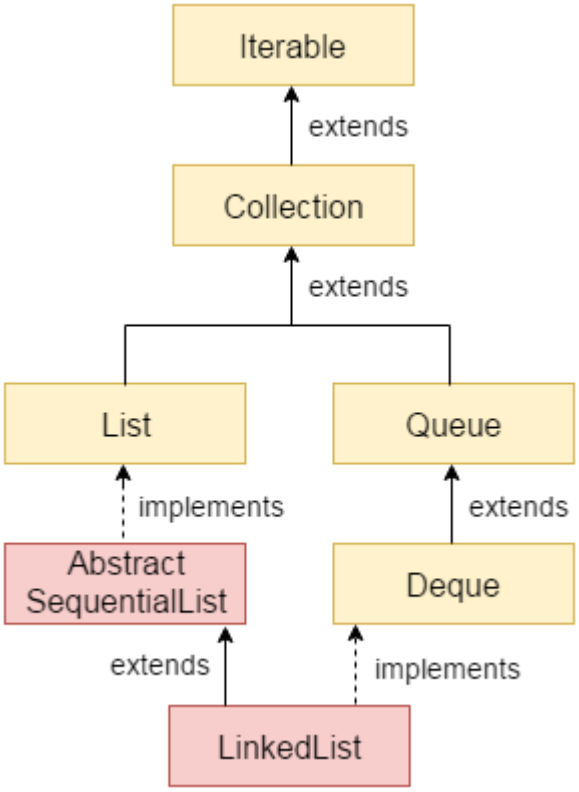
Java LinkedList class



Java LinkedList class uses a 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 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.

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, Seria lizable

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.*;

public 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());

  }

 }

}
```

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

Java LinkedList example to add elements

Here, we see different ways to add elements.

```
1.  import java.util.*;
2.  public class LinkedList2{
3.   public static void main(String args[]){
4.   LinkedList<String> ll=new LinkedList<String>();
5.          System.out.println("Initial list of elements: "+ll);
6.          ll.add("Ravi");
7.          ll.add("Vijay");
8.          ll.add("Ajay");
9.          System.out.println("After invoking add(E e) method: "+ll);
10.         //Adding an element at the specific position
11.         ll.add(1, "Gaurav");
12.         System.out.println("After invoking add(int index, E element) method: "+ll);
13.         LinkedList<String> ll2=new LinkedList<String>();
14.         ll2.add("Sonoo");
15.         ll2.add("Hanumat");
16.         //Adding second list elements to the first list
17.         ll.addAll(ll2);
18.         System.out.println("After invoking addAll(Collection<? extends E> c) method: "+ll);
19.         LinkedList<String> ll3=new LinkedList<String>();
20.         ll3.add("John");
21.         ll3.add("Rahul");
22.         //Adding second list elements to the first list at specific position
23.         ll.addAll(1, ll3);
24.         System.out.println("After invoking addAll(int index, Collection<? extends E> c) method: "+ll);
25.         //Adding an element at the first position
26.         ll.addFirst("Lokesh");
27.         System.out.println("After invoking addFirst(E e) method: "+ll);
28.         //Adding an element at the last position
29.         ll.addLast("Harsh");
30.         System.out.println("After invoking addLast(E e) method: "+ll);
31.
32.  }
33. }
```

```
Initial list of elements: []
After invoking add(E e) method: [Ravi, Vijay, Ajay]
After invoking add(int index, E element) method: [Ravi, Gaurav, Vijay, Ajay]
After invoking addAll(Collection<? extends E> c) method:
[Ravi, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addAll(int index, Collection<? extends E> c) method:
[Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addFirst(E e) method:
[Lokesh, Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addLast(E e) method:
[Lokesh, Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat, Harsh]
```

Java LinkedList example to remove elements

Here, we see different ways to remove an element.

```
import java.util.*;

public 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);

        }

      }
```

```
Initial list of elements: [Ravi, Vijay, Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav,
Harsh, Amit]
After invoking remove(object) method: [Ravi, Ajay, Anuj, Gaurav, Harsh, Virat,
Gaurav, Harsh, Amit]
After invoking remove(index) method: [Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav,
Harsh, Amit]
Updated list : [Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav, Harsh, Amit, Ravi,
Hanumat]
After invoking removeAll() method: [Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav,
Harsh, Amit]
After invoking removeFirst() method: [Gaurav, Harsh, Virat, Gaurav, Harsh, Amit]
After invoking removeLast() method: [Gaurav, Harsh, Virat, Gaurav, Harsh]
After invoking removeFirstOccurrence() method: [Harsh, Virat, Gaurav, Harsh]
After invoking removeLastOccurrence() method: [Harsh, Virat, Gaurav]
After invoking clear() method: []
```

Java LinkedList Example to reverse a list of elements

```
1.  import java.util.*;
2.  public class LinkedList4{
3.   public static void main(String args[]){
4.
5.    LinkedList<String> ll=new LinkedList<String>();
6.          ll.add("Ravi");
7.          ll.add("Vijay");
8.          ll.add("Ajay");
9.          //Traversing the list of elements in reverse order
10.         Iterator i=ll.descendingIterator();
11.         while(i.hasNext())
12.         {
13.            System.out.println(i.next());
14.         }
15.
```

```
    }
}
```

```
Output: Ajay
Vijay
Ravi
```

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;
}
}
public 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);
    }
}
}
```

Output:

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

Difference between ArrayList and LinkedList

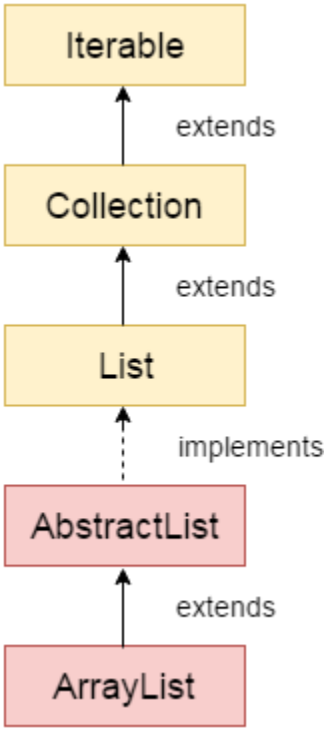
ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

However, there are many differences between ArrayList and LinkedList classes that are given below.

ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.

3) An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.

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 occur if any element is removed from the array list.

Hierarchy of ArrayList class

As shown in the 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<? 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 Java ArrayList

Method	Description
void add(int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean add(E e)	It is used to append the specified element at the end of 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(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()	It is used to remove all of the elements from this list.
void ensureCapacity(int requiredCapacity)	It is used to enhance the capacity of an ArrayList instance.
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(Object o)	It is used to remove the first occurrence of the specified element.
boolean removeAll(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(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.
void retainAll(Collection<?> c)	It is used to retain all the elements in the list that are present in the specified collection.
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 specified comparator.
Spliterator<E> spliterator()	It is used to create spliterator over the elements in a list.
List<E> subList(int fromIndex, int toIndex)	It is used to fetch all the elements lies within the given range.
int size()	It is used to return the number of elements present in the list.
void trimToSize()	It is used to trim the capacity of this ArrayList instance to be the list's current size.

Java Non-generic Vs. Generic Collection

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

Java new generic collection allows you to have only one type of object in a collection. Now it is type safe so typecasting is not required at runtime.

Let's see the old non-generic example of creating java collection.

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

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

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

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

For more information on Java generics, click here Java Generics Tutorial.

Java ArrayList Example

```
1.  import java.util.*;
2.   class ArrayList1{
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.        //Invoking arraylist object
10.       System.out.println(list);
11.   }
12.  }
13. }
```

```
[Ravi, Vijay, Ravi, Ajay]
```

Ways to iterate the elements of the collection in java

There are various ways to traverse the collection elements:

1. By Iterator interface.
2. By for-each loop.
3. By ListIterator interface.
4. By for loop.
5. By forEach() method.
6. By forEachRemaining() method.

Iterating Collection through Iterator interface

Let's see an example to traverse ArrayList elements using the Iterator interface.

```
1.  import java.util.*;
2.  class ArrayList2{
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.  }
```

Test it Now

```
Ravi
Vijay
Ravi
Ajay
```

Iterating Collection through the for-each loop

Let's see an example to traverse the ArrayList elements using the for-each loop

```
1.    import java.util.*;
2.    class ArrayList3{
3.     public static void main(String args[]){
4.      ArrayList<String> al=new ArrayList<String>();
5.      al.add("Ravi");
6.      al.add("Vijay");
7.      al.add("Ravi");
8.      al.add("Ajay");
9.       //Traversing list through for-each loop
10.     for(String obj:al)
11.       System.out.println(obj);
12.   }
13.  }
```

```
Ravi
Vijay
Ravi
Ajay
```

Iterating Collection through remaining ways

Let's see an example to traverse the ArrayList elements through other ways

```
import java.util.*;

class ArrayList4{

 public static void main(String args[]){

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

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

        list.add("Vijay");

        list.add("Ravi");

        list.add("Ajay");



        System.out.println("Traversing list through List Iterator:");

        //Here, element iterates in reverse order
```

```
            ListIterator<String> list1=list.listIterator(list.size());
            while(list1.hasPrevious())
            {
               String str=list1.previous();
               System.out.println(str);
            }
        System.out.println("Traversing list through for loop:");
          for(int i=0;i<list.size();i++)
          {
           System.out.println(list.get(i));
          }

        System.out.println("Traversing list through forEach() method:");
        //The forEach() method is a new feature, introduced in Java 8.
           list.forEach(a->{ //Here, we are using lambda expression
              System.out.println(a);
            });

           System.out.println("Traversing list through forEachRemaining() method:");
            Iterator<String> itr=list.iterator();
            itr.forEachRemaining(a-> //Here, we are using lambda expression
            {
           System.out.println(a);
            });
    }
   }
```

```
Traversing list through List Iterator:
Ajay
Ravi
Vijay
Ravi
Traversing list through for loop:
```

```
Ravi
Vijay
Ravi
Ajay
Traversing list through forEach() method:
Ravi
Vijay
Ravi
Ajay
Traversing list through forEachRemaining() method:
Ravi
Vijay
Ravi
Ajay
```

User-defined class objects in Java ArrayList

Let's see an example where we are storing Student class object in an array list.

```
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);

 }

}

}
```

```
101 Sonoo 23
102 Ravi 21
103 Hanumat 25
```

Java ArrayList Serialization and Deserialization Example

Let?s see an example to serialize an ArrayList object and then deserialize it.

```
1.  import java.io.*;
2.  import java.util.*;
3.   class ArrayList6 {
4.
5.         public static void main(String [] args)
6.         {
7.           ArrayList<String> al=new ArrayList<String>();
8.           al.add("Ravi");
9.           al.add("Vijay");
10.          al.add("Ajay");
11.
12.          try
13.          {
14.             //Serialization
15.             FileOutputStream fos=new FileOutputStream("file");
16.             ObjectOutputStream oos=new ObjectOutputStream(fos);
17.             oos.writeObject(al);
18.             fos.close();
19.             oos.close();
20.             //Deserialization
21.             FileInputStream fis=new FileInputStream("file");
22.             ObjectInputStream ois=new ObjectInputStream(fis);
23.           ArrayList  list=(ArrayList)ois.readObject();
24.           System.out.println(list);
25.          }catch(Exception e)
26.          {
27.             System.out.println(e);
28.          }
29.       }
```

```
30.     }

        [Ravi, Vijay, Ajay]
```

Java ArrayList example to add elements

Here, we see different ways to add an element.

```
1.  import java.util.*;
2.   class ArrayList7{
3.   public static void main(String args[]){
4.    ArrayList<String> al=new ArrayList<String>();
5.           System.out.println("Initial list of elements: "+al);
6.           //Adding elements to the end of the list
7.           al.add("Ravi");
8.           al.add("Vijay");
9.           al.add("Ajay");
10.          System.out.println("After invoking add(E e) method: "+al);
11.          //Adding an element at the specific position
12.          al.add(1, "Gaurav");
13.          System.out.println("After invoking add(int index, E element) method: "+al);
14.          ArrayList<String> al2=new ArrayList<String>();
15.          al2.add("Sonoo");
16.          al2.add("Hanumat");
17.          //Adding second list elements to the first list
18.          al.addAll(al2);
19.          System.out.println("After invoking addAll(Collection<? extends E> c) method: "+al);
20.          ArrayList<String> al3=new ArrayList<String>();
21.          al3.add("John");
22.          al3.add("Rahul");
23.          //Adding second list elements to the first list at specific position
24.          al.addAll(1, al3);
25.          System.out.println("After invoking addAll(int index, Collection<? extends E> c) method: "+al);
26.
27.  }
28. }
```

```
Initial list of elements: []
After invoking add(E e) method: [Ravi, Vijay, Ajay]
After invoking add(int index, E element) method: [Ravi, Gaurav, Vijay, Ajay]
After invoking addAll(Collection<? extends E> c) method:
[Ravi, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addAll(int index, Collection<? extends E> c) method:
[Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
```

Java ArrayList example to remove elements

Here, we see different ways to remove an element.

```
1.  import java.util.*;
2.   class ArrayList8 {
3.
4.          public static void main(String [] args)
5.          {
6.            ArrayList<String> al=new ArrayList<String>();
7.            al.add("Ravi");
8.            al.add("Vijay");
9.            al.add("Ajay");
10.           al.add("Anuj");
11.           al.add("Gaurav");
12.           System.out.println("An initial list of elements: "+al);
13.           //Removing specific element from arraylist
```

```
14.           al.remove("Vijay");
15.           System.out.println("After invoking remove(object) method: "+al);
16.           //Removing element on the basis of specific position
17.           al.remove(0);
18.           System.out.println("After invoking remove(index) method: "+al);
19.
20.           //Creating another arraylist
21.           ArrayList<String> al2=new ArrayList<String>();
22.           al2.add("Ravi");
23.           al2.add("Hanumat");
24.           //Adding new elements to arraylist
25.           al.addAll(al2);
26.           System.out.println("Updated list : "+al);
27.           //Removing all the new elements from arraylist
28.           al.removeAll(al2);
29.           System.out.println("After invoking removeAll() method: "+al);
30.           //Removing elements on the basis of specified condition
31.           al.removeIf(str -> str.contains("Ajay"));   //Here, we are using Lambda expression
32.           System.out.println("After invoking removeIf() method: "+al);
33.           //Removing all the elements available in the list
34.           al.clear();
35.           System.out.println("After invoking clear() method: "+al);
36.         }
37.      }
```

```
An initial list of elements: [Ravi, Vijay, Ajay, Anuj, Gaurav]
After invoking remove(object) method: [Ravi, Ajay, Anuj, Gaurav]
After invoking remove(index) method: [Ajay, Anuj, Gaurav]
Updated list : [Ajay, Anuj, Gaurav, Ravi, Hanumat]
After invoking removeAll() method: [Ajay, Anuj, Gaurav]
After invoking removeIf() method: [Anuj, Gaurav]
After invoking clear() method: []
```

Java ArrayList example of retainAll() method

```
1.    import java.util.*;
2.    class ArrayList9{
3.     public static void main(String args[]){
4.      ArrayList<String> al=new ArrayList<String>();
5.      al.add("Ravi");
6.      al.add("Vijay");
7.      al.add("Ajay");
8.      ArrayList<String> al2=new ArrayList<String>();
9.      al2.add("Ravi");
10.     al2.add("Hanumat");
11.     al.retainAll(al2);
12.     System.out.println("iterating the elements after retaining the elements of al2");
13.     Iterator itr=al.iterator();
14.     while(itr.hasNext()){
15.      System.out.println(itr.next());
16.     }
17.    }
18.   }
```

```
iterating the elements after retaining the elements of al2
Ravi
```

Java ArrayList example of isEmpty() method

```
1.    import java.util.*;
2.     class ArrayList10{
3.
4.           public static void main(String [] args)
```

```
5.          {
6.            ArrayList<String> al=new ArrayList<String>();
7.            System.out.println("Is ArrayList Empty: "+al.isEmpty());
8.            al.add("Ravi");
9.            al.add("Vijay");
10.           al.add("Ajay");
11.           System.out.println("After Insertion");
12.           System.out.println("Is ArrayList Empty: "+al.isEmpty());
13.       }
14.     }
```

```
Is ArrayList Empty: true
After Insertion
Is ArrayList Empty: false
```

Java ArrayList example of set() and get() method

```
1.  import java.util.*;
2.   class ArrayList11 {
3.
4.          public static void main(String [] args)
5.          {
6.            ArrayList<String> al=new ArrayList<String>();
7.               al.add("Ravi");
8.               al.add("Vijay");
9.               al.add("Ajay");
10.              System.out.println("Before update: "+al.get(1));
11.              //Updating an element at specific position
12.              al.set(1,"Gaurav");
13.              System.out.println("After update: "+al.get(1));
14.       }
15.     }
```

```
Before update: Vijay
After update: Gaurav
```

Java ArrayList Example: Book

Let's see an ArrayList example where we are adding books to list 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 ArrayListExample {
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.  }
```

Test it Now

Output:

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

Set in Java

- Set is an interface which extends Collection. It is an unordered collection of objects in which duplicate values cannot be stored.
- Basically, Set is implemented by HashSet, LinkedHashSet or TreeSet (sorted representation).
- Set has various methods to add, remove clear, size, etc to enhance the usage of this interface

```
// Java code for adding elements in Set
import java.util.*;
public class Set_example
{
    public static void main(String[] args)
    {
        // Set deonstration using HashSet
        Set<String> hash_Set = new HashSet<String>();
        hash_Set.add("Geeks");
        hash_Set.add("For");
        hash_Set.add("Geeks");
        hash_Set.add("Example");
        hash_Set.add("Set");
        System.out.print("Set output without the duplicates");

        System.out.println(hash_Set);

        // Set deonstration using TreeSet
        System.out.print("Sorted Set after passing into TreeSet");
        Set<String> tree_Set = new TreeSet<String>(hash_Set);
        System.out.println(tree_Set);
    }
}
```

(Please note that we have entered a duplicate entity but it is not displayed in the output. Also, we can directly sort the entries by passing the unordered Set in as the parameter of TreeSet).

Output:

```
Set output without the duplicates[Geeks, Example, For, Set]
Sorted Set after passing into TreeSet[Example, For, Geeks, Set]
```

Note: As we can see the duplicate entry "Geeks" is ignored in the final output, Set interface doesn't allow duplicate entries.

Now we will see some of the basic operations on the Set i.e. Union, Intersection and Difference.

Let's take an example of two integer Sets:

- [1, 3, 2, 4, 8, 9, 0]
- [1, 3, 7, 5, 4, 0, 7, 5]

Union
In this, we could simply add one Set with other. Since the Set will itself not allow any duplicate entries, we need not take care of the common values.

Expected Output:

```
Union : [0, 1, 2, 3, 4, 5, 7, 8, 9]
```

Intersection
We just need to retain the common values from both Sets.

Expected Output:

```
Intersection : [0, 1, 3, 4]
```

Difference
We just need to remove all the values of one Set from the other.

Expected Output:

```
Difference : [2, 8, 9]
```

```
// Java code for demonstrating union, intersection and difference
// on Set
import java.util.*;
public class Set_example
{
```

```
    public static void main(String args[])
    {
        Set<Integer> a = new HashSet<Integer>();
        a.addAll(Arrays.asList(new Integer[] {1, 3, 2, 4, 8, 9, 0}));
        Set<Integer> b = new HashSet<Integer>();
        b.addAll(Arrays.asList(new Integer[] {1, 3, 7, 5, 4, 0, 7, 5}));

        // To find union
        Set<Integer> union = new HashSet<Integer>(a);
        union.addAll(b);
        System.out.print("Union of the two Set");
        System.out.println(union);

        // To find intersection
        Set<Integer> intersection = new HashSet<Integer>(a);
        intersection.retainAll(b);
        System.out.print("Intersection of the two Set");
        System.out.println(intersection);

        // To find the symmetric difference
        Set<Integer> difference = new HashSet<Integer>(a);
        difference.removeAll(b);
        System.out.print("Difference of the two Set");
        System.out.println(difference);
    }
}
```

Output:

```
Union of the two Set[0, 1, 2, 3, 4, 5, 7, 8, 9]
Intersection of the two Set[0, 1, 3, 4]
Difference of the two Set[2, 8, 9]
```