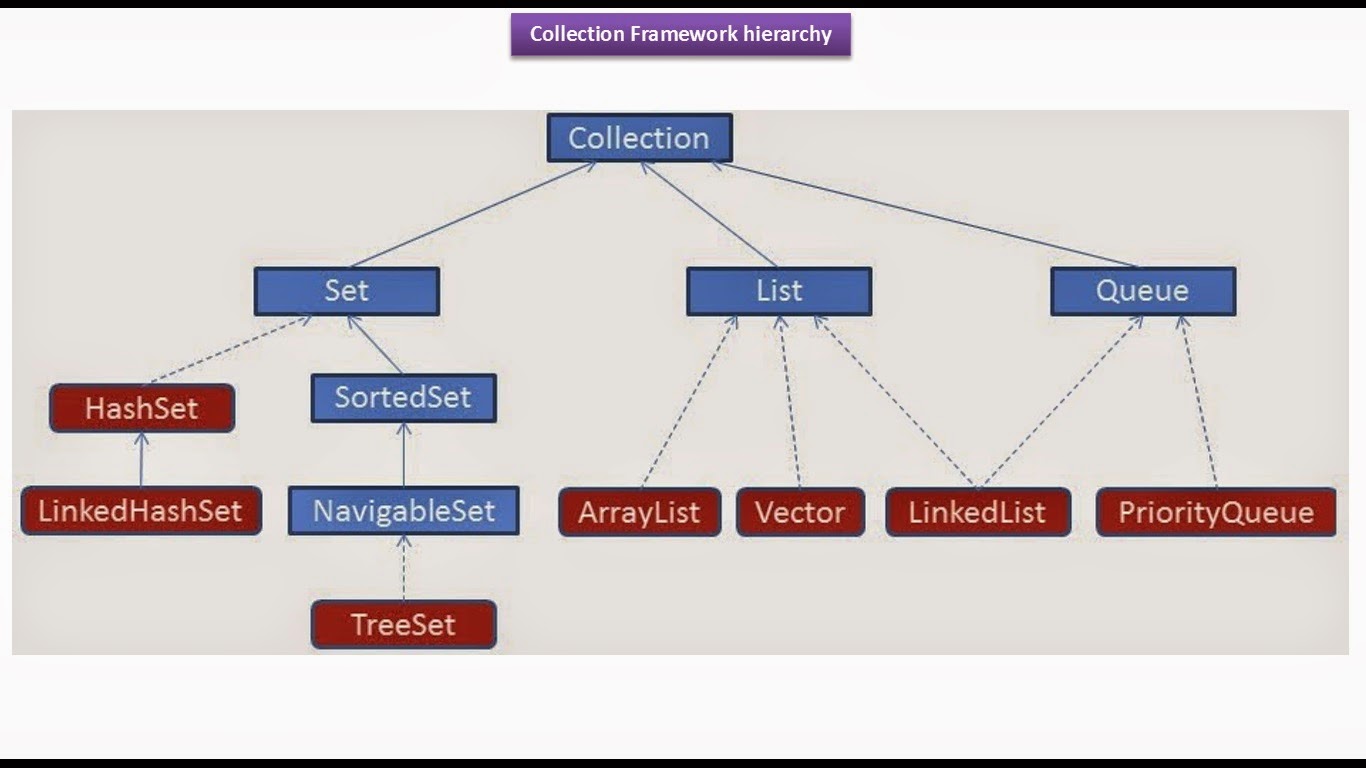
# Java Collections Framework

The [**Java Collections Framework**](http://docs.oracle.com/javase/tutorial/collections/intro/index.html) is a collection of interfaces and classes which helps in storing and processing the data efficiently. This framework has several useful classes which have tons of useful functions which makes a programmer task super easy. I have written several tutorials on Collections and below are the links of those. All the tutorials are shared with examples and source codes to help you understand better.

#### Collections Framework hierarchy



## List

A List is an ordered Collection (sometimes called a sequence). Lists may contain duplicate elements. Elements can be inserted or accessed by their position in the list, using a zero-based index.

* [**ArrayList**](http://beginnersbook.com/2014/08/arraylist-in-java/)
* [**LinkedList**](http://beginnersbook.com/2014/08/java-linkedlist-class/)
* [**Vector**](http://beginnersbook.com/2014/08/java-vector-class/)

## Set

A Set is a Collection that cannot contain duplicate elements. There are three main implementations of Set interface: HashSet, TreeSet, and LinkedHashSet. HashSet, which stores its elements in a hash table, is the best-performing implementation; however it makes no guarantees concerning the order of iteration. TreeSet, which stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashSet. LinkedHashSet, which is implemented as a hash table with a linked list running through it, orders its elements based on the order in which they were inserted into the set (insertion-order).

* [**HashSet**](http://beginnersbook.com/2013/12/hashset-class-in-java-with-example/)
* [**LinkedHashSet**](http://beginnersbook.com/2013/12/linkedhashset-class-in-java-with-example/)
* [**TreeSet**](http://beginnersbook.com/2013/12/treeset-class-in-java-with-example/)

## Map

A Map is an object that maps keys to values. A map cannot contain duplicate keys. There are three main implementations of Map interfaces: HashMap, TreeMap, and LinkedHashMap.  
HashMap: it makes no guarantees concerning the order of iteration  
TreeMap: It stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashMap.  
LinkedHashMap: It orders its elements based on the order in which they were inserted into the set (insertion-order).

* [**HashMap**](http://beginnersbook.com/2014/08/java-hashmap-class/)
* [**TreeMap**](http://beginnersbook.com/2013/12/treemap-in-java-with-example/)
* [**LinkedHashMap**](http://beginnersbook.com/2013/12/linkedhashmap-in-java/)

## Iterator/ListIterator

Both Iterator and ListIterator are used to iterate through elements of a collection class. Using Iterator we can traverse in one direction (forward) while using ListIterator we can traverse the collection class on both the directions(backward and forward). To know more differences between these two refer this article: [**Difference between Iterator and ListIterator**](http://beginnersbook.com/2014/06/difference-between-iterator-and-listiterator-in-java/).

* [**Iterator**](http://beginnersbook.com/2014/06/java-iterator-with-examples/)
* [**ListIterator**](http://beginnersbook.com/2014/06/listiterator-in-java-with-examples/)

## Benefits of the Java Collections Framework

The Java Collections Framework provides the following benefits:

* **Reduces programming effort:** By providing useful data structures and algorithms, the Collections Framework frees you to concentrate on the important parts of your program rather than on the low-level "plumbing" required to make it work. By facilitating interoperability among unrelated APIs, the Java Collections Framework frees you from writing adapter objects or conversion code to connect APIs.
* **Increases program speed and quality:** This Collections Framework provides high-performance, high-quality implementations of useful data structures and algorithms. The various implementations of each interface are interchangeable, so programs can be easily tuned by switching collection implementations. Because you're freed from the drudgery of writing your own data structures, you'll have more time to devote to improving programs' quality and performance.
* **Allows interoperability among unrelated APIs:** The collection interfaces are the vernacular by which APIs pass collections back and forth. If my network administration API furnishes a collection of node names and if your GUI toolkit expects a collection of column headings, our APIs will interoperate seamlessly, even though they were written independently.
* **Reduces effort to learn and to use new APIs:** Many APIs naturally take collections on input and furnish them as output. In the past, each such API had a small sub-API devoted to manipulating its collections. There was little consistency among these ad hoc collections sub-APIs, so you had to learn each one from scratch, and it was easy to make mistakes when using them. With the advent of standard collection interfaces, the problem went away.
* **Reduces effort to design new APIs:** This is the flip side of the previous advantage. Designers and implementers don't have to reinvent the wheel each time they create an API that relies on collections; instead, they can use standard collection interfaces.
* **Fosters software reuse:** New data structures that conform to the standard collection interfaces are by nature reusable. The same goes for new algorithms that operate on objects that implement these interfaces.