Collections FrameWork

Java platform includes a collections framework.
A collection is an object that represents a group of objects.
A collection framework is a unified architecture for representing and manipulating collections, enabling collections to be manipulated independently of implementation details.

Advantages:

(i) Reduces programming effort
(ii) Increases performance
(iii) Provides interoperability between unrelated APIs
(iv) Reduces the effort required to learn APIs.
(v) Reduces the effort required to design and implement APIs
(vi) Fosters software reuse.

The collection framework consists of

(i) Collection interfaces: Represents different types of collections such as sets, lists and maps.
(ii) General purpose implementations.
(iii) Legacy implementations
(iv) Special purpose implementations.
(v) Concurrent implementations
(vi) Wrapper implementations
(vii) Convenience implementations.

(viii) Abstract Implementations
(ix) Algorithms.
(x) Infrastructure.
(xi) Array Utilities.

Collection Interfaces :-

The collection interfaces are divided into 2 groups.

(i) java.util.Collection
- java.util.Set
- java.util.SortedSet
  - .NavigableSet
  - .Queue
  - .concurrent.BlockingQueue
  - .concurrent.TransferQueue
  - .Deque
  - .concurrent.BlockingDeque.

(ii) java.util.Map
- java.util.SortedMap
  - .NavigableMap
  - .concurrent.ConcurrentMap
  - .concurrent.ConcurrentNavigableMap

→ Collections that do not support modification operations such as add, remove and clear etc. are referred to as unmodifiable.

→ Collections that additionally guarantee that no change in the collection object will be visible are referred to as immutable.

→ Lists that support fast indexed element access are known as random access lists.

Collections in Java :-

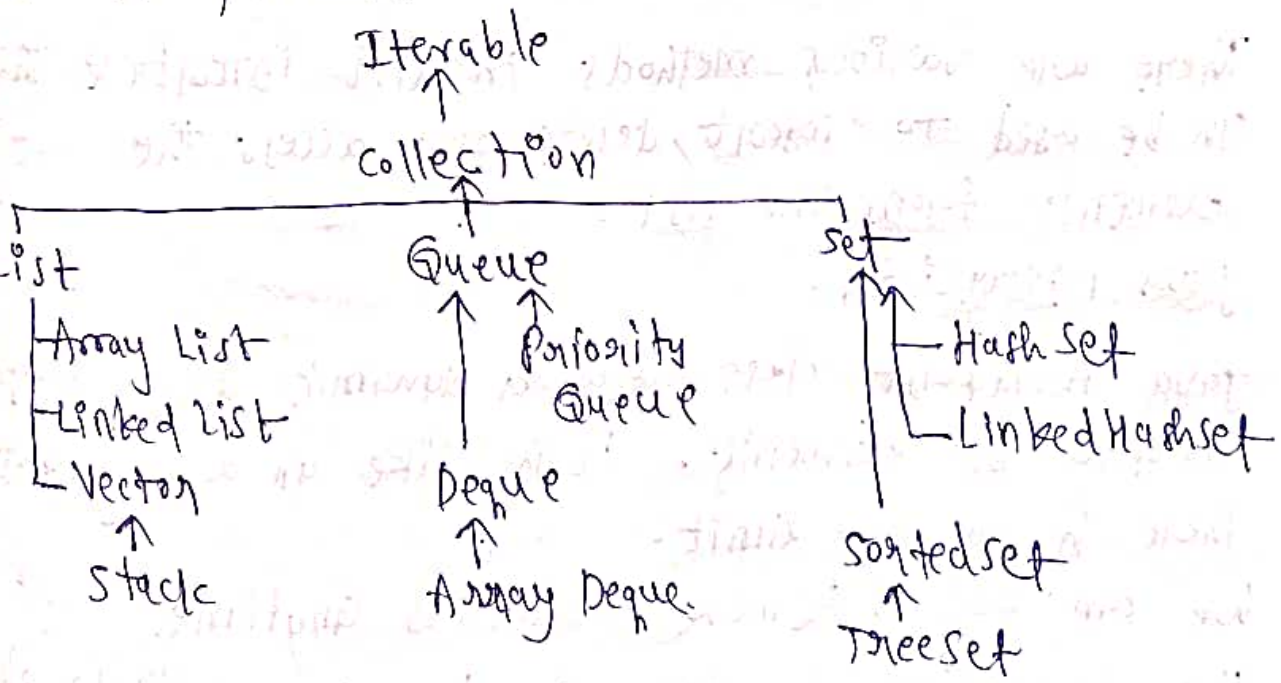
The collection in java is a framework that provides an architecture to store and manipulate the group of objects.

Java collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Framework in Java :-

→ It provides readymade architecture.
→ It represents a set of classes and interfaces
→ It is optional.



Collection Interface :-

The collection interface is the interface which is implemented by all the classes in the collection framework.

List Interface :-

List interface is the child interface of collection interface.

It is a list type data structure in which we can store the ordered collection of objects.

It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector and Stack.

```
List <data-type> list1 = new ArrayList();
List <data-type> list2 = new LinkedList();
List <data-type> list3 = new Vector();
List <data-type> list4 = new Stack();
```

There are various methods in list interface that can be used to insert, delete and access the elements from the list.

Java ArrayList :-

Java ArrayList class uses a dynamic array for storing the elements. It is like an array, but there is no size limit.

We can add or remove elements anytime.

The ArrayList in java can have the duplicate elements

The important points about the java arraylist are
→ java arraylist can contain duplicate elements.
→ maintains insertion order
→ it is non synchronized.
→ allows random access.
→ manipulation is a little bit slower than the linked list
→ we can not create an array list of the primitive types. It uses the required wrapper types.

```
ArrayList<Integer> al = new ArrayList<Integer>();
```

Ex:
```
import java.util.*;
public class ArrayListExample{
  public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();
  list.add("Mango");
  list.add("Apple");
  list.add("Banana");
  System.out.println(list);
}}
```

Java LinkedList:-

It uses a doubly linked list to store the elements.
It provides a linked-list data structure.
→ Java LinkedList class can contain duplicate elements
→ maintains insertion order
→ It is non synchronized.

→ Manipulation is fast.
→ Can be used as a list, stack (or) queue

Eg:-
```
public class LinkedListDemo{
    public static void main(String args[]){
        LinkedList<String> al=new LinkedList<String>();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ajay");
        Iterator<String> itr=al.iterator();
        while(itr.hasNext()){
            s.o.p(itr.next());
        }
    }
}
```

l.remove(
l.addFirst("
l.addLast("
l.removeFirst
l.removeLast(

ArrayList	LinkedList
→ It uses a dynamic array to store the elements	→ It uses a doubly linked list to store the elements
→ Manipulation is slow	→ Manipulation is fast.
→ Can acts as a list only	→ Can acts as a list and Queue.
→ It is better for sorting and accessing	→ It is better for manipulating.

Java Hash set :-

It is used to create a collection that uses a hash table for storage.

→ Hash set stores the elements by using the mechanism called hashing.
→ It contains unique elements
→ It allows null value.
→ It is non synchronized.
→ It does n't maintain the insertion order.
→ It is best approach for search operations.

```
import java.util.*;
class HashSetDemo {
public static void main (String args[]) {
HashSet<String> set=new HashSet();
set.add("One");
set.add("Two");
Iterator<String> i=set.iterator();
while (i.hasNext())
{
S.o.p(i.next());
}
}
}
```

```
set.remove();
set.removeAll();
set.removeIf();
set.clear();
```

Java Treeset class:-

Java Treeset class implements the set interface that uses tree for storage.

→ Java Treeset class contains unique elements like Hashset

→ Access and retrieval times are fast.

→ doesn't allow null element.

→ It is non synchronized.

→ It maintains ascending order.

Eg:

```
import java.util.*;
    class TreeSetDemo{
    public static void main(String args[]){
  TreeSet<String> al=new TreeSet<String>();
  al.add("Ravi");
  al.add("Vijay");
  Iterator<String> itr=al.iterator();
  while(itr.hasNext())
    {
     S.O.P(itr.next());
    }
  }
}
```

Priority Queue:

It is used when the objects are supposed to be processed based on the priority.
It is known that a Queue follows FIFO algorithm.
The elements of the priority queue are ordered a/c to the natural ordering.

→ Priority Queue doesn't permit null.
→ These are unbound queues.
→ It is not thread-safe.
→ The queue retrieval operations poll, remove, peek.
→ It provides O(log(n)) time for add and poll methods.

```
import java.util.*;
class PriorityQueueDemo {
public static void main(String args[])
{
PriorityQueue<Integer> pq = new PriorityQueue
                            <Integer>();
pq.add(10);
pq.add(20);
pq.add(15);
S.O.P (pq.peek());
S.O.P( pq.poll());
S.O.P( pq.peek());
}
}
```

```
SOP(pq);
pq.remove();
```

Eg:- import java.util.*;
public class PriorityQueueDemo{
public static void main(String args[])
{
PriorityQueue<String> pq=new PriorityQueue<>();
pq.add("Geeks");
pq.add("java");
pq.add("Astd");
Iterator itr=pq.iterator();
while(itr.hasNext())
{
S.op(itr.next()+"\n");
}
}
}

Array Deque:-

It provides a way to apply resizable array in addition to the implementation of the Deque interface. It is also known as Array Double ended Queue or Array Deck.

It allows users to add or remove elements from both sides of the queue.

→ It have no capacity restrictions.
→ They are not thread safe.
→ Does not support concurrent access by multiple threads.

→ Null elements are prohibited in the array deque.
→ It is faster than stack & linked list.

Array Deque can implements two interfaces
(i) Queue Interface
(ii) Deque Interface.

```
import java.util.*;
public class ArrayDequeDemo {
public static void main(String args[]) {
Deque<String> deque = new ArrayDeque<String>();
deque.add("Ravi");
deque.add("Vijay");
deque.add("Ajay");
for (String str : deque)
  {
  s.o.p(str);
  }
}
}
```

Hash Table :-

It can implements a hash table, which maps key to values. It inherits Dictionary class and implements the Map interface.

→ It is an array of list. Each list is known as a bucket. It contains values based on the key.
→ Hash table class contains unique elements.
→ Doesn't allow null key or null value.
→ It is synchronized
→ Default capacity of hashtable is 11, load factor is 0.75.

Eg:- import java.util.*;
class HashTableDemo {
public static void main (String args[]) {
Hashtable<Integer, String> hm = new Hashtable
<Integer, String>();
hm.put(100, "Amit");
hm.put(102, "Ravi");
hm.put(101, "Vijay");
hm.put(103, "Rahul");
for (Map.Entry m: hm.entrySet()) {
S.o.p(m.getKey() + " " + m.getValue());
}
}

Properties Class :-

Properties contains key and value pair both as a string.
The java.util.Properties class is the subclass of Hashtable.
It can be used to get property value based on the property key.
It provides methods to get data from the properties file and store data into the properties file.
It can be used to get the properties of a system.

Ex:-
```
import java.util.*;
import java.io.*;
public class Test {
public static void main(String args[])
throws Exception {
FileReader fr=new FileReader("db.properties");
Properties p=new Properties();
p.load(fr);
S.op(p.getProperty("user"));
S.op(p.getProperty("password"));
}
}
```

Java Stack :-

The stack is a linear data structure that is used to store the collection of objects.

It is based on LIFO.

Java collection framework provides many interfaces and classes to store the collection of objects.

Stack class provides different operations as push, pop, search etc.

The stack data structure has the 2 most important operations that are push and pop.

The push operation inserts an element into the stack and pop operation removes an element from the top of the stack.

Empty stack:- If the stack has no element is known as empty stack. When the stack is empty the value of the top is -1.

→ It implements the interfaces List, Collection, Iterable, Cloneable, Serializable.

Ex:-
```
import java.util.Iterator;
import java.util.Stack;
public class StackIterationDemo{
public static void main(String args[]) {
Stack stk = new Stack();
stk.push("Audi");
stk.push("Ferrari");
stk.push("BMW");
Iterator itr = stk.iterator();
while(itr.hasNext())
{
Object values = itr.next();
SOP(values);
}
}
}
```

Java Vector:-

Vector is like the dynamic array, which can grow or shrink its size.

We can store n-number of elements in it as there is no size limit.

It is a part of java collection framework.
→ It implements the list interface.
→ Vector is synchronized.
→ It contains many legacy methods.
→ Vector class is thread-safe.

Eg:-
```
import java.util.*;
public class VectorDemo
public static void main (String args[]){
Vector<String> vc=new Vector<String>();
vc.add("Tiger");
vc.add("Lion");
vc.add("Dog");
SOP("Elements are"+vc);
}
}
```

Java Bitset:-

Bitset class implements a vector of bits.
The Bitset grows automatically as more bits are needed.
Bitset class comes under java.util. package.
Each component of bitset contains at least one Boolean value.
The and() method is used to perform logical AND operation.

The andNot() method is used to clear the entire bit in this BitSet.

Ex:-
```
import java.util.BitSet;
public class BitSetDemo{
public static void main(String args[]){
BitSet b1=new BitSet();
b1.set('a');
b1.set('b');
b1.set('c');
b1.set(4);
b1.set(5);
BitSet b2=new BitSet();
b2.set(19);
b1.and(b2);
SOP("BitSet 1 :"+b1);
}
}
```

Date:-

The java.util.Date represents the date and time in java. It provides constructors and methods to deal with date and time in java.

java.sql.Data represents the only data in java

Ex:-
```
import java.util.*;
import java.time.*;
public class DateDemo{
public static void main(String args[]){
LocalDateTime dtm = LocalDateTime.now();
```

```
sop("The date is:" + dtm.toLocal Date());
    }
}
```

Calendar :-

Java Calendar class is an abstract class that provides methods for converting date b/w a specific instant in time and a set of calender fields such as MONTH, YEAR, HOUR etc.

```
Eg:- import java.util.*;
public class CalendarDemo{
public static void main(String args[]){
Calendar c = Calendar.getInstance();
sop("At present calendars year:" + c.get(Calendar.YEAR));
sop("At " " Day:" + c.get(Calendar.DATE));
}
}
```

Random Number :-

Random numbers are the numbers that use a large set of numbers and selects a number using the mathematical algorithm.

Using the random() method, we can generate random number in Java.

Java math class has many methods for different mathematical operations.

Ex:- import java.lang.Math;
public class RandomDemo{
public static void main(String args[])
{
SOP("Random Number:" + Math.random());
}
}

Formatter:-

java.util.Formatter class provides support for layout justification and alignment, common formats for numeric, string and date/time etc.

→ Formatters are not necessarily safe for multithreaded access.

Ex:- import java.util.*;
class FormatterDemo{
public static void main(String args[]){
Formatter f=new Formatter();
f.format("%d", 11);
SOP(f);
}
}

O/P:- 11.1

Java Scanner :-

Scanner class in Java is found in the java.util package.

Java provides various way to read input from the keyboard, one of them is java.util.Scanner;

Ex:-

```
import java.util.*;
public class ScannerDemo{
public static void main (String args[]){
Scanner sc=new Scanner(System.in);
Sop ("Enter name:");
String name = sc.nextLine();
Sop ("Name is:" + name);
sc.close();
}}
```

Multithreading:-

Multithreaded programming in java is introduced inorder to speed up the process of execution by splitting a single program into a number of threads that are executed concurrently.
A thread can be created either by implementing thread class or Runnable Interface.

Thread:-

A Thread can be a set of executable instructions that are executed independently.
A program can be divided into multiple subprograms, and each subprogram is called a thread.
Every individual Thread is executed separately.

Synchronization:-

It is a process of enabling single thread to access shared resources.
In multithreaded programming, synchronization threads are essential.

notify():-

This method is used to resume the initial thread which is in sleep mode.

notifyall():-

This method is used to resume all the threads that are in sleep mode. These threads can be

executed depending on its priority.

wait():-
This method is used to send the invoking thread into sleep mode.
The thread which is sent to sleep mode can be resumed with the help of notify() & notifyall().

String Tokenizer:-
It separates string / text into tokens delimited by \s, \t, \n, \r and \f.
It takes a string as input and parses it into tokens. This method is known as parsing.

-StringTokenizer(String s)

Java Thread Model:-

In java, multithreading is preferred more than the single threaded system.

Single threaded system make use of a method named event loop along with polling.

In single threaded model, a single thread is responsible for blocking the execution of other threads, unless it completes its execution.
Hence, the resources are wasted.

Multithreading in java offers the following benefits

→ It removes the loop and polling mechanism.
→ A single thread can be stopped without

affecting the execution of other parts of the program. Java thread model can be defined into the following 3 parts.

(i) Thread Priorities
(ii) Synchronization
(iii) Messaging :

A program can be divided into a number of threads where in every individual thread can communicate with each other.
Java supports this messaging service with low-cost for the threads to communicate. It provides a set of methods that support inter-thread communication.

Creation of Thread :-

A thread can be created in two ways
(i) By implementing Runnable Interface
(ii) By extending Thread class.

By Implementing Runnable Interface :-

The simple and easiest way for creating a thread is to create a class which implements the "Runnable" interface.

Syntax : public void run().

```
Eg: public class RunnableDemo implements Runnable
{
    public void run() {
        S.op ("Thread has ended");
    }
    public static void main (String args[]) {
        RunnableDemo rd = new RunnableDemo();
        Thread t1 = new Thread(rd);
        t1.start();
    } SOP ("Hi world");
}
```

Extending Thread class:-

This is another way for creating a thread.

```
Eg: class ThreadDemo extends Thread
{
    public void run()
    {
        for (int i=1 ; i<5; i++)
        {
            sop (this.getName() + "Running");
            try {
                Thread.sleep (1000);
            }
            catch (Exception e)
            {
                sop("Caught exception:" + e);
            }
        }
    }
}
```

```
public class ThreadExample
{
  public static void main(String args[])
  {
    ThreadDemo t1 = new ThreadDemo();
    ThreadDemo t2 = new ThreadDemo();
    t1.start();
    t2.start();
  }
}
```

Thread Priorities:-

Every thread is assigned a priority.
The thread with highest priority, that thread contains less access time.
The priority does not depends on the execution speed of thread.
If a child thread is created, then the priority of it is similar to its parent thread.

Syntax:- final void setPriority(int priority)
setPriority is used to set the priority for the thread.
The various priorities for the threads are follows

MIN_PRIORITY = 1
NORM_PRIORITY = 5
MAX_PRIORITY = 10

```
class Priority extends Thread
{
  public void run()
  {
    for(int i=1; i<10; i++)
    {
      String str = Thread.currentThread().
                   getName();
      sop(str+":"+i);
    }
  }
public static void main(String args[])
{
  Demo d1 = new Demo();
  Demo d2 = new Demo();
  Demo d3 = new Demo();
  d1.setName("First");
  d2.setName("Second");
  d3.setName("Third");
  d1.setPriority(5);
  d2.setPriority(10);
  d3.setPriority(1);
  d1.start();
  d2.start();
  d3.start();
  SOP("First thread:"+d1.getPriority());
  SOP("Second thread:"+d2.getPriority());
}
}
```

The thread with highest porority will be executed first.

Synchronizing Threads:-

Thread synchronization is the concurrent execution of two (or more threads.
Threads should be synchronized to avoid critical resources. Otherwise, conflicts may arise when parallel running threads attempt to modify a common variable at the same time.

Ex:-
```
Import java.io.*;
class One extends Thread
{
public void run()
{
for (int i=0;i<100;i++)
{ try {
        Thread.sleep(1000);
      }
    Catch (InterruptedException e)
      {
        Sop(e);
      }
    Sop("Good Morning");
}
}
}
```

By for thread two and Thread Three

2000
Hello

3000
Welcome

```
class ThreadEx
{
public static void main(String args[])
{
  One t1=new One();
  Two t2=new Two();
 Three tt=new Three();
 Thread t3=new Thread(tt);
  t1.setName("One");
  t2.setName("Two");
  t3.setName("Three");
   Sop(t1); Sop(t2); Sop(t3);
 Thread t=Thread.CurrentThread();
  Spp(t);
 t1.start();
  t2.start();
  t3.start();
}
}
```

Inter Thread Communication:-

In inter thread communication, multiple threads can communicate with each other by exchanging messages.

Any two threads can communicate before switching to other threads.

Interthread Communication can be supported by using following methods,

notify() :- This method is used to resume the init of thread which is in sleep mode.

notifyall() :- This method is used to resume all the threads that are in sleep mode. These threads can be executed depending on its priority.

wait() :- This method is used to send the invoking thread into sleep mode.

The thread which is sent to sleep mode can be resumed with the help of notify() (or) notifyall() method.

In addition to this, user can set time to the thread by declaring the timer as an argument for wait() method.

After the completion of timer, the thread can be resumed automatically.

Ex:- class Interthread
{
float r = 0.0;
synchronized void result()
{ sop ("The method called to display circle area");
if(r == 0.0)
{ sop ("Enter radius");
try {
wait(); }

```
        catch(Exception e)
        {
            sop(e);
        }
    }
    Sop("The area is:" 3.14 * radius * radius);
}
synchronized void etvalup(float x)
{
    Sop("Enter radius");
        r=x;
        sop("Radius received");
        notify();
    }
}
class MultiThread
{
    public static void main(String args[])
    {
        final InterThread it=new InterThread();
        new Thread()
        {
            public void run()
            {
                it.result();
            }
        }.start();
```

```
newThread()
  { public void run()
      { it.setValue(10.5);
      }
    }.start();
  }
}
```