Collection

A collection is a group of objects. In Java, these objects are called elements of the collection.

a collection is an object or container which stores a group of other objects as a single unit or single entity. Therefore, it is also known as container object or collection object in java.

A container object means it contains other objects. In simple words, a collection is a container that stores multiple elements together.

JVM(Java Virtual Machine) stores the reference of other objects into a collection object. It never stores physical copies of other objects because other objects are already available in the memory and storing another copy of objects into a collection object would be a wasting of memory.

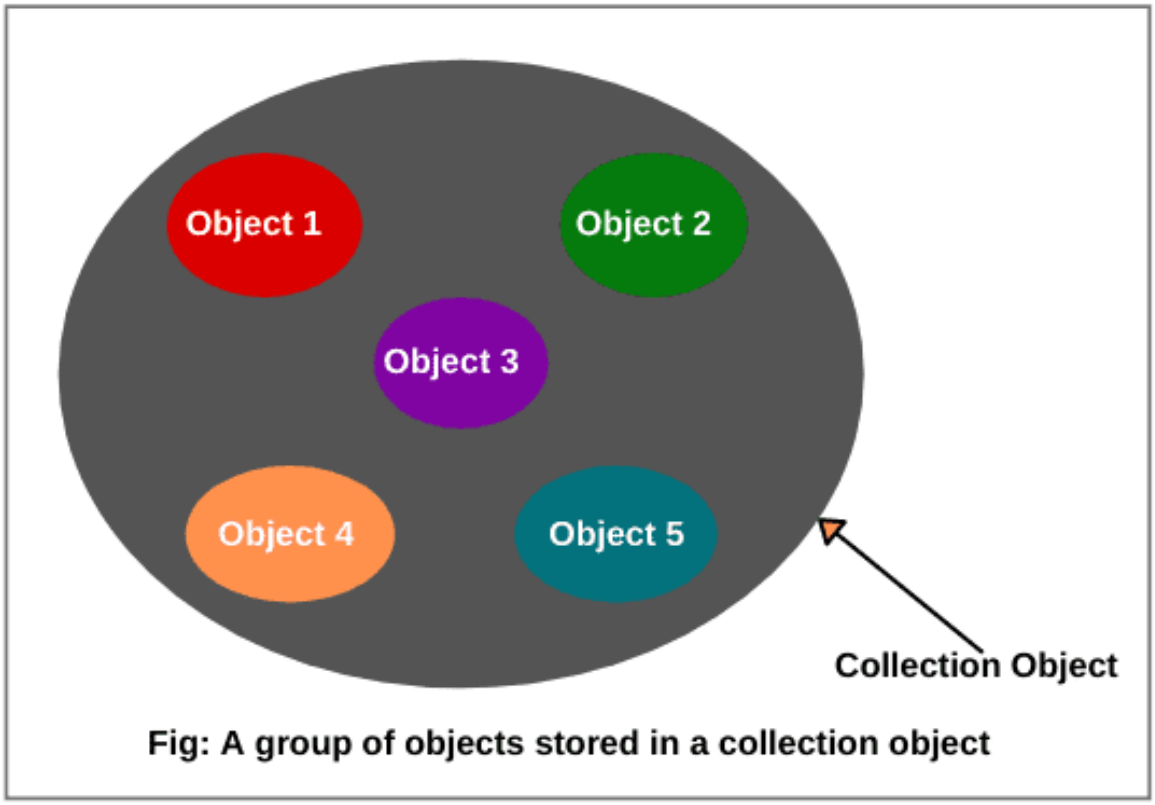


Fig: A group of objects stored in a collection object

Types of Objects Stored in Collection (Container) Object

1. Homogeneous objects:

Homo means same. Homogeneous objects are a group of multiple objects that belong to the same class.

2. Heterogeneous objects:

Hetero means different. Heterogeneous objects are a group of different objects that belong to different classes.

These objects can also be further divided into two types. They are as follows:

1. Duplicate objects:

The multiple objects of a class that contains the same data are called duplicate objects. For example, suppose we create two person objects Person p1 and Person p2. Both of these objects have the same data.

```
Person p1 = new Person( "abc");
Person p2 = new Person("abc");
```

Since the above two objects have the same data “abc” therefore, these are called duplicate objects.

2. Unique objects:

The multiple objects of a class that contains different data are called unique objects. For example:

```
Person p1 = new Person("abcd");
Person p2 = new Person("abcde");
```

A unique or duplicate object depends on its internal data.

What is Collections Framework in Java?

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

A Java collections framework is a sophisticated hierarchy of several predefined interfaces and implementation classes that can be used to handle a group of objects as a single entity.

In other words, a collections framework is a class library to handle groups of objects. It
is present in java.util package. It allows us to store, retrieve, and update a group of objects.

Collections framework in Java supports two types of containers:

- One for storing a collection of elements (objects), that is simply called a collection.
- The other, for storing key/value pairs, which is called a map.

Key Interfaces in Collections Framework

Java programming language built the collections framework around four core interfaces:

- Collection
- List
- Set
- Map

List of Interfaces defined in java.util package

Collection	List	Queue
Comparator	ListIterator	RandomAccess
Deque	Map	Set
Enumeration	Map.Entry	SortedMap
EventListener	NavigableMap	SortedSet
Formattable	NavigableSet	
Iterator	Observer	

Some of the most commonly used implementations of the Collections Framework in Java are as:

- ArrayList
- LinkedList
- HashSet
- HashMap

- TreeMap
- TreeSet

List of classes defined in java.util package

AbstractCollection	EventObject	Random
AbstractList	FormattableFlags	ResourceBundle
AbstractMap	Formatter	Scanner
AbstractQueue	AbstractSequentialList	HashMap
AbstractSet	HashSet	Stack
ArrayDeque	Hashtable	StringTokenizer
ArrayList	LinkedList	Vector
Collections	EnumMap	EnumSet
Calender	LinkedHashMap	TreeMap

Collection Hierarchy in Java

All the interfaces and classes for the collection framework are located in java.util package.

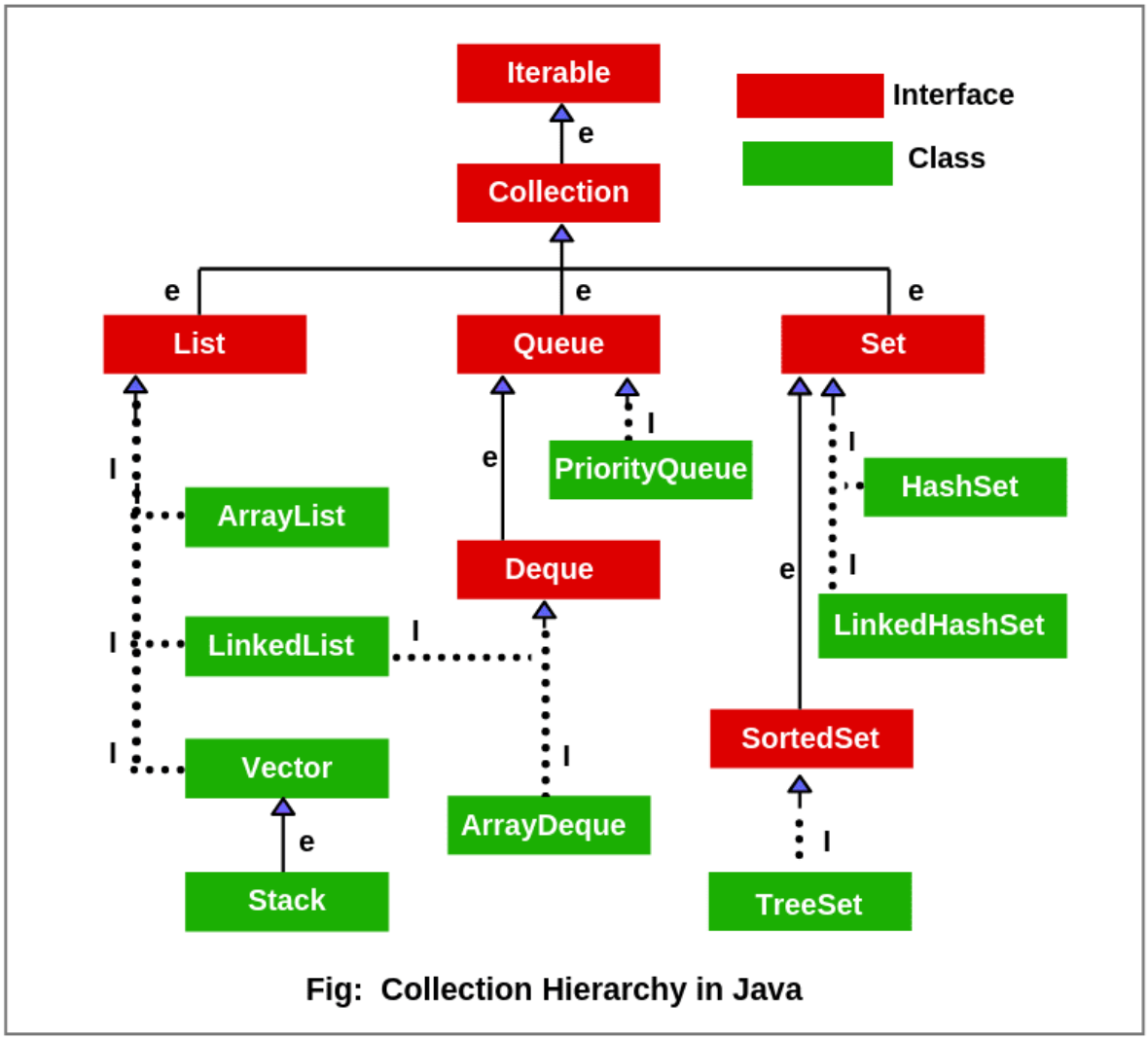


Fig: Collection Hierarchy in Java

e→ extends, I→ implements

Extends: Extends is a keyword that is used for developing inheritance between two classes and two interfaces.

Implements: Implements is a keyword used for developing inheritance between class and interface.

Collection Interface in Java

1. The basic interface of the collections framework is the Collection interface which is the root interface of all collections in the API (Application Programming Interface).

It is placed at the top of the collection hierarchy in java. It provides the basic operations for
adding and removing elements in the collection.

2. Collection interface extends the Iterable interface. The iterable interface has only one method called iterator(). The function of the iterator method is to return the iterator object. Using this iterator object, we can iterate over the elements of the collection.

3. List, Queue, and Set have three components which extends the Collection interface. A map is not inherited by Collection interface.

List Interface

1. This interface represents a collection of elements whose elements are arranged sequentially ordered.

2.List maintains an order of elements means the order is retained in which we add elements, and the same sequence we will get while retrieving elements.

3. We can insert elements into the list at any location. The list allows storing duplicate elements in Java.

4. ArrayList, vector, and LinkedList are three concrete subclasses that implement the list interface.

Set Interface

1.This interface represents a collection of elements that contains unique elements. i.e, It is used to store the collection of unique elements.

2.Set interface does not maintain any order while storing elements and while retrieving, we may not get the same order as we put elements. All the elements in a set can be in any order.

3. Set does not allow any duplicate elements.

4. HashSet, LinkedHashSet, TreeSet classes implements the set interface and sorted interface extends a set interface.

5. It can be iterated by using Iterator but cannot be iterated using ListIterator.

SortedSet Interface

1. This interface extends a set whose iterator transverse its elements according to their natural ordering.

2. TreeSet implements the sorted interface.

Queue Interface

1.A queue is an ordered of the homogeneous group of elements in which new elements are added at one end(rear) and elements are removed from the other end(front). Just like a queue in a supermarket or any shop.

2. This interface represents a special type of list whose elements are removed only from the head.

3.LinkedList, Priority queue, ArrayQueue, Priority Blocking Queue, and Linked Blocking Queue are the concrete subclasses that implement the queue interface.

Deque Interface

1. A deque (double-ended queue) is a sub-interface of queue interface. It is usually pronounced “deck”.

2. This interface was added to the collection framework in Java SE 6.

3. Deque interface extends the queue interface and uses its method to implement deque.

4. It is a linear collection of elements in which elements can be inserted and removed from either end. i.e, it supports insertion and removal at both ends of an object of a class that implements it.
5. LinkedList and ArrayDeque classes implement the Deque interface.

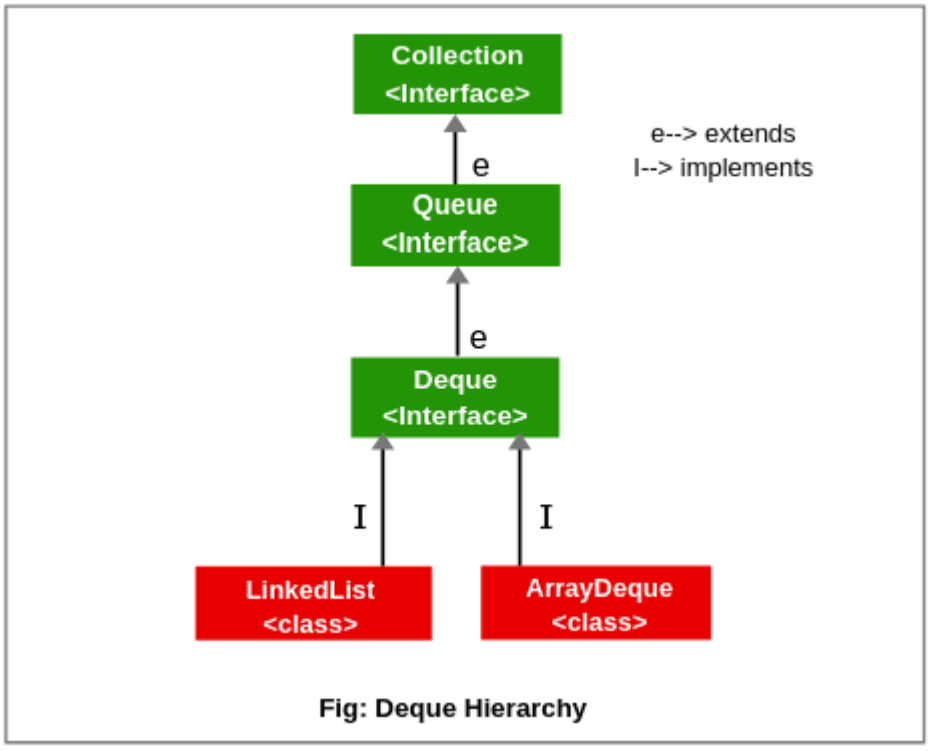
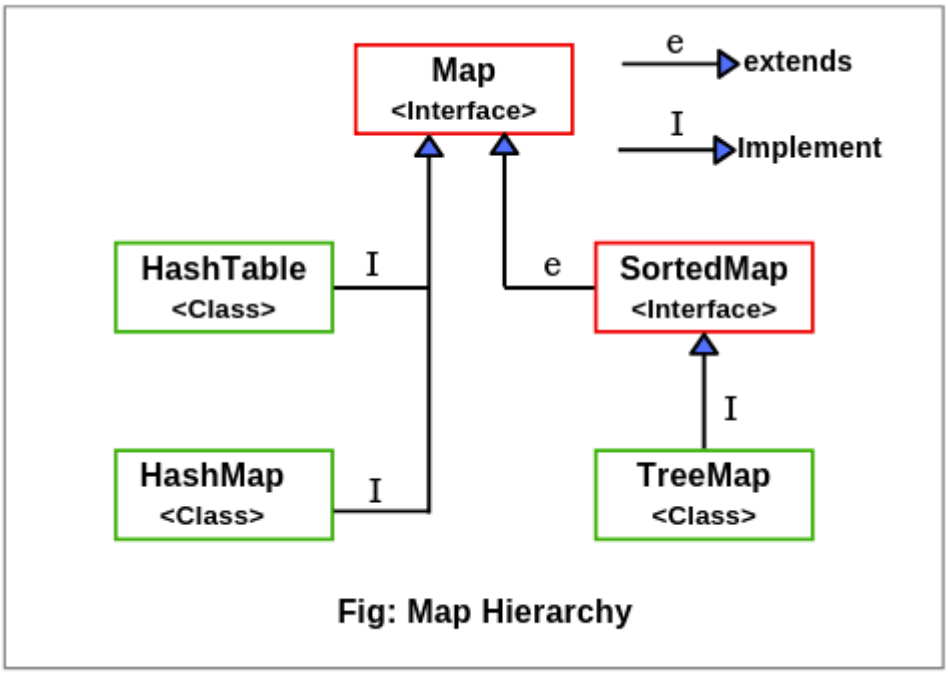


Fig: Deque Hierarchy

Map Interface

Map interface is not inherited by the collection interface. It represents an object that stores and retrieves elements in the form of a Key/Value pairs and their location within the Map are determined by a Key.

1. Map uses a hashing technique for storing key-value pairs. It doesn't allow to store the duplicate keys but duplicate values are allowed.
2. HashMap, HashTable, LinkedHashMap, TreeMap classes implements Map interface.



SortedMap Interface

This interface represents a Map whose elements are stored in their natural ordering. It extends the Map interface which in turn is implemented by TreeMap classes.

Methods of Collection Interface in Java

1. **add():** This method is used to add or insert an element in the collection.

2. **addAll():** This method adds a collection of elements to the collection. It returns true if the elements are added otherwise returns false.

3. **clear():** This method clears or removes all the elements from the collection.

4. **contains():** It checks that element is present or not in a collection.

5. **containsAll():** This method checks that specified a collection of elements are present or not. It returns true if the calling collection contains all specified elements otherwise return
false.

6. **equals():** It checks for equality with another object.

7. **hashCode():** It returns the hash code number for the collection. Its return type is an integer.

8. **isEmpty():** It returns true if a collection is empty.

9. **iterator():** It returns an iterator.

10. **remove():** It removes a specified element from the collection.

11. **removeAll():** The removeAll() method removes all elements from the collection.

12. **retainAll():** This method is used to remove all elements from the collection except the specified collection. It returns true if all the elements are removed otherwise returns false.

13. **size():** The size() method returns the total number of elements in the collection.

14. **toArray():** It returns the elements of a collection in the form of an array.

15. **Object[ ] toArray():** Returns an array that contains all the elements stored in the invoking collection.

Collections Class in Java

The collections classes implement the collection interfaces. They are defined in java.util package.

1. **AbstractCollection:** It implements most of the collection interface. It is a superclass for all of the concrete collection classes.

2. **AbstractList:** It extends AbstractCollection and implements most of the List interface.

3. **AbstractQueue:** It extends AbstractCollection and implements the queue interface.

4. **AbstractSequentialList:** It extends AbstractList and uses sequential order to access elements.

5. **AbstractSet:** Extends AbstractCollection and implements most of the set interface.

---

6. **ArrayList:** It implements a dynamic array by extending AbstractList.

7. **EnumSet:** Extends AbstractSet for use with enum elements.

8. **HashSet:** Extends AbstractSet for use with a hash table.

---

9. **LinkedHashSet:** Extends HashSet to allow insertion-order iterations.

10. **LinkedList:** Implements a linked list by extending AbstractSequentialList.

11. **PriorityQueue:** Extends AbstractQueue to support a priority-based queue.

12. **TreeSet:** Extends AbstractSet and implements the SortedSet interface.