R–Creation

track of IT

215/318, Hasina Monjil (1st Floor), Muradpur, Panchlaish, Chittagong.
Website: www.rcreation-bd.com, Contact: +8801722-964303

Lecture – 14 (Collection Framework)

Collections in Java

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

What is Collection in java

Collection represents a single unit of objects i.e. a group.

What is framework in java

- provides readymade architecture.
- represents set of classes and interface.
- is optional.

What is Collection framework

Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

1. Interfaces and its implementations i.e. classes
2. Algorithm

Java ArrayList class

Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

The important points about Java ArrayList class are:

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.

- Java ArrayList class is non-synchronized.
- Java ArrayList allows random access because array works at the index basis.
- In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Java ArrayList Example

```
import java.util.*;
class TestCollection1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  list.add("Jon");//Adding object in arraylist
  list.add("Tony");
  list.add("Sejan");
  list.add("Lincoln");
  //Traversing list through Iterator
  Iterator itr=list.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Two ways to iterate the elements of collection in java

There are two ways to traverse collection elements:

1. By Iterator interface.
2. By for-each loop.

In the above example, we have seen traversing ArrayList by Iterator. Let's see the example to traverse ArrayList elements using for-each loop.

Iterating Collection through for-each loop

```
import java.util.*;
class TestCollection2{
 public static void main(String args[]){
  ArrayList<String> al=new ArrayList<String>();
  al.add("Jon");
  al.add("Tony");
  al.add("Sejan");
  al.add("Ershad");
  for(String  obj:al)
    System.out.println(obj);
 }
}
```

User-defined class objects in Java ArrayList

Let's see an example where we are storing Student class object in array list.

```
class Student{
  int rollno;
  String name;
  int age;
  Student(int rollno,String name,int age){
   this.rollno=rollno;
   this.name=name;
   this.age=age;
  }
}

import java.util.*;
public class TestCollection3{
 public static void main(String args[]){
  //Creating user-defined class objects
  Student s1=new Student(101,"Jon",23);
  Student s2=new Student(102,"Sejan",21);
  Student s2=new Student(103,"Tony",25);
  //creating arraylist
  ArrayList<Student> al=new ArrayList<Student>();
  al.add(s1);//adding Student class object
  al.add(s2);
  al.add(s3);
  //Getting Iterator
  Iterator itr=al.iterator();
  //traversing elements of ArrayList object
  while(itr.hasNext()){
   Student st=(Student)itr.next();
   System.out.println(st.rollno+" "+st.name+" "+st.age);
  }
 }
}
```

Example of addAll(Collection c) method

```
import java.util.*;
class TestCollection4{
 public static void main(String args[]){
  ArrayList<String> al=new ArrayList<String>();
  al.add("Jon");
  al.add("Sejan");
  al.add("Ershad");
  ArrayList<String> al2=new ArrayList<String>();
  al2.add("Lincoln");
  al2.add("Rumman");
  al.addAll(al2);//adding second list in first list
  Iterator itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Java LinkedList class

Java LinkedList class uses doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.

- Java LinkedList class maintains insertion order.
- Java LinkedList class is non-synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
- Java LinkedList class can be used as list, stack or queue.

Java LinkedList Example

```
import java.util.*;
public class TestCollection7{
 public static void main(String args[]){

  LinkedList<String> al=new LinkedList<String>();
  al.add("Jon");
  al.add("Tony");
  al.add("Sagar");
  al.add("Sejan");

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

Java List Interface

List Interface is the subinterface of Collection.It contains methods to insert and delete elements in index basis.It is a factory of ListIterator interface

Example of ListIterator Interface

```
import java.util.*;
public class TestCollection8{
public static void main(String args[]){
ArrayList<String> al=new ArrayList<String>();
al.add("Jon");
al.add("Tony");
al.add("Sejan");
al.add(1,"Ershad");
System.out.println("element at 2nd position: "+al.get(2));
ListIterator<String> 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());
}
}
}
```

Java Vector

Vector implements a dynamic array. It is similar to ArrayList, but with two differences −

- Vector is synchronized.
- Vector contains many legacy methods that 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.

Let's see a simple example of java Vector class that uses Enumeration interface.

```
import java.util.*;
class TestVector1{
 public static void main(String args[]){
  Vector<String> v=new Vector<String>();//creating vector
  v.add("umesh");//method of Collection
  v.addElement("Jon");//method of Vector
  v.addElement("kabir");
  //traversing elements using Enumeration
  Enumeration e=v.elements();
  while(e.hasMoreElements()){
   System.out.println(e.nextElement());
  }
 }
}
```

Difference between ArrayList and Vector

ArrayList and Vector both implements List interface and maintains insertion order.

But there are many differences between ArrayList and Vector classes that are given below.

ArrayList	**Vector**
1) ArrayList is **not synchronized**.	Vector is **synchronized**.
2) ArrayList **increments 50%** of current array size if number of element exceeds from its capacity.	Vector **increments 100%** means doubles the array size if total number of element exceeds than its capacity.
3) ArrayList is **not a legacy** class, it is introduced in JDK 1.2.	Vector is a **legacy** class.
4) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object.
5) ArrayList uses **Iterator** interface to traverse the elements.	Vector uses **Enumeration** interface to traverse the elements. But it can use Iterator also.

Java HashMap class

Java HashMap class implements the map interface by using a hashtable. It inherits AbstractMap class and implements Map interface.

The important points about Java HashMap class are:

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

Hierarchy of HashMap class

As shown in the above figure, HashMap class extends AbstractMap class and implements Map interface.

HashMap class declaration

Let's see the declaration for java.util.HashMap class.

1. public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Serializable

HashMap class Parameters

Let's see the Parameters for java.util.HashMap class.

- **K**: It is the type of keys maintained by this map.
- **V**: It is the type of mapped values.

Java HashMap Example

```
import java.util.*;
class TestCollection13{
 public static void main(String args[]){
  HashMap<Integer,String> hm=new HashMap<Integer,String>();
  hm.put(100,"Jon");
  hm.put(101,"Kabir");
  hm.put(102,"Sagar");
  for(Map.Entry m:hm.entrySet()){
   System.out.println(m.getKey()+" "+m.getValue());
  }
 }
}
```

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. Map contains only unique keys.

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

Map.Entry Interface

Entry is the sub interface of Map. So we will be accessed it by Map.Entry name. It provides methods to get key and value.

Methods of Map.Entry interface

Method	Description
Object getKey()	It is used to obtain key.
Object getValue()	It is used to obtain value.

Java Map Example: Generic (New Style)

```
import java.util.*;
class MapInterfaceExample{
 public static void main(String args[]){
  Map<Integer,String> map=new HashMap<Integer,String>();
  map.put(100,"Jon");
  map.put(101,"Sagar");
  map.put(102,"Tony");
  for(Map.Entry m:map.entrySet()){
   System.out.println(m.getKey()+" "+m.getValue());
  }
 }
}
```