This method returns true if some key equal to the key exists within the map, else return false. |
| boolean equals(Object o) | It is used to compare the specified Object with the Map. |
| V get(Object key) | This method returns the object that contains the value associated with the key. |
| boolean isEmpty() | This method returns true if the map is empty; returns false if it contains at least one key. |
| V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction) | If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value. |
| V replace(K key, V value) | It replaces the specified value for a specified key. |
| boolean replace(K key, V oldValue, V newValue) | It replaces the old value with the new value for a specified key. |
| void replaceAll(BiFunction<? super K,? super V,? extends V> function) | It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception. |
| Collection<V> values() | It returns a collection view of the values contained in the map. |
| int size() | This method returns the number of entries in the map. |

**Java HashMap Example**

**import** java.util.\*;

**public** **class** HashMapExample1{

**public** **static** **void** main(String args[]){

   HashMap<Integer,String> map=**new** HashMap<Integer,String>();//Creating HashMap

   map.put(1,"Mango");  //Put elements in Map

   map.put(2,"Apple");

   map.put(3,"Banana");

   map.put(4,"Grapes");

   System.out.println("Iterating Hashmap...");

**for**(Map.Entry m : map.entrySet()){

    System.out.println(m.getKey()+" "+m.getValue());

   }

}

}

To get the key and value elements, we should call the getKey() and getValue() methods. The Map.Entry interface contains the *getKey()* and *getValue()* methods. But, we should call the entrySet() method of Map interface to get the instance of Map.Entry.

**No Duplicate Key on HashMap**

You cannot store duplicate keys in HashMap. However, if you try to store duplicate key with another value, it will replace the value.

**import** java.util.\*;

**public** **class** HashMapExample2{

**public** **static** **void** main(String args[]){

   HashMap<Integer,String> map=**new** HashMap<Integer,String>();//Creating HashMap

   map.put(1,"Mango");  //Put elements in Map

   map.put(2,"Apple");

   map.put(3,"Banana");

   map.put(1,"Grapes"); //trying duplicate key

   System.out.println("Iterating Hashmap...");

**for**(Map.Entry m : map.entrySet()){

    System.out.println(m.getKey()+" "+m.getValue());

   }

}

}

**Java HashMap example to add() elements**

**import** java.util.\*;

**class** HashMap1{

**public** **static** **void** main(String args[]){

   HashMap<Integer,String> hm=**new** HashMap<Integer,String>();

    System.out.println("Initial list of elements: "+hm);

      hm.put(100,"Amit");

      hm.put(101,"Vijay");

      hm.put(102,"Rahul");

      System.out.println("After invoking put() method ");

**for**(Map.Entry m:hm.entrySet()){

       System.out.println(m.getKey()+" "+m.getValue());

      }

      hm.putIfAbsent(103, "Gaurav");

      System.out.println("After invoking putIfAbsent() method ");

**for**(Map.Entry m:hm.entrySet()){

           System.out.println(m.getKey()+" "+m.getValue());

          }

      HashMap<Integer,String> map=**new** HashMap<Integer,String>();

      map.put(104,"Ravi");

      map.putAll(hm);

      System.out.println("After invoking putAll() method ");

**for**(Map.Entry m:map.entrySet()){

           System.out.println(m.getKey()+" "+m.getValue());

          }

 }

}

**Java HashMap example to remove() elements**

**import** java.util.\*;

**public** **class** HashMap2 {

**public** **static** **void** main(String args[]) {

    HashMap<Integer,String> map=**new** HashMap<Integer,String>();

      map.put(100,"Amit");

      map.put(101,"Vijay");

      map.put(102,"Rahul");

      map.put(103, "Gaurav");

    System.out.println("Initial list of elements: "+map);

    //key-based removal

    map.remove(100);

    System.out.println("Updated list of elements: "+map);

    //value-based removal

    map.remove(101);

    System.out.println("Updated list of elements: "+map);

    //key-value pair based removal

    map.remove(102, "Rahul");

    System.out.println("Updated list of elements: "+map);

   }

}

**Java HashMap example to replace() elements**

**import** java.util.\*;

**class** HashMap3{

**public** **static** **void** main(String args[]){

   HashMap<Integer,String> hm=**new** HashMap<Integer,String>();

      hm.put(100,"Amit");

      hm.put(101,"Vijay");

      hm.put(102,"Rahul");

      System.out.println("Initial list of elements:");

**for**(Map.Entry m:hm.entrySet())

     {

        System.out.println(m.getKey()+" "+m.getValue());

     }

     System.out.println("Updated list of elements:");

     hm.replace(102, "Gaurav");

**for**(Map.Entry m:hm.entrySet())

     {

        System.out.println(m.getKey()+" "+m.getValue());

     }

     System.out.println("Updated list of elements:");

     hm.replace(101, "Vijay", "Ravi");

**for**(Map.Entry m:hm.entrySet())

     {

        System.out.println(m.getKey()+" "+m.getValue());

     }

     System.out.println("Updated list of elements:");

     hm.replaceAll((k,v) -> "Ajay");

**for**(Map.Entry m:hm.entrySet())

     {

        System.out.println(m.getKey()+" "+m.getValue());

     }

 }

}

**Difference between HashSet and HashMap**

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

**Java HashMap Example: Book**

**import** java.util.\*;

**class** Book {

**int** id;

String name,author,publisher;

**int** quantity;

**public** Book(**int** id, String name, String author, String publisher, **int** quantity) {

**this**.id = id;

**this**.name = name;

**this**.author = author;

**this**.publisher = publisher;

**this**.quantity = quantity;

}

}

**class** MapExample {

**public** **static** **void** main(String[] args) {

    //Creating map of Books

    Map<Integer,Book> map=**new** HashMap<Integer,Book>();

    //Creating Books

    Book b1=**new** Book(101,"Let us C","Yashwant Kanetkar","BPB",8);

    Book b2=**new** Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

    Book b3=**new** Book(103,"Operating System","Galvin","Wiley",6);

    //Adding Books to map

    map.put(1,b1);

    map.put(2,b2);

    map.put(3,b3);

    //Traversing map

**for**(Map.Entry<Integer, Book> entry:map.entrySet()){

**int** key=entry.getKey();

        Book b=entry.getValue();

        System.out.println(key+" Details:");

        System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

    }

}

}