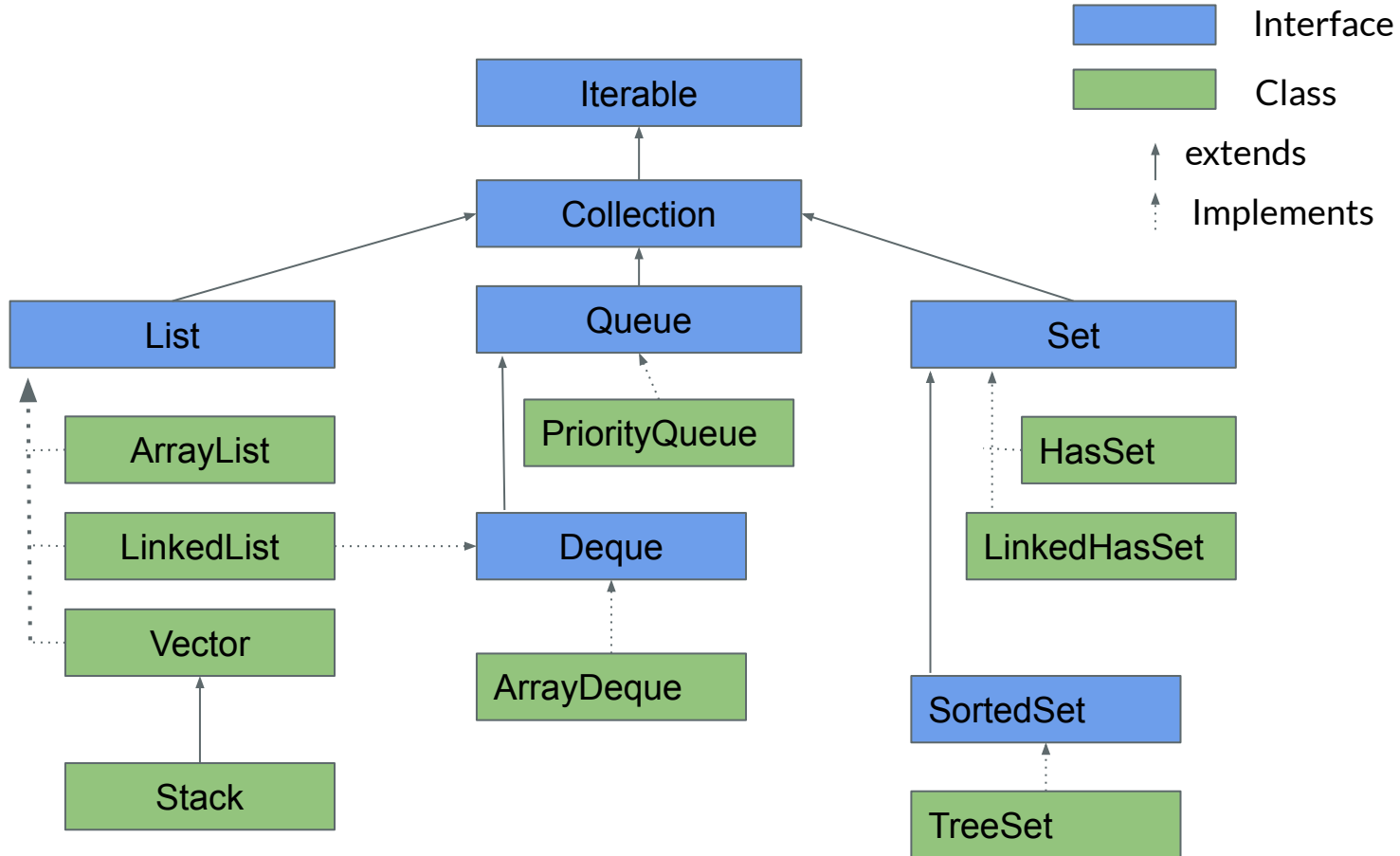# Collections

Dhvani Undhad

Collection framework

-   The JAVA collection framework is a collection of Interfaces and classes of helps in storing and processing the data efficiently.
-   This framework has several useful classes which have lots of useful functions which makes a program easy to code.
-   Basically we can think of a collections is a framework that provides an architecture to store and manipulate the group of object.
-   Java collection means a single unit of objects.
-   It represent a set of Interfaces and classes.

Hierarchy of Collection Framework

Collection – List

- A List is an ordered collection.
- May contains duplicate elements.
- Elements can be inserted or accessed by their position (Index based)in the list.
- List interface is implemented by the classes ArrayList, LinkedList, Vector and Stack.

To instantiate the List Interface:

List <data-type> list1= **new** ArrayList();

List <data-type> list2 = **new** LinkedList();

List <data-type> list3 = **new** Vector();

List <data-type> list4 = **new** Stack();

ArrayList

- The ArrayList implements the List Interface.

- It uses a Dynamic array to store duplicate element of different data types.

- The ArrayList maintains the insertion order and is non-synchronized.

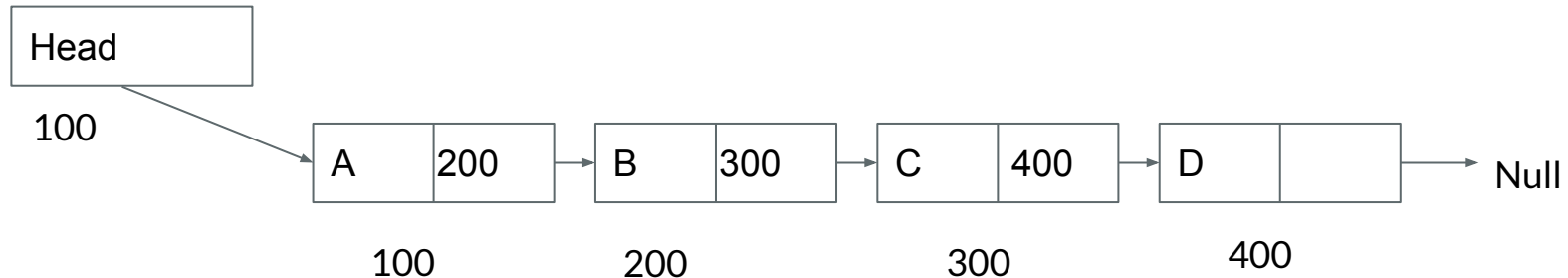- The elements stored in ArrayList can be randomly accessed.

  For Example:

```java
class Collection1{

public static void main(String args[]){

ArrayList<String> list=new ArrayList<String>();//Creating arraylist

list.add("Dhvani");//Adding object in arraylist

list.add("Meena");

list.add("Manasvi");

list.add("Vineet");

//Traversing list through Iterator

Iterator itr=list.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}
```

Output: Dhvani
        Meena
        Manasvi
        Vineet

LinkedList

- LinkedList implements the collection Interface.
- There are two types of LinkedList : 1) Singly LinkedList 2) Doubly LinkedList
- It uses a doubly LinkedList to store an elements internally.
- It can store duplicate elements.
- It maintains the insertion order and is not synchronized.
- Manipulation is fast and easy in LinkedList because no shifting is required.

```
┌─────────┐
│ Head    │
└─────────┘
   100
          ┌───┬─────┐   ┌───┬─────┐   ┌───┬─────┐   ┌───┬─────┐
          │ A │ 200 │ → │ B │ 300 │ → │ C │ 400 │ → │ D │     │ → Null
          └───┴─────┘   └───┴─────┘   └───┴─────┘   └───┴─────┘
            100           200           300           400
```

For Example:

```java
public class LinkedList1 {

    public static void main(String args[])
    {
        LinkedList<String> ll = new LinkedList<>();

        ll.add("Dhvani");
        ll.add("Undhad");
        ll.add(1, "Lalitbhai");

        System.out.println(ll);
    }
}
```

Output:
[Dhvani, Lalitbhai, Undhad]

Collection – Set

- A set is an interface and extends collection which do not contains duplicate elements.
- We can store at least one Null value in Set except TreeSet.
- There are three main implementation of Set:

1) HashSet: Stores it's elements in Hash table and does not maintains order.

2) TreeSet : Stores elements based on their values.

3) LinkedHasSet : Maintains orders of it's elements in which they were inserted into Set.

-

HashSet

Methods in HashSet:

- **boolean add(Element e)**: It adds the element e to the list.
- **void clear()**: It removes all the elements from the list.
- **Object clone()**: This method returns a shallow copy of the HashSet.
- **boolean contains(Object o)**: It checks whether the specified Object o is present in the list or not. If the object has been found it returns true else false.
- **boolean isEmpty()**: Returns true if there is no element present in the Set.
- **int size()**: It gives the number of elements of a Set.
- **boolean(Object o)**: It removes the specified Object o from the Set.

Set can be instantiated as:

1. Set<data-type> s1 = new HashSet<data-type>();

2. Set<data-type> s2 = new LinkedHashSet<data-type>();

3. Set<data-type> s3 = new TreeSet<data-type>();

For Example:

```java
1.  public class CollectionTest{
2.  public static void main(String args[]){
3.  Set<String> set=new HashSet<String>(); //Creating HashSet and adding duplicate elements
4.  set.add("Dhvani");
5.  set.add("Vijay");
6.  set.add("Dhvani");
7.  Iterator<String> itr=set.iterator();  //Traversing elements
8.  while(itr.hasNext()){
9.  System.out.println(itr.next());
10. }
11. }
12. }
```

Output: Vijay
        Dhvani

TreeSet

- It's same as HashSet except it's sorts it's elements in the ascending order while HashSet doesn't.
- It allows null element.

For Example:

```
1.    TreeSet<String> al=new TreeSet<String>();
2.     al.add("Zayad");
3.     al.add("Shrutil");
4.     al.add("Coby");
5.     //Traversing elements
6.     Iterator<String> itr=al.iterator();
7.     while(itr.hasNext()){
8.      System.out.println(itr.next());
9.     }
```

Output:

Coby
Shruti
Zayad

LinkedHashSet

- It contains only unique elements only like HashSet.
- Maintains the insertion order and is non synchronized.
- It allows Null elements.

For Example:

LinkedHashSet<String> set=**new** LinkedHashSet<String>(); //Creating HashSet and adding duplicate elements
1. set.add("Dhvani");
2. set.add("Vijay");
3. set.add("Dhvani");
4. set.add(Null);
5. Iterator<String> itr=set.iterator(); //Traversing elements
6. **while**(itr.hasNext()){
7. System.out.println(itr.next());
8. }

Output:
Dhvani
Vijay
Dhvani
Null

Collection – Map

- A map contains values on the basis of key. For example Key and Value pair.
- A map contains unique Key.
- A map can not be traversed like ArrayList and HashSet through Iterator, so you need to convert it into Set using KeySet() and entrySet() method.

Class	Description
HashMap	**HashMap is the implementation of Map, but it doesn't maintain any order.**
LinkedHashMap	**LinkedHasMap is implementation of Map. It inherits HashMap class. It maintains insertion order.**
TreeMap	**TreeMap is the implementation of Map and SortedMap. It maintains ascending order.**

HashMap

For Example:

```java
HashMap<Integer, String> hmap = new HashMap<Integer, String>();

        hmap.put(12, "Company");
        hmap.put(2, "Managers");
        hmap.put(7, "Customers");
        hmap.put(49, "Employee");
        hmap.put(3, "Headquarters");

        //Converting HashMap into Set.
        Set set = hmap.entrySet();
        Iterator iterator = set.iterator();

        while(iterator.hasNext()) {
            Map.Entry mapData = (Map.Entry) iterator.next();
            System.out.println("Key: " + mapData.getKey() + " ,Value: " + mapData.getValue());
        }
```

Output:
Key: 49 ,Value: Employee
Key: 2 ,Value: Managers
Key: 3 ,Value: Headquarters
Key: 7 ,Value: Customers
Key: 12 ,Value: Company

Hashtable

- HashMap allows one null key and any number of null values, but Hashtable doesn't allow null keys and null values.
- It will gives you an NullPointerException.

  For example:

```
Hashtable<String , Integer> table = new Hashtable<>();
        //Hashtable does not allow null Key or values and throws Null Pointer during execution
        try {
                table.put(null, 5);
                table.put("Dhvani", 6);
                table.put("Neha", null);
        }
        catch(NullPointerException e){
                System.out.println("Hashtable does not allow null key or values");
        }
        System.out.println("Hashtable: "+table.get("Neha"));
```

Output:
Hashtable does not allow null key or values
Hashtable: null

Collection – Queue and Deque

- Queue is an Interface in java which extends Collection Interface.
- It is an ordered list and in Queue elements retrieved as FIFO (First In First Out) orders.

Queue

Retrieve here

Head

| 1 | 2 | 3 | 4 |

Tail

Add here
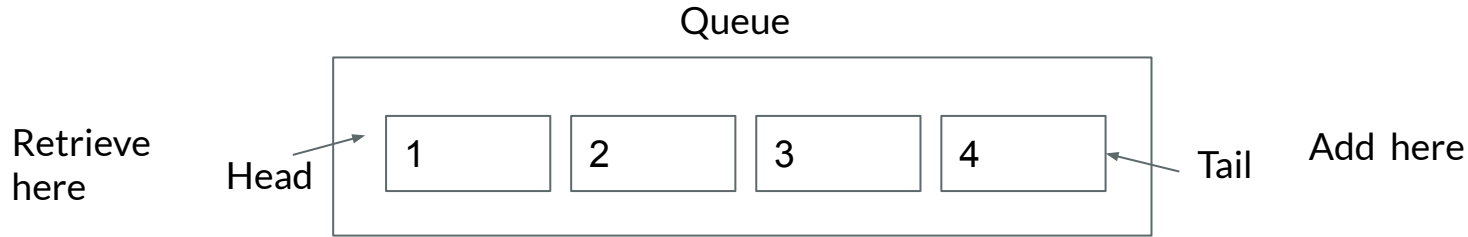
- Where as Deque Interface is subtype of Queue Interface and it is double- ended Queue.
- Deque uses LIFO(Last In First Out) or Stack Data Structure.
- It supports addition or removal of elements from both the sides.

Add element

First

| 1 | 2 | 3 | 4 | 5 |

Last

Add element

Remove element

Remove element