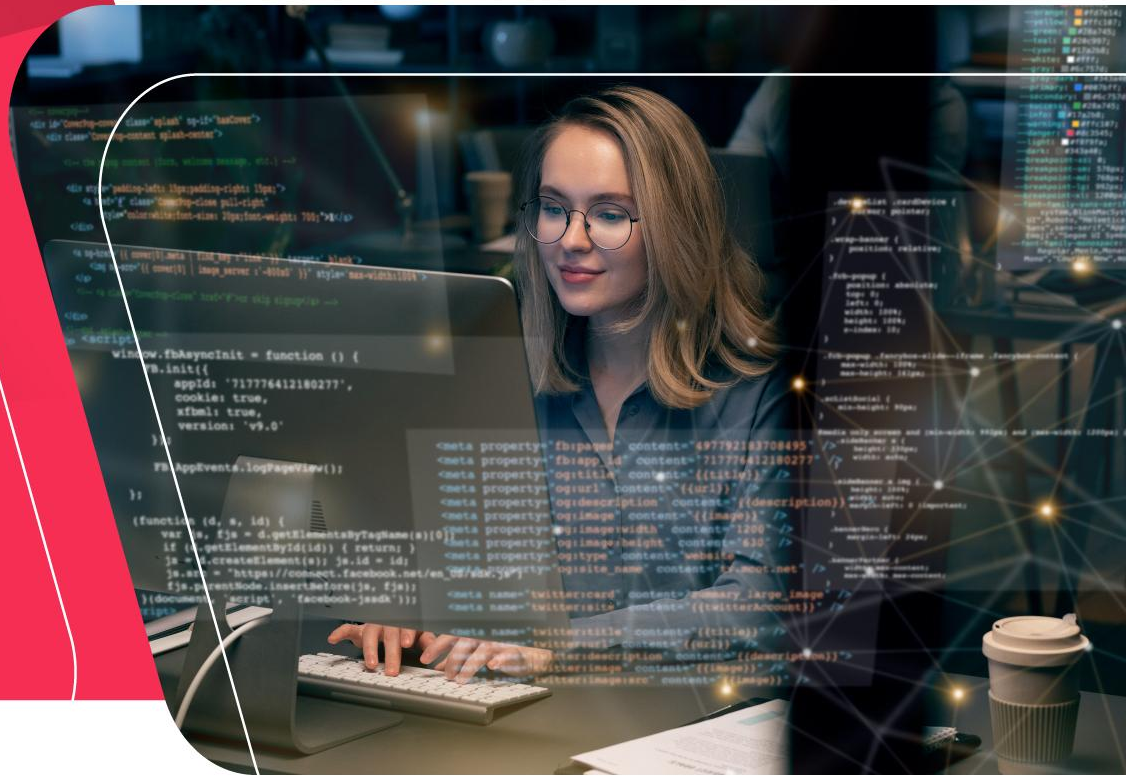
TIA-Academy

Java Bootcamp

Day 09



Technologies will be Use

- JDK 8/**11**/15
- JRE 8/**11**/15
- Writing Code using Notepad++ (For Windows) or Vim (For Linux and Mac)
- Compiling with Command Prompt (For Windows) or Terminal (For Linux and Mac)
- ***Forbidden of using IDE Based Development Tools as of Intellij IDEA or Eclipse***

TIA-Academy

ASSIGNMENT 01

TIA-Academy

Collections/Data Structures in Java

Collections

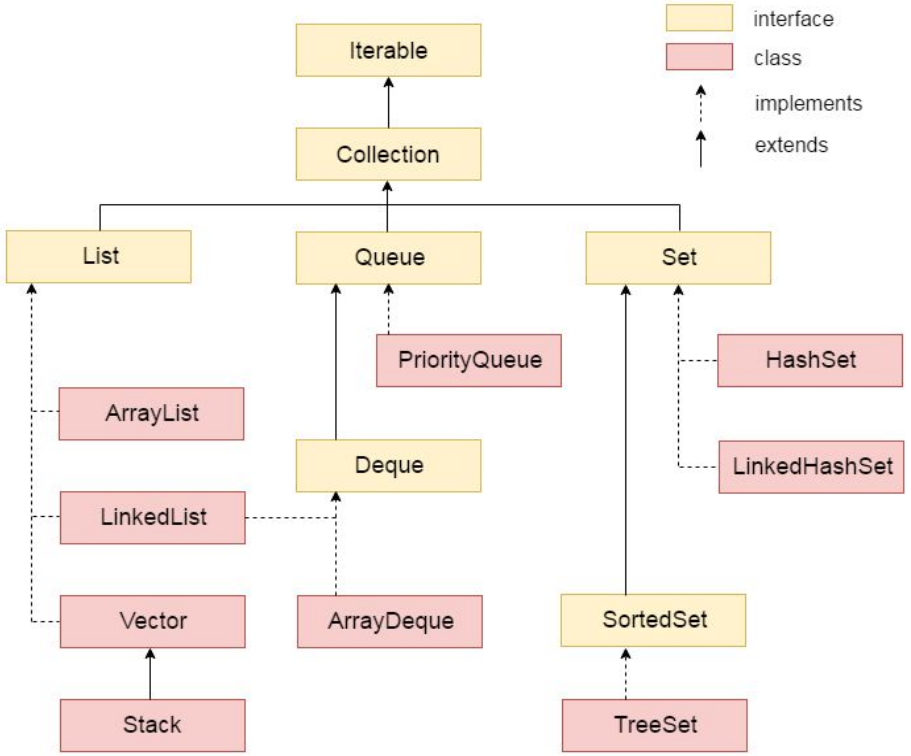
- **Collections in java** is a framework that provides an architecture to store and manipulate the group of objects.
- All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.
- Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (**ArrayList**, Vector, **LinkedList**, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

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:
  - Interfaces and its implementations i.e. classes
  - Algorithm

Hierarchy of Collection Framework

- Let us see the hierarchy of collection framework.
- The **java.util** package contains all the classes and interfaces for Collection framework.



Hierarchy of Collection Framework

No.	Method	Description
1	public boolean add(Object element)	is used to insert an element in this collection.
2	public boolean addAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(Object element)	is used to search an element.
9	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public boolean isEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public int hashCode()	returns the hashcode number for collection.

Iterator interface

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

- There are only three methods in the Iterator interface. They are:

No.	Method	Description
1	public boolean hasNext()	It returns true if iterator has more elements.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is rarely used.

Java ArrayList

- 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 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

Java ArrayList

- Hierarchy of ArrayList class, Java ArrayList class extends AbstractList class which implements List interface.
- The List interface extends Collection and Iterable interfaces in hierarchical order.
- Let's see the declaration for java.util.ArrayList class.

```
public class ArrayList<E> extends AbstractList<E> implements List<E>, Rand
omAccess, Cloneable, Serializable
```

Java ArrayList

- Constructors of Java ArrayList

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Java ArrayList

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
boolean addAll(Collection c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
void clear()	It is used to remove all of the elements from this list.
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.
Object[] toArray(Object[] a)	It is used to return an array containing all of the elements in this list in the correct order.
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean addAll(int index, Collection c)	It is used to insert all of the elements in the specified collection into this list, starting at the specified position.
Object clone()	It is used to return a shallow copy of an ArrayList.
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
void trimToSize()	It is used to trim the capacity of this ArrayList instance to be the list's current size.

- Methods of Java ArrayList

Java ArrayList Example

```
import java.util.*;
class TestCollection1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  list.add("Ravi");//Adding object in arraylist
  list.add("Vijay");
  list.add("Ravi");
  list.add("Ajay");
  //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:
  - By Iterator interface.
  - 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("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");
  for(String obj:al)
    System.out.println(obj);
 }
}
```

User-defined class objects in Java ArrayList

```
class Student{
  int rollno;
  String name;
  int age;

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

User-defined class objects in Java ArrayList

```
import java.util.*;
public class TestCollection3{
 public static void main(String args[]){
  //Creating user-defined class objects
  Student s1=new Student(101,"Sonoo",23);
  Student s2=new Student(102,"Ravi",21);
  Student s3=new Student(103,"Hanumat",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);
  }
 }
}
```

TIA-Academy

ASSIGNMENT 02

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("Ravi");
  al.add("Vijay");
  al.add("Ajay");
  ArrayList<String> al2=new ArrayList<String>();
  al2.add("Sonoo");
  al2.add("Hanumat");
  al.addAll(al2);//adding second list in first list
  Iterator itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Example of removeAll() method

```
import java.util.*;
class TestCollection5{
 public static void main(String args[]){
  ArrayList<String> al=new ArrayList<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ajay");
  ArrayList<String> al2=new ArrayList<String>();
  al2.add("Ravi");
  al2.add("Hanumat");
  al.removeAll(al2);
  System.out.println("iterating the elements after removing the elements of al2...");
  Iterator itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
  }
}
```

Example of retainAll() method

```
import java.util.*;
class TestCollection6{
 public static void main(String args[]){
  ArrayList<String> al=new ArrayList<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ajay");
  ArrayList<String> al2=new ArrayList<String>();
  al2.add("Ravi");
  al2.add("Hanumat");
  al.retainAll(al2);
  System.out.println("iterating the elements after retaining the elements of al2...");
  Iterator itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

TIA-Academy

ASSIGNMENT 03

TIA-Academy

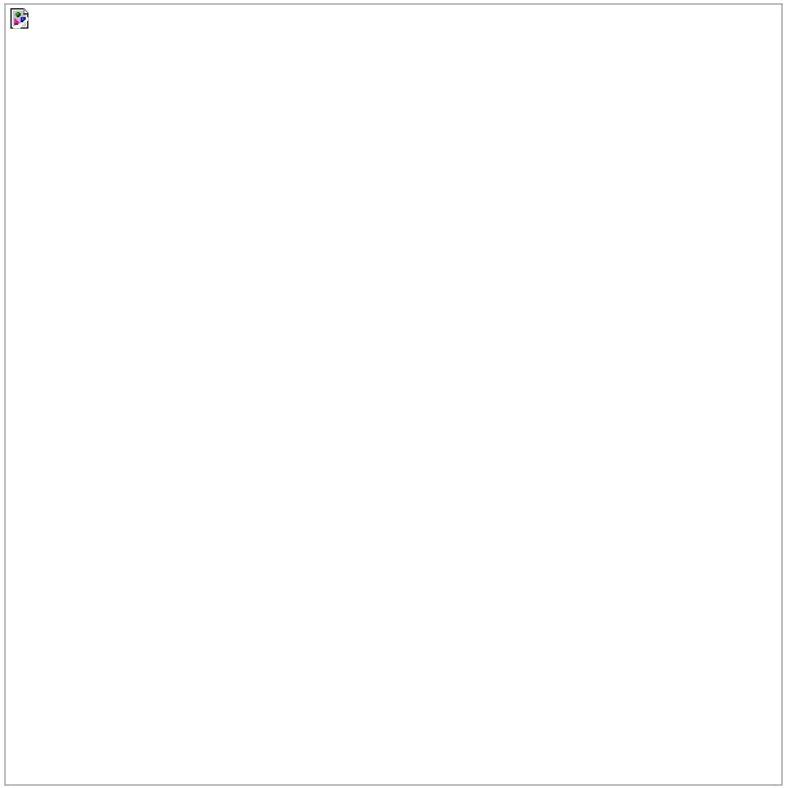
Java Map

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.

Java Map Hierarchy

- There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap, and TreeMap.
- The hierarchy of Java Map is given beside:



Java Map

- 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.
- A Map can't be traversed, so you need to convert it into Set using *keySet()* or *entrySet()* method.

Class	Description
HashMap	HashMap is the implementation of Map, but it doesn't maintain any order.
LinkedHashMap	LinkedHashMap is the implementation of Map. It inherits HashMap class. It maintains insertion order.
TreeMap	TreeMap is the implementation of Map and SortedMap. It maintains ascending order.

Java Map Example: Non-Generic (Old Style)

- Output :

```
1 Amit
2 Jai
5 Rahul
6 Amit
```

```
//Non-generic
import java.util.*;
public class MapExample1 {
        public static void main(String[] args) {
            Map map=new HashMap();

            //Adding elements to map
            map.put(1,"Amit");
            map.put(5,"Rahul");
            map.put(2,"Jai");
            map.put(6,"Amit");
            //Traversing Map

            Set set=map.entrySet();//Converting to Set so that we can traverse
            Iterator itr=set.iterator();
            while(itr.hasNext()){
                //Converting to Map.Entry
                 //so that we can get key and value separately
                Map.Entry entry=(Map.Entry)itr.next();
                System.out.println(entry.getKey()+" "+entry.getValue());
            }
        }
}
```

Java Map Example: Generic (New Style)

- Output :

```
102 Rahul
100 Amit
101 Vijay
```

```
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());
  }
 }
}
```

Java Map Example: comparingByKey()

- Output :

```
100=Amit
101=Vijay
102=Rahul
```

```
import java.util.*;

class MapExample3{
      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");

            //Returns a Set view of the mappings contained in this map
            map.entrySet()

            //Returns a sequential Stream with this collection as its source
            .stream()

            //Sorted according to the provided Comparator
            .sorted(Map.Entry.comparingByKey())

            //Performs an action for each element of this stream
            .forEach(System.out::println);
      }
}
```

Java Map Example: comparingByValue()

- Output :

```
100=Amit
102=Rahul
101=Vijay
```

```
import java.util.*;

class MapExample5{
      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");

              //Returns a Set view of
              //the mappings contained in this map
              map.entrySet()

              //Returns a sequential Stream
              //with this collection as its source
              .stream()

              //Sorted according to the provided Comparator
              .sorted(Map.Entry.comparingByValue())

              //Performs an action for each element of this stream
              .forEach(System.out::println);
      }
}
```

TIA-Academy

ASSIGNMENT 04

TIA-Academy

Thank You