Apache Commons Collections Equivalents

CollectionUtils (source)

CollectionUtils	Guava
`void addAll(Collection, Enumeration)`	`Iterators.addAll(collection, Iterators.forEnumeration(enumeration))`
`void addAll(Collection, Iterator)`	`Iterators.addAll(collection, iterator)`
`void addAll(Collection, Object[])`	`Collections.addAll(collection, array)` (JDK)
`boolean addIgnoreNull(Collection, Object)`	`if (o != null) { collection.add(o); }`
`int cardinality(Object, Collection)`	`Iterables.frequency(collection, object)`
`Collection collect(Collection, Transformer)`	`newArrayList(Collections2.transform(input, function))`
`Collection collect(Collection, Transformer, Collection)`	`output.addAll(Collections2.transform(input, function))`
`Collection collect(Iterator, Transformer)`	`newArrayList(Iterators.transform(input, function))`
`Collection collect(Iterator, Transformer, Collection)`	`Iterators.addAll(output, Iterators.transform(input, function))`
`boolean containsAny(Collection coll1, Collection coll2)`	`!Collections.disjoint(coll1, coll2)` (JDK)
`int countMatches(Collection, Predicate)`	`Iterables.size(Iterables.filter(collection, predicate))`
`Collection disjunction(Collection, Collection)`	`Sets.symmetricDifference(set1, set2)`
`boolean exists(Collection, Predicate)`	`Iterables.any(collection, predicate)`
`void filter(Collection, Predicate)`	`Iterables.removeIf(collection, not(predicate))` (see also `Iterables.transform`, which creates a view instead of mutating the input)
`Object find(Collection, Predicate)`	`Iterables.find(collection, predicate)`
`void forAllDo(Collection, Closure)`	`for (Object o : collection) { closure.execute(o); }`
`Object get(Object, int)`	`Iterables.get(o, index)`, supplemented with calls to `entrySet()`, `forEnumeration()`, etc.

`Map getCardinalityMap(Collection)`	`ImmutableMultiset.copyOf(collection)`
`Object index(Object, int)`	`Iterables.get(o, index)`, supplemented with calls to `keySet()`, `forEnumeration()`, etc.
`Object index(Object, Object)`	`Iterables.get(o, index)`, supplemented with calls to `entrySet()`, `forEnumeration()`, etc.
`Collection intersection(Collection, Collection)`	`Sets/Multisets.intersection(a, b)`
`boolean isEmpty(Collection)`	`collection == null`
`boolean isEqualCollection(Collection, Collection)`	If both are `Set`s or `Multiset`s, use `equals()`; otherwise `ImmutableMultiset.copyOf(a).equals(ImmutableMultiset.copyOf(b)`
`boolean isFull(Collection)`	No equivalent--no `BoundedCollection` type.
`boolean isNotEmpty(Collection)`	`collection != null && !collection.isEmpty()`
`boolean isProperSubCollection(Collection, Collection)`	No equivalent--check that `a.size() < b.size()` and then use the check described below.
`boolean isSubCollection(Collection, Collection)`	`Multisets.containsOccurrences(ImmutableMultiset.copyOf(coll1), ImmutableMultiset.copyOf(coll2))`
`int maxSize(Collection)`	No equivalent--no `BoundedCollection` type.
`Collection predicatedCollection(Collection, Predicate)`	`Constraints.constrainedCollection/List/Set`/etc.
`Collection removeAll(Collection, Collection)`	`newArrayList(Iterables.filter(collection, Predicates.not(Predicates.in(remove))))`
`Collection retainAll(Collection, Collection)`	`newArrayList(Iterables.filter(collection, Predicates.in(retain)))`
`void reverseArray(Object[])`	`Lists.reverse(Arrays.asList(array))` (returns an inverse `List` view without modifying array)
`Collection select(Collection, Predicate)`	`newArrayList(Iterables.filter(collection, predicate))`
`void select(Collection, Predicate, Collection)`	`Iterables.addAll(output, Iterables.filter(input, predicate))`
`Collection selectRejected(Collection, Predicate)`	`newArrayList(Iterables.filter(collection, Predicates.not(predicate)))`
`void selectRejected(Collection,`	`Iterables.addAll(output, Iterables.filter(input,`

`Predicate, Collection)`	`Predicates.not(predicate)))`
`int size(Object)`	`Collection/Map.size()`, `array.length`, `Iterables/Iterators.size` (with `forEnumeration()` if necessary)
`boolean sizeIsEmpty(Object)`	`Collection/Map.isEmpty()`, `array.length == 0`, `Iterables/Iterators.isEmpty` (with `forEnumeration()` if necessary)
`Collection subtract(Collection, Collection)`	No equivalent--create an `ArrayList` containing `a` and then call `remove` on it for each element in `b`.
`Collection synchronizedCollection(Collection)`	`Collections.synchronizedCollection(collection)` (JDK)
`void transform(Collection, Transformer)`	No equivalent for transforming a `Collection` in place... not very useful. Prefer transformed views (`Lists/Collections2.transform`) or copies of them.
`Collection transformedCollection(Collection, Transformer)`	No equivalent for transforming `Object`s that are added to a `Collection`... a `ForwardingCollection` could easily handle this, though.
`Collection typedCollection(Collection, Class)`	`Collections.checkedCollection/Set/List`/etc. (JDK)
`Collection union(Collection, Collection)`	`Sets.union(a, b)`
`Collection unmodifiableCollection(Collection)`	`Collections.unmodifiableCollection/Set/List`/etc. (JDK) Consider `ImmutableCollection` types if you want immutability.