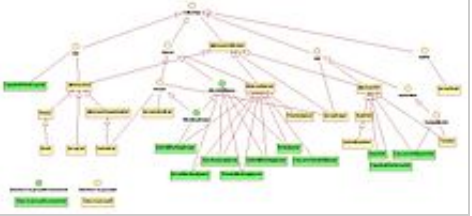
Java collections framework

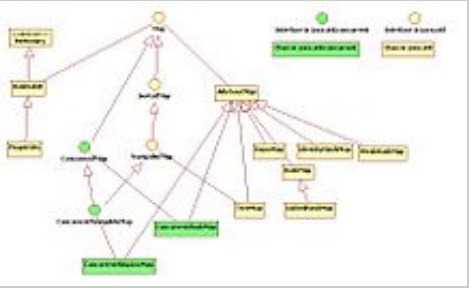
From Wikipedia, the free encyclopedia

The **Java collections framework** (JCF) is a set of classes and interfaces that implement commonly reusable collection data structures.[1]

Although it is a framework, it works in a manner of a **library**. The JCF provides both interfaces that define various collections and classes that implement them.



class- and interface hierarchy of java.util.Collection



class- and interface hierarchy of java.util.Map

Contents

History

Collection implementations in pre-JDK 1.2 versions of the Java platform included few data structure classes, but did not contain a collections framework.[2] The standard methods for grouping Java objects were via the array, the `Vector (http://download.oracle.com/javase/7/docs/api/java/util/Vector.html)` , and the `Hashtable (http://download.oracle.com/javase/7/docs/api/java/util/Hashtable.html)` classes, which unfortunately were not easy to extend, and did not implement a standard member interface.[3]

To address the need for reusable collection data structures, several independent frameworks were developed,[2] the most used being Doug Lea's *Collections package*,[4] and ObjectSpace *Generic Collection Library* (JGL),[5] whose main goal was consistency with the C++ Standard Template Library (STL).[6]

The collections framework was designed and developed primarily by Joshua Bloch, and was introduced in JDK 1.2. It reused many ideas and classes from Doug Lea's *Collections package*, which was deprecated as a result.[4] Sun choose not to use the ideas of JGL, because they wanted a compact framework, and consistency with C++ was not one of their goals.[7]

Doug Lea later developed a concurrency package, comprising new Collection-related classes.[8] An updated version of these concurrency utilities was included in JDK 5.0 as of JSR 166.

Architecture

Almost all collections in Java are derived from the **java.util.Collection (http://download.oracle.com/javase/7/docs/api/java/util/Collection.html)** interface. Collection defines the basic parts of all collections. The interface states the add() and remove() methods for adding to and removing from a collection respectively. Also required is the toArray() method, which converts the collection into a simple array of all the elements in the collection. Finally, the contains() method checks if a specified element is in the collection. The Collection interface is a subinterface of **java.util.Iterable (http://download.oracle.com/javase/7/docs/api/java/util/Iterable.html)** , so the iterator() method is also provided. All collections have an iterator that goes through all of the elements in the collection. Additionally, Collection is a generic. Any collection can be written to store any class. For example, Collection<String> can hold strings, and the elements from the collection can be used as strings without any casting required.[9]

There are three main types of collections:

- Lists: always ordered, may contain duplicates and can be handled the same way as usual arrays
- Sets: cannot contain duplicates and provide random access to their elements
- Maps: connect unique keys with values, provide random access to its keys and may host duplicate values

List Interface

Lists are implemented in the JCF via the **java.util.List (http://download.oracle.com/javase/7/docs/api/java/util/List.html)** interface. It defines a list as essentially a more flexible version of an array. Elements have a specific order, and duplicate elements are allowed. Elements can be placed in a specific position. They can also be searched for within the list. Two concrete classes implement List. The first is **java.util.ArrayList (http://download.oracle.com/javase/7/docs/api/java/util/ArrayList.html)** , which implements the list as an array. Whenever functions specific to a list are required, the class moves the elements around within the array in order to do it. The other implementation is **java.util.LinkedList (http://download.oracle.com/javase/7/docs/api/java/util/LinkedList.html)** . This class stores the elements in nodes that each have a pointer to the previous and next nodes in the list. The list can be traversed by following the pointers, and elements can be added or removed simply by changing the pointers around to place the node in its proper place.[10]

Queue Interfaces

The **java.util.Queue (http://download.oracle.com/javase/7/docs/api/java/util/Queue.html)** interface defines the queue data structure, which stores elements in the order in which they are inserted. New additions go to the end of the line, and elements are removed from the front. It creates a first-in first-out system. This interface is implemented by **java.util.LinkedList (http://download.oracle.com/javase/7/docs/api/java/util/LinkedList.html)** , **java.util.ArrayDeque (http://download.oracle.com/javase/7/docs/api/java/util/ArrayDeque.html)** , and **java.util.PriorityQueue (http://download.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html)** . LinkedList, of course, also implements the List interface and can also be used as one. But it also has the Queue methods. ArrayDeque implements the queue as an array. Both LinkedList and ArrayDeque also implement the **java.util.Deque (http://download.oracle.com/javase/7/docs/api/java/util/Deque.html)**

interface, giving it more flexibility.[11]

java.util.Queue (http://download.oracle.com/javase/7/docs/api/java/util/Queue.html) can be used more flexibly with its subinterface, **java.util.BlockingQueue (http://download.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html)** . The BlockingQueue interface works like a regular queue, but additions to and removals from the queue are blocking. If remove is called on an empty queue, it can be set to wait either a specified time or indefinitely for an item to appear in the queue. Similarly, adding an item is subject to an optional capacity restriction on the queue, and the method can wait for space to become available in the queue before returning.[12]

The **java.util.Queue (http://download.oracle.com/javase/7/docs/api/java/util/Queue.html)** interface is expanded by the **java.util.Deque (http://download.oracle.com/javase/7/docs/api/java/util/Deque.html)** subinterface. Deque creates a double-ended queue. While a regular queue only allows insertions at the rear and removals at the front, the deque allows insertions or removals to take place both at the front and the back. A deque is like a queue that can be used forwards or backwards, or both at once. Additionally, both a forwards and a backwards iterator can be generated. The Deque interface is implemented by **java.util.ArrayDeque (http://download.oracle.com/javase/7/docs/api/java/util/ArrayDeque.html)** and **java.util.LinkedList (http://download.oracle.com/javase/7/docs/api/java/util/LinkedList.html)** .[13]

The **java.util.concurrent.BlockingDeque (http://download.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingDeque.html)** interface works similarly to **java.util.concurrent.BlockingQueue (http://download.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html)** . The same methods for insertion and removal with time limits for waiting for the insertion or removal to become possible are provided. However, the interface also provides the flexibility of a deque. Insertions and removals can take place at both ends. The blocking function is combined with the deque function.[14]

PriorityQueue Class

java.util.PriorityQueue (http://download.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html) implements **java.util.Queue (http://download.oracle.com/javase/7/docs/api/java/util/Queue.html)** , but also alters it. Instead of elements being ordered by the order in which they are inserted, they are ordered by priority. The method used to determine priority is either the compareTo() method in the elements or a method given in the constructor. The class creates this by using a heap to keep the items sorted.[15]

Set Interfaces

Java's **java.util.Set (http://download.oracle.com/javase/7/docs/api/java/util/Set.html)** interface defines the set. A set can't have any duplicate elements in it. Additionally, the set has no set order. As such, elements can't be found by index. Set is implemented by **java.util.HashSet (http://download.oracle.com/javase/7/docs/api/java/util/HashSet.html)** , **java.util.LinkedHashSet (http://download.oracle.com/javase/7/docs/api/java/util/LinkedHashSet.html)** , and

java.util.TreeSet (http://download.oracle.com/javase/7/docs/api/java/util/TreeSet.html) . HashSet uses a hash table. More specifically, it uses a **java.util.HashMap (http://download.oracle.com/javase/7/docs/api/java/util/HashMap.html)** to store the hashes and elements and to prevent duplicates. **java.util.LinkedHashSet (http://download.oracle.com/javase/7/docs/api/java/util/LinkedHashSet.html)** extends this by creating a doubly linked list that links all of the elements by their insertion order. This ensures that the iteration order over the set is predictable. **java.util.TreeSet (http://download.oracle.com/javase/7/docs/api/java/util/TreeSet.html)** uses a red-black tree implemented by a **java.util.TreeMap (http://download.oracle.com/javase/7/docs/api/java/util/TreeMap.html)** . The red-black tree makes sure that there are no duplicates. Additionally, it allows TreeSet to implement **java.util.SortedSet (http://download.oracle.com/javase/7/docs/api/java/util/SortedSet.html)** .[16]

The **java.util.Set (http://download.oracle.com/javase/7/docs/api/java/util/Set.html)** interface is extended by the **java.util.SortedSet (http://download.oracle.com/javase/7/docs/api/java/util/SortedSet.html)** interface. Unlike a regular set, the elements in a sorted set are sorted, either by the element's compareTo() method, or a method provided to the constructor of the sorted set. The first and last elements of the sorted set can be retrieved, and subsets can be created via minimum and maximum values, as well as beginning or ending at the beginning or ending of the sorted set. The SortedSet interface is implemented by **java.util.TreeSet (http://download.oracle.com/javase/7/docs/api/java/util/TreeSet.html)** [17]

java.util.SortedSet (http://download.oracle.com/javase/7/docs/api/java/util/SortedSet.html) is extended further via the **java.util.NavigableSet (http://download.oracle.com/javase/7/docs/api/java/util/NavigableSet.html)** interface. It's similar to SortedSet, but there are a few additional methods. The floor(), ceiling(), lower(), and higher() methods find an element in the set that's close to the parameter. Additionally, a descending iterator over the items in the set is provided. As with SortedSet, **java.util.TreeSet (http://download.oracle.com/javase/7/docs/api/java/util/TreeSet.html)** implements NavigableSet.[18]

Map Interfaces

Maps are defined by the **java.util.Map (http://download.oracle.com/javase/7/docs/api/java/util/Map.html)** interface in Java. Maps are simple data structures that associate a key with a value. The element is the value. This lets the map be very flexible. If the key is the hash code of the element, the map is essentially a set. If it's just an increasing number, it becomes a list. Maps are implemented by **java.util.HashMap (http://download.oracle.com/javase/7/docs/api/java/util/HashMap.html)** , **java.util.LinkedHashMap (http://download.oracle.com/javase/7/docs/api/java/util/LinkedHashMap.html)** , and **java.util.TreeMap (http://download.oracle.com/javase/7/docs/api/java/util/TreeMap.html)** . HashMap uses a hash table. The hashes of the keys are used to find the values in various buckets. LinkedHashMap extends this

by creating a doubly linked list between the elements. This allows the elements to be accessed in the order in which they were inserted into the map. TreeMap, in contrast to HashMap and LinkedHashMap, uses a red-black tree. The keys are used as the values for the nodes in the tree, and the nodes point to the values in the map.[19]

The **java.util.Map (http://download.oracle.com/javase/7/docs/api/java/util/Map.html)** interface is extended by its subinterface, **java.util.SortedMap (http://download.oracle.com/javase/7/docs/api/java/util/SortedMap.html)** . This interface defines a map that's sorted by the keys provided. Using, once again, the compareTo() method or a method provided in the constructor to the sorted map, the key-value pairs are sorted by the keys. The first and last keys in the map can be called. Additionally, submaps can be created from minimum and maximum keys. SortedMap is implemented by **java.util.TreeMap (http://download.oracle.com/javase/7/docs/api/java/util/TreeMap.html)** .[20]

The **java.util.NavigableMap (http://download.oracle.com/javase/7/docs/api/java/util/NavigableMap.html)** interface extends **java.util.SortedMap (http://download.oracle.com/javase/7/docs/api/java/util/SortedMap.html)** in various ways. Methods can be called that find the key or map entry that's closest to the given key in either direction. The map can also be reversed, and an iterator in reverse order can be generated from it. It's implemented by **java.util.TreeMap (http://download.oracle.com/javase/7/docs/api/java/util/TreeMap.html)** .[21]

See also

- Container (data structure)
- Standard Template Library
- Java concurrency

References

1. ^ "Lesson: Introduction to Collections" (http://download.oracle.com/javase/tutorial/collections/intro/index.html) . Oracle Corporation. http://download.oracle.com/javase/tutorial/collections/intro/index.html. Retrieved 2010-12-22.
2. ^ ***a*** ***b*** "Java Collections Framework" (http://www.digilife.be/quickreferences/PT/Java%20Collections%20Framework.pdf) . IBM. http://www.digilife.be/quickreferences/PT/Java%20Collections%20Framework.pdf. Retrieved 2011-01-01.
3. ^ "Get started with the Java Collections Framework" (http://www.javaworld.com/jw-11-1998/jw-11-collections.html) . JavaWorld. 1998-01-11. http://www.javaworld.com/jw-11-1998/jw-11-collections.html. Retrieved 2011-01-01. "*Before Collections made its most welcome debut, the standard methods for grouping Java objects were via the array, the Vector, and the Hashtable. All three of these collections have different methods and syntax for accessing members: arrays use the square bracket ([]) symbols, Vector uses the elementAt method, and Hashtable uses* `get` *and* `put` *methods.*"
4. ^ ***a*** ***b*** Doug Lea. "Overview of the collections Package" (http://gee.cs.oswego.edu/dl/classes/collections/index.html) . http://gee.cs.oswego.edu/dl/classes/collections/index.html. Retrieved 2011-01-01. "*The Sun Java Development Kit JDK1.2 finally includes a standard set of collection classes. While there are some design and implementation differences, the JDK1.2 package contains most of the same basic abstractions, structure, and functionality as this package. For this reason, this collections package will NOT be further updated*"
5. ^ "Generic Collection Library for Java™" (http://www.stanford.edu/group/coursework/docsTech/jgl/) .

http://www.stanford.edu/group/coursework/docsTech/jgl/. Retrieved 2011-01-01.
6. ^ "Need a good set of abstract data structures? ObjectSpace's JGL packs a punch!" (http://www.javaworld.com/javaworld/jw-06-1997/jw-06-jgl.html) . JavaWorld. 1997-01-06. http://www.javaworld.com/javaworld/jw-06-1997/jw-06-jgl.html. Retrieved 2011-01-01. "*As with Java itself, the Java Generic Library borrows heavily from the C++ camp: It takes the best from C++'s STL, while leaving the C++ warts behind. Most C++ programmers today will know of their STL, but few are managing to exploit its potential.*"
7. ^ "The battle of the container frameworks: which should you use?" (http://www.javaworld.com/javaworld/jw-01-1999/jw-01-jglvscoll.html) . JavaWorld. 1999-01-01. http://www.javaworld.com/javaworld/jw-01-1999/jw-01-jglvscoll.html. Retrieved 2011-01-01. "*Comparing ObjectSpace Inc.'s JGL and Sun's Collections Framework turns out to be like comparing apples and kiwi fruits. At first sight, the two frameworks seem to be competing for the same developers, but after a closer inspection it is clear that the two cannot be compared fairly without acknowledging first that the two frameworks have different goals. If, like Sun's documentation states, Collections is going to homogenize Sun's own APIs (core API, extensions, etc.), then clearly Collections has to be great news, and a good thing, even to the most fanatic JGL addict. Provided Sun doesn't break its promise in this area, I'll be happy to invest my resources in adopting Collections in earnest.* "
8. ^ Doug Lea. "Overview of package util.concurrent Release 1.3.4" (http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html) . http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html. Retrieved 2011-01-01. "*Note: Upon release of J2SE 5.0, this package enters maintenance mode: Only essential corrections will be released. J2SE5 package java.util.concurrent includes improved, more efficient, standardized versions of the main components in this package.*"
9. ^ http://docs.oracle.com/javase/7/docs/api/java/lang/Iterable.html
10. ^ http://download.oracle.com/javase/6/docs/api/java/util/List.html
11. ^ http://download.oracle.com/javase/6/docs/api/java/util/Queue.html
12. ^ http://download.oracle.com/javase/6/docs/api/java/util/concurrent/BlockingQueue.html
13. ^ http://download.oracle.com/javase/6/docs/api/java/util/Deque.html
14. ^ http://download.oracle.com/javase/6/docs/api/java/util/concurrent/BlockingDeque.html
15. ^ http://download.oracle.com/javase/6/docs/api/java/util/PriorityQueue.html
16. ^ http://download.oracle.com/javase/6/docs/api/java/util/Set.html
17. ^ http://download.oracle.com/javase/6/docs/api/java/util/SortedSet.html
18. ^ http://download.oracle.com/javase/6/docs/api/java/util/NavigableSet.html
19. ^ http://download.oracle.com/javase/6/docs/api/java/util/Map.html
20. ^ http://download.oracle.com/javase/6/docs/api/java/util/SortedMap.html
21. ^ http://download.oracle.com/javase/6/docs/api/java/util/NavigableMap.html

External links

- Collections Lessons (http://javalessons.com/cgi-bin/fun/java-tutorials-main.cgi?sub=adv&ses=ao789)
- CollectionSpy (http://www.collectionspy.com) — A profiler for Java's Collections Framework.
- Generic Types (http://www.onjava.com/pub/a/onjava/excerpt/javaian5_chap04/)
- Java 6 Collection Tutorial (http://tutorials.jenkov.com/java-collections/index.html) — By Jakob Jenkov, Kadafi Kamphulusa
- Java Generics and Collections (http://www.onjava.com/pub/a/onjava/excerpt/javagenerics_chap05/)
- Taming Tiger: The Collections Framework (http://www-128.ibm.com/developerworks/java/library/j-tiger07195/)
- 'The Collections Framework' (http://java.sun.com/javase/6/docs/technotes/guides/collections/index.html) (Sun Java SE 6 documentation)
- 'The Java Tutorials - Collections' by Josh Bloch (http://java.sun.com/docs/books/tutorial/collections/)
- 'Which Java Collection to use?' (http://www.janeve.me/articles/which-java-collection-to-use) — by Janeve George

Retrieved from "http://en.wikipedia.org/w/index.php?title=Java_collections_framework&oldid=514339577"
Categories: Java programming language
