**Date: 20-02-2023**

**Day 1. Interview Preparation.**

**Collections**

**-------------**

**1)What is the difference between Array and Collection classes?**

|  | **Arrays** | **Collection** |
| --- | --- | --- |
| **1** | **Arrays are fixed in size that is once we create an array we can not increased or decreased based on our requirement.** | **Collection are growable in nature that is based on our requirement. We can increase or decrease of size.** |
| **2** | **Write in memory Arrays are not recommended to use.** | **Write in  memory collection are recommended to use.** |
| **3** | **With respect to performance Arrays are recommended to use.** | **With respect to performance collection are not recommended to use.** |
| **4** | **Arrays can hold only homogeneous data types elements.** | **Collection can hold both homogeneous and heterogeneous elements.** |
| **5** | **There is no underlying data structure for arrays and hence ready made  method support is not available.** | **Every collection class is implemented based on some standard data structure and hence for every requirement ready made method support is available being a performance. we can use these method directly and We are not responsible to implement these methods.** |
| **6** | **Arrays can hold both object and primitive data type .** | **Collection can hold only object types but not primitive datatypes such as int, long, short, etc.** |

**2)Name the core Collection interfaces & their implementing Collection classes**

**3)What is the root interface in collection hierarchy?List its methods**

**4)Explain difference between**

**5)ArrayList & Vector**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| 1) ArrayList is **not synchronized**. | Vector is **synchronized**. |
| 2) ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity. | Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity. |
| 3) ArrayList is **not a legacy** class. It is introduced in JDK 1.2. | Vector is a **legacy** class. |
| 4) ArrayList is **fast** because it is non-synchronized. | Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object. |
| 5) ArrayList uses the **Iterator** interface to traverse the elements. | A Vector can use the **Iterator** interface or **Enumeration** interface to traverse the elements. |

**6)ArrayList & LinkedList**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| 5) The memory location for the elements of an ArrayList is contiguous. | The location for the elements of a linked list is not contagious. |
| 6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList. | There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized. |
| 7) To be precise, an ArrayList is a resizable array. | LinkedList implements the doubly linked list of the list interface. |

**7)HashSet & TreeSet**

|  |  |  |
| --- | --- | --- |
| **Properties** | **HashSet** | **TreeSet** |
| **Implementation** | It internally uses HashMap to store the elements. | Internally it uses TreeMap to store the elements. |
| **Data** **Structure** | The Data structure that HashSet uses is HashTable. | The Data Structure that TreeSet uses is a Red-Black Tree. |
| **Time** **Complexity** | To add or remove the element from HashSet, the time complexity is O(1). | For adding or removing the element from TreeSet, the time complexity is O(log(n)). |
| **Null** **Values** | Only one null element can be stored in the HashSet. | No null elements are allowed. |
| **Comparison** | The hashCode() or equals() method is used to compare the elements it uses. | The compare() and compareTo() method is used for comparison. |
| **Sorted** **values** | There is no guarantee that the elements will be stored in sorted order. | TreeSet maintains the Sorted order. |
| **Performance** | It is faster than TreeSet. | TreeSet is slower than HashSet. |
| **Values to Store** | HashSet can store Heterogenous values. | You can store only homogenous values in TreeSet. |
| **Methods** | In comparison to TreeSet, HashSet offers fewer methods. | TreeSet has more methods in comparison to HashSet. |
| **Iteration order** | To iterate HashSet elements, it uses the arbitrary method. | It has sorted values. |

**8)HashSet & HashMap**

| **Sr. No.** | **Key** | **HashMap** | **HashSet** |
| --- | --- | --- | --- |
| 1 | Implementation | Hashmap is the implementation of Map interface. | Hashset on other hand is the implementation of set interface. |
| 2 | Internal implementation | Hashmap internally do not implements hashset or any set for its implementation. | Hashset internally uses Hashmap for its implementation. |
| 3 | Storage of elements | HashMap Stores elements in form of key-value pair i.e each element has its corresponding key which is required for its retrieval during iteration. | HashSet stores only objects no such key value pairs maintained. |
| 4 | Method to add element | Put method of hash map is used to add element in hashmap. | On other hand add method of hashset is used to add element in hashset. |
| 5 | Index performance | Hashmap due to its unique key is faster in retrieval of element during its iteration. | HashSet is completely based on object so compared to hashmap is slower. |
| 6 | Null Allowed | Single null key and any number of null value can be inserted in hashmap without any restriction. | On other hand Hashset allows only one null value in its collection,after which no null value is allowed to be added. |

**9)Hashtable & HashMap**

| **S. No.** | **Hashmap** | **Hashtable** |
| --- | --- | --- |
| 1. | No method is synchronized. | Every method is synchronized. |
| 2. | Multiple threads can operate simultaneously and hence hashmap’s object is not thread-safe. | At a time only one thread is allowed to operate the Hashtable’s object. Hence it is thread-safe. |
| 3. | Threads are not required to wait and hence relatively performance is high. | It increases the waiting time of the thread and hence performance is low. |
| 4. | Null is allowed for both key and value. | Null is not allowed for both key and value. Otherwise, we will get a null pointer exception. |
| 5. | It is introduced in the 1.2 version. | It is introduced in the 1.0 version. |
| 6. | It is non-legacy. | It is a legacy. |

**10)HashMap & TreeMap**

| **Basis of comparison** | **HashMap** | **TreeMap** |
| --- | --- | --- |
| Basic | HashMap does not keep track of the order of insertions. | TreeMap preserves insertion order. |
| Interface Implements | Map, Cloneable, and Serializable interfaces are all ones that are implemented by HashMap. | TreeMap is capable of being Cloned and Serialized, in addition to implementing the NavigableMap interface. |
| Data Structure | A Hash Table serves as the foundation for HashMap's underlying data structure. | The Red-Black Tree is the foundational data structure that TreeMap is built upon. |
| Null Keys and Values | The Null key can be used once in HashMap, and the Null value can be used any number of times. | TreeMap does not let the use of a null key, but it does permit the use of a null value any number of times. |
| Extends and implements | The HashMap class is an extension of the AbstractMap class and an implementation of the Map interface. | The TreeMap class extends the AbstractMap base class and implements the SortedMap and NavigableMap interfaces respectively. |
| Performance | HashMap processes operations more quickly. | When compared to HashMap, the operation speed of TreeMap is lower. |
| Order of elements | HashMap does not keep track of order. | The elements are arranged in their natural order (ascending). |
| Homogeneous/ Heterogeneous | HashMap supports heterogeneous elements because it does not perform key sorting. | Because of sorting, TreeMap allows homogeneous values as keys. |
| Uses | When we do not want key-value pairs to be in sorted order, the HashMap data structure should be utilised. | When we need the key-value pair to be in sorted (ascending) order, we should utilise the TreeMap. |

**11)Iterator &ListIterator**

| **Iterator** | **ListIterator** |
| --- | --- |
| Can traverse elements present in Collection only in the forward direction. | Can traverse elements present in Collection both in forward and backward directions. |
| Helps to traverse Map, List and Set. | Can only traverse List and not the other two. |
| Indexes cannot be obtained by using Iterator. | It has methods like nextIndex() and previousIndex() to obtain indexes of elements at any time while traversing List. |
| Cannot modify or replace elements present in Collection | We can modify or replace elements with the help of set(E e) |
| Cannot add elements and it throws ConcurrentModificationException. | Can easily add elements to a collection at any time. |
| Certain methods of Iterator are next(), remove() and hasNext(). | Certain methods of ListIterator are next(), previous(), hasNext(), hasPrevious(), add(E e). |

**12)Iterator & Enumeration**

| **Iterator** | **Enumeration** |
| --- | --- |
| Iterator is a universal cursor as it is applicable for all the collection classes. | Enumeration is not a universal cursor as it applies only to legacy classes. |
| Iterator has the remove() method. | Enumeration does not have the remove() method. |
| Iterator can do modifications (e.g using remove() method it removes the element from the Collection during traversal). | Enumeration interface acts as a read only interface, one can not do any modifications to Collection while traversing the elements of the Collection. |
| Iterator is not a legacy interface. Iterator can be used for the traversal of HashMap, LinkedList, ArrayList, HashSet, TreeMap, TreeSet . | Enumeration is a legacy interface which is used for traversing Vector, Hashtable. |

**13)Set & List**

|  |  |  |
| --- | --- | --- |
| **S.No** | **List** | **Set** |
| 1. | The list implementation allows us to add the same or duplicate elements. | The set implementation doesn't allow us to add the same or duplicate elements. |
| 2. | The insertion order is maintained by the List. | It doesn't maintain the insertion order of elements. |
| 3. | List allows us to add any number of null values. | Set allows us to add at least one null value in it. |
| 4. | The List implementation classes are LinkedList and ArrayList. | The Set implementation classes are TreeSet, HashSet and LinkedHashSet. |
| 5. | We can get the element of a specified index from the list using the get() method. | We cannot find the element from the Set based on the index because it doesn't provide any get method(). |
| 6. | It is used when we want to frequently access the elements by using the index. | It is used when we want to design a collection of distinct elements. |
| 7. | The method of List interface listiterator() is used to iterate the List elements. | The iterator is used when we need to iterate the Set elements. |

**14)Set & Map**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Set** | **Map** |
| 1. | Set is used to construct the mathematical Set in Java. | Map is used to do mapping in the database. |
| 2. | It cannot contain repeated values. | It can have the same value for different keys. |
| 3. | Set doesn't allow us to add the same elements in it. Each class that implements the Set interface contains only the unique value. | Map contains unique key and repeated values. In Map, one or more keys can have the same values, but two keys cannot be the same. |
| 4. | We can easily iterate the Set elements using the keyset() and the entryset() method of it. | Map elements cannot be iterated. We need to convert Map into Set for iterating the elements. |
| 5. | Insertion order is not maintained by the Set interface. However, some of its classes, like LinkedHashSet, maintains the insertion order. | The insertion order is also not maintained by the Map. However, some of the Map classes like TreeMap and LinkedHashMap does the same. |

**15)Queue and Stack**

| **Stacks** | **Queues** |
| --- | --- |
| Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list. | Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list. |
| Insertion and deletion in stacks takes place only from one end of the list called the top. | Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list. |
| Insert operation is called push operation. | Insert operation is called enqueue operation. |
| Delete operation is called pop operation. | Delete operation is called dequeue operation. |
| In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list. | In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element. |
| Stack is used in solving problems works on recursion. | Queue is used in solving problems having sequential processing. |
| Stack does not have any types. | Queue is of three types – 1. Circular Queue 2. Priority queue 3. double-ended queue. |
| Can be considered as a vertical collection visual. | Can be considered as a horizontal collection visual |

**16)Queue & List**

|  |  |
| --- | --- |
| **Static Queue** | **Singly Linked List** |
| **Queue is a collection of one or more elements arranged in memory in a contiguous fashion.** | **A linked list is a collection of one or more elements arranged in memory in a dis-contiguous fashion.** |
| **Static Queue is always fixed size.** | **List size is never fixed.** |
| **In Queue, only one and single type of information is stored because static Queue implementation is through Array.** | **List also stored the address for the next node along with it’s content.** |
| **Static Queue is index based.** | **Singly linked list is reference based.** |
| **Insertion can always be performed on a single end called *REAR* and deletion on the other end called *FRONT*.** | **Insertion as well as deletion can performed any where within the list.** |
| **Queue is always based on FIFO.** | **List may be based on FIFI or LIFO etc.** |
| **Queue have two pointer FRONT and REAR.** | **While List has only one pointer basically called HEAD.** |

**17)Collection and Collections**

| **Collection** | **Collections** |
| --- | --- |
| It is an interface. | It is a utility class. |
| It is used to represent a group of individual objects as a single unit. | It defines several utility methods that are used to operate on collection. |
| The Collection is an interface that contains a static method since java8. The Interface can also contain abstract and default methods. | It contains only static methods. |

**18)Comparable and Comparator**

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price. | The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc. |
| 2) Comparable **affects the original class**, i.e., the actual class is modified. | Comparator **doesn't affect the original class**, i.e., the actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is present in **java.lang** package. | A Comparator is present in the **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. |

**19)What is the difference between Array and ArrayList?**

|  |  |  |
| --- | --- | --- |
| **Basis** | **Array** | **ArrayList** |
| **Definition** | An **array** is a dynamically-created object. It serves as a container that holds the constant number of values of the same type. It has a contiguous memory location. | The **ArrayList** is a class of Java **Collections** framework. It contains popular classes like **Vector, HashTable**, and **HashMap**. |
| **Static/ Dynamic** | Array is **static** in size. | ArrayList is **dynamic** in size. |
| **Resizable** | An array is a **fixed-length** data structure. | ArrayList is a **variable-length** data structure. It can be resized itself when needed. |
| **Initialization** | It is mandatory to provide the size of an array while initializing it directly or indirectly. | We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size. |
| **Performance** | It performs **fast** in comparison to ArrayList because of fixed size. | ArrayList is internally backed by the array in Java. The resize operation in ArrayList slows down the performance. |
| **Primitive/ Generic type** | An array can store both **objects** and **primitives** type. | We cannot store **primitive** type in ArrayList. It automatically converts primitive type to object. |
| **Iterating Values** | We use **for** loop or **for each** loop to iterate over an array. | We use an **iterator** to iterate over ArrayList. |
| **Type-Safety** | We cannot use generics along with array because it is not a convertible type of array. | ArrayList allows us to store only **generic/ type, that's why it is type-safe.** |
| **Length** | Array provides a **length** variable which denotes the length of an array. | ArrayList provides the **size()** method to determine the size of ArrayList. |
| **Adding Elements** | We can add elements in an array by using the **assignment**operator. | Java provides the **add()** method to add elements in the ArrayList. |
| **Single/ Multi-Dimensional** | Array can be **multi-dimensional**. | ArrayList is always **single-dimensional**. |

**20)Why we override equals() & hashcode() method**

**21)How to synchronize List, Set and Map elements**

**22)What is the advantage of generic collection**

**23)How to convert ArrayList to Array and Array to ArrayList?**

**24)How to reverse ArrayList?**

**25)How to sort ArrayList?**

**26)Which collection classes are synchronized or thread-safe**

**27)What is the difference between peek(),poll() and remove() method of the Queue interface ?**

**28)Write java code showing insertion,deletion and retrieval of HashMap object ?**

**29)What is the difference between HashMap and ConcurrentHashMap ?**

**30) Why Map interface does not extend the Collection interface in Java Collections Framework ?**

**31)Write the code for iterating the list in different ways in java ?**

**32)Suppose there is an Employee class. We add Employee class objects to the ArrayList.**

**33)Mention the steps need to be taken , if I want to sort the objects in ArrayList using the**

**employeeId attribute present in Employee class.**

**34)How can an ArrayList be synchronized without using Vector?**

**35)What is NavigableMap in Java? What is a benefit over Map?**

**36)What is Java Priority Queue ?**

**Coding Questions**

**-----------------**

**1. Check the given number is EVEN or ODD.**

**2. Write a Java Program to find the Factorialof given number.**

**3. Find the Factorialof a number using Recursion.**

**4. Swap two numberswithout using third variable approach 1.**

**5. Swap two numberswithout using third variable approach 2.**

**6. Swap two numberswithout using third variable approach 3.**

**7. How to check the given number is Positive or Negativein Java?**

**8. Write a Java Program to find whether given number is Leap year or NOT?**

**10. Write a Java Program to print the digits of a Given Number.**

**11. Write a Java Program to print all the Factors of the Given number.**

**12. Write a Java Program to find sum of the digits of a given number.**

**13. Write a Java Program to find the smallest of 3 numbers(a,b,c) without using < or > symbol?**

**14. How to add two numbers without using the arithmetic operators in Java?**

**15. Write a java program to Reverse a given number.**

**16. Write a Java Program to find GCD of two given numbers.**

**17. Write a java program to LCM of TWO given number.**

**18. Write a java program to LCM of TWO given numberusing Prime Factors method.**

**19. Check whether the Given Numberis a Palindrome or NOT.**

**20. Write a Java Program to print all the Prime Factorsof the Given Number.**

**21. Write a Java Program to check whether the Given Number is Prime Number or NOT.**

**22. Write a Java Program to print Prime Numbers from 1 to N.**

**23. Write a Java Program to check whether the given number is Armstrong Numberor NOT.**

**24. Write a Java Program to print Armstrong Numbersbetween 1 to 1000.**

**25. Write a Java Program to check whether the given number is Perfect Numberor NOT.**

**26. Write a Java Program to print Perfect Numbersbetween 1 to 1000.**