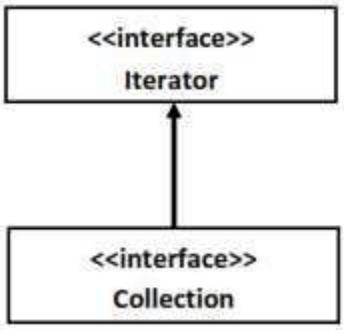
Collections Overview, The Collection Interfaces, The collection Classes. The Arrays Class.

Collections Overview

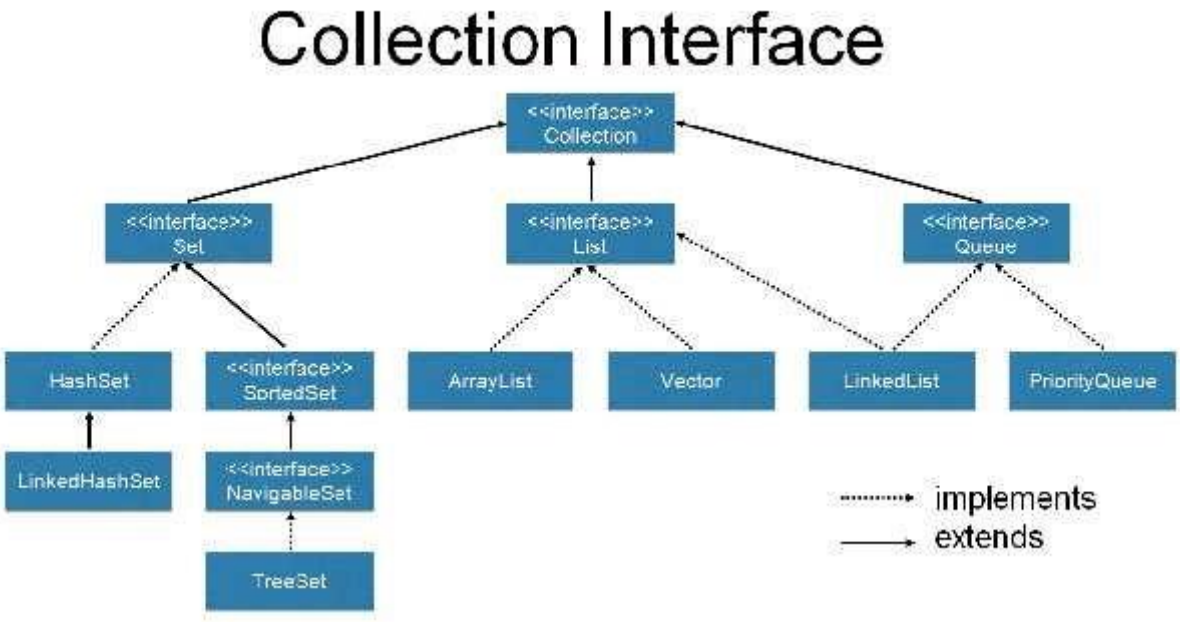
- A *collections framework* is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:
- **Interfaces:**
  - These are abstract data types that represent collections.
  - In object-oriented languages, interfaces generally form a hierarchy
- **Implementations:**
  - These are the concrete implementations of the collection interfaces.
- **Algorithms:**
  - These are the methods of implementation classes, such as searching and sorting, on objects that implement collection interfaces.

The Collection interfaces

- **java.util.Collection** is the root interface in the collections hierarchy
- *The **data structures derived from Collection interface are known as collections framework**.*
- The collection interfaces determine the fundamental nature of the collection classes.
- Collection is a generic interface with following declaration:
- **interface Collection <E>** ***// E specifies the type of objects that the collection will hold.***
- Collection extends the Iterable interface. This means that all collections can be cycled through by use of for-each style for loop.



- The collection framework defines several interfaces as shown below.



The collection interfaces packages in java.util

Interfaces	**Description**
Collections	Enables you to work with groups of objects, it is at top of the collection.
Dequeue	Extends Queue to handle a double-ended queue. This can be used to implement a stack or a queue.
List	Extends Collection to handle lists of objects.
NavigableSet	Extends SortedSet to handle the retrieval of an element based on how close its value is to another value.
Queue	Extends Collection to handle Queues
Set	Extends Collection to handle sets, which must contain unique elements
SortedSet	Extends Set to handle sorted sets.

The Collection class

- The collection interfaces are implemented by several classes. Some of the classes provide full implementation that can be used as-is. Others are abstract.
- The standard collection classes packaged in java.util

Class	Description
AbstractCollection	Implements parts of the collection interface
AbstractList	Extends **AbstractCollection** and implements parts of the **List** interface
AbstractQueue	Extends **AbstractCollection** and implements parts of the **Queue** interface
AbstractSequentialList	Extends **AbstractList** for use by a collection designed for sequential rather than random of its elements.
AbstractSet	Extends **AbstractCollection** and implements parts of the Set interface
ArrayList	Implements a dynamic array by extending **AbstractList**
ArrayDeque	Implements a dynamic double-ended queue by extending **AbstractCollection**. It also implements the **Deque** interface
EnumSet	Extends **AbstractSet** for use with enum elements
HashSet	Implements a set sorted in a hash table by extending **AbstractSet.**
LinkedHashSet	Extends **HashSet** to allow insertion-order iterations.
LinkedList	Implements a linked list by extending **AbstractSequentialList.** Also implements **Deque**.
PriorityQueue	Implements a priority-based queue by extending **AbstractQueue**.
TreeSet	Implements a set sorted in a tree by extending **AbstractSet** also implements **SortedSet**

The Arrays Class

- The **Arrays class** provides various methods that are useful when working with **arrays**.
  - These methods help bridge the gap between collections and arrays.
- Example
  - The ***sort( )*** method sorts an array so that it is arranged in ascending order.

```
int [ ] myArray = { 10, 58, 6, 20, 15 };
Arrays.sort (myArray);

for ( Object ob : myArray) {
     System.out.println(ob);
}
```

Output:
6
10
15
20
58