**Java Collections Framework:**

**Student1, student2, stu3, stu4 – Student is collection**

**Collection is group of objects. Here student is a collection which has stu1, stu2 etc objects**

**We can call collection as a single entity which can be used to refer multiple objects.**

**Collections framework is nothing but a group of collection.**

**Collections is nothing but a group of classes and interfaces by which we can represent group of objects into a single entity**

**Why we need this collection?**

**int a =10;**

**int b=20;**

**int c=30;**

**Suppose we want to store 100 values, in that case creating 100 variables and storing its value is not the efficient way. This can be efficiently done using array concept**

**int stu[]= new int[100]; 🡪 we can store only integer values**

**Object stu[] = new Object[100]; 🡪 we can store values of multiple data types but the memory is fixed**

**Disadvantages of using an array:**

1. **In an array memory is fixed, we cant extend the memory at runtime**
2. **Array is homogenous which means that I can store values of a single data type. In a integer array, we can store only integer value and cannot store float values**
3. **In an array, there are no predefined or readymade methods available in array.**

**To overcome the above challenges, we are going to use collection framework**

1. **We can extend the memory size at runtime using collection**
2. **we can store values of multiple data types in a single collection**
3. **In collection, we can many predefined methods using which we can manipalute values at a faster rate**

**Collection (Interface)**

Queue

(Interface)

Set (Interface)

List (Interface)

There is a class called **Collections** which implements the **Collection Interface** and it is present inside **java.util package**.

Classes implementing **the List Interface are ArrayList, LinkedList, Vector which extends Stack interface. Vector is legacy class which is available in the older versions of Java**

Classes implementing **Set are HashSet, LinkedHashset, Tree set**

Classes implementing **Queue are PriorityQueue**

|  |  |
| --- | --- |
| **List(Interface)** | **Set (Interface)** |
| List allows insertion of dupplicate values | Set accepts only unique values |
| List maintains insertion order | Set doesn’t main order |

**Queue** follows the FIFO (first in first out) order.

**Map** **Interface**:

Map interface is not a child interface of Collection interface and it a independent interface in Java.

Map follows Key and value pair structure. For each value, there would be key associated with it.

Keys should be unique but values can be duplicate

|  |  |
| --- | --- |
| Key | Value |
| 101 | Java |
| 102 | C |
| 102 | Python |

A = java, c ,python,c++

B= java, python

A.removeAll(B) 🡪 removed Java and python from A

A.retainAll(B) 🡪 will retain Java and Python in A and deletes all the other elements in A collection

**Methods that are available in the Collection Interface:**

1. add(obj) -> add a specific element to a collection
2. remove(obj) -> remove a specific element from a collection
3. clear() -> will remove all the elements in the collection
4. size() -> returns the size of the collection
5. isEmpty() -> this method will return True, if the collection is empty. It will return False, if the collection has elements
6. contains(obj) -> to search for a specific element from a collection
7. addAll(Collection c) -> inserts a new collection into the existing collection
8. removeAll(Collection c) -> removes specific collection from the existing collection
9. retainAll(Collection c) -> retains a specific collection from the existing collection
10. toArray(collection c) -> convert a collection to array

**Methods that are available specific for List Interface:**

1. allows duplicates
2. maintains order of insertion
3. Allows null values
4. add (index, obj) -> adds an element at the specific index
5. addAll(index, collection c) -> adds a collection at the specific index
6. remove(index, obj) -> remove element from a specific index
7. removeAll(index, collection) -> removes the collection from the specific index
8. get(index) -> to retrieve an element from a specific index
9. set(index, obj) -> replaces the element at that specific index

A = java, c ,python,c++

A.add(1, “Ruby”)

A= java, Ruby, c, python, c++

A.set(1,”Go”)

A= java, Go, c, python, c++

ArrayList and LinkedList will have all the above methods

**LinkedList:**

1. It maintains Insertion order
2. Allows duplicates
3. Allows null values

LinkedList has implemented the List interface and also Deque (extends the Queue Interface) interface in Java. It means that Linkedlist will have all the methods that Arraylist has. On top of it, it will have some more methods with regard to Deque interface.

Operations of List are,

1. Adding/removing elements from a collection 🡪 LinkedList
2. Retrieving elements from a collection using get() 🡪ArrayList

ArrayList structure

|  |  |  |
| --- | --- | --- |
| A | B | C |

LinkedList has A, B, C and D as its values.

Structure of LinkedList,

|  |  |  |
| --- | --- | --- |
| Previous memory value | Value | Next Memory value |

Address of A is 101, B is 102, C is 103 and D is 104

Node 1

|  |  |  |
| --- | --- | --- |
| Null | A | 102 |

Node 2

|  |  |  |
| --- | --- | --- |
| 101 | B | 103 |

Node 3

|  |  |  |
| --- | --- | --- |
| 102 | C | 104 |

Node 4

|  |  |  |
| --- | --- | --- |
| 103 | D | Null |

Queue 🡪 FIFO (First in First out)

Insertion at top

|  |
| --- |
| D |
| C |
| Index 1 ->B |
| Index 0 ->A |

|  |
| --- |
| D |
| C |
| Index 1 ->B |
| Index 0 ->A |

Deletion at bottom

Stack 🡪 First In Last out (FILO)

Insertion and Deletion happens at top

|  |
| --- |
| D |
| C |
| Index 1 ->B |
| Index 0 ->A |

Methods specific to LinkedList:

1. addFirst() -> adds element to the first index
2. addLast() -> adds element at the last index
3. removeFirst() -> remove the first element in Linkedlist
4. removeLast() -> remove last element in Linkedlist
5. getFirst()-> fetch the first element in Linkedlist
6. getLast() -> fetch the last element in linkedlist

Set Interface:

**Collection (Interface)**

Queue

(Interface)

Set (Interface)

List (Interface)

HashSet, LinkedHashSet, TreeSet are the different implementations of the Set.

**HashSet**:

1. Wont allow duplicate values
2. Insertion order is not maintained
3. Hash set is not thread-safe and so it would be faster in execution
4. Searching operations would be more efficient in HashSet
5. Null values are allowed
6. It can store both Homogenous and Heterogenous data
7. Since HashSet doesn’t maintain order, there is no concept of index here. Because of this reason, there is no get() method in set
8. The default size of HashSet can store 16 values and it has a default fill ratio/load factor of 0.75

**ArrayList**:

Default value is 10

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

If I try to insert more than the default value(10), a new arraylist will be created and all the values will be copied to the new arraylist and it would be referenced. The old arraylist obj would be destroyed

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

HashSet:

It can store upto 16 values as default

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |

If the 75% of the memory is occupied, then a new Hashset will be created and its values will be copied. It wont wait until the complete memory is exhausted unlike ArrayList.

**Methods that are available in HashSet():**

1. add()
2. addAll(collection c)
3. remove()
4. removeAll(Collection c)
5. isEmpty()
6. contains()
7. containsAll(collection c)
8. To iterate over an Hashset, we can do using foeach loop, iterator and for loop
9. If we need to sort, shuffle. We need to convert the Hashset to Arraylist and then use sort and shuffle methods

**LinkedHashSet:**

1. Duplicates are not allowed
2. Maintains Insertion order since it has extended the Linkedlist class whereas HashSet doesn’t maintain insertion order
3. Load factor/ fill ration are 0.75 which is same as HashSet
4. Default memory is 16

**TreeSet:**

TreeSet has implemented the Sorted Interface which is the child interface of Set Interface. Because of this reason, TreeSet maintains Ascending order for homogenous data and natural order for heterogenous data.

1. Duplicates ae not allowed
2. Load factor/ fill ratio is 0.75
3. Default memory is 16

**Queue:**

**FIFO (First In First Out)**

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **3** | **4** |

Head

Tail

There are 2 classes implementing the Queue Interface,

1. PriorityQueue
2. LinkedList

|  |  |
| --- | --- |
| LinkedList | PriorityQueue |
| Insertion order maintained | Insertion order is maintained |
| Duplicates allowed | Duplicates allowed |
| Heterogenous data | Homogenous data |

Both these are used for inserting values to the queue

1. add() - If there is an issue in inserting the elements, then add() will throw exception
2. offer() - If there is an issue in inserting the elements, then offer() will throw null (false) value

If we want to fetch the element at head,

1. element() -> If there is any problem while fetching, then element() will return No such Element exception
2. peek() -> If there is any problem while fetching, then peek() will return null

If we want to fetch and remove the element at the head,

1. remove() -> If there is any problem while removing, then remove() will throw exception
2. poll() -> If there is any problem while removing, then poll() will throw null

Above 6 methods are common both for LinkedList and PriorityQueue

**Map Interface:**

Map interface is not the child interface of Collection interface and it is an independent interface.

Map stores in the values in the form of Key, value pair.

|  |  |
| --- | --- |
| Emp Id (Key) | Emp Name(Value) |
| 101 | John |
| 102 | Smith |
| 103 | David |

Classes implementing the Map interface,

1. HashMap (After java 1.2)
2. TreeMap (After java 1.2)
3. LinkendHashMap (After java 1.2)
4. Hashtable (Before Java 1.2 🡪 legacy)

HashMap:

1. Stores the value in the form of Key, value pair. We can call each key value pair as an entry. In the above example, we have 3 entries.
2. Duplicate keys are not allowed whereas duplicate values are allowed
3. HashMap has implemented the hashcode concept because of which insertion order is not preserved
4. In Hashmap, we can have one null key whereas we can many null values
5. HashMap has a default value of 16 and load factor of 0.75
6. Hashmap is faster in search ing when compared to other classes of Map interface because Hashmap doesn’t maintain order

Methods that are present in Map interface,

1. put(k,v) -> to add an entry to a map
2. putAll(collection c) -> to add an entire map to the existing map
3. remove(key) -> removes the value of the specific key
4. clear() -> to clear the complete map
5. containsKey(key)-> it will return true, if that specific key is present whereas it will return false if that key is not present
6. containsValue(value) -> it will return true, if that specific value is present whereas it will return false if that value is not present
7. isEmpty(map) -> if the map is empty, it will return true
8. size() -> to return the size of the map
9. keyset()-> it will return the keys in the form of set
10. values()-> it will return values in the form of collections
11. entrySet()-> will return all the entries from the map as a set

**Entry (Interface):**

Entry interface is the sub interface of Map interface.

1. getKey() 🡪 it will return the key of that particular entry
2. getValue()-> it will return the value of that particular entry
3. setValue()🡪 we can update the value of that particular entry

All the above methods are present in all the implementing classes of Map Interface

Hashtable:

Similarities:

|  |  |
| --- | --- |
| HashMap | Hashtable |
| Stores values in key, value pair | Stores values in key, value pair |
| Insertion order is not maintained | Insertion order is not maintained |
|  |  |

Differences:

|  |  |
| --- | --- |
| HashMap | Hashtable |
| Hashmap allows one null key and multiple null values | Null key and null value is not allowed in hashtable |
| It has a default value of 16 and fill ratio was 0.75 | It has a initial capacity of 11 and the fill ratio is 0.75 |
| It is not-synchronized and so it allows multiple threads at the same time | It is synchronized and it allows only one thread at a time |
| Execution would be faster | Execution would be slower when compared to HashMap |

LinkedHashMap and TreeMap:

|  |  |
| --- | --- |
| LinkedHashMap | TreeMap |
| Maintains Insertion Order | Maintains Ascending Order i.e. natural order of the keys |
| One null key and multiple null values are allowed | Null key is not allowed in Treemap |