MIRACLE SOFTWARE SYSTEMS, INC.

Collection Framework

Table of contents

1. Collection.....3
2. List Interface.....3
2.1. Array List.....3
2.2. Linked List.....5
2.3 Vector.....5
3.Set Interface.....6
3.2. TreeSet.....7
3.3. LinkedHashSet:.....7
4. Map Interface:.....7
4.1. HashMap.....7
4.2. HashTable.....8
4.3. TreeMap.....9
4.4. LinkedHashMap.....9
5. ListIterator Interface.....10
6. Properties class.....11
7.Generics.....12
7.1 Writing generic methods.....13
7.2 Writing Generic Class.....14
8. Advantages of Collections over Arrays.....14

1. Collection

A Group of individual objects as a single entity is called "Collection".

What is Framework ?
.Provides readymade architecture.
.represents set of classes and interface.

What is Collection Framework ?
Collection framework represents a unified architecture for storing and manipulating group of object. It defines several classes & Interface,which an be used to represent a group of objects as a single entity. All Collection Framework contains the following:

- Interfaces
- Implementations
- Algorithm

Terminology:

Java	c++
Collection	Container
Collection Framework	STL(Standard Template Library)

Note: The **Java.util** package contains all the classes and interfaces for Collection framework.

Benefits of Collection Framework
.It provides useful data structures and algorithms that reduces programming effort due which we need not to write them ourselves.
.Collection is resizable and can grow.

Collection simply means a single unit of objects. Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

2. List Interface

It is the Child Interface of Collection. If we want to represent a group of individual objects where insertion order is preserved & duplicates are allowed. Then we should go for List. Vector and Stack classes are re engineered in 1.2 Version to set into Collection Framework.

Additional Operations Supported by "List" Interface over "Collection"
.Positional access-manipulate elements based on their numerical position in the list.
.Search-searches for a specified objects in the list and returns its numerical position.
.Iteration-extends Iterator semantics to take advantage of the list's sequential nature.
.Range view-performs arbitrary range operations on the list.

Implementations of "list" Interface are described below.

2.1. Array List

The ArrayList class extends AbstractList and implements the list interface. Array list supports dynamic arrays that can grow as needed. Think of ArraysList as vector without the synchronization overhead. Standard java arrays are of a fixed length. After

arrays are created,they cannot grow or shrink,which means that you must know in advance how many elements an array will hold. Arrays lists are created with an initial size. when this size is exceeded, the collection is automatically enlarged. when objects are removed,the array may be shrunk.

Constructor:
ArrayList AL=new ArrayList();
ArrayList defines following methods.
Void add(int index,object element)
-Inserts the specified element at the specified position index in this list.
Boolean add(Object o)
-Appends the specified element to the end of this list.
Boolean addAll(Collection c)
-Appends all of the elements in the specified collection to the end of this list.
Boolean addAll(int index,Collection c)
-Inserts all of the elements in the specified collection into this list,starting at the specified position.
Void clear()
Removes all of the elements from list.
Object clone()
Returns a shallow copy of this Arraylist.

```
Example:Demo of using ArrayList.
Import java.util.*;
public class ArrayListExample {
 public void doArrayListExample() {
ArrayList listA=new ArrayList();
listA.add("This");
listA.add("is");
listA.add("my");
listA.add("Program");
listA.add("on");
listA.add("ArrayList");
System.out.println("Objects in an ArrayList using index.\n");
for(int j=0;j<listA.size();j++) {
System.out.println("["+j+"]-"+listA.get(j));
}
int locationIndex=listA.indexOf("This");
System.out.println("location of\"This\"is:"+locationIndex);
System.out.println("LastIndex of\"This\"+listA.lastIndexOf("This"));
List listSub=listA.sublist(2,listA.size());
System.out.println("New sub-List from index 2 to "+listA.size());
System.out.println("\nOriginal sub List :"+Sub);
Collections.sort(listsub);
System.out.println("Sorted sub list:"+listSub);
Collections.reverse(listSub);
System.out.println("\nReversed subList :"+listSub);
System.out.println("\nIs List A empty?"+listA.isempty());
System.out.println("\n\"ArrayList"is Contained?="+listA.contains("ArrayList"));
System.out.println("\n\Remove\"my\"from ListA./\n);
listA.remove("my");
System.out.println("\n\Remove 3 position Element from ListA./\n");
listA.remove(3);
System.out.println("\update 4th position Element in listA.\n");
listA.set(4,"MA RAJU");
System.out.println("List A (before)Remove All:"+listA);
```

```
listA.clear();
System.out.println("List A (after) Remove All:"+listA);
}
public static void main(String[] args) {
 ArrayListExample listExample=new ArrayListExample();
listExample.doArrayListExample();
}
}
```

2.2. Linked List

Use it you frequently add elements to the beginning of the list or iterate over the list to delete elements from its interior.Implements serializable & clonable interfaces but not RandomAccess interfaces. Best suitable if our frequent operation is retrival.
Constructor:
1.LinkedList l=new LinkedList();
Creates an empty Linked object.
2.LinkedList l=new LinkedList(Collection c)
For interconversion between Collection objects.

LinkedList Specific Methods:
Usually we can use LinkedList to implements stacks & queues to support this requirement
LinkedList class define the Following Six Specific methods.
1.Void addFirst(Object o)
2.Void addLast(Object o)
3.Object removeFirst();
4.Object removeLast();
5.Object getFirst();
6.Object getLast();

2.3 Vector

Vector implements a dynamic array. It is similar to ArrayList,but with two Differences:
Vector is synchronized and vector contains many legacy methods are not part of the collections framework. Vector proves to be very useful if you don't know the size of the array in advance,or you just need one that can change sizes over the lifetime of a program. The Vector class supports four constructors. The first form creates a default vector,which has an initial size of 10:Vector().

The second form creates a vector whose initial capacity is specified by size:
Vector(int size).The third form creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward:Vector(int size,int incr).

Vector Defines the following methods:
Boolean add(Object o)
-Appends the specified element to the end of this vector.
Void addElement(Object obj)
-Appends the specified component to the end of this vector,increasing its size by one.
Int capacity()
-Returns all of the element from this vector.
Boolean contains(objects elem)
-Tests if the specified object is a component in this vector.
Void clear()
Removes all of the elements from Vector.

Object firstElement()
Returns the first component(item at index 0) of this vector.
Object get(int index)
Returns the element at the specified position in this vector.

Example:Demo working on Vector

```
import java.util.*;
class vector
{
public static void main (String [] args)
{
Vector v=new Vector ();
v.addElement (new Integer (10));
v.addElement (new Float (100.37f));
v.addElement (new Boolean (true));
v.addElement ("K.V.R");
System.out.println ("SIZE = "+v.size ());
System.out.println ("CONTENTS = "+v);
Enumeration e
while (en.hasMoreElements ())
{
Object val=en.nextElement ();
System.out.println (val);
}
}
};
```

Difference between ArrayList and Vector

ArrayList	Vector
1.No Method is Synchronized.	1.Every Method is synchronized
2.Multiple Threads access ArrayList Simultaneously. Hence ArrayList is not a Thread Safe	2.At any point only one thread is allowed to operate on vector object at any time. Hence Vector object is Thread safe.
3.Threads are not required to wait,& Hence performance is high.	3.It increases waiting time of threads & Hence performance is low.
4.Introduced in 1.2 Version & Hence it is non-Legacy.	4.Introduced in 1.0 version & Hence it is legacy.

3.Set Interface
A collection that cannot contain duplicate elements. Models the mathematical set abstraction and is used to represent sets. The set interface contains only methods inherited from collection and adds the restriction that duplicate elements are prohibited and some of the examples like

- Cards comprising a poker hand
- courses making up a student's schedule

- the process running on a machine

3.1. HashSet:

HashSet is much faster than TreeSet (constant-time versus log-time for most operations) but offer no ordering guarantees and mostly commonly used implementations.

Example: Demo of using HashSet.

```
Import java.util.*;
public class HashSetDemo  {
public void doHasSetExample() {
HashSet setA=new HashSet();
SetA.add("This");
SetA.add("is");
SetA.add("my");
SetA.add("firsy");
SetA.add("program");
SetA.add("using");
SetA.add("HashSet");
System.out.println("\Retrieve objects in an HashSet container using index.\n");
Iterator i=setA.iterator();
while(i.hasNext()) {
System.out.println(i.next());
}
System.out.println("\nIs Set A empty? "+setA.isEmpty());
System.out.println("\n\"HashSet\" is Contained?="+setA.contains("HashSet");
System.out.println("\n\Remove\ "This\" element from setA.\n");
setA.remove("my");
System.out.println("\n\Remove 3rd Position  element from SetA.\n");
setA.remove(3);
System.out.println("Set A (before) remove all:"+setA);
SetA.clear();
System.out.println("Set A (after) remove all:"+setA);
}
public static void main(String[] args)  {
HashSetDemo setExample=new HashSetDemo();
setExample.doHashSetExample();
}
}
```

3.2. TreeSet

The TreeSet implementation is useful when you need to extract elements from a collection in a stored manner. TreeSet provides an implementation of the set interface that use a tree for storage. Objects are stored in sorted ,ascending order. Acess and retrieval times are quite fast,which makes treeset an excellent choice when storing large amounts of sorted information that must be found quickly.

3.3. LinkedHashSet:

It is a class which is implements both the hash table and linked list implementation of set interface. The orders of its elements are based on the order in which they were inserted into the set (Insertion-order).

Example:To demonstrate set interfaces:

The numbers are added in the following order 2 3 4 1 2

Set type=java.util.HashSet[3,2,4,1]

Set type=java.util.TreeSet[1,2,3,4]

Set type=java.util.LinkedSet[2,3,4,1]

We can traverse the collection using traditional for loop but to increase the efficiency,we can use Enumeration or Iterator or ListIterator.
.The elements() method of the collection interface returns Enumeration.
.The iterator() method of the collection interface returns Iterator.
.listIterator() method of the collection interface returns ListIterator.
Enumeration:Is used to traverse the collection.

4. Map Interface:
If we want to represent a group of objects as Key-Value pairs then we should go for Map.Both Key&Values are objects only. Duplicate Keys are not allowed,But values can be duplicated.

4.1. HashMap
A HashMap contains values based on the key. It implements the Map interface and extends AbstractMap class.
.It contains only unique elements.
.It may have one null key and multiple null values.
.It maintains no order.

Example: Demo of working with HashMap

```
import.java.util.*;
class Simple{
public static void main(String args[]){
HashMap hm=new HashMap();
hm.put(100,"Amit");
hm.put(101,"Vijay");
hm.put(102,"Rahul");
Set set=hm.entrySet();
Iterator itr=set.iterator();
while(itr.hasNext()){
Map.Entry m=(Map.Entry)itr.next();
System.out.println(m.getKey()+" "+m.getValue());
}
}
}
```

4.2. HashTable
A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key. It implements the Map interface and extends Dictionary class.

It contains only unique elements.

It may have not have any null key or value.
It is synchronized.

Example: Demo of working with HashTable

```
import java.util.*;
class Simple{
public static void main(String args[]){
Hashtable hm=new Hashtable();

hm.put(100,"Amit");
hm.put(102,"Ravi");
hm.put(101,"Vijay");
hm.put(103,"Rahul");
Set set=hm.entrySet();
Iterator itr=set.iterator();
while(itr.hasNext()){
Map.Entry m=(Map.Entry)itr.next();
System.out.println(m.getKey()+""+m.getValue());
}
}
}
```

What is difference between HashMap and Hashtable?

HashMap	HashTable
1.HashMap is not Synchronized.	1.HashTable is Synchronized
2.Hash map can contain one null key and multiple null values.	2. Hash table cannot contain any null key nor null.

4.3. TreeMap

A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.It contains only unique elements.It cannot have null key but can have multiple null values.It is same as HashMap instead maintains ascending order.

Example: Demo working of TreeMap

```
import java.util.*;
class Simple{
public static void main(String args[]){
TreeMap hm=new TreeMap();

hm.put(100,"Amit");
hm.put(102,"Ravi");
hm.put(101,"Vijay");
hm.put(103,"Rahul");

Set set=hm.entrySet();
Iterator itr=set.iterator();
while(itr.hasNext()){
Map.Entry m=(Map.Entry)itr.next();
System.out.println(m.getKey()+""+m.getValue());
}
}
}
```

4.4. LinkedHashMap

A LinkedHashMap contains values based on the key. It implements the Map interface and extends HashMap class.It contains only unique elements.It may have one null key and multiple null values.It is same as HashMap instead maintains insertion order.

Example: Demo on LinkedHashMap working

```
import java.util.*;
class Simple{
public static void main(String args[]){
LinkedHashMap hm=new LinkedHashMap();
hm.put(100,"Amit");
hm.put(101,"Vijay");
hm.put(102,"Rahul");
Set set=hm.entrySet();
Iterator itr=set.iterator();
while(itr.hasNext()){
Map.Entry m=(Map.Entry)itr.next();
System.out.println(m.getKey()+""+m.getValue());
}
}
}
```

Order of map elements in iteration:The order of the elements obtained from a map depends on the type of map they came from.
.TreeMap: The elements are ordered by key.
.HashMap: The elements will be in an unpredictable,"chaotic",order.
.LinkedHashMap: The elements will be ordered by entry order or last reference order,depending on type of linkedHashMap.

Property	HashSet	LinkedHashSet	TreeSet
1.UnderLying D.S	Hash Table	Hash Table+Linked List	Balanced Tree
2.Insertion Order	Not Preserved	Preserved	Not Preserved
3.Sorting Order	Na	Na	Preserved
4.Heterogenous objects	Allowed	Allowed	Not Allowed
5.Duplicates objects	Not Allowed	Not Allowed	Not Allowed
6.Null Acceptance	Allowed	Allowed	Not Allowed

5. ListIterator Interface

ListIterator Interface is used to traverse the element in backward and forward direction.

Commonly used methods of ListIterator Interface:

1.public boolean hasNext();
2.public Object next();
3.public boolean hasPrevious();
4.public Object previous();

Example of ListIterator Interface:

```
Import java.util.*;
class Simple5{
public static void main(String args[]){
ArrayList al=new ArrayList();
al.add("Amit");
al.add("Vijay");
al.add("Kumar");
al.add(1,"Sachin");
System.out.println("element at 2nd  position:"+al.get(2));
ListIterator itr=al.listIterator();
System.out.println("traversing elements in forward direction...");
while(itr.hasNext()){
System.out.println(itr.next());
}
System.out.println("traversing elements in backward direction...");
while(itr.hasPrevious()){
System.out.println(itr.previous());
}
}
}
```

Note:

All the above Interfaces(List,Set) ment for representing a group of individual objects. If we want to represent a group of objects as key-Value pairs then we should go for "Maps".Map is not child Interface of Collection.

6. Properties class

It is the child class of HashTable. In our program if any thing which changes frequently(like database usernames,passwords,url) never recommended to hardcode the value in the java program. Because for every change,we have recompile,rebuild, redeployee the application & sometime even sever restart also required. Which creates a big business impact to the client. We have to configure those variables(inside properties files & we have to read those values from javacode). The main advantage of this approach is,If any change in the properties file just redeployment is enough which is not a business impact to the client.

Constructor:

1.Properties p=new Properties();
In the case of properties both Key & value should be string type.

Methods:
1.String getProperty(String Propertyname)
Returns the value associated with specified property.
2.String setProperty(String pname,String pvalue)
To set a new property.
3.Enumeration propertyNames();
4.Void load(InputStream is)
To load the properties from properties files into java properties object.
5.Void store(OutputStream os,String comment)
To update properties from properties object into properties file

Easy Maintenance: If any information is changed from the properties file, you don't need to recompile the java class. It is mainly used to contain variable information i.e. to be changed. This class provides methods for the following:
.loading key/value pairs into a properties object from a stream,
.retrieving a value from its key,
.listing the keys and their values,
.enumerating over the keys,and
.saving the properties to a stream.

Example: Demo on Properties.

```
Import java.util.*;
import java.io.*;
class propertiesDemo
{
   public static void main(String args[])throws Io Exeception
     {
        properties p=new properties();
        FileInputStream fis=new FileInputStream("abc.properties");
        p.load(fis);
        System.out.println(p);
        String s=p.getProperty("suresh");
        System.out.println(s);
        p.setProperty("chapala","999999");
        FileOutputStream fos=new FileOutputStream("abc.properties");
        p.store(fos,"updated by suresh for Demo class");
      }
    }
```

7.Generics
Generics are more useful in application development to write the application with more generality.

In J2SE 1.0 to J2SE1.4 earlier,the collection types were like below.	In J2SE 5.0,they have been changed to like below.
Interface collection{	Interface collection<E>{

void add(Object obj);
}
Interface List extends Collection {
Object get(int idx);
}
Class ArrayLst implements List {
}
	void add(E obj);
	}
	Interface List<E>extends<E>Collection {
	E get(int idx);
	}
	Class ArrayLst<E>implements List<E> {
	}

Here the "E" represents the type of the element of the collection. It is called a "type parameter".When you use them,you need to specify a type as the actual value of E.

Example:
List <String>list=new ArrayList<String>(); // It reads "a List of Strings"
It means that you can call add() and pass a String object but not other types of objects.When you call get(),it will return a string,not an object:

```
List.add("hello"); //OK
list.add(new Integer(10)); // ERROR!. But in J2SE 1.4 it would be ok
String s=list.get(0); //No need to type cast to String anymore
```

When you provide a type argument,resulting types such as List<String> or ArrayList<String> are called "Paremeterized types",while the original types (ArrayList) are called "raw types".The act of providing a type argument is called "invocation".

Similarly,the set interface an its implementation classes are also generic:

```
Set<Integer>Set =new TreeSet<Integer>();//Integer implements Comparable
set.add(new Integer(2));
set.add(5);//Ok.Autoboxing.
Set.add("hello");//Error!
if(set.contains(2)) { //it will be true
}
```

Iterators are also generic.

```
List<String>list =new ArrayList<String>();//a List of Strings
list.add("a");
list.add("b");
for(Iterator<String>iter=list.iterator();iter.hasNext();) {
String s=itr.next(); //No need to type cast
System.out.println(s);
}
```

Now onwards we can make all map imlemented classes as generic.

```
Import java.util.HashMap;
public class HashGeneric {
  public static void main(String args[]) {
HashMap<Integer String>Map=new HashMap<Integer,String>();
Map.put(1,"suresh");
Map.put(43,"satish");
Map.put(143,"Arun");
String name=map.get(43);
System.out.println(name);
}
}
```

Example:Displaying the elements of arrayList using foreach.

```
Import java.util.*;
Public class ForeachDemo {
static void iterate(collection<String>c ) {
for(String s:c)
system.out.println(s);
  }
public static void main(String args[]) {
List<String>l=new ArrayList<String>();
I.add("suresh");
I.add("satish");
iterate(i);
  }
  }
```

7.1 Writing generic methods

We can write a method,which can be generic,means it can be called by any parameter and behavior of the method will be according to the parameter passed.For example assume that we need to write a method which takes an array and converts it to list.assume that we want to do it with integers,we can write the method easily and what if we want to achieve the same functionality with Strings,do we need to write one method.Not necessary in 1.5 we can achieve this easily with generic methods.

Example: Demonstration of writing generic methods.

```
Import java.util.*;
public class GenericDemo {
public static void main(String[] args) { GenericDemo genericMethod=new
GenericDemo(); List<String>sList=genericMethod.convertArrayToList(new
String[]{"2","3"); System.out.println(slist.toString());
List<Integer>iList=genericMethod.convertArrayToList(new Integer[]{2,3});
System.out.println(ilist.toString());
     }
```

```
public <T>List <T> convertArrayToList(T[] array)  {
List<T>list =new ArrayList<T>();
for (T.element:array) {
list.add(element);
     }
return list;
     }
     }
```

7.2 Writing Generic Class

Java5.0 allows class definitions with parameters for types.These classes that have type parameters are called parameterized class or generic definitions,or,simply,generic class.
.A class definition with a type parameters is stored in a file and complied just like any other class.
.Once a parameterized class is complied,it can be used like any other class.
.However,the class type plugged in for the type parameter must be specified before it can be used in a program.
.Doing this is said to instantiate the generic class.

Ex: Sample<String>object=new Sample<String>();

.A class that is defined with a parameter for type is called a generic class or a parameterized class.
.The type parameter is included in angular brackets after the class name in the class definition heading.
.Any non-Keyword identifier can be used fro the type parameter,but by convention,the parameter starts with an uppercase letter.
.The type parameters can be used like other types used in the definition of a class.

Example: Demonstration of writing generic class.

```
  Public class GenericClass<T> {
       private T a;
       public GenericClass(T x) {
           a=x;
             }
           public T show;
            {
           return a;
             }
          public static void main(String a[])
            {
       GenericClass<Integer>igen=new GenericClass<Integer>(13);
       System.out.println("integer="+igen.show());
       GenericClass<Integer>igen=new GenericClass<String>("java");
        System.out.println("string="+sgen.show());
            }
            }
```

8. Advantages of Collections over Arrays

1.Collections are growable in nature.Hence based on our requirement we can increase or decrease the size.
2.Collections can hold both Homogeneous and heterogeneous objects.

3.Every Collection class is implemented based on some data structure ,Hence readymade method support is available for every Requirement.