Collections

## Array List:

The underlying data structure is resizable array and internally it uses indexes to store the elements.

It implements list, Serializable, Cloneable and Random access interfaces.

### Key Points:

Duplicate objects are allowed.

Insertion order is preserved.

Null insertion is possible.

Heterogeneous objects are allowed.

ArrayList implementation is not synchronized.

It is best suitable if our frequent operation is retrieval.

It is not recommended to use, if our frequent operations are insert or update.

ArrayListExample1.java

In this example, we are creating an Array list object and adding duplicate elements to it.

We are using for – each loop to iterate over the list elements. Next we are sorting the elements by using the Collections utility class and using an Iterator to iterate the list elemetns.

ArrayListExample2.java

In this example, we are creating an Array list by adding the custom employee objects. Here we are trying to add and remove employee elements from the list.

Note: Remove method will remove an employee if and only if the employee class provides an implementation to the equals() method.

## Linked List:

The underlying data structure is double linked list.

It implements list, Serializable, Cloneable interfaces but **not Random access** interfaces.

### Key Points:

Duplicate objects are allowed.

Insertion order is preserved.

Null insertion is possible.

Heterogeneous objects are allowed.

ArrayList implementation is not synchronized.

Best choice if our frequent operation is insertion or deletion.

LinkedListSample.java

In this example, we are creating an Linked list and we are using add() method to add elements to list and get() method to retrieve the elements. As we are using generics, we no need to do the type casting while retrieving the objects.

## Vector:

The underlying data structure is resizable array.

It implements list, Serializable, Cloneable and Random access interfaces.

### Key Points:

Duplicate objects are allowed.

Insertion order is preserved.

Null insertion is possible.

Heterogeneous objects are allowed.

It is best suitable if our frequent operation is retrieval.

It is not recommended to use, if our frequent operations are insert or update.

**Every method is synchronized. So vector by default thread safe.**

Below are the few vector specific methods.

addElement()

removeElement()

firstElement()

lastElement()

VectorSample.java

In this example, we are creating an Vector and we are using add() method to add the elements to it.

Next, we are using the Enumerator object to iterate the vector elements.

## Stack:

It is a child class of Vector.

It stores the elements in the last-in-first-out (LIFO) order.

It contains only one constructor.

StackSample.java

In this example, we created an instance of stack and added heterogeneous objects by using the push() method. It also demonstrates that how peek and pop operations are working.

## Hash Set:

The underlying data structure is Hash Table. It implements serializable and cloneable interfaces.

### Key Points:

Duplicate objects are **NOT** allowed. Violation leads to false and we won’t get any CTE or RTE.

Insertion order is **NOT** preserved and it is based on the hash code of the object.

Null insertion is possible.

Heterogeneous objects are allowed.

It is non-synchronized.

HashSet.java

In this example, we created an instance of Hash Set and added three elements to it. After that we tried to add duplicate element which returns false.

## Linked Hash Set:

It is the child class of Hash Set. The underlying data structure is Hash Table + Linked List.

It is exactly same as Hash Set except that it saves insertion order because of linked list.

i.e. elements are stored in the same order in which they adde.

LinkedHashSetSample.java

Execute the above sample and verify the order of the elements.

## Tree Set:

The underlying data structure is Balanced Tree and it stores the elements in ascending order.

### Key Points:

Duplicate objects are **NOT** allowed. Violation leads to false and we won’t get any CTE or RTE.

Insertion order is **NOT** preserved and all elements are inserted according to some sorting order.

Heterogeneous objects are **NOT** allowed violation leads to Class Cast Exception

Null insertion is possible only for the empty tree set.

We will get NullPointerException, if we are trying to insert NULL value to the non empty tree set.

We can use **Comparator Interface** along with tree set, if we want to store the elements in custom sorting order.

TreeSetSample.java

Execute the above sample and the output must display the elements in ascending order.

TreeSetSample2.java

## Priority Queue:

## Hash Map:

The underlying data structure is Hash Table. It is used to store a group of objects as a key value pairs.

An instance of HashMap has two parameters that affect its performance: **initial capacity** and **load factor**.

It doesn’t save the insertion order of the elements.

### Key Points:

Insertion order is **NOT** preserved and it is based on the hashcode of the keys.

Duplicate keys are not allowed and values can be duplicated.

NULL key allowed only once and NULL values are allowed multiple times.

It is not synchronized.

To identify the duplicate keys internally it uses the .equals() method.

HashMapSample.java

## Linked Hash Map:

It is the child class of Hash Map. The underlying data structure is Hash Table + Linked List.

It is exactly same as Hash Map except that it saves insertion order because of linked list.

i.e. elements are stored in the same order in which they added.

LinkedHashMapSample.java

Execute the above sample and verify the order of the elements.

## Identity Hash Map:

This class implements the map interface and internally it uses the Hash Table.

To identify the duplicate keys internally it uses the == operator. i.e Reference comparison

IdentityHashMapSample.java

In order to find the difference between hash map and identity hash map,

First run the above program and check the output

Next, instead of creating the Identity hash map object in the above sample, create Hash Map object and execute it again.

## Weak Hash Map:

A hashtable-based Map implementation with **weak keys**. An entry in a WeakHashMap will automatically be removed when its key is no longer in use.

Garbage Collector dominates the weak hash map.

WeekHashMapSample.java

In order to find the difference between hash map and weak hash map,

First run the above program and check the output

Next, comment the weak hash map object creation line & uncomment its above line and run it again.

## Tree Map:

The underlying data structure is “RED BLACK Tree” and it stores the elements according to the natural sorting of the keys.

### Key Points:

Insertion order is NOT preserved.

Duplicate keys are NOT allowed.

Duplicate values are allowed.

Null insertion is possible only for the empty tree map.

We will get NullPointerException, if we are trying to insert NULL value to the non empty tree map.

We can use **Comparator Interface** along with tree map, if we want to store the elements in custom sorting order.