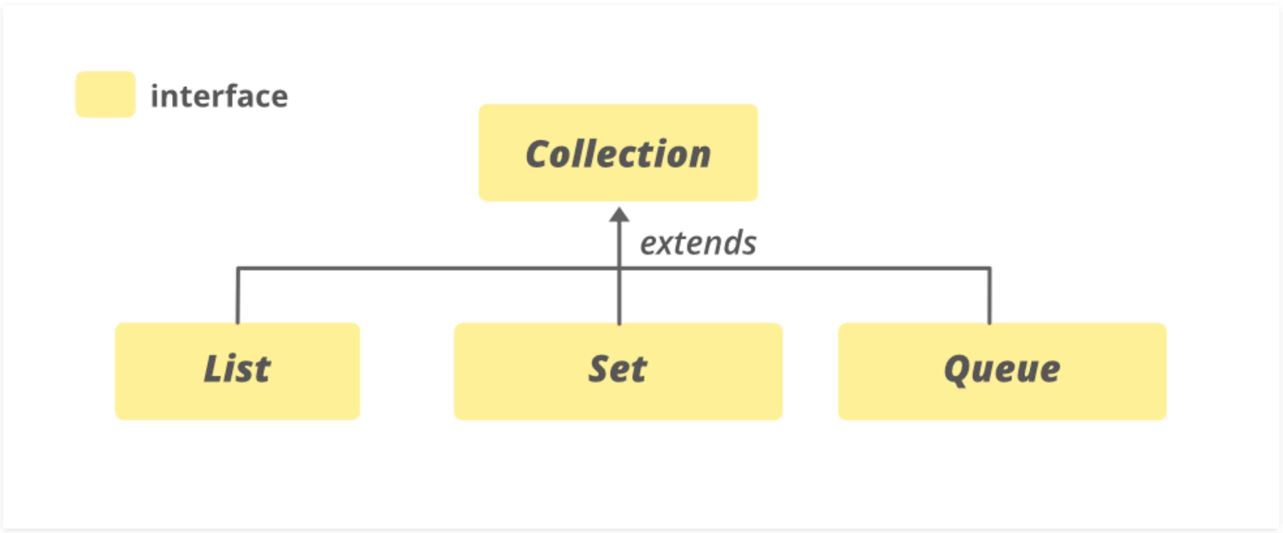
Collection Interface in Java with Examples

Difficulty Level : Medium Last Updated : 07 Aug, 2021

The **Collection** interface is a member of the Java Collections Framework. It is a part of **java.util** package. It is one of the root interfaces of the Collection Hierarchy. The Collection interface is not directly implemented by any class. However, it is implemented indirectly via its subtypes or subinterfaces like List, Queue, and Set.

For Example, the HashSet class implements the Set interface which is a subinterface of the Collection interface. If a collection implementation doesn't implement a particular operation, it should define the corresponding method to throw **UnsupportedOperationException**.

The Hierarchy of Collection



It implements the **Iterable<E>** interface. The sub-interfaces of Collection are **BeanContext**, **BeanContextServices**, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, **EventSet**, List<E>, NavigableSet<E>, Queue<E>, Set<E>, SortedSet<E>, **TransferQueue<E>** .

SubInterfaces of Collection Interface

All the Classes of the Collection Framework implement the subInterfaces of the Collection Interface. All the methods of Collection interfaces are also contained in it's subinterfaces. These subInterfaces are sometimes called as **Collection Types** or **SubTypes of Collection.** These include the following:

List: This is a child interface of the collection interface. This interface is dedicated to the data of the list type in which we can store all the ordered collection of the objects. This also allows duplicate data to be present in it. This list interface is implemented by various classes like ArrayList, Vector, Stack, etc. Since all the subclasses implement the list, we can instantiate a list object with any of these classes. For example,

List <T> al = new ArrayList<> ();

List <T> ll = new LinkedList<> ();

List <T> v = new Vector<> ();

Where T is the type of the object

Set: A set is an unordered collection of objects in which duplicate values cannot be stored. This collection is used when we wish to avoid the duplication of the objects and wish to store only the unique objects. This set interface is implemented by various classes like HashSet, TreeSet, LinkedHashSet, etc. Since all the subclasses implement the set, we can instantiate a set object with any of these classes. For example,

Set<T> hs = new HashSet<> ();

Set<T> lhs = new LinkedHashSet<> ();

Set<T> ts = new TreeSet<> ();

Where T is the type of the object.

SortedSet: This interface is very similar to the set interface. The only difference is that this interface has extra methods that maintain the ordering of the elements. The sorted set interface extends the set interface and is used to handle the data which needs to be sorted. The class which implements this interface is TreeSet. Since this class implements the SortedSet, we can instantiate a SortedSet object with this class. For example,

SortedSet<T> ts = new TreeSet<> ();

Where T is the type of the object.

Queue: As the name suggests, a queue interface maintains the FIFO(First In First Out) order similar to a real-world queue line. This interface is dedicated to storing all the elements where the order of the elements matter. For example, whenever we try to book a ticket, the tickets are sold at the first come first serve basis. Therefore, the person whose request arrives first into the queue gets the ticket. There are various classes like PriorityQueue, Deque, ArrayDeque, etc. Since all these subclasses implement the queue, we can instantiate a queue object with any of these classes. For example,

Queue <T> pq = new PriorityQueue<> ();

Queue <T> ad = new ArrayDeque<> ();

Where T is the type of the object.

Deque: This is a very slight variation of the queue data structure. Deque, also known as a double-ended queue, is a data structure where we can add and remove the elements from both the ends of the queue. This interface extends the queue interface. The class which implements this interface is ArrayDeque. Since this class implements the deque, we can instantiate a deque object with this class. For example,

Deque<T> ad = new ArrayDeque<> ();

Where T is the type of the object.

Declaration:

```
public interface Collection<E> extends Iterable<E>
```

Here, **E** is the type of elements stored in the collection.

Example:

```
// Java program to illustrate Collection interface

import java.io.*;
import java.util.*;

public class CollectionDemo {
    public static void main(String args[])
    {

        // creating an empty LinkedList
        Collection<String> list = new LinkedList<String>();

        // use add() method to add elements in the list
        list.add("Geeks");
        list.add("for");
        list.add("Geeks");

        // Output the present list
        System.out.println("The list is: " + list);

        // Adding new elements to the end
        list.add("Last");
        list.add("Element");

        // printing the new list
        System.out.println("The new List is: " + list);
    }
}
```

Output

```
The list is: [Geeks, for, Geeks]
The new List is: [Geeks, for, Geeks, Last, Element]
```

Implementing Classes

The Collection interface is implemented by AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, **AttributeList**, **BeanContextServicesSupport**, **BeanContextSupport**, ConcurrentHashMap.KeySetView, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, **JobStateReasons**, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, **RoleList**, **RoleUnresolvedList**, Stack, **SynchronousQueue**, TreeSet, Vector.

Syntax:

```
Collection<E> objectName = new ArrayList<E>();
```

Here, **E** is the type of elements stored in the collection.

Note: In the above syntax, we can replace any class with ArrayList if that class implements the Collection interface.

Basic Operations

1. Adding Elements

The add(E e) and addAll(Collection c) methods provided by Collection can be used to add elements.

```
// Java code to illustrate adding
// elements to the Collection

import java.io.*;
import java.util.*;

public class AddingElementsExample {
    public static void main(String[] args)
    {

        // create an empty array list with an initial
        // capacity
        Collection<Integer> list1 = new ArrayList<Integer>(5);

        // use add() method to add elements in the list
        list1.add(15);
        list1.add(20);
        list1.add(25);
        // prints all the elements available in list
        for (Integer number : list1) {
            System.out.println("Number = " + number);
        }

        // Creating an empty ArrayList
        Collection<Integer> list2 = new ArrayList<Integer>();

        // Appending the collection to the list
        list2.addAll(list1);

        // displaying the modified ArrayList
        System.out.println("The new ArrayList is: " + list2);
    }
}
```

Output

```
Number = 15
Number = 20
Number = 25
The new ArrayList is: [15, 20, 25]
```

2. Removing Elements

The **remove(E e)** and **removeAll(Collection c)** methods can be used to remove a particular element or a Collection of elements from a collection.

```
// Java program to demonstrate removing
// elements from a Collection

import java.util.*;

public class RemoveElementsExample {
    public static void main(String[] argv) throws Exception
    {

        // Creating object of HashSet<Integer>
        Collection<Integer> set1 = new HashSet<Integer>();

        // Populating arrset1
        set1.add(1);
        set1.add(2);
        set1.add(3);
        set1.add(4);
        set1.add(5);

        // print set1
        System.out.println("Initial set1 : " + set1);

          // remove a particular element
          set1.remove(4);

          // print modified set1
        System.out.println("set1 after removing 4 : " + set1);

        // Creating another object of HashSet<Integer>
        Collection<Integer> set2 = new HashSet<Integer>();
        set2.add(1);
        set2.add(2);
        set2.add(3);

        // print set2
        System.out.println("Collection Elements to be removed : " + set2);

        // Removing elements from set1
        // specified in set2
        // using removeAll() method
        set1.removeAll(set2);

        // print arrset1
        System.out.println("set 1 after removeAll() operation : " + set1);
    }
}
```

Output

```
Initial set1 : [1, 2, 3, 4, 5]
set1 after removing 4 : [1, 2, 3, 5]
Collection Elements to be removed : [1, 2, 3]
set 1 after removeAll() operation : [5]
```

3. Iterating

To iterate over the elements of Collection we can use **iterator()** method.

```
// Java code to illustrate iterating
// over a Collection

import java.util.*;

public class IteratingExample {

    public static void main(String[] args)
    {
        // Create and populate the list
        Collection<String> list = new LinkedList<>();

        list.add("Geeks");
        list.add("for");
        list.add("Geeks");
        list.add("is");
        list.add("a");
        list.add("CS");
        list.add("Students");
        list.add("Portal");

        // Displaying the list
        System.out.println("The list is: " + list);

        // Create an iterator for the list
        // using iterator() method
        Iterator<String> iter = list.iterator();

        // Displaying the values after iterating
        // through the list
        System.out.println("\nThe iterator values" + " of list are: ");
        while (iter.hasNext()) {
            System.out.print(iter.next() + " ");
        }
    }
}
```

Output

```
The list is: [Geeks, for, Geeks, is, a, CS, Students, Portal]

The iterator values of list are:
Geeks for Geeks is a CS Students Portal
```

Methods of Collection

METHOD	DESCRIPTION
add(E e)	Ensures that this collection contains the specified element (optional operation).
addAll(Collection<? extends E> c)	Adds all the elements in the specified collection to this collection (optional operation).
clear()	Removes all the elements from this collection (optional operation).
contains(Object o)	Returns true if this collection contains the specified element.
containsAll (Collection<?> c)	Returns true if this collection contains all the elements in the specified collection.
equals(Object o)	Compares the specified object with this collection for equality.
hashCode()	Returns the hash code value for this collection.
isEmpty()	Returns true if this collection contains no elements.

METHOD	DESCRIPTION
iterator()	Returns an iterator over the elements in this collection.
parallelStream()	Returns a possibly parallel Stream with this collection as its source.
remove(Object o)	Removes a single instance of the specified element from this collection, if it is present (optional operation).
removeAll(Collection<?> c)	Removes all of this collection's elements that are also contained in the specified collection (optional operation).
removeIf(Predicate<? super E> filter)	Removes all the elements of this collection that satisfy the given predicate.
retainAll(Collection<?> c)	Retains only the elements in this collection that are contained in the specified collection (optional operation).
size()	Returns the number of elements in this collection.
spliterator()	Creates a Spliterator over the elements in this collection.
stream()	Returns a sequential Stream with this collection as its source.
toArray()	Returns an array containing all the elements in this collection.
toArray (IntFunction<T[]> generator)	Returns an array containing all the elements in this collection, using the provided generator function to allocate the returned array.
toArray(T[] a)	Returns an array containing all the elements in this collection; the runtime type of the returned array is that of the specified array.

Methods declared in interface java.lang.Iterable

METHOD	DESCRIPTION
forEach(Consumer<? super T> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.

Reference: https://docs.oracle