Collections in Java

Why Collection use over Array

### Main differences between Array and collection?

Array and Collection are somewhat similar regarding storing the references of objects and manipulating the data, but they differ in many ways. The main differences between the array and Collection are defined below:

* Arrays are always of fixed size, i.e., a user can not increase or decrease the length of the array according to their requirement or at runtime, but In Collection, size can be changed dynamically as per need.
* Arrays can only store homogeneous or similar type objects, but in Collection, heterogeneous objects can be stored.
* Arrays cannot provide the ?ready-made? methods for user requirements as sorting, searching, etc. but Collection includes readymade methods to use.

Collection

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Collection Framework is a combination of classes and interface, which is used to store and manipulate the data in the form of objects.

It provides various classes such as **ArrayList, LinkedList, Vector, Stack, and HashSet, LinkedHashSet,TreeSet, and PriorityQueue, ArrayDeque etc**. and interfaces such as **Itrable,Collection,List, Queue, Set, Deque ,SortedSet** etc. for this purpose.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many **interfaces (Set, List, Queue, Deque)** and **classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).**

#### What is Collection in Java

A Collection represents a single unit of objects, i.e., a group.

#### What is a framework in Java

* It provides readymade architecture.
* It represents a set of classes and interfaces.
* It is optional.

#### What is Collection framework

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes

### Difference between Collection and Collections?

The differences between the Collection and Collections are given below.

* The Collection is an interface whereas Collections is a class.
* The Collection interface provides the standard functionality of data structure to List, Set, and Queue. However, *Collections class is to sort and synchronize the collection elements.*
* The Collection interface provides the methods that can be used for data structure whereas *Collections class provides the static methods which can be used for various operation on a collection*.

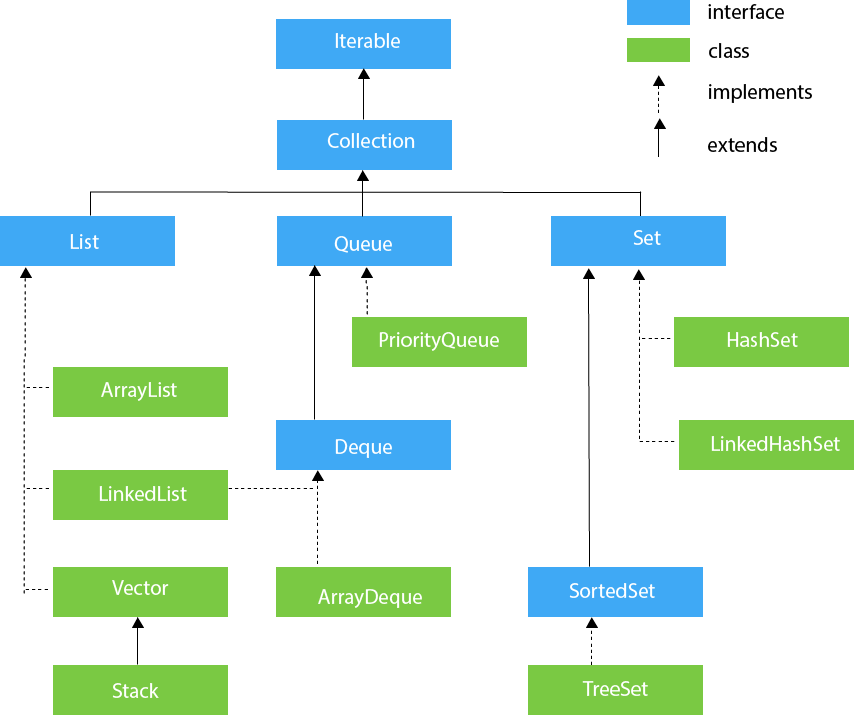
### Advantage of the generic collection?

There are three main advantages of using the generic collection.

* If we use the generic class, we don't need typecasting.
* It is type-safe and checked at compile time.
* Generic confirms the stability of the code by making it bug detectable at compile time.

### Hierarchy of Collection Framework

Let us see the hierarchy of Collection framework. The **java.util** package contains all the classes and interfaces for the Collection framework.



## Iterable Interface

The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e.,

1. **Iterator<T> iterator()**

It returns the iterator over the elements of type T.

### Iterator interface

|  |
| --- |
| Iterator interface provides the facility of iterating the elements in a forward direction only. |

#### Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean hasNext() | It returns true if the iterator has more elements otherwise it returns false. |
| 2 | public Object next() | It returns the element and moves the cursor pointer to the next element. |
| 3 | public void remove() | It removes the last elements returned by the iterator. It is less used. |

## Collection Interface

The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.

Some of the methods of Collection interface are Boolean add ( Object obj), Boolean addAll ( Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.

## List Interface

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

1. List <data-type> list1= **new** ArrayList();
2. List <data-type> list2 = **new** LinkedList();
3. List <data-type> list3 = **new** Vector();
4. List <data-type> list4 = **new** Stack();

There are various methods in List interface that can be used to insert, delete, and access the elements from the list.

The classes that implement the List interface are given below.

## ArrayList

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed. Consider the following example.

Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

The important points about Java ArrayList class are:

* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non synchronized.
* Java ArrayList allows random access because array works at the index basis.
* In Java ArrayList class, manipulation is slow because a lot of shifting needs to occur if any element is removed from the array list.

**public** **class** ArrayList<E> **extends** AbstractList<E> **implements** List<E>, RandomAccess, Cloneable, Serializable

### Java Non-generic Vs. Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

Java new generic collection allows you to have only one type of object in a collection. Now it is type safe so typecasting is not required at runtime.

Let's see the old non-generic example of creating java collection.

1. ArrayList al=**new** ArrayList();//creating old non-generic arraylist

Let's see the new generic example of creating java collection.

1. ArrayList<String> al=**new** ArrayList<String>();//creating new generic arraylist

In a generic collection, we specify the type in angular braces. Now ArrayList is forced to have the only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

### Ways to iterate the elements of the collection in java

There are various ways to traverse the collection elements:

1. By Iterator interface.
2. By for-each loop.
3. By ListIterator interface.
4. By for loop.
5. By forEach() method.
6. By forEachRemaining() method.

## LinkedList

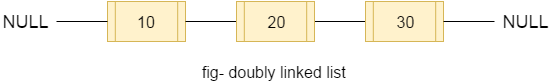
LinkedList implements the Collection interface. It uses Singly and doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.

### Hierarchy of LinkedList class

As shown in the above diagram, Java LinkedList class extends AbstractSequentialList class and implements List and Deque interfaces.

### Doubly Linked List

In the case of a doubly linked list, we can add or remove elements from both sides.



### LinkedList class declaration

Let's see the declaration for java.util.LinkedList class.

1. **public** **class** LinkedList<E> **extends** AbstractSequentialList<E> **implements** List<E>, Deque<E>, Cloneable, Serializable
2. Traverse Data one by one using Iterator
3. Print or get() Values using Index
4. Java LinkedList class maintains insertion order.
5. Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure.
6. Java LinkedList class can contain duplicate elements.
7. Java LinkedList class maintains insertion order.
8. Java LinkedList class is non synchronized.
9. In Java LinkedList class, manipulation is fast because no shifting needs to occur.
10. Java LinkedList class can be used as a list, stack or queue.

Difference between ArrayList and LinkedList

1. ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.
2. However, there are many differences between ArrayList and LinkedList classes that are given below.

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |

## Java ListIterator Interface

ListIterator Interface is used to traverse the element in a backward and forward direction.

### ListIterator Interface declaration

1. **public** **interface** ListIterator<E> **extends** Iterator<E>

 ListIterator<String> itr=al.listIterator();

 System.out.println("Traversing elements in forward direction");

**while**(itr.hasNext()){

         System.out.println("index:"+itr.nextIndex()+" value:"+itr.next());

 }

 System.out.println("Traversing elements in backward direction");

**while**(itr.hasPrevious()){

  System.out.println("index:"+itr.previousIndex()+" value:"+itr.previo

         }

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **ListIterator** |
| 1) | The Iterator traverses the elements in the forward direction only. | ListIterator traverses the elements in backward and forward directions both. |
| 2) | The Iterator can be used in List, Set, and Queue. | ListIterator can be used in List only. |
| 3) | The Iterator can only perform remove operation while traversing the collection. | ListIterator can perform ?add,? ?remove,? and ?set? operation while traversing the collection. |

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **Enumeration** |
| 1) | The Iterator can traverse legacy and non-legacy elements. | Enumeration can traverse only legacy elements. |
| 2) | The Iterator is fail-fast. | Enumeration is not fail-fast. |
| 3) | The Iterator is slower than Enumeration. | Enumeration is faster than Iterator. |
| 4) | The Iterator can perform remove operation while traversing the collection. | The Enumeration can perform only traverse operation on the collection. |

## Vector

Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

Traverse Using **Enumeration in vector** while in ArrayList and Linked we can Traverse using Iterator

v.addElement("irfan");//method of Vector

//traversing elements using Enumeration

  Enumeration e=v.elements();

**while**(e.hasMoreElements()){

   System.out.println(e.nextElement());

  }

Difference between ArrayList and Vector

ArrayList and Vector both implements List interface and maintains insertion order.

However, there are many differences between ArrayList and Vector classes that are given below.

Traverse Using **Enumeration in vector** while in ArrayList can Traverse using Iterator

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| 1) ArrayList is **not synchronized**. | Vector is **synchronized**. |
| 2) ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity. | Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity. |
| 3) ArrayList is **not a legacy** class. It is introduced in JDK 1.2. | Vector is a **legacy** class. |
| 4) ArrayList is **fast** because it is non-synchronized. | Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object. |
| 5) ArrayList uses the **Iterator** interface to traverse the elements. | A Vector can use the **Iterator** interface or **Enumeration** interface to traverse the elements. |
| ArrayList is not ?thread-safe? as it is not synchronized. | Vector list is ?thread-safe? as it?s every method is synchronized |

## Stack

The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

## Queue Interface

Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

Or

Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, first element is removed first and last element is removed at last.

### Queue Interface declaration

1. **public** **interface** Queue<E> **extends** Collection<E>

Queue interface can be instantiated as:

1. Queue<String> q1 = **new** PriorityQueue();
2. Queue<String> q2 = **new** ArrayDeque();

## PriorityQueue

The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

The PriorityQueue class provides the facility of using queue. But it does not orders the elements in FIFO manner. It inherits AbstractQueue class.

### PriorityQueue class declaration

Let's see the declaration for java.util.PriorityQueue class.

1. **public** **class** PriorityQueue<E> **extends** AbstractQueue<E> **implements** Serializable

An example where we are adding books to queue and printing all the books. The elements in PriorityQueue must be of Comparable type. String and Wrapper classes are Comparable by default. To add user-defined objects in PriorityQueue, you need to implement Comparable interface.

## Deque Interface

Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both the side. Deque stands for a double-ended queue which enables us to perform the operations at both the ends.

Java Deque Interface is a linear collection that supports element insertion and removal at both ends. Deque is an acronym for **"double ended queue".**

## Deque Interface declaration

1. **public** **interface** Deque<E> **extends** Queue<E>

Deque can be instantiated as:

1. Deque d = **new** ArrayDeque();

## ArrayDeque

ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.

ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

The ArrayDeque class provides the facility of using deque and resizable-array. It inherits AbstractCollection class and implements the Deque interface.

The important points about ArrayDeque class are:

* Unlike Queue, we can add or remove elements from both sides.
* Null elements are not allowed in the ArrayDeque.
* ArrayDeque is not thread safe, in the absence of external synchronization.
* ArrayDeque has no capacity restrictions.
* ArrayDeque is faster than LinkedList and Stack

### ArrayDeque class declaration

Let's see the declaration for java.util.ArrayDeque class.

1. **public** **class** ArrayDeque<E> **extends** AbstractCollection<E> **implements** Deque<E>, Cloneable, Serializable

## Set Interface

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

Set can be instantiated as:

1. Set<data-type> s1 = **new** HashSet<data-type>();
2. Set<data-type> s2 = **new** LinkedHashSet<data-type>();
3. Set<data-type> s3 = **new** TreeSet<data-type>();

## HashSet

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

The important points about Java HashSet class are:

* HashSet stores the elements by using a mechanism called **hashing.**
* HashSet contains unique elements only.
* HashSet allows null value.
* HashSet class is non synchronized.
* HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
* HashSet is the best approach for search operations.
* The initial default capacity of HashSet is 16, and the load factor is 0.75.

## Difference between List and Set

### A List Interface can contain duplicate elements whereas Set Interface contains unique elements only.

### HashSet class declaration

Let's see the declaration for java.util.HashSet class.

1. **public** **class** HashSet<E> **extends** AbstractSet<E> **implements** Set<E>, Cloneable, Serializable

## LinkedHashSet

LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

The important points about Java LinkedHashSet class are:

* Java LinkedHashSet class contains unique elements only like HashSet.
* Java LinkedHashSet class provides all optional set operation and permits null elements.
* Java LinkedHashSet class is non synchronized.
* Java LinkedHashSet class maintains insertion order.

### LinkedHashSet class declaration

Let's see the declaration for java.util.LinkedHashSet class.

1. **public** **class** LinkedHashSet<E> **extends** HashSet<E> **implements** Set<E>, Cloneable, Serializable

## SortedSet Interface

SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order. The SortedSet provides the additional methods that inhibit the natural ordering of the elements.

The SortedSet can be instantiated as:

1. SortedSet<data-type> set = **new** TreeSet();

## TreeSet

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of **TreeSet is quite fast.** **The elements in TreeSet stored in ascending order.**

The important points about Java TreeSet class are:

* Java TreeSet class contains unique elements only like HashSet.
* Java TreeSet class access and retrieval times are quiet fast.
* Java TreeSet class doesn't allow null element.
* Java TreeSet class is non synchronized.
* Java TreeSet class maintains ascending order.

### TreeSet class declaration

Let's see the declaration for java.util.TreeSet class.

1. **public** **class** TreeSet<E> **extends** AbstractSet<E> **implements** NavigableSet<E>, Cloneable, Serializable

Let's see a TreeSet example where we are adding books to set and printing all the books. The elements in TreeSet must be of a Comparable type. String and Wrapper classes are Comparable by default. To add user-defined objects in TreeSet, you need to implement the Comparable interface.

### Similarities Between HashSet, LinkedHashSet and TreeSet In Java :

* All three doesn’t allow duplicate elements.
* All three are not synchronized.
* All three are Cloneable and Serializable.
* Thread safe: If we want to use HashSet, LinkedHashSet and TreeSet in multi-threading environment then first we make it externally synchronize because both LinkedHashSet and TreeSet are not thread-safe.
* Iterator returned by all three is fail-fast in nature. i.e You will get ConcurrentModificationException if they are modified after the creation of Iterator object.
* **What is mean by fail-fast** in nature it means in tharaing we have 3 different method m1 m2 and m2 and if we access or modify any values say A then if method m1 modify A to b then same time m2 also access A then at time M2 will get b this is called fail-fast in nature

Similarity between PriorityQueue and TreeSet

**ThreadSafety**

First similarities between PriorityQueue and TreeSet is that both are not thread-safe, which means you cannot share them between multiple threads. If multiple threads are required to modify the TreeSet at the same time, then you must synchronize their access externally.

**Ordering**

The third similarity between PriorityQueue and TreeSet is that both can be used to access elements in a particular order. TreeSet is a SortedSet, hence it always keeps the elements in the order defined by their Comparator or natural order if there is no Comparator defined, while PriorityQueue will always make sure that head contains the lowest or highest value depending upon the order imposed by Comparator or Comparable.

**Eligibility**

When I say eligibility, which means which objects can be stored in PrioritySet and TreeSet? Is there any restriction or all objects are allowed? Well, there is, you can only store objects which implement Comparable or Comparator in both PriorityQueue and TreeSet because the collection classes are responsible for keeping their commitment i.e. PriorityQueue must adjust after every insertion or deletion to keep the lowest or highest element in head position. Similarly, TreeSet must re-arrange elements so that they remain the sorted order specified by Comparator or natural order imposed by Comparable.

Difference between PriorityQueue and TreeSet

**Underlying Data Structure**

This first and foremost difference is underlying data structure. PriorityQueue is a Queue and that's why it provides the functionality of FIFO data structure, while TreeSet is a Set and doesn't provide the Queue API.

**Duplicate Elements**

The second difference between them can be derived from the first difference i.e. properties of their underlying data structure. Since TreeSet is a Set it doesn't allow duplicate elements but PriorityQueue may contain duplicates. In the case of ties, the head of the priority queue is chosen arbitrarily.

**Availability**

The 5th and last difference between PriorityQueue and TreeSet class are that former was added in JDK 1.5 while TreeSet was available from JDK 1.4 itself. This is not a very significant difference in the age of Java 8, but if you are working with legacy systems still running on Java 1.5, a point worth remembering.

**Ordering**

The fourth difference, which is more subtle than you think because in similarities I have told that both are responsible for keeping some ordering. The key point is that in TreeSet all elements remain in the sorted order, while in priority queue apart from root, which is guaranteed to be smallest or largest depending upon Comparing logic, rest of element may or may not follow any ordering.

This means if you store same elements in the TreeSet and PriorityQueue and iterate over them, then their order will be different. TreeSet will print them in sorted order but PriorityQueue will not until you are always getting the element from the head. You can read a good core Java book e.g. Java: The Complete Reference, Ninth Edition to learn more about PriorityQueue implementation in Java.