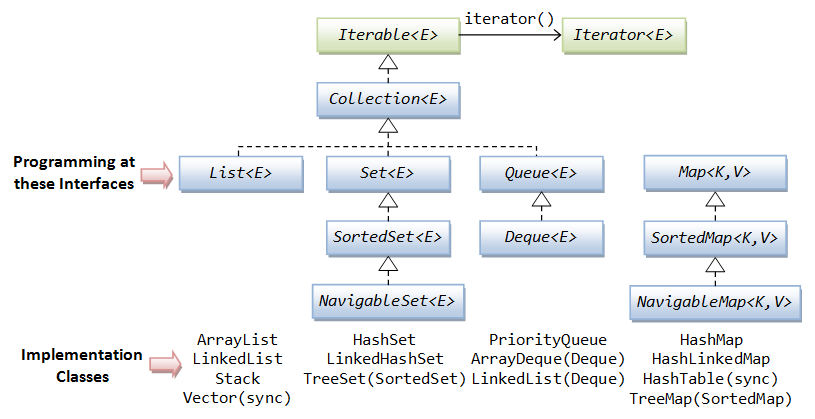
**Collections Framework**  
A Collection is a group of individual objects represented as a single unit. Java provides Collection Framework which defines several classes and interfaces to represent a group of objects as a single unit.

The Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are two main root interfaces of Java collection classes.

**Advantages of Collection Framework:**

1. Consistent API : The API has basic set of interfaces like Collection, Set, List, or Map. All those classes (such as ArrayList, LinkedList, Vector etc) which implements, these interfaces have some common set of methods.
2. Reduces programming effort: The programmer need not to worry about design of Collection rather than he can focus on its best use in his program.
3. Increases program speed and quality: Increases performance by providing high-performance implementations of useful data structures and algorithms.
4. **Hierarchy of Collection Framework**
5. Collection Map
6. / / \ \ |
7. / / \ \ |
8. Set List Queue Dequeue SortedMap
9. /
10. /
11. SortedSet
12. **Core Interfaces in Collections**
13. Note that this diagram shows only core interfaces.
14. **Collection :** Root interface with basic methods like add(), remove(),
15. contains(), isEmpty(), addAll(), ... etc.
17. **Set :** Doesn't allow duplicates. Example implementations of Set
18. interface are HashSet (Hashing based) and TreeSet (balanced
19. BST based). Note that TreeSet implements **SortedSet**.
20. **List :** Can contain duplicates and elements are ordered. Example
21. implementations are LinkedList (linked list based) and
22. [ArrayList](http://www.geeksforgeeks.org/array-vs-arraylist-in-java/) (dynamic array based)
23. **Queue :** Typically order elements in FIFO order except exceptions
24. like PriorityQueue.
25. **Deque :** Elements can be inserted and removed at both ends. Allows
26. both LIFO and FIFO.
27. **Map :** Contains Key value pairs. Doesn't allow duplicates. Example
28. implementation are [HashMap and TreeMap](http://www.geeksforgeeks.org/hashmap-treemap-java/).
29. TreeMap implements **SortedMap**.
30. The difference between Set and Map interface is, in Set we have only
31. keys, but in Map, we have key value pairs.



Iterators in Java

Iterators are used in [Collection framework](http://www.geeksforgeeks.org/collections-in-java-2/) in Java to retrieve elements one by one. There are three iterators.

**Iterator:**

It is a **universal** iterator as we can apply it to any Collection object. By using Iterator, we can perform both read and remove operations. It is improved version of Enumeration with additional functionality of remove-ability of a element.

Iterator must be used whenever we want to enumerate elements in all Collection framework implemented interfaces like Set, List, Queue, Deque and also in all implemented classes of Map interface. Iterator is the **only** cursor available for entire collection framework.

Iterator object can be created by calling *iterator()* method present in Collection interface.

// Here "c" is any Collection object. itr is of

// type Iterator interface and refers to "c"

Iterator itr = c.**iterator**();

Iterator interface defines **three** methods:

// Returns true if the iteration has more elements

**public boolean hasNext();**

// Returns the next element in the iteration

// It throws **NoSuchElementException** if no more

// element present

**public Object next();**

// Remove the next element in the iteration

// This method can be called only once per call

// to next()

**public void remove();**

*remove()* method can throw two exceptions

* *UnsupportedOperationException :*If the remove operation is not supported by this iterator
* *IllegalStateException :*If the next method has not yet been called, or the remove method has already been called after the last call to the next method

|  |
| --- |
| // Java program to demonstrate Iterator  import java.util.ArrayList;  import java.util.Iterator;    public class Test  {      public static void main(String[] args)      {          ArrayList al = new ArrayList();            for (int i = 0; i < 10; i++)              al.add(i);            System.out.println(al);            // at beginning itr(cursor) will point to          // index just before the first element in al          Iterator itr = al.iterator();            // checking the next element availabilty          while (itr.hasNext())          {              //  moving cursor to next element              int i = (Integer)itr.next();                // getting even elements one by one              System.out.print(i + " ");                // Removing odd elements              if (i % 2 != 0)                 itr.remove();          }          System.out.println();          System.out.println(al);      }  } |