Apache Commons Collections

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com


https://www.facebook.com/tutorialspointindia


https://twitter.com/tutorialspoint

About the Tutorial

The Apache Commons Collections are the components of the Apache Commons which are derived from Java API and provides component architecture for the Java language. Commons-Collections seek to build upon the JDK classes by providing new interfaces, implementations and utilities. This tutorial covers most of the topics required for a basic understanding of Apache Commons Collections and to get a feel of how it works.

Audience

This tutorial has been prepared for the beginners to help them understand the basic to advanced concepts related to Apache Commons Collections.

Prerequisites

Before you start practicing various types of examples given in this reference, we assume that you are already aware about computer programs and computer programming languages.

Copyright & Disclaimer

Table of Contents

1. Apache Commons Collections — Overview

Commons Collections augments Java Collections Framework. It provides several features to make collection handling easy. It provides many new interfaces, implementations and utilities.

The main features of Commons Collections are as follows:

- **Bag** – Bag interfaces simplifies the collections, which have multiple number of copies of each object.
- **BidiMap** – BidiMap interfaces provide Bi-Directional maps, which can be used to lookup values using keys or keys using values.
- **MapIterator** – MapIterator interface provide simple and easy iteration over maps.
- **Transforming Decorators** – Transforming decorators can alter every object of a collection as and when it is added to the collection.
- **Composite Collections** – Composite collections are used, where multiple collections are required to be handled uniformly.
- **Ordered Map** – Ordered Maps retain the order, in which elements are added in.
- **Ordered Set** – Ordered Sets retain the order, in which elements are added in.
- **Reference map** – Reference map allows key/values to be garbage collected under close control.
- **Comparator implementations** – Many Comparator implementations are available.
- **Iterator implementations** – Many Iterator implementations are available.
- **Adapter Classes** – Adapter classes are available to convert array and enumerations to collections.
- **Utilities** – Utilities are available to test or create typical set-theory properties of collections such as union, intersection. Supports Closure.

2. Apache Commons Collections — Environment Setup

Local Environment Setup

If you are still willing to set up your environment for Java programming language, then this section guides you on how to download and set up Java on your machine. Please follow the steps mentioned below to set up the environment.

Java SE is freely available from the link https://www.oracle.com/technetwork/java/archive-139210.html So, you download a version based on your operating system.

Follow the instructions to download Java and run the **.exe** to install Java on your machine. Once, you have installed Java on your machine, you would need to set environment variables to point to correct installation directories.

Setting up the Path for Windows 2000/XP

We are assuming that you have installed Java in ***c:\Program Files\java\jdk*** **directory**

- Right-click on 'My Computer' and select 'Properties'.
- Click on the 'Environment variables' button under the 'Advanced' tab.
- Now, alter the 'Path' variable so that it also contains the path to the Java executable. Example, if the path is currently set to **'C:\WINDOWS\SYSTEM32'**, then change your path to read **'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'**.

Setting up the Path for Windows 95/98/ME

We are assuming that you have installed Java in ***c:\Program Files\java\jdk*** **directory**.

- Edit the 'C:\autoexec.bat' file and add the following line at the end – **'SET PATH=%PATH%;C:\Program Files\java\jdk\bin'**

Setting up the Path for Linux, UNIX, Solaris, FreeBSD

Environment variable PATH should be set to point to where, the Java binaries have been installed. Refer to your shell documentation, if you have trouble doing this.

Example, if you use bash as your shell, then you would add the following line to the end of your '**.bashrc: export PATH=/path/to/java:$PATH'**

Popular Java Editors

To write your Java programs, you need a text editor. There are many sophisticated IDEs available in the market. But for now, you can consider one of the following:

- **Notepad** – On Windows machine, you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.
- **Netbeans** – It is a Java IDE that is open-source and free, which can be downloaded from https://www.netbeans.org/index.html.
- **Eclipse** – It is also a Java IDE developed by the eclipse open-source community and can be downloaded from https://www.eclipse.org/.

Download Common Collections Archive

Download the latest version of Apache Common Collections jar file from commons-collections4-4.1-bin.zip. At the time of writing this tutorial, we have downloaded ***commons-collections4-4.1-bin.zip*** and copied it into **C:\>Apache folder**.

OS	Archive name
Windows	commons-collections4-4.1-bin.zip
Linux	commons-collections4-4.1-bin.tar.gz
Mac	commons-collections4-4.1-bin.tar.gz

Set Apache Common Collections Environment

Set the **APACHE_HOME** environment variable to point to the base directory location where Apache jar is stored on your machine. Assuming, we've extracted commons-collections4-4.1-bin.zip in Apache folder on various Operating Systems as follows: -

OS	Output
Windows	Set the environment variable APACHE_HOME to C:\Apache
Linux	export APACHE_HOME=/usr/local/Apache
Mac	export APACHE_HOME=/Library/Apache

Set CLASSPATH Variable

Set the **CLASSPATH** environment variable to point to the Common Collections jar location. Assuming, you have stored commons-collections4-4.1-bin.zip in Apache folder on various Operating Systems as follows:

OS	Output
Windows	Set the environment variable CLASSPATH to %CLASSPATH%;%APACHE_HOME%\commons-collections4-4.1-bin.jar;.;
Linux	export CLASSPATH=$CLASSPATH:$APACHE_HOME/commons-collections4-4.1-bin.jar:.
Mac	export CLASSPATH=$CLASSPATH:$APACHE_HOME/commons-collections4-4.1-bin.jar:.

3. Apache Commons Collections — Bag Interface

New Interfaces are added to supports bags. A Bag defines a collection which, counts the number of times an object appears in the collection. For example, if a Bag contains {a, a, b, c} then getCount("a") will return 2 while uniqueSet() returns the unique values.

Interface Declaration

Following is the declaration for org.apache.commons.collections4.Bag<E> interface:

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

Methods

The methods for bag inference are as follows:

Sr.No.	Method & Description
1	**boolean add(E object)** (Violation) Adds one copy of the specified object to the Bag.
2	**boolean add(E object, int nCopies)** Adds nCopies copies of the specified object to the Bag.
3	**boolean containsAll(Collection<?> coll)** (Violation) Returns true if the bag contains all elements in the given collection, respecting cardinality.
4	**int getCount(Object object)** Returns the number of occurrences (cardinality) of the given object currently in the bag.
5	**Iterator<E> iterator()** Returns an Iterator over the entire set of members, including copies due to cardinality.
6	**boolean remove(Object object)** (Violation) Removes all occurrences of the given object from the bag.
7	**boolean remove(Object object, int nCopies)**

	Removes nCopies copies of the specified object from the Bag.
8	**boolean removeAll(Collection<?> coll)** (Violation) Remove all elements represented in the given collection, respecting cardinality.
9	**boolean retainAll(Collection<?> coll)** (Violation) Remove any members of the bag that are not in the given collection, respecting cardinality.
10	**int size()** Returns the total number of items in the bag across all types.
11	**Set<E> uniqueSet()** Returns a Set of unique elements in the Bag.

Methods Inherited

This interface inherits methods from the following interface:

- java.util.Collection

Example of Bag Interface

An example of BagTester.java is as follows:

```
import org.apache.commons.collections4.Bag;
import org.apache.commons.collections4.bag.HashBag;

public class BagTester {
   public static void main(String[] args) {
      Bag<String> bag = new HashBag<>();

      //add "a" two times to the bag.
      bag.add("a" , 2);

      //add "b" one time to the bag.
      bag.add("b");
```

```
      //add "c" one time to the bag.
      bag.add("c");

      //add "d" three times to the bag.
      bag.add("d",3);

      //get the count of "d" present in bag.
      System.out.println("d is present " + bag.getCount("d") + " times.");
      System.out.println("bag: " +bag);

      //get the set of unique values from the bag
      System.out.println("Unique Set: " +bag.uniqueSet());

      //remove 2 occurrences of "d" from the bag
      bag.remove("d",2);
      System.out.println("2 occurences of d removed from bag: " +bag);
      System.out.println("d is present " + bag.getCount("d") + " times.");
      System.out.println("bag: " +bag);
      System.out.println("Unique Set: " +bag.uniqueSet());
   }
}
```

Output

You will see the following output:

```
d is present 3 times.
bag: [2:a,1:b,1:c,3:d]
Unique Set: [a, b, c, d]
2 occurences of d removed from bag: [2:a,1:b,1:c,1:d]
d is present 1 times.
bag: [2:a,1:b,1:c,1:d]
Unique Set: [a, b, c, d]
```

4. Apache Commons Collections — BidiMap Interface

New Interfaces are added to supports bidirectional Map. Using bidirectional map, a key can be lookup using value and value can be lookup using key easily.

Interface Declaration

Following is the declaration for **org.apache.commons.collections4.BidiMap<K,V>** interface:

```
public interface BidiMap<K,V>
   extends IterableMap<K,V>
```

Methods

The methods for the BidiMap Interface are as follows:

Sr.No.	Method & Description
1	**K getKey(Object value)** Gets the key that is currently mapped to the specified value.
2	**BidiMap<V,K> inverseBidiMap()** Gets a view of this map where the keys and values are reversed.
3	**V put(K key, V value)** Puts the key-value pair into the map, replacing any previous pair.
4	**K removeValue(Object value)** Removes the key-value pair that is currently mapped to the specified value (optional operation).
5	**Set<V> values()** Returns a Set view of the values contained in this map.

Methods Inherited

This interface inherits methods from the following interfaces:

- org.apache.commons.collections4.Get

- org.apache.commons.collections4.IterableGet
- org.apache.commons.collections4.Put
- java.util.Map

Example of BidiMap Interface

An example of BidiMapTester.java is as follows:

```
import org.apache.commons.collections4.BidiMap;
import org.apache.commons.collections4.bidimap.TreeBidiMap;

public class BidiMapTester {
   public static void main(String[] args) {
      BidiMap<String, String> bidi = new TreeBidiMap<>();

      bidi.put("One", "1");
      bidi.put("Two", "2");
      bidi.put("Three", "3");

      System.out.println(bidi.get("One"));
      System.out.println(bidi.getKey("1"));
      System.out.println("Original Map: " + bidi);

      bidi.removeValue("1");
      System.out.println("Modified Map: " + bidi);
      BidiMap<String, String> inversedMap = bidi.inverseBidiMap();
      System.out.println("Inversed Map: " + inversedMap);
   }
}
```

Output

When you run the code, you will see the following output:

```
1
One
Original Map: {One=1, Three=3, Two=2}
Modified Map: {Three=3, Two=2}
Inversed Map: {2=Two, 3=Three}
```

5. Apache Commons Collections — MapIterator Interface

The JDK Map interface is pretty difficult to iterate as Iteration to be done on EntrySet or over the KeySet objects. MapIterator provides simple iteration over Map. Following example illustrates the same.

Example of MapIterator Interface

An example for MapIteratorTester.java is as follows:

```
import org.apache.commons.collections4.IterableMap;
import org.apache.commons.collections4.MapIterator;
import org.apache.commons.collections4.map.HashedMap;

public class MapIteratorTester {
   public static void main(String[] args) {
      IterableMap<String, String> map = new HashedMap<>();

      map.put("1", "One");
      map.put("2", "Two");
      map.put("3", "Three");
      map.put("4", "Four");
      map.put("5", "Five");

      MapIterator<String, String> iterator = map.mapIterator();
      while (iterator.hasNext()) {
         Object key = iterator.next();
         Object value = iterator.getValue();

         System.out.println("key: " + key);
         System.out.println("Value: " + value);

         iterator.setValue(value + "_");
      }

      System.out.println(map);
```

```
   }
}
```

Output

The output is stated below:

```
key: 3
Value: Three
key: 5
Value: Five
key: 2
Value: Two
key: 4
Value: Four
key: 1
Value: One
{3=Three_, 5=Five_, 2=Two_, 4=Four_, 1=One_}
```

6. Apache Common Collections — OrderedMap Interface

OrderedMap is a new interface for maps to retain the order in which elements are added. LinkedMap and ListOrderedMap are two available implementations. This interfaces supports iterator that of Map and allows iteration in both directions either forwards or backwards in a Map. Following example illustrates the same.

Example of MapIterator Interface

An example of OrderedMapTester.java is as given below:

```
import org.apache.commons.collections4.OrderedMap;
import org.apache.commons.collections4.map.LinkedMap;

public class OrderedMapTester {
   public static void main(String[] args) {
      OrderedMap<String, String> map = new LinkedMap<String, String>();
      map.put("One", "1");
      map.put("Two", "2");
      map.put("Three", "3");

      System.out.println(map.firstKey());
      System.out.println(map.nextKey("One"));
      System.out.println(map.nextKey("Two"));
   }
}
```

Output

The result will be as follows:

```
One
Two
Three
```

7. Apache Commons Collections — Ignore Null

CollectionUtils class of Apache Commons Collections library provides various utility methods for common operations covering wide range of use cases. It helps avoid writing boilerplate code. This library is very useful prior to jdk 8 as similar functionalities are now provided in Java 8's Stream API.

Check for Not Null Elements

addIgnoreNull() method of CollectionUtils can be used to ensure that only non-null values are getting added to the collection.

Declaration

Following is the declaration for

org.apache.commons.collections4.CollectionUtils.addIgnoreNull() method:

```
public static <T> boolean addIgnoreNull(Collection<T> collection, T object)
```

Parameters

- **collection** – The collection to add to, must not be null.
- **object** – The object to add, if null it will not be added.

Return Value

True if the collection changed.

Exception

- **NullPointerException** – If the collection is null.

Example

The following example shows the usage of **org.apache.commons.collections4.CollectionUtils.addIgnoreNull()** method. We are trying to add a null value and a sample non-null value.

```
import java.util.LinkedList;
import java.util.List;

import org.apache.commons.collections4.CollectionUtils;

public class CollectionUtilsTester {
```

```
   public static void main(String[] args) {
      List<String> list = new LinkedList<String>();

      CollectionUtils.addIgnoreNull(list, null);
      CollectionUtils.addIgnoreNull(list, "a");

      System.out.println(list);

      if(list.contains(null)) {
         System.out.println("Null value is present");
      } else {
         System.out.println("Null value is not present");
      }
   }
}
```

Output

The output is mentioned below:

```
[a]
Null value is not present
```

8. Apache Commons Collections — Merge & Sort

CollectionUtils class of Apache Commons Collections library provides various utility methods for common operations covering wide range of use cases. It helps avoid writing boilerplate code. This library is very useful prior to jdk 8 as similar functionalities are now provided in Java 8's Stream API.

Merging two sorted lists

collate() method of CollectionUtils can be used to merge two already sorted lists.

Declaration

Following is the declaration for

org.apache.commons.collections4.CollectionUtils.collate() method:

```
public static <O extends Comparable<? super O>> List<O>
   collate(Iterable<? extends O> a, Iterable<? extends O> b)
```

Parameters

- **a** – The first collection, must not be null.
- **b** – The second collection, must not be null.

Return Value

A new sorted List, containing the elements of Collection a and b.

Exception

- **NullPointerException** – If either collection is null.

Example

The following example shows the usage of **org.apache.commons.collections4.CollectionUtils.collate()** method. We'll merge two sorted lists and then print the merged and sorted list.

```
import java.util.Arrays;
import java.util.List;

import org.apache.commons.collections4.CollectionUtils;



public class CollectionUtilsTester {
```

```
   public static void main(String[] args) {
      List<String> sortedList1 = Arrays.asList("A","C","E");
      List<String> sortedList2 = Arrays.asList("B","D","F");
      List<String> mergedList = CollectionUtils.collate(sortedList1,
sortedList2);
      System.out.println(mergedList);
   }
}
```

Output

The output is as follows:

```
[A, B, C, D, E, F]
```

9. Apache Commons Collections — Transforming Objects

CollectionUtils class of Apache Commons Collections library provides various utility methods for common operations covering wide range of use cases. It helps avoid writing boilerplate code. This library is very useful prior to jdk 8 as similar functionalities are now provided in Java 8's Stream API.

Transforming a list

collect() method of CollectionUtils can be used to transform a list of one type of objects to list of different type of objects.

Declaration

Following is the declaration for

org.apache.commons.collections4.CollectionUtils.collect() method:

```
public static <I,O> Collection<O> collect(Iterable<I> inputCollection,
   Transformer<? super I,? extends O> transformer)
```

Parameters

- **inputCollection** – The collection to get the input from, may not be null.
- **Transformer** – The transformer to use, may be null.

Return Value

The transformed result (new list).

Exception

- **NullPointerException** – If the input collection is null.

Example

The following example shows the usage of **org.apache.commons.collections4.CollectionUtils.collect()** method. We'll transform a list of string to list of integer by parsing the integer value from String.

```
import java.util.Arrays;
import java.util.List;

import org.apache.commons.collections4.CollectionUtils;
import org.apache.commons.collections4.Transformer;
```

```
public class CollectionUtilsTester {
   public static void main(String[] args) {
      List<String> stringList = Arrays.asList("1","2","3");

      List<Integer> integerList = (List<Integer>)
CollectionUtils.collect(stringList,
         new Transformer<String, Integer>() {

         @Override
         public Integer transform(String input) {
            return Integer.parseInt(input);
         }
      });

      System.out.println(integerList);
   }
}
```

Output

When you use the code, you will get the following code:

```
[1, 2, 3]
```

10. Apache Commons Collections — Filtering Objects

CollectionUtils class of Apache Commons Collections library provides various utility methods for common operations covering wide range of use cases. It helps avoid writing boilerplate code. This library is very useful prior to jdk 8 as similar functionalities are now provided in Java 8's Stream API.

filter() method

filter() method of CollectionUtils can be used to filter a list to remove objects which do not satisfy condition provided by predicate passed.

Declaration

Following is the declaration for

org.apache.commons.collections4.CollectionUtils.filter() method:

```
public static <T> boolean filter(Iterable<T> collection,
   Predicate<? super T> predicate)
```

Parameters

- **collection** – The collection to get the input from, may not be null.
- **predicate** – The predicate to use as a filter, may be null.

Return Value

True if the collection is modified by this call, false otherwise.

Example

The following example shows the usage of **org.apache.commons.collections4.CollectionUtils.filter()** method. We'll filter a list of integer to get even numbers only.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.apache.commons.collections4.CollectionUtils;
import org.apache.commons.collections4.Predicate;



public class CollectionUtilsTester {
```

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

      List<Integer> integerList = new ArrayList<Integer>();
      integerList.addAll(Arrays.asList(1,2,3,4,5,6,7,8));

      System.out.println("Original List: " + integerList);
      CollectionUtils.filter(integerList, new Predicate<Integer>() {

         @Override
         public boolean evaluate(Integer input) {
            if(input.intValue() % 2 == 0) {
               return true;
            }
            return false;
         }
      });

      System.out.println("Filtered List (Even numbers): " + integerList);
   }
}
```

Output

It will produce the following result:

```
Original List: [1, 2, 3, 4, 5, 6, 7, 8]
Filtered List (Even numbers): [2, 4, 6, 8]
```

filterInverse() method

filterInverse() method of CollectionUtils can be used to filter a list to remove objects, which satisfy condition provided by predicate passed.

Declaration

Following is the declaration for

org.apache.commons.collections4.CollectionUtils.filterInverse() method:

```
public static <T> boolean filterInverse(Iterable<T> collection,
   Predicate<? super T> predicate)
```

Parameters

- **collection** – The collection to get the input from, may not be null.
- **predicate** – The predicate to use as a filter, may be null.

Return Value

True if the collection is modified by this call, false otherwise.

Example

The following example shows the usage of **org.apache.commons.collections4.CollectionUtils.filterInverse()** method. We'll filter a list of integer to get odd numbers only.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.apache.commons.collections4.CollectionUtils;
import org.apache.commons.collections4.Predicate;

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

      List<Integer> integerList = new ArrayList<Integer>();
      integerList.addAll(Arrays.asList(1,2,3,4,5,6,7,8));

      System.out.println("Original List: " + integerList);
      CollectionUtils.filterInverse(integerList, new Predicate<Integer>() {

         @Override
         public boolean evaluate(Integer input) {
            if(input.intValue() % 2 == 0) {
               return true;
            }
            return false;
         }

      });
```

```
      System.out.println("Filtered List (Odd numbers): " + integerList);
   }
}
```

Output

The result is as stated below:

```
Original List: [1, 2, 3, 4, 5, 6, 7, 8]
Filtered List (Odd numbers): [1, 3, 5, 7]
```

11. Apache Commons Collections — Safe Empty Checks

CollectionUtils class of Apache Commons Collections library provides various utility methods for common operations covering wide range of use cases. It helps avoid writing boilerplate code. This library is very useful prior to jdk 8 as similar functionalities are now provided in Java 8's Stream API.

Checking non-empty list

isNotEmpty() method of CollectionUtils can be used to check if a list is not empty without worrying about null list. So null check is not required to be placed everywhere before checking the size of the list.

Declaration

Following is the declaration for

org.apache.commons.collections4.CollectionUtils.isNotEmpty() method:

```
public static boolean isNotEmpty(Collection<?> coll)
```

Parameters

- **coll** – The collection to check, may be null.

Return Value

True if non-null and non-empty.

Example

The following example shows the usage of **org.apache.commons.collections4.CollectionUtils.isNotEmpty()** method. We'll check a list is empty or not.

```
import java.util.List;

import org.apache.commons.collections4.CollectionUtils;

public class CollectionUtilsTester {
   public static void main(String[] args) {
      List<String> list = getList();
      System.out.println("Non-Empty List Check: " + checkNotEmpty1(list));
      System.out.println("Non-Empty List Check: " + checkNotEmpty1(list));
```

```
   }

   static List<String> getList() {
      return null;
   }

   static boolean checkNotEmpty1(List<String> list) {
      return !(list == null || list.isEmpty());
   }

   static boolean checkNotEmpty2(List<String> list) {
      return CollectionUtils.isNotEmpty(list);
   }
}
```

Output

The output is given below:

```
Non-Empty List Check: false
Non-Empty List Check: false
```

Checking empty list

isEmpty() method of CollectionUtils can be used to check if a list is empty without worrying about null list. So null check is not required to be placed everywhere before checking the size of the list.

Declaration

Following is the declaration for

org.apache.commons.collections4.CollectionUtils.isEmpty() method:

```
public static boolean isEmpty(Collection<?> coll)
```

Parameters

- **coll** – The collection to check, may be null.

Return Value

True if empty or null.

Example

The following example shows the usage of **org.apache.commons.collections4.CollectionUtils.isEmpty()** method. We'll check a list is empty or not.

```
import java.util.List;

import org.apache.commons.collections4.CollectionUtils;

public class CollectionUtilsTester {
   public static void main(String[] args) {
      List<String> list = getList();
      System.out.println("Empty List Check: " + checkEmpty1(list));
      System.out.println("Empty List Check: " + checkEmpty1(list));
   }

   static List<String> getList() {
      return null;
   }

   static boolean checkEmpty1(List<String> list) {
      return (list == null || list.isEmpty());
   }

   static boolean checkEmpty2(List<String> list) {
      return CollectionUtils.isEmpty(list);
   }
}
```

Output

Given below is the output of the code:

```
Empty List Check: true
Empty List Check: true
```

12. Apache Commons Collections — Inclusion

CollectionUtils class of Apache Commons Collections library provides various utility methods for common operations covering wide range of use cases. It helps avoid writing boilerplate code. This library is very useful prior to jdk 8 as similar functionalities are now provided in Java 8's Stream API.

Checking sublist

isSubCollection() method of CollectionUtils can be used to check if a collection contains the given collection or not.

Declaration

Following is the declaration for

org.apache.commons.collections4.CollectionUtils.isSubCollection() method:

```
public static boolean isSubCollection(Collection<?> a,
   Collection<?> b)
```

Parameters

- **a** – The first (sub) collection, must not be null.
- **b** – The second (super) collection, must not be null.

Return Value

True if and only if a is a sub-collection of b.

Example

The following example shows the usage of **org.apache.commons.collections4.CollectionUtils.isSubCollection()** method. We'll check a list is part of another list or not.

```
import java.util.Arrays;
import java.util.List;

import org.apache.commons.collections4.CollectionUtils;

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

```
      //checking inclusion
      List<String> list1 = Arrays.asList("A","A","A","C","B","B");
      List<String> list2 = Arrays.asList("A","A","B","B");

      System.out.println("List 1: " + list1);
      System.out.println("List 2: " + list2);
      System.out.println("Is List 2 contained in List 1: "
         + CollectionUtils.isSubCollection(list2, list1));
   }
}
```

Output

You will receive the following output:

```
List 1: [A, A, A, C, B, B]
List 2: [A, A, B, B]
Is List 2 contained in List 1: true
```

13. Apache Commons Collections — Intersection

CollectionUtils class of Apache Commons Collections library provides various utility methods for common operations covering wide range of use cases. It helps avoid writing boilerplate code. This library is very useful prior to jdk 8 as similar functionalities are now provided in Java 8's Stream API.

Checking intersection

intersection() method of CollectionUtils can be used to get the common objects between two collections(intersection).

Declaration

Following is the declaration for **org.apache.commons.collections4.CollectionUtils.intersection()** method:

```
public static <O> Collection<O> intersection(Iterable<? extends O> a,
   Iterable<? extends O> b)
```

Parameters

- **a** – The first (sub) collection, must not be null.
- **b** – The second (super) collection, must not be null.

Return Value

The intersection of the two collections.

Example

The following example shows the usage of **org.apache.commons.collections4.CollectionUtils.intersection()** method. We'll get the intersection of two lists.

```
import java.util.Arrays;
import java.util.List;

import org.apache.commons.collections4.CollectionUtils;

public class CollectionUtilsTester {
   public static void main(String[] args) {
      //checking inclusion
```

```
      List<String> list1 = Arrays.asList("A","A","A","C","B","B");
      List<String> list2 = Arrays.asList("A","A","B","B");

      System.out.println("List 1: " + list1);
      System.out.println("List 2: " + list2);
      System.out.println("Commons Objects of List 1 and List 2: "
         + CollectionUtils.intersection(list1, list2));
   }
}
```

Output

When you run the code, you will see the following output:

```
List 1: [A, A, A, C, B, B]
List 2: [A, A, B, B]
Commons Objects of List 1 and List 2: [A, A, B, B]
```

14. Apache Commons Collections — Subtraction

CollectionUtils class of Apache Commons Collections library provides various utility methods for common operations covering wide range of use cases. It helps avoid writing boilerplate code. This library is very useful prior to jdk 8 as similar functionalities are now provided in Java 8's Stream API.

Checking Substraction

subtract() method of CollectionUtils can be used to get the new collection by subtracting objects of one collection from other.

Declaration

Following is the declaration for **org.apache.commons.collections4.CollectionUtils.subtract()** method:

```
public static <O> Collection<O> subtract(Iterable<? extends O> a,
   Iterable<? extends O> b)
```

Parameters

- **a** – The collection to subtract from, must not be null.
- **b** – The collection to subtract, must not be null.

Return Value

A new collection with the results.

Example

The following example shows the usage of **org.apache.commons.collections4.CollectionUtils.subtract()** method. We'll get the subtraction of two lists.

```
import java.util.Arrays;
import java.util.List;

import org.apache.commons.collections4.CollectionUtils;

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

      //checking inclusion
```

```
      List<String> list1 = Arrays.asList("A","A","A","C","B","B");
      List<String> list2 = Arrays.asList("A","A","B","B");

      System.out.println("List 1: " + list1);
      System.out.println("List 2: " + list2);
      System.out.println("List 1 - List 2: "
         + CollectionUtils.subtract(list1, list2));
   }
}
```

Output

When you execute the above code, you should see the following output:

```
List 1: [A, A, A, C, B, B]
List 2: [A, A, B, B]
List 1 - List 2: [A, C]
```

15. Apache Commons Collections — Union

CollectionUtils class of Apache Commons Collections library provides various utility methods for common operations covering wide range of use cases. It helps avoid writing boilerplate code. This library is very useful prior to jdk 8 as similar functionalities are now provided in Java 8's Stream API.

Checking union

union() method of CollectionUtils can be used to get the union of two collections.

Declaration

Following is the declaration for **org.apache.commons.collections4.CollectionUtils.union()** method:

```
public static <O> Collection<O> union(Iterable<? extends O> a,
   Iterable<? extends O> b)
```

Parameters

- **a** – The first collection, must not be null.
- **b** – The second collection, must not be null.

Return Value

The union of the two collections.

Example

The following example shows the usage of **org.apache.commons.collections4.CollectionUtils.union()** method. We'll get the union of two lists.

```
import java.util.Arrays;
import java.util.List;

import org.apache.commons.collections4.CollectionUtils;

public class CollectionUtilsTester {
   public static void main(String[] args) {
      //checking inclusion
```

```
      List<String> list1 = Arrays.asList("A","A","A","C","B","B");
      List<String> list2 = Arrays.asList("A","A","B","B");

      System.out.println("List 1: " + list1);
      System.out.println("List 2: " + list2);
      System.out.println("Union of List 1 and List 2: "
         + CollectionUtils.union(list1, list2));
   }
}
```

Output

This produces the following output:

```
List 1: [A, A, A, C, B, B]
List 2: [A, A, B, B]
Union of List 1 and List 2: [A, A, A, B, B, C]
```