10 Collections Tricks

… you're not using but should be

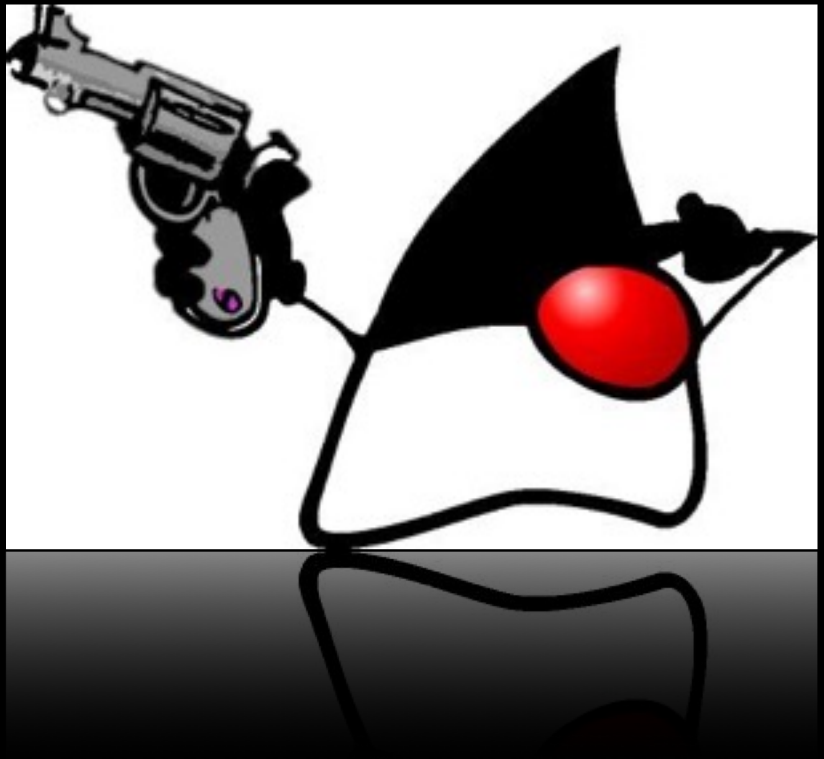
https://github.com/steveswing/ten-collections-tricks.git

Steve Swing

- Java Developer since 1999
- Independent Consultant since 2003
- Associated with Manifest Solutions since 2004
  - Bootcamp Instructor
- CodeMash Speaker (1st time 2016)
- @sswing

Java Collections

“It’d be a shame for all your nice source code to disappear all at once.” – Guido Duke



https://duke.kenai.com/gun/Gun.jpg

No… not Guido Duke Collections…

Java Collections **API**

Java Collections API

- Arrays
- Lists
  - ArrayList
  - LinkedList
- Maps
  - HashMap
  - LinkedHashMap
  - SortedMap
    - TreeMap
- Sets
  - HashSet
  - LinkedHashSet
  - SortedSet
    - TreeSet
- Queue
- Deque
- Collections class
  - sort()
  - binarySearch()
- Comparators
- Comparable

Bonus Java Collections API: `java.util.concurrent.*`

- Lists
  - CopyOnWriteArrayList
- Maps
  - ConcurrentHashMap
  - ConcurrentSkipListMap
- Sets
  - CopyOnWriteArraySet
  - ConcurrentSkipListSet

- Queue
  - BlockingQueue
    - ArrayBlockingQueue, DelayQueue, ForwardBlockingQueue, LinkedBlockingQueue, LinkedTransferQueue, PriorityBlockingQueue, SynchronousQueue
- Deque
  - BlockingDeque
    - ForwardBlockingDeque, LinkedBlockingDeque

Guava (fka Google Collections) API:

`com.google.common.collect.*`

Utility Classes

- Collections2
- Iterables
- Iterators
- Lists
- Maps
- Multimaps
- MultimapBuilder
- Multisets
- Queues
- Sets
- Tables

Collection Classes

Queue

- ArrayListMultimap
- ArrayTable
- ConcurrentHashMultiset
- ContiguousSet
- EnumBiMap, EnumHashBiMap, EnumMultiset
- HashBasedTable, HashBiMap, HashMultimap, HashMultiset
- LinkedHashMultimap, LinkedHashMultiset, LinkedListMultimap
- Forwarding* Decorator Pattern
- Immutable*

https://github.com/google/guava/wiki/NewCollectionTypesExplained

Guava (Google Collections) API:

com.google.common.collect Classes with Utility Methods

- Lists
  - asList()
  - newArrayList()
  - newLinkedList()
- Maps
  - newHashMap()
  - newLinkedHashMap()
  - SortedMap
    - newTreeMap()
- Sets
  - newHashSet()
  - newLinkedHashSet()
  - SortedSet
    - newTreeSet()

- Queues
- Collections2 class
  - filter()
  - transform()
- MapMaker
- Ordering
- Tables

Apache Commons Collections

Utility Classes

- BagUtils
- CollectionUtils
- ComparatorUtils
- IterableUtils
- IteratorUtils
- ListUtils
- MapUtils
- MultiMapUtils
- MultiSetUtils
- PredicateUtils
- QueueUtils
- SetUtils
- SplitMapUtils
- TransformerUtils
- TrieUtils

- Bag, Trie - unique to commons-collections
- Some functionality overlap with Guava Collections2

List<T>

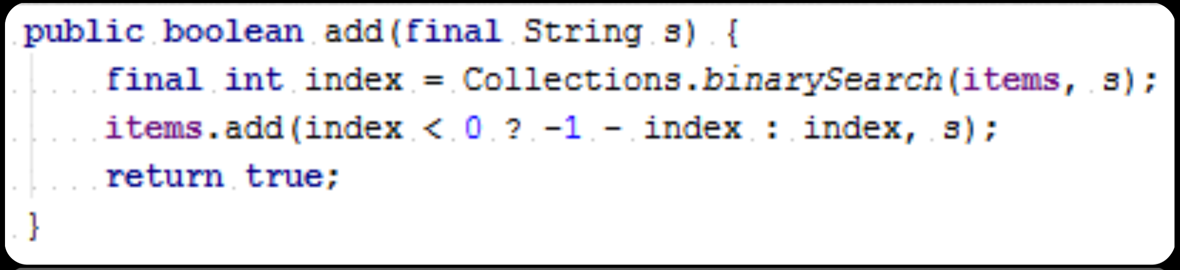
- Ordered
- Allows duplicates
- Positional access (zero-based)
- Search
- Iteration
- Range-view

- Applies to
  - ArrayList<T>
  - LinkedList<T>

List<T>
ArrayList<T>

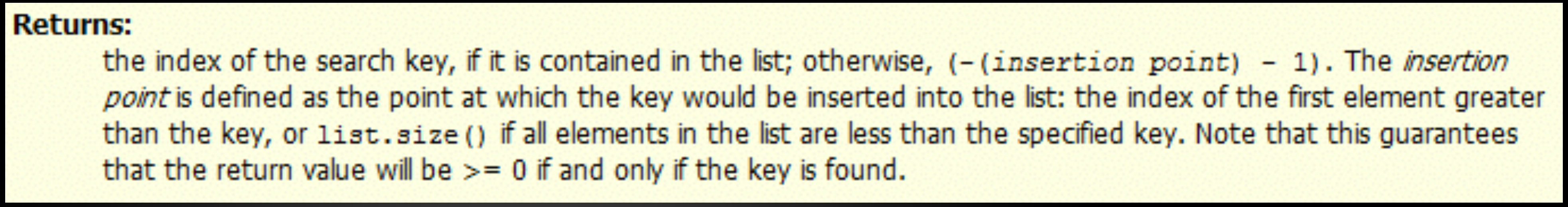
- Ordered
- Allows duplicates
- Positional access (zero-based)
- Search
- Iteration
- Range-view
- Memory allocated in contiguous blocks

- **Trick #0**: build an ArrayList in (natural) sorted order



```java
public boolean add(final String s) {
    final int index = Collections.binarySearch(items, s);
    items.add(index < 0 ? -1 - index : index, s);
    return true;
}
```

Subtract the negative index from minus one to get the insertion point.



Returns:

the index of the search key, if it is contained in the list; otherwise, `(-(insertion point) - 1)`. The *insertion point* is defined as the point at which the key would be inserted into the list: the index of the first element greater than the key, or `list.size()` if all elements in the list are less than the specified key. Note that this guarantees that the return value will be >= 0 if and only if the key is found.

Comparators
(an aside)

- Comparator is an interface of a comparison function.
- Not to be confused with Comparable though related
- Comparable is for "self" while Comparator is for others.
- **Trick #1:** Force specific "sort" order with a specialized comparator.
- For extreme control over String sorting see java.text.RuleBasedCollator (it extends Collator that implements Comparator).

List<T>
LinkedList<T>

- Ordered
- Allows duplicates
- Doubly linked list
- Not random access – index access traversal required from nearest end.
- Benefits are scalable memory allocation and garbage collection.
- Also implements Iterable<T>, Deque<T>, Queue<T>

`Set<T>`

- Unique – No Duplicates!
- Ordered & Unordered
- Some implementations disallow null elements.

- HashSet<T>
- LinkedHashSet<T>
- TreeSet<T>
- EnumSet<E>

`Set<T>`
`HashSet<T>`

- Unique – No Duplicates!
- Unordered
- Iterate from Collection – order is not guaranteed.

- **Note**: Constructing HashSets.

`Set<T>`

`LinkedHashSet<T>`

- Unique – No Duplicates!
- Retains insertion order
- Iterator (from Collection) insertion order.

- **Bonus Tip**: Use LinkedHashSet when consistent order but not necessarily sorted order is required.

Set<T>

SortedSet<T> → TreeSet<T>

- Unique – No Duplicates!
- Ordered
- Elements must implement Comparable<T> or the SortedSet<T> must have a Comparator<T> when constructed.
- Supports subsets (headSet, tailSet)

- **Trick #2**: Use SortedSet<T>.first() and SortedSet<T>.last() as min() max() for any number of values.
- **Trick #3**: Use .headSet(), .tailSet() methods to find the next item in sequence for sparse values. For example Holidays/Weekends.

`Map<K, V>`

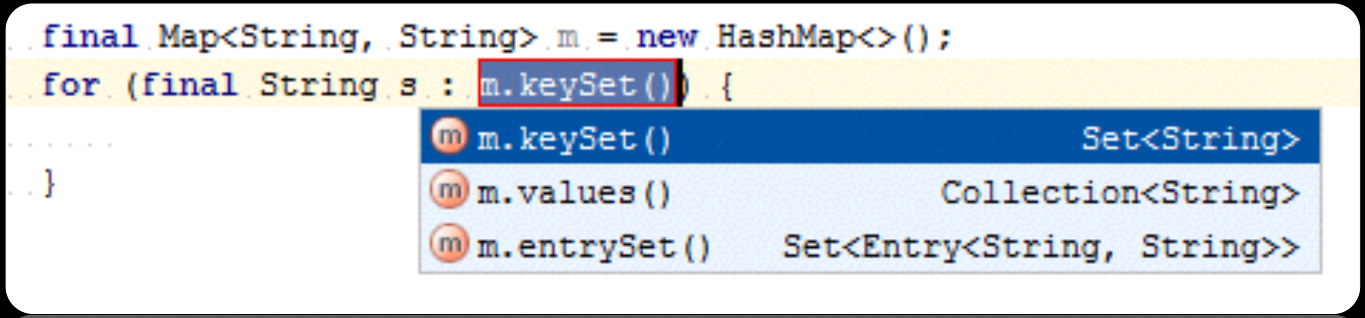
- Collection of key-value pairs
- Ordered & unordered
- May permit null key and null values

- HashMap<K,V>
- LinkedHashMap<K,V>
- SortedMap<K,V>
  - TreeMap<K,V>

Map<K, V> HashMap<K, V>

- Collection of key-value pairs
- Iteration order is not guaranteed
- Permits null key and null values
- Use immutable objects as keys otherwise behavior is unpredictable.
- Key values are hashed in put but not rehashed when modified.

- **Trick #4**: Use appropriate Map iteration.



```
final Map<String, String> m = new HashMap<>();
for (final String s : m.keySet()) {
}
```

m.keySet()	Set<String>
m.values()	Collection<String>
m.entrySet()	Set<Entry<String, String>>

`Map<K, V>`
`LinkedHashMap<K, V>`

- Collection of key-value pairs
- Ordered by insertion order
- Permits null key and null values
- Use immutable objects as keys otherwise behavior is unpredictable.
- Key values are hashed in put but not rehashed when modified.
- Many features suitable for LRU cache implementations.

- **Bonus Tip**: Use LinkedHashMap when consistent key order but not necessarily sorted order is required.
- **Bonus Tip:** If your project already uses Guava use CacheBuilder for full-featured caching functionality instead of building it yourself.

`Map<K, V>`

`SortedMap<K, V> → TreeMap<K, V>`

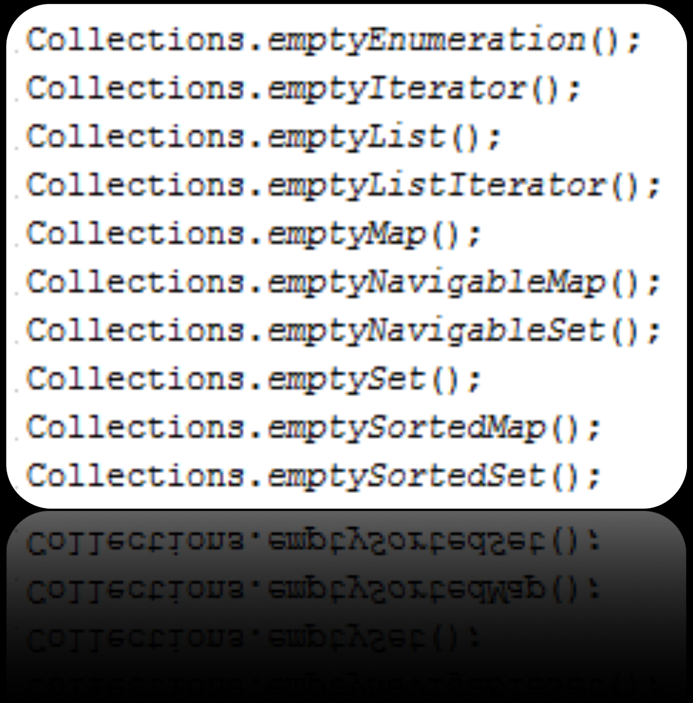
- Collection of key-value pairs
- Ordered natural or Comparator
- Permits null key and null values if Comparator is null-safe.
- Use immutable objects as keys otherwise behavior is unpredictable.

- **Trick #5**: Use .headMap() and .tailMap() where you also need a companion object.

Collections (class)

- Many helper methods for working with collections.
- Be aware of performance costs for different collection types.
- Static methods like .sort() and .binarySearch() are very useful.

- **Trick #6a**: Use .emptyXXX() methods when returning empty immutable collections instead of constructing your own.



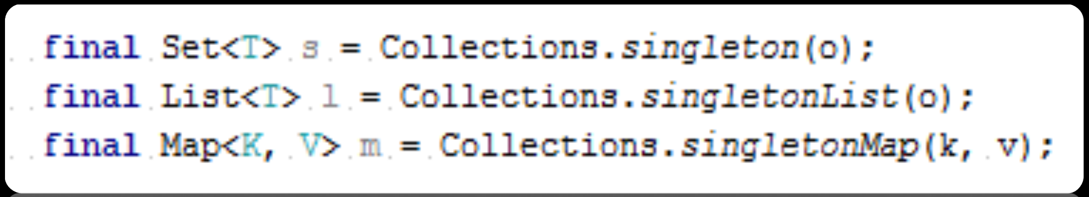
```
Collections.emptyEnumeration();
Collections.emptyIterator();
Collections.emptyList();
Collections.emptyListIterator();
Collections.emptyMap();
Collections.emptyNavigableMap();
Collections.emptyNavigableSet();
Collections.emptySet();
Collections.emptySortedMap();
Collections.emptySortedSet();
```

https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

Collections (class)

- Many helper methods for working with collections.
- Be aware of performance costs for different collection types.
- Static methods like .sort() and .binarySearch() are very useful.

- **Trick #6b**: Use .singletonXXX() methods when returning an immutable collection with only one element.



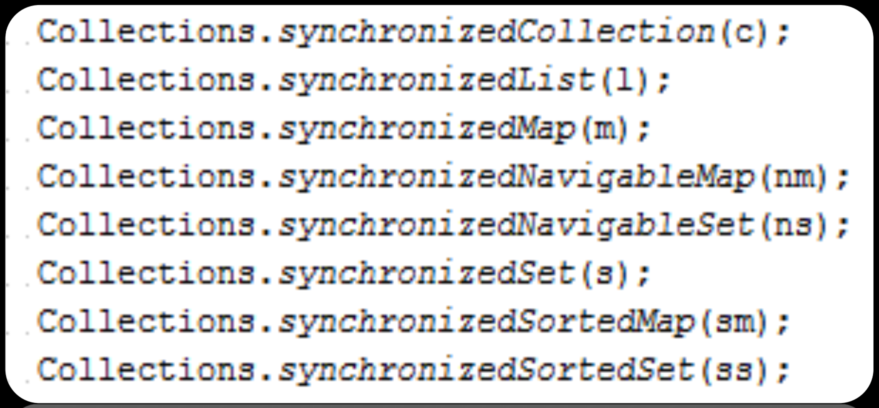
```
final Set<T> s = Collections.singleton(o);
final List<T> l = Collections.singletonList(o);
final Map<K, V> m = Collections.singletonMap(k, v);
```

https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

Collections (class)

- Many helper methods for working with collections.
- Be aware of performance costs for different collection types.
- Static methods like .sort() and .binarySearch() are very useful.

- **Trick #6c**: Use .synchronizedXXX() methods when returning a collection requiring concurrent access.



```
Collections.synchronizedCollection(c);
Collections.synchronizedList(l);
Collections.synchronizedMap(m);
Collections.synchronizedNavigableMap(nm);
Collections.synchronizedNavigableSet(ns);
Collections.synchronizedSet(s);
Collections.synchronizedSortedMap(sm);
Collections.synchronizedSortedSet(ss);
```

https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

Guava Collections2 (class)

- Collections2.filter()
  - Lists.filter()
  - Maps.filter()
  - Iterables.filter()
- Collections2.transform()
  - Lists.transform()
  - Maps.transform()
  - Iterables.transform()

- .filter() methods take Collection, List, Map, Iterable, etc. plus a Predicate.
- .transform() methods take Collection, List, Map, Iterable, etc. plus a Function object that takes an object and returns one
- **Trick #7:** Use Maps.uniqueIndex() to build a map.
- Beware of Guava lazy "views."

Collections in Java 8

Big API changes

- New Packages
  - java.util.function
  - java.util.stream
- Modified Packages & Classes
  - java.io
    - BufferedReader
  - java.lang
    - AutoCloseable, ThreadLocal, String, Iterable, CharSequence, Boolean, Integer, Long, Float, Double
  - java.nio.file.Files
  - java.util
    - Arrays, BitSet, Collection, Comparator, Iterator, List, Map, Map.Entry, LinkedHashMap, Random, TreeMap

- New Classes
  - java.lang.Objects
  - java.util
    - PrimitiveIterator
    - Spliterator
    - DoubleSummaryStatistics
    - IntSummaryStatistics
    - LongSummaryStatistics
    - Optional
    - OptionalDouble
    - OptionalInt
    - OptionalLong
    - Spliterators
    - SplittableRandom
    - StringJoiner

https://docs.oracle.com/javase/8/docs/technotes/guides/language/lambda_api_jdk8.html

Collections in Java 8

Impacts on Guava

- com.google.common.base.Objects
  - Deprecated methods & class
    - ~~firstNonNull()~~
    - ~~toStringHelper()~~
    - ~~ToStringHelper~~
  - Moved to com.google.common.base.MoreObjects.
  - Guava Function doesn't extend java.util.function.Function and Guava Predicate does not extend java.util.function.Predicate. However, your non-anonymous implementation can implement both interfaces.

https://docs.oracle.com/javase/8/docs/technotes/guides/language/lambda_api_jdk8.html

Thinking Functionally

- If I'm iterating over a collection there's likely a functional method I can call.
- Functions (including lambdas) are object instances. How should we test them?
- Stream operations are either intermediate or terminal operations.
  - Intermediate operations continue the pipeline and produce a result.
  - Terminal operations consume the stream and end the pipeline.
- java.util.stream.Collectors has many useful functional methods.

- **Trick #8:** Value Translators.
- **Trick #9:** Function Tables.

Collections Development Strategy

Can't move to Java 8 yet?

- Use Guava or Commons-Collections to begin adopting Functional style in use of collections.
- Use Guava static methods if diamond operator is not available (pre-7) e.g. Lists.newArrayList().
- Beware of mixing Predicate/Function/Transformer classes from different libraries — fortunately the compiler will tell you.

Already using Java 8

- Use Java Collections features everywhere possible for new development.
- Take advantage of Streams, Lambdas.
- Use Guava or Commons Collections if functionality doesn't exist in Java Collections.
- Only convert existing code to Java 8 if absolutely required.

For a different opinion see:
https://github.com/google/guava/wiki/FunctionalExplained

Best Trick

Learn the Java Collections API!

“Needs Deprecated?” API

- Avoid Vector
- Avoid Hashtable
- Avoid Stack

- Use Collections.synchronizedList() or java.util.concurrent.CopyOnWriteArrayList
- Use Collections.synchronizedMap()
- Use Deque<T>

Questions?

Resources

- @sswing
- Source: https://github.com/steveswing/ten-collections-tricks
- Java Collections: http://docs.oracle.com/javase/tutorial/collections/index.html
- Google Guava: https://github.com/google/guava/wiki
- Commons-collections: http://commons.apache.org/proper/commons-collections/