UNIT – IV - Syllabus

The Collections Framework (java.util)- Collections overview, Collection Interfaces, Interface Iterator,
List, Set, ArrayList, LinkedListHashSet and ArrayDeque Classes.

Introduction and Overview of Collections in Java:

Hierarchy of Java Topics:

- Java and OOPs
- Collections Framework
- Exception Handling
- Multi-Threading
- Streams
- Strings etc.

Before Collections:

There are 4 ways to store values in JVM

1. Using variables : can store only **one value**
2. Using class object : can store multiple **fixed number of values** of **different types**
3. Using array object : can store multiple **fixed number of values** of **same** type
4. Using collections : **can store multiple objects of same and different types without size limitation**

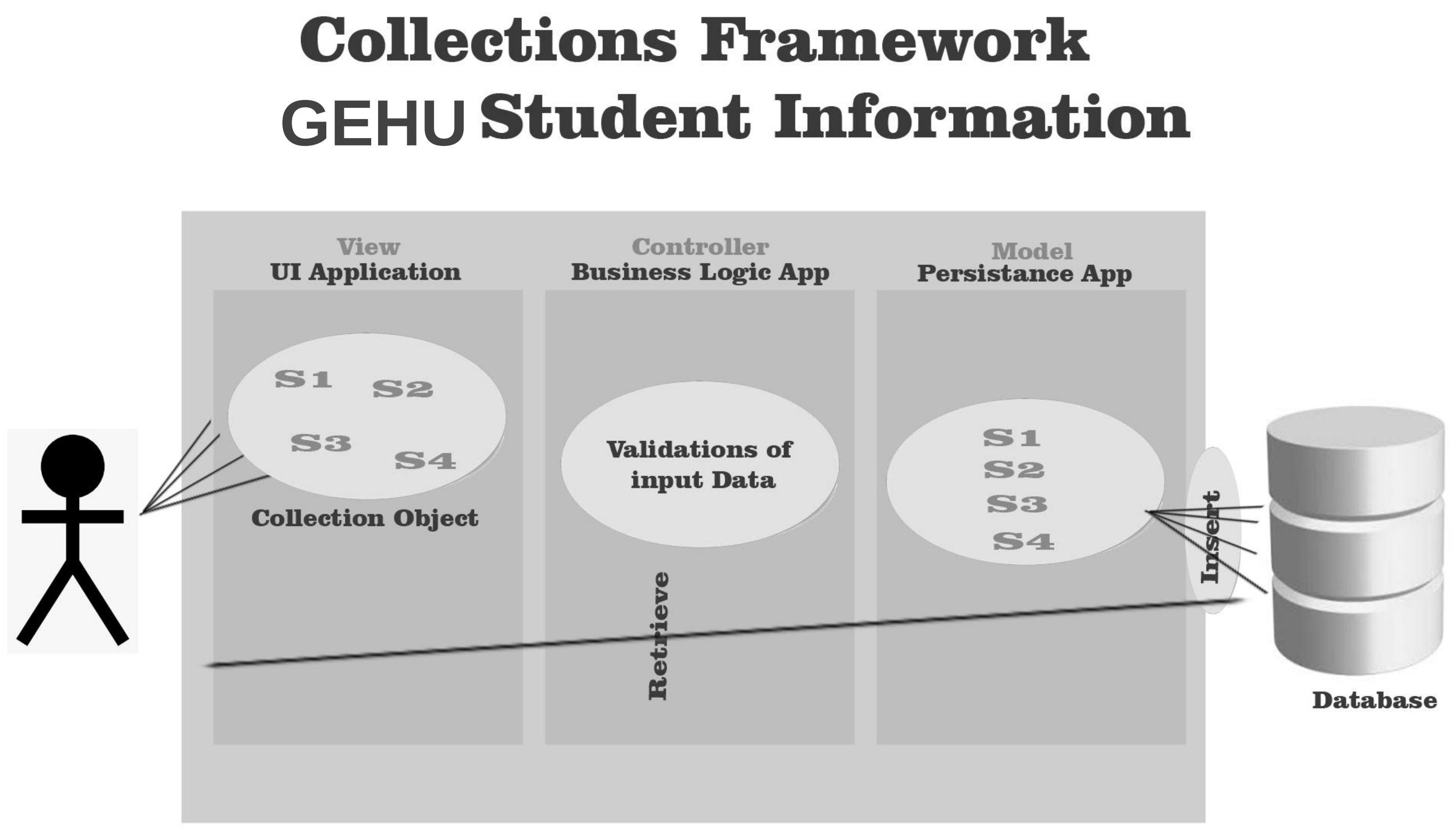
What is Collection?

In general terms a collection is a **"group of objects"**

Technical Definition:

Collection is a **container object**. It is used for storing homogeneous and heterogeneous, unique and duplicate objects without size limitation and further it is used for sending objects at a time from one class methods to other class methods as method arguments and return type with single name across multiple layers of the project.

- Before java 1.2 it is called simply **Java.util package**
- From java 1.2 onwards its all classes together called as **Collections Framework**
- From java 5 onwards they have added new concepts called **Collections and Generics**



Java Collections Framework all classes are divided into following **3 different types of classes.**

Classes

1. Container Objects classes
2. Cursor Objects classes
3. Utility Objects classes

1. Container Objects classes: Following **4 Interfaces** are called **container objects**

1. List <T> → HashSet <T>, LinkedHashSet<T>
2. Set <T> → Stack <T>, LinkedList<T>, ArrayList<T>, Vector<T>
3. Map <K,V> → HashMap <K,V>, Hashtable<K,V>
4. Queue <T> → LinkedList <T>

<T> Represents generic type parameter, i.e which type of elements are being stored.

2. Cursor objects are retrieving objects

1. Enumeration
2. Iterator
3. ListIterator

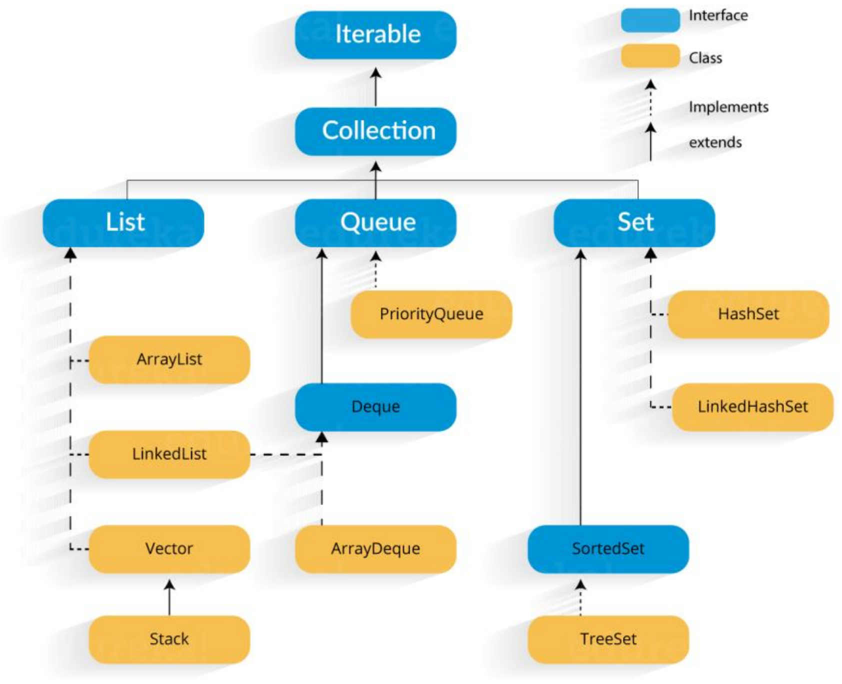
3. Utility Objects: it is nothing but a class that is given to perform different operations on container objects are called utility **objects** i.e two classes

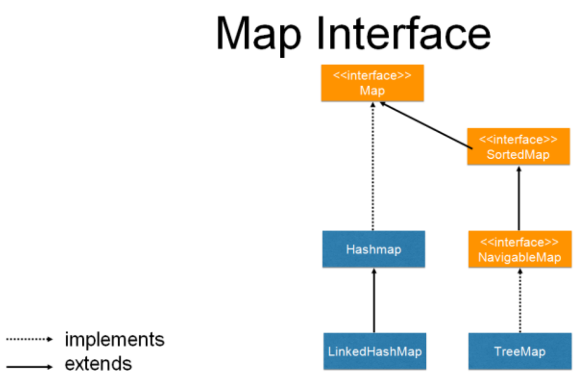
1. Collections class
2. Arrays

The **Collection in Java** is a framework that provides architecture to store and manipulate the group of objects.

1. Collections are the containers that groups multiple items in a single unit
2. It provides architecture to store and manipulate a group of objects
3. Using collections various operations can be performed on the data like
   - searching,
   - sorting,
   - insertion,
   - manipulation,
   - deletion etc.
4. Java collection framework provide many Interfaces and classes

Collection Interfaces and class hierarchy



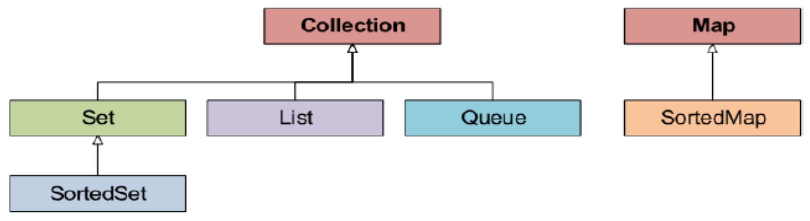


Iterable is the root interface of the **Java collection** classes. The **Collection** interface extends **Iterable** interface, so all subtypes of **Collection** implement the **Iterable** interface. This interface stands to represent data-structures whose value can be traversed one by one. This is an important property.

Simple Hirarchy

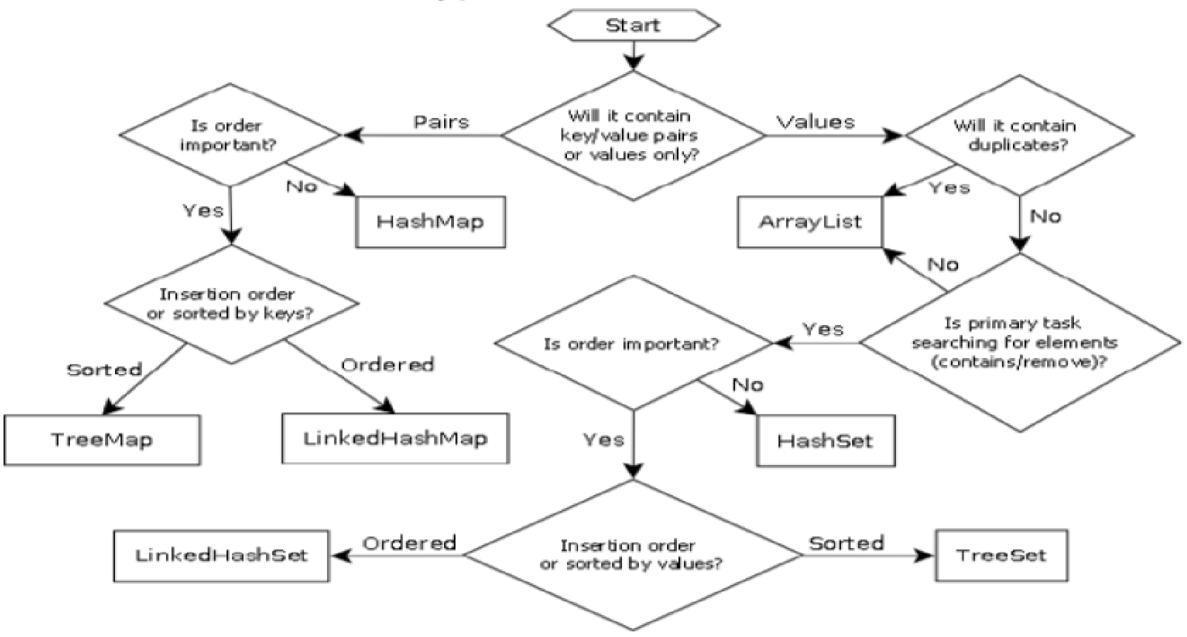
Hierarchy of collections

JDK does not provide direct implementations of the Collection and Map interfaces, but there are many implementations of more specific sub-interfaces :



Choosing the Righ Collection

Choosing the right collection - WHEN



Start
Will it contain key/value pairs or values only?
Pairs
Values
Is order important?
No
HashMap
Yes
Insertion order or sorted by keys?
Sorted
TreeMap
Ordered
LinkedHashMap
Will it contain duplicates?
Yes
ArrayList
No
Is primary task searching for elements (contains/remove)?
No
Yes
Is order important?
No
HashSet
Yes
Insertion order or sorted by values?
Ordered
LinkedHashSet
Sorted
TreeSet

Sets:

A set represents a group of elements arranged just like an array. The set will grow dynamically when the elements are stored into it. A set will **not allow duplicate elements**. If we try to pass for same element that is already available in the set, then it is not stored into the set.

List:

Lists are like sets. They store a group of elements. But **lists allow duplicate values** to be stored.

Queues:

A Queue represents arrangement of elements in **FIFO (First In First Out) order**. This means that an element that is stored as a first element into the queue will be removed first from the queue.

Maps:

Maps store elements in the form of key and value pairs. **If the key is provided then it's correspond value can be obtained**. Of course, the keys should have unique values.

Remember, in all the cases the 'elements' refer to 'objects' only. This means we cannot store primitive data types in the collection objects. We can store only objects since the main aim of collections is to handle objects only, not the primitive data types.

In the hierarchy we can divided into two groups

1. If your requirement is to group key-value pair, then choose MAP
2. If your requirement (operations are on only values) is to have only the values, then choose collection interfaces

Retrieving Elements from Collections

Following 4 ways to retrieve any element from a collection object

- Using ***for-each*** loop.
- Using ***Iterator*** interface.
- Using ***ListIterator*** interface.
- Using ***Enumeration*** interface.

1. For-each loop:

This is like for loop which repeatedly executes a group of statements for each element of the collection. The format is:

```
for(variable : collection-object)
{
        Statements:
}
```

Here, the variable assumes each element of the collection-object and the loop is executed as many times as there are number of elements in the collection-object. If collection-object has **n** elements the loop is executed exactly **n** times and the variable stores each element in each step.

Example using for-each loop

```
          class ForEachDemo{
            public static void main(String args[]){

             int arr[] = {12,13,14,44};          //declaring an array

             for(int i : arr){                    //traversing the array with for-each loop
               System.out.println(i);
             }
      }
}
```

Output:

```
12
13
14
44
```

2. Iterator Interface: Iterator is an interface that contains methods to retrieve the elements one by one from a collection object. It retrieves elements only in forward direction. It has 3 methods:

Method	Description
boolean hasNext()	This method returns true if the iterator has more elements.
element next()	This method returns the next element in the iterator.
void remove()	This method removes the last element from the collection returned by the iterator.

3. ListIterator Interface: ListIterator is an interface that contains methods to retrieve the elements from a collection object, both in forward and reverse directions. It can retrieve the elements in forward and backward direction. It has the following important methods:

Method	Description
boolean hasNext()	This method returns true if the ListIterator has more elements when traversing the list in forward direction.
element next()	This method returns the next element.
void remove()	This method removes the list last element that was returned by the next () or previous () methods.
boolean hasPrevious()	This method returns true if the ListIterator has more elements when traversing the list in reverse direction.
element previous()	This method returns the previous element in the list.

4. Enumeration Interface: This interface is useful to retrieve elements one by one like Iterator. It has 2 methods.

Method	Description
boolean hasMoreElements()	This method tests Enumeration has any more elements.
element nextElement()	This returns the next element that is available in Enumeration.

HashSet Class

HashSet Class: HashSet represents a set of elements (objects). It does **not guarantee the order** of elements. Also it **does not allow the duplicate** elements to be stored.

- We can write the HashSet class as : class HashSet <T>
- We can create the object as : HashSet <String> hs = new HashSet<String> ();

The following constructors are available in HashSet:

- **HashSet();**
- **HashSet (int capacity);** Here capacity represents how many elements can be stored into the HashSet initially. This capacity may increase automatically when more number of elements is being stored.

HashSet Class Methods:

Method	Description
boolean add(obj)	This method adds an element obj to the HashSet. It returns true if the element is added to the HashSet, else it returns false. If the same element is already available in the HashSet, then the present element is not added.
boolean remove(obj)	This method removes the element obj from the HashSet, if it is present. It returns true if the element is removed successfully otherwise false.
void clear()	This removes all the elements from the HashSet
boolean contains(obj)	This returns true if the HashSet contains the specified element obj.
boolean isEmpty()	This returns true if the HashSet contains no elements.
int size()	This returns the number of elements present in the HashSet.

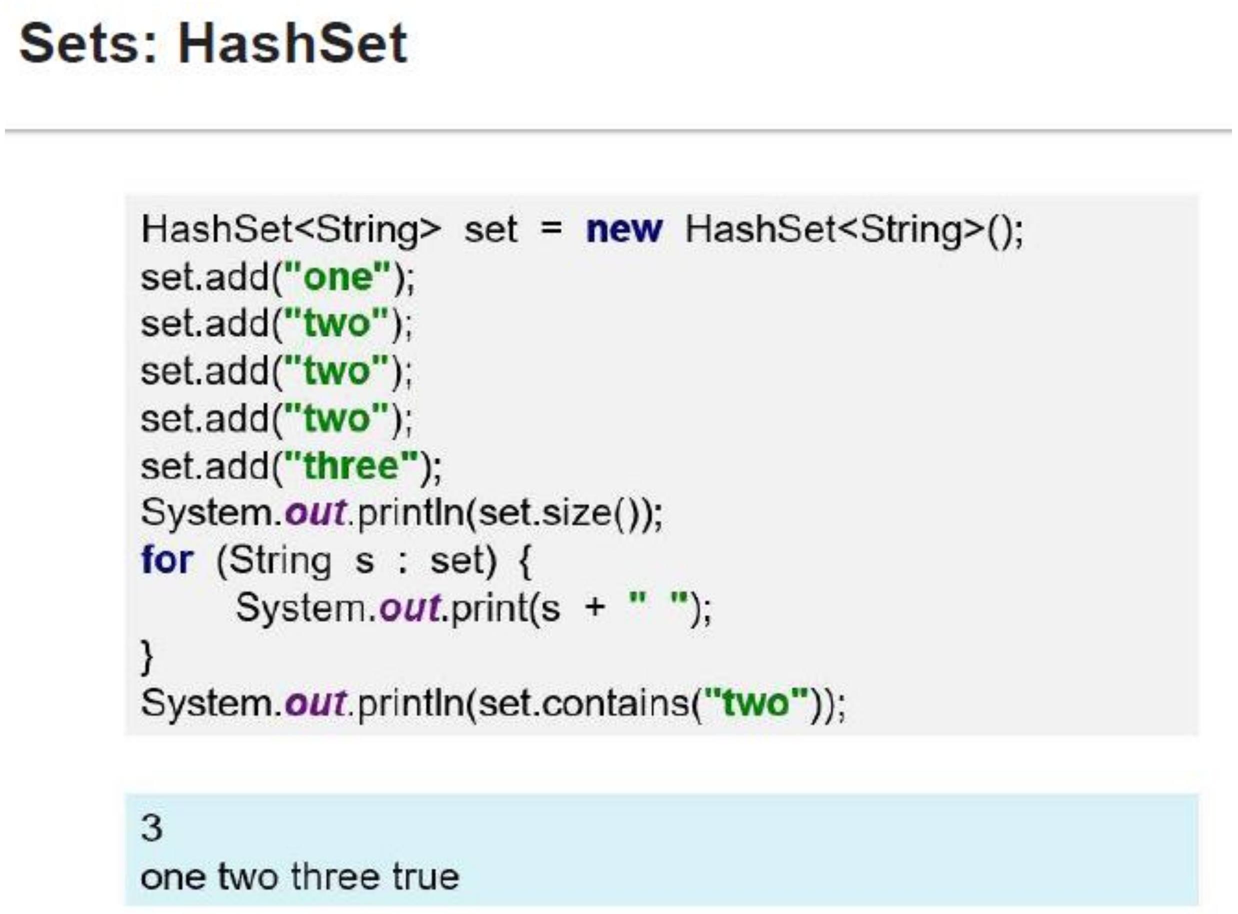
Program : Write a program which shows the use of HashSet and Iterator.

```
package com.myPack;

import java.util.HashSet;
import java.util.Iterator;

public class HashSetDemo {

     public static void main(String[] args) {
          //create a HashSet to store Strings
         HashSet <String> hs = new HashSet<String> ();
         //Store some String elements
         hs.add ("Anil");
         hs.add ("Akshara");
         hs.add ("Babji");
         hs.add ("Charan");
         hs.add ("Raman");
        // hs.add("India");
         //view the HashSet
         System.out.println ("HashSet = " + hs);
         //add an Iterator to hs
         Iterator<String> it = hs.iterator ();
         //display element by element using Iterator
         System.out.println ("Elements Using Iterator: ");
         while (it.hasNext() )
         {
          String s = (String) it.next ();
             System.out.println(s);
         }
     }
  }
```

Output:

```
HashSet = [Akshara, Raman, Babji, Anil, Charan]
Elements Using Iterator:
Akshara
Raman
Babji
Anil
Charan
```

Program on Hashset



Sets: HashSet

```
HashSet<String> set = new HashSet<String>();
set.add("one");
set.add("two");
set.add("two");
set.add("two");
set.add("three");
System.out.println(set.size());
for (String s : set) {
      System.out.print(s + " ");
}
System.out.println(set.contains("two"));
```

```
3
one two three true
```

LinkedHashSet Class

LinkedHashSet Class: This is a subclass of HashSet class and does not contain any additional members on its own. LinkedHashSet internally uses a linked list to store the elements. It is a generic class that has the declaration:

class LinkedHashSet <T>

Stack Class: A stack represents a group of elements stored in **LIFO** (Last In First Out) order.

This means that the element which is stored as a last element into the stack will be the first element to be removed from the stack. Inserting the elements (Objects) into the stack is called **push** operation and removing the elements from stack is called **pop** operation.

Searching for an element in stack is called **peep** operation. Insertion and deletion of elements take place only **from one side of the stack**, called **top** of the stack.

We can write a Stack class as:

class Stack<E>

e.g.: Stack<Integer> obj = new Stack<Integer> ();

Stack Class Methods:

Method	**Description**
boolean empty()	this method tests whether the stack is empty or not. If the stack is empty then true is returned otherwise false.
element peek()	this method returns the top most object from the stack without removing it.
element pop()	this method pops the top-most element from the stack and returns it.
element push(element obj)	this method pushes an element obj onto the top of the stack and returns that element.
int search(Object obj)	This method returns the position of an element obj from the top of the stack. If the element (object) is not found in the stack then it returns -1.

Program : Write a program to perform different operations on a stack.

```
package com.myPack;

import java.util.*;

class Stack1
{
	int top = -1, st[]=new int[5];

	void push(int el) {
		st[++top]=el;
	}

	int pop(){
		return(st[top--]);
	}

	void display()
	{
		System.out.println("\nStack elements from top to
bottom\n");
		for(int i=top;i>=0;i--)
		System.out.println(st[i]);
	}

	boolean isFull(){
		return(top==5-1);
	}

	boolean isEmpty(){
		return(top==-1);
	}
}
```

```
public class Stack {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		Stack1 s = new Stack1();
		int el = 0, ch=1;

		while(ch != 4) {
System.out.println("\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT");
			System.out.println("ENTER YOUR CHOICE");
			ch=sc.nextInt();


switch(ch){
		case 1 : if(s.isFull())
				System.out.println("\nstack is full");
				 else {
				System.out.println("Enter element");
				el=sc.nextInt();
				s.push(el);
				 }break;

		case 2: if(s.isEmpty())
				System.out.println("\nstack is empty");
				else {
				el=s.pop();
				System.out.println("\nDeleted element =
"+el);
				}break;

			case 3:   if(s.isEmpty())
				System.out.println("\nstack is empty");
				else
				s.display();
				break;

			case 4: break;

	default:System.out.println("\nEnter correct choice");
		}
		}
		sc.close();
```

```
          }
}
```

OUTPUT:

```
C:\Users\Anil\Desktop\2018 Java>javac Stack.java

C:\Users\Anil\Desktop\2018 Java>java Stack

1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER YOUR CHOICE
1
Enter element
10

1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER YOUR CHOICE
1
Enter element
20

1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER YOUR CHOICE
1
Enter element
30

1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER YOUR CHOICE
3

Stack elements from top to bottom

30
20
10
```

```
1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER YOUR CHOICE
1
Enter element
40

1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER YOUR CHOICE
3

Stack elements from top to bottom

40
30
20
10

1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER YOUR CHOICE
1
Enter element
50

1.PUSH
2.POP
3.DISPLAY
4.EXIT
ENTER YOUR CHOICE
3

Stack elements from top to bottom

50
40
30
20
10

1.PUSH
2.POP
3.DISPLAY
```

```
4.EXIT
ENTER YOUR CHOICE
2

Deleted element = 50

1.PUSH
2.POP
3.DISPLAY
4.EXIT
                              ENTER YOUR CHOICE
```

ArrayList Class

ArrayList Class: An ArrayList is like an array, which can grow in memory dynamically.

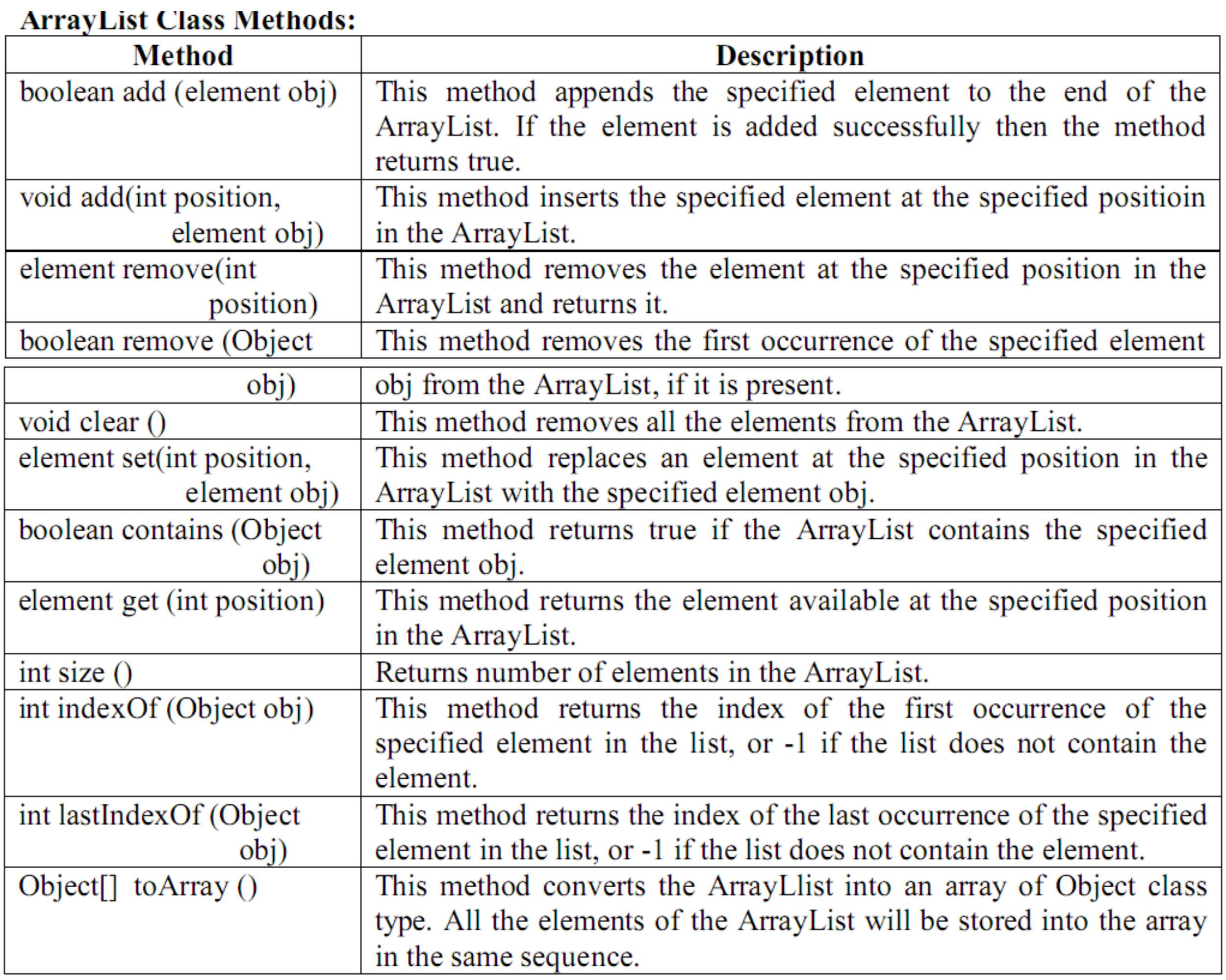
ArrayList is not synchronized. This means that when more than one thread acts simultaneously on the ArrayList object, the results may be incorrect in some cases.

ArrayList class can be written as:

class ArrayList <E>

We can create an object to ArrayList as:

ArrayList <String> arl = new ArrayList<String> ();



ArrayList Class Methods:

Method	Description
boolean add (element obj)	This method appends the specified element to the end of the ArrayList. If the element is added successfully then the method returns true.
void add(int position, element obj)	This method inserts the specified element at the specified positioin in the ArrayList.
element remove(int position)	This method removes the element at the specified position in the ArrayList and returns it.
boolean remove (Object obj)	This method removes the first occurrence of the specified element obj from the ArrayList, if it is present.
void clear ()	This method removes all the elements from the ArrayList.
element set(int position, element obj)	This method replaces an element at the specified position in the ArrayList with the specified element obj.
boolean contains (Object obj)	This method returns true if the ArrayList contains the specified element obj.
element get (int position)	This method returns the element available at the specified position in the ArrayList.
int size ()	Returns number of elements in the ArrayList.
int indexOf (Object obj)	This method returns the index of the first occurrence of the specified element in the list, or -1 if the list does not contain the element.
int lastIndexOf (Object obj)	This method returns the index of the last occurrence of the specified element in the list, or -1 if the list does not contain the element.
Object[] toArray ()	This method converts the ArrayLlist into an array of Object class type. All the elements of the ArrayList will be stored into the array in the same sequence.

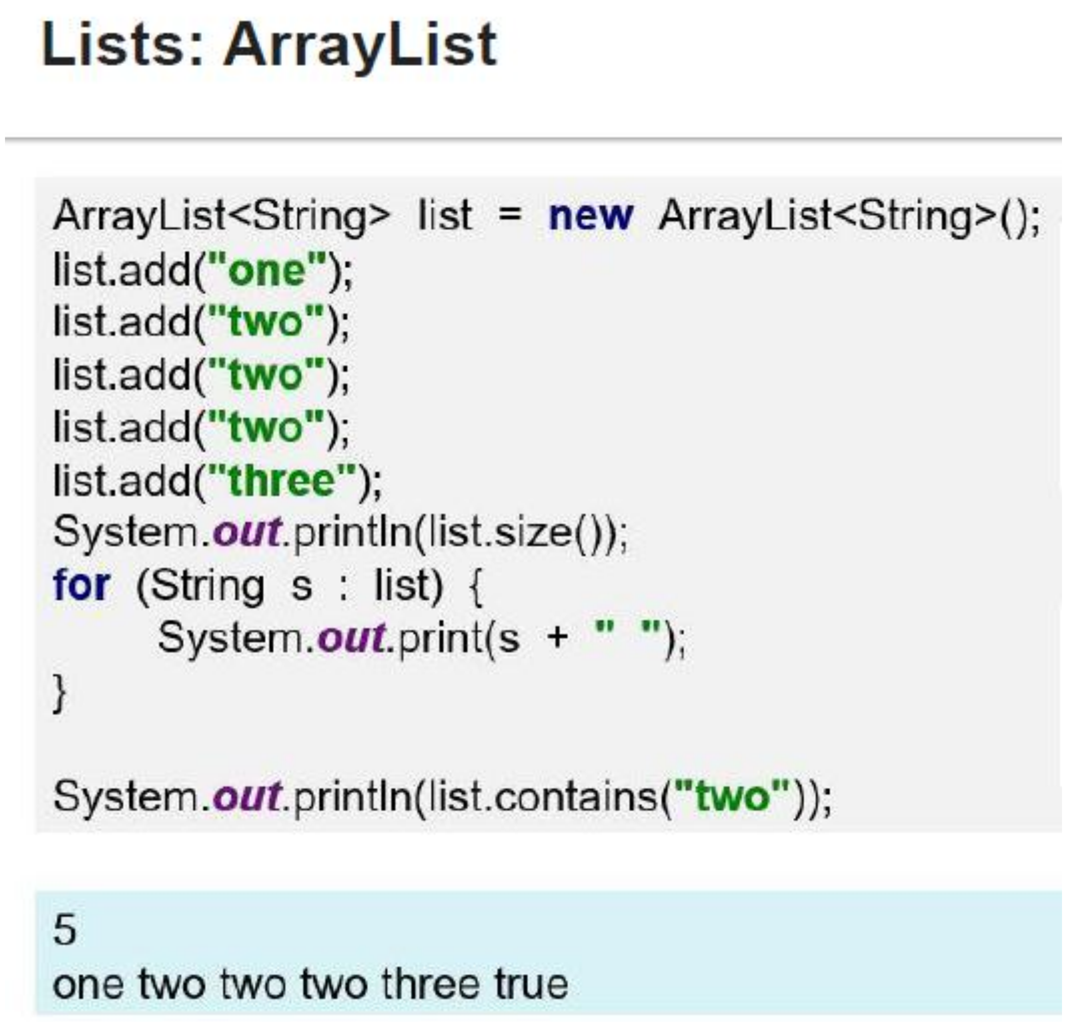
Program : Write a program that shows the use of ArrayList class.

```
package com.myPack;

import java.util.ArrayList;
import java.util.Iterator;

public class ArrayListDemo {
     public static void main(String[] args) {
          ArrayList <String> al = new ArrayList<String>();
         al.add ("Asia");
         al.add ("North America Carona");
         al.add ("South America");
         al.add ("Africa");
         al.add ("Europe");
         al.add (1,"Australia");
         al.add (2, "Antarctica");
         al.add (4, "Carona");
         System.out.print ("Size of the Array List is: \n" +
al.size ());
         System.out.print ("\nRetrieving elements in
ArrayList using Iterator :\n");
         Iterator<String> it = al.iterator ();
         while (it.hasNext () )
           System.out.print ("\n" +it.next ());
     }
   }
```

Output:

```
Size of the Array List is:
8
Retrieving elements in ArrayList using Iterator :

Asia
Australia
Antarctica
North America Carona
Carona
South America
Africa
Europe
```

Program on ArrayList



Lists: ArrayList

```java
ArrayList<String> list = new ArrayList<String>();
list.add("one");
list.add("two");
list.add("two");
list.add("two");
list.add("three");
System.out.println(list.size());
for (String s : list) {
      System.out.print(s + " ");
}

System.out.println(list.contains("two"));
```

```
5
one two two two three true
```

LinkedList Class: A linked list contains a group of elements in the form of nodes. Each node will have three fields- the data field contatins data and the link fields contain references to previous and next nodes.A linked list is written in the form of:

class LinkedList<E>

we can create an empty linked list for storing String type elements (objects) as:

LinkedList <String> ll = new LinkedList<String> ();

LinkedList Class methods:

Method	Description
boolean add (element obj)	This method adds an element to the linked list. It returns true if the element is added successfully.
void add(int position, element obj)	This method inserts an element obj into the linked list at a specified position.
void addFirst(element obj)	This method adds the element obj at the first position of the linked list.
void addLast(element obj)	This method adds the element obj at the last position of the linked list.
element removeFirst ()	This method removes the first element from the linked list and returns it.
element removeLast ()	This method removes the last element from the linked list and returns it.
element remove (int position)	This method removes an element at the specified position in the linked list.
void clear ()	This method removes all the elements from the linked list.
element get (int position)	This method returns the element at the specified position in the linked list.
element getFirst ()	This method returns the first element from the list.
element getLast ()	This method returns the last element from the list.
element set(int position, element obj)	This method replaces the element at the specified position in the list with the specified element obj.
int size ()	Returns number of elements in the linked list.
int indexOf (Object obj)	This method returns the index of the first occurrence of the specified element in the list, or -1 if the list does not contain the element.
int lastIndexOf (Object obj)	This method returns the index of the last occurrence of the specified element in the list, or -1 if the list does not contain the element.
Object[] toArray()	This method converts the linked list into an array of Object class type. All the elements of the linked list will be stored into the array in the same sequence.

Note: In case of LinkedList counting starts from 0 and we start counting from 1.

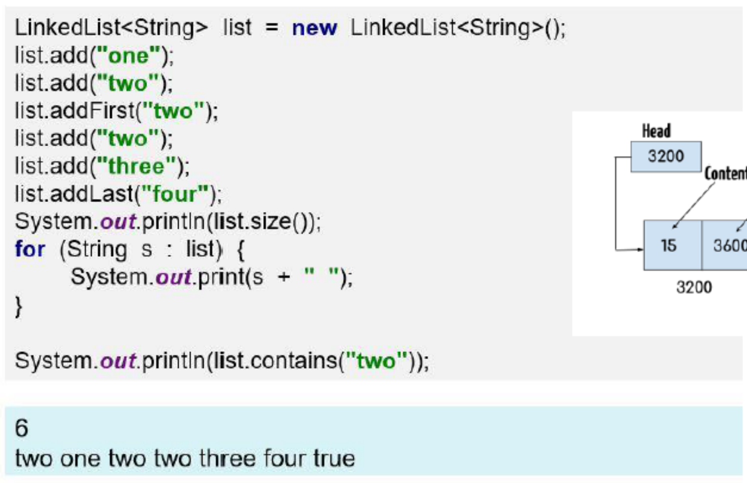
Program : Write a program that shows the use of LinkedList class.

```
package com.myPack;

import java.util.LinkedList;

public class LinkedListDemo {

    public static void main(String[] args) {
        LinkedList <String> ll = new LinkedList<String>();
       ll.add ("Anil");
       ll.add ("Bharathi");
       ll.add ("Charan");
       ll.add ("Chiranjeevi");
       ll.addFirst ("Eega");
       ll.add (1,"Abishai");
       ll.add (2,"Abhimanyu");
System.out.println ("Elements in Linked List is :\n "+ ll);
System.out.println ("Size of the Linked List is : \n" +
ll.size() );
    }
  }
```

Output:

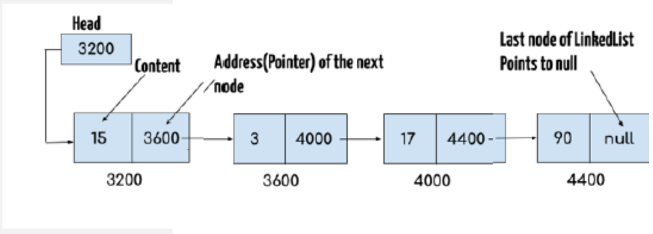
```
Elements in Linked List is:
 [Eega, Abishai, Abhimanyu, Anil, Bharathi, Charan,
Chiranjeevi]

Size of the Linked List is:
  7
```

Program on linked List



```
LinkedList<String>  list  =  new  LinkedList<String>();
list.add("one");
list.add("two");
list.addFirst("two");
list.add("two");
list.add("three");
list.addLast("four");
System.out.println(list.size());
for  (String  s  :  list)  {
        System.out.print(s  +  "  ");
}

System.out.println(list.contains("two"));
```

```
6
two one two two three four true
```


Head
3200
Content
Address(Pointer) of the next node
Last node of LinkedList Points to null
15
3600
3
4000
17
4400
90
null
3200
3600
4000
4400

Short Answer Questions

1. What is the use of Iterator class?
2. What is the use of String Tokenizer class?
3. Explain the use of String Tokenizer with example?
4. Explain any three methods defined by iterator?
5. Explain the methods defined by Vector.
6. What is the difference between array and vector?
7. Discuss the methods of Stack class
8. What is the need of Generics?

Long Answer Questions

1. What is a vector? How does it differ from array, list?
2. Differentiate between ArrayList and Vector.
3. List the methods of Stack class.
4. Write a program to stores the names of bank depositors and their current balances by using hash table?
5. Write a program which stores a list of strings in an ArrayList and then displays the contents of the list.