

ArrayLists

1)

```
1 // Short program that creates a list of the birthdays of the
2 // first 5 U.S. Presidents and that puts them into sorted order.
3
4 import java.util.*;
5
6 public class CalendarDateTest {
7     public static void main(String[] args) {
8         ArrayList<CalendarDate> dates =
9             new ArrayList<CalendarDate>();
10        dates.add(new CalendarDate(2, 22)); // Washington
11        dates.add(new CalendarDate(10, 30)); // Adams
12
13        dates.add(new CalendarDate(4, 13)); // Jefferson
14        dates.add(new CalendarDate(3, 16)); // Madison
15        dates.add(new CalendarDate(4, 28)); // Monroe
16
17        System.out.println("birthdays = " + dates);
18        Collections.sort(dates);
19        System.out.println("birthdays = " + dates);
20    }
21 }
```

2)

Table 10.1 Basic ArrayList Methods

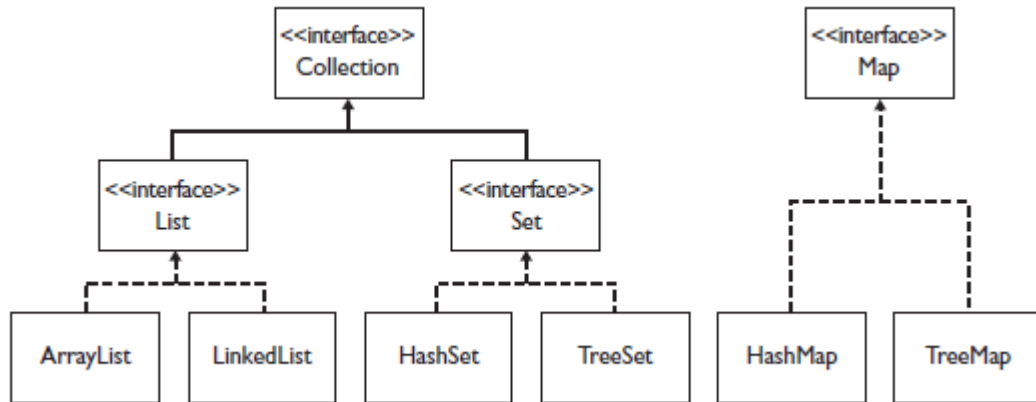
Method	Description	ArrayList<String> example
<code>add(value)</code>	adds the given value at the end of the list	<code>list.add("end");</code>
<code>add(index, value)</code>	adds the given value at the given index, shifting subsequent values right	<code>list.add(1, "middle");</code>
<code>clear()</code>	removes all elements from the list	<code>list.clear();</code>
<code>get(index)</code>	gets the value at the given index	<code>list.get(1)</code>
<code>remove(index)</code>	removes the value at the given index, shifting subsequent values left	<code>list.remove(1);</code>
<code>set(index, value)</code>	replaces the value at the given index with the given value	<code>list.set(2, "hello");</code>
<code>size()</code>	returns the current number of elements in the list	<code>list.size()</code>

3)

Table 11.3 Useful Static Methods of the Collections Class

Method	Description
<code>binarySearch(list, value)</code>	Searches a sorted list for a given element value and returns its index
<code>copy(destinationList, sourceList)</code>	Copies all elements from the source list to the destination list
<code>fill(list, value)</code>	Replaces every element in the given list with the given value
<code>max(list)</code>	Returns the element with the highest value
<code>min(list)</code>	Returns the element with the lowest value
<code>replaceAll(list, oldValue, newValue)</code>	Replaces all occurrences of the old value with the new value
<code>reverse(list)</code>	Reverses the order of the elements in the given list
<code>rotate(list, distance)</code>	Shifts each element to the right by the given number of indexes, moving the final element to the front
<code>shuffle(list)</code>	Rearranges the elements into random order
<code>sort(list)</code>	Rearranges the elements into sorted (nondecreasing) order
<code>swap(list, index1, index2)</code>	Switches the element values at the given two indexes

4)



5) Other ArrayList Examples:

Example 2

```
//Traversing an ArrayList of Integer.  
//Print the elements of list, one per line.  
for (Integer num : list)  
    System.out.println(num);
```

Example 3

```
/* Precondition: List list is an ArrayList that contains Integer
 *               values sorted in increasing order.
 * Postcondition: value inserted in its correct position in list. */
public static void insert(List<Integer> list, Integer value)
{
    int index = 0;
    //find insertion point
    while (index < list.size() &&
           value.compareTo(list.get(index)) > 0)
        index++;
    //insert value
    list.add(index, value);
}
```

NOTE

Suppose value is larger than all the elements in list. Then the insert method will throw an `IndexOutOfBoundsException` if the first part of the test is omitted, namely `index < list.size()`.

Example 4

```
//Return an ArrayList of random integers from 0 to 100.
public static List<Integer> getRandomIntList()
{
    List<Integer> list = new ArrayList<Integer>();
    System.out.print("How many integers? ");
    int length = IO.readInt();    //read user input
    for (int i = 0; i < length; i++)
    {
        int newNum = (int) (Math.random() * 101);
        list.add(new Integer(newNum));
    }
    return list;
}
```

NOTE

1. The variable `list` is declared to be of type `List<Integer>` (the interface) but is instantiated as type `ArrayList<Integer>` (the implementation).
2. The `add` method in `getRandomIntList` is the `List` method that appends its parameter to the end of the list.

Example 5

```
//Swap two values in list, indexed at i and j.
public static void swap(List<E> list, int i, int j)
{
    E temp = list.get(i);
    list.set(i, list.get(j));
    list.set(j, temp);
}
```

Example 6

```
//Print all negatives in list a.
//Precondition: a contains Integer values.
public static void printNegs(List<Integer> a)
{
    System.out.println("The negative values in the list are: ");
    for (Integer i : a)
        if (i.intValue() < 0)
            System.out.println(i);
}
```

Example 7

```
//Change every even-indexed element of strList to the empty string.
//Precondition: strList contains String values.
public static void changeEvenToEmpty(List<String> strList)
{
    boolean even = true;
    int index = 0;
    while (index < strList.size())
    {
        if (even)
            strList.set(index, "");
        index++;
        even = !even;
    }
}
```

Iterators:

Definition of an Iterator

An *iterator* is an object whose sole purpose is to traverse a collection, one element at a time. During iteration, the iterator object maintains a current position in the collection, and is the controlling object in manipulating the elements of the collection.

The Iterator<E> Interface

The package `java.util` provides a generic interface, `Iterator<E>`, whose methods are `hasNext`, `next`, and `remove`. The Java Collections API allows iteration over each of its collections classes.

THE METHODS OF `Iterator<E>`

```
boolean hasNext()
```

Returns `true` if there's at least one more element to be examined, `false` otherwise.

```
E next()
```

Returns the next element in the iteration. If no elements remain, the method throws a `NoSuchElementException`.

```
void remove()
```

Deletes from the collection the last element that was returned by `next`. This method can be called only once per call to `next`. It throws an `IllegalStateException` if the `next` method has not yet been called, or if the `remove` method has already been called after the last call to `next`.

Using a Generic Iterator

To iterate over a parameterized collection, you must use a parameterized iterator whose parameter is the same type.

Example 1

```
List<String> list = new ArrayList<String>();  
<code to initialize list with strings>  
//Print strings in list, one per line.  
Iterator<String> itr = list.iterator();  
while (itr.hasNext())  
    System.out.println(itr.next());
```

NOTE

1. Only classes that allow iteration can use the for-each loop. This is because the loop operates by using an iterator. Thus, the loop in the above example is equivalent to

```
for (String str : list)    //no iterator in sight!  
    System.out.println(str);
```

2. Recall, however, that a for-each loop cannot be used to remove elements from the list. The easiest way to “remove all occurrences of ...” from an ArrayList is to use an iterator.

Example 2

```
//Remove all 2-character strings from strList.  
//Precondition: strList initialized with String objects.  
public static void removeTwos(List<String> strList)  
{  
    Iterator<String> itr = strList.iterator();  
    while (itr.hasNext())  
        if (itr.next().length() == 2)  
            itr.remove();  
}
```

Example 3

```
//Assume a list of integer strings.  
//Remove all occurrences of "6" from the list.  
Iterator<String> itr = list.iterator();  
while (itr.hasNext())  
{  
    String num = itr.next();  
    if (num.equals("6"))  
    {  
        itr.remove();  
        System.out.println(list);  
    }  
}
```

If the original list is 2 6 6 3 5 6 the output will be

```
[2, 6, 3, 5, 6]  
[2, 3, 5, 6]  
[2, 3, 5]
```


Example 6

```
//Remove all negatives from intList.  
//Precondition: intList contains Integer objects.  
public static void removeNegs(List<Integer> intList)  
{  
    Iterator<Integer> itr = intList.iterator();  
    while (itr.hasNext())  
        if (itr.next().intValue() < 0)  
            itr.remove();  
}
```

NOTE

In a given program, the declaration

```
Iterator<SomeType> itr = list.iterator();
```

must be made every time you need to initialize the iterator to the beginning of the list.