Collection Framework

A group of individual objects that are represented as a single unit is known as a Java Collection of Objects. In Java, a separate framework named the "Collection Framework" has been defined in JDK 1.2 which holds all the Java Collection Classes and Interface in it. In Java, the Collection interface (java.util.Collection) and Map interface (java.util.Map) are the two main "root" interfaces of Java collection classes.

A framework is a set of classes and interfaces which provide a ready-made architecture.

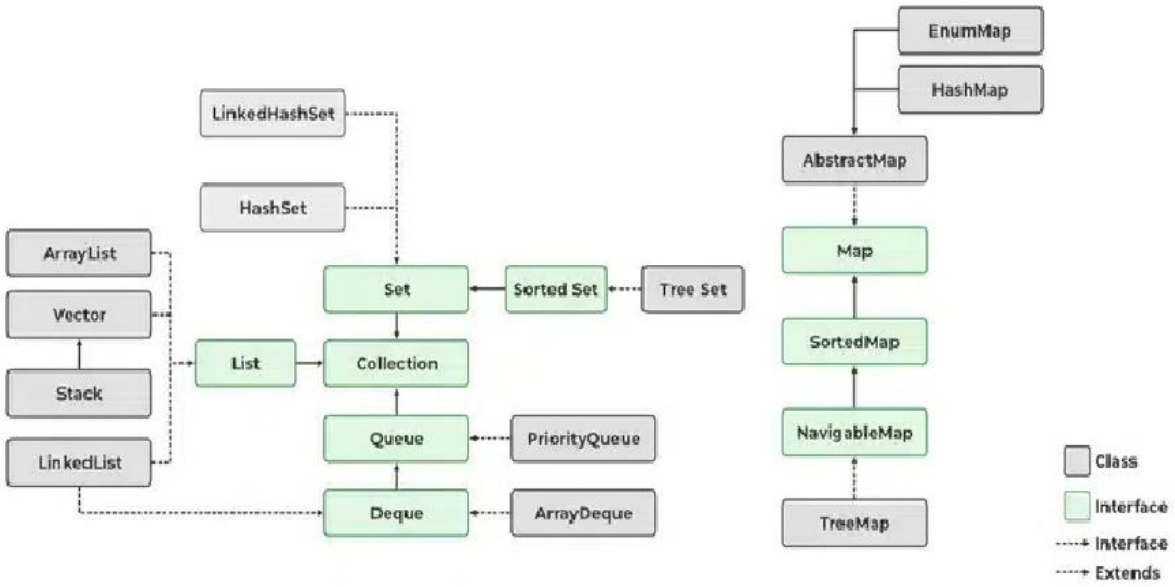
Before the Collection Framework (or before JDK 1.2) was introduced, the standard methods for grouping Java objects (or collections) were **Arrays** or **Vectors**, or **Hashtables**. All of these collections had no common interface. Therefore, though the main aim of all the collections is the same, the implementation of all these collections was defined independently and had no correlation among them.

Now we have same set of methods for all the varied collection classes.

The advantages of the collection framework.

1. **Consistent API:** The API has a basic set of interfaces like *Collection*, *Set*, *List*, or *Map*, all the classes (ArrayList, LinkedList, Vector, etc) that implement these interfaces have *some* common set of methods.
2. **Reduces programming effort:** A programmer doesn't have to worry about the design of the Collection but rather he can focus on its best use in his program. Therefore, the basic concept of Object-oriented programming (i.e.) abstraction has been successfully implemented.
3. **Increases program speed and quality:** Increases performance by providing high-performance implementations of useful data structures and algorithms because in this case, the programmer need not think of the best implementation of a specific data structure. He can simply use the best implementation to drastically boost the performance of his algorithm/program.

Hierarchy of Collection classes and Interfaces



Methods of Collection Interface:

Method	Description
add(Object)	This method is used to add an object to the collection.
addAll(Collection c)	This method adds all the elements in the given collection to this collection.
clear()	This method removes all of the elements from this collection.
contains(Object o)	This method returns true if the collection contains the specified element.
containsAll(Collection c)	This method returns true if the collection contains all of the elements in the given collection.
equals(Object o)	This method compares the specified object with this collection for equality.

Method	Description
hashCode()	This method is used to return the hash code value for this collection.
isEmpty()	This method returns true if this collection contains no elements.
iterator()	This method returns an iterator over the elements in this collection.
max()	This method is used to return the maximum value present in the collection.
parallelStream()	This method returns a parallel Stream with this collection as its source.
remove(Object o)	This method is used to remove the given object from the collection. If there are duplicate values, then this method removes the first occurrence of the object.
removeAll(Collection c)	This method is used to remove all the objects mentioned in the given collection from the collection.
removeIf(Predicate filter)	This method is used to remove all the elements of this collection that satisfy the given predicate.
retainAll(Collection c)	This method is used to retain only the elements in this collection that are contained in the specified collection.
size()	This method is used to return the number of elements in the collection.

Method	Description
spliterator()	This method is used to create a Spliterator over the elements in this collection.
stream()	This method is used to return a sequential Stream with this collection as its source.
toArray()	This method is used to return an array containing all of the elements in this collection.

Interfaces extending the Collection Interface:

1. **Iterable Interface**
   The main functionality of this interface is to provide an iterator for the collections. Therefore, this interface contains only one abstract method which is the iterator. It returns the
   ```
   Iterator iterator();
   ```

2. **Collection Interface**
   This interface contains all the basic methods which every collection has like adding the data into the collection, removing the data, clearing the data, etc.

3. **List Interface**

   This interface is dedicated to the data of the list type in which we can store all the ordered collections of the objects. This also allows duplicate data to be present in it. This list interface is implemented by various classes like ArrayList, Vector, Stack, etc. Since all the subclasses implement the list, we can instantiate a list object with any of these classes.

   For example:

   ```
   List <T> al = new ArrayList<> ();
   List <T> ll = new LinkedList<> ();
   List <T> v = new Vector<> ();
   Where T is the type of the object
   ```