Collection Framework

We are going for collections to overcome the drawbacks of Arrays.

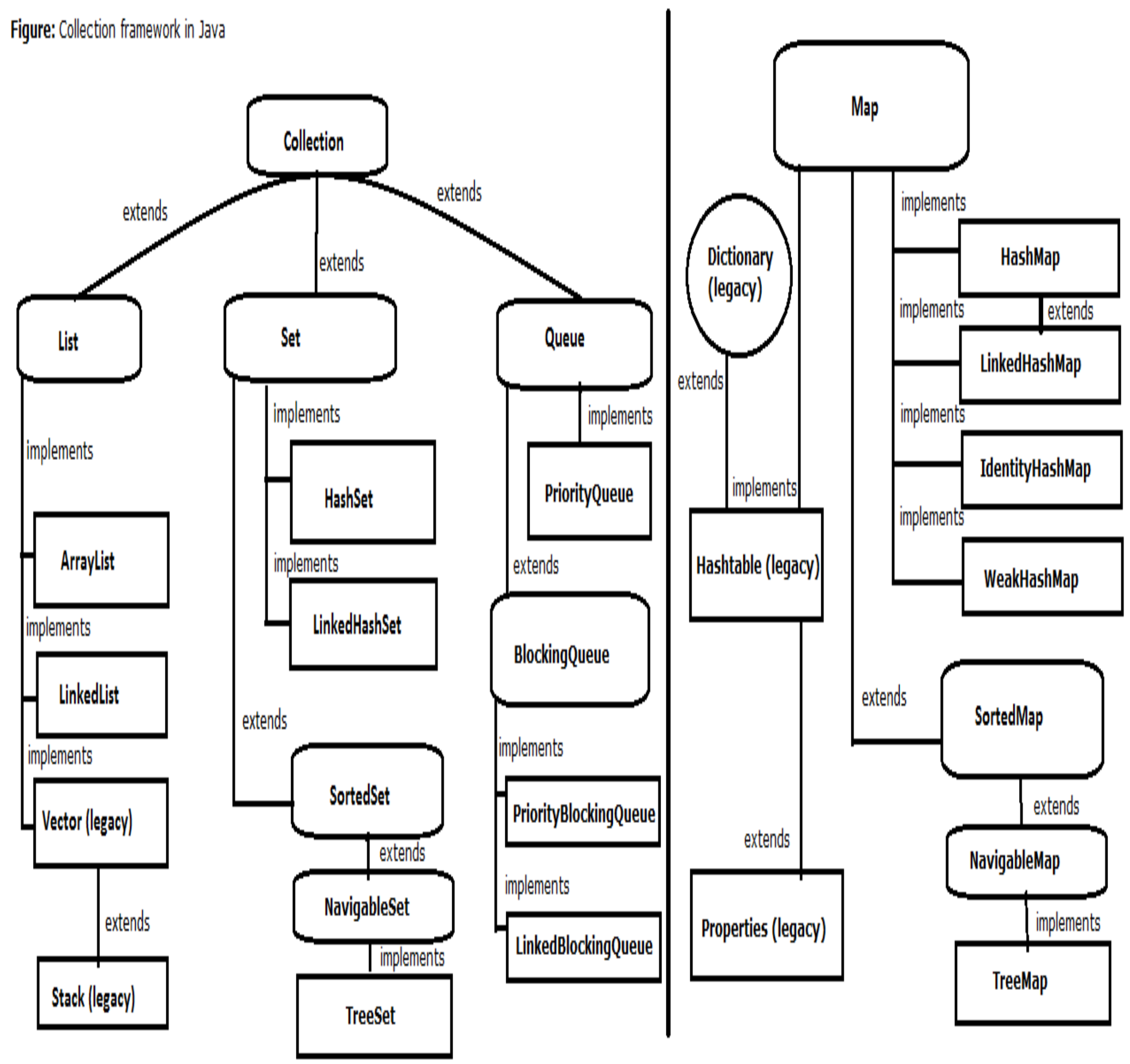
Differences between Arrays and Collections

Arrays	**Collections**
Arrays are fixed in size.	Collections are growable in nature.
Memory point of view arrays are not recommended to use.	Memory point of view collections are highly recommended to use.
Performance point of view arrays are recommended to use.	Performance point of view collections are not recommended to use.
Arrays can hold only homogeneous data type elements	Collections can hold both homogeneous and heterogeneous elements.
5) There is no underlying data structure for arrays and hence there is no ready made method support.	Every collection class is implemented based on some standard data structure and hence ready made method support is available.
Arrays can hold both primitives and object types.	Collections can hold only objects but not primitives.

Definition :

- When you want to store a group of objects in a single entity then we will go for collection.
- Collection is a group of classes and interfaces.
- Collection interface defines the most common methods which can be applicable for any collection object.There is no concrete class which implements Collection interface directly.

Collection Framework Hierarchy :



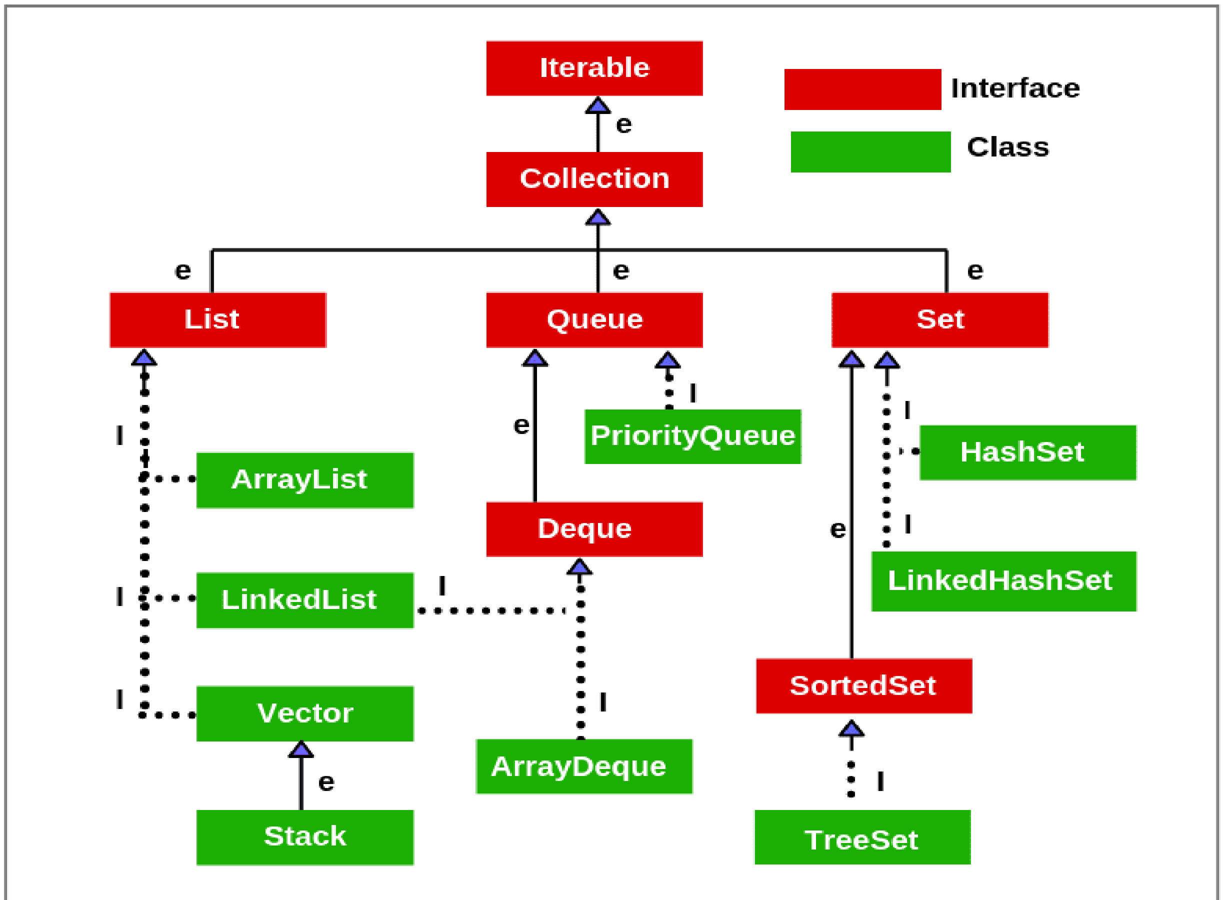


Fig: Collection Hierarchy in Java

List :

- It is the child interface of Collection**.**
- If we want to represent a group of individual objects as a single entity where **"duplicates are allow and insertion order must be preserved"** then we should go for List interface.

List interface defines the following specific methods :

1. boolean add(int index,Object o);
2. boolean addAll(int index,Collection c);
3. Object get(int index);
4. Object remove(int index);
5. Object set(int index,Object new);
6. Int indexOf(Object o);
7. Int lastIndexOf(Object o);
8. ListIterator listIterator()

ArrayList :

- The underlying data structure is a resizable array (or) growable array.
- Default Capacity of arraylist is 10.
- Duplicate objects are allowed.
- Insertion order preserved.
- Heterogeneous objects are allowed.
- Null insertion is possible
- Implements Serializable, Cloneable interfaces and RandomAccess
- If our frequent operation is retrieval then we will go for arraylist.

LinkedList :

- The underlying data structure is double LinkedList.
- If our frequent operation is insertion (or) deletion in the middle then LinkedList is the best choice.
- Duplicate objects are allowed.
- Insertion order is preserved.
- Heterogeneous objects are allowed.
- Null insertion is possible.
- Implements Serializable and Cloneable interfaces but not RandomAccess.

Methods present in linked list are :

- void addFirst(Object o);
- void addLast(Object o);
- Object getFirst();
- Object getLast();
- Object removeFirst();
- Object removeLast();

Vector:

- The underlying data structure is a resizable array (or) growable array.
- Duplicate objects are allowed.
- Insertion order is preserved.
- Heterogeneous objects are allowed.
- Null insertion is possible.
- Implements Serializable, Cloneable and RandomAccess interfaces.

Methods Present in vector are :

- addElement(Object o);
- removeElement(Object o);
- removeElementAt(int index);

- removeAllElements();
- Object elementAt(int index);
- Object firstElement();
- Object lastElement();

Stack :

- It is the child class of Vector.
- Whenever the last in first out(LIFO) order is required then we should go for Stack.

Set :

- It is the child interface of Collection.
- If we want to represent a group of individual objects as a single entity **"where duplicates are not allowed and insertion order is not preserved"** then we should go for Set interface.

HashSet:

- The underlying data structure is Hashtable.
- Insertion order is not preserved and it is based on the hash code of the objects.
- Duplicate objects are not allowed.
- If we are trying to insert duplicate objects we won't get compile time error and runtime error add() method simply returns false.
- Heterogeneous objects are allowed.
- Null insertion is possible.(only once)
- Implements Serializable and Cloneable interfaces but not RandomAccess.
- HashSet is best suitable, if our frequent operation is "Search".
- Introduced in 1.2v

LinkedHashSet :

- It is the child class of Set**.**
- The underlying data structure is a combination of LinkedList and Hashtable.
- Insertion order is preserved.
- Introduced in 1.4v.

Sorted Set :

- It is the child interface of Set.
- If we want to represent a group of "unique objects" according to some sorting order then we should go for SortedSet.

NavigableSet :

- It is the child interface of SortedSet.
- It provides several methods for navigation purposes.

TreeSet:

- The underlying data structure is a balanced tree.
- Duplicate objects are not allowed.
- Insertion order is not preserved and it is based on some sorting order of objects.
- Heterogeneous objects are not allowed if we are trying to insert heterogeneous objects then we will get ClassCastException.
- Null insertion is possible(only once)

Queue :

- It is the child interface of Collection.
- If we want to represent a group of individual objects prior to processing then we should go for queue concept

The 3 cursors of java :

- If we want to get objects one by one from the collection then we should go for a cursor. There are 3 types of cursors available in java. They are:
  1. Enumeration
  2. Iterator
  3. ListIterator

Iterator:

- We can use Iterator to get objects one by one from any collection object.
- We can apply the Iterator concept for any collection object and it is a universal cursor.
- While iterating the objects by Iterator we can perform both read and remove operations
- We can get an Iterator object by using the iterator() method of the Collection interface.

Comparable interface:

- Comparable interface present in java.lang package and contains only one method compareTo() method.

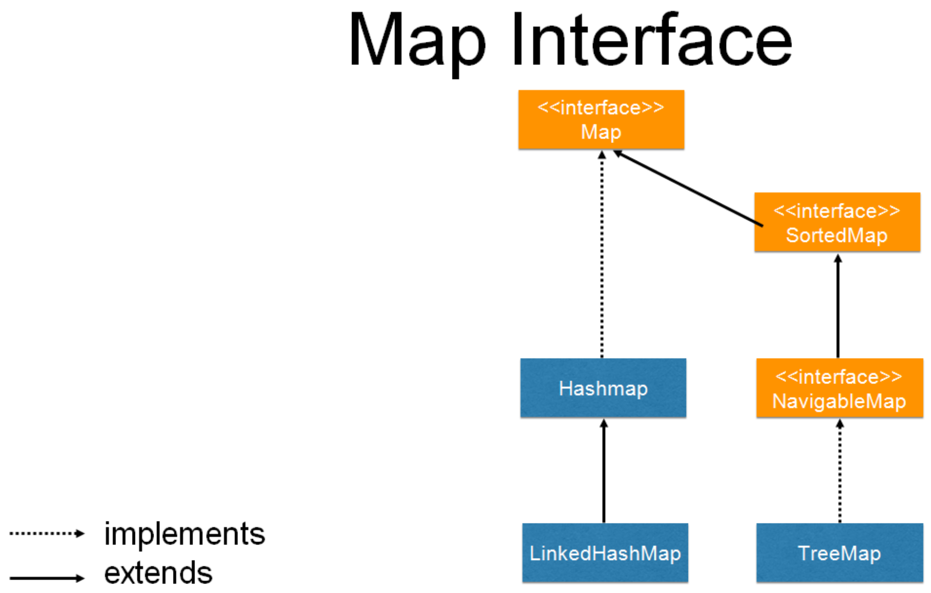
  public int compareTo(Object obj);

Comparator interface:

- Comparator interface present in java.util package defines the following 2 methods.
  1. public int compare(Object obj1,Object Obj2);
  2. public boolean equals(Object obj);

Comparable	**Comparator**
Comparable meant for default natural sorting order.	Comparator meant for customized sorting order
Present in java.lang package.	Present in java.util package.
Contains only one method. compareTo() method	Contains 2 methods. Compare() method. Equals() method.
String class and all wrapper Classes implements Comparable interface.	The only implemented classes of Comparator are Collator and RuleBasedCollator.

Map :



- If we want to represent a group of objects as "key-value" pair then we should go for Map interface.
- Both key and value are objects only.
- Duplicate keys are not allowed but values can be duplicated.
- Each key-value pair is called "one entry".

HashMap:

- The underlying data structure is Hashtable.
- Duplicate keys are not allowed but values can be duplicated.
- Insertion order is not preserved and it is based on hash code of the keys.
- Heterogeneous objects are allowed for both key and value.

- Null is allowed for keys(only once) and for values(any number of times).
- It is best suitable for Search operations

LinkedHashMap:

- The underlying data structure is a combination of Hashtable + LinkedList.
- Insertion order is preserved
- Introduced in 1.4v.

TreeMap:

- The underlying data structure is RED-BLACK Tree.
- Duplicate keys are not allowed but values can be duplicated.
- Insertion order is not preserved and all entries will be inserted according to some sorting order of keys.
- For the empty TreeMap as first entry the null key is allowed but after inserting that entry if we are trying to insert any other entry we will get NullPointerException.
- There are no restrictions for null values.