ClientUDP -:

```
import java.io.IOException;
import java.net.*;
import java.util.*;

public class clientUDP implements Runnable {

	int portnumber ;
	static byte[] ipAdd ;
	static String	ipAddress;

	public clientUDP(int portnumber){
		this.portnumber = portnumber;
		new Thread(this).start();
	}

	public static void main(String args[]){
		System.out.println("Enter the number of threads you wanna start-:");
		Scanner sc =new Scanner(System.in);
		int threads = sc.nextInt();

		System.out.println("Enter the ip address of the server-: ");
		Scanner sc1 = new Scanner(System.in);
		ipAddress = sc1.nextLine();
		ipAdd = ipAddress.getBytes();

		System.out.println("------------------ TCP throughput and latency for threads-: "+threads+"-----------");

		if (threads == 1){
			new clientUDP(5555);
		}

		if (threads == 2){
			new clientUDP(7777);
			new clientUDP(6666);
		}



	}
```

```
// To create the string of the specified size given as the argument
public String createString(int size){
    int i = 0;
    byte[] sample = new byte[size];
    StringBuilder sb = new StringBuilder();
    while(i<size){
        sb.append('c');
        i++;
    }
    String sc = sb.toString();
    return sc;
}

@Override
public void run() {
    // TODO Auto-generated method stub

    DatagramSocket client = null;
    try {

        byte[] send ;
        client = new DatagramSocket();
        InetAddress ip1 = InetAddress.getByName(ipAddress);


        int[] bufferSize = {1,1024,63535};
        for (int i = 0; i < bufferSize.length; i++) {

        String sendString = createString(bufferSize[i]);
        send = sendString.getBytes();
        long startTime = System.nanoTime();

        // Create a data gram packet with ip and portnumber as argument in the constructor
        DatagramPacket sPacket = new DatagramPacket(send,send.length,ip1,portnumber);
        client.send(sPacket);

        //receive buffer the message
        byte[] receiveBuffer = new byte[65536];
        DatagramPacket getReply = new DatagramPacket(receiveBuffer,receiveBuffer.length);

        // Get the data from the server
        byte[] data= getReply.getData();
```

```
                String data1 = data.toString();
                long stopTime = System.nanoTime();
                long total = stopTime-startTime;

                System.out.printf("The throughput of size %4d is %6.4f
MBps\n",bufferSize[i],(15258.78*bufferSize[i])/total);
                System.out.printf("The latency of size %4d is %6.4f
ms\n",bufferSize[i],(total*1000/(15258.78*bufferSize[i])));

                }

            } catch (UnknownHostException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        }

}
```

serverUDP -:

```
import java.io.IOException;
import java.net.*;

public class serverUDP implements Runnable {
    int portnumber;


    public serverUDP(int portnumber){
        this.portnumber = portnumber;
        new Thread(this).start();
    }


    public static void main( String args[]) throws UnknownHostException{
```

```
        InetAddress ip1= InetAddress.getLocalHost();
        System.out.println("The Ip address of the server is -:"+ip1);

        int port1 = 5555;
        int port2 = 6666;
        int port3 = 7777;

        new serverUDP(port1);
        new serverUDP(port2);
        new serverUDP(port3);

    }



    @SuppressWarnings("resource")
    @Override
    public void run() {
        // TODO Auto-generated method stub
        try {
            DatagramSocket socket;
            socket = new DatagramSocket(portnumber);
            // byte array of buffer size to get the data from client
            byte[] Buffer = new byte[65537];
            DatagramPacket rPacket = new DatagramPacket(Buffer,Buffer.length);




            while(true)
            {
                socket.receive(rPacket);
                System.out.println("Data received");
                // Data received from the client
                byte[] receive = rPacket.getData();

                String s = new String(receive,0,rPacket.getLength());

                // Send the same size of data to the client
                DatagramPacket reply = new
DatagramPacket(s.getBytes(),s.getBytes().length,rPacket.getAddress(),rPacket.getPort());
                // To send data to client back
                socket.send(reply);
```

```
                }

            } catch (SocketException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        }

}
```

TcpServer -:

```
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;

public class TcpServer implements Runnable{

    int portnumber ;


    public TcpServer(int portnumber ){


        this.portnumber = portnumber;
        // Starts the new thread
        new Thread(this).start();
    }

    public static void main(String args[]) throws IOException, InterruptedException{
        // To get the ip address of the server
        InetAddress ip = InetAddress.getLocalHost();
```

```
            System.out.println("The ip address of the server is-:" + ip);


            int port = 7777;
            int port1 = 8888;
            int port2 = 9999;


            new TcpServer(port);
            new TcpServer(port1);
            new TcpServer(port2);


            }

        @Override
        public void run() {
            // TODO Auto-generated method stub
            int[] bufferSize = {1,1024,65536};

            try {
                ServerSocket server = new ServerSocket(portnumber);

                @SuppressWarnings("resource")
                // Create a server socket
                Socket serverSocket = new Socket();
                // Listens to the incoming request
                serverSocket = server.accept();




                ObjectInputStream obj_in = new
ObjectInputStream(serverSocket.getInputStream());
                ObjectOutputStream obj_out = new
ObjectOutputStream(serverSocket.getOutputStream());
//              while(true){
                for (int i = 0; i < 3; i++) {
                // To recieve the data from the client
                Object obj = obj_in.readObject();
                System.out.println("Data Received");
                // To send the data back to client of same size
                obj_out.writeObject(obj);
```

```
//                  }
                    }
            } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
            } catch (ClassNotFoundException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
            }




    }

}
```

TcpClient -:

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.Scanner;

public class TcpClient implements Runnable {
    static int threads;
    int size;
    static String mike;
    int portnumber;
    static String sen;

    public TcpClient(int portnumber, String size ){
        this.portnumber = portnumber;
        this.sen = size;
        // To start the new thread
        new Thread(this).start();
    }

    public static void main(String args[]){
        input();
```

```
        if (threads == 1){
        new TcpClient(7777,sen);
        }

        else if (threads ==2){
            new TcpClient(8888,sen);
            new TcpClient(9999,sen);
        }

        else{
            System.out.println("Enter the value of threads correctly-:");
        }

    }

    public static void input(){
        System.out.println("Enter the number of threads u wanna start");
        Scanner sc = new Scanner(System.in);
        threads = sc.nextInt();

        System.out.println("Enter the ip address of the server-: ");
        Scanner sc1 = new Scanner(System.in);
        sen = sc1.nextLine();

        System.out.println("------------------ TCP throughput and latency for threads-: "+threads+"-----------");

    }

    // To create the string of size specified in the argument
    public static String stringSize(int stringSize){
        int i = 0;
        StringBuilder sb = new StringBuilder();
        String sc;
        while(i<= stringSize){
            sb.append('a');
            i++;
        }
        sc = sb.toString();
        return sc;
    }

    @Override
    public void run() {
```

```
        // TODO Auto-generated method stub
        int[] arraySize = {1,1024,65536};

        try {
            // Create socket connection to specefied ip adress and port number
            Socket client = new Socket(sen,portnumber);
            ObjectOutputStream obj_out = new
ObjectOutputStream(client.getOutputStream());
            ObjectInputStream obj_in = new ObjectInputStream(client.getInputStream());

            for (int i = 0; i < arraySize.length; i++) {


            String inputString = stringSize(arraySize[i]);
            byte[ ] sizeass = inputString.getBytes();
            long starttime = System.nanoTime();

            // To send data to the server
            obj_out.writeObject(inputString);

            // To receive data from server(ack)
            Object obj = obj_in.readObject(); ;
            long stopTime = System.nanoTime();
            long total = stopTime-starttime;

            System.out.printf("The throughput with buffer size %4d is %6.4f
MBps\n",arraySize[i],(15258.78*arraySize[i])/total);
            System.out.printf("The latency with buffer size %4d is %6.2f
ms\n",arraySize[i],(total*1000/(15258.78*arraySize[i])));
            }
        } catch (IOException e) {
            // TOttDO Auto-generated catch block
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

}
```

DISK PERFORMANCE PROGRAM

diskPerformance - :

```
import java.io.*;
import java.util.*;



public class diskPerformance implements Runnable {

	static String readFileName= randomString(0);
	static int threads;




	public static void main(String args []) throws IOException, InterruptedException{
	newFile();
	System.out.println("Enter the number of threads you want to create-:");
	Scanner sc = new Scanner(System.in);
	threads = sc.nextInt();

	Thread[] threadlist = new Thread[threads];

	for (int i = 0; i < threadlist.length; i++) {
		threadlist[i] = new Thread(new diskPerformance());
		threadlist[i].start();
	}


	for (int i = 0; i < threadlist.length; i++) {
		threadlist[i].join();
	}

	}



	//@ To create a 10MB file for reading the data
	public static void newFile() throws IOException{
		File newFile = new File(readFileName+".txt");
		newFile.createNewFile();
		String writeData = randomString((2*1024*1024)+(8*1024));
		BufferedWriter bf = new BufferedWriter(new FileWriter(newFile));
		// 10MB file is created for reading sequentially and randomly
```

```
        for (int i = 0; i < 5; i++) {
        bf.write(writeData);
        }
//          System.out.println("The size of the file is"+(newFile.length()/(1024*1024))+ "MB");

    }



    //Create a random string and return it for writing the file name and
    //writing data in the file created for sequential write and random
    //write

    public static synchronized String randomString(double size)
    {

        StringBuilder sb= new StringBuilder();
        for (char c = 'a';    c <= 'z'; c++) {
            sb.append(c);
        }

        char[] forwriting;
        forwriting = sb.toString().toCharArray();
        Random random = new Random();
        StringBuilder sb1 = new StringBuilder();
        if(size == 0){
            for (int i = 0; i <10 ; i++) {
                char ch1 = forwriting[random.nextInt(forwriting.length)];
                sb1.append(ch1);
        }
        }
        else{
        for (int i = 0; i <size ; i++) {
            char ch1 = forwriting[random.nextInt(forwriting.length)];
            sb1.append(ch1);
        }
        }
        return sb1.toString();
    }



    // For writing sequentially and measure the time with the data size
    //1byte 1KB and 1MB
```

```
	public static synchronized void sequentialWrite() throws IOException
	{

		int[] stringSize = {1,1024,1024*1024};
		String writeData;

		File newFile = new File(randomString(0)+".txt");
		for (int i = 0; i < stringSize.length; i++) {

			newFile.createNewFile();
		writeData = randomString(stringSize[i]);

		FileWriter fw = new FileWriter(newFile);
		BufferedWriter bf = new BufferedWriter(fw);
		long startTime = System.nanoTime();
		if(stringSize[i] == 1){
			for (int j = 0; j < 10*1024*1024; j++) {
				bf.write(writeData);
			}
		}

		else if(stringSize[i]==1024){
			for (int j2 = 0; j2 < 10*1024; j2++) {
					bf.write(writeData);
				}
		}
		else{
			for (int j = 0; j < 10; j++) {


		bf.write(writeData);
			}
		}
		bf.close();
		long stopTime = System.nanoTime();
		long total = stopTime - startTime;
//		System.out.printf("%4dbytes: %6.2fmsec%n", stringSize[i], (total / 1000000.0));
		System.out.printf("The throughput of the sequential write for buffersize %4dbytes is %6.2f MBps%n", stringSize[i],(10000.0/(total/1000000.0)));
		System.out.printf("The latency of the sequential write for buffersize %4dbytes is %6.2f ms%n", stringSize[i],((total/10000000.0)));
		}

	}
```

```
// To read sequentially from the file that is generated at the start.
// It will read the data with the buffer size of 1Byte 1 KB and 1 MB
// Calculate the throughput and latency and display on the console

public static synchronized void sequentialRead() throws FileNotFoundException
{
	double storeTime[] = new double[3];
	int time = 0;

	File myfile = new File(readFileName+".txt");

	int[ ] bufferArray = {1,1024,1024*1024};
	int bufferSize;
	System.out.println("----------------------Sequential Read----------------------");
	for (int i = 0; i < bufferArray.length; i++) {
		bufferSize = bufferArray[i];

	try(FileInputStream fin = new FileInputStream(myfile)){
		long startTime = System.nanoTime();
		byte[] byteSize = new byte[bufferSize];
		while((fin.read(byteSize)) != -1){

		}
		long stopTimer = System.nanoTime();
		long total = stopTimer-startTime;

		System.out.printf("The throughput of the sequential read for buffersize %4dbytes is %6.2f MBps%n", bufferArray[i],(10000.0/(total/1000000.0)));
		System.out.printf("The latency of the sequential write for buffersize %4dbytes is %6.2f ms%n", bufferArray[i],((total/10000000.0)));

	} catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}

	}
}

// To randomly write the data on the disk
// Write the data with buffersize of varying size of 1 Byte 1KB and 1MB

public static synchronized void randomWrite()
```

```
{

        String stringSize ;
        // Array of 1byte,1KB and 1MB
        int[] stringWrite = {1,1024,1024*1024};

        String path = randomString(0)+".txt";
        System.out.println("-----------------------Random Write-----------------------");
        for (int i = 0; i < stringWrite.length; i++) {
            //stores randomly generated string ov varying size
            stringSize = randomString(stringWrite[i]);
        try(RandomAccessFile raf = new RandomAccessFile(path, "rw")){
            long startTime = System.nanoTime();
            if(stringWrite[i] == 1){
                for (int j = 0; j < 1024*1024*10; j++) {
                    raf.writeChars(stringSize);
                }
            }
            else if(stringWrite[i] == 1024){
                for (int j = 0; j < 1024*10; j++) {
                    raf.writeChars(stringSize);
                }
            }
            else{
                for (int j = 0; j < 10; j++) {
                    raf.writeChars(stringSize);
                }
            }

            long stopTime = System.nanoTime();
            long total = stopTime-startTime;
            System.out.printf("The throughput of the random write for buffersize %4dbytes is %6.2f MBps%n", stringWrite[i],(10000.0/(total/1000000.0)));
            System.out.printf("The latency of the sequential write for buffersize %4dbytes is %6.2f ms%n", stringWrite[i],((total/10000000.0)));

        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

```
        }

    }

    // To randomly read the data from the file created
    // Will read the data with varying buffer size 1 Byte, 1KB and 1MB

    public static synchronized void randomRead()
    {

        String stringSize ;
        // Array of 1byte,1KB and 1MB
        int[] stringWrite = {1,1024,1024*1024};

        System.out.println("----------------------Random Read----------------------");
        for (int i = 0; i < stringWrite.length; i++) {
            //stores randomly generated string of varying size of 1Byte 1KB and 1MB
            stringSize = randomString(stringWrite[i]);
        try(RandomAccessFile raf = new RandomAccessFile(readFileName+".txt", "rw")){
            long startTime = System.nanoTime();
            raf.seek(0);
            raf.readLine();
            long stopTime = System.nanoTime();
            long total = stopTime-startTime;
            System.out.printf("The throughput of the random read for buffersize %4dbytes is %6.2f MBps%n", stringWrite[i],(10000.0/(total/1000000.0)));
            System.out.printf("The latency of the sequential write for buffersize %4dbytes is %6.2f ms%n", stringWrite[i],(total/10000000.0));

        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        }

    }

    @Override
    public void run() {
        // TODO Auto-generated method stub
```

```
            try {
                sequentialWrite();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            try {
                sequentialRead();
            } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            randomWrite();
            randomRead();

        }


}
```

----------CPU Performance----------

CpuBenchmark -:

```
import java.util.Scanner;
//import java.util.concurrent.Callable;

public class CpuBenchmark implements Runnable{

    long t_iter;
    Thread thread;
    CpuBenchmark(long t_iter) throws InterruptedException
    {
        this.t_iter= t_iter;
        //thread.start();
        //Thread.currentThread().join();

    }

    public static void main(String args[]) throws InterruptedException
    {
        int k=0,j=0;
```

```
    float IOPS = 0;
    float GIOPS =0;
    System.out.println("Enter the number of threads");
    Scanner scanner = new Scanner(System.in);
    int threads = scanner.nextInt();
    System.out.println("Enter opreations");
    Scanner scanner1 = new Scanner(System.in);
    long operations = scanner1.nextLong();
    Thread[] threadArray = new Thread[threads];
    long sTime = System.currentTimeMillis();
    System.out.println(sTime);
    while(k<threads)
    {
        CpuBenchmark bm = new CpuBenchmark(operations);
        threadArray[k] = new Thread(bm);
        threadArray[k].start();
        k++;
    }

    // Will start the new threads of array
    while(j<threads)
    {
        threadArray[j].join();;
        j++;
    }

    long total = System.currentTimeMillis() - sTime;
    System.out.println("Total time taken"+ total);
    IOPS = operations*20*1000*threads/total;
    GIOPS = IOPS/1000000000;
    System.out.println("ILOPS : " + IOPS);
    System.out.println("GILOPS : " + GIOPS);
    scanner.close();
    scanner1.close();
}

public void run() {

    // Operations to be done for calculating IOPS
    long a = 40,b = 200,c = 20 ;
    long j ;
    for(j =0; j<t_iter; j++)
    {
        //System.out.println(Thread.currentThread().getName());
```

```
            a = a+10;
            b = b+j;
            a =a+b;
            b =b+b;
            c = a*b;
            b = 70+c;
            c = a+10;
            a = b+c;
            a = a+a;
            a = 100*c;
            b= 200+c;
            b = c+j;
            c = c+b;
            c = a+b;
            a = b+c;
            a =a+j;
            c = c*a;
            b= a+c;
            b= b+c;
            j++;

        }
        //System.out.println(j);

    }

}
```

CpuBenchmark -:

```
import java.util.Scanner;

public class CpuBenchmarkFloat implements Runnable {

    long t_iter;
    Thread thread;
    CpuBenchmarkFloat(long t_iter) throws InterruptedException
    {
        this.t_iter= t_iter;
        //thread.start();
        //Thread.currentThread().join();
```

```
}

public static void main(String args[]) throws InterruptedException
{
    int k=0,j=0;
    float IOPS = 0;
    float GIOPS =0;
    System.out.println("Enter the number of threads");
    Scanner scanner = new Scanner(System.in);
    int threads = scanner.nextInt();
    System.out.println("Enter opreations");
    Scanner scanner1 = new Scanner(System.in);
    long operations = scanner1.nextLong();
    Thread[] threadArray = new Thread[threads];
    long sTime = System.currentTimeMillis();
    System.out.println(sTime);
    while(k<threads)
    {
        CpuBenchmark bm = new CpuBenchmark(operations);
        threadArray[k] = new Thread(bm);
        threadArray[k].start();
        k++;
    }

    while(j<threads)
    {
        threadArray[j].join();;
        j++;
    }

    long total = System.currentTimeMillis() - sTime;
    System.out.println("Total time taken"+ total);
    IOPS = operations*20*1000*threads/total;
    GIOPS = IOPS/1000000000;
    System.out.println("FLOPS : " + IOPS);
    System.out.println("GFLOPS : " + GIOPS);
    scanner.close();
    scanner1.close();
}

public void run() {
```

```
        double a = 40,b = 200,c = 20 ;
        long j ;
        for(j =0; j<t_iter; j++)
        {
            //System.out.println(Thread.currentThread().getName());
            a = a+10.89;
            b = b+j;
            a =a+b;
            b =b+b;
            c = a*b;
            b = 70.88+c;
            c = a+10.09;
            a = b+c;
            a = a+a;
            a = 100.8*c;
            b= 200.49+c;
            b = c+j;
            c = c+b;
            c = a+b;
            a = b+c;
            a =a+j;
            c = c*a;
            b= a+c;
            b= b+c;
            j++;

        }
        //System.out.println(j);

    }
}
```

Cpu2 -:

```
import java.io.*;
import java.util.*;
import java.util.concurrent.*;
/*import java.util.Scanner;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
```

```
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
*/
public class Cpu2 implements Callable<ArrayList<Long>> {


            long sTime;
            Cpu2(long startTime)
            {
                this.sTime = startTime;
            }
            //call method implementing callable
            public ArrayList<Long> call() {
                // TODO Auto-generated method stub
                long j = 0;
                long a = 5,b = 10;
                long c = 0,e =0, f = 0,x= 89, y= 92, z =88;
                long fTime = sTime;
                long TimeNow = sTime;
                //boolean flag = true;
                ArrayList<Long> list = new ArrayList<Long>();
                //  multiplication operation
                while(TimeNow<=(fTime+600000))
                {
                    a = a+10;
                    a = x+y;
                    b = 90+17;
                    c = x*y;
                    e= 100+b;
                    f = 10+90;
                    a = a+ 90*89;
                    a = 666+88;
                    f = 12+x;
                    e = 99*b;
                    a = 10000*12;
                    b = 78*+40;
                    c= 12*70;
                    a = a+b;
                    b = a+50;
                    c = 9000*1000;
                    a = x*y;
                    b = 10+f;
                    y = a*b;
                    x =c+e;
```

```
                    j++;
                    TimeNow = System.currentTimeMillis();

                    if(TimeNow>=(sTime+1000))
                    {

                        list.add(j*22);
                        sTime = TimeNow;
                        j =0;
                    }

                }
                return list;
            }

            public static void writetoFile (String filename, long[]array)
throws IOException{
                  BufferedWriter oWriter = null;
                  oWriter = new BufferedWriter(new FileWriter(filename));
                  for (int i = 0; i < array.length; i++) {
                    oWriter.write(Long.toString(array[i]));
                    oWriter.newLine();
                  }
                  oWriter.flush();
                  oWriter.close();
                }


            @SuppressWarnings("unchecked")
            public static void main(String args[]) throws IOException,
InterruptedException, ExecutionException{
                System.out.println("Enter the number of threads you wanna start
-:");
                Scanner sc = new Scanner(System.in);
                int threads = sc.nextInt();
                ArrayList<Long> tolist = new ArrayList<Long>();
                ArrayList<ArrayList<Long>> ListLists = new
ArrayList<ArrayList<Long>>();
                long[] sampleList = new long[600];
                int k=0,j=0;
                // Executor service used to create a pool of threads

                ExecutorService execPool = Executors.newFixedThreadPool(4);
```

```
                Future<ArrayList<Long>>[] future = new Future[threads];

                long startTime = System.currentTimeMillis();
                System.out.println(startTime);
                while(j<threads)
                {
                    future[j] = execPool.submit(new Cpu2(startTime));
                    j++;
                }
                while(k<threads)
                {
                    tolist = future[k].get();
                    for(int n=0;n<tolist.size();n++)
                    {
                        sampleList[n] = sampleList[n]+tolist.get(n);
                    }
                    k++;
                }

                writetoFile("cpuSamples.txt",sampleList);
                System.out.println("end");
            }




}
```

Cpu2float -:

```
import java.io.*;
import java.util.*;
import java.util.concurrent.*;
/*import java.util.Scanner;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
*/
```

```
public class Cpu2float implements Callable<ArrayList<Long>> {


            long sTime;
            Cpu2float(long startTime)
            {
                this.sTime = startTime;
            }
            //call method implementing callable
            public ArrayList<Long> call() {
                // TODO Auto-generated method stub
                double a = 40,b = 200,c = 20 ;
                long j =0;
                long fTime = sTime;
                long TimeNow = sTime;
                //boolean flag = true;
                ArrayList<Long> list = new ArrayList<Long>();
                //  multiplication operation
                while(TimeNow<=(fTime+600000))
                {
                    a = a+10.89;
                    b = b+j;
                    a =a+b;
                    b =b+b;
                    c = a*b;
                    b = 70.88+c;
                    c = a+10.09;
                    a = b+c;
                    a = a+a;
                    a = 100.8*c;
                    b= 200.49+c;
                    b = c+j;
                    c = c+b;
                    c = a+b;
                    a = b+c;
                    a =a+j;
                    c = c*a;
                    b= a+c;
                    b= b+c;
                    j++;
                    TimeNow = System.currentTimeMillis();

                    if(TimeNow>=(sTime+1000))
                    {
```

```
                    list.add(j*22);
                    sTime = TimeNow;
                    j =0;
                }

            }
            return list;
        }

        public static void writetoFile (String filename, long[]array)
throws IOException{
              BufferedWriter oWriter = null;
              oWriter = new BufferedWriter(new FileWriter(filename));
              for (int i = 0; i < array.length; i++) {
                oWriter.write(Long.toString(array[i]));
                oWriter.newLine();
              }
              oWriter.flush();
              oWriter.close();
            }


        @SuppressWarnings("unchecked")
        public static void main(String args[]) throws IOException,
InterruptedException, ExecutionException{
            System.out.println("Enter the number of threads you wanna start
-:");
            Scanner sc = new Scanner(System.in);
            int threads = sc.nextInt();
            ArrayList<Long> tolist = new ArrayList<Long>();
            ArrayList<ArrayList<Long>> ListLists = new
ArrayList<ArrayList<Long>>();
            long[] sampleList = new long[600];
            int k=0,j=0;
            // Executor service used to create a pool of threads

            ExecutorService execPool = Executors.newFixedThreadPool(4);
            Future<ArrayList<Long>>[] future = new Future[threads];

            long startTime = System.currentTimeMillis();
            System.out.println(startTime);
            while(j<threads)
            {
```

```
                future[j] = execPool.submit(new Cpu2float(startTime));
                j++;
            }
            while(k<threads)
            {
                tolist = future[k].get();
                for(int n=0;n<tolist.size();n++)
                {
                    sampleList[n] = sampleList[n]+tolist.get(n);
                }
                k++;
            }

            writetoFile("cpuSamples.txt",sampleList);
            System.out.println("end");
        }



}
```