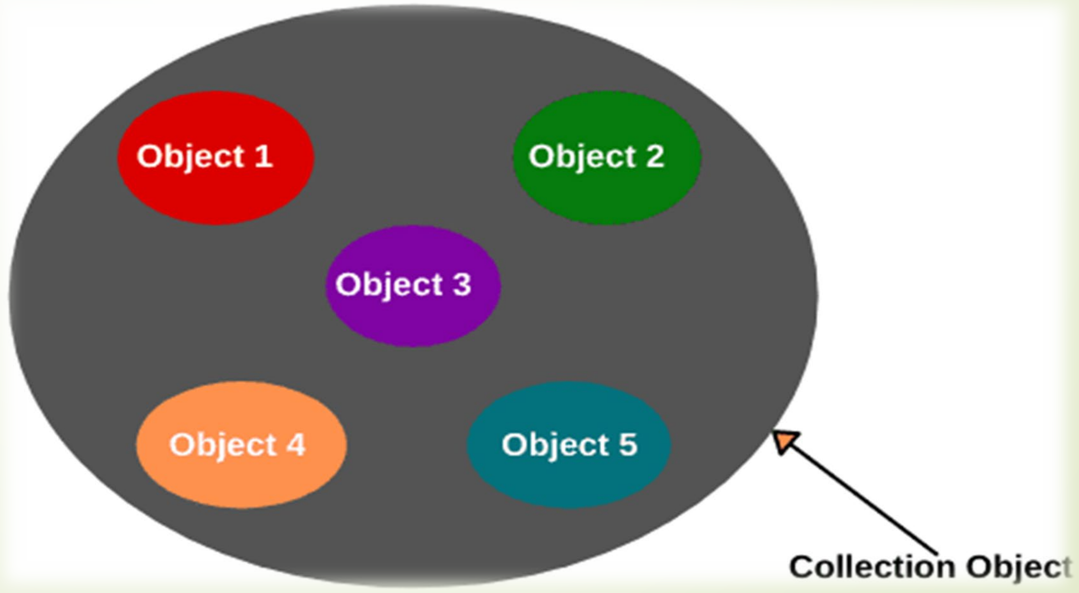
Collection in Java

Collection in Java

- A collection is a group of objects. In Java, these objects are called elements of the collection.
- Initially, collection framework is simply called **Java.util** package or Collection API.
- Later on, Sun Microsystem had introduced the collection framework in Java 1.2.
- It was developed and designed by “**Joshua Bloch**”.
- Later on, after Java 1.2, it is known as collections framework.
- From Java 1.5 onwards, The Sun Microsystem added some more new concepts called Generics.

Collection in Java

1. Technically, a collection is an object or container which stores a group of other objects as a single unit or single entity.
2. Therefore, it is also known as container object or collection object in java.
3. A container object means it contains other objects.
4. In simple words, a collection is a container that stores multiple elements together.



Differences between Arrays and Collections ?

Arrays	Collections
1) Arrays are fixed in size.	1) Collections are growable in nature.
2) Memory point of view arrays are not recommended to use.	2) Memory point of view collections are highly recommended to use.
3) Performance point of view arrays are recommended to use.	3) Performance point of view collections are not recommended to use.
4) Arrays can hold only homogeneous data type elements.	4) Collections can hold both homogeneous and heterogeneous elements.
5) There is no underlying data structure for arrays and hence there is no readymade method support.	5) Every collection class is implemented based on some standard data structure and hence readymade method support is available.
6) Arrays can hold both primitives and object types.	6) Collections can hold only objects but not primitives.

Types of Objects Stored in Collection

There are two types of objects that can be stored in a collection or container object.

1. **Homogeneous objects:** Homo means same. Homogeneous objects are a group of multiple objects that belong to the same class.

   Example:

   For example, suppose we have created three objects class 'Student'.

   Student s1,

   Student s2,

   Student s3,

Since these three objects belong to the same class that's why they are called homogeneous objects.

Types of Objects Stored in Collection (Conti..)

There are two types of objects that can be stored in a collection or container object.

2. **Heterogeneous objects:** Hetero means different. Heterogeneous objects are a group of different objects that belong to different classes.

For example:

Suppose we have created two different objects of different classes such as one object Student s1, and another one object Employee e1.

Here, student and employee objects together are called a collection of heterogeneous objects.

Types of Objects Stored in Collection (Conti..)

These objects can also be further divided into two types.

1. **Duplicate objects:**

```
Person p1 = new Person( “cdac");
Person p2 = new Person(“cdac");
```

2. **Unique objects:**

```
Person p1 = new Person(“cdac");
Person p2 = new Person(“delhi");
```

Advantages:

- The collections framework reduces the development time and the burden of designers, programmers, and users.
- Your code is easier to maintain because it provides useful data structure and interfaces which reduce programming efforts.
- The size of the container is **growable** in nature.
- It implements high-performance of useful data structures and algorithms that increase the performance.
- It enables software reuse.

Java Collections Framework

- A framework in java is a set of several classes and interfaces which provide a ready-made architecture.
- It is a hierarchy of several predefined interfaces and implementation classes that can be used to handle a group of objects as a single entity.
- In other words, a collections framework is a class library to handle groups of objects.
- It is present in **java.util** package.
- It allows us to store, retrieve, and update a group of objects.

Collections Framework Example:

1. The **Computer** is a framework; it can be used by all people according to their requirements.

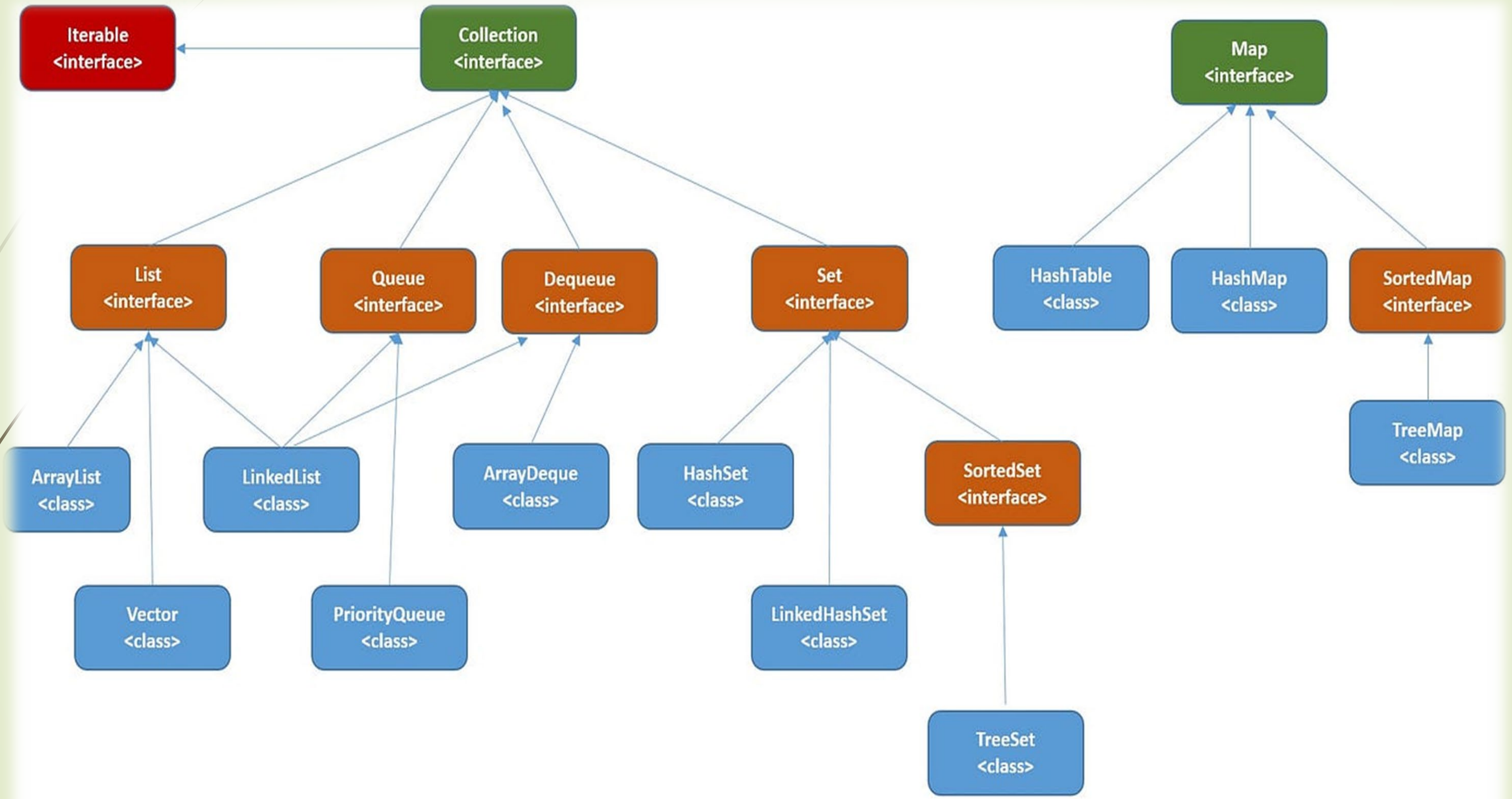
2. **Switchboard** is a framework it can be used by all people according to their requirement.

3. In java **struts**, **hibernate**, **spring technologies** are frameworks, all these technologies are used by different companies for developing projects according to the project requirements.

Limitation of Collections Framework in Java

1. Be careful to use the appropriate cast operation.
2. Compile type checking is not possible.

Collection Hierarchy in Java



Summary of Collection Hierarchy in Java

1. The key interfaces in collection hierarchy are *List, Set, Queue, and Map*.
2. The key implementation classes in the collection hierarchy are *ArrayList, LinkedList, HashSet, TreeSet, PriorityQueue, HashMap, and TreeMap.*
3. The most commonly used methods of collection interface include *add(), remove(), size(), isEmpty(), contains(), and clear().*

List Interface in Java

- A list in Java is a collection for storing elements in sequential order.
- Sequential order means the first element, followed by the second element, followed by the third element, and so on.
- Java list is a sub-interface of the collection interface that is available in **java.util** package.
- Sub interface means an interface that extends another interface is called sub interface.
- Here, the list interface extends the collection interface.
- The general declaration of list interface in java is as follows:

public interface List<E> extends Collection<E>

List Interface in Java (Conti..)

- It is an ordered collection where elements are maintained by index positions because it uses an index-based structure.
- It is used to store a collection of elements where duplicate elements are allowed.
- List interface in java has four concrete subclasses. They are **ArrayList**, **LinkedList**, **Vector**, and **Stack**. These four subclasses implements the list interface.
- **ArrayList** and **LinkedList** are widely used in Java programs to create a list.

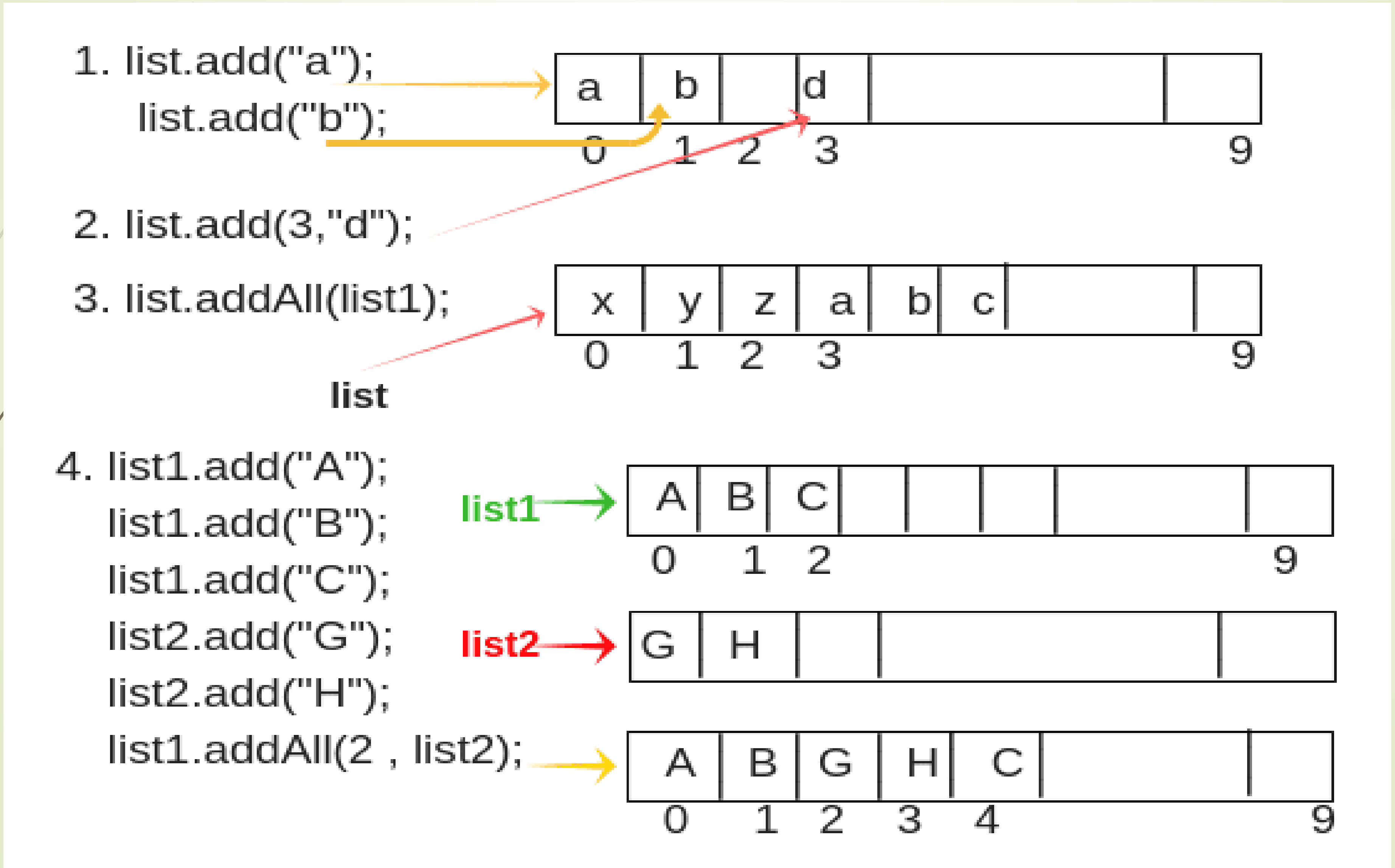
Features of Java List Interface

- The list allows storing duplicate elements in Java.
- In the list, we can add an element at any position.
- It maintains **insertion order**. i.e. List can preserve the insertion order by using the index.
- It allows for storing many **null** elements.
- Java list uses a resizable array for its implementation. **Resizable** means we can increase or decrease the size of the array.
- It is an **indexed-based** structure.

Features of Java List Interface (Cont..)

- It provides a special Iterator called a **ListIterator** that allows accessing the elements in the forward direction using **hasNext**() and **next()** methods.
- In the reverse direction, it accesses elements using **hasPrevious()** and **previous()** methods.
- We can add, remove elements of the collection, and can also replace the existing elements with the new element using **ListIterator**.

List Interface Methods



```
1. list.add("a");
   list.add("b");

2. list.add(3,"d");

3. list.addAll(list1);

4. list1.add("A");
   list1.add("B");
   list1.add("C");
   list2.add("G");
   list2.add("H");
   list1.addAll(2 , list2);
```

ArrayList in Java

1. ArrayList in Java is a dynamic array that allows us to store multiple objects of any class or data type.
2. It is similar to an array, but there is no fixed size limit.
3. It was added to the Java programming in JDK 1.2 as part of the Java Collections Framework.
4. If the initial capacity of the array is exceeded, a new array with larger capacity is created automatically and all the elements from the current array are copied to the new array.
5. Elements in ArrayList are placed according to the zero-based index. That is the first element will be placed at 0 index and the last element at index (n-1) where n is the size of ArrayList.

ArrayList in Java (Conti..)

6. ArrayList class also implements 3 marker interfaces in Java. They are **Random Access**, **Cloneable**, and **Serializable**.
7. A marker interface is an interface that does not have any methods or any member variables.
8. It is also called an empty interface because of no field or methods.
9. It creates an empty ArrayList with a default initial capacity of 10.
10. Once ArrayList is reached its maximum capacity, the ArrayList class automatically creates a new array with a larger capacity.

New capacity = (current capacity*3/2) + 1 = 10*3/2 + 1 = 16

The old default array list with the collection of objects is automatically gone into the garbage collection.

Similarly, If we try to insert 17th element, new capacity= 16*3/2 + 1 = 25.

Serializable Interface

1. A serializable interface is a marker interface that is used to send the group of objects over the network. It is present in the **java.io** package.

2. It helps in sending the data from one class to another class. Usually, we use collections to hold and transfer objects from one place to another place.

3. To provide support for this requirement, every collections class already implements **Serializable** and **Cloneable**.

Cloneable Interface

1. A cloneable interface is present in **java.lang** package.

2. It is used to create exactly duplicate objects. When the data or group of objects came from the network, the receiver will create duplicate objects.

3. The process of creating exactly duplicate objects is known as **cloning**. It is a very common requirement for collection classes.

Features of ArrayList in Java

1. **ArrayList** is a resizable array or growable array that means the size of **ArrayList** can increase or decrease in size at runtime.
2. It uses **an index-based** structure in java.
3. **Duplicate** elements are allowed in the array list.
4. Any number of null elements can be added to **ArrayList**.
5. It maintains the insertion order in Java. That is insertion order is preserved.
6. **Heterogeneous** objects are allowed everywhere except **TreeSet** and **TreeMap**. Heterogeneous means different elements.
7. **ArrayList** is not synchronized. That means multiple threads can use the same ArrayList objects simultaneously.
8. **ArrayList** implements random access because it uses an index-based structure.

Set in Java

- Set is an unordered collection of elements. That means the order is not maintained while storing elements. While retrieving we may not get the same order as that we put elements.

- It is used to store a collection of elements without duplicate. That means it contains only unique elements.

- Java Set uses map based structure for its implementation.

- It can be iterated by using Iterator but cannot be iterated by using ListIterator.

- Most of the set implementations allow adding only one null element. Tree set does not allow to add null element.

Set in Java

- Set is not an indexed-based structure like a list in Java. Therefore, we cannot access elements by their index position.

- It does not provide any get method like a list.

- Set is an interface that was introduced in Java 1.2 version. It is a generic interface that is declared as:

  interface Set<E>

Here, E defines the type of elements that the set will hold.

Features of Java Set

1. Set is an unordered collection of elements. That means the order is not maintained while storing elements. While retrieving we may not get the same order as that we put elements.

2. It is used to store a collection of elements without duplicate. That means it contains only unique elements.

3. Java Set uses map based structure for its implementation.

4. It can be iterated by using Iterator but cannot be iterated by using **ListIterator**.

Features of Java Set

5. Most of the set implementations allow adding only one null element. Tree set does not allow to add null element.

6. Set is not an indexed-based structure like a list in Java. Therefore, we cannot access elements by their index position.

7. It does not provide any get method like a list.

Points to remember about Set Interface

1. Set is an interface in the **java.util** package that stores a collection of unique elements with no duplicates allowed.

2. **HashSet**, **TreeSet**, and **LinkedHashSet** are the three types of set in Java, each with its unique features and implementations.

3. Set interface provides methods such as **add(), remove(), contains(), size(),** and **clear()** for performing various operations on the set collections.

4. It also provides **addAll(), removeAll(),** and **retainAll()** for performing set operations on multiple sets.

Points to remember about Set Interface

5. Set interface does not provide methods for accessing elements by their index or position.

6. Set interface is not thread-safe, and the implementation classes may require synchronization for concurrent access.

7. We usually use set interface for solving problems when we need to store a collection of unique elements, finding common elements, removing duplicates, and performing set operations.

HashSet in Java

- **HashSet** in Java is an unordered collection of elements (objects) that contains only unique elements.
- It internally uses **Hashtable** data structure to store a set of unique elements.
- It is much faster and gives constant-time performance for searching and retrieving elements.
- **HashSet** was introduced in Java 1.2 version and it is present in java.util.HashSet package.
- **HashSet** class extends **AbstractSet** class and implements the Set interface.
- The **AbstractSet** class itself extends **AbstractCollection** class.

Java HashSet class declaration

HashSet is a concrete generic class that can be declared in general form as below:

```
public class HashSet<E> extends AbstractSet<E>
                  implements Set<E>, Cloneable, Serializable
```

Here, E defines the type of elements that the set will hold.

Features of HashSet

- The underlying data structure of **HashSet** is **Hashtable**.
- A hash table stores data by using a mechanism called **hashing**.
- **HashSet** does not allow duplicate elements. If we try to add a duplicate element in **HashSet**, the older element would be overwritten.
- It allows only one null element. If we try to add more than one null element, it would still return only one null value.
- **HashSet** does not maintain any order in which elements are added. The order of the elements will be unpredictable. It will return in any random order.
- **Hashing** provides constant execution time for methods like **add(), remove(), contains(), and size()** even for the large set.

Features of HashSet (Conti..)

- **HashSet** class is not synchronized which means it is not thread-safe. Therefore, multiple threads can use the same **HashSet** object at the same time and will not give the deterministic final output.

- If you want to synchronize **HashSet**, use **Collections.synchronizeSet()** method.

LinkedList in Java | Methods

LinkedList in Java

1. LinkedList in Java is a linear data structure that uses a doubly linked list internally to store a group of elements. A doubly linked list consists of a group of nodes that together represents a sequence in the list. It stores the group of elements in the sequence of nodes.

2. Each node contains three fields: a data field that contains data stored in the node, left and right fields contain references or pointers that point to the previous and next nodes in the list. A pointer indicates the addresses of the next node and the previous node. Elements in the linked list are called nodes.

3. Since the previous field of the first node and the next field of the last node do not point to anything, we must set it with the null value.

LinkedList in Java

4. **LinkedList** in Java is a very convenient way to store elements (data). When we store a new element in the linked list, a new node is automatically created.

5. Its size will grow with the addition of each and every element. Therefore, its initial **capacity is zero**.

6. When an element is removed, it will automatically shrink.

7. Adding elements into the **LinkedList** and removing elements from the **LinkedList** are done quickly and take the same amount of time (i.e. constant time).

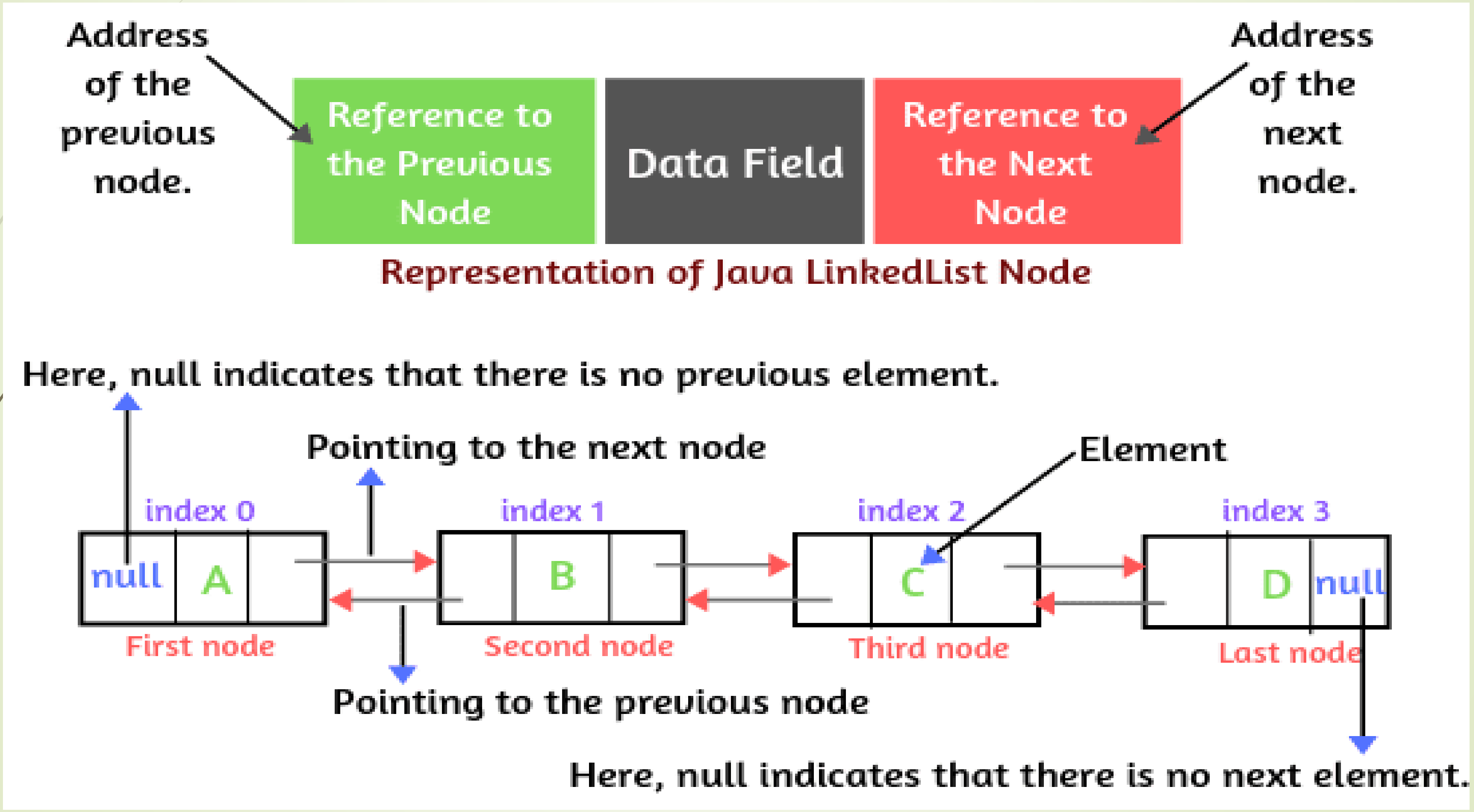
8. So, it is especially useful in situations where elements are inserted or removed from the middle of the list.

LinkedList in Java

9. In the Linked List, the elements are not stored in the consecutive memory location.

10. An element (often called node) can be located anywhere in the free space of the memory by connecting each other using the left and right sides of the node portion.

LinkedList in Java


Address of the previous node.
Reference to the Previous Node
Data Field
Reference to the Next Node
Address of the next node.
Representation of Java LinkedList Node
Here, null indicates that there is no previous element.
Pointing to the next node
Element
index 0
index 1
index 2
index 3
null
A
B
C
D
null
First node
Second node
Third node
Last node
Pointing to the previous node
Here, null indicates that there is no next element.

Java LinkedList class declaration

LinkedList is a generic class, just like ArrayList class that can be declared as:

```
class LinkedList<E>
```

Java LinkedList Constructors:

SN	Constructor	Description
1.	LinkedList()	It is used to create an empty LinkedList object.
2.	LinkedList(Collection c)	It is used to construct a list containing the elements of the given collection.

Features of LinkedList class

1. The underlying data structure of **LinkedList** is a doubly **LinkedList** data structure. It is another concrete implementation of the **List interface** like an array list.

2. Java **LinkedList** class allows storing duplicate elements.

3. **Null** elements **can** be added to the linked list.

4. **Heterogeneous** elements are allowed in the linked list.

5. **LinkedList** is the best choice if your frequent operation is insertion or deletion in the middle.

Features of LinkedList class

6. Insertion and removal of elements in the **LinkedList** are fast because, in the linked list, there is no shifting of elements after each adding and removal. The only reference for next and previous elements changed.

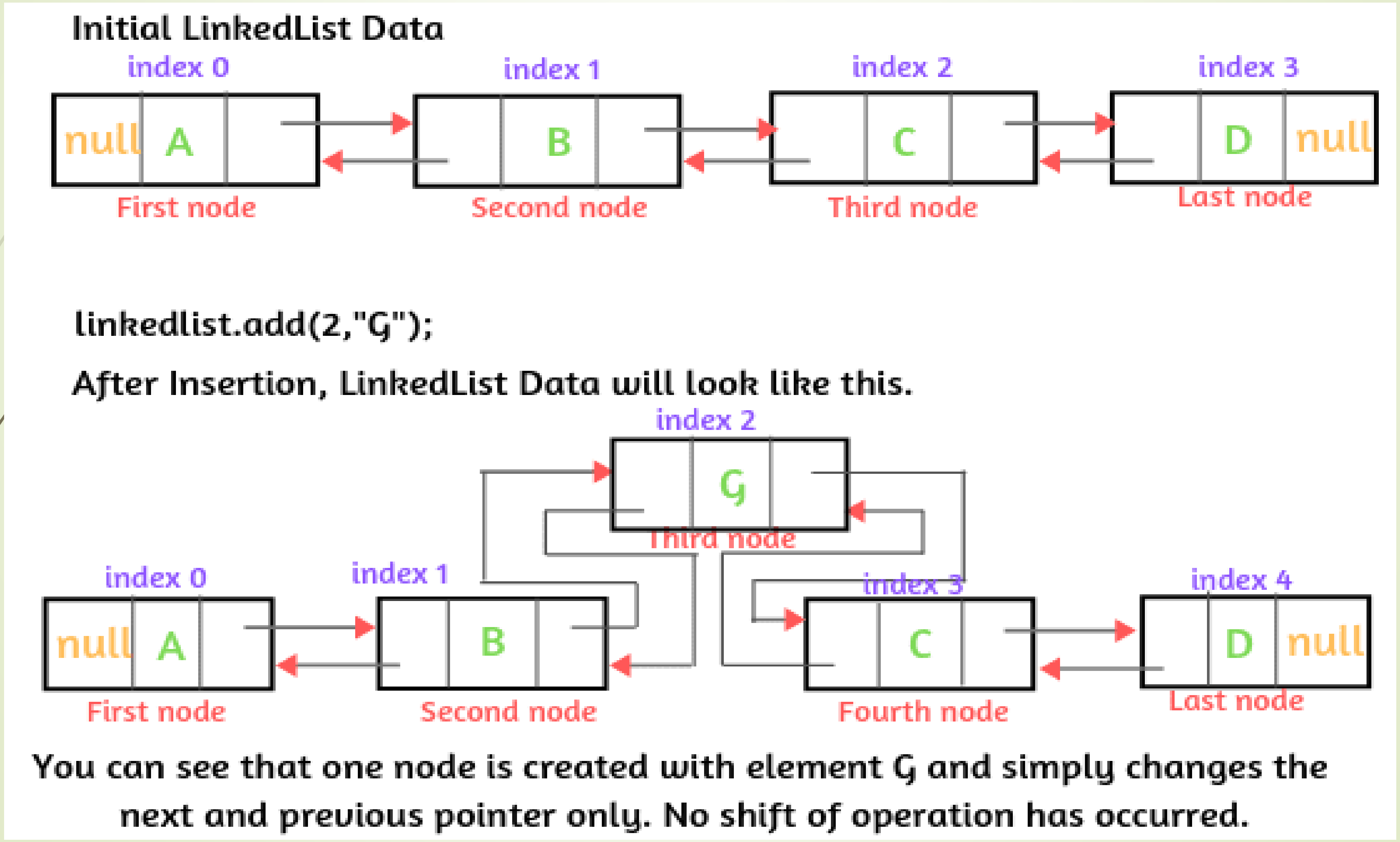
7. Retrieval (getting) of elements is very slow in **LinkedList** because it traverses from the beginning or ending to reach the element.

8. The **LinkedList** can be used as a “stack“. It has **pop()** and **push()** methods which make it function as a **stack**.

9. Java **LinkedList** does not implement random access interface. So, the element cannot be accessed (getting) randomly. To access the given element, we have to traverse from the beginning or ending to reach elements in the **LinkedList**.

10. We can iterate linked list elements by using **ListIterator**.

How does insertion work in LinkedList?

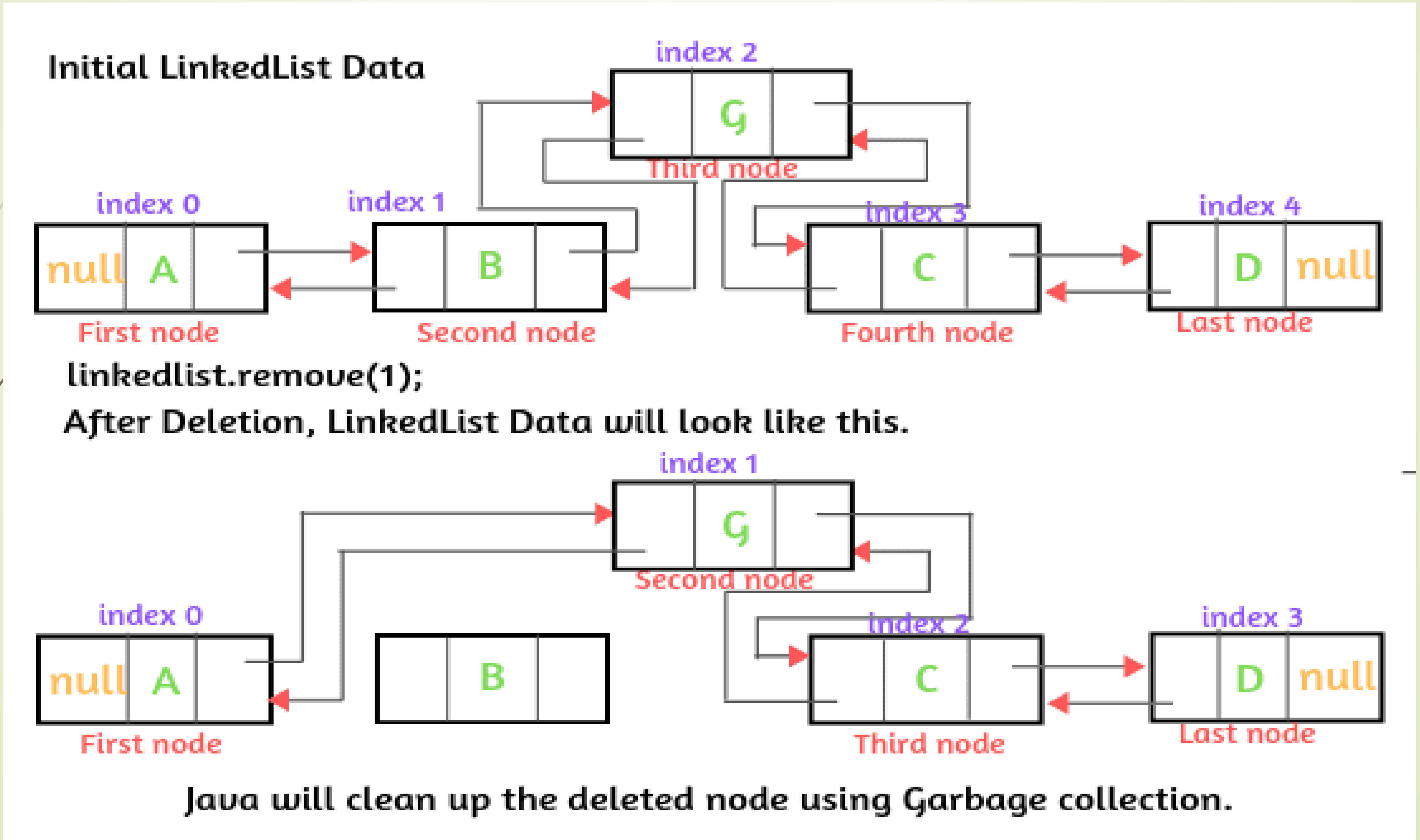

Initial LinkedList Data

linkedlist.add(2,"G");

After Insertion, LinkedList Data will look like this.

You can see that one node is created with element G and simply changes the next and previous pointer only. No shift of operation has occurred.

How does deletion work in LinkedList?



LinkedHashSet in Java

1. **LinkedHashSet** in Java is a concrete class that implements set interface and extends **HashSet** class with a doubly linked list implementation. It internally uses a linked list to store the elements in the set.

2. Java **LinkedHashSet** class is the same as **HashSet** class, except that it maintains the ordering of elements in the set in which they are inserted.

3. That is, **LinkedHashSet** not only uses a hash table for storing elements but also maintains a double-linked list of the elements in the order during iteration.

4. In other words, elements in the **HashSet** are not ordered, but elements in the **LinkedHashSet** can be retrieved in the same order in which they were inserted into the set.

Features of LinkedHashSet

Java **LinkedHashSet** class contains unique elements like **HashSet**. It does not allow to insert of duplicate elements.

LinkedHashSet class permits to insert null element.

LinkedHashSet class in Java is non-synchronized. That means it is not thread-safe.

LinkedHashSet class preserve the insertion order of elements

It is slightly slower than **HashSet**.

Linked hash set is very efficient for insertion and deletion of elements.

TreeSet in Java

A **TreeSet** in Java is another important implementation of the Set interface that is similar to the **HashSet** class, with one added improvement.

It sorts elements in ascending order while **HashSet** does not maintain any order.

Java **TreeSet** implements **SortedSet** interface. It is a collection for storing a set of unique elements (objects) according to their natural ordering.

It creates a sorted collection that uses a tree structure for the storage of elements or objects. In simple words, elements are kept in sorted, ascending order in the tree set.

For example, a set of books might be kept by height or alphabetically by title and author.

TreeSet in Java

In Java **TreeSet**, access and retrieval of elements are quite fast because of using tree structure. Therefore, **TreeSet** is an excellent choice for quick and fast access to large amounts of sorted data.

The only restriction with using tree set is that we cannot add duplicate elements in the tree set.

TreeSet is a generic class that is declared as:

class **TreeSet<T>**

Features of TreeSet class in Java

1. Java **TreeSet** contains unique elements similar to the **HashSet**. It does not allow the addition of a duplicate element.

2. The access and retrieval times are **quite fast**.

3. **TreeSet** does not allow inserting null element.

4. **TreeSet** class is non-synchronized. That means it is not thread-safe.

5. **TreeSet** maintains the ascending order. When we add elements into the collection in any order, the values are automatically presented in sorted, ascending order.

6. Java **TreeSet** internally uses a **TreeMap** for storing elements.

Which is better to use: HashSet or TreeSet?

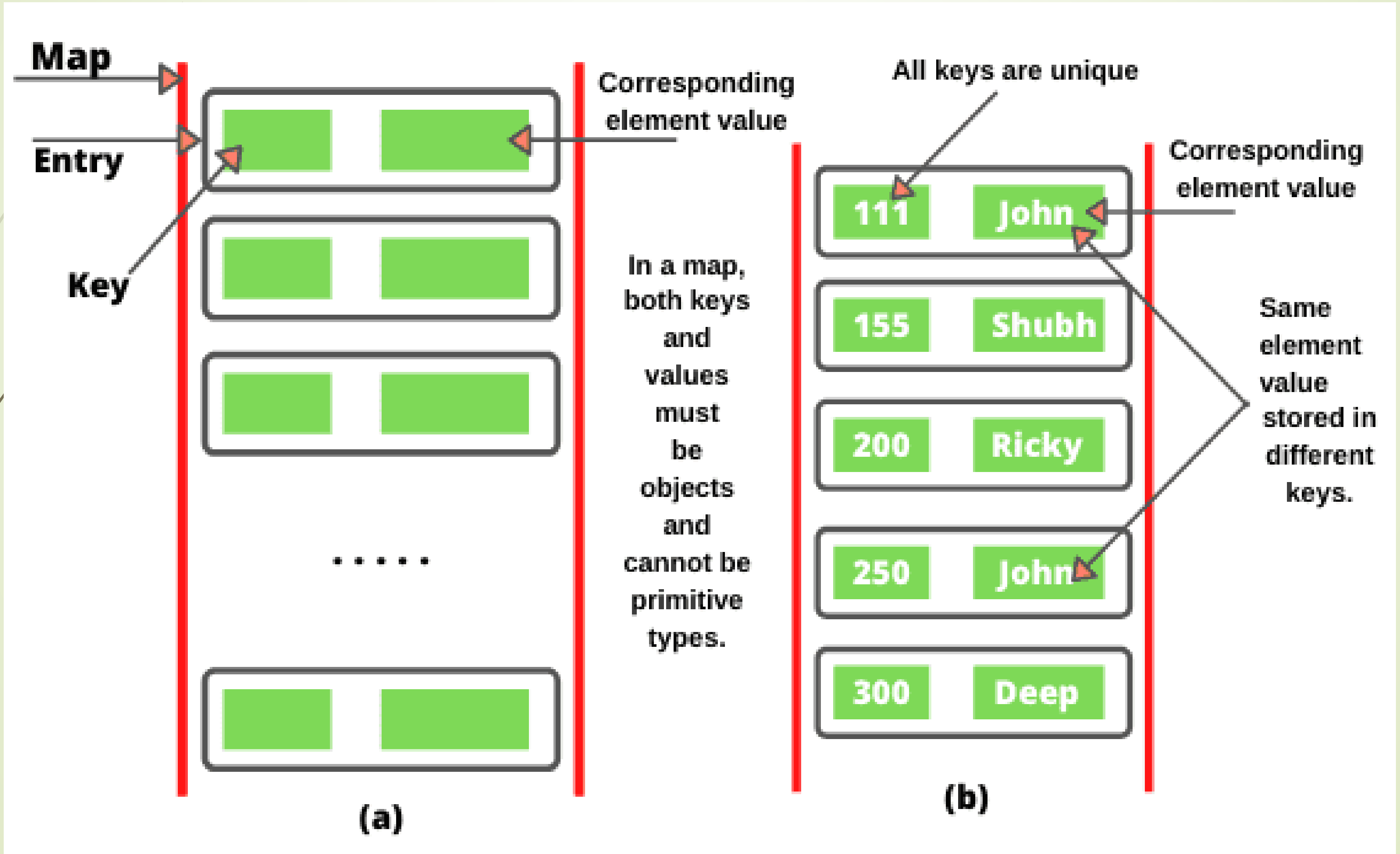
If you want to store unique elements in sorted order then use **TreeSet**, otherwise, use **HashSet** with no ordering of elements. This is because **HashSet** is much faster than **TreeSet**.

Map in Java | Methods

Map in Java

1. A **map** in Java is a container object that stores elements in the form of **key and value pairs**.
2. A **key** is a unique element (object) that serves as an "**index**" in the map.
3. The element that is associated with a key is called value. A **map** stores the values associated with **keys**.
4. In a map, both keys and values must be objects and **cannot be** primitive types.
5. A map cannot have duplicate keys. Each key maps to only one value. This type of mapping is called **one-to-one mapping** in Java.
6. All keys must be unique, but values may be duplicate (i.e. the same value can be stored to several different keys).

Map in Java


Map
Entry
Key
Corresponding element value
In a map, both keys and values must be objects and cannot be primitive types.
(a)
All keys are unique
Corresponding element value
111
John
155
Shubh
200
Ricky
250
John
300
Deep
Same element value stored in different keys.
(b)

Points to remember

The main difference between **maps** and **sets** is that maps contain keys and values, whereas **sets** contain only **keys**.

Map interface in Java is defined in **java.util.Map** package. It is a part of the Java collections framework but it does not extend the collection interface.

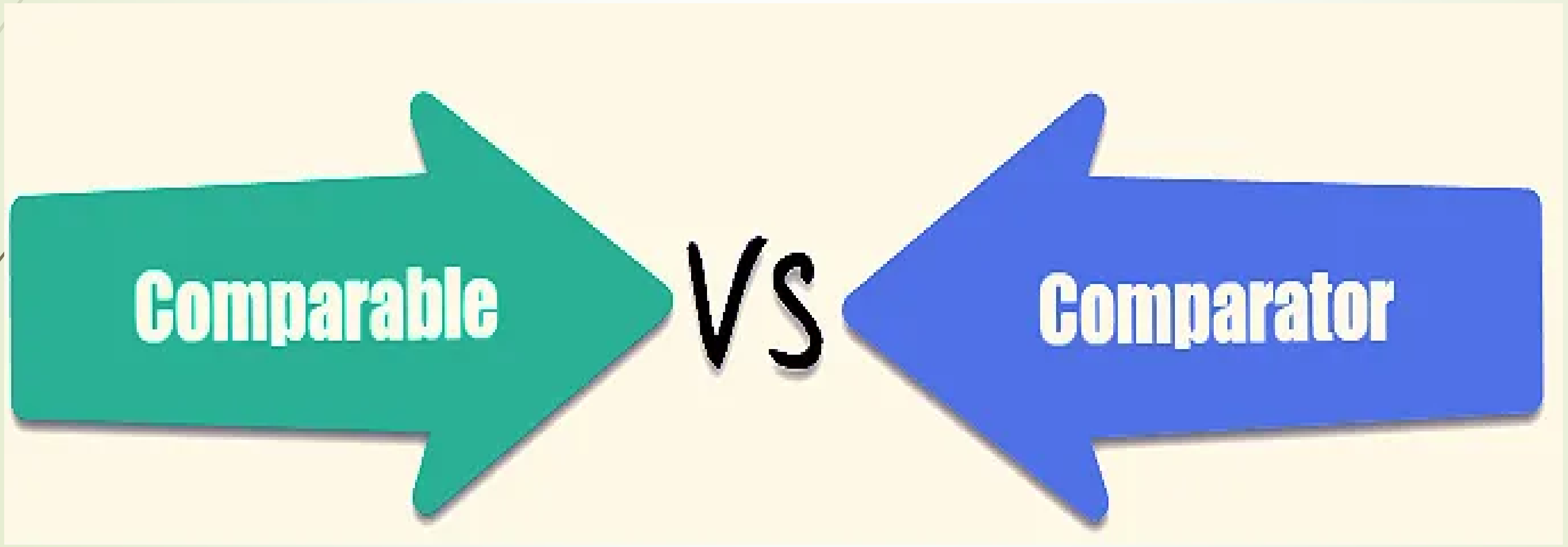
Java map interface is defined like this:

public interface Map<K, V>   // Map is a generic.

In this syntax, **K** defines the type of keys and **V** defines the type of values.

For example, a mapping of Integer keys and String values can be represented with a **Map<Integer,String>.**

Comparable and Comparator Interface



Why we need Comparable and Comparator Interface?

- Comparing primitive values like int, char, float is very easy and can be done with comparison operators like <, >, == etc.

- But comparing objects is a little different. For example, how would you compare two Employees? how would you compare two Students?

- You need to explicitly define how the objects of user-defined classes should be compared.

- For this purpose, Java provides two interfaces called Comparable and Comparator.

Comparable Interface

- The **Comparable** interface is used to define the natural ordering of objects of a class.
- This means that if a class implements the **Comparable** interface, it provides a way to compare its instances based on specific criteria or attributes.
- By default, a user-defined class is not comparable. That is, its objects can't be compared. To make an object comparable, the class must implement the **Comparable** interface.
- The **Comparable** interface has a single method called **compareTo()** that you need to implement in order to define how an object compares with the supplied object:

```
public interface Comparable<T> {
        public int compareTo(T  o);
}
```

Comparable Interface

When you define the **compareTo()** method in your classes, you need to make sure that the return value of this method is:

negative, if this object is less than the supplied object.

zero, if this object is equal to the supplied object.

positive, if this object is greater than the supplied object.

Predefined Java classes like:

String,

Date,

LocalDate,

LocalDateTime, etc

implement the **Comparable** interface to define the ordering of their instances.

Comparator Interface

- java.lang.Comparable interface provides only one way to compare this object with another one, what are the options in case we need to compare 2 objects to sort them correctly.
- How about sorting Person with a name or age? What if we want to override the natural ordering? We need to create a Comparator for these requirements.

public Interface Comparator<T>

Difference between Comparable and Comparator in Java

S.No.	Comparable	Comparator
1.	It supports single-sorting sequences. We can classify the group based on a single element.	It supports multiple sorting sequences. Here, we can classify the group based on the multiple elements.
2.	It gives **compareTo**() technique to sort elements.	It gives **compare**() technique to sort elements.
3.	It does not influence the original class.	It influence the original class.
4.	It is found in a **java.lang** container.	It is found in the **java.util** container.

When to use Comparable and Comparator interfaces in Java

Both Comparable and Comparator interfaces are used to sort the collection or array of elements (objects). But there are the following differences in use. They are as follows:

Use of Comparable interface in Java

1. Comparable interface is used for the natural sorting of objects. For example, if we want to sort Employee class by **employeeId** is natural soring.

2. **Comparable** is used to sort collection of elements based on only a single element (or logic) like name.

3. **Comparable** interface is used when we want to compare itself with another object.

When to use Comparable and Comparator interfaces in Java

Use of Comparator interface in Java

1. Comparator interface is used when we want to perform custom sorting on the collection of elements. It gives exact control over the ordering. For example, if we want to sort Employee class by name and age, it will be custom sorting.

2. Comparator is used to sort collection of elements based on multiple elements (or logics) like name, age, class, etc.

3. Comparator interface is used when we want to compare two different objects.

Vector in Java | Vector Class, Example

Vector in Java

Vector class in Java was introduced in JDK 1.0 version.

It is present in Java.util package.

It is a dynamically resizable array (growable array) which means it can grow or shrink as required.

Features of Vector class

1. The underlying Data structure for vector class is the resizable array or **growable** array.
2. Duplicate elements are allowed in the vector class.
3. It preserves the insertion order in Java.
4. Null elements are allowed in the Java vector class.
5. Heterogeneous elements are allowed in the vector class. Therefore, it can hold elements of any type and any number.
6. Most of the methods present in the vector class are synchronized. That means it is a thread-safe. Two threads cannot access the same vector object at the same time.

Features of Vector class

7. Vector class is preferred where we are developing a multi-threaded application but it gives poor performance because it is thread-safety.

8. Vector is rarely used in a non-multithreaded environment due to synchronized which gives you poor performance in searching, adding, delete, and update of its element.

9. It can be iterated by a simple for loop, **Iterator**, **ListIterator**, and **Enumeration**.

10. Vector is the best choice if the frequent operation is retrieval (getting).

Java Vector Class Constructors

Constructor	Description
vector()	It creates an empty vector with default initial capacity of 10.
vector(int initialCapacity)	It creates an empty vector with specified initial capacity.
vector(int initialCapacity, int capacityIncrement)	It creates an empty vector with specified initial capacity and capacity increment.
vector(Collection c)	It creates a vector that contains the element of collection c.

When to Use Vector?

The vector can be used when:

1. We want to store **duplicate** and **null** elements.

2. We are developing a multi-threaded application but it can reduce the performance of the application because it is "**thread-safety".**

3. **Vector** is more preferred when the retrieval of elements is more as compared to insertion and removals of elements.

4. **Vector** is a good choice if you want to access the data in the list rapidly and a poor choice if the data in the list is modified frequently.

Difference between ArrayList and Vector in Java

ArrayList	Vector
ArrayList is not synchronized.	Vector is synchronized.
Since ArrayList is not synchronized. Hence, its operation is faster as compared to vector.	Vector is slower than ArrayList.
ArrayList was introduced in JDK 2.0.	Vector was introduced in JDK 1.0.
ArrayList is created with an initial capacity of 10. Its size is increased by 50%.	Vector is created with an initial capacity of 10 but its size is increased by 100%.
In the ArrayList, Enumeration is fail-fast. Any modification in ArrayList during the iteration using Enumeration will throw **ConcurrentModificatioException**.	Enumeration is fail-safe in the vector. Any modification during the iteration using **Enumeration** will not throw any exception.

Queue in Java | Methods

Queue in Java

1. A Queue in Java is a collection of elements that implements the **"First-In-First-Out"** order. In simple words, a queue represents the arrangement of elements in the first in first out (**FIFO**) fashion.

2. That means an element that is stored as a first element into the queue will be removed first from the queue. That is, the first element is removed first and last element is removed at last.

3. Java Collection Framework added Queue interface in **Java 5.0.** It is present in **java.util.Queue**

Features of Java Queue interface

1. Java Queue interface orders elements in **First In First Out** policy.

2. Elements can be accessed and removed only from the front (head) of the queue.

3. Elements can be added only from the back (tail) of the queue.

4. Queue does not allow to add the null object.

How to Create a Queue in Java?

A Queue interface can be implemented in java by using either any one of four classes: **LinkedList** class, **AbstractQueue** class, **PriorityQueue** class, and **ArrayDeque** class.

For simple **FIFO** queues, **LinkedList** and **ArrayDeque** classes are the best choices to create a queue.

```
Queue<String> q = new LinkedList<>();

Queue<Integer> q = new ArrayDeque<>();
```

LinkedHashMap in Java | Methods

LinkedHashMap in Java

1. **LinkedHashMap** in Java is a concrete class that is **HashTable** and **LinkedList** implementation of Map interface.

2. It stores entries using a **doubly-linked list.**

3. This class extends the **HashMap** class with a **linked-list** implementation that supports an ordering of the entries in the **map**.

4. In **LinkedHashMap**, the insertion order of elements is preserved because it is based on the **Key** insertion order, that is, the order in which keys are inserted in the map.

LinkedHashMap class declaration

```
public class LinkedHashMap<K, V>
  extends HashMap<K, V>
  implements Map<K, V>
```

Here, K defines the type of keys, and V defines the type of values.

Features of LinkedHashMap in Java

1. The underlying data structure of **LinkedHashMap** is **HashTable** and **LinkedList**.
2. Java **LinkedHashMap** maintains the insertion order.
3. The entries in Java **LinkedHashMap** can be retrieved either in the order in which they were inserted into the map (known as insertion order) or in the order in which they were last accessed, from least to most recently accessed.
4. **LinkedHashMap** contains unique elements. It contains values based on keys.
5. **LinkedHashMap** allows only one null key but can have multiple null values.

Features of LinkedHashMap in Java

6. **LinkedHashMap** in Java is non synchronized. That is, multiple threads can access the same **LinkedHashMap** object simultaneously.

7. The default initial capacity of **LinkedHashMap** class is 16 with a load factor of **0.75.**

TreeMap in Java | Methods

TreeMap in Java | Methods

TreeMap in Java is a concrete class that is a red-black tree based implementation of the Map interface.

It provides an efficient way of storing key/value pairs in sorted order automatically and allows rapid retrieval. It is present in **java.util.TreeMap.**

A **TreeMap** implementation provides guaranteed log(n) time performance for **checking**, **adding**, **retrieval**, and **removal** operations.

TreeMap in Java

The two main differences between **HashMap** and **TreeMap** is that:

HashMap is an unordered collection of elements while **TreeMap** is sorted in the ascending order of its keys.

The keys are sorted either using **Comparable** interface or **Comparator** interface.

HashMap allows only one null key while **TreeMap** does not allow any null key.

TreeMap Class Declaration

TreeMap is a generic class that can be declared as follows:

public class **TreeMap<K,V>**
 extends **AbstractMap<K,V>**
 implements **NavigableMap<K, V>**, **Cloneable**, **Serializable**

Here, **K** defines the type of keys, and **V** defines the type of values.

Important Features of TreeMap

1. The underlying data structure of Java **TreeMap** is a **red-black binary search** tree.

2. **TreeMap** contains only unique elements.

3. **TreeMap** cannot have a null key but can have multiple null values.

4. Java **TreeMap** is non synchronized. That means it is not thread-safe.

5. We can create a synchronized version of map by calling **Collections.synchronizedMap()** on the map.

6. **TreeMap** in Java maintains ascending order.

Important Features of TreeMap

7. The mappings are sorted in **Treemap** either according to the natural ordering of keys or by a comparator that is provided during the object creation of **TreeMap** depending upon the constructor used.

8. Java **TreeMap** determines the order of entries by using either Comparable interface or Comparator class.

9. The iterator returned by **TreeMap** is fail-fast. That means we cannot modify map during iteration.