Collection Classes

1- What is Collection in java?

- in generally Programming term A **collection** is an object that groups multiple elements into a single unit.

2- What is Collection Framework in java?

- Collection Framework in java like a toolbox that helps you to store, organize and manage the group of multiple Objects. It is a Deta Structure in java it implemented by using java.util Package.

3- Why Collection Born?

- Before the Collection framework was introduced in Java, arrays were the primary way to store multiple elements in a single unit. However, arrays have some significant limitations:
- **Fixed Size**: Once an array is created, its size cannot be changed.
- **Lack of Built-in Methods**: Arrays do not provide built-in methods for common operations like searching, sorting, insertion, or deletion.
- **No Standard Data Structures**: Arrays do not support different types of data structures like lists, sets, or maps.
- To overcome these limitations, the **Collection framework** was introduced in Java.
- **Advantages of the Collection Framework:**
- Collection is **dynamically growable**.
- It provides different **data structures** (like List, Set, Map).
- It supports many **in-built methods** for **insertion, deletion, searching, and storing**.

4- Collection Framework Provided Interfaces?

- Collection Framework means **working with single or group of objects**.
- To work with a single object collection, we use the Collection interface.
- To **work with group of objects as key-value pairs**, we use the **Map interface**.
- The main interfaces provided by the Collection Framework are:
-

1. **List** –
   - Allows **duplicate elements**
   - **Maintains insertion order**
   - Examples: `ArrayList`, `LinkedList`
2. **Set** –
   - **Doesn't allow duplicates**
   - **No guaranteed order** (unless using `LinkedHashSet` or `TreeSet`)
   - Examples: `HashSet`, `TreeSet`
3. **Queue** –
   - Follows **FIFO (First-In, First-Out)** order
   - Useful for tasks like processing orders, messages, etc.
   - Example: `PriorityQueue`, `ArrayDeque`
4. **Map** –
   - Stores data in **key-value pairs**
   - **Doesn't allow duplicate keys**, but can have duplicate values
   - Examples: `HashMap`, `TreeMap`

- **The framework also provides algorithms like sorting and searching, and it supports generics, which makes the collections type-safe. It's widely used in real-world Java development for storing, retrieving, and manipulating data efficiently."**

Collection Classes

5- What are the Legacy Classes in java?

- Legacy classes and interfaces refer to those which were **introduced before the Collection Framework**, i.e., before **JDK 1.2**.
- They are now considered **outdated**, but still **supported for backward compatibility**.
- Example: **Vector, Stack, Hashtable, Enumeration.**

6- What is fail-Safe and Fail-Fast in java Collections?

- **Fail-Fast** is a mechanism in Java Collections where **iterator throws ConcurrentModificationException** if the collection is **structurally modified during iteration** (like adding or removing elements directly from the collection while using a loop or iterator).
- We will not get this Exception if you modify iterator own method like add, remove or if you work on copy of collection it allows to safe Modification (using copyonWriteArrayList () method) known as fail-Safe iterator
- **Fail-safe:** If we **modify the collection using iterator's own methods** like iterator.remove(), it will **not throw an exception**.
- Also, if we **work on a copy of the collection**, then it's safe.
  Example: Using CopyOnWriteArrayList — this is called a **Fail-Safe iterator**.

7- What is push and pop Operation?

- Inserting an element into a stack is push Operation where extracting an element from the top of stack is known as pop Operation

8- List Interface (I) in java?

==========

- List is a sub-interface of Sequence Collection It extends from the **Collection interface**.
- It is used to store a group of elements in a proper **sequence (order) based on indexes** because it internally us e Array Concept it allows Duplicate and Null value.
- We use **List interface** when we want to **Store elements in order, allow duplicates Access elements by index Perform sorting and searching Operation because it Use different types of Lists**
- **ArrayList** – Best for searching and random access
- **LinkedList** – Best for insertion and deletion

9- How many Ways we can fetch or retrieve the Object from the Collections?

- We fetch so many Ways the Collections Comonly We use.
- 1) By using traditional for-Loop/Ordinary For-Loop [1.0V]
- 2) by using foreach Loop [1.5V]
- 3) by using toString method.
- 4) by using Enumeration interface **(hasMoreElements())** [1.0V]
- 5) by using iterator [1.2]
- 6) by using listiterator [1.2V]
- 7) by using foreach (Consumer) method [1.8V)
- 8) by using splititerator [1.8]
- 9) by using method reference [1.8]
- 10) by using Stream Api [1.8]

10- What is Enumeration in java?

- It is a predefined interface available in java.util package from JDK 1.0 onwards (Legacy interface).

Collection Classes

- We can use Enumeration interface to fetch or retrieve the Objects one by one from the Collection because it is a cursor. (hasMoreElements (), nextElement ()).
- It will only work with legacy Collections classes.

11- Iterator (I)?

- It is a predefined interface available in java.util package available from 1.2 version.
- is used to fetch/retrieve the elements from the Collection in forward direction only because it is also a cursor. (iterator ()).

12- ListIterator<E> interface?

- It is a predefined interface available in java.util package and it is the sub interface of Iterator available from JDK 1.2v.
- it is used to retrieve the Collection object in both the direction i.e. in forward direction as well as in backward direction listIterator ();

13- ArrayList <E> in java?

==========

- ArrayList is a **predefined class** in Java, available in the **java.util package**, and it **implements the List interface**.
- It **accepts null and duplicate values** It can store both **homogeneous** (same type) and **heterogeneous** (different types) elements.
- ArrayList is **dynamically growable**, which means the size increases automatically when needed.
- It stores elements in **order**, based on **index**, just like an **array**, so it's also called a **dynamic array**.

14- Uses Of ArrayList and capacity?

- Best for **retrieving or fetching data when duplicate allow and Thread Sefty is not Required**
- Initial capacity is **10**, same as Vector.

15- Time Complicity of ArrayList?

- The time complicity of ArrayList to insertion and deletion of an element from the middle or first would be O(n) because n number of elements are relocating her index position it take time so it not good choice for modification.
- On the Other hand, time Complicity of ArrayList for retrieving the element from the collection is always (1) Because it using get () method you get the object randomly or ArrayList implements Random Access so you can fetch the element randomly.

16- Different Between Vector and ArrayList?

- **Both are implementation classes of the List interface**, which is part of the Collection Framework.
- Both allow null and duplicate values.
- **Both store elements based on index**, because **internally they use array**
- **Both are dynamically growable, but their capacity growth is different:**
- in case of **Vector**, when the list is full, its **capacity becomes double**.
- **In case of ArrayList, when the list is full, its capacity increases by 50%.**
- - Formula: (currentCapacity * 3) / 2 + 1
- We should go with Vector when Thread Sefty is Require on the other hand we should go with ArrayList

  When Thread Sefty not Require but Collections class provide method called

  Collections.synchronizedList(null)which is create Thread same List.

Collection Classes

17- What is O(n) and O(1)?

- **O(n)** and **O(1)** are part of **Time Complexity**, which tells us **how much time a program or operation will take** based on the size of data.
- **O(n)** – means linear Time: means the **time taken increases with the number of elements (n)**.
- If elements increase, time also increases.
- **O(1)** -means Constant time: means the **time taken is always the same**, no matter how many elements are there.

18- LinkedList <E>

==========

- LinkedList is an **implementation class of the List interface**, which is under the **Collection interface**, and available in **java.util package** from **JDK 1.2 version**.
- It stores data in **non-contiguous memory locations**.
- LinkedList uses a **Doubly Linked List** Internally That means LinkedList is a **collection of nodes**, where each node is **interlinked** with each other. Means One node contains **reference of next and previous node**, along with the **data**.
- It **doesn't have any default capacity** like ArrayList or Vector.
- LinkedList is **suitable for modifying data**, especially in the **middle or at the beginning**.
- because there is **no shifting**, insertion and deletion are very fast —
  ✅ Time Complexity is **O(1)**.

19- Different Between ArrayList and LinkedList in java?

- Both are **implementation classes of List interface**, which is part of the **Collection interface**.
- In case of **ArrayList**, elements are stored in **contiguous memory locations.**
- **(All elements are placed side by side in memory.)**
- On the other hand, **LinkedList** stores elements in **nodes**, which are placed in **non-contiguous memory locations.**
- **(Each node is connected to the next and previous one using references.)**
- **ArrayList** is **suitable for fetching or retrieving** elements. Time complexity is **O(1)** using index.
- **LinkedList** is **suitable for modifying data**, especially in the **middle or beginning**. time complexity is **O(1)** for insertion/deletion (if pointer is already available).

20- Set <E> interface

==========

- Set is a sub interface of the Collection interface, available in the java.util package since JDK 1.2.
- It **does not allow duplicate elements**, unlike ArrayList and LinkedList which can store duplicates.
- It **does not maintain any specific order** of elements because internally It does not use Array concept, Actually It uses hashing algorithm). **Or/**
- `Set` interface does not maintain any specific order of elements because **internally it does not use the Array concept**.
- Actually, it uses a **hashing algorithm**, through which elements are stored in **buckets** using the **hashing technique**.
- Because of this, the **time complexity** for inserting and deleting elements is usually **O(1)** — which makes it very efficient.

21- Why we use Set interface in java?

- We use the **Set interface** when we want to **store a group of unique elements** meaning, **no duplicates are allowed**.

Collection Classes

- Internally, most Set classes (like HashSet) use **hashing algorithm**, which makes them **fast** for operations like **adding**, **removing**, and **searching**. **(Unique Email Addresses for Event)**

22- What is Hashing Algorithm in java?

- **Hashing algorithm** is a technique through which we can **search**, **insert**, and **delete** elements in a **more efficient way** compared to our traditional indexing methods.
- Hashing algorithm, internally uses Hashtable data structute, Hashtable data structure internally uses Bucket data structure.
- When we insert an element, the **hashing algorithm** calculates a **hash code** using a **hash function**.
- Hash function is mainly used to find the bucket location in the hash table data structure
- Hash function using a formula to find the bucket Location **(Hash table = key% table length=bucket Location)**

23- HashSet <E>

==========

- HashSet is a **predefined class** available in the **java.util package**, and it is part of the **Set interface**.
- It is an **unordered** and **unsorted** set.
- It **doesn't allow duplicate elements** or **doesn't maintain any order** of elements (not insertion order or sorting order).
- It allows **both homogeneous and heterogeneous** data types.
- Internally, it uses a **Hashtable data structure**.
- The **default capacity is 16 and load Fator or Fill Ratio is 0.75.**
- HashSet is **best used for fast searching operations**, because hashing is very efficient.

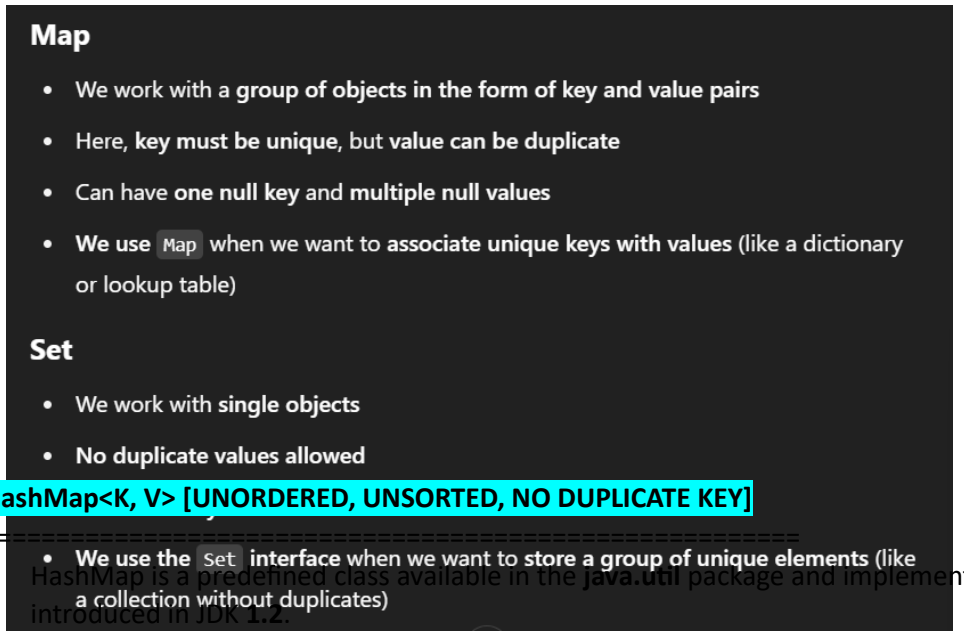
24- Map <Key, Value>

==========

- In Java, **Map** is an interface available in the **java.util** package, introduced in **JDK 1.2** as part of the **Java Collections Framework**.
- In the entire Collection Framework, we work with **single objects** and **groups of objects**:
- **Single:** To work with **single elements**, Java provides the **Collection** interface and its subtypes like List, Set, and Queue.
- **Group Of Object:** On the other hand, it provides the **Map** interface to work with a **group of objects in the form of key and value pairs**.
- The Map interface is used to represent a collection of **key-value pairs**, where **Keys must be unique** (no duplicates allowed), **Values can be duplicated**. **(Student Roll Numbers and Names)**

25- Different between Map and Set Interface?

- Both are part of the Collections Framework, available in java.util package.



26- HashMap<K, V> [UNORDERED, UNSORTED, NO DUPLICATE KEY]

==========

- HashMap is a predefined class available in the **java.util** package and implements the **Map** interface, introduced in JDK **1.2**.
- HashMap is an **unordered and unsorted** map means it does not guarantee any specific order of elements.

Collection Classes

- It works with **multiple objects** in the form of **key and value pairs**.
- It inserts elements based on the **hashCode()** of the key by using **hashing algorithm** and stores them in **buckets**.
- It **does not allow duplicate keys**, but **duplicate values are allowed**.
- To avoid duplicate keys and to manage keys properly, we must follow the **hashCode () == op and equals () method contract**.
- Default Capacity is 16 and load factor is 0.75 Recio.

27- How HashMap internally Work?

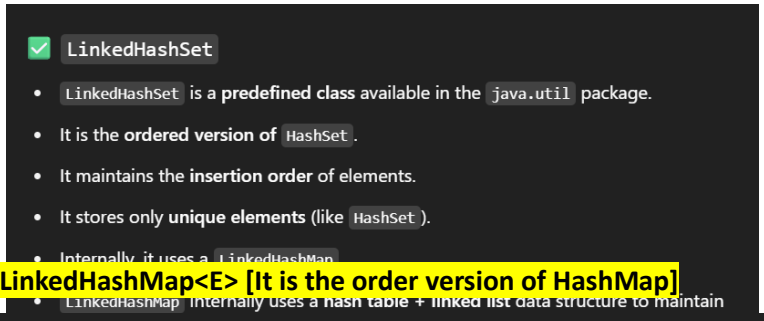
- While working with HashSet or HashMap, every key object must be checked because duplicate keys are not allowed.
- When we **add a new key**, internally it uses:
- **hashCode ()** method – to generate a hash code.
- **== operator** – to check if both objects are the same (reference).
- **equals ()** method – to check if both objects are same (content).
-
- **First**, it calls the **hashCode ()** method on the key to calculate the **hash value**.
- Based on the hash, it finds the **bucket index** (location) where the entry might be stored.
-
- Then it checks if any **existing key** is already present in that bucket:
- If **no key** is present → the new key-value pair is directly inserted.
- If **key is present**, it means **hash collision** has happened.
- When hash collision happens (same bucket), it does this:
- First it uses == operator to check if both keys are the same object (same memory).
- If yes → new value will **replace the old value**.
- If not same → it uses **equals(Object obj)** method.
- → keys are **same (duplicate)** → value is **updated**.
- If equals () return false → keys are different → new key-value is added in the same bucket using Singly LinkedList LinkedList.

28- equals () and hashCode () method contract:

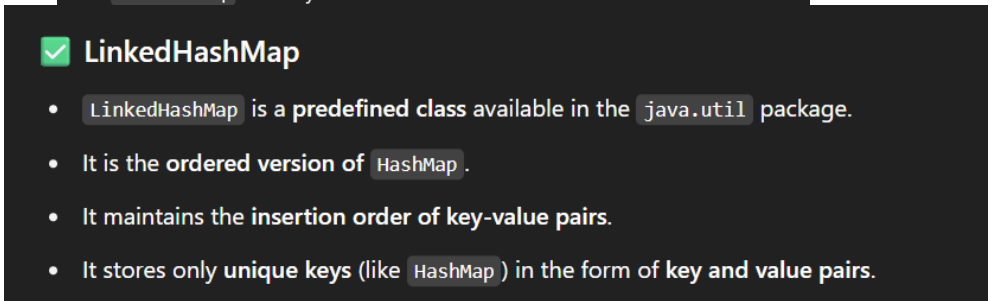
- Both the methods are working together to find out the duplicate objects in the Map.
- If equals () method invoked on two objects and it returns true then hashcode of both the objects must be same.
- IF TWO OBJECTS ARE HAVING SAME HASH CODE THEN IT MAY BE SAME OR DIFFERENT BUT IF EQUALS (OBJECT OBJ) METHOD RETURNS TRUE THEN BOTH OBJECTS MUST RETURN SAME HASHCODE.

29- LinkedHashSet<E> [It is the order version of HashSet]

==========



30- LinkedHashMap<E> [It is the order version of HashMap]

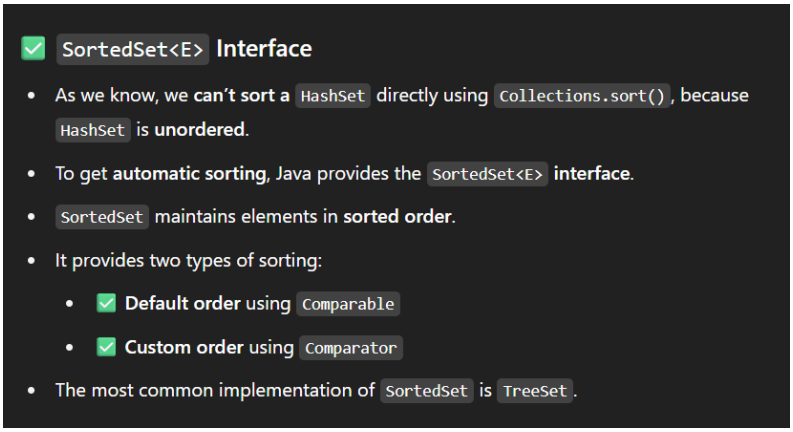


Collection Classes

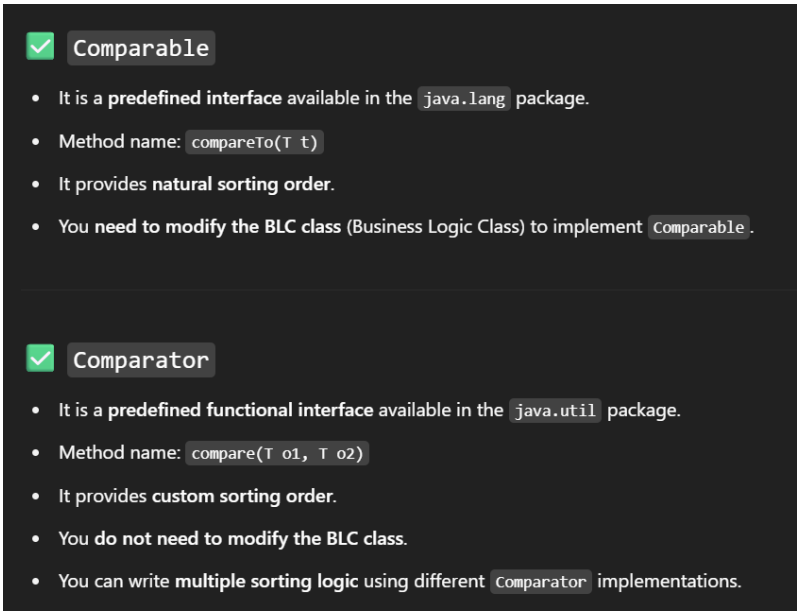
31- SortedSet<Set<E>

==========

-



32- Comparable/ Comparator



33- TreeSet<E>

- public class TreeSet<E> extends AbstractSet<E> implements NavigableSet<E>, Clonable, Serializable