22/11/2024

VIGNESH KUMAR S

TESTING

Java Questions

1.WHAT IS DIFFERNCE BETWEEN COLLECTION CLASS AND COLLECTION INTERFACE:

- **Collection Interface**: It is a part of Java's Collection Framework. It defines the basic operations for handling groups of objects like adding, removing, checking if an item exists, and retrieving the size of the collection. However, it doesn't provide implementations for these operations.
- **Collection Class**: These are concrete classes that implement the Collection interface and provide actual functionality for storing and manipulating data. For example, ArrayList, HashSet, and LinkedList are classes that implement the Collection interface.

Collection class (ArrayList).Programming

```
import java.util.*;

public class CollectionExample {

 public static void main(String[] args) {

Collection<String> fruits = new ArrayList<>();

fruits.add("Apple");

fruits.add("Banana");

fruits.add("Orange");

System.out.println("Size of collection: " + fruits.size());
System.out.println("Does the collection contain 'Apple'? " +
fruits.contains("Apple"));

fruits.remove("Banana");

System.out.println("Fruits after removal: " + fruits);

}

 }
```

Output:

```
Size of collection: 3

Does the collection contain 'Apple'? true

Fruits after removal: [Apple, Orange]
```

2. 15 interfaces in java

Comparable<T>

- **Description**: This interface is used to define a natural ordering of objects. It allows objects of a class to be compared with each other.
- **Methods**:
  - int compareTo(To)

Iterable<T>

- **Description**: Represents a collection of elements that can be iterated using an iterator.
- **Methods**:
  - Iterator<T> iterator ()

Collection<E>

- **Description**: It is the root interface in the collection hierarchy. It defines common operations like add, remove, contains, and size.
- **Methods**:
  - boolean add(E e)
  - boolean remove(Object o)
  - int size()

Queue<E>

- **Description**: Represents a collection designed for holding elements before processing. It follows the FIFO (First-In-First-Out) principle.
- **Methods**:
  - boolean offer(E e)
  - E poll()
  - E peek()

Deque<E>

- **Description**: A subtype of Queue, it allows elements to be added or removed from both ends.
- **Methods**:
  - void addFirst(E e)
  - void addLast(E e)
  - E removeFirst()
  - E removeLast()

Map<K, V>

- **Description**: Represents a collection of key-value pairs. It is not a subtype of Collection.
- **Methods**:
  - V put(K key, V value)

    - V get(Object key)
    - Set<Map.Entry<K, V>> entrySet()

SortedSet<E>

- **Description**: A Set that maintains its elements in ascending order.
- **Methods**:
    - E first()
    - E last()
    - SortedSet<E> subSet(E fromElement, E toElement)

Iterator<E>

- **Description**: Provides methods to iterate over elements of a collection.
- **Methods**:
    - boolean hasNext()
    - E next()
    - void remove()

Runnable

- **Description**: This interface is used to represent a task that can be executed by a thread.
- **Methods**:
    - void run()

Event Listener

- **Description**: This is a marker interface for all event listener interfaces in Java. Event-driven programming (e.g., GUI) uses this interface.
- **No methods** (it is a marker interface).

3.THROW AND THROWS

throw in Java

- **Definition**: The throw keyword is used to **explicitly throw an exception** from within a method or a block of code. It allows you to manually create and throw an exception (either predefined or custom).
- **Usage**: You can throw an exception using throw followed by an instance of an exception.

throws in Java

- **Definition**: The throws keyword is used in a **method signature** to indicate that the method **may throw one or more exceptions**. It tells the calling code that it needs to handle the specified exceptions, either by using try-catch blocks or by propagating the exception further using throws.
- **Usage**: You use throws in the method signature to declare exceptions that might be thrown during the method's execution.

4.Hashset , Hashmap, Treeset :

HashSet is a **collection** class in Java that implements the **Set** interface. It is part of the Java Collections Framework and is defined in the java.util package. A HashSet is used to store **unique elements** (no duplicates) and does not maintain the order of elements. It is backed by a hash table, which means it offers **constant-time** performance for basic operations like add(), remove(), and contains().

A HashMap in Java is a part of the java.util package and is a collection class that implements the Map interface. It stores key-value pairs where each key is unique, and each key maps to exactly one value.

import java.util.HashMap;

```
public class HashMapExample {
  public static void main(String[] args) {
          HashMap<String, Integer> map = new HashMap<>();


      // Adding key-value pairs to the map
      map.put("Apple", 10);    // Key: "Apple", Value: 10
      map.put("Banana", 20);   // Key: "Banana", Value: 20
      map.put("Orange", 30);   // Key: "Orange", Value: 30
      map.put("Mango", 40);    // Key: "Mango", Value: 40


      System.out.println("HashMap: " + map);
      System.out.println("Value for key 'Banana': " + map.get("Banana"));



      if (map.containsKey("Apple")) {
 System.out.println(Apple exists in the map.");
      }
      map.remove("Mango");
      System.out.println("Updated HashMap after removing Mango: " + map);
```

```
        System.out.println("Iterating over HashMap:");
        for (String key : map.keySet()) {
            System.out.println(key + ": " + map.get(key));
        }
    }
}
```

Tree Map

A TreeMap in Java is a part of the java.util package and implements the Map interface. It is a **sorted** map that maintains its keys in ascending order, based on their natural ordering or by a custom comparator provided during its construction. Unlike a HashMap, which does not maintain any order of its elements, a TreeMap guarantees that its keys will be in a sorted order (by default in ascending order). The values in a TreeMap can be accessed via their associated keys, just like in any other map.

Vector:

In Java, a Vector is a growable array of objects. It is part of the java.util package and implements the List interface. A Vector behaves similarly to an ArrayList but with a few key differences. It is a **dynamic array** that automatically expands as elements are added, and it is **synchronized**, making it thread-safe (which can lead to performance overhead in single-threaded environments). A Vector can hold any type of object, and it allows for accessing elements via indices.

Why interface start with i and end with able ?

Why the Change in Naming Convention?

- **Simplicity and Readability**: Java encourages a simpler, more intuitive approach. Interfaces that define capabilities or behaviors are named directly (e.g., Runnable, Serializable, Comparable) rather than being prefixed with "I". This makes the code cleaner and easier to read.
- **No Need for the "I" Prefix**: The Java community has moved towards naming interfaces without the "I" prefix, as the context of an interface is usually clear through the usage and implementation patterns.

In modern Java, it's better to follow **conventions like using descriptive names** without the "I" prefix, but using the "able" suffix when the interface represents a capability or functionality. For example:

- Serializable instead of ISerializable
- Cloneable instead of ICloneable