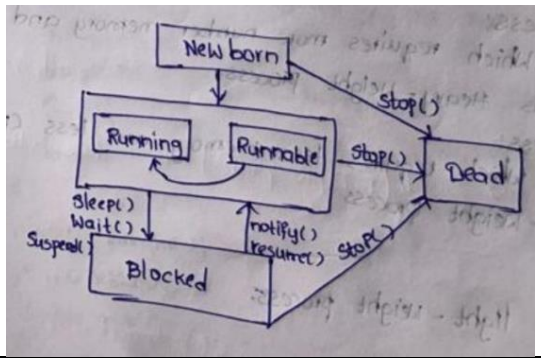
MULTITHREADING

Process: A program in execution Is called as a process.
Processes are of two types They are,
1.Heavy-weight process
2.Light-weight process
Heavy-weight process:
A process which requires more memory and more CPU time is called Heavy-weight process.
Light-weight process:
A process which requires less memory and less CPU time is called Light-weight process.
Thread:
A thread is a light-weight process.
(or)
A thread is the path followed when executing a program.
(or)
A thread is a single sequential flow of control in a program

Lifecycle of a thread:
A thread can be in one of the following states
1.Newborn state
2.Runnable state
3.Running state
4.Blocked state
1.Newborn state
When an object of the thread class is created then it is said to be a New born state.
2.Runnable state
When the thread is in a queue waiting for the cpu then it is said to be in Runnable state.
3.Running state
When the thread is being executed by the cpu then it is said to be Running state.
4. Blocked state
A thread enters into blocked state if one of the following methods are called
1.sleep ()
2.wait ()
3.suspend ()
5.Dead state
Once the code of the thread is executed then it enters into dead state from the blocked state if a thread wants to enter into running state then the following methods are called
1.notify ()
2.resume ()



Creation of threads:

A thread can be created using two methods

1.By extending the thread class

2.By implementing the Runnable interface

Assining priorities to threads:

Set priority () method is used to assign priority to the threads.

MIN-PRIORITY-1
NORM- PRIORITY-5 } constants
MAX- PRIORITY-10

Types of threads:

Threads are classified into two types

1.user threads

2.Daemon threads

User threads

The threads which are created by the user (or)programmer are called user threads.

Daemon threads

The threads which are executing in the background process are called Daemon threads.

// Java program for creating multiple threads using thread class.

```
Class A extends Thread
 {
  Public void run ()
   {
     For (int i=1; i<=5; i=i+1)
      {
         System.out.print (“\n\n\t i=” +i);
      }
   }
 }
   Class B extends Thread
    {
      Public void run ()
      {
        For (int j=1! j<=5; j=j+1)
         {
            System.out.print ("\n\n\t j= "+j);
         }
    }
 }
Class C extends Thread
{
 Public void run ()
 {
  For (int k=1; k=5; k=k+1)
  {
   System.out.print ("\n\n\t k "+k);
  }
 }
}
```

```
Class Demo1
{
  Public static void main (String args[])throwsException
  {
    Aa=new A ();
    Bb=new B ();
    Cc=new C ();
    a.start ();
    b.start ();
    c.start ();
  }
}
```

// Java program for creating multiple threads using runnable interface

```
class A implements Runnable
{
  public void run ()
  {
    for (int i=1; i<=s; i="+i);
    {
      System.out.print("\n\n\t i= "+i);
    }
  }
}
class B implements Runnable
{
  Public void run ()
  {
    For (int j=1; j<=5; j=j+1)
    {
      System.out.print('\n\n\t j="+j);
    }
  }
}
  Class C implements Runnable
{
  Public void run ()
  {
   For (int k=1; k<=5; k=k+1)
    {
      System.out.print("\n\n\t k="+k);
    }
  }
}
 Class Demo2
 {
   Public static void main (String args[])throwsException
    {
      A a= new A ();
      B b= new B ();
```

```
        C c= new C ();
        Thread t1=new thread (a);
        Thread t2=new thread (b);
        Thread t3=new thread (c);
        T1.start ();
        t2.start ();
        t3.start ();
      }
   }
```

//Java program for assigning priorities to threads

```
Class A extends Thread
{
 Public void run ()
  {
    For (int i=1; i<=s; i=i+1)
     {
     System.out.print(\n\n\t i= "+i);
     }
  }
}
 Class B extends Thread
  {
   Public void run ()
    {
      For (int j=1; j<=s; j=j+1);
       {
         System.out.print (\n\n\t j= "+j);
       }
    }
 }
   Class C extends thread
   {
     Public void run ()
      {
        For (int k=1; k<=s; k=k+1)
         {
           System.out.print("\n\n\t k= "+k);
         }
      }
   }
     Class Demo3
     {
       Public static void main (String args[])throwsException
       {
         A a= new A ();
         B b= new B ();
         C c= new C ();
          a.set priority (Thread.MAX_PRIORITY);
```

```
        b.set priority (Thread.NORM_PRIORITY);
        c.set priority (Thread.MIN_PRIORITY);
        a.start ();
        b.start ();
        c.start ();
      }
```

INTER THREAD COMMUNICATION:

The communication between threads is known as Inter thread communication.

SYNCHRONIZATION:

Synchronization refers to the co-operation and co-ordination among multiple processes in accessing a single resource.

Synchronisation can be achieved in java using three methods:

1.synchronized block
2.synchronized method
3.static block.

1. synchronized block:

```
synchronised
{

}
```

2. synchronized method:

```
public synchronised void run ()
{

}
```

3. static block:

```
static
{

}
```

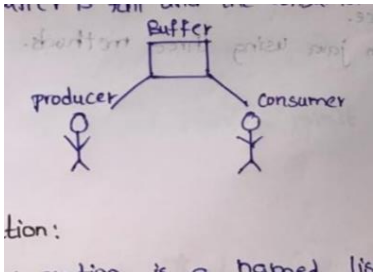
CLASSICAL PROBLEMS OF SYNCHRONISATION:

1.producer-consumer problem
2.Reader- writers problem
3.Dining philosopher's problem

PRODUCER -CONSUMER PROBLEM:

1.A producer is a process which produces items.
2.A consumer is a process which consumes items.
3.A buffer is a common area shared by both the producer and consumer.

The size of the buffer is limited to one item.
When the buffer is empty,the producer has to produce an item and place it in the buffer.
When the buffer is full,the consumer has to consume the item is placed in the buffer.
Both processes should access buffer without conflict and this is achieved through semaphore.
A semaphore is a binary variable.
A semaphore is associated with the buffer and if the value of the semaphore is zero then it indicates that the buffer is empty and producer has to act. If the value of the semaphore is one then it indicates that the buffer is full and consumer has to act

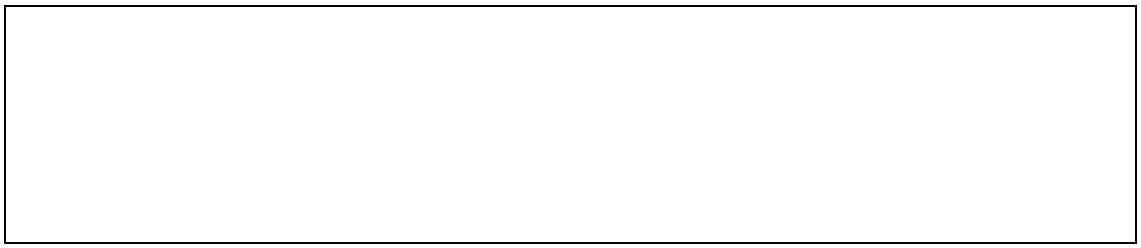


//Java program for executing multiple threads simultaneously

```
Class Sample 1 executive Thread
{
  Public void run ()
   {
     While(true)
     {
       System.out.print("\n\n\t GOOD MORNING");
        try
            {
             Sleep (1000);
            }
            Catch (Exception e1)
          {
             System.out.print(e1);
          }
     }
   }
}
  Class Sample 2 extends Thread
   {
     Public void run ()
     {
       While(true)
        {
         System.out.print("\n\n\t HELLO");
          try
           {
             Sleep (2000);
            }
```

```
                Catch (Exception e2)
                 {
                    System.out.print(e2);
                 }
            }
       }

}
 Class Sample 3 extends Thread
  {
  Public void run ()
    {
      While(true)
       {
          System.out.print("\n\n\t WELCOME");
           try
            {
              Sleep (3000);
            }
              Catch (Exception e3)
               {
                  System.out.print(e3);
               }
            }
       }
    }
      Class Sampledemo
       {
         Public static void main (String args[]) throws Exception
        {
           Sample s1= new sample1();
           Sample s2=new sample2();
           Sample s3=new sample3();
           s1.start ();
           s2.start ();
           s3.start  ();
         }
        }
```



UNIT - IV - PART - II

THE COLLECTIONS FRAMEWORK (java.util.)

The Collection interfaces

* Collection interface
* List interface
* Set interface
* SortedSet interface

The Collection classes

* ArrayList class
* LinkedList class
* HashSet class
* TreeSet class
* Vector class
* Stack class
* Hashtable class
* Properties class
* StringTokenizer class
* BitSet class
* Date class
* Calendar class
* Random class
* Formatter class
* Scanner class
* Priority Queue class

ArrayList

An ArrayList is a variable-length array of object references.

An ArrayList can increase or decrease in size.

LinkedList

A LinkedList stores elements using double linked DSL

It is nonSynchronized

TreeSet

It stores the elements using a tree.

HashSet

It stores the elements using hashing

Vector

A Vector acts like a dynamic array.

It is synchronized

Stack

A Stack is a datastructure which follows the principle of Last-In-first-out

Hashtable

A Hashtable stores key-value pairs in a hash table.

Properties

It is used to maintain lists of values in which the key is a string and the value is also a string.

StringTokenizer

It is used to parse the given string based on the given delimiter.

BitSet

A BitSet class creates a special type of array that holds bit values.

Date

The Date class encapsulates the current date and time.

Calendar

The Calendar class provides methods for retrieving

* Year
* Month
* day
* hour
* minute and
* second.

Random

The Random class is used to generate pseudo random numbers.

Scanner

The Scanner class is used to receive input from the keyboard.

Formatter

The formatter class is used for layout justification and alignment.

Priority Queue

A priority Queue is a queue in which all the elements are processed based on priority.

Array Deque

An ArrayDeque describes an implementation of a resizable array structure which implements the Deque interface.

ArrayList

```
// Java program to demonstrate ArrayList

import java.util.*;

class ArrayListDemo
 {
   public static void main(String args[])
    {
       ArrayList al = new ArrayList();

       al.add("KIRAN");
       al.add("AJAY");
       al.add("VIJAY");
       System.out.print("\n\nTHE CONTENTS OF ARRAYLIST
                                          ARE---");

       System.out.print(al);
    }
 }
```

LinkedList

```
// Java program to demonstrate LinkedList

import java.util.*;
class LinkedListDemo
 {
   public static void main(String args[])
    {
       LinkedList ll = new LinkedList();
       ll.add("KIRAN");
       ll.add("AJAY");
```

```
        ll.add("VIJAY");

    System.out.print("\n\nIt THE CONTENTS OF LINKED LIST
                                        ARE ...");

    System.out.print(ll);

     }

  }
```

HashSet

```
// Java program to demonstrate HashSet
import java.util.*;
class  HashSetDemo
{
  public  static  void  main(String  args[])
  {
     HashSet  hs = new HashSet();
     hs.add("KIRAN");
     hs.add("AJAY");
     hs.add("VIJAY");
     System.out.print(hs);
   }
}
```

TreeSet

```
// Java program to demonstrate TreeSet
import java.util.*;
class TreeSetDemo
{
  public static void main(String argS[])
  {
    TreeSet ts = new TreeSet();
    ts.add("KIRAN");
    ts.add("AJAY");
    ts.add("VIJAY");
    System.out.print("\n\nIt THE CONTENTS OF TREESET
                                        ARE ---");
    System.out.print(ts);
  }
}
```

HashMap

```
// Java program to demonstrate HashMap
import java.util.*;
class HashMapDemo
{
  public static void main(String args[])
  {
```

```
HashMap  hm = new HashMap();
hm.put("KIRAN", new Integer(74));
hm.put("AJAY", new Integer(66));
hm.put("VIJAY", new Integer(80));
Set s = hm.entrySet();
Iterator itr = s.iterator();
  While(i.hasNext())
  {
    Map.Entry  me = (Map.Entry) i.next();
    System.out.print(me.getKey() + ": ");
    System.out.print(me.getValue());
  }
 }
}
```

TreeMap

```
// Java program to demonstrate TreeMap
import java.util.*;
Class TreeMapDemo
{
  public static void main(String args[])
  {
```

```
TreeMap  tm = new TreeMap();
tm.put("KIRAN", new Integer(74));
tm.put("AJAY", new Integer(66));
tm.put("VIJAY", new Integer(80));
Set s = tm.entrySet();
Iterator itr = s.iterator();
While( i.hasNext())
{
   Map.Entry me = (Map.Entry) i.next();
   System.out.print(me.getKey() + ":");
   System.out.print(me.getValue());
}
}
}
```

Vector

```
// Java program to demonstrate Vector
import java.util.*;
class VectorDemo
{
  public static void main(String args[])
  {
     Vector v = new Vector();
```

```
V.addElement(new Integer(1));
V.addElement(new Integer(2));
V.addElement(new Integer(3));

Enumeration enm = V.elements();

System.out.print("\n\nIt THE ELEMENTS IN THE VECTOR
                                    ARE...");

 While ( enm.hasMoreElements() )
   {
     System.out.print( enm.nextElement() + " ");
   }
```

Stack

```
// Java program to demonstrate Stack.
import java.util.*;
class Stackdemo
{
 public static void main(String args[])
 {
    Stack st = new Stack();
    st.push(10);
    St.push(20);
    st.push(30);
int k = St.pop();
     System.out.print("\n\nIt THE CONTENTS OF STACK
                                    ARE...");

     System.out.print(st);
  }
}
```

Hashtable

```
// Java program to demonstrate Hashtable
import java.util.*;
class HashtableDemo
{
  public static void main(String args[])
  {
    Hashtable ht = new Hashtable();
    Enumeration names;
    String str;

    ht.put("KIRAN", 74);
    ht.put("AJAY", 80);
    ht.put("VIJAY", 84);
    names = ht.keys();
    while(names.hasMoreKeys())
    {
      str = (String) names.nextElement();
      System.out.print(str + ":" + ht.get(str));
    }
  }
}
```

Properties


```
// Java program to demonstrate properties
import java.util.*;
class Propertiesdemo
{
 public static void main(String args[])
 {
    Properties p = new Properties();
    Set s;
    String str;
    p.put("Maharashtra", "Mumbai");
    p.put("Karnataka", "Bengaluru");
    p.put("WestBengal", "Kolkata");
    s = p.keySet();
    Iterator itr = s.iterator();
    while(itr.hasNext())
    {
       str = (String) itr.next();
       System.out.print("THE CAPITAL OF " + str +
                        " IS .." + p.getProperty(str));
    }
 }
}
```

StringTokenizer

(13)

```
// Java program to demonstrate StringTokenizer.
import java.util.*;
class Stringtokenizerdemo
{
    public static void main(String args[])
    {
        String s;
        Scanner sc = new Scanner(System.in);
        System.out.print("\n\nIt ENTER A STRING----");
        s = sc.nextLine();
        StringTokenizer st = new StringTokenizer(s," ");
        System.out.print("\n\nIt THE TOKENS OF THE
                          GIVEN STRING ARE ---");

        While(st.hasMoreTokans())
        {
          System.out.println(st.nextToken());
        }
    }
}
```

BitSet

```
// Java program to demonstrate BitSet
import java.util.*;
class BitSetDemo
{
 public static void main(String args[])
 {
    BitSet bs1 = new BitSet(16);
    BitSet bs2 = new BitSet(16);
    for(int i=0; i<16; i=i+1)
    {
      if(i%2 == 0)
      { bs1.set(i);
      }
      if(i%5 != 0)
      {
        bs2.set(i);
      }
    }
System.out.print("\n\nt INITIAL PATTERN IN BS1");
System.out.print(bs1);
System.out.print("\n\nt INITIAL PATTERN IN BS2");
System.out.print(bs2);
bs2.and(bs1);
System.out.print("\n BS2 AND BS1");
System.out.print(bs2);
```

```
        bs2.or(bs1);
        System.out.print("\n BS2 OR BS1");
        System.out.print(bs2);
        bs2.xor(bs1);
        System.out.print("\n BS2 XOR BS1");
        System.out.print(bs2);
    }
}
```

Date

```
// Java program to demonstrate Date
import java.util.*;
class DateDemo
{
  public static void main(String args[])
  {
    Date d = new Date();
    System.out.print(d);
  }
}
```

Calendar

```
// Java program to demonstrate Calendar
import java.util.*;
class CalendarDemo
{
  public static void main(String args[])
  {
    Calendar c = Calendar.getInstance();
    System.out.print("\n\nt YEAR: " + c.get(Calendar.
                                        YEAR));
    System.out.print("\n\nt MONTH: " + c.get(Calendar.
                                        MONTH));
    System.out.print("\n\nt DATE: " + c.get(Calendar.
                                        DATE));
  }
}
```

Random

```
// Java program to demonstrate Random
import java.util.*;
class RandomDemo
{
  public static void main(String args[])
  {
```

```
Random r = new Random();
System.out.print("\n\n\t AN INTEGER RANDOM
                    NUMBER IS..." +r.nextInt());

System.out.print("\n\n\t A FLOAT RANDOM
                    NUMBER IS---" +r.nextFloat());

System.out.print("\n\n\t A DOUBLE RANDOM
                    NUMBER IS---" +r.nextDouble());

    }

}
```

Formatter

```
// Java program to demonstrate formatter
import java.util.*;
class FormatterDemo
{
  public static void main(String args[])
  {
    Formatter f = new Formatter();
    f.format("%s ---%d --- %f", "KIRAN", 74, 37.5);
    System.out.print(f.out);
  }
}
```

Scanner

```
// Java program to demonstrate Scanner
import java.util.*;
class ScannerDemo
{
 public static void main(String args[])
 {
    int a;
    float b;
    double c;
    Scanner sc = new Scanner(System.in);
    System.out.print("\n\n\t ENTER AN INTEGER..");
    a =sc.nextInt();
    System.out.print("\n\n\t ENTER A FLOAT VALUE..");
    b =sc.nextFloat();
    System.out.print("\n\n\t ENTER A DOUBLE VALUE");
    c = sc.nextDouble();
    System.out.print("\n\n\t THE GIVEN INTEGER IS ..."+a);
    System.out.print("\n\n\t THE GIVEN FLOAT IS ..."+b);
    System.out.print("\n\n\t THE GIVEN DOUBLE IS ..."+c);
 }
}
```

PriorityQueue


```
// Java program to demonstrate PriorityQueue
import java.util.*;
class PQDemo
{
  public static void main(String args[])
  {
    PriorityQueue pq = new PriorityQueue();
    Pq.add(10);
    Pq.add(20);
    pq.add(30);
    System.out.print(Pq);
  }
}
```

ArrayDeque

```
// Java program to demonstrate ArrayDeque.
import java.util.*;
class ADQDemo
{
  public static void main(String args[])
  {
    Deque dq = new ArrayDeque();
    dq.add("RAVI");
    dq.add("VIJAY");
    dq.add("AJAY");
```

```
    for (String s : da)
    {
      System.out.print(s);
    }
  }
}
```