The collections framework was designed to meet several goals, such as −

* The framework had to be high-performance. The implementations for the fundamental collections (dynamic arrays, linked lists, trees, and hashtables) were to be highly efficient.
* The framework had to allow different types of collections to work in a similar manner and with a high degree of interoperability.
* The framework had to extend and/or adapt a collection easily.

Towards this end, the entire collections framework is designed around a set of standard interfaces. Several standard implementations such as **LinkedList, HashSet,** and **TreeSet**, of these interfaces are provided that you may use as-is and you may also implement your own collection, if you choose.

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. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
* **Implementations, i.e., Classes** − These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
* **Algorithms** − These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be polymorphic: that is, the same method can be used on many different implementations of the appropriate collection interface.

In addition to collections, the framework defines several map interfaces and classes. Maps store key/value pairs. Although maps are not *collections* in the proper use of the term, but they are fully integrated with collections.

The Collection Interfaces

The collections framework defines several interfaces. This section provides an overview of each interface −

|  |  |
| --- | --- |
| **Sr.No.** | **Interface & Description** |
| 1 | [**The Collection Interface**](https://www.tutorialspoint.com/java/java_collection_interface.htm)  This enables you to work with groups of objects; it is at the top of the collections hierarchy. |
| 2 | [**The List Interface**](https://www.tutorialspoint.com/java/java_list_interface.htm)  This extends **Collection** and an instance of List stores an ordered collection of elements. |
| 3 | [**The Set**](https://www.tutorialspoint.com/java/java_set_interface.htm)  This extends Collection to handle sets, which must contain unique elements. |
| 4 | [**The SortedSet**](https://www.tutorialspoint.com/java/java_sortedset_interface.htm)  This extends Set to handle sorted sets. |
| 5 | [**The Map**](https://www.tutorialspoint.com/java/java_map_interface.htm)  This maps unique keys to values. |
| 6 | [**The Map.Entry**](https://www.tutorialspoint.com/java/java_mapentry_interface.htm)  This describes an element (a key/value pair) in a map. This is an inner class of Map. |
| 7 | [**The SortedMap**](https://www.tutorialspoint.com/java/java_sortedmap_interface.htm)  This extends Map so that the keys are maintained in an ascending order. |
| 8 | [**The Enumeration**](https://www.tutorialspoint.com/java/java_enumeration_interface.htm)  This is legacy interface defines the methods by which you can enumerate (obtain one at a time) the elements in a collection of objects. This legacy interface has been superceded by Iterator. |

The Collection Classes

Java provides a set of standard collection classes that implement Collection interfaces. Some of the classes provide full implementations that can be used as-is and others are abstract class, providing skeletal implementations that are used as starting points for creating concrete collections.

The standard collection classes are summarized in the following table −

|  |  |
| --- | --- |
| **Sr.No.** | **Class & Description** |
| 1 | **AbstractCollection**  Implements most of the Collection interface. |
| 2 | **AbstractList**  Extends AbstractCollection and implements most of the List interface. |
| 3 | **AbstractSequentialList**  Extends AbstractList for use by a collection that uses sequential rather than random access of its elements. |
| 4 | [**LinkedList**](https://www.tutorialspoint.com/java/java_linkedlist_class.htm)  Implements a linked list by extending AbstractSequentialList. |
| 5 | [**ArrayList**](https://www.tutorialspoint.com/java/java_arraylist_class.htm)  Implements a dynamic array by extending AbstractList. |
| 6 | **AbstractSet**  Extends AbstractCollection and implements most of the Set interface. |
| 7 | [**HashSet**](https://www.tutorialspoint.com/java/java_hashset_class.htm)  Extends AbstractSet for use with a hash table. |
| 8 | [**LinkedHashSet**](https://www.tutorialspoint.com/java/java_linkedhashset_class.htm)  Extends HashSet to allow insertion-order iterations. |
| 9 | [**TreeSet**](https://www.tutorialspoint.com/java/java_treeset_class.htm)  Implements a set stored in a tree. Extends AbstractSet. |
| 10 | **AbstractMap**  Implements most of the Map interface. |
| 11 | [**HashMap**](https://www.tutorialspoint.com/java/java_hashmap_class.htm)  Extends AbstractMap to use a hash table. |
| 12 | [**TreeMap**](https://www.tutorialspoint.com/java/java_treemap_class.htm)  Extends AbstractMap to use a tree. |
| 13 | [**WeakHashMap**](https://www.tutorialspoint.com/java/java_weakhashmap_class.htm)  Extends AbstractMap to use a hash table with weak keys. |
| 14 | [**LinkedHashMap**](https://www.tutorialspoint.com/java/java_linkedhashmap_class.htm)  Extends HashMap to allow insertion-order iterations. |
| 15 | [**IdentityHashMap**](https://www.tutorialspoint.com/java/java_identityhashmap_class.htm)  Extends AbstractMap and uses reference equality when comparing documents. |

The *AbstractCollection, AbstractSet, AbstractList, AbstractSequentialList* and *AbstractMap* classes provide skeletal implementations of the core collection interfaces, to minimize the effort required to implement them.