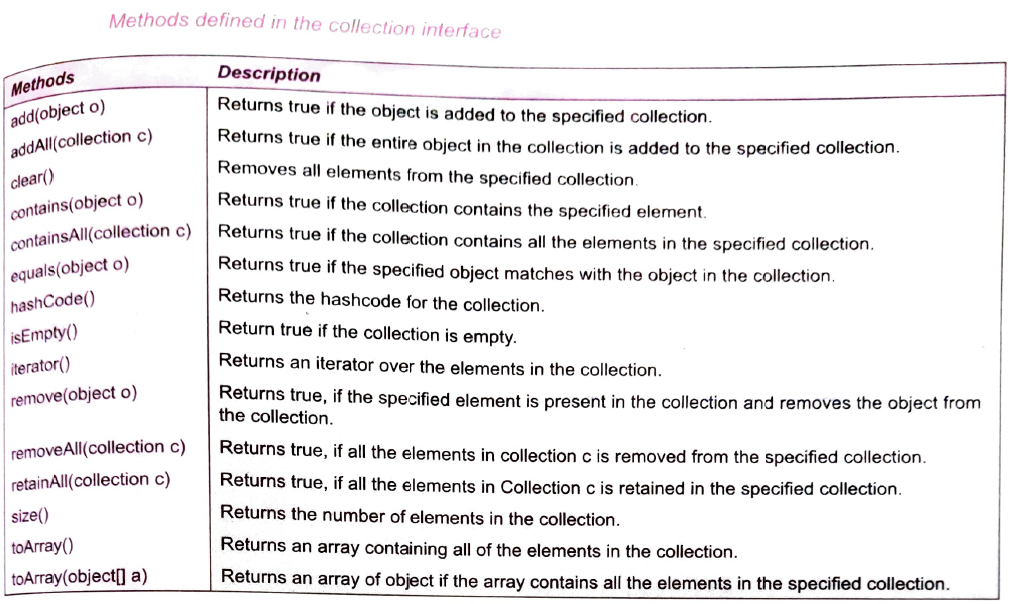
Java Collections

1. The collections framework contains many interfaces, such as collection, Map, and iterator. Other interfaces of the framework extend these interfaces. The interfaces available in the collection framework can be structured as shown in below table. The interfaces List and set are the subinterfaces of the collection interface. The SortedMap interface is the sub-interface of the Map interface. The listIterator interface of the Iterator

|  |  |
| --- | --- |
| Interface | Description |
| Collection | Collection of elements |
| List(extends Collection) | Sequence of elements |
| Queue(extends Collection) | Special type of list |
| Set(extends Collection) | Collection of unique elements |
| SortedMap(extends Set) | Sorted collection of unique elements |
| Map | Collection of key and value pairs, which must be unique |
| SortedMap(extends Map) | Sorted collection of unique key value pairs |
| Iterator | Object used to traverse through collection |
| ListIterator(extends Iterator) | Object used to traverse through sequence |

1. All collection classes implement the collection interface. The collection interface defines some methods, which enables us to access the objects of collection, List of them is as bellow



1. **Set interface** extends the Collection interface and it contains the methods that are inherited from

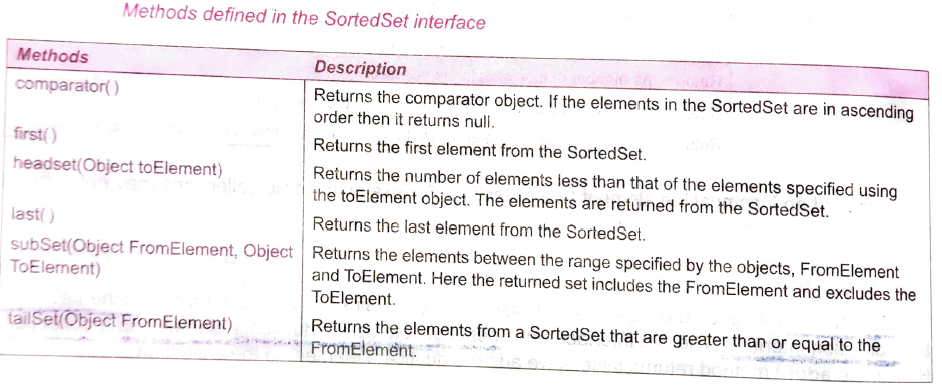
The Collection interface. The Set interface does not allow the use of duplicate elements in a collection.

🡪Hence, the add() method returns false, if we add the duplicate element to the collection.

1. **List interface** contains an ordered sequence of elements available in a collection. It allows

duplicate elements in the List. The List interface inherits the methods of the Collection interface. In addition to these methods the List interface also contains the methods described below :

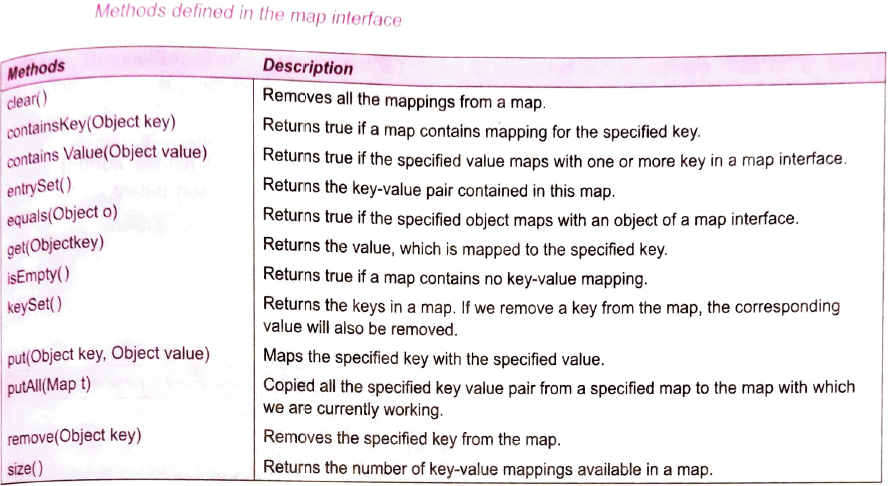
1. **SortedSet** **interface** is used to sort the elements of a collection in increasing order. The SortedSet interface extends the set interface, which in turn extends the Collection intertace. Ihe SortedSet intertace does not allow duplicate elements. In addition to the methods defined by the Set interface, the SortedSet interface contains the methods listed below :



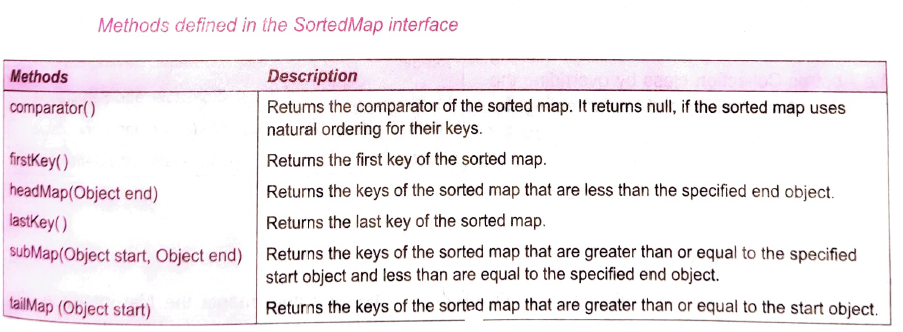
1. **Queue interface** extends Collection interface and declares the behavior of a queue, which is often a **First-In First-Out(FIFO)** list. In a queue, elements can only be removed from the head of the queue. The Queue interface defines a few methods as listed below :



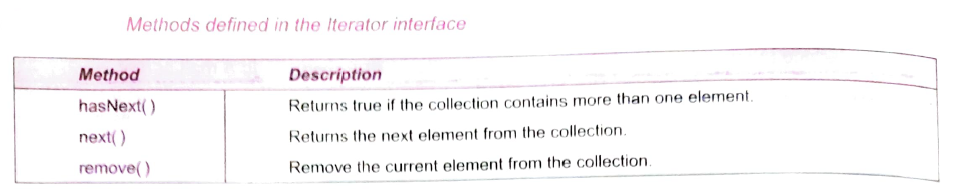
1. **Map interface** maps unique key element to their values. For example, in main server, each mail id is mapped to a unique password. The map interface allows us to view the elements of a collection as set of key, collection of values, and the mapping of key pairs. Some of the methods of the Map interface is as follow :



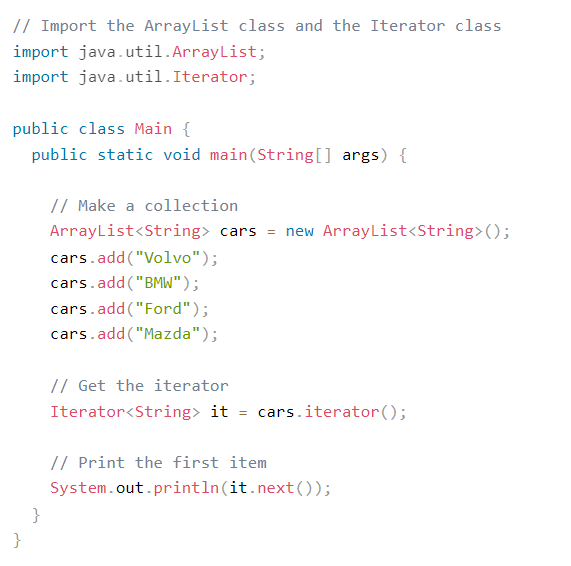
1. **SortedMap interface** extends the Map Interface. The SortedMap interface contains elements in ascending order. In this, the sorting is based on the keys. The functionality of the SortedMap is analogous to the functionality of the SortedSet interface. This Map interface is implemented in the TreelMap class. Methods of the sortedMap interface is as follow :



1. **Iterator** **interface** enables us to sequentially traverse and success the element contained in a collection. The element of a collection can be accessed using the methods defined by the iterator interface. The methods of the iterator interface is as follows :



🡺Example code of the iterator:

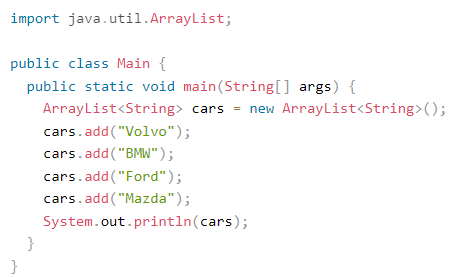


1. **ArrayList** **class** is a resizable array, which can be found in the java.util package.

🡪The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an ArrayList whenever you want. The syntax is also slightly different



🡺The example program of an array list is as follow :



🡺Multi-dimensional Array List(**V.I.M.P.**)

🡪Two dimensional Array list is represented by the following method :



🡪We can add all the edges (0, 1), (1, 2), and (2, 0), to our 2-D ArrayList by :

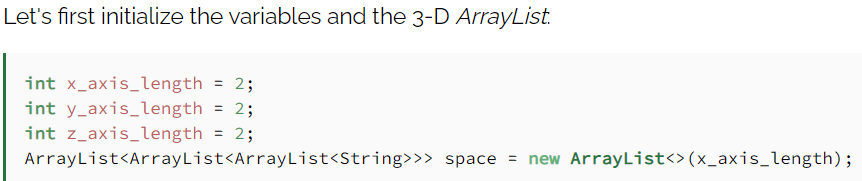


🡪WE can get the element at the (0,1) by : **graph.get(0).get(1);**

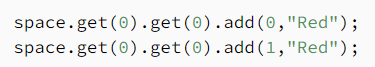
🡪Here we have to first get the first array’s value and than by help of another get method, we will get the value at the another array

🡺Three dimensional Array List :

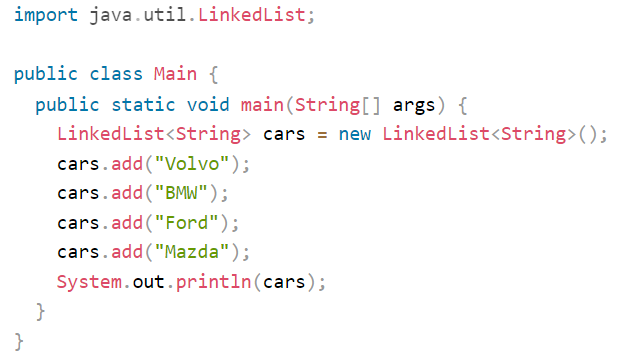
For simplicity, let's assume that we are creating a (2 x 2 x 2) 3-D space. It will have eight points: (0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), and (1, 1, 1).



🡪 And than we can add value to points in space. Let's add Red color for points (0, 0, 0) and (0, 0, 1):

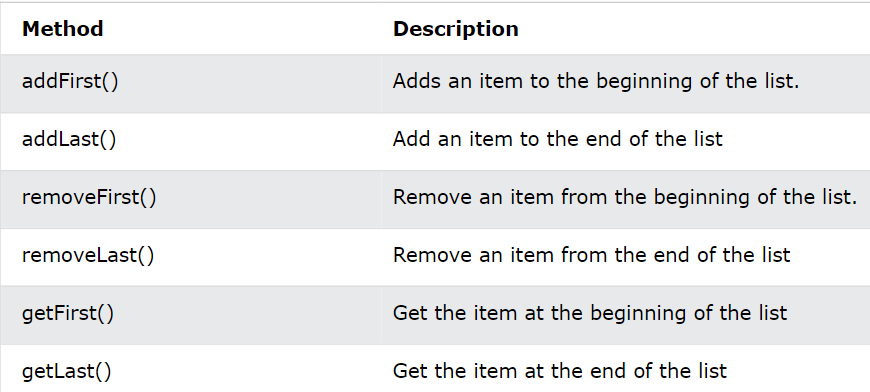


1. **LinkedList** is almost identical to that of the ArrayList, The LinkedList class has all of the same methods as the ArrayList class because they both implement the List interface. This means that you can add items, change items, remove items and clear the list in the same way.





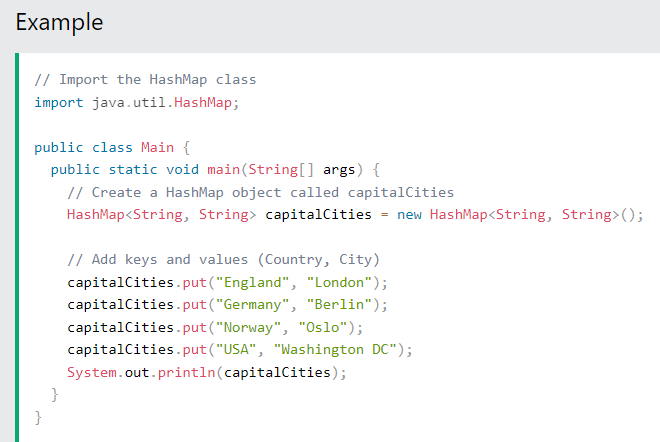
🡪LinkedList methods



1. **Java HashMap**

In the [ArrayList](https://www.w3schools.com/java/java_arraylist.asp) chapter, you learned that Arrays store items as an ordered collection, and you have to access them with an index number (int type). A HashMap however, store items in "**key**/**value**" pairs, and you can access them by an index of another type (e.g. a String).

One object is used as a key (index) to another object (value). It can store different types: String keys and Integer values, or the same type, like: String keys and String values:



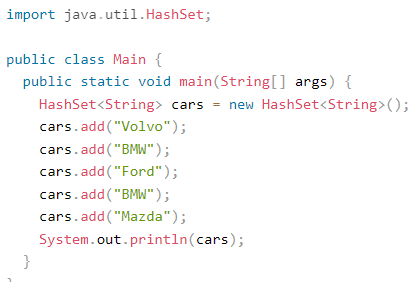
🡪Accessing an item in HashMap 

🡪All other methods like remove(), clear(), size(), for-each loop for iteration are same as that are used in the other collections

1. **A HashSet** is a collection of items where every item is unique, and it is found in the java.util package:

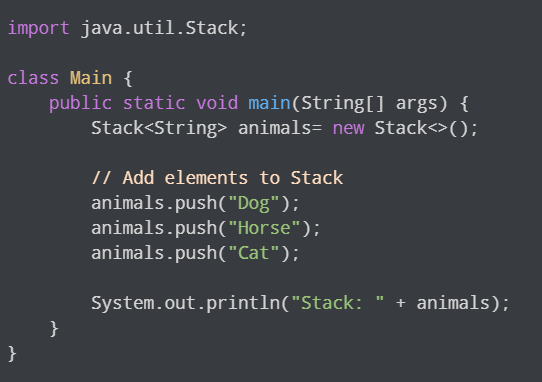
🡪 The HashSet class is used to create a collection and store it in a hash table, Each collection refers to a unique value called hash code.The hash code is Used as an index to associate with the object, which is stored in the hash table.

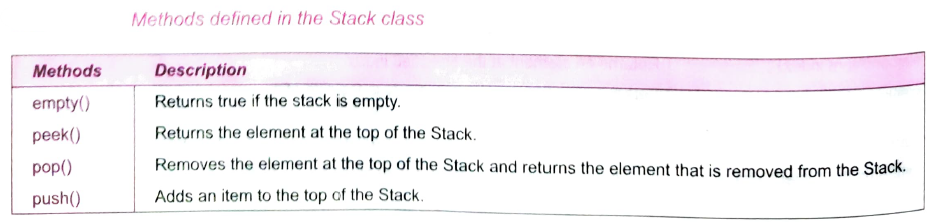
🡺Example :



1. **Stack class** that models and implements a Stack data structure. The class is based on the basic principle of last-in-first-out(LIFO). In addition to the basic push and pop operations, the class provides three more functions of empty, search, and peek.

🡪Example



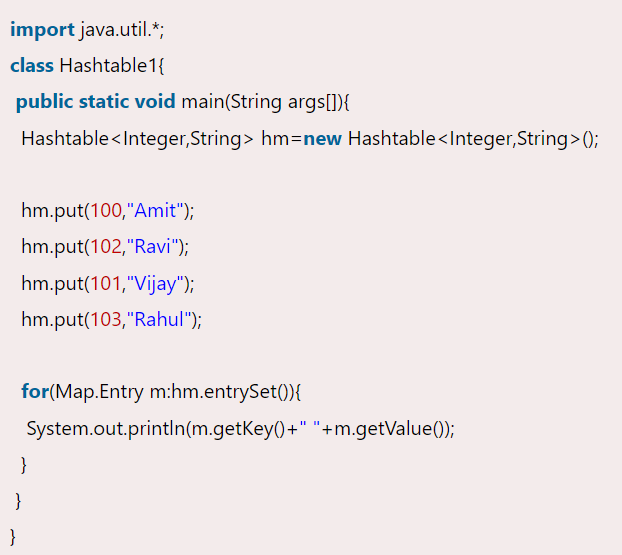


1. **Hashtable** class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value. To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.

🡪Hashtable stores key/value pair in hash table.

🡪In Hashtable we specify an object that is used as a key, and the value we want to associate to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.

🡪The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.

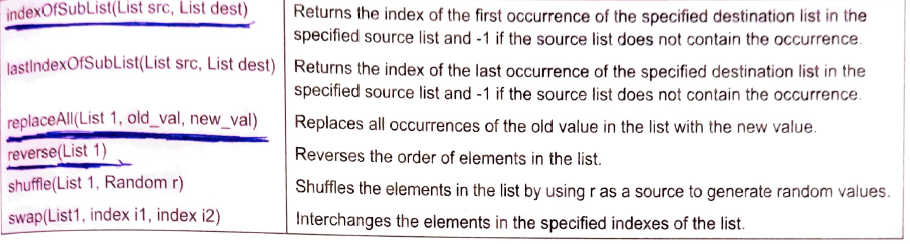
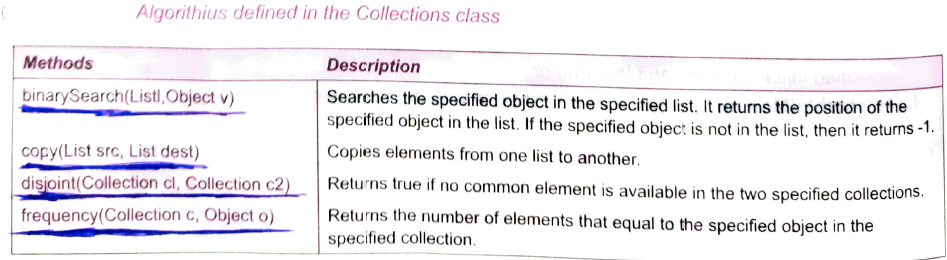


1. **OVERVIEW OF ALGORITHMS**

The collections framework supports several algorithms that allow us to operate on collections. We can use these algorithms to sort, shuffle, manipulate, and search a set of elements in a collection. Some of the algorithms available in the collections framework are:

1. Sorting
2. Shuffling
3. Manipulating
4. Searching

🡪These algorithms are contained in the Collection class



🡪The algorithms can be use by first making the object of the algorithm and than use of the required method

Example: 