List Class

Contains methods for the List collection type.

Namespace

System

Usage

The list methods are all instance methods, that is, they operate on a particular instance of a list. For example, the following removes all elements from `myList`:

```
myList.clear();
```

Even though the `clear` method does not include any parameters, the list that calls it is its implicit parameter.

> **Note**
>
> - When using a custom type for the list elements, provide an `equals` method in your class. Apex uses this method to determine equality and uniqueness for your objects. For more information on providing an `equals` method, see Using Custom Types in Map Keys and Sets.
> - If the list contains String elements, the elements are case-sensitive. Two list elements that differ only by case are considered distinct.

For more information on lists, see Lists.

- List Constructors
- List Methods

List Constructors

The following are constructors for `List`.

- List<T>()
  Creates a new instance of the `List` class. A list can hold elements of any data type T.
- List<T>(listToCopy)
  Creates a new instance of the `List` class by copying the elements from the specified list. T is the data type of the elements in both lists and can be any data type.
- List<T>(setToCopy)
  Creates a new instance of the `List` class by copying the elements from the specified set. T is the data type of the elements in the set and list and can be any data type.

List<T>()

Creates a new instance of the `List` class. A list can hold elements of any data type T.

```
// Create a list
List<Integer> ls1 = new List<Integer>();
// Add two integers to the list
ls1.add(1);
ls1.add(2);
```

List<T>(listToCopy)

Creates a new instance of the `List` class by copying the elements from the specified list. T is the data type of the elements in both lists and can be any data type.

Signature

```
public List<T>(List<T> listToCopy)
```

Parameters

listToCopy

Type: List<T>

The list containing the elements to initialize this list from. T is the data type of the list elements.

Example

```
List<Integer> ls1 = new List<Integer>();
ls1.add(1);
ls1.add(2);
// Create a list based on an existing one
List<Integer> ls2 = new List<Integer>(ls1);
// ls2 elements are copied from ls1
System.debug(ls2);// DEBUG|(1, 2)
```

List<T>(setToCopy)

Creates a new instance of the `List` class by copying the elements from the specified set. T is the data type of the elements in the set and list and can be any data type.

Signature

```
public List<T>(Set<T> setToCopy)
```

Parameters

setToCopy

Type: Set<T>

The set containing the elements to initialize this list with. T is the data type of the set elements.

Example

```
Set<Integer> s1 = new Set<Integer>();
s1.add(1);
s1.add(2);
// Create a list based on a set
List<Integer> ls = new List<Integer>(s1);
// ls elements are copied from s1
```

List Methods

The following are methods for `List`. All are instance methods.

- **add(listElement)**
  Adds an element to the end of the list.
- **add(index, listElement)**
  Inserts an element into the list at the specified index position and shifts all subsequent elements one index position to the right.
- **addAll(fromList)**
  Adds all of the elements in the specified list to the list that calls the method. Both lists must be of the same type.
- **addAll(fromSet)**
  Add all of the elements in specified set to the list that calls the method. The set and the list must be of the same type.
- **clear()**
  Removes all elements from a list, consequently setting the list's length to zero.
- **clone()**
  Makes a duplicate copy of a list.
- **contains(listElement)**
  Returns `true` if the list contains the specified element.
- **deepClone(preserveId, preserveReadonlyTimestamps, preserveAutonumber)**
  Makes a duplicate copy of a list of sObject records, including the sObject records themselves.
- **equals(list2)**
  Compares this list with the specified list and returns `true` if both lists are equal; otherwise, returns `false`.
- **get(index)**
  Returns the list element stored at the specified index.
- **getSObjectType()**
  Returns the token of the sObject type that makes up a list of sObjects.
- **hashCode()**
  Returns the hashcode corresponding to this list and its contents.
- **indexOf(listElement)**
  Returns the index of the first occurrence of the specified element in this list. If this list does not contain the element, returns -1.
- **isEmpty()**
  Returns true if the list has zero elements.
- **iterator()**
  Returns an instance of an iterator for this list.
- **remove(index)**
  Removes the list element stored at the specified index, returning the element that was removed.
- **set(index, listElement)**
  Sets the specified value for the element at the given index.
- **size()**
  Returns the number of elements in the list.
- **sort()**
  Sorts the items in the list in ascending order.
- **toString()**
  Returns the string representation of the list.

add(listElement)

Parameters

listElement

Type: Object

Return Value

Type: Void

Example

```
List<Integer> myList = new List<Integer>();
myList.add(47);
Integer myNumber = myList.get(0);
system.assertEquals(47, myNumber);
```

add(index, listElement)

Inserts an element into the list at the specified index position and shifts all subsequent elements one index position to the right.

Signature

```
public Void add(Integer index, Object listElement)
```

Parameters

index

Type: Integer

listElement

Type: Object

Return Value

Type: Void

Example

In this example, a list with six elements is created. Integers are added to the first and second index positions, and the subsequent elements are shifted to the right. After the two methods are called, the list has eight total elements.

```
List<Integer> myList = new Integer[6];
myList.add(0, 47);
myList.add(1, 52);
system.debug(myList); // DEBUG|(47, 52, null, null, null, null, null, null)
system.assertEquals(52, myList.get(1));
```

addAll(fromList)

Adds all of the elements in the specified list to the list that calls the method. Both lists must be of the same type.

Signature

```
public Void addAll(List fromList)
```

Return Value

Type: Void

addAll(fromSet)

Add all of the elements in specified set to the list that calls the method. The set and the list must be of the same type.

Signature

```
public Void addAll(Set fromSet)
```

Parameters

fromSet

Type: Set

Return Value

Type: Void

clear()

Removes all elements from a list, consequently setting the list's length to zero.

Signature

```
public Void clear()
```

Return Value

Type: Void

clone()

Makes a duplicate copy of a list.

Signature

```
public List<Object> clone()
```

Return Value

Type: List<Object>

Usage

The cloned list is of the same type as the current list.

Note that if this is a list of sObject records, the duplicate list will only be a shallow copy of the list. That is, the duplicate will have references to each object, but the sObject records themselves will not be duplicated. For example:

To also copy the sObject records, you must use the `deepClone` method.

Example

```
Account a = new Account(Name='Acme', BillingCity='New York');

Account b = new Account();
Account[] q1 = new Account[]{a,b};
```

```
        q1[0].BillingCity);
    System.assertEquals(
        'San Francisco',
        q2[0].BillingCity);
```

contains(listElement)

Returns `true` if the list contains the specified element.

Signature

```
public Boolean contains(Object listElement)
```

Parameters

listElement

Type: Object

Return Value

Type: Boolean

Example

```
List<String> myStrings = new List<String>{'a', 'b'};
Boolean result = myStrings.contains('z');
System.assertEquals(false, result);
```

deepClone(preserveId, preserveReadonlyTimestamps, preserveAutonumber)

Makes a duplicate copy of a list of sObject records, including the sObject records themselves.

Signature

```
public List<Object> deepClone(Boolean preserveId, Boolean preserveReadonlyTimestamps, Boolean preserveAutonumber)
```

Parameters

preserveId

Type: Boolean

The optional *preserveId* argument determines whether the IDs of the original objects are preserved or cleared in the duplicates. If set to `true`, the IDs are copied to the cloned objects. The default is `false`, that is, the IDs are cleared.

preserveReadonlyTimestamps

Type: Boolean

The optional *preserveReadonlyTimestamps* argument determines whether the read-only timestamp and user ID fields are preserved or cleared in the duplicates. If set to `true`, the read-only fields `CreatedById`, `CreatedDate`, `LastModifiedById`, and `LastModifiedDate` are copied to the cloned objects. The default is `false`, that is, the values are cleared.

preserveAutonumber

Type: Boolean

Type: List<Object>

Usage

The returned list is of the same type as the current list.

> **Note**
>
> - `deepClone` only works with lists of sObjects, not with lists of primitives.
> - For Apex saved using Salesforce API version 22.0 or earlier, the default value for the *preserve_id* argument is `true`, that is, the IDs are preserved.

To make a shallow copy of a list without duplicating the sObject records it contains, use the `clone` method.

Example

This example performs a deep clone for a list with two accounts.

```
Account a = new Account(Name='Acme', BillingCity='New York');

Account b = new Account(Name='Salesforce');

Account[] q1 = new Account[]{a,b};

Account[] q2 = q1.deepClone();
q1[0].BillingCity = 'San Francisco';

System.assertEquals(
     'San Francisco',
     q1[0].BillingCity);

System.assertEquals(
     'New York',
     q2[0].BillingCity);
```

This example is based on the previous example and shows how to clone a list with preserved read-only timestamp and user ID fields.

```
insert q1;

List<Account> accts = [SELECT CreatedById, CreatedDate, LastModifiedById,
                       LastModifiedDate, BillingCity
                       FROM Account
                       WHERE Name='Acme' OR Name='Salesforce'];

// Clone list while preserving timestamp and user ID fields.
Account[] q3 = accts.deepClone(false,true,false);

// Verify timestamp fields are preserved for the first list element.
System.assertEquals(
    accts[0].CreatedById,
    q3[0].CreatedById);
System.assertEquals(
    accts[0].CreatedDate,
    q3[0].CreatedDate);
System.assertEquals(
    accts[0].LastModifiedById,
    q3[0].LastModifiedById);
System.assertEquals(
```

Compares this list with the specified list and returns `true` if both lists are equal; otherwise, returns `false`.

Signature

```
public Boolean equals(List list2)
```

Parameters

list2

Type: List

The list to compare this list with.

Return Value

Type: Boolean

Usage

Two lists are equal if their elements are equal and are in the same order. The `==` operator is used to compare the elements of the lists.

The `==` operator is equivalent to calling the `equals` method, so you can call `list1.equals(list2);` instead of `list1 == list2;`.

get(index)

Returns the list element stored at the specified index.

Signature

```
public Object get(Integer index)
```

Parameters

index

Type: Integer

Return Value

Type: Object

Usage

To reference an element of a one-dimensional list of primitives or sObjects, you can also follow the name of the list with the element's index position in square brackets as shown in the example.

Example

```
List<Integer> myList = new List<Integer>();
myList.add(47);
Integer myNumber = myList.get(0);
system.assertEquals(47, myNumber);
```

```
List<String> colors = new String[3];
colors[0] = 'Red';
colors[1] = 'Blue';
colors[2] = 'Green';
```

```
public Schema.SObjectType getSObjectType()
```

Return Value

Type: Schema.SObjectType

Usage

Use this method with describe information to determine if a list contains sObjects of a particular type.

Note that this method can only be used with lists that are composed of sObjects.

For more information, see Understanding Apex Describe Information.

Example

```
// Create a generic sObject variable.
SObject sObj = Database.query('SELECT Id FROM Account LIMIT 1');

// Verify if that sObject  variable is an Account token.
System.assertEquals(
   Account.sObjectType,
   sObj.getSObjectType());

// Create a list of generic sObjects.
List<sObject> q = new Account[]{};

// Verify if the list of sObjects
// contains Account tokens.
System.assertEquals(
   Account.sObjectType,
   q.getSObjectType());
```

hashCode()

Returns the hashcode corresponding to this list and its contents.

Signature

```
public Integer hashCode()
```

Return Value

Type: Integer

indexOf(listElement)

Returns the index of the first occurrence of the specified element in this list. If this list does not contain the element, returns -1.

Signature

```
public Integer indexOf(Object listElement)
```

Parameters

listElement

Type: Object

Return Value

```
List<String> myStrings = new List<String>{'a', 'b', 'a'};
Integer result = myStrings.indexOf('a');
System.assertEquals(0, result);
```

isEmpty()

Returns true if the list has zero elements.

Signature

```
public Boolean isEmpty()
```

Return Value

Type: Boolean

iterator()

Returns an instance of an iterator for this list.

Signature

```
public Iterator iterator()
```

Return Value

Type: Iterator

Usage

From the returned iterator, you can use the iterable methods `hasNext` and `next` to iterate through the list.

> **Note**
>
> You don't have to implement the `iterable` interface to use the `iterable` methods with a list.

See Custom Iterators.

Example

```
public class CustomIterator
   implements Iterator<Account>{

   private List<Account> accounts;
   private Integer currentIndex;

   public CustomIterator(List<Account> accounts){
       this.accounts = accounts;
       this.currentIndex = 0;
   }

   public Boolean hasNext(){
       return currentIndex < accounts.size();
   }

   public Account next(){
       if(hasNext()) {
           return accounts[currentIndex++];
```

remove(index)

Removes the list element stored at the specified index, returning the element that was removed.

Signature

```
public Object remove(Integer index)
```

Parameters

index

Type: Integer

Return Value

Type: Object

Example

```
List<String> colors = new String[3];
colors[0] = 'Red';
colors[1] = 'Blue';
colors[2] = 'Green';
String s1 = colors.remove(2);
system.assertEquals('Green', s1);
```

set(index, listElement)

Sets the specified value for the element at the given index.

Signature

```
public Void set(Integer index, Object listElement)
```

Parameters

index

Type: Integer

The index of the list element to set.

listElement

Type: Object

The value of the list element to set.

Return Value

Type: Void

Usage

To set an element of a one-dimensional list of primitives or sObjects, you can also follow the name of the list with the element's index position in square brackets.

Example

```
List<String> colors = new String[3];
colors[0] = 'Red';
colors[1] = 'Blue';
colors[2] = 'Green';
```

size()

Returns the number of elements in the list.

Signature

```
public Integer size()
```

Return Value

Type: Integer

Example

```
List<Integer> myList = new List<Integer>();
Integer size = myList.size();
system.assertEquals(0, size);

List<Integer> myList2 = new Integer[6];
Integer size2 = myList2.size();
system.assertEquals(6, size2);
```

sort()

Sorts the items in the list in ascending order.

Signature

```
public Void sort()
```

Return Value

Type: Void

Usage

Using this method, you can sort primitive types, SelectOption elements, and sObjects (standard objects and custom objects). For more information on the sort order used for sObjects, see Sorting Lists of sObjects. You can sort custom types (your Apex classes) if they implement the Comparable interface. Alternatively, a class implementing the Comparator interface can be passed as a parameter to the `List.sort` method.

When you use `sort()` methods on `List<Id>s` that contain both 15-character and 18-character IDs, IDs for the same record sort together in API version 35.0 and later.

Example

In the following example, the list has three elements. When the list is sorted, the first element is null because it has no value assigned. The second element and third element have values of 5 and 10.

```
q1[0] = 10;
q1[1] = 5;

q1.sort();
// Verify sorted list. Elements are sorted in nulls-first order: null, 5, and 10
system.assertEquals(null, q1.get(0));
system.assertEquals(5, q1.get(1));
system.assertEquals(10, q1.get(2));
```

toString()

Returns the string representation of the list.

Signature

```
public String toString()
```

Return Value

Type: String

Usage

When used in cyclic references, the output is truncated to prevent infinite recursion. When used with large collections, the output is truncated to avoid exceeding total heap size and maximum CPU time.

- Up to 10 items per collection are included in the output, followed by an ellipsis (...).
- If the same object is included multiple times in a collection, it's shown in the output only once; subsequent references are shown as `(already output)`.