The Java Collections Framework
Excerpts from an article by Dan Becker

JavaWorld | Nov 1, 1998

JDK 1.2 introduces a new framework for collections of objects, called the Java Collections Framework (JCF). “Oh no,” you groan, “not another API, not another framework to learn!” But wait, before you turn away, hear me out: the Collections framework is worth your effort and will benefit your programming in many ways. Three big benefits come immediately to mind:

• It dramatically increases the readability of your collections by providing a standard set of interfaces to be used by many programmers in many applications.
• It makes your code more flexible by allowing you to pass and return interfaces instead of concrete classes, generalizing your code rather than locking it down.
• It offers many specific implementations of the interfaces, allowing you to choose the collection that is most fitting and offers the highest performance for your needs.

And that’s just for starters!

The Java Collections advantage: An overview
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. These differences have long led programmers down the path to inconsistency in implementing their own collections -- some emulate the Vector access methods and some emulate the Enumeration interface.

To further complicate matters, most of the Vector methods are marked as final; that is, you cannot extend the Vector class to implement a similar sort of collection. We could create a collection class that looked like a Vector and acted like a Vector, but it couldn't be passed to a method that takes a Vector as a parameter.

The Java Collections Framework offers a number of advantages over using no framework or using the old Vector and Hashtable:

• A usable set of collection interfaces
  By implementing one of the basic interfaces -- Collection, Set, List, or Map -- you ensure your class conforms to a common API and becomes more regular and easily understood. So, whether you are implementing an SQL database, a color swatch matcher, or a remote chat application, if you implement the Collection interface, the operations on your collection of objects are well-known to your users. The standard interfaces also simplify the passing and returning of collections to and from class methods and allow the methods to work on a wider variety of collections.

• A basic set of collection implementations
  The JCF includes HashSet and TreeSet, ArrayList and LinkedList, and HashMap and Map. Using an existing, common implementation makes your code shorter and quicker to download. Also, using existing Core Java code ensures that any improvements to the base code will also improve the performance of your code.

• Other useful enhancements
  Each collection now returns an Iterator, an improved type of Enumeration that allows element operations such as insertion and deletion. The Iterator is "fail-fast," which means you get an exception if the list you're iterating is changed by another user. Also, list-based collections such as Vector return a ListIterator that allow bi-directional iteration and
updating. Several collections (`TreeSet` and `TreeMap`) implicitly support ordering. Use these classes to maintain a sorted list with no effort. You can find the smallest and largest elements or perform a binary search to improve the performance of large lists. You can sort other collections by providing a collection-compare method (a `Comparator` object) or an object-compare method (the `Comparable` interface). Finally, a static class `Collections` provides unmodifiable (read-only) and synchronized versions of existing collections. The unmodifiable classes are helpful to prevent unwanted changes to a collection. The synchronized version of a collection is a necessity for multithreaded programs.

One advantage of the Java Collections Framework is a consistent and regular API. The API is codified in a basic set of interfaces, `Collection`, `Set`, `List`, or `Map`. The `Collection` interface contains basic collection operations such as adding, removing, and tests for membership (containment). Any implementation of a collection, whether it is one provided by the Java Collections Framework or one of your own creations, will support one of these interfaces. Because the Collections framework is regular and consistent, you will learn a large portion of the frameworks simply by learning these interfaces.