Collection Classes

Java Collections

The **Collection in Java** is a framework that provides an architecture to store and manipulate the 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 means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList PriorityQueue, LinkedHashSet, TreeSet).

Java Collections

- **What is a framework in Java**
- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.

Java Collections

- **What is Collection framework**
- The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1.Interfaces and its implementations, i.e., classes

2.Algorithm

Algorithms are another important part of the collection mechanism.

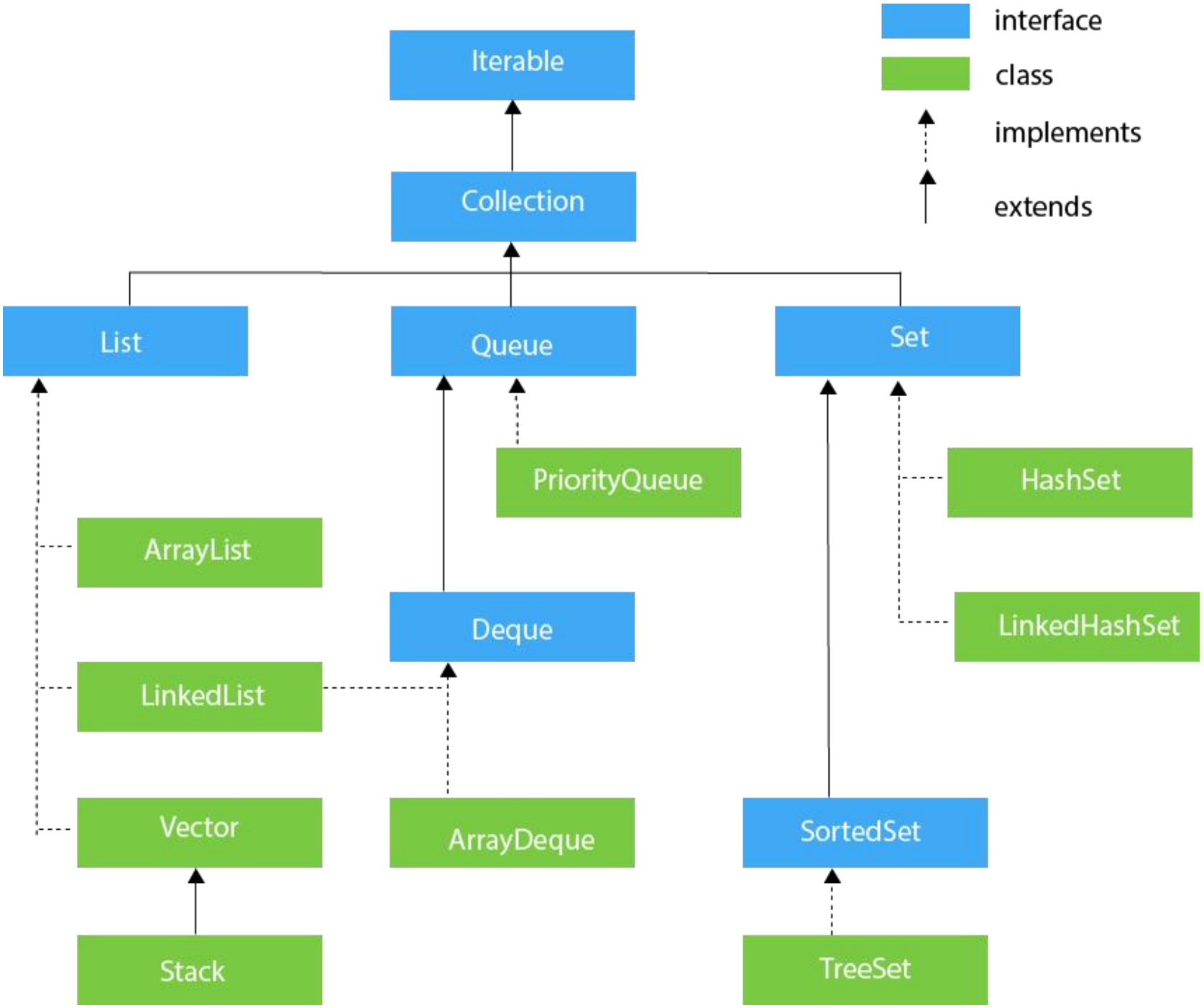
Algorithms operate on collections and are defined as static methods within the Collections class.

Thus, they are available for all collections.

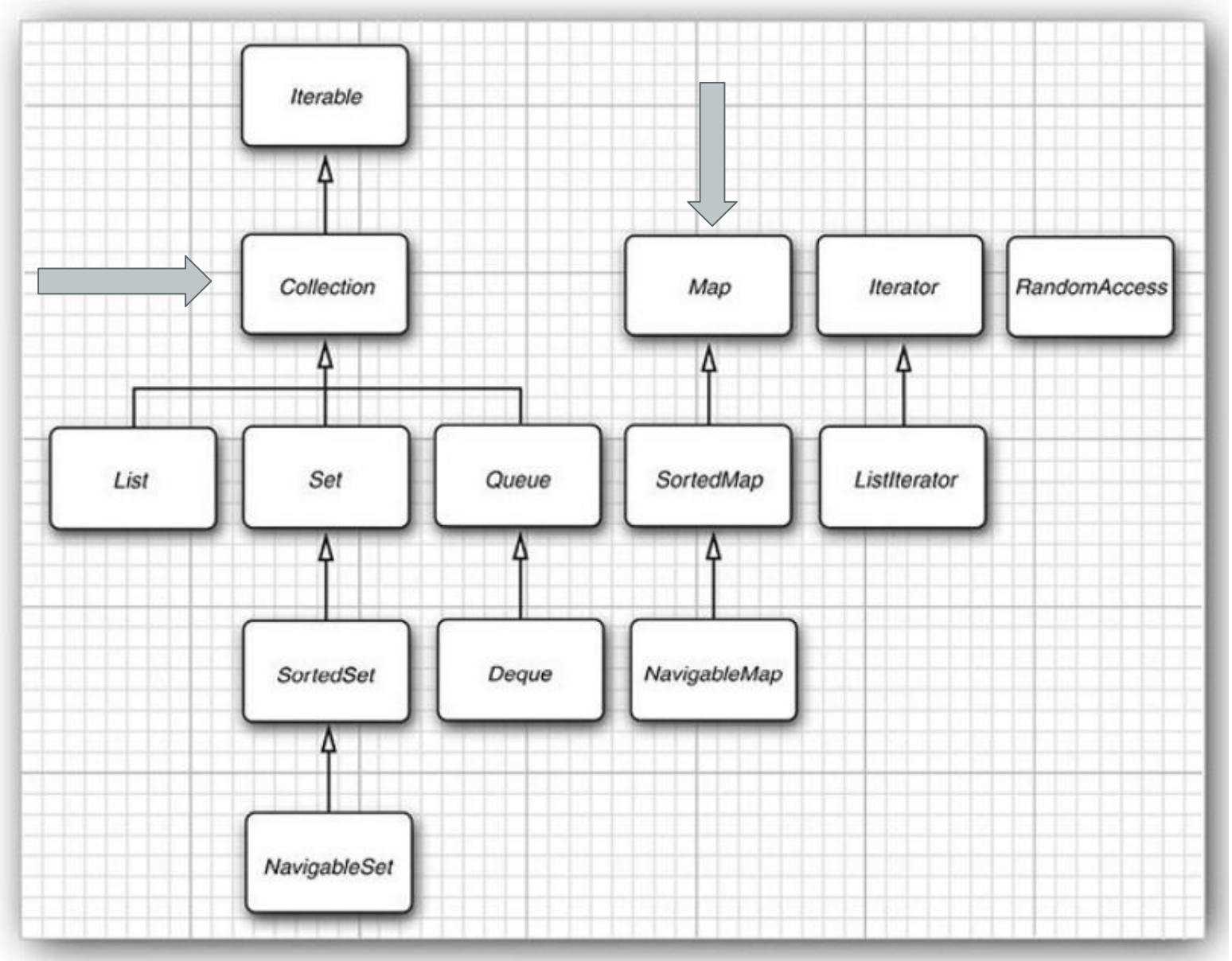
Each collection class need not implement its own

Java Collections

- The algorithms provide a standard means of manipulating collections
- **Hierarchy of Collection Framework**
- Let us see the hierarchy of Collection framework.
- The **java.util** package contains all the classes and interfaces for the Collection framework.


interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

Interfaces in Collection



Java Collections

- The Collection interface is the interface which is implemented by all the classes in the collection framework.
- It declares the methods that every collection will have.
- In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.

- **Methods of Collection interface**
- There are many methods declared in the Collection interface. They are as follows:

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.
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.

Iterator interface

- Iterator interface provides the facility of iterating the elements in a forward direction only.

Iterable Interface

- The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.
- It contains only one abstract method. i.e.,
- Iterator<T> iterator()
- It returns the iterator over the elements of type T.

Java List

- **List** in Java provides the facility to maintain the *ordered collection*.
- It contains the index-based methods to insert, update, delete and search the elements.
- It can have the duplicate elements also.
- We can also store the null elements in the list.
- The List interface is found in the java.util package and inherits the Collection interface.
- The implementation classes of List interface are ArrayList, LinkedList, Stack and Vector
- . The ArrayList and LinkedList are widely used in Java programming.

Java List Method

void add(int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean add(E e)	It is used to append the specified element at the end of a list.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of a list.
boolean addAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void clear()	It is used to remove all of the elements from this list.
boolean equals(Object o)	It is used to compare the specified object with the elements of a list.

int hashcode()	It is used to return the hash code value for a list.
E get(int index)	It is used to fetch the element from the particular position of the list.
boolean isEmpty()	It returns true if the list is empty, otherwise false.
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
<T> T[] toArray(T[] a)	It is used to return an array containing all of the elements in this list in the correct order.

How to create List

The ArrayList and LinkedList classes provide the implementation of List interface.
examples to create the List:

```
//Creating a List of type
String using ArrayList
List<String> list=new
ArrayList<String>();

//Creating a List of type
Integer using ArrayList
List<Integer> list=new
ArrayList<Integer>();
```

List Example

```
import java.util.*;
public class ListExample1{
public static void main(String args[]){
 //Creating a List
 List<String> list=new ArrayList<String>();
 //Adding elements in the List
 list.add("Mango");
 list.add("Apple");
 list.add("Banana");
 list.add("Grapes");
 //Iterating the List element using for-each loop
 for(String fruit:list)
  System.out.println(fruit);

}
}
```

```
Mango
Apple
Banana
Grapes
Press any key to continue . . .
```

Get and Set Element in List

```
import java.util.*;
public class ListExample2{
 public static void main(String args[]){
 //Creating a List
 List<String> list=new ArrayList<String>();
 //Adding elements in the List
 list.add("Mango");
 list.add("Apple");
 list.add("Banana");
 list.add("Grapes");
 //accessing the element
 System.out.println("Returning element: "+list.get(1));
 //changing the element
 list.set(1,"Dates");
 //Iterating the List element using for-each loop
 for(String fruit:list)
  System.out.println(fruit);

 }
}
```

```
Returning element: Apple
Mango
Dates
Banana
Grapes
Press any key to continue . . .
```

```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class ListExample5 {
public static void main(String[] args) {
    //Creating list of Books
    List<Book> list=new ArrayList<Book>();
    //Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
    Book b2=new Book(102,"Data Communications and Networking","Forouzan","Mc Graw Hill",4);
    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
    //Adding Books to list
    list.add(b1);
    list.add(b2);
    list.add(b3);
    //Traversing list
    for(Book b:list){
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
    }
}
}
```

Java ArrayList

- Java ArrayList class uses a dynamic array for storing the elements. It is like an array, but there is no size limit.
- We can add or remove elements anytime. So, it is much more flexible than the traditional array.
- It is found in the java.util package.
- The ArrayList in Java can have the duplicate elements also.
- ArrayList is a generic class that has this declaration: class ArrayList<E>
- Here, E specifies the type of objects that the list will

The important points about the Java ArrayList class are:

- It implements the List interface so we can use all the methods of the List interface here.
- Java ArrayList class maintains insertion order
- Java ArrayList class is non synchronized
- Java ArrayList allows random access because the array works on an index basis.
- In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.

- **We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases. For example:**

Method of ArrayList

Method	Description
add(int index, Object element)	This method is used to insert a specific element at a specific position index in a list.
add(Object o)	This method is used to append a specific element to the end of a list.
addAll(Collection C)	This method is used to append all the elements from a specific collection to the end of the mentioned list, in such an order that the values are returned by

clear()	This method is used to remove all the elements from any list.
clone()	This method is used to return a shallow copy of an ArrayList in Java.
contains? (Object o)	Returns true if this list contains the specified element.
get?(int index)	Returns the element at the specified position in this list.
indexOf(Object O)	The index the first occurrence of a specific element is either returned

remove?(int index)	Removes the element at the specified position in this list.
remove? (Object o)	Removes the first occurrence of the specified element from this list, if it is present.
removeIf?(Predicate filter)	Removes all of the elements of this collection that satisfy the given predicate.

toArray()	This method is used to return an array containing all of the elements in the list in the correct order.
toArray(Object[] O)	It is also used to return an array containing all of the elements in this list in the correct order same as the previous method

Key Point of ArrayList

- ArrayList Duplicates Are Allowed.
- Insertion Order is Preserved.
- Heterogeneous objects are allowed.
- Null insertion is possible.

Java ArrayList

```
import java.util.*;
 public class ArrayListExample1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
       list.add("Mango");//Adding object in arraylist
       list.add("Apple");
       list.add("Banana");
       list.add("Grapes");
       //Printing the arraylist object
       System.out.println(list);
 }
}
```

ArrayListDemo.java

Iterating ArrayList using Iterator

```
import java.util.*;
public class ArrayListExample2{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  list.add("Mango");//Adding object in arraylist
  list.add("Apple");
  list.add("Banana");
  list.add("Grapes");
  //Traversing list through Iterator
  Iterator itr=list.iterator();//getting the Iterator
  while(itr.hasNext()){//check if iterator has the elements
   System.out.println(itr.next());//printing the element and move to next
  }
 }
}
```

Java LinkList

- LinkedList is a generic class that has this declaration: class LinkedList<E>
- Here, E specifies the type of objects that the list will hold.
- LinkedList has the two constructors

  LinkedList( )

  LinkedList(Collection c)
- The first constructor builds an empty linked list.

The second constructor builds a linked list that is initialized with the elements of the collection c.

- Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure

Java LinkList

- Java LinkedList class can contain duplicate elements
- Java LinkedList class maintains insertion order
- Java LinkedList class can be used as a list, stack or queue.
- LinkedList implements the Deque interface, so it have access to the methods defined by Deque.
- For example, to add elements to the start of a list, you can use addFirst( ) or offerFirst( ).
- To add elements to the end of the list, use addLast( ) or offerLast( ).
- To obtain the first element, you can use getFirst( ) or
- peekFirst( ).

- To remove the first element, use removeFirst( ) or pollFirst( ).
- To remove the last element, use removeLast( ) or pollLast( ).

addFirst(E e)	This method Inserts the specified element at the beginning of this list.
addLast(E e)	This method Appends the specified element to the end of this list.

clear()	This method removes all of the elements from this list.
clone()	This method returns a shallow copy of this LinkedList.
contains(Object o)	This method returns true if this list contains the specified element.
element()	This method retrieves but does not remove, the head (first element) of this list.
get(int index)	This method returns the element at the specified position in this list.
getFirst()	This method returns the first element in this list.
getLast()	This method returns the last element

pop()	This method Pops an element from the stack represented by this list.
push(E e)	This method pushes an element onto the stack represented by this list.
remove()	This method retrieves and removes the head (first element) of this list.
remove(int index)	This method removes the element at the specified position in this list.
remove(Object o)	This method removes the first occurrence of the specified element from this list if it is present.
removeFirst()	This method removes and returns the first element from this list.

removeLast()	This method removes and returns the last element from this list.
size()	This method returns the number of elements in this list.
toArray()	This method returns an array containing all of the elements in this list in proper sequence (from first to last element).
toString()	This method returns a string containing all of the elements in this list in proper sequence (from first to the last element), each element is separated by commas

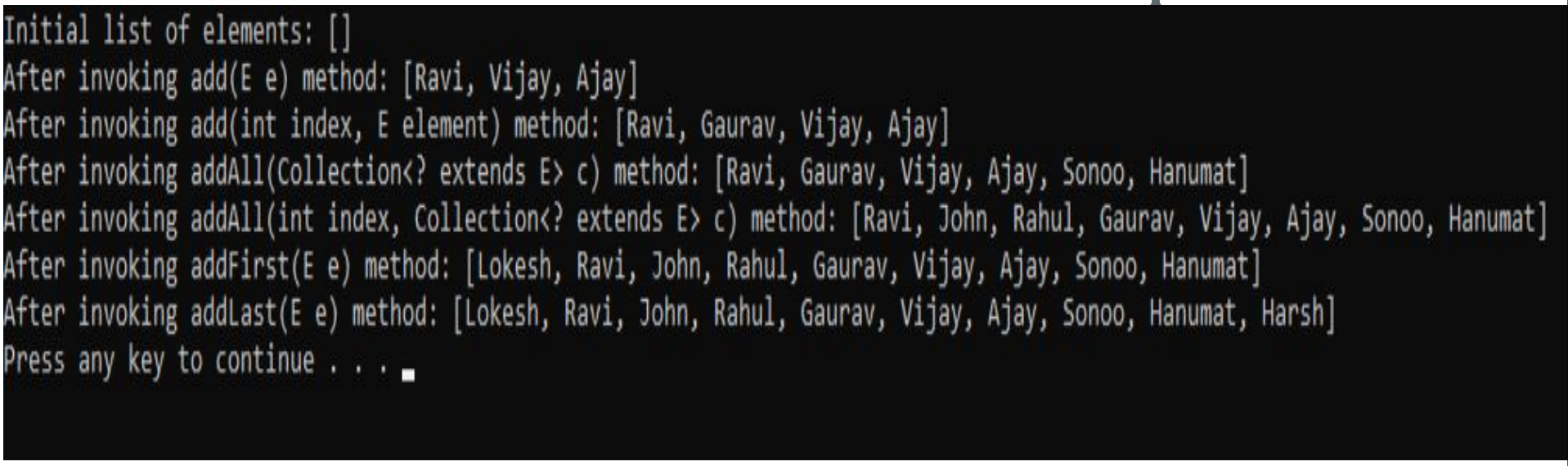
Java LinkedList Example

```
import java.util.*;
public class LinkedList1{
 public static void main(String args[]){

  LinkedList<String> al=new LinkedList<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");

  Iterator<String> itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

LinkListDemo.java

```
import java.util.*;
public class LinkedList2{
 public static void main(String args[]){
 LinkedList<String> ll=new LinkedList<String>();
           System.out.println("Initial list of elements: "+ll);
           ll.add("Ravi");
           ll.add("Vijay");
           ll.add("Ajay");
           System.out.println("After invoking add(E e) method: "+ll);
           //Adding an element at the specific position
           ll.add(1, "Gaurav");
           System.out.println("After invoking add(int index, E element) method: "+ll);
           LinkedList<String> ll2=new LinkedList<String>();
           ll2.add("Sonoo");
           ll2.add("Hanumat");
           //Adding second list elements to the first list
           ll.addAll(ll2);
           System.out.println("After invoking addAll(Collection<? extends E> c) method: "+ll);
           LinkedList<String> ll3=new LinkedList<String>();
           ll3.add("John");
           ll3.add("Rahul");
           //Adding second list elements to the first list at specific position
           ll.addAll(1, ll3);
           System.out.println("After invoking addAll(int index, Collection<? extends E> c) method: "+ll);
           //Adding an element at the first position
           ll.addFirst("Lokesh");
           System.out.println("After invoking addFirst(E e) method: "+ll);
           //Adding an element at the last position
           ll.addLast("Harsh");
           System.out.println("After invoking addLast(E e) method: "+ll);

 }
```

Java LinkedList Example



Java Deque interface

- Java Deque Interface is a linear collection that supports element insertion and removal at both ends. Deque is an acronym for "double ended queue".

  A deque can perform both the insertion and deletion using the LIFO (Last In First Out) principle

boolean add(object)	It is used to insert the specified element into this deque and return true upon success.
boolean offer(object)	It is used to insert the specified element into this deque.
Object remove()	It is used to retrieve and removes the head of this deque.
Object poll()	It is used to retrieve and removes the head of this deque, or returns null if this deque is empty.
Object element()	It is used to retrieve, but does not remove, the head of this deque.
Object peek()	It is used to retrieve, but does not remove, the head of this deque, or returns null if this deque is empty.

Object peekFirst()	The method returns the head element of the deque.
Object peekLast()	The method returns the last element of the deque.

ArrayDeque class

- It implements the Deque interface. It adds no methods of its own. ArrayDeque creates a dynamic array and has no capacity restrictions.
- It grows and shrinks as per usage. It also inherits the AbstractCollection class.

- The important points about ArrayDeque class are:

1. Unlike Queue, we can add or remove elements from both sides.
2. Null elements are not allowed in the ArrayDeque.
3. ArrayDeque has no capacity restrictions.
4. It is faster than LinkedList and Stack.

Java ArrayDeque Example

```
import java.util.*;
public class ArrayDequeExample {
    public static void main(String[] args) {
    //Creating Deque and adding elements
    Deque<String> deque = new ArrayDeque<String>();
    deque.add("Ravi");
    deque.add("Vijay");
    deque.add("Ajay");
    //Traversing elements
    for (String str : deque) {
    System.out.println(str);
    }
    }
}
```

ArrayDqueuDemo.java

```
import java.util.*;
public class DequeExample {
public static void main(String[] args) {
    Deque<String> deque=new ArrayDeque<String>();
    deque.offer("arvind");
    deque.offer("vimal");
    deque.add("mukul");
    deque.offerFirst("jai");
    System.out.println("After offerFirst Traversal...");
    for(String s:deque){
        System.out.println(s);
    }
    //deque.poll();
    //deque.pollFirst();//it is same as poll()
    deque.pollLast();
    System.out.println("After pollLast() Traversal...");
    for(String s:deque){
        System.out.println(s);
    }
}
}
```

```
After offerFirst Traversal...
jai
arvind
vimal
mukul
After pollLast() Traversal...
jai
arvind
vimal
```

Java HashSet

- Java HashSet class is used to create a collection that uses a hash table for storage. It implements Set interface.
- The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called hashing.
- HashSet *contains* unique elements only.
- HashSet allows null value.
- HashSet doesn't maintain the insertion order. elements are inserted on the basis of their hashcode.
- HashSet is the best approach for search operations.

HashSet Methods

1)	boolean	add(E e)	It is used to add the specified element to this set if it is not already present.
2)	void	clear()	It is used to remove all of the elements from the set.
3)	object	clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
4)	boolean	contains(Object o)	It is used to return true if this set contains the specified element.
5)	boolean	isEmpty()	It is used to return true if this set contains no elements.

boolean	remove(Object o)	It is used to remove the specified element from this set if it is present.
int	size()	It is used to return the number of elements in the set.

Java HashSet Example

```
import java.util.*;
class HashSet1{
 public static void main(String args[]){
  //Creating HashSet and adding elements
    HashSet<String> set=new HashSet();
            set.add("One");
            set.add("Two");
            set.add("Three");
            set.add("Four");
            set.add("Five");
            Iterator<String> i=set.iterator();
            while(i.hasNext())
            {
            System.out.println(i.next());
            }
  }
}
```

Java HashSet Example

```
import java.util.*;
class HashSet3{
 public static void main(String args[]){
  HashSet<String> set=new HashSet<String>();
            set.add("Ravi");
            set.add("Vijay");
            set.add("Arun");
            set.add("Sumit");
            System.out.println("An initial list of elements: "+set);
            //Removing specific element from HashSet
            set.remove("Ravi");
            System.out.println("After invoking remove(object) method: "+set);

            set.removeIf(str->str.contains("Vijay"));
            System.out.println("After invoking removeIf() method: "+set);
            //Removing all the elements available in the set
            set.clear();
            System.out.println("After invoking clear() method: "+set);
 }
}
```

```
An initial list of elements: [Vijay, Ravi, Arun, Sumit]
After invoking remove(object) method: [Vijay, Arun, Sumit]
After invoking removeIf() method: [Arun, Sumit]
After invoking clear() method: []
Press any key to continue . . .
```

Java TreeSet class

- The important points about Java TreeSet class are:
  The TreeMap in Java is a concrete implementation of the java.util.SortedMap interface.

  It provides an ordered collection of key-value pairs,
  A TreeMap is implemented using a Red-Black tree, which is a type of self-balancing binary search tree

- Java TreeSet class contains unique elements only like HashSet.
- Java TreeSet class access and retrieval times are quiet fast.
- Java TreeSet class doesn't allow null element.
- Java TreeSet class is non synchronized.
- Java TreeSet class maintains ascending order.

Methods of Java TreeSet class

oolean add(E e)	It is used to add the specified element to this set if it is not already present.
b o o l e a n addAll(Collection<? extends E> c)	It is used to add all of the elements in the specified collection to this set.
E ceiling(E e)	It returns the equal or closest greatest element of the specified element from the set, or null there is no such element.
Iterator descending Iterator()	It is used to iterate the elements in descending order.

E higher(E e)	It returns the closest greatest element of the specified element from the set, or null there is no such element.
Iterator iterator()	It is used to iterate the elements in ascending order.
E lower(E e)	It returns the closest least element of the specified element from the set, or null there is no such element.
boolean contains(Object o)	It returns true if this set contains the specified element.
boolean isEmpty()	It returns true if this set contains no elements.

boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
void clear()	It is used to remove all of the elements from this set.
E first()	It returns the first (lowest) element currently in this sorted set.
E last()	It returns the last (highest) element currently in this sorted set.
int size()	It returns the number of key-value pairs exists in the hashtable.
E higher(E e)	It returns the closest greatest element of the specified element from the set, or null there is no such element.

E pollFirst()	It is used to retrieve and remove the lowest(first) element.
E pollLast()	It is used to retrieve and remove the highest(last) element.
SortedSet headSet (E toElement)	It returns the group of elements that are less than the specified element.

Java TreeSet Examples

```
import java.util.*;
class TreeSet1{
 public static void main(String args[]){
  //Creating and adding elements
  TreeSet<String> al=new TreeSet<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");
  //Traversing elements
  Iterator<String> itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Java TreeSet Examples

```
import java.util.*;
class TreeSet3{
 public static void main(String args[]){
 TreeSet<Integer> set=new TreeSet<Integer>();
          set.add(24);
          set.add(66);
          set.add(12);
          set.add(15);
          System.out.println("Highest Value: "+set.pollFirst());
          System.out.println("Lowest Value: "+set.pollLast());
 }
}
```

```
Highest Value: 12
Lowest Value: 66
```

```java
import java.util.*;
class TreeSet4{
 public static void main(String args[]){
  TreeSet<String> set=new TreeSet<String>();
          set.add("A");
          set.add("B");
          set.add("C");
          set.add("D");
          set.add("E");
          System.out.println("Initial Set: "+set);

          System.out.println("Reverse Set: "+set.descendingSet());

          System.out.println("Head Set: "+set.headSet("C", true));

          System.out.println("SubSet: "+set.subSet("A", false, "E", true));

          System.out.println("TailSet: "+set.tailSet("C", false));
 }
}
```

```
Initial Set: [A, B, C, D, E]
Reverse Set: [E, D, C, B, A]
Head Set: [A, B, C]
SubSet: [B, C, D, E]
TailSet: [D, E]
```

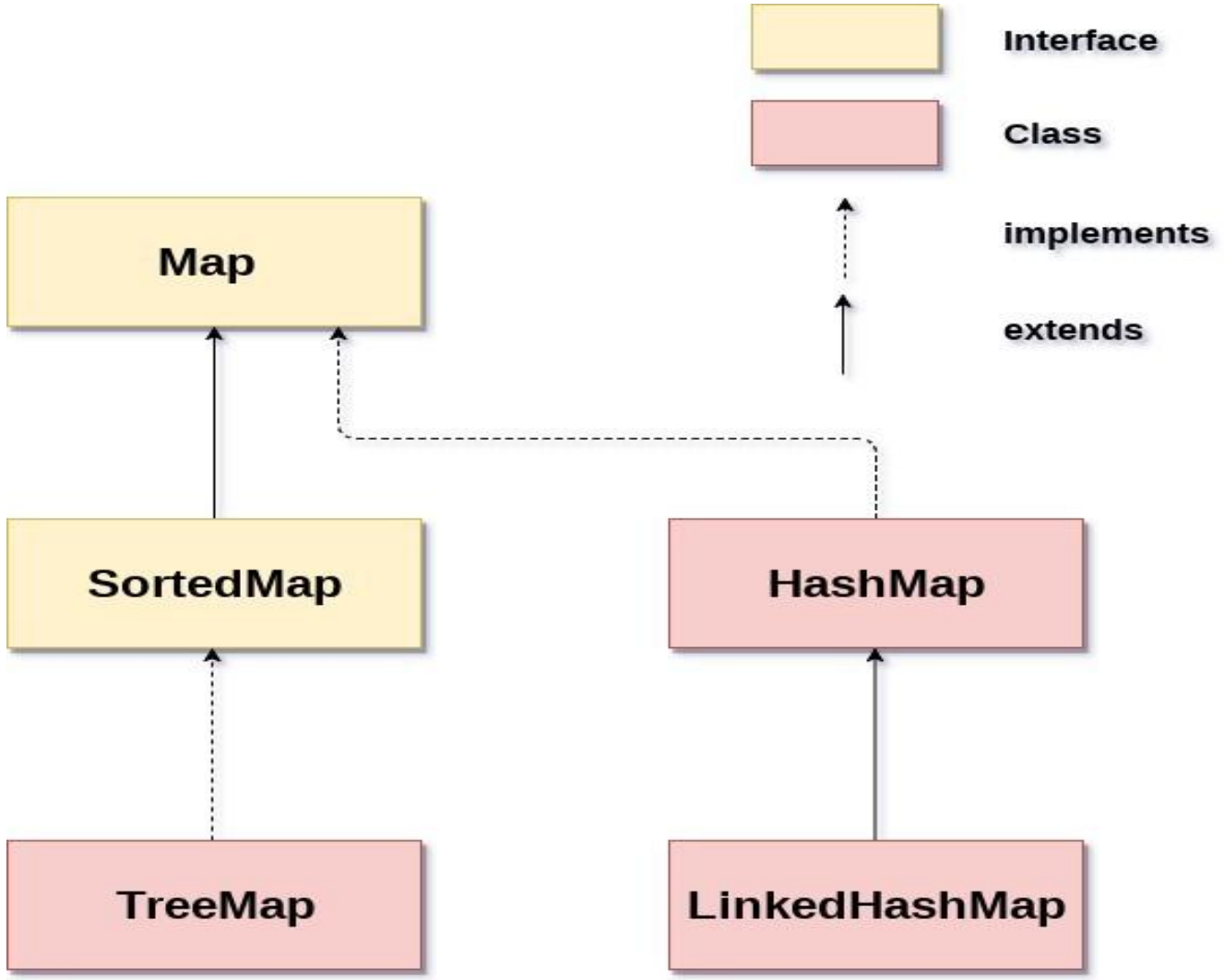
Java Map Interface

A 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.

A Map is useful if you have to search, update or delete elements on the basis of a key.

A Map doesn't allow duplicate keys, but you can have duplicate values.
HashMap and LinkedHashMap allow null keys and values, but TreeMap doesn't allow any null key or value.

Java Map Hierarchy



M

```
Map hm = new HashMap();
// Obj is the type of the object to be stored in Map
```

Methods of Map interface

clear()	This method is used in Java Map Interface to clear and remove all of the elements or mappings
containsKey(Object)	This method is used to check whether a particular key is being mapped into the Map or not. It takes the key element as a parameter and returns True if that element is mapped in the map.
putAll(Map)	This method is used in Map Interface in Java to copy all of the mappings from the specified map to this map.
values()	This method is used in Java Map Interface to create a collection out of the values of the map. It basically returns a Collection view of the values in the HashMap

containsValue(Object)	It takes the value as a parameter and returns True if that value is mapped by any of the keys in the map.
equals(Object)	This method is used in Java Map Interface to check for equality between two maps. It verifies whether the elements of one map passed as a parameter is equal to the elements of this map or not.
get(Object)	This method is used to retrieve or fetch the value mapped by a particular key mentioned in the parameter.
isEmpty()	This method is used to check if a map is having any entry for key and value pairs. If no mapping exists, then this returns true.
keySet()	This method is used in Map Interface to return a Set view of the keys contained in this map.

put(Object, Object)	This method is used in Java Map Interface to associate the specified value with the specified key in this map.
remove(Object)	This method is used in Map Interface to remove the mapping for a key from this map if it is present in the map.
size()	This method is used to return the number of key/value pairs available in the map.

Java Map Example

```
import java.util.*;
class MapExample2{
 public static void main(String args[]){
  Map<Integer,String> map=new HashMap<Integer,String>();
  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());
  }
 }
}
```

Output :
101 Amit
102 Vijay
103 Rahul

Java HashMap Class

- Java HashMap class implements the Map interface which allows us to store key and value pair, where keys should be unique.
- If you try to insert the duplicate key, it will replace the element of the corresponding key.
- It is easy to perform operations using the key index like updation, deletion, etc.
- HashMap class is found in the java.util package.
- It allows us to store the null elements as well, but there should be only one null key.
- Since Java 5, it is denoted as HashMap<K,V>, where K stands for key and V for value.

Java HashMap Class

- Java HashMap contains values based on the key.
- Java HashMap contains only unique keys.
- Java HashMap may have one null key and multiple null values.
- Java HashMap maintains no order.

Java HashMap Example

```
import java.util.*;
public class HashMapExample1{
 public static void main(String args[]){
   HashMap<Integer,String> map=new HashMap<Integer,String>();//Creating HashMap
   map.put(1,"Mango");  //Put elements in Map
   map.put(2,"Apple");
   map.put(3,"Banana");
   map.put(4,"Grapes");

   System.out.println("Iterating Hashmap...");
   for(Map.Entry m : map.entrySet()){
    System.out.println(m.getKey()+" "+m.getValue());
   }
}
}
```

```
Iterating Hashmap...
1 Mango
2 Apple
3 Banana
```

No Duplicate Key on HashMap

```
import java.util.*;
public class HashMapExample2{
 public static void main(String args[]){
   HashMap<Integer,String> map=new HashMap<Integer,String>();//Creating HashMap
   map.put(1,"Mango");  //Put elements in Map
   map.put(2,"Apple");
   map.put(3,"Banana");
   map.put(1,"Grapes"); //trying duplicate key

   System.out.println("Iterating Hashmap...");
   for(Map.Entry m : map.entrySet()){
    System.out.println(m.getKey()+" "+m.getValue());
   }
}
}
```

```
Iterating Hashmap...
1 Grapes
2 Apple
3 Banana
```

Java HashMap example to remove() elements

```
import java.util.*;
public class HashMap2 {
   public static void main(String args[]) {
    HashMap<Integer,String> map=new HashMap<Integer,String>();
      map.put(100,"Amit");
      map.put(101,"Vijay");
      map.put(102,"Rahul");
      map.put(103, "Gaurav");
    System.out.println("Initial list of elements: "+map);
    //key-based removal
    map.remove(100);
    System.out.println("Updated list of elements: "+map);
    //value-based removal
    map.remove(101);
    System.out.println("Updated list of elements: "+map);
    //key-value pair based removal
    map.remove(102, "Rahul");
    System.out.println("Updated list of elements: "+map);
   }
}
```

Java HashMap example to remove() elements

```
Initial list of elements: {100=Amit, 101=Vijay, 102=Rahul, 103=Gaurav}
Updated list of elements: {101=Vijay, 102=Rahul, 103=Gaurav}
Updated list of elements: {102=Rahul, 103=Gaurav}
Updated list of elements: {103=Gaurav}
```