Multi - threading

Thread: Thread is a light weight process executes parallelly;

⇒ Whenever multiple threads are executing simultaneously then it is called multi-threading.

⇒ Multi threading is useful for gaming applications, for animations,...etc.

⇒ In java threads can be created with two approaches

1) By extending the Thread class
2) By implementing the Runnable interface.

Creating the thread by extending Thread class:-

step-1) Create a undefined class by extending the Thread class.

```
Class Mythread extends Thread.
{
}
```

step-2) The predefined Thread class has run() with zero statements. that should be "override" in the current class

```
Class Mythread extends Thread
{
    Public void run()
    {
        Statements;
    }
}
```

step-3) In the main() create an object of Mythread class.

```
Mythread t = new Mythread();
```

Step-4) Call the start() with respect to created object.

10/2/22

Java program to create the user thread along with main thread.

```
Class Mythread extends Thread.
{
    Public void run()
    {
        for (int k=125; k<=136; k++)
        {
            System.out.println("user thread " + k);
        }
    }
}
Class Demothread
{
    Public static void main(String[] args)
    {
        for (int i = 1; i<=10; i++)
        {
            System.out.println("main thread"),
        }
        Mythread ob = new Mythread();
        ob.start();          // multi-threading
    }
}
```

In the above program the start() is used for multi threads in which the output is unpredictable

but run() method executes in an order which is called uniprogramming. (not multi threading).

start()
1. allocates memory for Thread
2. copy the run()
3. execute the Thread.

sleep(): This method belongs to Thread class which makes the current thread execution in waiting mode (or) sleep mode with the specified number of micro seconds.

Syntax:

```
threadobj.sleep(int);
```

```
class csethread extends Thread
{
    public void run() throw InterruptedException
    {
        for(int i=10; i>=1; i--)
        {
            System.out.println("my thread" + i);
            Thread.sleep(1000);
        }
    }
}
class Sleepdemo
{
    Public static void main(String args[]) throws
                                    InterruptedException.
```

```
{
    for(int j=5; j<=25 ; j+=5)
    {
        System.out.println("main thread"+j);
    }
    Csethread t=new Csethread();
    t.start();
} // main close
} // class close.
```

output

```
main thread 5
main thread 10
main thread 15
main thread 20
main thread 25
my thread 10
my thread 9
my thread 8
my thread 7
my thread 6
my thread 5
my thread 4
my thread 3
my thread 2
my thread 1
```

Runnable interface:-

We can create the user threads by implementing Runnable interface. This interface contains one and only abstract method, run()

Step-1:- Create a class which by implements Runnable.

```
class Myth implements Runnable.
{

}
```

Step-2:- Concrete the run()

```
class Myth implements Runnable
{
    Public void run()
    {
        statements;
    }
}
```

run() it won't create a new thread & it will be in same stack as main

step-3:- Inside the main() create an object for Myth class

```
Myth mt=new Myth();
```

11/6/14
//step-4:- By using the implemented class object // call the run()

step-4:- create an object of predefined Thread class & pass implemented object (step-3) as constructor argument.

```
Thread t = new Thread(mt);
```

step-5:- By using Thread class object call the start().

```
t.start();
```

Creating the threads by implementing Runnable interface.

```
class Myth1 implements Runnable
{
    public void run()
    {
        for(int i=100; i>=50; i--)
        {
            System.out.println("Apple Thread"+i);
        }
    }
}
class Threaddemo1nf
{
    public static void main(String args[])
```

```
        }
            Mythi mt = new Mythi();
            Thread t = new Thread(mt);
            {t.start();
              for (int m = 10; m<= 25; m++)
              {
                System.out.println(" Main thredd "+m);
              }
            }
        }
    }

class Myname
{
    void display()
    {
        String s = "Harshitha";
        System.out.println(s);
    }
}

class Myth
{
    Public void run()
    {
        Myname mn = new Myname();
        mn.display();
    }
}
```

⇒ main class

Create a class with a method to accept a string & print it for 10 times. call this method inside the run method. and also print your mothername for 15 times inside the main method. (Extending thread & implementing runnable)

```
// by extending Thread class
class Myname
{
    void display()
    {
        String s = "Harshitha";
        for (int i = 1; i <= 10; i++)
        {
            System.out.println(i + "." + s);
        }
    }
}
class Myth extends Thread
{
    public void run()
    {
        Myname mn = new Myname();
        mn.display();
    }
}
class ThreadStr
{
    public static void main(String[] args)
```

```
{
    for(int j=1; j<=15; j++)
    {
        System.out.println(j + "." + My mother name is Neelima");
    }
    Myth mt = new Myth();
    mt.start();
}
}
```

Thread class methods:-

The predefined Thread class has several methods to toget or toset data to the threads.

1. getName():- This method returns string data type with the names of user defined thread as

```
Thread-0, Thread-1, ------
```

To get the mainly thread information we use

```
Thread.currentthread.getName();
```

2. SetName():- By using this method we can set our own names to the userdefined threads.

```
Void setName(String s)
{
    ...
}
```

Thread priorities:-

In java CPU scheduler desides which thread will be executed at first based on priority.

3. getPriority():- Return an integer in a range from 1 to 10. When the user is not giving priority for thread then JVM sets 5 as normal priority

4. setPriority():- This method takes integer as args in between 1 to 10.

1 for MIN-PRIORITY

10 for MAX-PRIORITY.

Ex:- t.setPriority(MAX-PRIORITY);

(or)

t.setpriority(10);

Java program to demonstrate thread methods.

```
//by implements Runnable interface
Class Thread1 implements Runnable
{
   public void run()
   {
      System.out.println("write assignment");
   }
}
Class Thread2 implements Runnable
{
      Public void run()
```

```
        {
            System.out.println(" submit on monday");
        }
    }
class Pr
{
    Public static void main (String args[])
    {
        Thread1 t1 = new Thread1();
        Thread1 t2 = new Thread2();
        Thread td1 = new Thread(t1);
        Thread td2 = new Thread(t2);
        td1.setName("Harshitha");
        td2.setName("Java");
        td1.setPriority(10);
        td2.setPriority(2);
        System.out.println("main-thread");
        td1.start();
        td2.start();
        System.out.println(td1.getName()+" "
                           + td1.getPriority());
        System.out.println(td2.getName()+" "
                           + td2.getPriority());
    }
}
```

```
// by extends Thread class
class Myth1 extends Thread
{
    public void run()
    {
        System.out.println("write Assignment");
    }
}
class Myth2 extends Thread
{
    public void run()
    {
        System.out.println("Submit on monday");
    }
}
class Pr
{
    public static void main(String args[])
    {
        Myth1 mt1=new Myth1();
        Myth2 mt2=new Myth2();
        mt1.setName("Harshitha");
        mt2.setName("Palacharla");
        mt1.setPriority(10);
        mt2.setPriority(2);
        System.out.println("main-Thread");
        mt1.start();
        mt2.start();
        System.out.println(mt1.getName()+" "
                +mt1.getPriority());
```

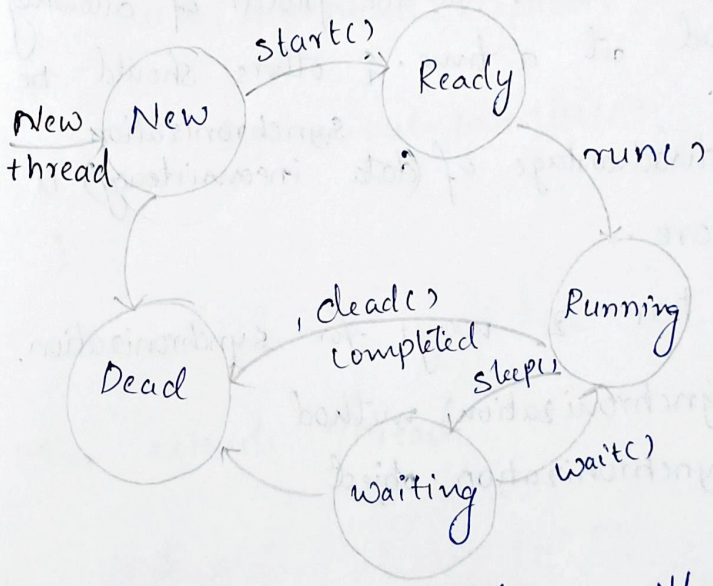
```
            System.out.println(mt2.getName()+" "+
                               mt2.getPriority());
        }
}

//accepting string (implements Runnable interface)

class Myframe
{
    void display()
    {
        String s="Harshitha";
        for(int i=1;i<=10;i++)
        {
            System.out.println(i+"."+s);
        }
    }
}
class Myth2 implements Runnable
{
    public void run()
    {
        Myname mn=new Myname();
        mn.display();
    }
}
class Runstr
{
    public static void main(String args())
    {
        for(int j=1;j<=15;j++)
```

// Life cycle of a thread.

```
{
    System.out.println(j+"."+"My mother name is Deelima");
}
Myth1 mt1 = new Myth1();
Thread t1 = new MythThread(mt1);
t1.start();
}
}
```

14/12/22

Life cycle of a thread:



When you create the thread, it will be in new state.

⇒ By using the start(), the thread state change from new to ready with run() code.

⇒ we know that processor executes multiple threads simultaneously, when the run() code is executed then

Thread state is running

→ if the thread needs any other resources it must be wait until it gets the information. The programmer intenally can sent the thread in sleep mode. Both cases thread will be in waiting state.

⇒ After processing run(). code thread will be in dead state.

*** Synchronization:-

When multiple threads are executing simultaneously then data inconsistency problem arises.

Synchronization is the process of allowing only one method at a time. & others should be in waiting state.

The disadvantage of Synchronization (~~data~~ inconsistency) is waiting time is more.

Java has 2 ways for synchronization.

1. Synchronization method
2. Synchronization object.

```
// Synchronization method
Class Data
{
    synchronized void printdata (int x)
    {
        for (int i= 1 ; i <= 5; i++)
        {
            System.out.println (x*i);
            try
            {
                Thread.sleep(400);
            }
            catch (InterruptedException ie)
            {
                System.out.println(e);
            }
        }
    }
}
class My1 extends Thread
{
    Data d1;
    My1 (Data dob)
    {
        d1 = dob;
    }
    public void run( )
    {
        d1.printdata(5);
```

```
        }
    }
class My2 extends Thread
{
    Data d2;
    My2(Data dob)
    {
        d2 = dob;
    }
    Public void run()
    {
        d2.printdata(16);
    }
}
Class Pr
{
    Public static void main(String args[])
    {
        Data dob = new Data();
        My1 t1 = new My1(dob);
        My2 t2 = new My2(dob);
        t1.start();
        t2.start();
    }
}
```

Synchronization object

like synchronized method we can create synchronized block (or) Synchronized object.

Now in multithreading only one object can access the resource at a time.

Synchronized is the modifier applied to method (or) block but not class (or) variables.

```
// Synchronization object
class Data
{
    void printdata (int n)
    {
        Synchronized (this)
        {
            for(int i=1; i<=5; i++)
            {
                System.out.println(n*i);
                try
                {
                    Thread.sleep(400);
                }
                Catch (Exception e)
                {
                    System.out.println(e);
                }
            }
        }
    }
}
```

```
Class My1 extends Thread
{
    Data d1;
    My1(Data dob)
    {
        d1 = dob;
    }
    Public void run()
    {
        d1.printdata(5);
    }
}
Class My2 extends Thread
{
    Data d2
    My2(Data dob)
    {
        d2 = dob;
    }
    Public void run()
    {
        d2.printdata(7);
    }
}
Class R
{
    Public static void main(String[] args)
```

```
{
    Data dob = new Data();
    My1 t1 = new Data(dob);
    My2 t2 = new Data(dob);
    t1.start();
    t2.start();
  }
}
```

15/3/22

Inter thread communication:-

Threads should be communicated each other to process the code successfully.

⇒ Java provides the 3 methods wait(), notify() and notifyall() in Object class

⇒ Any class may utilize this 3 methods. ~~since~~

Since Every class is child class to Object class.

⇒ producer & consumer problem must need inter thread communication to access the produced products

⇒ In banking applications deposit & withdraw threads needed inter thread communication.

```
Class Bank
{
    int amount = 25,000;
    Synchronized void withdraw(int amt)
    {
        System.out.println("withdraw procell starts");
        if (amount < amt)
        {
            try
            {
                wait();
            }
            catch (Exception e)
            {
                System.out.println(e)
            }
        }
        System.out.println(" The balance is ".+ amount);
        amount = amount - amt;
    }   System.out.println(" After withdraw the balance is "+
                                                    amount);
    Synchronized void deposit(int dmt)
    {
        System.out.println("Deposit procell starts");
        /*
        if (dmt == 0)
        {
            try
            {
                wait();
            }
            catch(Exception e)
            {
            } System.out.println(e);
```

```
notify() */
amount = amount + dmt;  → System.out.println("the balance is"+ amount);
notify()                → System.out.println("After deposit
                            the balance is "+ amount);
}

}
class Mythread1 extends Thread
{
    Bank b;
    Mythread1(Bank ob)
    {
        b= ob;
    }
    Public void run()
    {
        b.withdraw(30000);
        //b.deposit(10000);
    }
}

class Mythread2 extends
                Thread
{
    Bank b;
    Mythread2(Bank ob)
    {
        b=ob;
    }
    Public void run()
    {
        b.deposit(10000);
    }
}

class Prio1
{
    Public void main(String args[])
    {
        Bank ob = new Bank();
        Mythread1 t1 = new Mythread1(ob);
        t1.start();
        Mythread2 t2= new Mythread2(ob);
        t2.start();
    }
}
```

withdraw process starts
Deposit porcess starts
The balance is 25000
After deposit the balance is 35000
The balance is 35000
After withdraw the balance is 5000

16/12

Collections framework:-

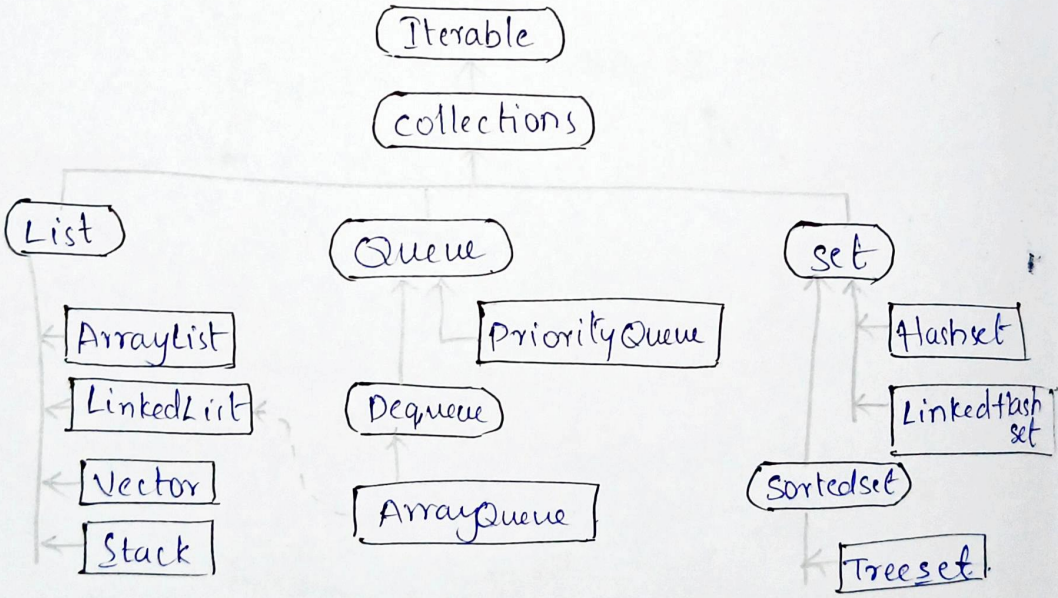
- Java collections are introduced from jdk 1.2 version but became popular from java 1.5 the version with Autoboxing & generics features.

⇒ Collections ∈ to java.util package

⇒ Collection can be defined as "group of objects as a single entity & perform operations".

⇒ In the java.util package collections framework has set of interfaces & classes.

The following diagram shows the classes with Rectangle box & interfaces with Rounded Rectangle



Characteristics of data structure in java:-

1. Homogeneous

1. All data structures/collections in java allows Heterogeneous objects.

2. In ArrayList, LinkedList, Vector, Stack allows duplicate objects. In set related class allows only unique objects

3. The the order you created the data structure will remain same till end, then it is called Insertion order is preserved.
In List related classes insertion order is preserved.

4. We can insert "null" value at any position.

5. mutable v/s immutable.

ArrayList:-

→ It is a collection of heterogeneous objects
→ It allows duplicate objects
→ order in preserved
→ we can assign a "null" value.
⇒ It is the concrete class in the java.util package. When you create an object, it is an entity for collection of different category of objects.
⇒ Square brackets ([ ]) are used to represent the ArrayList.
⇒ ArrayList methods, In ArrayList class has several concrete methods to operate on group of objects. By using ArrayList class object we can call those methods.

isEmpty():- This method returns a boolean value "true" when ArrayList contains no objects

Syntax:-

```
boolean isEmpty()
{
    return (boolean);
}
```

add():- This method is used to insert one object into the array list.
This method is a overloaded method with different args

```
void add(obj)
{

}
```

This will add the specified object at end of the Arraylist

```
void add (Position, obj)
{

}
```

This method also add object at cleired position.

⇒ To display the array list objects we can use the object name in the println method.

Java program to perform inserting operation & displaying the Arraylist.

```
import java.util.ArrayList ;
class Proo7
{
    Public Static void main (String[] args)
    {
        ArrayList al = new ArrayList ();
        System.out.println ("Array list is empty"
                            + al.isEmpty());
        al.add (200);      // [200]
        al.add (300);      // [200,30]
        al.add (1, 100);   // [200, 100, 30]
        System.out.println (al);   // [200, 100, 30]
    }
}
```

remove() :- By using this method we can remove the objects through indexed(d) directly.

remove(index) :- This method removes the specified indexed object from the ArrayList. When the index is not available then it raises IllArrayIndexOutOfBoundsException.

get() :- By using this method we can gets the object value or object information through it index.

```
object get(int index)
{

}
```

Ex:-

```
a1 = [11, "cse", 7.9, 11]
System.out.println(al.get(2));
```

Set() :- By using this method we can add a new object at desired position & also we can replace the object content with new content.

Syntax:-

Object.set(pos, object)

Ex :- al.set(2, "Harshitha");

size() : It red wr returns an integer value which is equal to the no.of objects in the ArrayList collections

Syntax:-

```
int variable = object.size();
```

```
import java.util.Arraylist;
class Pr1234
{
    Public static void main(String[] args)
    {
        Arraylist al = new Arraylist();
        System.out.println("the total objects are"
                                    + al.size());
        al.add(10);
        al.add("Ai");
        System.out.println(al.size()); //2
        System.out.println(al.get(0)); //10
        System.out.println(al); // [10, "Ai"]
        al.set(1, "cse");
        System.out.println(al); // [10, "cse"]
    }
}
```

//Vector:-

Write a java program by using ArrayList with integers & display the last but one element

```
import java.util.ArrayList;
class Pr124
{
    public static void main(String[] args)
    {
        ArrayList al = new ArrayList();
        System.out.println("Entering 5 elements");
        al.add(10);
        al.add(10);
        al.add(30);
        al.add(40);
        al.add(50);
        System.out.println(al.get(al.size()-2));
    }
}
```

Write a java program to convert the ArrayList into an Array.

We know that collections are two types.

As normal & Generic

In Generic only one class objects are allowed & not permittable other class objects.

Syntax:-

```
Collection<class_name> obj = new Collection<class_name>();
```

```
Collection<class_name> obj = new Collection<class_name>();
```

Ex:- ArrayList<Integer> al = new ArrayList<Integer>();

```
// Arraylist to Array
import java.util.*;
class Aal
{
    Public static void main(String[] args)
    {
        ArrayList<Integer> al = new ArrayList<Integer>(6);
        al.add(100);
        al.add(20);
        al.add(35);
        Collections.sort(al); //[20,35,100]
        Integer ia[]=new Integer[al.size()];
        ia = al.toArray(ia);
        //System.out.println(ia);
        for (int i=0; i<ia.length ; i++)
        {
            System.out.println(ia[i]);
        }
    }
}
```

LinkedList :-

- LinkedList is the linear Data structure which can't store objects in a continuous representation.
- Internally LinkedList contains node with data & address
- In java LinkedList are generics which can holds same class objects.
- LinkedList class extends AbstractSequentialList and implements List, Queue, Dequeue.
- This class has two constructors.
  - LinkedList() :- creates a empty linked list,
  - LinkedList(Collections obj) :- creates a LinkedList with specified objects

Syntax :-

```
LinkedList<class_name> objname = new LinkedList
                                 <class_name>();
```

Eg. `LinkedList<String> ll = new LinkedList<String>();`

Methods :-

- add(), get(), set(), size() is also available in LinkedList class.

Java program create an empty LinkedList & add 4 elements of Double objects. Access the 3rd position object & print it. modify the 1st object data & print the whole LinkedList.

```
import java.util.*;
class Pr125
{
    Public static void main(String[] args)
    {
        // LinkedList <class_name>
        // Linkedlist <Pr125> ob = new LinkedList <Pr125>();
        LinkedList <Double> ob = new LinkedList<Double>();
        ob.add(10.85);
        ob.add(11.25);
        ob.add(9.74);
        ob.add(4.57);
        System.out.println(ob.get(2));      // 9.74
        System.out.println(ob.set(0,2.9));  // 2.9
        System.out.println(ob);             // [2.9, 11.25, 9.74, 4.57]
    }
}
```

⇒ LinkedList contains so many methods among them the following access objects at first and last positions.

1) addFirst() or OfferFirst()

This methods are used to add the new object at 0th position.

2) addLast() (or) OfferLast():-
⇒ used to add the new object at last position

3) getFirst() (or) PeekFirst():-
⇒ These methods are useful to obtain specific first position object data

4) getLast() (or) peekLast():-
⇒ used to obtain last position object data.

5) removeFirst() (or) pollFirst():-
⇒ these methods are used to delete the 0th indexed object.

6) removeLast() (or) pollLast():-
⇒ these methods are used to delete the last indexed object.

Write a java program to create a LinkedList with 5 elements objects of Integer class. print once by using. get() and peek() of 2nd object. Remove the First and Last two indexed elements.

poll
peek
offer

```
import java.util.*;
class Pr126
{
    Public static void main(String[] args)
    {
        LinkedList<Integer> ll = new LinkedList<Integer>();
        ll.offerFirst(10);
        ll.offer(20);
        ll.offer(30);
        ll.offer(40);
        ll.offerLast(50);
        System.out.println("The objects are");
```

```
for(int i=0; i<5; i++)
{
    //System.out.println(peek
    System.out.println(ll.peek(i));
}
System.out.println(ll.get(2));
ll.pollFirst();
ll.pollLast();
System.out.println(ll);
}
}
```

Create a Arraylist with Employee class objects with Generic data. Add 5 objects & print 2nd index. employee information.

```
import java.util.*;
class Emp
{
    int eno;
    String ename;
    double salary;
    Emp(int enum, String name, double sal)
    {
        eno=enum;
        ename=name;
        salary=sal;
    }
}
```

```
class Pr127
{
    Public static void main(String[] args)
    {
        Emp e1, e2;
        ArrayList<Emp> al = new ArrayList<Emp>();
        Emp e3 = new Emp(123, "Harshitha", 100000);
        Emp e4 = new Emp(124, "Neelima", 150000);
        Emp e5 = new Emp(125, "krishna", 15000);
        al.add(e1);
        al.add(e2);
        al.add(e3);
        al.add(e4);
        al.add(e5);
        System.out.println(al.get(e3));
```

Note:- Java allows user defined classes as collection generics

Hash set:-

Another data structure in java where

→ insertion order is not preserved
→ Duplicates are not allowed
→ null object is allowed
→

Hash function performs some operation the result is added to the Data Structure. This is done internally.

Hashset is the Genericclass with following constructors

Hashset( ) → creates empty set.

Hashset (int cap) → intial capacity set is created

methods:-

add( ) → adds the object at last
remove(obj) → removes/deletes the specified object.
size( ) → returns the no of objects
isEmpty( ) → if set is empty then it gives true, otherwise false
contains( ) → This method returns true if the specified object is available otherwise false

```
// using Hashset (order is not preserved)
import java.util.*;
class Hashsetdemo
{
    Public static void main (String[] args)
    {
        Hashset<Integer> hs = new Hashset<Integer>();
        System.out.println ("Is empty :- " + hs.isEmpty());
        hs.add(10);
        hs.add(100);
        hs.add(27);
        hs.add(117);
        //hs.add(117);
        hs.add(230);
        System.out.println ("The objects size is " + hs.size());
        System.out.println ("The objects are " + hs);
        hs.remove(100);
        System.out.println ("does 27 is available (true/false)" +
                            hs.contains(27));
        System.out.println ("Resultant objects are " + hs);
    }
```

output:-

Is empty:- true
The objects size in 5
The objects are [100, 117, 70, 10, 27]
does 27 in available (true/false): true
Resultant objects are [117, 70, 10, 27]

```
//using LinkedHashSet (order is preserved)
import java.util.
class LHaset
{
    public static void main(String args[])
    {
        LinkedHashset<Integer> lhs = new LinkedHashset
                                     <Integer>();
        System.out.println(" Is empty:- " + lhs.isEmpty
                                            ());
        lhs.add(10);
        lhs.add(100);
        lhs.add(27);
        lhs.add(117);
        lhs.add(70);
        System.out.println(" The objects size in "
                                + lhs.size());
        System.out.println(" The objects are " + lhs);
        lhs.remove(100);
        System.out.println(" does 27 in available:-"
                            (true/false)" + lhs.contains(27));
        System.out.println(" Resultant objects are "+hs);
    }
}
```

Output:-
Is empty: true
The objects size is 5
The objects are [10, 100, 27, 117, 70]

does 27 is available (true/false) :- true
Resultant objects are [10, 27, 117, 70]

// using Treeset (asending order)

```
import java.util.*;
Class Trset
{
    Public static void main(String args())
    {
        Treeset<Integer> ts=new Treeset<Integer>();
        System.out.println("Is empty "+ts.isEmpty());
        ts.add(10);
        ts.add(100);
        ts.add(27);
        ts.add(117);
        ts.add(70);
        System.out.println("The objects size is "+
                                    ts.size());
        System.out.println("The objects are "+ts);
        ts.remove(100);
        System.out.println("does 27 is available
                        (true/false) :- "+ts.contains(27));
        System.out.println("The Resultant objects are "
                                    +ts);
    }
}
```

Output:-
Is empty :- true
The objects size is 5
The objects are [10, 27, 70, 100, 117]

doc 27 is available (true\false) :- true
Resultant objects are. [10, 27, 70, 117]

Treeset:-

The Treeset class contains the following methods

SubSet() :- It forms another set with all the objects in the given range

//obj.
Let us Treeset be ["A", "B", "C", "D", "E", "F", "G"]

Subset:-

Syntax:

obj.subset("B", "G") //[B, C, D, E, F, G]

obj.subset("B", false, "G", false) [C, D, E, F]

headset() :- By using this method we can form another set from staring object to given range.

obj.headset("c"); //["A", "B", "C"]

obj.headset("C", false); //["A", "B"]

tailset() :- By using this method we can form set with the range of objects from specified object to last object.

obj.tailset("C"); // ["C", "D", "E", "F", "G"]

obj.tailset("c", false); // ["D", "E", "F", "G"]

Priority Queue :-

→ Another Data Structure where the objects are arranged based on priority internally.

→ It does not allows null values, duplicates

→ The insertion order is not preserved.

→ The methods are not synchronized.

→ The constructors are of two types

PriorityQueue()

PriorityQueue(int cap)

add(obj) :- adds a new object if it is not in the priority Queue

remove(obj) :- remove the specified object from the PriorityQueue

size() :→ It returns the size of the objects

clear() :- It clears all the objects in the priority Queue

peek() :- This method retrives the first object but not deleted

Poll() :- This method retrives the first object & also deleted.

Offer() :- adds a new object to the priority Queue.

```
import java.util.*;
Class Prq
{
    Public static void main(String[] args)
    {
        Priority Queue<Integer> p = new Priority Queue<Integer>();
                                    Empty constructor
```

```
System.out.println(P);      //[]
P.add(25);
P.offer(30);
P.add(125);
System.out.println("The PQ is " + P); //[125, 25, 30]
System.out.println("The total object is " + P.size());
System.out.println("The deleted object is" + P.poll());
}
}
```

ArrayDeque :-

→ This class has several constructors & methods to perform linear data structure "queue".

→ Constructors

ArrayDeque()
ArrayDeque(int cap)

→ methods :-

add(obj)
remove(obj)
size()
clear()
peek()
poll()
offer()
offerLast() :- add a new object at the last.
pollFirst() :- This method retrives the first object & also deleted.

→ This methods are not synchronized.

```
import java.util.*;
class Adeq
{
    Public static void main(String arg[])
    {
        //ArrayDeque ad
        ArrayDeque<Integer> ad = new ArrayDeque<Integer>();

        System.out.println(ad);
        ad.add(10);
        ad.add(25);
        ad.add(30);
        System.out.println(ad.size());
        System.out.println(ad.poll());
        System.out.println(ad);
    }
}
```

Hash table:-

⇒ Hash table deals with the objects with key & value.
⇒ It is formed with two generic classes. One for key object & another for value object.
⇒ This table arrange the elements with hashing code.
⇒ Insertion order is not preserved
⇒ Keys are unique & may be values can be duplicate

Constructor:-

Hashtable() → creates an empty Hashtable
Hashtable(int cap) → creates a Hashtable with specified objects.

methods:-

size() :- No. of objects available in Hashtable. returns integer.

put() :- this method inserts a new object with key & value.

```
Put (key, value);
```

hashCode() :- returns an integer that is the code applied to form the objects.

~~setKeys~~ keySet() :- returns a list with the all available keys.

values() :- returns the list with all the values.

contains(obj) :- If the specific object available in the Hashtable it returns true, otherwise false same as containsValue()

~~contains~~ (key)

containsKey() :- returns true if the specified key is available. otherwise false.

```
import java.util.*;
class Ht
{
    public static void main (String args[])
    {
        Hashtable < Integer, String > hs = new
                                Hashtable <Integer, String> ();
        hs.put (1, "A");
        hs.put (2, "B");
        hs.put (3, "C");
        hs.put (4, "D");
```

```
System.out.println(hs); // {16="AI", 5="CSE", 27="IOT"}
System.out.println("The values are "+this.value());
System.out.println(this.hashCode());
System.out.println("The keys are "+this.keySet());
System.out.println("size "+hs.size());
System.out.println("Removing "+this.remove());
System.out.println(hs.containsKey(5)); // true
}
}
```

Stack:-

⇒ It is similar to the Arraylist & LinkedList but the difference is Stack is Sychronized.

⇒ In stack collection insertion order is preserved with LIFO manner.

Constructor:-

Stack() :- this creates an empty stack with default size.

methods:-

The major operation on stack is push() and pop() for adding the object & deleting the object respectively.

push(obj) :- Inserts a new object into the stack. (adds at last).

pop() :- This method retrives & removes the last inserted object.

search( ) :- It takes object as a argument & returns the position of the object.

size( ) :- Returns the total no. of objects in the stack

```
import java.util.*;
class stackdemo
{
    public static void main (String args[])
    {
        Stack <Integer> st = new Stack <Integer>();
        System.out.println("The objects are ");
        st.push(10);
        st.push(20);
        st.push(30);
        st.push(40);
        st.push(50);
        System.out.println(st);
        st.pop();
        System.out.println(st);
        System.out.println(st.search(40));
    }
}
```

Note :- The search() returns the index of searched object.

Ex:- let s be [10, 35, 47, 9]

```
s.search(9); //1
s.search(47); //2
```

Vector :- Vector is the one of the collection where insertion order is preserved, duplicates are allowed & synchronized.

⇒ Here threads are safe due to the synchronization.

⇒ Vector class implements List interface,

Constructors

Vector( ) creates an empty vector

Vector(int cap) creates a vector with specified objects

methods:-

add( ) :- adds the object at the end.

isEmpty( ):- returns 'true' if it is empty otherwise false

clear(obj) :- clear the object specified;

size( ) :- returns the size of the objects

remove( ):- deletes the object

//elementAt(i

elementAt(index):- This method returns the object which is in the specified index. when index is not available throws ~~ArrayArray~~IndexOutOfBoundsExcep

firstElement( ):- returns the object which in at first position.

lastElement( ):- returns the object which in at last position of the vector

indexOf(obj):- returns the index of specified object.

```
import java.util.*;
class msr
{
    public static void main(String args[])
    {
        Vector<Integer> vt = new Vector<Integer>();
        for(int i=1; i<=5; i++)
        {
            vt.add(5*i);
        }
        System.out.println(vt);          // [5, 10, 15, 20, 25]
        System.out.println(vt.size());   // 5
        System.out.println(vt.firstElement()); // 5
        System.out.println(vt.lastElement());  // 25.
    }
}
```

Properties:- In a set where the key & values are string type objects. then this is called properties.

getProperty():- It takes string as a argument as a key & gives its value as a object.

```
Properties pt = new Properties()
```

StringTokenizer:- In java to break up the strings into substrings we use StringTokenizer class

⇒ When you break the string then the result will be substrings also called "Tokens".

~~Syntax~~:-

Syntax:-

```
//StringTokenizer new=
StringTokenizer obj= new StringTokenizer (String);
```

In this constructor it accepts the string & white space is the delimeter we can change the delimeter in two argument constructor StringTokenizer (String, delimeter);

```
simport
import java.util.*;
class Strtok
{
    public static void main (String args[])
    {
        System.out.println(" Enter the objects");
        Scanner sc= new Scanner( System.in);
        String s= sc.next();
        StringTokenizer st= new StringTokenizer(s);
        System.out.println (st.nextToken());
    }
}
```

methods:-

1) hasTokens() (or) hasElements():*/ hasMoreTokens() (or) hasMoreElements

By using this method we can know it has tokens or not. If tokens are available returns true otherwise false.

countTokens():- returns an integer value which in total no. of token.

nextToken():- This method returns an object as token. If no more token in available it returns null.

```
import java.util.*;
class stTok
 {
     public static void main(String args[])
     {
         StringTokenizer st=new StringTokenizer("Java is most powerful");
         System.out.println("No.of token" + st.countTokens());
         while(st.hasMoreTokens())
         {
             System.out.println(st.nextToken());
         }
     }
 }
```

}

BitSet :-

Each component of the BitSet has a Boolean Value

→ These component BitSet Value Can be changed with other Bitset objects

Constructor

BitSet() :- This Constructor Creates an empty BitSet by default the Set Values are false.

methods :-

.set(obj) :- This method inserts a new object into the BitSet.

and(obj) :- It compares the two BitSet objects where the Bitpositions are true then it combines both. The Common objects in both sets.

obj1.or(obj2) :- It denotes that the objects of Bitset 1. Internally it examines the Bitset Values which are true.

obj.equals(obj2) when both objects are same returns true otherwise false

```
import java.util.*;
class Bitsetdemo
{
  Public static void main (String args[])
  {
    BitSet b1 = new BitSet();
    BitSet b2 = new Bitset();
    b1.set(1);
    b1.set(2);
    b1.set(4);
    b1.set(6);
    System.out.println(b1); // {1,2,4,6}
    b2.set(20);
    b2.set(4);
    b2.set(6);
    System.out.println(b2); // {20,4,6}
    System.out.println(b1.equals(b2));
                                // false
  }
}
```

Date

Date class is the most useful class to work with real time applications.

→ It encapsulates data and time.

```
cl.add(Calendar.DATE(15));
System.Out.println(cl.getTime());
}
}
```

Random

In util package Random is the class used to access Random datatype Value

→ It is sub class to the object class the methods nextInt(), nextBoolean() next(), nextLine() are also accessed through Random class object.

Creation of an object

```
Random obj = new Random();
```

Java program to print two boolean Random Values two double Random Values and 20 Random Integer values in the range of 1 to 100.

Ans:

```
import java.util.*;
class RandomDemo
{ Public Static Void main(String[] args)
  {
    Random r = new Random();
    boolean b1 = r.nextBoolean();
    System.Out.println(b1);
    boolean b2 = r.nextBoolean();
```

```
System.out.println(b2);
double d1 = nextDouble();
System.out.println(d1);
double d2 = next Double();
System.out.println(d2);
for(int k=1; k<=20; k++)
    { int i = r.nextInt(100);
      System.out.println(i);
    }
  }
}
```

Formatter :-

As a Java developer we have to produce a clean and clear output to the user. for this the allingment should be properly arranged.

Java has Formatter class in util Package to set a normal allingment to print the results.

```
import java.util.*;
class Batham
{ Public static Void main(String[] args)
  { Formatter fr = new Formatter();
    for(int i= 1; i<=5; i++)
    { fr.format("%d", i);
```

```
        System.out.println(fr);
    }
    fr.close();
    }
}
```

Eg:-

```
import java.util.*
class Name
{
    Public Static Void main(String[] ar
    {
        Formatter f = new Formatter
        Scanner sc = new Scanner(System.
        for(int i=1; i<=6 ; i++)
        {
            char i = s.nextchar();
            f.format("%c", i);
            System.out.println(f);
        }
        f.close();
    }}
```

Scanner:-

Copy from unit-2