1)CRC:

```
 import java.util.*;
public class CRC {
   public  static void main(String args[]){
      Scanner sc = new Scanner(System.in);
      int size_data,size_gen;
      System.out.println("Enter Data size");
      size_data = sc.nextInt();
      int data[] = new int[size_data];
      for(int i=0;i<size_data;i++){
         System.out.println("Enter bit:"+(size_data-i)+":");
         data[i] = sc.nextInt();
      }
      System.out.println("Enter Generator size");
      size_gen = sc.nextInt();
      int gen[] = new int[size_gen];
      for(int i=0;i<size_gen;i++){
         System.out.println("Enter bit:"+(size_gen-i)+":");
         gen[i] = sc.nextInt();
      }
      int rem[] = divide_data(data,gen);
      System.out.println("Generated crc code is:");
      for(int i=0;i<data.length;i++)
          System.out.println(data[i]);
      for(int i=0;i<gen.length;i++)
          System.out.println(gen[i]);
      int sentData[] = new int[data.length+rem.length-1];
      System.out.println("Enter bits in array you ant to send:");
      for(int i=0;i<sentData.length;i++){
         System.out.println("Enter bit:"+(sentData.length-1)+":");
         sentData[i] = sc.nextInt();
      }
      recieveData(sentData,gen);
   }
   static int[] divide_data(int oldData[],int gen[]){
      int rem[] = new int[gen.length];
      int data[] = new int[oldData.length+ gen.length];
      System.arraycopy(oldData,0,data,0,oldData.length);
      System.arraycopy(data,0,rem,0,gen.length);
      for(int i=0;i<oldData.length;i++){
         System.out.println("First data bit:"+rem[0]);
         System.out.println("Remainder:");
         if(rem[0]==1){
            for(int j=1;j<gen.length;j++){
```

```
                rem[j-1] = exor(rem[j],gen[j]);
                System.out.println(rem[j-1]);
            }
        }else{
            for(int j=1;j<gen.length;j++){
                rem[j-1] = exor(rem[j],0);
                System.out.println(rem[j-1]);
            }
        }
        rem[gen.length-1] = data[i+gen.length];
        System.out.println(rem[gen.length-1]);
    }
    return  rem;
  }
  static  int exor(int a,int b){
    if(a==b) return 0;
    return  1;
  }
  static  void recieveData(int data[],int gen[]){
    int rem[] = divide_data(data,gen);
    for(int i=0;i<rem.length;i++){
        if(rem[i]!=0){
            System.out.println("Corrupted data");
            return;
        }
    }
    System.out.println("Data Recieved Success");
  }

}
```

2)BellmanFOrd:

```
 import java.util.Scanner;

public class BellmanFord
{
   private int distances[];
   private int numberofvertices;
   public static final int MAX_VALUE = 999;

   public BellmanFord(int numberofvertices)
   {
      this.numberofvertices = numberofvertices;
```

```
    distances = new int[numberofvertices + 1];
}

public void BellmanFordEvaluation(int source, int adjacencymatrix[][])
{
    for (int node = 1; node <= numberofvertices; node++)
    {
        distances[node] = MAX_VALUE;
    }

    distances[source] = 0;
    for (int node = 1; node <= numberofvertices - 1; node++)
    {
        for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)
        {
            for (int destinationnode = 1; destinationnode <= numberofvertices; destinationnode++)
            {
                if (adjacencymatrix[sourcenode][destinationnode] != MAX_VALUE)
                {
                    if (distances[destinationnode] > distances[sourcenode]
                            + adjacencymatrix[sourcenode][destinationnode])
                        distances[destinationnode] = distances[sourcenode]
                                + adjacencymatrix[sourcenode][destinationnode];
                }
            }
        }
    }

    for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)
    {
        for (int destinationnode = 1; destinationnode <= numberofvertices; destinationnode++)
        {
            if (adjacencymatrix[sourcenode][destinationnode] != MAX_VALUE)
            {
                if (distances[destinationnode] > distances[sourcenode]
                        + adjacencymatrix[sourcenode][destinationnode])
                    System.out.println("The Graph contains negative egde cycle");
            }
        }
    }

    for (int vertex = 1; vertex <= numberofvertices; vertex++)
    {
        System.out.println("distance of source  " + source + " to "
```

```
                + vertex + " is " + distances[vertex]);
        }
    }

    public static void main(String... arg)
    {
        int numberofvertices = 0;
        int source;
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of vertices");
        numberofvertices = scanner.nextInt();

        int adjacencymatrix[][] = new int[numberofvertices + 1][numberofvertices + 1];
        System.out.println("Enter the adjacency matrix");
        for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)
        {
            for (int destinationnode = 1; destinationnode <= numberofvertices; destinationnode++)
            {
                adjacencymatrix[sourcenode][destinationnode] = scanner.nextInt();
                if (sourcenode == destinationnode)
                {
                    adjacencymatrix[sourcenode][destinationnode] = 0;
                    continue;
                }
                if (adjacencymatrix[sourcenode][destinationnode] == 0)
                {
                    adjacencymatrix[sourcenode][destinationnode] = MAX_VALUE;
                }
            }
        }

        System.out.println("Enter the source vertex");
        source = scanner.nextInt();

        BellmanFord bellmanford = new BellmanFord(numberofvertices);
        bellmanford.BellmanFordEvaluation(source, adjacencymatrix);
        scanner.close();
    }
}
```

3) LEAKy:

```
import java.util.Scanner;
```

```
public class leaky {
   public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        int bucket=0,op_rate,i,n,bsize;
        System.out.println("enter number of packets");
        n=sc.nextInt();
        int pkt[]=new int[n];
        System.out.println("enter he output rate of the bucket");
        op_rate=sc.nextInt();
        System.out.println("enter the bucket size");
        bsize=sc.nextInt();
        System.out.println("enter the arriving packets(size)");
        for(i=0;i<n;i++)
            pkt[i]=sc.nextInt();
        System.out.println("\nsec\tpsize\tbucketsize\taccept/reject\tpkt_send");

System.out.println("-------------------------------------------------------");
        for(i=0;i<n;i++)
        {
            System.out.print(i+1+"\t"+pkt[i]+"\t");
            if(bucket+pkt[i]<=bsize)
            {
                bucket+=pkt[i];

System.out.println(bucket+"\t\taccept\t\t"+min(bucket,op_rate)+"\n");
                bucket=sub(bucket,op_rate);
```

```
            }
            else
            {
                System.out.println(bucket+"\t\treject\t"+(bucket+pkt[i]-bsize)
+"\t"+min(bsize,op_rate)+"\n");
                bucket=bsize;
                bucket=sub(bucket,op_rate);
            }
        }
        while(bucket!=0)
        {
            System.out.print(++i +"\t 0
\t"+bucket+"\t\taccept\t\t"+min(bucket,op_rate)+"\n");
            bucket=sub(bucket,op_rate);
        }
        sc.close();
    }
    static int min(int a,int b)
    {
        return(a<b)?a:b;
    }
    static int sub(int a,int b)
    {
        return(a-b)>0?(a-b):0;
    }
```

```
}
```

4)RSA:

```
import java.math.*;
import java.util.Scanner;
public class RSA {
   public static void main(String args[])
   {
        Scanner sc=new Scanner(System.in);
        int p,q,n,z,d=0,e,i;
        System.out.println("Enter the number to be encrypted and decrypted");
        int msg=sc.nextInt();
        double c;
        BigInteger msgback;
        System.out.println("Enter 1st prime number p");
        p=sc.nextInt();
        System.out.println("Enter 2nd prime number q");
        q=sc.nextInt();

        n=p*q;
        z=(p-1)*(q-1);
        System.out.println("the value of z = "+z);

        for(e=2;e<z;e++)
        {
            if(gcd(e,z)==1)
            {
                break;
            }
        }
        System.out.println("the value of e = "+e);
        for(i=0;i<=9;i++)
        {
            int x=1+(i*z);
            if(x%e==0)
            {
                d=x/e;
                break;
            }
        }
        System.out.println("the value of d = "+d);
        c=(Math.pow(msg,e))%n;
        System.out.println("Encrypted message is : -");
        System.out.println(c);
        BigInteger N = BigInteger.valueOf(n);
        BigInteger C = BigDecimal.valueOf(c).toBigInteger();
        msgback = (C.pow(d)).mod(N);
        System.out.println("Decrypted message is : -");
        System.out.println(msgback);
```

```
    }

    static int gcd(int e, int z)
    {
        if(e==0)
            return z;
        else
            return gcd(z%e,e);
    }
}
```

5)TCP:

client:

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

public class Client
{
    // initialize socket and input output streams
    private Socket socket            = null;
    private DataInputStream  input   = null;
    private DataOutputStream out     = null;

    // constructor to put ip address and port
    public Client(String address, int port)
    {
        // establish a connection
        try
        {
            socket = new Socket(address, port);
            System.out.println("Connected");

            // takes input from terminal
            input  = new DataInputStream(System.in);

            // sends output to the socket
            out = new DataOutputStream(socket.getOutputStream());
        }
        catch(UnknownHostException u)
        {
            System.out.println(u);
        }
        catch(IOException i)
        {
```

```
            System.out.println(i);
        }

        // string to read message from input
        String line = "";

        // keep reading until "Over" is input
        while (!line.equals("Over"))
        {
            try
            {
                line = input.readLine();
                out.writeUTF(line);
            }
            catch(IOException i)
            {
                System.out.println(i);
            }
        }

        // close the connection
        try
        {
            input.close();
            out.close();
            socket.close();
        }
        catch(IOException i)
        {
            System.out.println(i);
        }
    }

    public static void main(String args[])
    {
        Client client = new Client("127.0.0.1", 5000);
    }
}
```

Server:

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

public class Server
{
```

```
//initialize socket and input stream
private Socket         socket   = null;
private ServerSocket   server   = null;
private DataInputStream in       =  null;

// constructor with port
public Server(int port)
{
    // starts server and waits for a connection
    try
    {
        server = new ServerSocket(port);
        System.out.println("Server started");

        System.out.println("Waiting for a client ...");

        socket = server.accept();
        System.out.println("Client accepted");

        // takes input from the client socket
        in = new DataInputStream(
            new BufferedInputStream(socket.getInputStream()));

        String line = "";

        // reads message from client until "Over" is sent
        while (!line.equals("Over"))
        {
            try
            {
                line = in.readUTF();
                System.out.println(line);

            }
            catch(IOException i)
            {
                System.out.println(i);
            }
        }
        System.out.println("Closing connection");

        // close connection
        socket.close();
        in.close();
    }
```

```
        catch(IOException i)
        {
            System.out.println(i);
        }
    }

    public static void main(String args[])
    {
        Server server = new Server(5000);
    }
}
```

6) UDP:

client:
```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class udpBaseClient_2
{
    public static void main(String args[]) throws IOException
    {
        Scanner sc = new Scanner(System.in);

        // Step 1:Create the socket object for
        // carrying the data.
        DatagramSocket ds = new DatagramSocket();

        InetAddress ip = InetAddress.getLocalHost();
        byte buf[] = null;

        // loop while user not enters "bye"
        while (true)
        {
            String inp = sc.nextLine();

            // convert the String input into the byte array.
            buf = inp.getBytes();

            // Step 2 : Create the datagramPacket for sending
            // the data.
            DatagramPacket DpSend =
                  new DatagramPacket(buf, buf.length, ip, 1234);
```

```
            // Step 3 : invoke the send call to actually send
            // the data.
            ds.send(DpSend);

            // break the loop if user enters "bye"
            if (inp.equals("bye"))
                break;
        }
    }
}
```

server:

```
   import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

public class udpBaseServer_2
{
    public static void main(String[] args) throws IOException
    {
        // Step 1 : Create a socket to listen at port 1234
        DatagramSocket ds = new DatagramSocket(1234);
        byte[] receive = new byte[65535];

        DatagramPacket DpReceive = null;
        while (true)
        {

            // Step 2 : create a DatgramPacket to receive the data.
            DpReceive = new DatagramPacket(receive, receive.length);

            // Step 3 : revieve the data in byte buffer.
            ds.receive(DpReceive);

            System.out.println("Client:-" + data(receive));

            // Exit the server if the client sends "bye"
            if (data(receive).toString().equals("bye"))
            {
                System.out.println("Client sent bye.....EXITING");
                break;
            }
```

```
            // Clear the buffer after every message.
            receive = new byte[65535];
        }
    }

    // A utility method to convert the byte array
    // data into a string representation.
    public static StringBuilder data(byte[] a)
    {
        if (a == null)
            return null;
        StringBuilder ret = new StringBuilder();
        int i = 0;
        while (a[i] != 0)
        {
            ret.append((char) a[i]);
            i++;
        }
        return ret;
    }
}
```