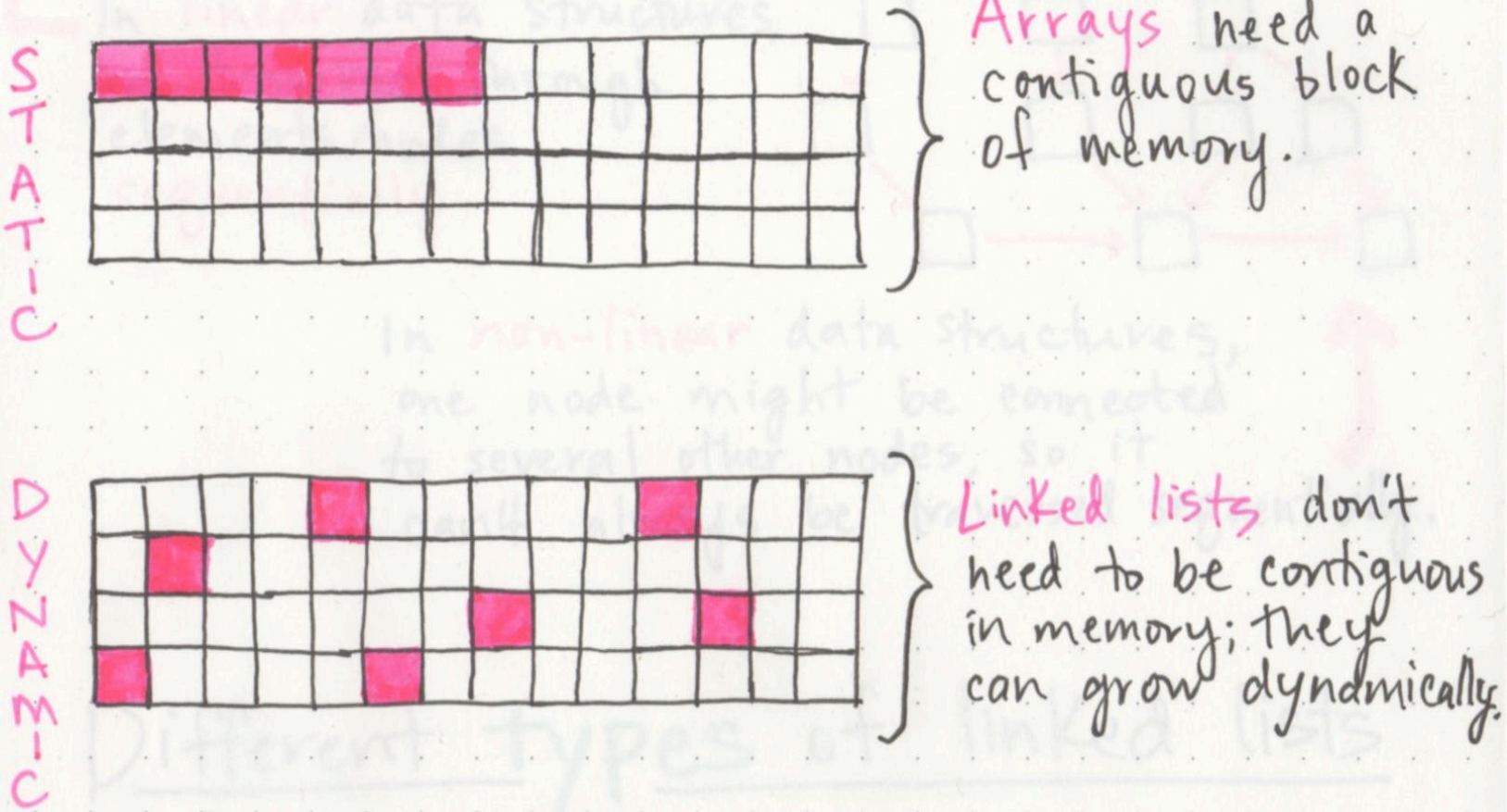
BridgeLabz

Employability Delivered

Java Core Concepts – Collections

Memory Allocation

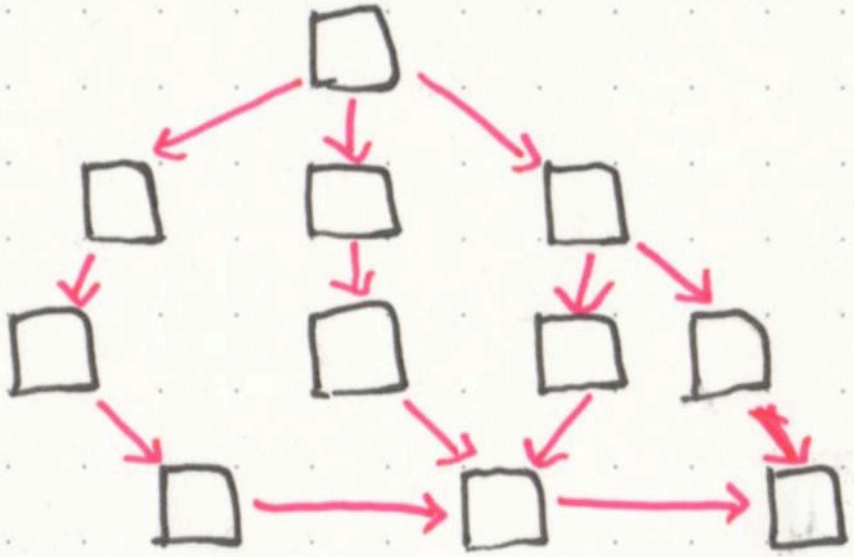


BridgeLabz
Employability Delivered

Arrays & Linked List

Linear vs. Non-linear structures

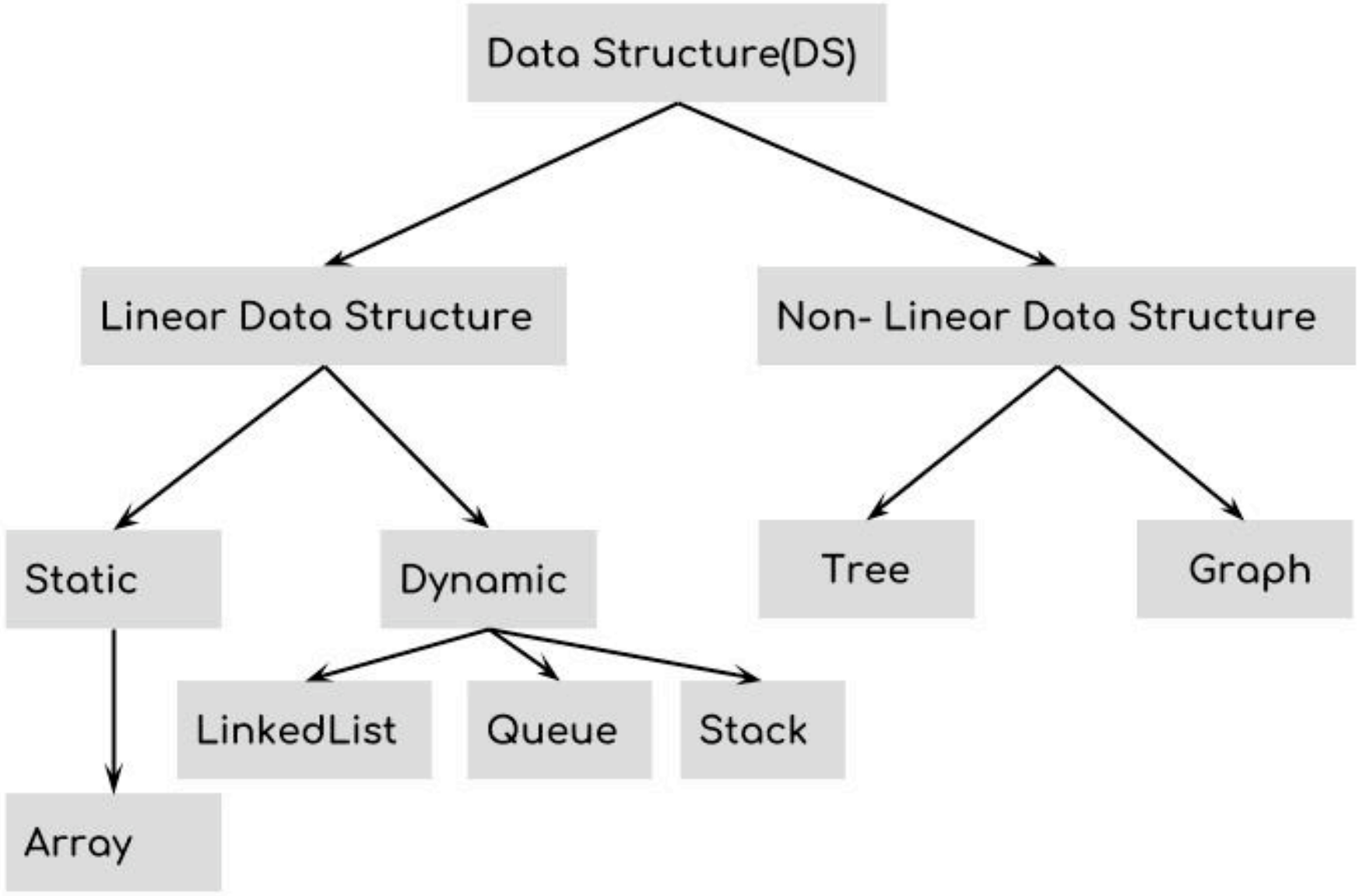
In linear data structures, we traverse through elements/nodes sequentially.



In non-linear data structures, one node might be connected to several other nodes, so it can't always be traversed sequentially.

BridgeLabz
Employability Delivered

BridgeLabz
Employability Delivered

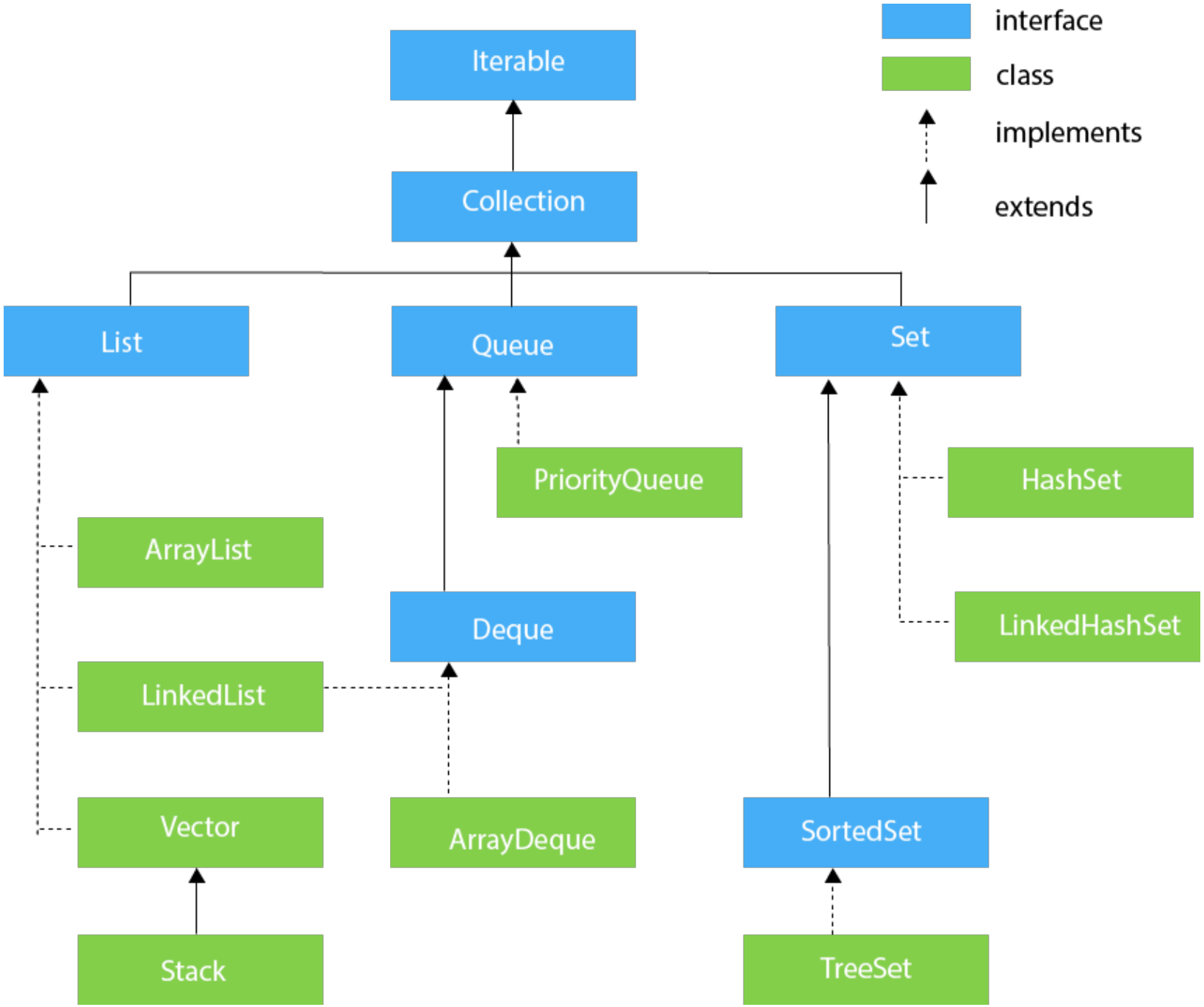


Data Structures

Java Collection Concepts

BridgeLabz
Employability Delivered

- The Collection in Java is a **framework** that provides **readymade** classes using interfaces to store and manipulate group of objects.
- Java Collections can achieve all the operations that you perform on a data such as **searching, sorting, insertion, manipulation, and deletion**.
- Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (**ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet**).Java **Generics**



BridgeLabz
Employability Delivered

Collection Framework

BridgeLabz
Employability Delivered

Methods of Collection interface

No.	Method	Description
1	public boolean add(E e)	It is used to insert an element in this collection.
2	public boolean addAll(Collection<? extends E> c)	It is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	It is used to delete an element from the collection.
4	public boolean removeAll(Collection<?> c)	It is used to delete all the elements of the specified collection from the invoking collection.
5	default boolean removeIf(Predicate<? super E> filter)	It is used to delete all the elements of the collection that satisfy the specified predicate.
6	public boolean retainAll(Collection<?> c)	It is used to delete all the elements of invoking collection except the specified collection.
7	public int size()	It returns the total number of elements in the collection.
8	public void clear()	It removes the total number of elements from the collection.
9	public boolean contains(Object element)	It is used to search an element.
10	public boolean containsAll(Collection<?> c)	It is used to search the specified collection in the collection.

BridgeLabz
Employability Delivered

Methods of Collection interface

No.	Method	Description
11	public Iterator iterator()	It returns an iterator.
12	public Object[] toArray()	It converts collection into array.
13	public <T> T[] toArray(T[] a)	It converts collection into array. Here, the runtime type of the returned array is that of the specified array.
14	public boolean isEmpty()	It checks if collection is empty.
15	default Stream<E> parallelStream()	It returns a possibly parallel Stream with the collection as its source.
16	default Stream<E> stream()	It returns a sequential Stream with the collection as its source.
17	default Spliterator<E> spliterator()	It generates a Spliterator over the specified elements in the collection.
18	public boolean equals(Object element)	It matches two collections.
19	public int hashCode()	It returns the hash code number of the collection.

Concrete List Classes

BridgeLabz
Employability Delivered

- **ArrayList –** Uses a **dynamic array** to store the **duplicate** element of **different data types**. The ArrayList class maintains the **insertion** order and is **non-synchronized**. The elements stored in the ArrayList class can be randomly **accessed** and is fast.
- **LinkedList –** Uses a **doubly linked list** internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the **manipulation** is fast because no shifting is required..
- **Vector –** Uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is **synchronized** and contains many methods that are not the part of Collection framework.
- **Stack –** Is the **subclass** of Vector. It implements the **last-in-first-out data structure**, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean **push**(), boolean **peek**(), boolean **push**(object o), which defines its properties.

Concrete Queue and Set Classes

BridgeLabz
Employability Delivered

- **Queue –** Implementing Classes like **PriorityQueue** maintain the **first-in-first-out** order along with priorities. Further In **Deque**, we can remove and add the elements from both the side. Deque stands for a **double-ended queue** which enables us to perform the operations at both the ends. **ArrayDeque** implements Deque interface.
- **Set –** Represents the unordered set of elements which doesn't allow to store the **duplicate** items. We can store at most one null value in Set. Set is implemented by
  - **HashSet** – uses Hashing to store elements,
  - **LinkedHashSet** – maintains **insertion** order and permits null elements, and
  - **TreeSet** – uses tree for storage, the **access and retrieval** time of TreeSet is quite fast because the elements in TreeSet stored in **ascending order**..

Map

BridgeLabz
Employability Delivered

- **Map –** Contains values on the basis of key, i.e. **key and value pair**. Each key and value pair is known as an **entry**. A Map contains unique **keys**. Map is like a **Dictionary**.
- A Map is useful if you have to search, update or delete elements on the basis of a key.
- There are two interfaces for implementing Map in java: **Map** and **SortedMap**, and three classes: **HashMap**, **LinkedHashMap**, and **TreeMap**.
  - HashMap – does not maintain any order,
  - LinkedHashMap – It inherits HashMap class. It maintains insertion order. and
  - TreeMap – It is the implementation of Map and SortedMap. It maintains ascending order...

Method	Description
V put(Object key, Object value)	It is used to insert an entry in the map.
void putAll(Map map)	It is used to insert the specified map in the map.
V putIfAbsent(K key, V value)	It inserts the specified value with the specified key in the map only if it is not already specified.
V remove(Object key)	It is used to delete an entry for the specified key.
boolean remove(Object key, Object value)	It removes the specified values with the associated specified keys from the map.
Set keySet()	It returns the Set view containing all the keys.
Set<Map.Entry<K,V>> entrySet()	It returns the Set view containing all the keys and values.
void clear()	It is used to reset the map.
V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)	It is used to compute its value using the given mapping function, if the specified key is not already associated with a value (or is mapped to null), and enters it into this map unless null.
V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a new mapping given the key and its current mapped value if the value for the specified key is present and non-null.
boolean containsValue(Object value)	This method returns true if some value equal to the value exists within the map, else return false.

Methods of Map interface

Method	Description
boolean containsKey(Object key)	This method returns true if some key equal to the key exists within the map, else return false.
boolean equals(Object o)	It is used to compare the specified Object with the Map.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
V get(Object key)	This method returns the object that contains the value associated with the key.
V getOrDefault(Object key, V defaultValue)	It returns the value to which the specified key is mapped, or defaultValue if the map contains no mapping for the key.
int hashCode()	It returns the hash code value for the Map
boolean isEmpty()	This method returns true if the map is empty; returns false if it contains at least one key.
V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)	If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
V replace(K key, V value)	It replaces the specified value for a specified key.
boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a specified key.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
Collection values()	It returns a collection view of the values contained in the map.
int size()	This method returns the number of entries in the map.

Methods of Map interface

```java
package javashowcase;

import java.util.*;

public class JavaCollectionDemo {

    public static void main(String[] args) {
        doListDemo();
        doStackDemo();
        doQueueDemo();
        doSetDemo();
        doMapDemo();
    }

    private static void doListDemo() {...}

    private static void doStackDemo() {...}

    private static void doQueueDemo() {...}

    private static void doSetDemo() {...}

    private static void doMapDemo() {...}

}
```

```java
    private static void doListDemo() {
        System.out.println("In doListDemo");
        //Creating List
        List<String> list = new LinkedList<~>();
        //Adding object to the list
        list.add("Ravi");
        list.add("Vijay");
        list.add("Ravi");
        list.add("Ajay");

        //Traversing list through Iterator
        Iterator itr=list.iterator();
        while(itr.hasNext()){...}
    }
```

```java
    private static void doStackDemo() {
        System.out.println("\nIn doStackDemo");
        Stack<String> stack = new Stack<~>();
        stack.push( item: "Ayush");
        stack.push( item: "Garvit");
        stack.push( item: "Amit");
        stack.push( item: "Ashish");
        stack.push( item: "Garima");
        String pop = stack.pop();
        Iterator<String> itr=stack.iterator();
        while(itr.hasNext()){...}
    }
```

```java
private static void doQueueDemo() {
    System.out.println("\nIn doQueueDemo");
    PriorityQueue<String> queue = new PriorityQueue<>();
    queue.add("Amit Sharma");
    queue.add("Vijay Raj");
    queue.add("JaiShankar");
    queue.add("Raj");
    System.out.println("head:"+queue.element());
    System.out.println("head:"+queue.peek());
    System.out.println("iterating the queue elements:");
    Iterator itr=queue.iterator();
    while(itr.hasNext()){...}
    queue.remove();
    queue.poll();
    System.out.println("after removing two elements:");
    Iterator<String> itr2=queue.iterator();
    while(itr2.hasNext()){...}
}
```

```java
private static void doSetDemo() {
    System.out.println("\nIn doSetDemo");
    Set<String> set = new LinkedHashSet<>();
    set.add("Ravi");
    set.add("Vijay");
    set.add("Ravi");
    set.add("Ajay");
    for (String str : set) {...}
}
```

```java
private static void doMapDemo() {
    Map<Integer,String> map=new HashMap<>();
    map.put(100,"Amit");
    map.put(101,"Vijay");
    map.put(102,"Rahul");
    //Elements can traverse in any order
    for(Map.Entry m:map.entrySet()){
        System.out.println(m.getKey()+" "+m.getValue());
    }
}
```

BridgeLabz

Employability Delivered

Thank You