**Collection:**

**What is Collection?**

* The Collection is a framework that provides an architecture to store and manipulate the group of objects.
* It provides operations that you perform on a data such as searching, sorting, insertion and deletion on the group of objects.
* Collection represents a single unit of objects as a group.

**Why do we use Collection?**

(Explain Difference between Array & Collection)

|  |  |
| --- | --- |
| Array | Collection |
| It is Fixed in Size. | It is Growable in nature. |
| It can hold only Homogeneous Data  Elements. | It can hold both Homogeneous and  Heterogeneous Elements. |
| With Respect to Memory Arrays are Not  Recommended to Use. | With Respect to Memory Collections are  Recommended to Use. |

**What is Difference between Collection & Collections?**

|  |  |
| --- | --- |
| Collection | Collections |
| It is an interface. | It is class. |
| It can be used to Represent a Group of  Individual Objects as a Single Entity. | It is used to sort and synchronize the  collection elements. |
| It provides the methods that can be used for data structure. | It provides the methods which can be used for various operations on a collection. |

**Which Collection object you have used in Project?**

* List - ArrayList for getting lists of object from Database.
* Set – HashSet for mapping POJOs using hibernate. (One-to-Many & Many-to-One)

**How Arraylist works?**

* When we create object of Arraylist, it create Arraylist instance with default capacity 10.
* Arraylist capacity increases with formula – New Capacity = ((3/2) x Old Capacity)+1
* When Arraylist increments with new capacity then data from old Arraylist is copied into new instance and old instance is destroyed.
* When we add or delete data into the Arraylist then multiple data shift operations are performed.
* Arraylist follows Indexing.

**Difference between Arraylist & Vector?**

|  |  |
| --- | --- |
| Arraylist | Vector |
| It is not Synchronised. | It is Synchronised. |
| It is not Thread Safe | It is Thread Safe |
| Its Default Capacity :- 10 and It increases  after by (Old+Old\*(3/2))+1 | Its Default Capacity :- 10 and It increases  after by Old+Old\*100% |
| Performance is high. | Performance is low. |
| Enumeration cannot be used. | Enumeration can be used. |

**How we synchronize Arraylist?**

By Default ArrayList Object is Non- Synchronized but we can get Synchronized Version ArrayList Object by using the following Method of Collections Class.

public static List synchronizedList(List l)

E.g. ArrayList al = new ArrayList ();

List l = Collections.synchronizedList(al);

**Explain all Constructors of Arraylist?**

ArrayList l = new ArrayList();

It creates an Empty ArrayList Object with Default Initial Capacity 10.

ArrayList l = new ArrayList(intinitialCapacity);

It creates an Empty ArrayList Object with specified Initial Capacity.

ArrayList l = new ArrayList(Collection c);

It creates an equivalent ArrayList Object for the given Collection Object.

**Why Arraylist is fast for retrieval operation?**

It implements RandomAccess Interface, hence Arraylist is fast for retrieval operation.

**Why set doesn’t allow duplicates?**

Set internally uses HashMap.

HashMap object is created in every Set implemented class.

Here HashMap stores key as all inserted elements and value as a dummy object created with new keyword.

As HashMap doesn’t accept duplicate keys, so set don’t allow duplicate values.

**How add () of set works?**

Its return type is Boolean, it returns true or false value.

It uses equals ( ) method of object class.

It implements HasMap internally.

**How Linkedlist works? (Why insertion & deletion is fast in Linkedlist?**

When we create an object of Linkedlist and add an element to it.

It stores element as a node in which previous & next node address is also stored.

Node format = ||prev. node addr.| (value) |next node addr.||

Due to previous & next node address is stored, hence while updation or insertion & deletion operation data shift operation need not to perform and it makes Linkedlist fast.

**Define Linkedlist?**

* It is one of implemented class of List interface in collection framework.
* It allows duplicate values, Insertion order is preserved & indexing is maintained.
* It implements Clonable, Serializable interfaces.
* It follows doubly linked list structure.
* It is mostly preferable for insertion & deletion operations.

**What is Map?**

* Map is used for store different object in the pair of “key” and “value”.
* In map, “key” should be unique.
* Insertion order will not be maintained in Map.

**Difference between Hashmap & Hashtable?**

|  |  |
| --- | --- |
| HashMap | Hashtable |
| It is not Synchronised. | It is Synchronised. |
| It allows multiple threads at a time. | It allows single thread at a time. |
| It is not thread safe. | It is thread safe. |
| Null key (once) & Null value is allowed. | Null key & Null value is not allowed. |
| Its performance is fast. | Its performance is slow. |

**How HashMap works?**

* When we create HashMap object, HashMap instance as per default capacity 16 buckets is created.
* When we perform add (put ( )) operation, it accepts data in key & value format.
* Internally hashing technique is used, that generates hashcode for key and also calculate index to find bucket location for inserting data in HashMap instance.
* It will store element at that location as a node format.
* ||previous node address| (Key) | (Value) |next node address||
* Now when we perform retrieval (get ( )) operation, it asks for key.
* Again hashing technique is used and bucket location is identified, then equals ( ) method is used to compare key content and if it returns true then value is retrieved.

**What is Hash Collision?**

* In HashMap, if two keys have same hashcodes then such situation is called as hash collision.
* In such case, while adding data, doubly Linkedlist is created to insert data.
* ||prev. node addr.| (Key1) | (Value1) |next node addr.||  || | (Key2) | (Value2) | ||
* And retrieval operation is performed using equals ( ) method.

**What happens when we put same keys in Map?**

* If we add a key-value pair where the key exists already, put method replaces the existing value of the key with the new value.

**What is Contract between equals () & hashcode ()?**

* If equals ( ) returns true, then objects must have same hashcodes.
* If equals ( ) returns false, then objects may or may not have same hashcodes.
* If hashcodes of objects are same, then we can’t conclude output of equals ( ), it may be true or may be false.
* If hashcodes of objects are different, then output of equals ( ) must be false.

**Difference between Hashmap & Synchronised (or Concurrent) Hashmap?**

|  |  |
| --- | --- |
| HashMap | Synchronised or Concurrent HashMap |
| It is non-Synchronized in nature. | It is Synchronized in nature. |
| It is not Thread-safe. | It is thread-safe. |
| Performance is high. | Performance is low. |
| It can throw  ConcurrentModificationException. | It doesn’t throw  ConcurrentModificationException |

**Difference between Comparable & Comparator?**

|  |  |
| --- | --- |
| Comparable | Comparator |
| This interface is from java.lang package. | This interface is from java.util package. |
| It is used for Default sorting. | It is used for Custom sorting. |
| It has only one method i.e. compareTo. | It has two methods i.e. compare & equals. |
| Programmer decides how sorting is to be  done. | User decides how sorting is to be done. |

**What is Identity Hashmap?**

In IdentityHashMap JVM will Use == Operator to Identify Duplicate Keys, which is meant for Reference Comparison.

e.g. Integer i=new Integer (5); Integer i1=new Integer (5);

Map m=new IdentityHashMap(); m.put(i, "java");

m.put(i1, "python");

System.out.println(m); => {5=java, 5=python}

**What is fail safe & fail fast iterator?**

* Using iterators we can traverse over the collections objects. The iterators can be either fail-safe or fail-fast.
* Fail-safe iterator means they will not throw any exception even if the collection is modified while iterating over it.
* Whereas Fail-fast iterator throw an exception (ConcurrentModificationException) if the collection is modified while iterating over it.
* Fail-Fast Iterators internal working:
* Every fail fast collection has a modCount field, to represent how many times the collection has changed/modified.
* So at every modification of this collection we increment the modCount value. For example the modCount is incremented in below cases:
* When one or more elements are removed.
* When one or more elements are added.
* When the collection is replaced with other collection.
* When the collection is sorted.
* So everytime there is some change in the collection structure, the mod count is incremented.
* Now the iterator stores the modCount value in the initialization as below: int expectedModCount = modCount;
* Now while the iteration is going on, expectedModCount will have old value of modCount. If there is any change made in the collection, the modCount will change and then an exception is thrown
* Unlike the fail-fast iterators, fail-safe iterators traverse over the clone of the collection. So even if the original collection gets structurally modified, no exception will be thrown.

**What is WeakHashmap?**

* In Case of WeakHashMap if an Object doesn't contain any References then it is Always Eligible for GC Even though it is associated with WeakHashMap.
* Garbage Collector Dominates WeakHashMap.
* Both null values and null keys are supported in WeakHashMap.
* It is not synchronised.