Tanvir s

Collection Framework | ArrayList, LinkedList & Collection Interface

What is framework in Java?

A framework is a popular and ready-made architecture that contains a set of classes and interfaces.

What are the benefits of the Collection Framework in Java?

The benefits of Collection Framework in Java are:

- Java collection framework offers highly efficient and effective data structures that enhance the accuracy and speed of the program.
- The program developed with the Java collection framework is easy to maintain.
- A developer can mix classes with other types that result in increasing the reusability of code.
- The Java collection framework enables programmers to modify the primitive collection types the way they like.

Explain Collections Class

java.util.Collections is a class consists of static methods that operate on collections. It contains polymorphic algorithms to operate on collections, “wrappers”. This class contains methods for algorithms, like binary sorting, search, shuffling, etc.

Collection vs Collections

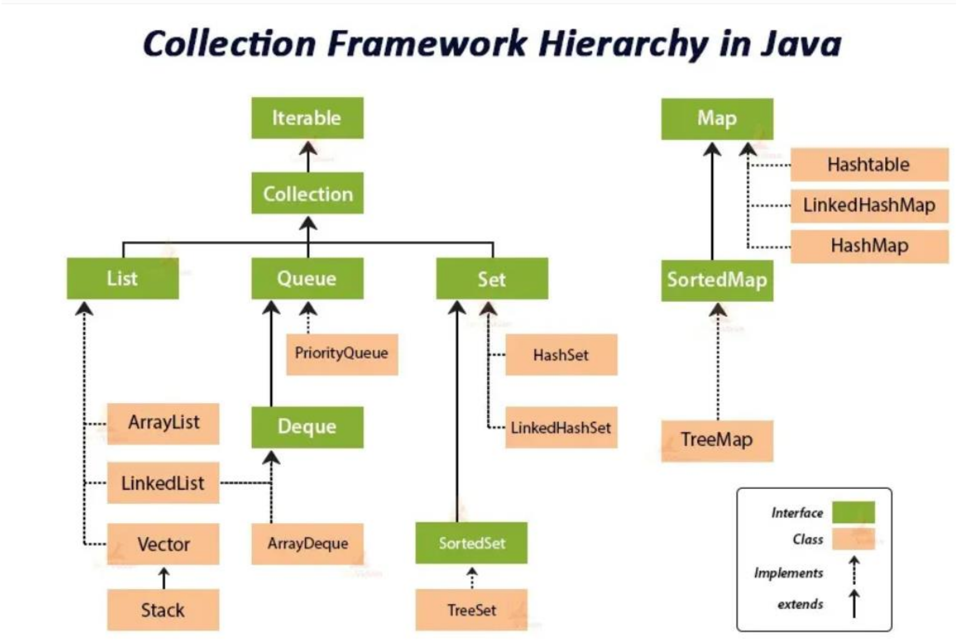
- Collection is an interface that represents group of Objects
- Collections is an utility class that provides method to operate collection type(like searching ,sorting, modifying)

Explain the basic interfaces of the Java collections framework

- Collection interface is root interface of java collections framework which represents a group of objects know as elements
- Collection interface is a part of java.util package

Methods in collection interface- add(),remove, contains(check elements),size(),clear(remove all elements), isEmpty(check empty or not),add All(we merge two list),remove all(intersection),clear(list empty) ,toarray(object to array)

- **Set:** Set is a collection having no duplicate elements. It uses hashtable for storing elements.
- **List:** List is an ordered collection that can contain duplicate elements. It enables developers to access any elements from its inbox. The list is like an array having a dynamic length.
- **MAP:** It is an object which maps keys to values. It cannot contain duplicate keys. Each key can be mapped to at least one value.



Explain the best practices in Java Collection Framework

The best practices in Java Collection Framework are:

1. **Choose the Correct Type of Collection**: Pick collections based on the specific need—List for ordered elements, Set for unique items, Map for key-value pairs, and Queue for FIFO processing.
2. **Write Programs in Terms of Interfaces**: Code to interfaces (e.g., List, Set) instead of concrete classes to make future implementation changes easier without affecting the rest of the code.
3. **Use Generics for Type Safety**: Generics provide compile-time type checking, preventing ClassCastException and making your code type-safe and more reliable.
4. **Use Immutable Classes**: Prefer immutable classes like String and Integer to ensure thread safety and avoid bugs; avoid custom equals() and hashCode() unless necessary.
5. **Use the Collections Utility Class**: Use Collections for common operations like sorting, searching, and creating synchronized/empty collections, reducing boilerplate and improving reusability.

- **List-> A List in Java is an ordered collection that allows duplicate elements and provides methods to manipulate elements, such as adding, removing, and accessing items by index**
    - ordered collection (like dynamic array)
    - Allow duplicates
    - List is Interface and it extends to collection
    - Array List ,Linked list, vector, Stack

ArrayList, Linked list, Vector, Stack

What are the various ways to iterate over a list?

Java collection Framework programmer can iterate over a list in two ways:

1) Using iterator, and 2) using it for each loop.

What is CopyOnWriteArrayList?

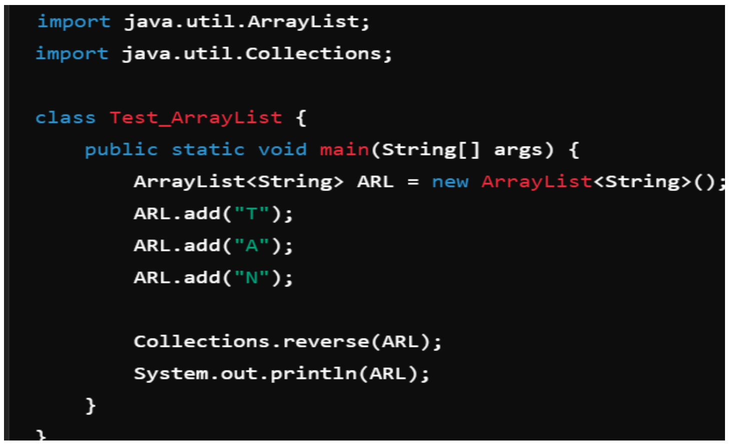
CopyOnWriteArrayList is a variant of ArrayList in which operations like add and set are implemented by creating a copy of the array. It is a thread-safe, and thereby it does not throw ConcurrentModificationException. This ArrayLists permits all the elements, including null.

What is ArrayList in Java?

Resizable array (automatically adjust their capacity), allow duplicates, random access, Array List can store **null** values. Due to adding removing TC is O(n)

Array List <String, Integer> AL=new Array List<>();

Internal working in ArrayList (new size=oldsize*3/2+1) e.g 10->increase by 6 ->10*3->30/2=15+1=16



```
import java.util.ArrayList;
import java.util.Collections;

class Test_ArrayList {
    public static void main(String[] args) {
        ArrayList<String> ARL = new ArrayList<String>();
        ARL.add("T");
        ARL.add("A");
        ARL.add("N");

        Collections.reverse(ARL);
        System.out.println(ARL);
    }
}
```

//

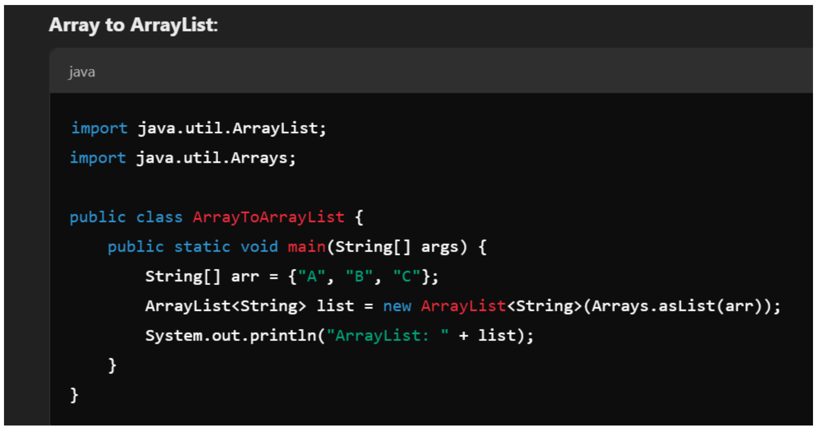
Explain the method to convert ArrayList to Array and Array to ArrayList

Programmers can convert an Array to ArrayList using asList() method of Arrays class. It is a static method of Arrays class that accept the List object. The syntax of asList() method is:

Arrays.asList(item)

Java programmers can convert ArrayList to the List object using syntax:

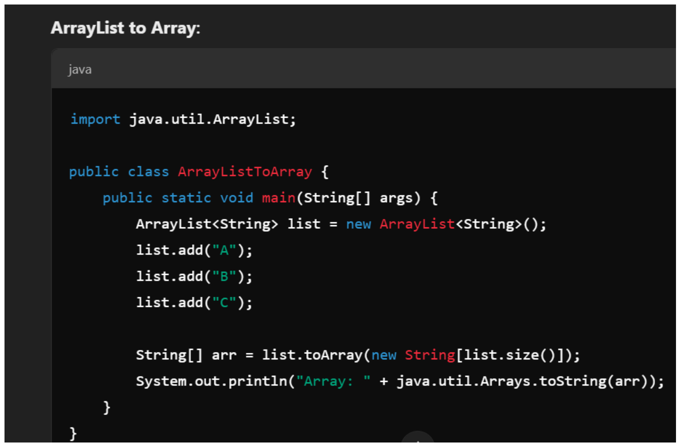
List_object.toArray(new String[List_object.size()])



Array to ArrayList:

java

```java
import java.util.ArrayList;
import java.util.Arrays;

public class ArrayToArrayList {
    public static void main(String[] args) {
        String[] arr = {"A", "B", "C"};
        ArrayList<String> list = new ArrayList<String>(Arrays.asList(arr));
        System.out.println("ArrayList: " + list);
    }
}
```



ArrayList to Array:

java

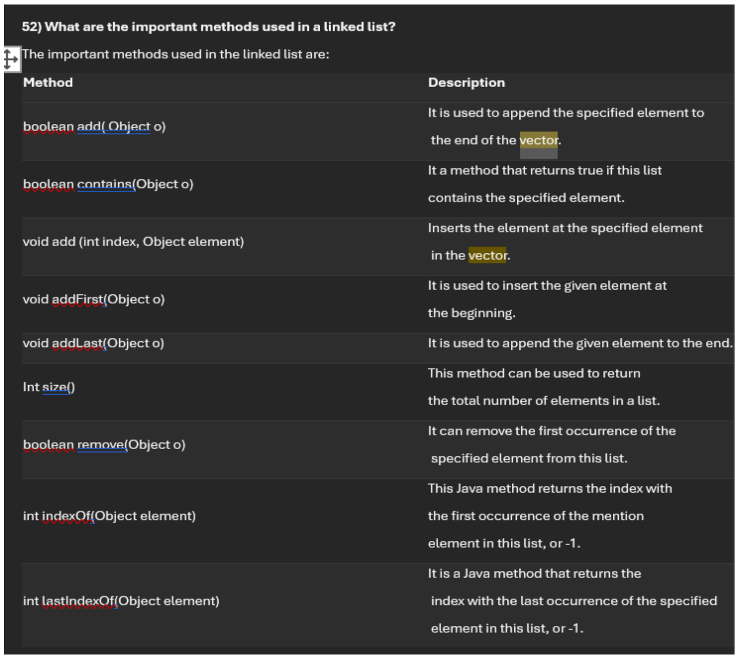
```java
import java.util.ArrayList;

public class ArrayListToArray {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();
        list.add("A");
        list.add("B");
        list.add("C");

        String[] arr = list.toArray(new String[list.size()]);
        System.out.println("Array: " + java.util.Arrays.toString(arr));
    }
}
```

Tanvir s

Linked List:- Linked list is class implementation of List and Deque interface its doubly linked list
List<> ls=new Linkedlist<>();

- Linked list store elements in separate containers called nodes where each contain element and like to next node
- No need to resize like Array list
- Faster when adding and removing elements

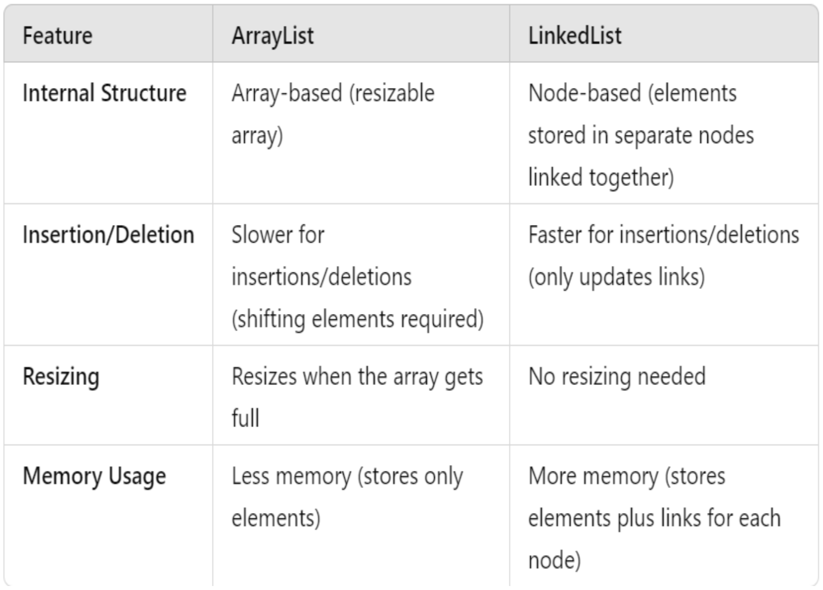
Same method and programmatic working as Array List



52) What are the important methods used in a linked list?

The important methods used in the linked list are:

Method	Description
boolean add(Object o)	It is used to append the specified element to the end of the vector.
boolean contains(Object o)	It a method that returns true if this list contains the specified element.
void add (int index, Object element)	Inserts the element at the specified element in the vector.
void addFirst(Object o)	It is used to insert the given element at the beginning.
void addLast(Object o)	It is used to append the given element to the end.
Int size()	This method can be used to return the total number of elements in a list.
boolean remove(Object o)	It can remove the first occurrence of the specified element from this list.
int indexOf(Object element)	This Java method returns the index with the first occurrence of the mention element in this list, or -1.
int lastIndexOf(Object element)	It is a Java method that returns the index with the last occurrence of the specified element in this list, or -1.



Feature	ArrayList	LinkedList
Internal Structure	Array-based (resizable array)	Node-based (elements stored in separate nodes linked together)
Insertion/Deletion	Slower for insertions/deletions (shifting elements required)	Faster for insertions/deletions (only updates links)
Resizing	Resizes when the array gets full	No resizing needed
Memory Usage	Less memory (stores only elements)	More memory (stores elements plus links for each node)

///

Explain linked list supported by Java

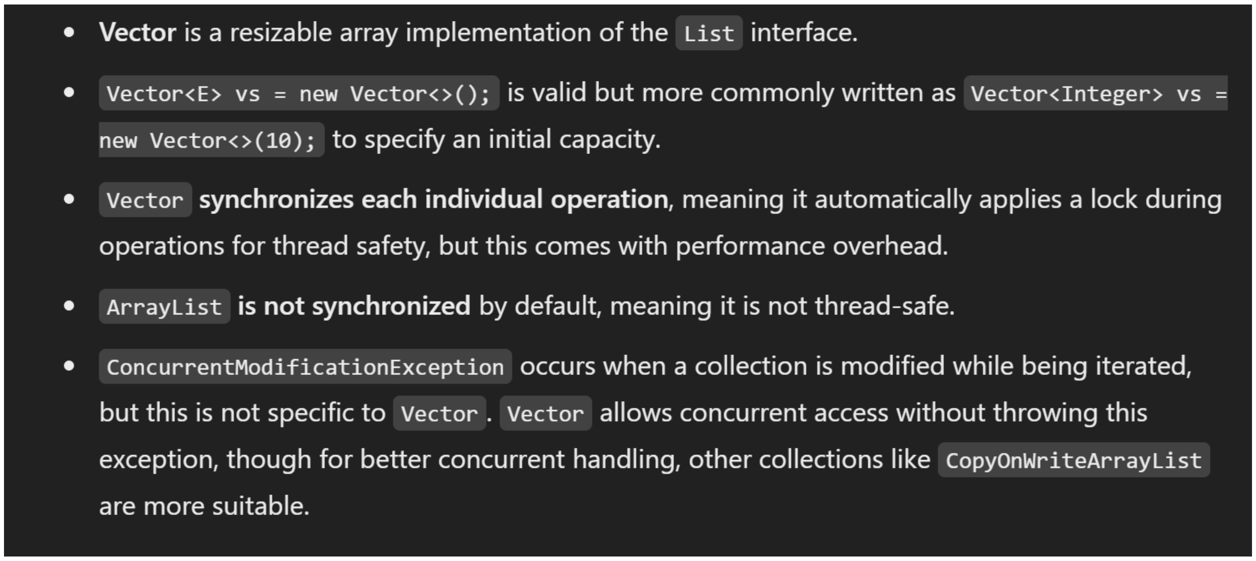
Two types of linked list supported by Java are:

- **Singly Linked list:** Singly Linked list is a type of data structure. In a singly linked list, each node in the list stores the contents of the node and a reference or pointer to the next node in the list. It does not store any reference or pointer to the previous node.
- **Doubly linked lists:** Doubly linked lists are a special type of linked list wherein traversal across the data elements can be done in both directions. This is made possible by having two links in every node, one that links to the next node and another one that connects to the previous node.

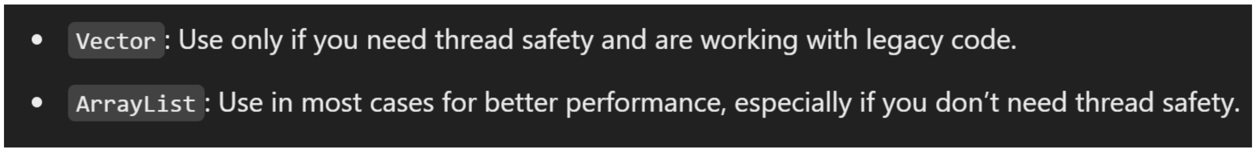
--

Tanvir s

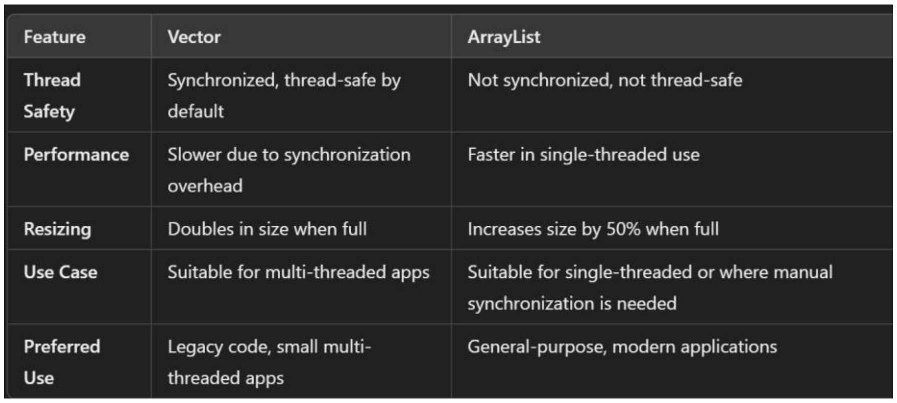
Vector: -



- **Vector** is a resizable array implementation of the `List` interface.
- `Vector<E> vs = new Vector<>();` is valid but more commonly written as `Vector<Integer> vs = new Vector<>(10);` to specify an initial capacity.
- `Vector` **synchronizes each individual operation**, meaning it automatically applies a lock during operations for thread safety, but this comes with performance overhead.
- `ArrayList` **is not synchronized** by default, meaning it is not thread-safe.
- `ConcurrentModificationException` occurs when a collection is modified while being iterated, but this is not specific to `Vector`. `Vector` allows concurrent access without throwing this exception, though for better concurrent handling, other collections like `CopyOnWriteArrayList` are more suitable.



- `Vector`: Use only if you need thread safety and are working with legacy code.
- `ArrayList`: Use in most cases for better performance, especially if you don’t need thread safety.



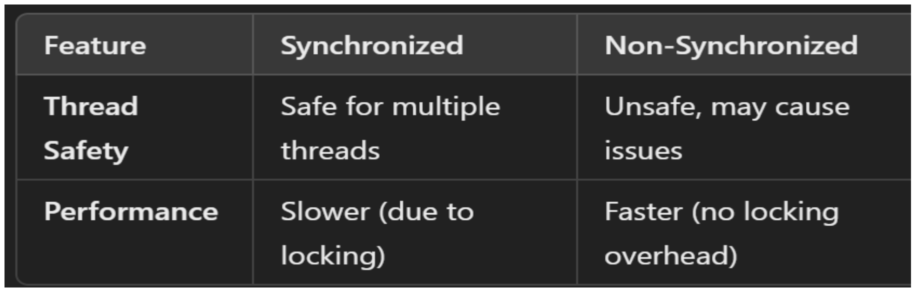
Feature	Vector	ArrayList
Thread Safety	Synchronized, thread-safe by default	Not synchronized, not thread-safe
Performance	Slower due to synchronization overhead	Faster in single-threaded use
Resizing	Doubles in size when full	Increases size by 50% when full
Use Case	Suitable for multi-threaded apps	Suitable for single-threaded or where manual synchronization is needed
Preferred Use	Legacy code, small multi-threaded apps	General-purpose, modern applications

----

Synchronized vs Non-Synchronized

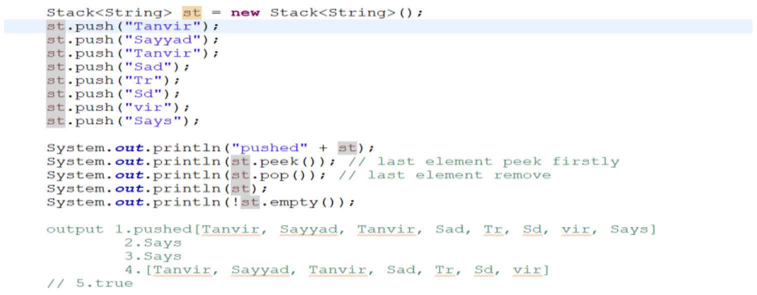
only one thread can access a particular section of code or data at a time. It is used to prevent issues when multiple threads try to modify data simultaneously, which could lead to data corruption or unexpected behavior.

Non-synchronization allows multiple threads to access and modify shared resources simultaneously, which can lead to errors.



Feature	Synchronized	Non-Synchronized
Thread Safety	Safe for multiple threads	Unsafe, may cause issues
Performance	Slower (due to locking)	Faster (no locking overhead)

Stack: **- stack is subclass of vector that implements a LIFO data Structure .it provides method push(),pop(),peek(),empty()**



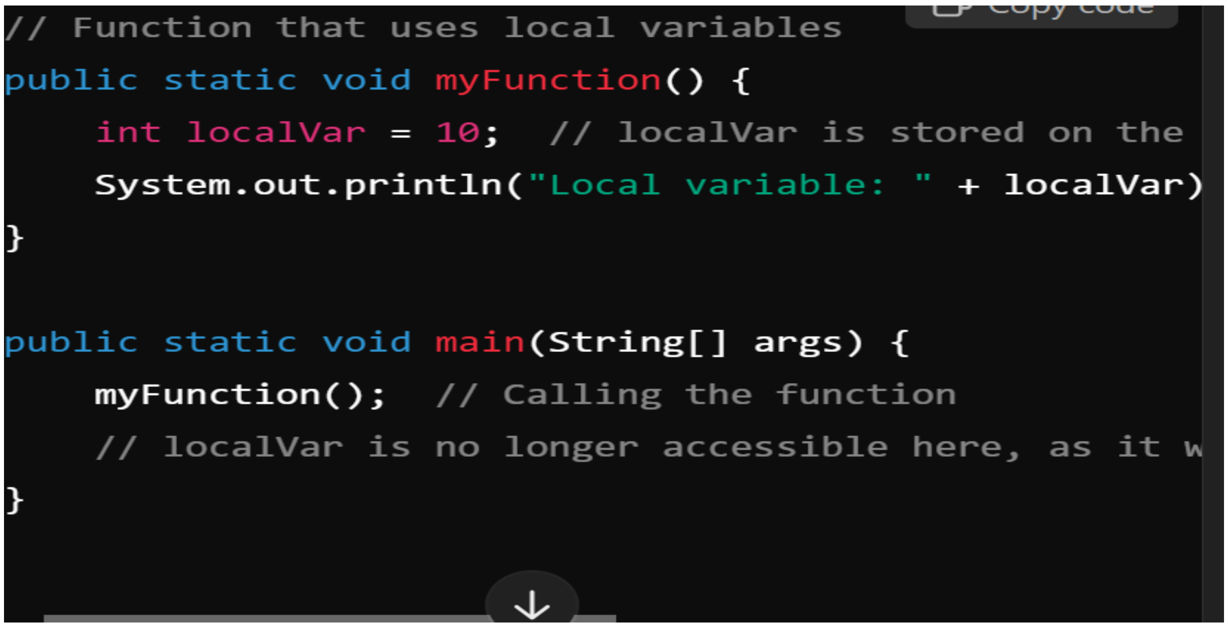
```
Stack<String> st = new Stack<String>();
st.push("Tanvir");
st.push("Sayyad");
st.push("Tanvir");
st.push("Sad");
st.push("Tr");
st.push("Sd");
st.push("vir");
st.push("Says");

System.out.println("pushed" + st);
System.out.println(st.peek()); // last element peek firstly
System.out.println(st.pop()); // last element remove
System.out.println(st);
System.out.println(!st.empty());

output 1.pushed[Tanvir, Sayyad, Tanvir, Sad, Tr, Sd, vir, Says]
        2.Says
        3.Says
        4.[Tanvir, Sayyad, Tanvir, Sad, Tr, Sd, vir]
// 5.true
```

//

What are the advantages of the stack?

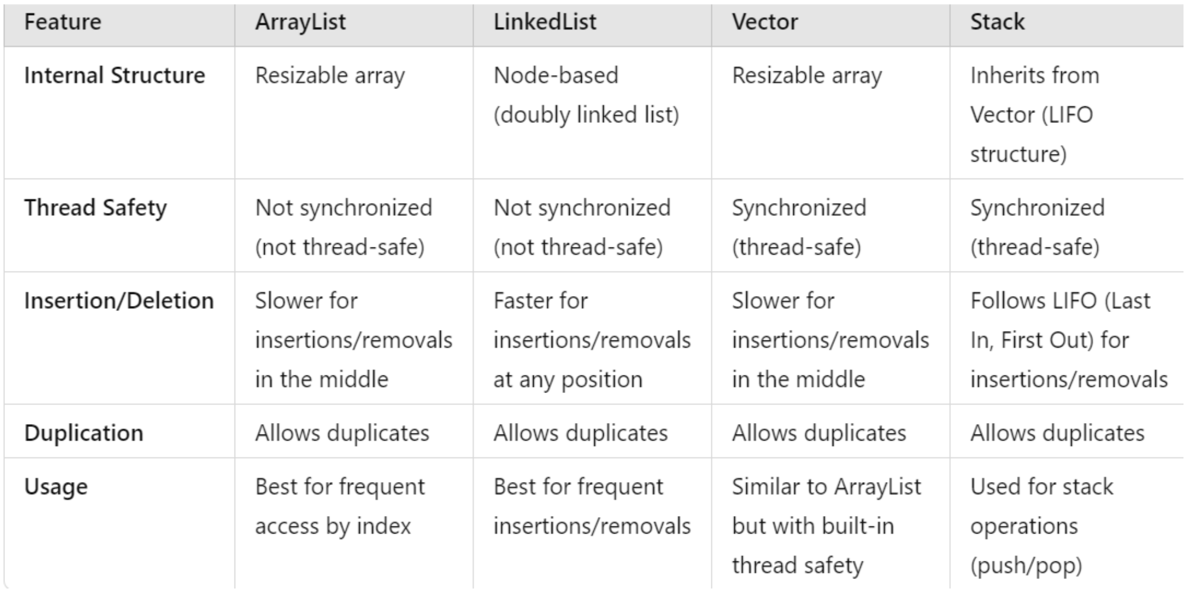
The advantages of the stack are:

- It helps you to manage the data in a Last In First Out (LIFO) method
- Local variables in functions are stored in the stack and removed when the function ends.
- A stack is used when we want variable is not used outside that function.()
- It allows you to control how memory is allocated and deallocated.
- Stack automatically cleans up the object.
- Not easily corrupted
- Variables cannot be resized.



```
// Function that uses local variables
public static void myFunction() {
    int localVar = 10;   // localVar is stored on the
    System.out.println("Local variable: " + localVar)
}

public static void main(String[] args) {
    myFunction();   // Calling the function
    // localVar is no longer accessible here, as it w
}
```

Tanvir s

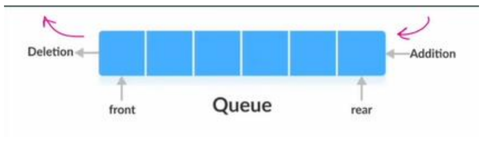


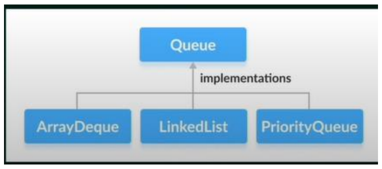
Feature	ArrayList	LinkedList	Vector	Stack
Internal Structure	Resizable array	Node-based (doubly linked list)	Resizable array	Inherits from Vector (LIFO structure)
Thread Safety	Not synchronized (not thread-safe)	Not synchronized (not thread-safe)	Synchronized (thread-safe)	Synchronized (thread-safe)
Insertion/Deletion	Slower for insertions/removals in the middle	Faster for insertions/removals at any position	Slower for insertions/removals in the middle	Follows LIFO (Last In, First Out) for insertions/removals
Duplication	Allows duplicates	Allows duplicates	Allows duplicates	Allows duplicates
Usage	Best for frequent access by index	Best for frequent insertions/removals	Similar to ArrayList but with built-in thread safety	Used for stack operations (push/pop)

Tanvir s

Java Queue Interface

The Queue is interface of the java collection framework provides the functionality of the queue Data structure .it extends collection interface(FIFO)

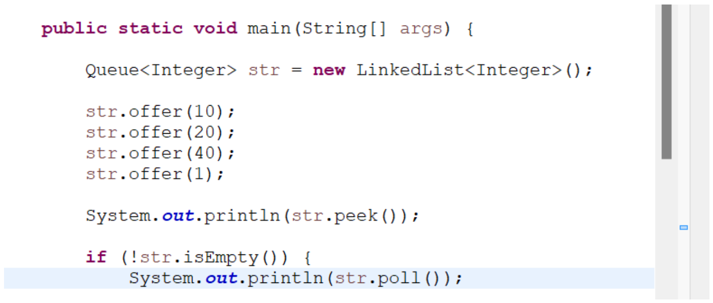




❖ **offer(E e)→** insert elements return true false (returns false if queue is full)	Boolean Add(E e)→insert element but return exception if queue is full
❖ **E poll()→** remove elements from queue and return null if q is empty	E remove →remove but return exception if queue is empty
❖ **E peek()→** retrieve element which will be remove only showing and return null if queue empty	E element→ retrieve element if q empty throw an exception

1. **Linked List Queue→**

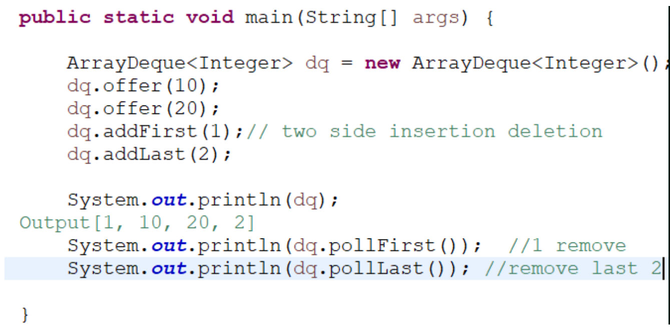
❖ The linked list class implements to queue Interface and provide method like peek(),poll(),offer()
❖ It follow FIFO



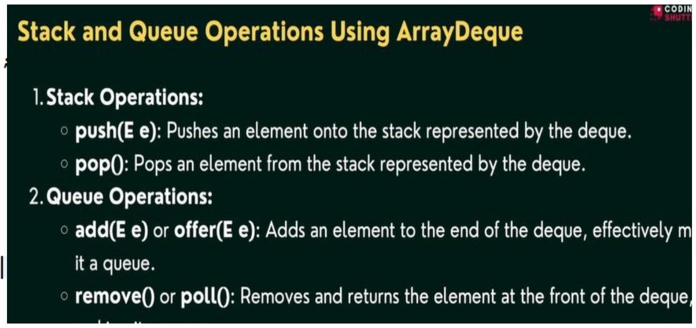
```
public static void main(String[] args) {

    Queue<Integer> str = new LinkedList<Integer>();

    str.offer(10);
    str.offer(20);
    str.offer(40);
    str.offer(1);

    System.out.println(str.peek());

    if (!str.isEmpty()) {
        System.out.println(str.poll());
```

2. **Array Deque→** Java.util.Deque

❖ array Deque is resizable array based implementation of the Deque(Double ended queue)
❖ it allows elements to be added or removal from both side of queue (pollFirst(),peekFirst(),offerFirst(),getFirst)



```
public static void main(String[] args) {

    ArrayDeque<Integer> dq = new ArrayDeque<Integer>();
    dq.offer(10);
    dq.offer(20);
    dq.addFirst(1);// two side insertion deletion
    dq.addLast(2);

    System.out.println(dq);
Output[1, 10, 20, 2]
    System.out.println(dq.pollFirst());   //1 remove
    System.out.println(dq.pollLast()); //remove last 2

}
```



Stack and Queue Operations Using ArrayDeque

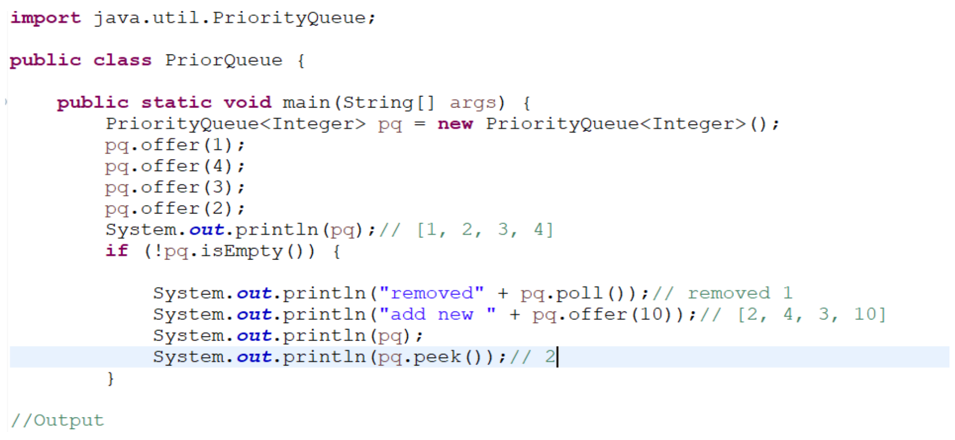
1. **Stack Operations:**
   - **push(E e)**: Pushes an element onto the stack represented by the deque.
   - **pop()**: Pops an element from the stack represented by the deque.
2. **Queue Operations:**
   - **add(E e)** or **offer(E e)**: Adds an element to the end of the deque, effectively m it a queue.
   - **remove()** or **poll()**: Removes and returns the element at the front of the deque,

is Java, DQ an interface that extends Queue interface. It gives support for the insertion and deletion of elements at both the end. This Queue is also called a double-ended queue.

//

3. **Priority queue→**

- According to priority elements are remove and insert
- Elements which is higher priority are removed first if two elements have same priority they are removed in the ordered they were added
- React like minheap means always remove smallest element but we can also custom it like maxheap
- Automatically ordered from smallest element ascending



```
import java.util.PriorityQueue;

public class PriorQueue {

    public static void main(String[] args) {
        PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
        pq.offer(1);
        pq.offer(4);
        pq.offer(3);
        pq.offer(2);
        System.out.println(pq);// [1, 2, 3, 4]
        if (!pq.isEmpty()) {

            System.out.println("removed" + pq.poll());// removed 1
            System.out.println("add new " + pq.offer(10));// [2, 4, 3, 10]
            System.out.println(pq);
            System.out.println(pq.peek());// 2
        }

//Output
```

//

Define BlockingQueue

BlockingQueue is an interface that extends Queue and allows thread-safe operations with blocking behavior. It waits for the queue to become non-empty when retrieving elements or non-full when adding elements.

multiple threads can run concurrently, sharing resources without interfering with each other.

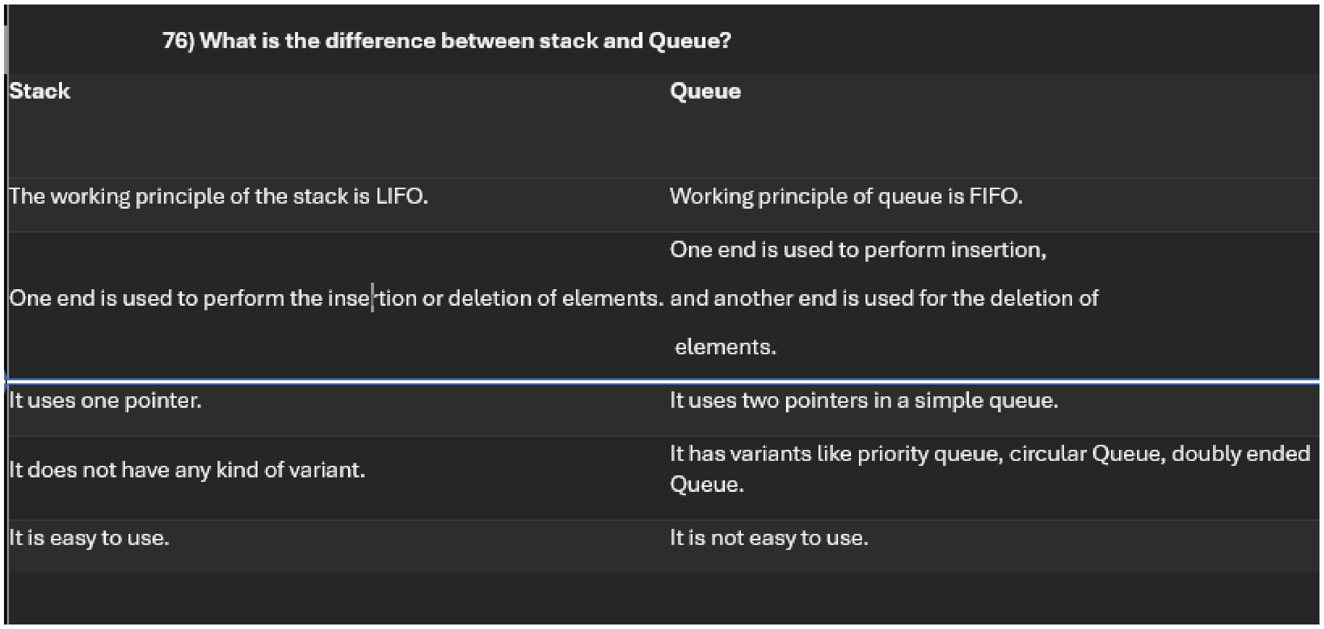
The syntax of BlockingQueue is:

public interface BlockingQueue<E> extends Queue <E>

Tanvir s

Explain Circular queues in Java

Circular Queue: It is a type of Queue in which user operations are performed based on the FIFO method. The last element is connected to the first position in order to make a circle.

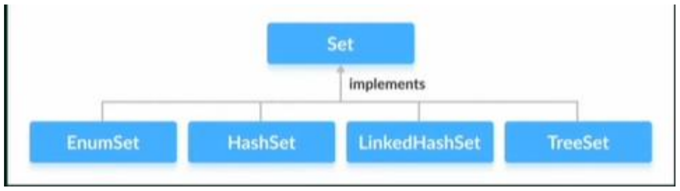


76) What is the difference between stack and Queue?

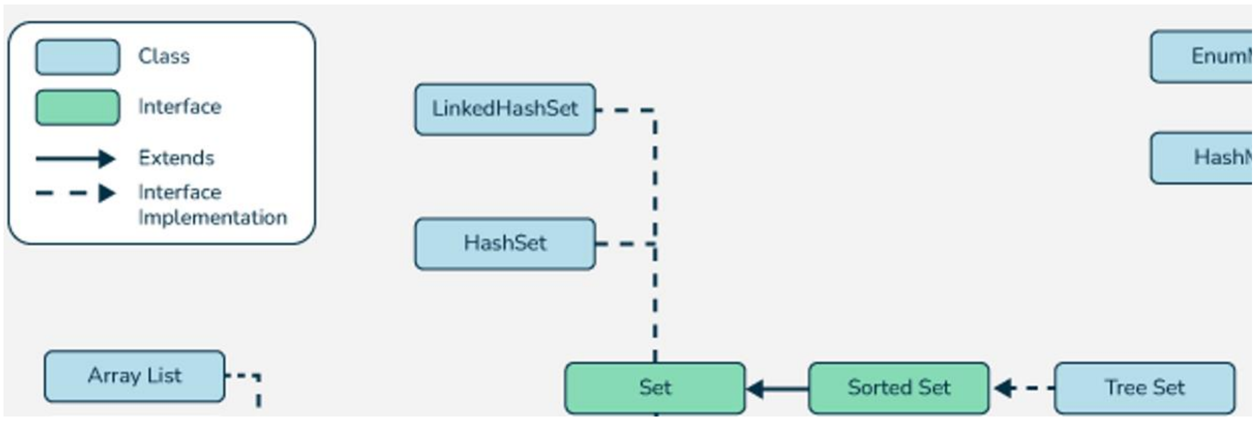
Stack	Queue
The working principle of the stack is LIFO.	Working principle of queue is FIFO.
One end is used to perform the insertion or deletion of elements.	One end is used to perform insertion, and another end is used for the deletion of elements.
It uses one pointer.	It uses two pointers in a simple queue.
It does not have any kind of variant.	It has variants like priority queue, circular Queue, doubly ended Queue.
It is easy to use.	It is not easy to use.

<u>Sets→</u>

Java Set interface



- **The set interface of java collection framework provides the features of the mathematical set in java**
- **It Extend the collection interface it cannot contains duplicate elements**
- **Time complexity is good o(n)**
-



Interfaces

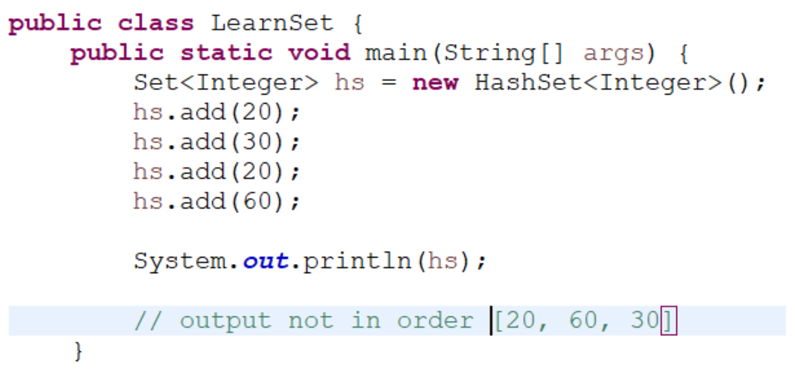
- Add()→add element to set
- addAll()→adds all element of the specified collection to the set
- remove()→remove specified elements from set
- remove all ()→remove all the elements
- retain all()→retains all the elements //intersection
- clear()→remove all the elements from set
- size()→return length of set
- contains→check elements pre

Define EnumSet

- EnumSet is a special type of Set in Java that only works with enum types. It stores enum values efficiently using bit vectors, and it does not allow null values. It's not synchronized and provides fast operations for enum collections. provides operations like adding, removing, and manipulating enums in a set-like structure.
- Eg. Main three
- enum Color { RED, GREEN, BLUE, YELLOW }
- EnumSet<Color> colors = EnumSet.of(Color.RED, Color.GREEN); // Red and Green
- EnumSet<Color> allColors = EnumSet.allOf(Color.class);   // All colors
  EnumSet<Color> noColors = EnumSet.noneOf(Color.class);   // Empty set

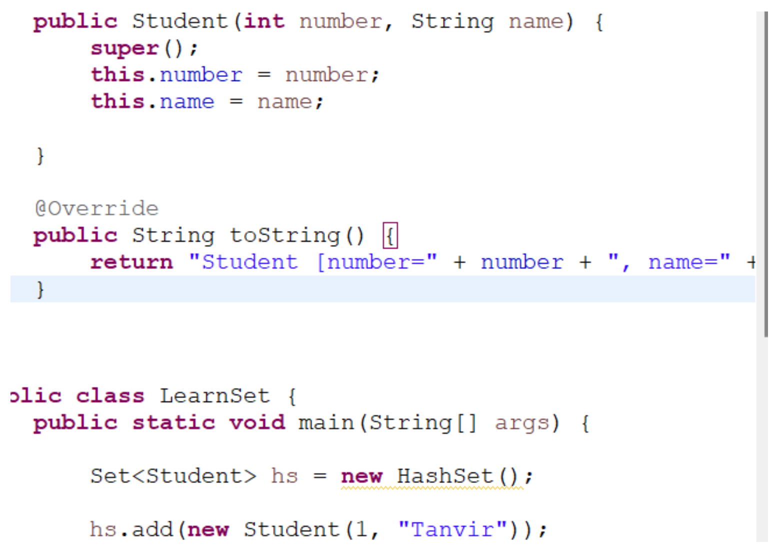
Java Hash set→

- ❖ in java hash set is commonly used if we have to access elements randomly
- ❖ hash set cannot contain duplicate hench each hash set has a unique hash code
- ❖ Hash code is function like hash code(input)→output //hash code has unique identity



```
public class LearnSet {
    public static void main(String[] args) {
        Set<Integer> hs = new HashSet<Integer>();
        hs.add(20);
        hs.add(30);
        hs.add(20);
        hs.add(60);

        System.out.println(hs);

        // output not in order [20, 60, 30]
    }
```

Hash set Using custom objects



```
    public Student(int number, String name) {
        super();
        this.number = number;
        this.name = name;

    }

    @Override
    public String toString() {
        return "Student [number=" + number + ", name=" +
    }


blic class LearnSet {
    public static void main(String[] args) {

        Set<Student> hs = new HashSet();

        hs.add(new Student(1, "Tanvir"));
```

What is the hashCode ()?

The hashCode () is a method that returns an integer hash code.

- ❖ hash set cannot contain duplicate hench each hash set has a unique hash code
- ❖ Hash code is function like hash code(input)→output //hash code has unique identity

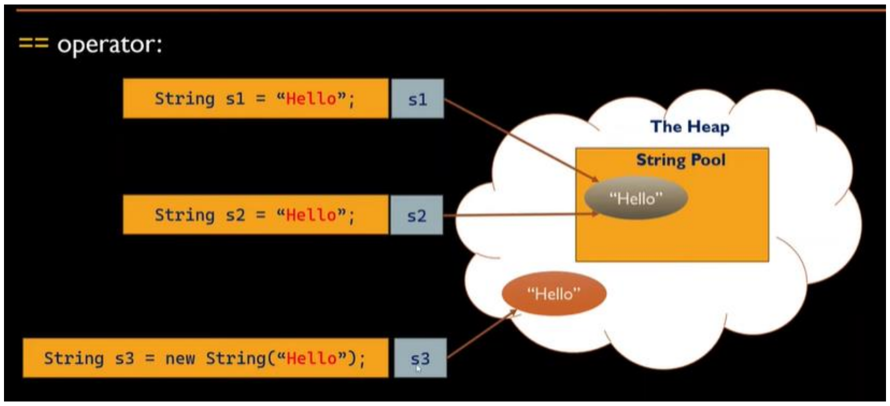
Hash code and Equals Methods

In java both method is used for comparing objects like equal or not

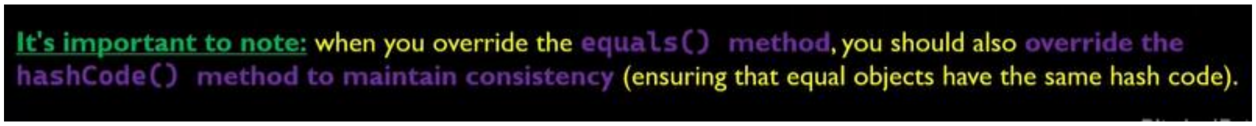
Equals () and == operator

"== Operator"

It compares the reference equality of two objects its check two objects references points to the same memory location or same object like below.



- ❖ here for non primitive(object) we can see reference (s1,s2) are refer to same String Object(hello) and in same memory location so output is True
- ❖ for primitive types (int float, char) it compares actual value of variables



It's important to note: when you override the equals() method, you should also override the hashCode() method to maintain consistency (ensuring that equal objects have the same hash code).

//

Explain Linked HashSet

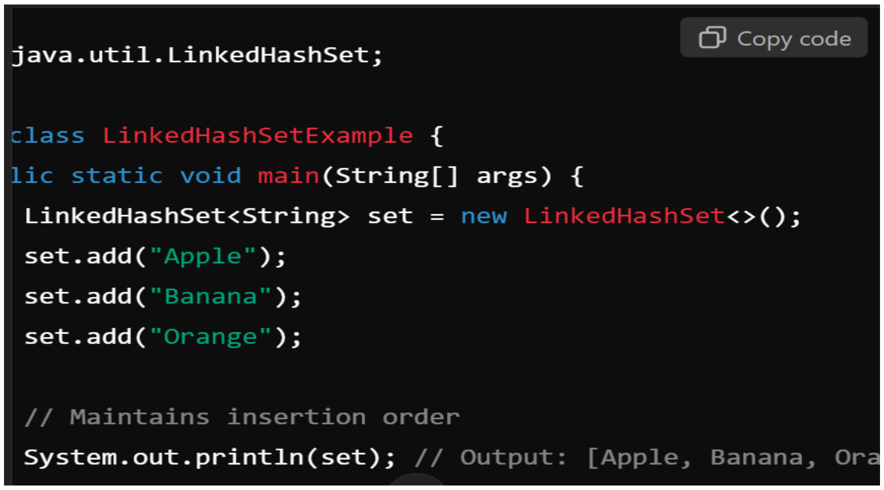
LinkedHashSet is a class in Java that implements the Set interface and extends HashSet. It is a **combination of a hash table** (for fast lookups) and a **doubly linked list** (to maintain insertion order). This means it behaves like a HashSet (stores unique elements) but also maintains the order in which elements were inserted.

Tanvir s

What are the two ways to remove duplicates from ArrayList?

Two ways to remove duplicates from ArrayList are:

- **HashSet:** Developer can use HashSet to remove the duplicate element from the ArrayList. The drawback is it cannot preserve the insertion order.
- **LinkedHashSet:** Developers can also maintain the order of insertion by using LinkedHashSet instead of HashSet.

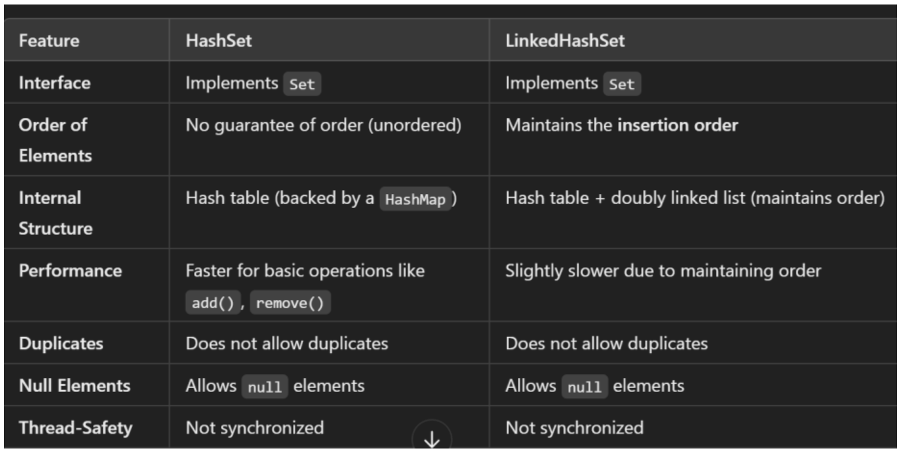


```
java.util.LinkedHashSet;

class LinkedHashSetExample {
lic static void main(String[] args) {
 LinkedHashSet<String> set = new LinkedHashSet<>();
 set.add("Apple");
 set.add("Banana");
 set.add("Orange");

 // Maintains insertion order
 System.out.println(set); // Output: [Apple, Banana, Ora
```

//

List various classes available in sets

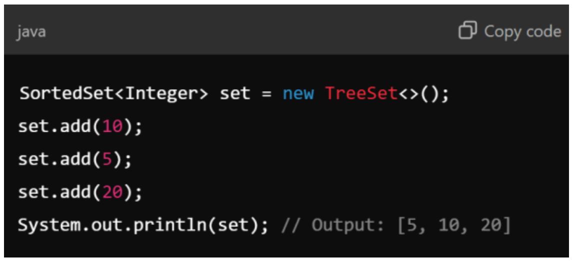
Various classes available in sets are: HashSet, TreeSetand, and LinkedHashSet.



Feature	HashSet	LinkedHashSet
Interface	Implements `Set`	Implements `Set`
Order of Elements	No guarantee of order (unordered)	Maintains the **insertion order**
Internal Structure	Hash table (backed by a `HashMap`)	Hash table + doubly linked list (maintains order)
Performance	Faster for basic operations like `add()`, `remove()`	Slightly slower due to maintaining order
Duplicates	Does not allow duplicates	Does not allow duplicates
Null Elements	Allows `null` elements	Allows `null` elements
Thread-Safety	Not synchronized	Not synchronized

SortedSet:

- **Definition: SortedSet is an interface in Java that defines a collection of elements sorted in a specific order (either natural or using a comparator).**
- Key Points:
    - Guarantees elements are stored in sorted order.
    - Provides methods like first(), last(), and subSet() to access sorted elements.
    - It doesn't define how elements are stored or managed, just that they must be sorted.



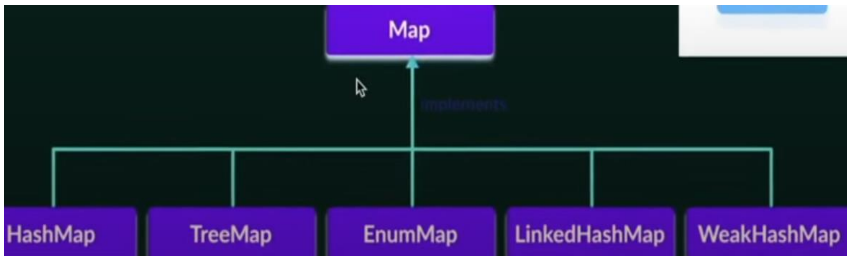
```java
SortedSet<Integer> set = new TreeSet<>();
set.add(10);
set.add(5);
set.add(20);
System.out.println(set); // Output: [5, 10, 20]
```

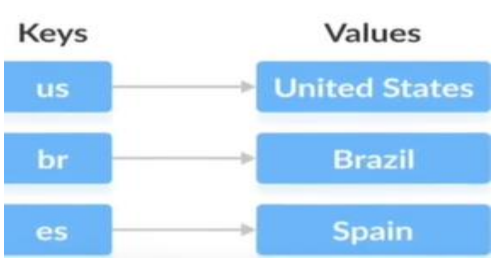
Tree Set:

- Definition: TreeSet is a class that implements the SortedSet interface and stores elements in a sorted order using a Red-Black Tree (a type of balanced binary search tree).
- Key Points:
    - Automatically sorts elements based on natural order or a custom comparator.
    - Does not allow null values.
    - Provides methods for navigating and manipulating the sorted collection, like first(), last(), and higher().
    - Operations like add(), remove(), and contains() are efficient, running in O(log n) time.

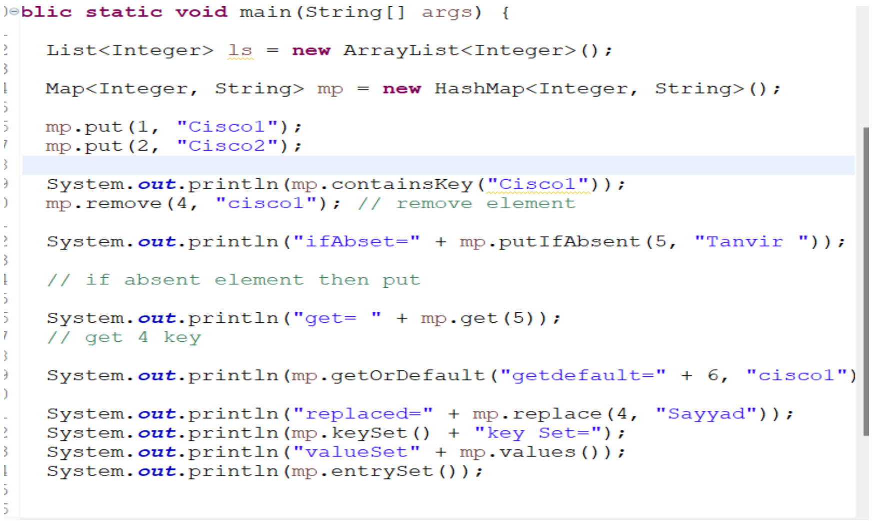
Map interface→

- In java elements of map are stored in key/value pairs. Keys are unique value associated with individual value
- A map cannot contains duplicate keys and each key is associated with single value





- Put (k, v)→for insert element with key value pair
- PutAll()→insert all entries from from the specified map to this map
- PutifAbset(k, v)→if absent keypair added
- Get(value or key)→get values
- getorDefault("key" defaultvalue)→if value is not present then get defaultvalue
- containsKey(k)→checks the specified key is present or not
- containsValue()→to check value
- remove()→to remove key and using this we can remove key pair
- keyset()→we get set of key\
- values()→we get set of value
- entrySet()→all key and value in present in map
- all complexity O(n)



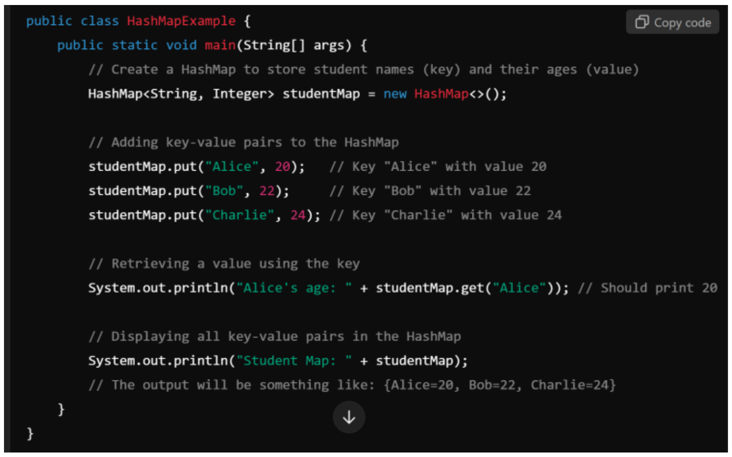
```
blic static void main(String[] args) {

  List<Integer> ls = new ArrayList<Integer>();

  Map<Integer, String> mp = new HashMap<Integer, String>();

  mp.put(1, "Cisco1");
  mp.put(2, "Cisco2");

  System.out.println(mp.containsKey("Cisco1"));
  mp.remove(4, "cisco1"); // remove element

  System.out.println("ifAbset=" + mp.putIfAbsent(5, "Tanvir "));

  // if absent element then put

  System.out.println("get= " + mp.get(5));
  // get 4 key

  System.out.println(mp.getOrDefault("getdefault=" + 6, "cisco1")

  System.out.println("replaced=" + mp.replace(4, "Sayyad"));
  System.out.println(mp.keySet() + "key Set=");
  System.out.println("valueSet" + mp.values());
  System.out.println(mp.entrySet());
```

Here are the types of Map in Java:

1. **HashMap**
2. **LinkedHashMap**
3. **TreeMap**
4. **Hashtable**
5. **WeakHashMap**
6. **IdentityHashMap**
7. **ConcurrentHashMap**

What is hashmap→

A HashMap is a collection in Java used to store key-value pairs, allowing fast access to values based on their corresponding keys. It is unordered and efficient for quick lookups.



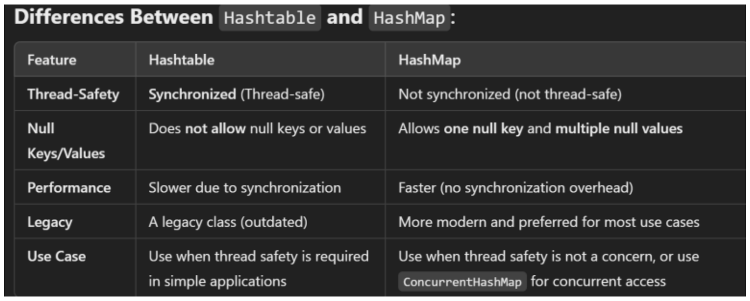
```
public class HashMapExample {
    public static void main(String[] args) {
        // Create a HashMap to store student names (key) and their ages (value)
        HashMap<String, Integer> studentMap = new HashMap<>();

        // Adding key-value pairs to the HashMap
        studentMap.put("Alice", 20);   // Key "Alice" with value 20
        studentMap.put("Bob", 22);     // Key "Bob" with value 22
        studentMap.put("Charlie", 24); // Key "Charlie" with value 24

        // Retrieving a value using the key
        System.out.println("Alice's age: " + studentMap.get("Alice")); // Should print 20

        // Displaying all key-value pairs in the HashMap
        System.out.println("Student Map: " + studentMap);
        // The output will be something like: {Alice=20, Bob=22, Charlie=24}
    }
}
```

//

What is hashtable in java

A **Hashtable** is a collection class in Java that implements the **Map** interface and is used to store key-value pairs. It is similar to **HashMap**, but with a few differences:

Hashtable does not allow null keys or null values. If you try to insert a null key or value, it throws a NullPointerException.



Differences Between `Hashtable` and `HashMap`:

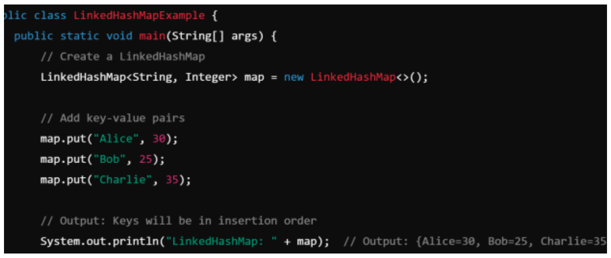
Feature	Hashtable	HashMap
Thread-Safety	**Synchronized** (Thread-safe)	Not synchronized (not thread-safe)
Null Keys/Values	Does **not allow** null keys or values	Allows **one null key** and **multiple null values**
Performance	Slower due to synchronization	Faster (no synchronization overhead)
Legacy	A legacy class (outdated)	More modern and preferred for most use cases
Use Case	Use when thread safety is required in simple applications	Use when thread safety is not a concern, or use `ConcurrentHashMap` for concurrent access

Explain Linkedhashmap

LinkedHashMap is a class that implements the Map interface and extends HashMap. It provides the same functionality as HashMap

but with one key difference: it maintains the insertion order of the entries

Allows One null Key and Multiple null Values: Just **like HashMap**

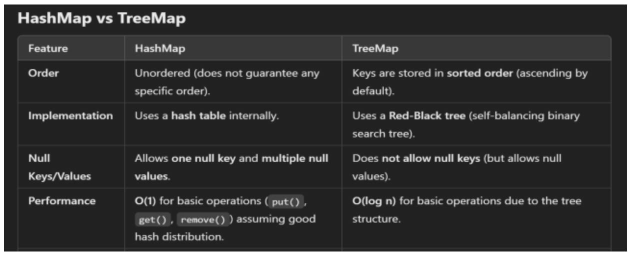


```
lic class LinkedHashMapExample {
 public static void main(String[] args) {
     // Create a LinkedHashMap
     LinkedHashMap<String, Integer> map = new LinkedHashMap<>();

     // Add key-value pairs
     map.put("Alice", 30);
     map.put("Bob", 25);
     map.put("Charlie", 35);

     // Output: Keys will be in insertion order
     System.out.println("LinkedHashMap: " + map);  // Output: {Alice=30, Bob=25, Charlie=35
```

//

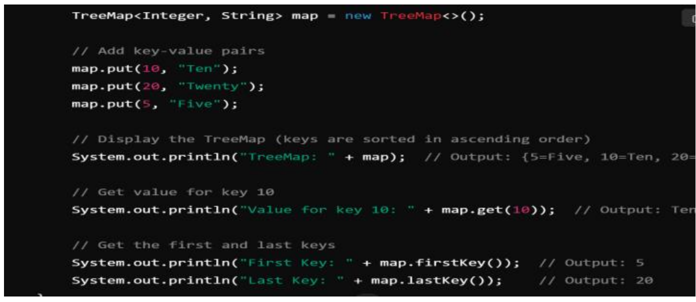
Explain Tree map in Java

TreeMap is a class that implements the Map interface and stores data in sorted order based on keys. It uses a Red-Black Tree internally to maintain the sorted order of keys. It also provides additional navigation methods like firstKey(), lastKey()



HashMap vs TreeMap

Feature	HashMap	TreeMap
Order	Unordered (does not guarantee any specific order).	Keys are stored in **sorted order** (ascending by default).
Implementation	Uses a **hash table** internally.	Uses a **Red-Black tree** (self-balancing binary search tree).
Null Keys/Values	Allows **one null key** and **multiple null values.**	Does **not allow null keys** (but allows null values).
Performance	**O(1)** for basic operations (`put()`, `get()`, `remove()`) assuming good hash distribution.	**O(log n)** for basic operations due to the tree structure.



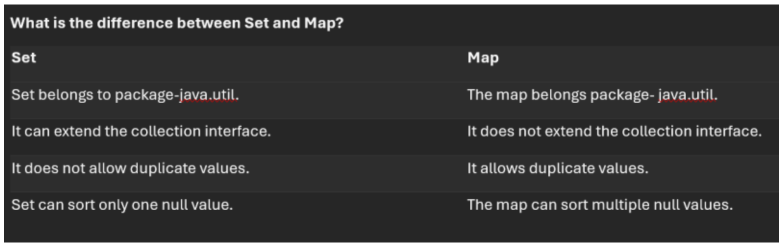
```
        TreeMap<Integer, String> map = new TreeMap<>();

        // Add key-value pairs
        map.put(10, "Ten");
        map.put(20, "Twenty");
        map.put(5, "Five");

        // Display the TreeMap (keys are sorted in ascending order)
        System.out.println("TreeMap: " + map);  // Output: {5=Five, 10=Ten, 20=

        // Get value for key 10
        System.out.println("Value for key 10: " + map.get(10));  // Output: Ten

        // Get the first and last keys
        System.out.println("First Key: " + map.firstKey());  // Output: 5
        System.out.println("Last Key: " + map.lastKey());    // Output: 20
```

//

What is WeakHashMap?

- **WeakHashMap** is a special type of map where the **keys** are stored as **weak references**. This allows the garbage collector to automatically remove entries when their keys are no longer in use.
- It's useful for **caching** and **memory-sensitive applications**, where you want the map to automatically clean up unused entries without needing explicit removal.

//



What is the difference between Set and Map?

Set	Map
Set belongs to package-java.util.	The map belongs package- java.util.
It can extend the collection interface.	It does not extend the collection interface.
It does not allow duplicate values.	It allows duplicate values.
Set can sort only one null value.	The map can sort multiple null values.

Tanvir s

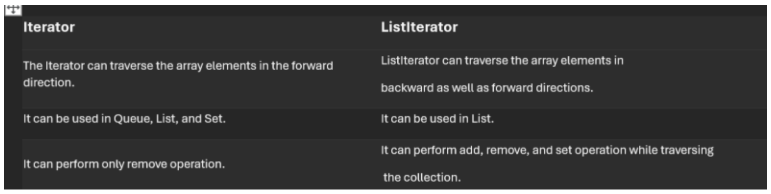
how to iterate map?

The developer cannot directly iterate map, but, this interface has two methods that gives view set of map. These methods are:

- **Set<Map.Entry<K, V>>entrySet():** It is a method that returns a set having the entries mention in the map. These entries are generally objected, which has type Map. Entry.

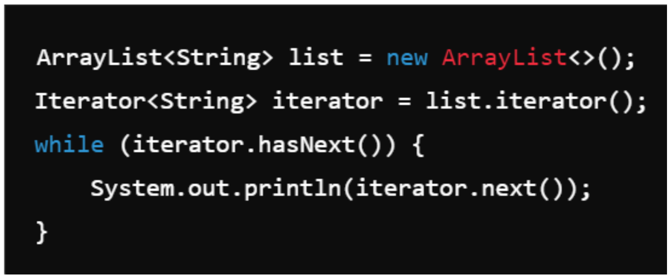
- **Set<K>keySet():** This Java method returns a set that having the map key.

Differentiate between Iterator and ListIterator



Iterator	ListIterator
The Iterator can traverse the array elements in the forward direction.	ListIterator can traverse the array elements in backward as well as forward directions.
It can be used in Queue, List, and Set.	It can be used in List.
It can perform only remove operation.	It can perform add, remove, and set operation while traversing the collection.

- iterator in java is an object that allows you to traverse through collection (like Arraylist, Hashset)
- it provides methods to access and remove elements during iteration
- process of repeatedly accessing each element in a collection or a sequence, one after another. This can be done using loops (e.g., for, while) or with an Iterator.



```
ArrayList<String> list = new ArrayList<>();
Iterator<String> iterator = list.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

What is the difference between Iterator and Enumeration?

The difference between Iterator and Enumeration

Iterator	Enumeration
The Iterator can traverse both legacies as well as non-legacy elements.	Enumeration can traverse only legacy elements.
The Iterator is fail-fast.	Enumeration is not fail-fast.
The Iterator is very slow compare to Enumeration.	Enumeration is fast compare to Iterator.
The Iterator can perform remove operation while traversing the collection.	The Enumeration can perform only traverse operation on the collecti

---

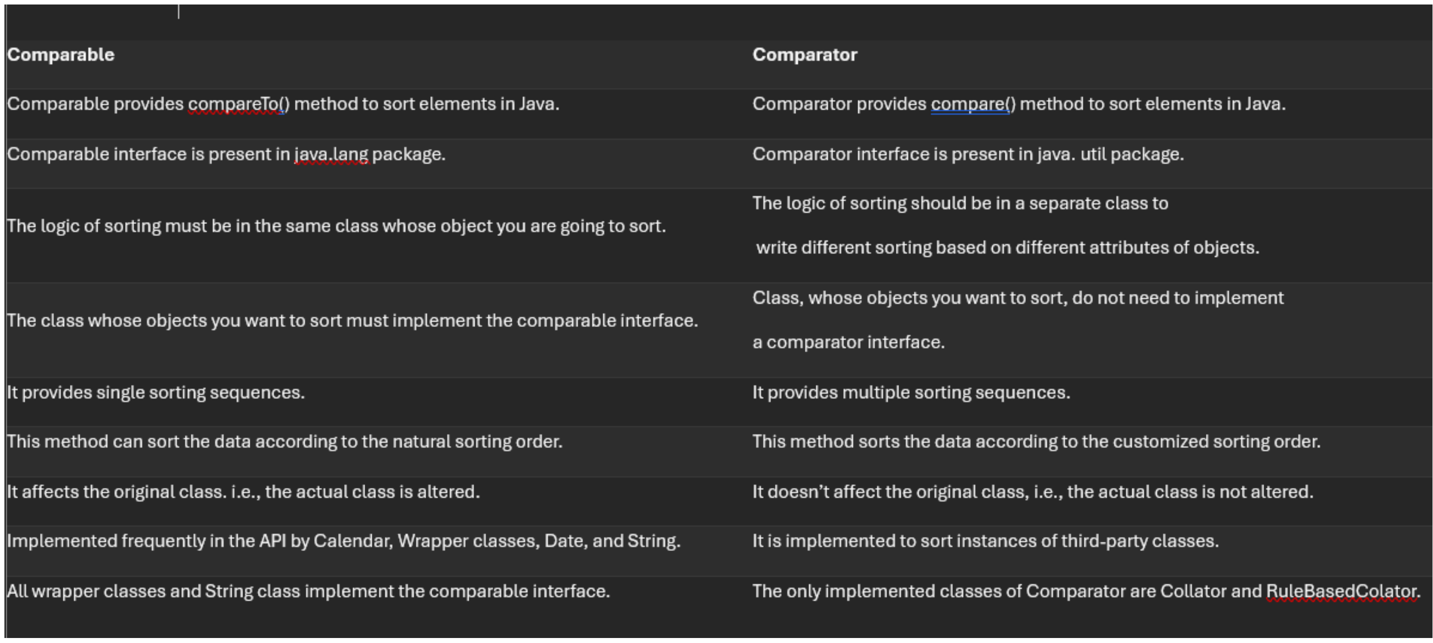
What is a good way to sort the Collection objects in Java?

A good way to sort Java collection objects is using Comparable and Comparator interfaces. A developer can use Collections.sort(), the elements are sorted based on the order mention in compareTo().

When a developer uses Collections, sort (Comparator), it sorts the objects depend on compare() of the Comparator interface.

Tanvir s

What is the difference between Comparable and Comparator?



Comparable	Comparator
Comparable provides compareTo() method to sort elements in Java.	Comparator provides compare() method to sort elements in Java.
Comparable interface is present in java.lang package.	Comparator interface is present in java. util package.
The logic of sorting must be in the same class whose object you are going to sort.	The logic of sorting should be in a separate class to write different sorting based on different attributes of objects.
The class whose objects you want to sort must implement the comparable interface.	Class, whose objects you want to sort, do not need to implement a comparator interface.
It provides single sorting sequences.	It provides multiple sorting sequences.
This method can sort the data according to the natural sorting order.	This method sorts the data according to the customized sorting order.
It affects the original class. i.e., the actual class is altered.	It doesn't affect the original class, i.e., the actual class is not altered.
Implemented frequently in the API by Calendar, Wrapper classes, Date, and String.	It is implemented to sort instances of third-party classes.
All wrapper classes and String class implement the comparable interface.	The only implemented classes of Comparator are Collator and RuleBasedColator.

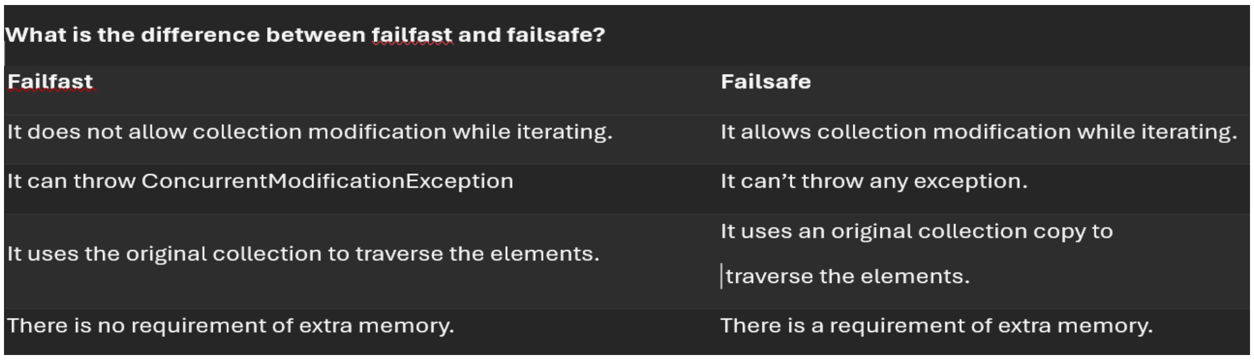
What is a Stack?

A stack is a special area of computer's memory that stores temporary variables created by a function. In stack, variables are declared, stored, and initialized during runtime.

Define emptySet() in the Java collections framework

Method emptySet() that returns the empty immutable set whenever programmers try to remove null elements. The set which is returned by emptySet() is serializable. The syntax of this method is:

public static final <T> Set<T> emptySet()



What is the difference between failfast and failsafe?

Failfast	Failsafe
It does not allow collection modification while iterating.	It allows collection modification while iterating.
It can throw ConcurrentModificationException	It can't throw any exception.
It uses the original collection to traverse the elements.	It uses an original collection copy to traverse the elements.
There is no requirement of extra memory.	There is a requirement of extra memory.

List collection views of a map interface

Collection views of map interface are: 1) key set view, 2) value set view, and 3) entry set view.

Define dictionary class

The Dictionary class is a Java class that has a capability to store key-value pairs.

What are the methods to make collection thread-safe?

The methods to make collection thread safe are:

- Collections.synchronizedList(list);
- Collections.synchronizedMap(map);
- Collections.synchronizedSet(set);

Explain UnsupportedOperationException

UnsupportedOperationException is an exception whch is thrown on methods that are not supported by actual collection type.

For example, Developer is making a read-only list using “Collections.unmodifiableList(list)” and calling call(), add() or remove() method. It should clearly throw UnsupportedOperationException.

Name the collection classes that gives random element access to its elements

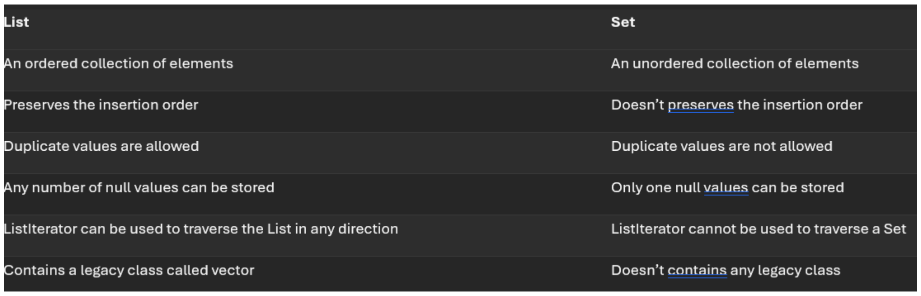
Collection classes that give random element access to its elements are: 1) ArrayList, 2) HashMap, 3) TreeMap, and 4) Hashtable.

Explain the design pattern followed by Iterator

The iterator follows the detail of the iterator design pattern. It provides developer to navigate through the objects collections using a common interface without knowing its implementation.

What is the peek() of the Queue interface?

Peek () is a method of queue interface. It retrieves all the elements but does not remove the queue head. In case if the Queue is empty, then this method will return null.



List	Set
An ordered collection of elements	An unordered collection of elements
Preserves the insertion order	Doesn’t preserves the insertion order
Duplicate values are allowed	Duplicate values are not allowed
Any number of null values can be stored	Only one null values can be stored
ListIterator can be used to traverse the List in any direction	ListIterator cannot be used to traverse a Set
Contains a legacy class called vector	Doesn’t contains any legacy class

Tanvir s

Explain for each loop with example

For-Each Loop is another form of for loop used to traverse the array. It reduces the code significantly, and there is no use of the index or rather the counter in the loop.

Explain diamond operator

Diamond operator enables the compiler to collect the type arguments of generic class. In Java SE, developer can substitute the parameterized constructor with an empty parameter sets (<>) known as diamond operator.

Explain randomaccess interface

RandomAccess interface is used by List implementations for the indication that they are supporting fast.

Name the collection classes that implement random access interface

Java.util package has classes that can implement random access interface are: CopyOnWriteArrayList, Stack, ArrayList, and Vector.

How to join multiple ArrayLists?

The list provides a addall() method multiple ArrayList in Java.

For example, consider two lists 1) areaList and 2) secondAreaList. A developer can join them using addall() like:

areaList.addAll(secondAreaList);

Which method is used to sort an array in ascending order?

Java collection framework method, Collections.sort() is used to sort an array in ascending order.

Explain Big-O notation with an example

The Big-O notation depicts the performance of an algorithm as the number of elements in ArrayList. A developer can use Big-O notation to choose the collection implementation. It is based on performance, time, and memory.

For example, ArrayList get(index i) is a method to perform a constant-time operation. It does not depend on the total number of elements available in the list. Therefore, the performance in Big-O notation is O(1).