Collection FrameWork

Yess Infotech

Collection framework:

Collection framework is used to store group of objects. It is a collection of multiple interfaces and classes which allow programmer to store heterogeneous types of objects.

Why prefer collection over an array:

1. Size of an array is fixed.
2. We cannot store heterogenous types of objects. We need to define datatype while declaring an array.
3. Performing crud operations is complex using array.

Advantages of collection:

1. Size of collection can change dynamically.
2. We can store heterogenous types of objects.
3. Many inbuild methods are available to perform crud operations easily.

Hierarchy Of collection Framework

Diagram

1. List:

1.It is an interface.

2. Present inside java.util package.
3. With the help of list we can store heterogenous types of objects.
4. It can store duplicates values.
5. We can create object reference variable for List but we cannot create Object for list. We can create object for its child classes i.e. ArrayList, LinkedList and Vector

6. Syntax:

```
List <data-type> list1= new ArrayList();
List <data-type> list2 = new LinkedList();
List <data-type> list3 = new Vector();
```

Methods Used In collection Framework:

1. To add an element
   a. add(value)
   b. add(index, value)
   c. addAll(collection c)

2. To remove an element
   a. remove(index)
   b. removeAll(collection c)

3. To search an element
   a. Contains(value)
   b. containsAll(collection c)

4. Some miscellaneous methods
    a. Size()
    b. isEmpty
    c. clear()
    d. get(index)

For Applying Loop on Collection

1.iterator()

2. hasNext()

3. next()

2. Set

1. It is an interface.

2. It is a child of collection interface.

3. Present inside java.util package.

4. It does not allow Duplicate value

5. It does not have index value.

6. We cannot create object for set interface but we can create object for its child class

Eg. HashSet, TreeSet, LinkedHashset

Syntax:

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>();

A. Hashset:
    1. Does not maintain insertion order.

B.LinkedHashSet:

    1. It maintains insertion order.

C. TreeSet:

    1. It stores elements in ascending order.
    2. It does not allow null.

Note:

Methods which requires index value does not supported in set.

We can not access specific element in set.

Queue

1. **Queue is an Interface**:
    - Queue is an interface in Java.
2. **Child of Collection Interface**:
    - Queue extends the Collection interface.
3. **Located in java.util Package**:
    - Queue is part of the java.util package.
4. **FIFO Order**:

- A Queue typically orders its elements in First-In-First-Out (FIFO) manner, where elements are added at the end and removed from the front.

5. **Cannot Instantiate Queue Directly**:
    - You cannot create an object of the Queue interface directly, but you can instantiate one of its implementing classes:
        - **Examples**: LinkedList, PriorityQueue, ArrayDeque
6. **Syntax**:

Queue<data-type> q1 = new LinkedList<data-type>();

Queue<data-type> q2 = new PriorityQueue<data-type>();

Queue<data-type> q3 = new ArrayDeque<data-type>();

Map Interface

1. Introduction:

- The Map interface is a part of the Java Collection Framework.
- It is used to store key-value pairs, where each key is unique.
- Map is not a subtype of the Collection interface.
- Unlike collections, which store individual elements, a Map stores pairs (key-value).

2. Characteristics of Map:

- **Unique Keys:** Each key in a Map must be unique. If a key is added twice, the old value is replaced by the new one.
- **Multiple Implementations:** Common implementations of Map include HashMap, TreeMap, and LinkedHashMap.

Syntax:

Map<KeyType, ValueType> map1 = new HashMap<KeyType, ValueType>();

Map<KeyType, ValueType> map2 = new LinkedHashMap<KeyType, ValueType>();

Map<KeyType, ValueType> map3 = new TreeMap<KeyType, ValueType>();

 HashMap:

- It does not maintain any order of keys.
- Allows one null key and multiple null values.

 LinkedHashMap:

- Maintains the insertion order of keys.
- Allows one null key and multiple null values.

 TreeMap:

- Stores keys in a sorted order (natural order or custom comparator).
- Does not allow null keys.

Commonly Used Methods in Map:

- **To Add or Update an Element:**
  - put(key, value): Associates the specified value with the specified key.
  - putAll(map): Copies all of the mappings from the specified map to this map.

- **To Remove an Element:**
  - remove(key): Removes the mapping for a key.
  - clear(): Removes all of the mappings from this map.
- **To Access an Element:**
  - get(key): Returns the value to which the specified key is mapped.
  - containsKey(key): Returns true if the map contains a mapping for the specified key.
  - containsValue(value): Returns true if the map maps one or more keys to the specified value.
  - keySet(): Returns a Set view of the keys contained in this map.
  - values(): Returns a Collection view of the values contained in this map.