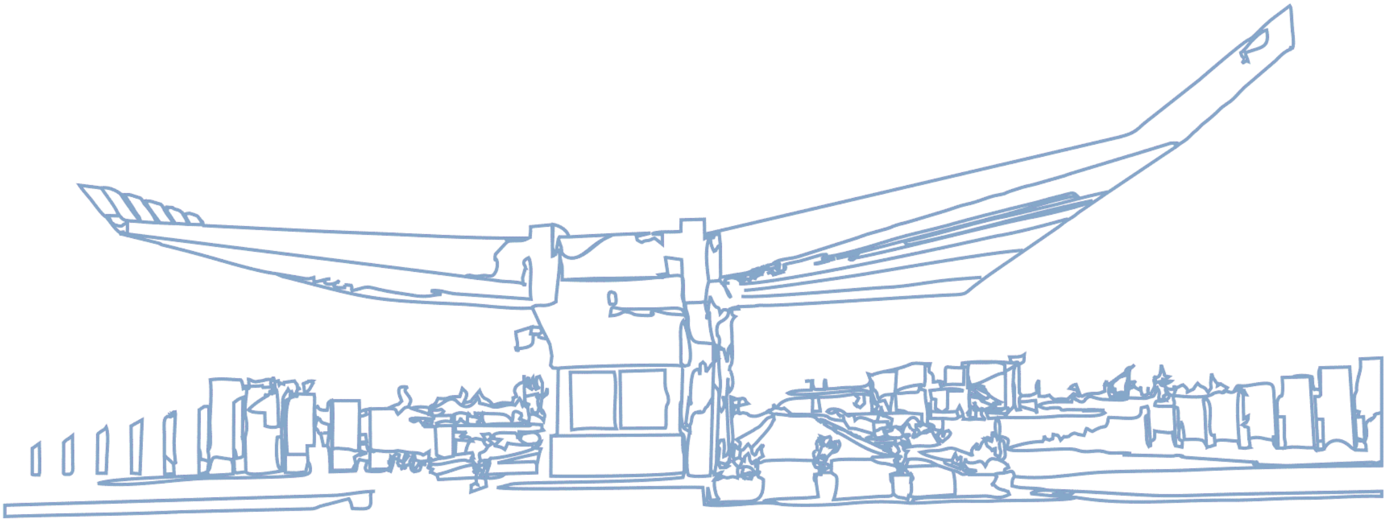
EPOKA UNIVERSITY | Computer Engineering Department

CEN 506 - DISTRIBUTED SYSTEMS

Assignment Nr. 1



PREPARED:
Baftjar TABAKU
Erli REÇI

04.01.2020
Epoka University
Tirana, ALBANIA

ACCEPTED:
Assoc.Prof.Dr. Dimitrios Karras

On the further proceeding part of the solution are included full solution of the exercises, with the respective screenshots and the video demonstration parts and source file for each solution.

Program used to execute this solution was IntellijIDea from JetBrains.

Students that worked for these solution are Baftjar Tabaku (btabaku16@epoka.edu.al) and Erli Reçi (ereci16@epoka.edu.al) .

Solutions are approached with the socket programming and multithreading using Java programming language also combined with maven project manager for better database management and connection, in a way that makes sense and illustrate the real time Distributed Systems.

The current process of the implementation of the exercises is supervised by Assoc.Prof.Dr. Dimitrios Karras (dkarras@epoka.edu.al).

Contributors GitHub: Baftjar Tabaku (https://github.com/BTabaku),

Erli Reçi (https://github.com/ErliReci).

Project GitHub link: https://github.com/BTabaku/DistributedSystemsExercises

1. Case study on Common Object Request Broker Architecture

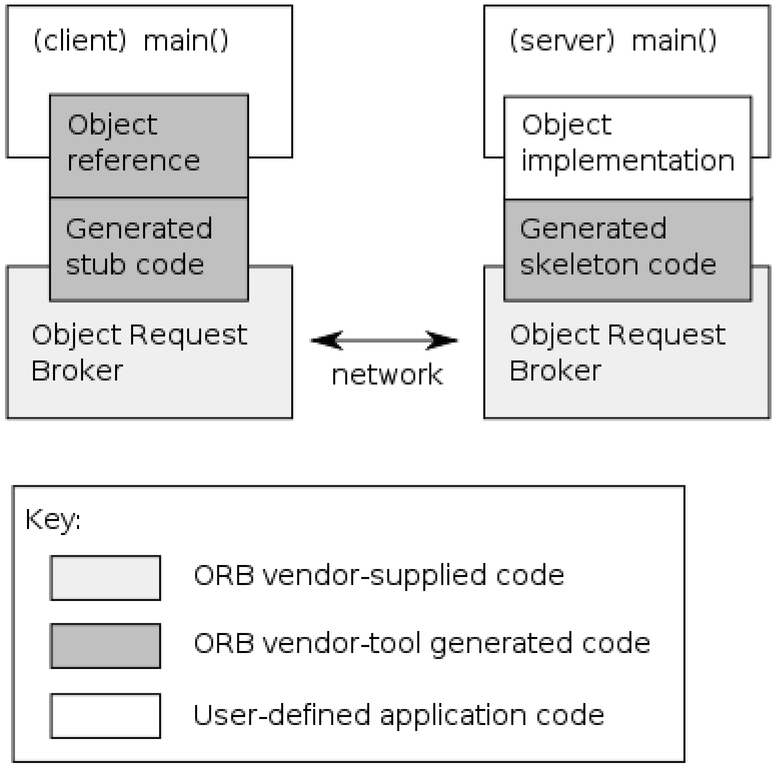


Image source from en.wikipedia.org

The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) designed to facilitate the communication of systems that are deployed on diverse platforms. CORBA enables collaboration between systems on different operating systems, programming languages, and computing hardware. CORBA uses an object-oriented model although the systems that use the CORBA do not have to be object-oriented. CORBA is an example of the distributed object paradigm.

The Java code implementation for the MultiThreadServer.java class

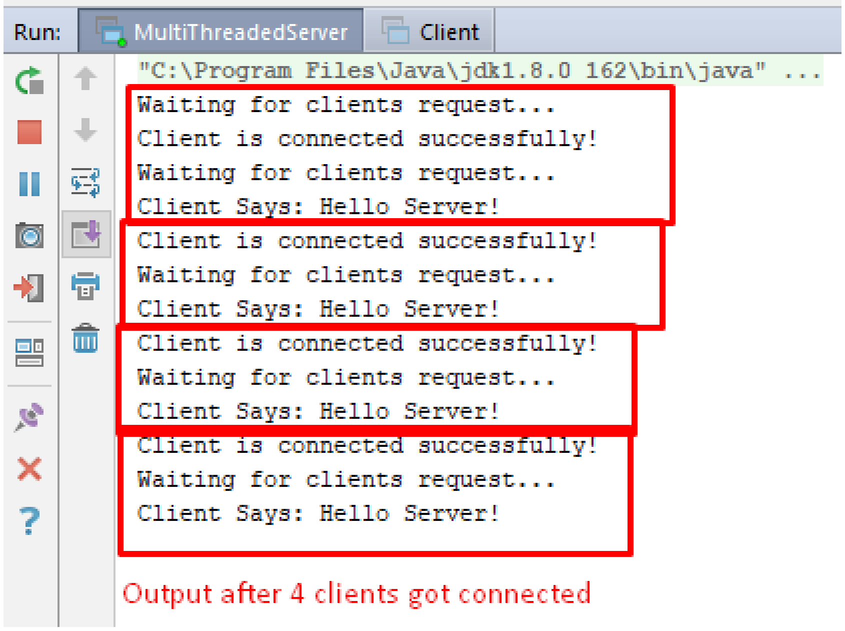
```
/*
 ** Author: Baftjar & Erli Reçi
 ** created on 1/10/2020 inside the package - Exercises.Exercise1
 **
 */

package Exercises.Exercise1;

import org.omg.CORBA.portable.UnknownException;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;

public class MultiThreadedServer {
    public static void main(String[] args) {
        try {
            //We assign a socked of server and an port no. with
ipd address
            //in this case 'localhost' ip address
            ServerSocket serverSocket = new ServerSocket(9090);
            while (true) {
                System.out.println("Waiting for clients
request...");
                //it accept request from clients
                Socket clientSocket = serverSocket.accept();
                //confirm client connection
                System.out.println("Client is connected
successfully!");
                ClientThread clientThread = new
ClientThread(clientSocket);
                clientThread.start();

            }
            //avoiding errors when creating the sockets
        } catch (Exception ex) {
            System.out.println("Other exceptions " +
ex.toString());
        }
    }
}

//client thread class
class ClientThread extends Thread {
    private Socket socket = null;

    public ClientThread(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try {
            PrintWriter pwOut = new
PrintWriter(socket.getOutputStream(), true);
            //Server writing to the client
            pwOut.println("Hello Client!!!");
            BufferedReader inputBufferedReader = new
BufferedReader(new InputStreamReader(socket.getInputStream()));
            //read from client
            String clientInput = inputBufferedReader.readLine();
            System.out.println(clientInput);
            //closing services after finishing
            inputBufferedReader.close();
            pwOut.close();
            socket.close();
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }
}
```

The Java code implementation for the Client.java class

```
/*
 ** Author: Baftjar Tabaku & Erli Reçi
 ** created on 1/10/2020 inside the package - Exercises.Exercise1
 **
 */

package Exercises.Exercise1;

import org.omg.CORBA.portable.UnknownException;

import java.io.*;
import java.net.InetAddress;
import java.net.Socket;

public class Client {

    public static void main(String[] args) {
        try {
            //We assign a socked of server and an port no. with ipd
address
            //in this case 'localhost' ip address
            //Passing the ip address of the server as the first
parameter
            InetAddress serverAddress =
InetAddress.getByName("localhost");
            Socket socket = new Socket(serverAddress, 9090);

            //input buffer reader for messages between client and
server
            //In this case client is waiting for the message from
the server
            //after sending the request connection
            PrintWriter pwOut = new
PrintWriter(socket.getOutputStream(), true);
            BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            //Printing message of the server
            System.out.println("Server says: "+input.readLine());
            //After receiving message , reply a message to the
server again
            pwOut.println("Client Says: Hello Server!");
            //closing all the services after finishing
            input.close();
            pwOut.close();
            socket.close();

            //avoiding errors when creating the sockets
        } catch (UnknownException e1) {
            System.out.println("Unknown host exception " +
e1.toString());
```

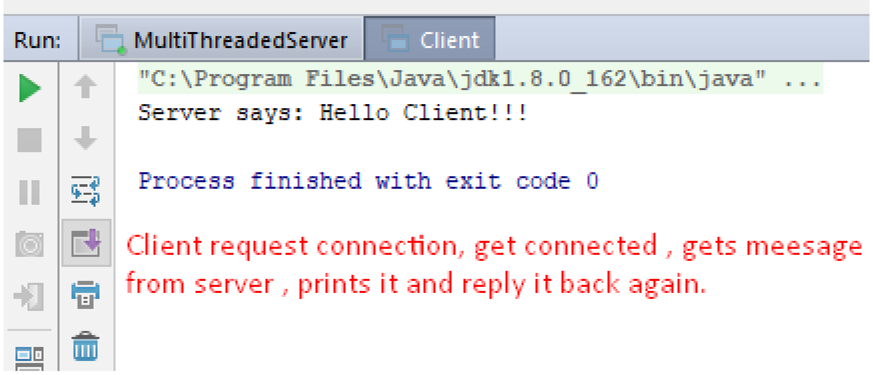
```
        } catch (IllegalArgumentException e3) {
            System.out.println("Illegal Argument Exception " +
e3.toString());
        } catch (IOException e2) {
            System.out.println("IOException " + e2.toString());
        } catch (Exception e4) {
            System.out.println("Other exceptions " +
e4.toString());
        }
    }
}
```

The output results from execution of multithreading server and client class:



```
"C:\Program Files\Java\jdk1.8.0 162\bin\java" ...
Waiting for clients request...
Client is connected successfully!
Waiting for clients request...
Client Says: Hello Server!
Client is connected successfully!
Waiting for clients request...
Client Says: Hello Server!
Client is connected successfully!
Waiting for clients request...
Client Says: Hello Server!
Client is connected successfully!
Waiting for clients request...
Client Says: Hello Server!
```

Output after 4 clients got connected



```
"C:\Program Files\Java\jdk1.8.0_162\bin\java" ...
Server says: Hello Client!!!

Process finished with exit code 0
```

Client request connection, get connected , gets meesage from server , prints it and reply it back again.

2. Implementation of Deadlock through Simulation

The **Deadlock** is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

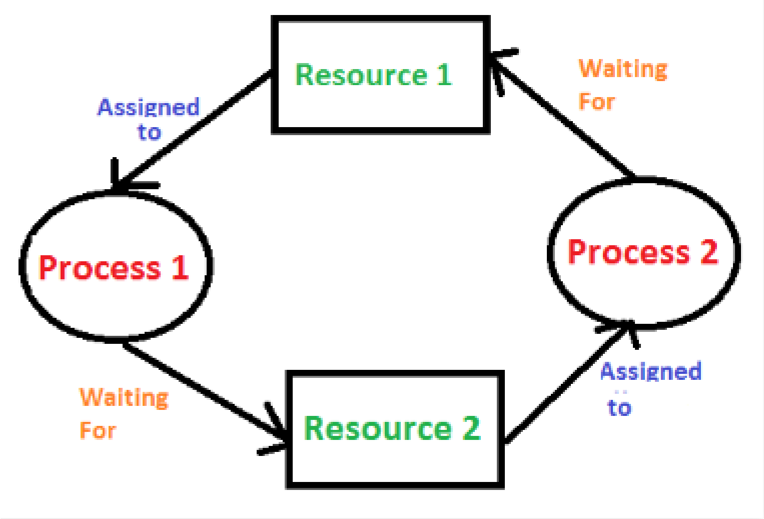


image from https://www.geeksforgeeks.org

Deadlock can arise if following four conditions hold simultaneously (Necessary Conditions)

Mutual Exclusion: One or more than one resource are non-sharable (Only one process can use at a time)

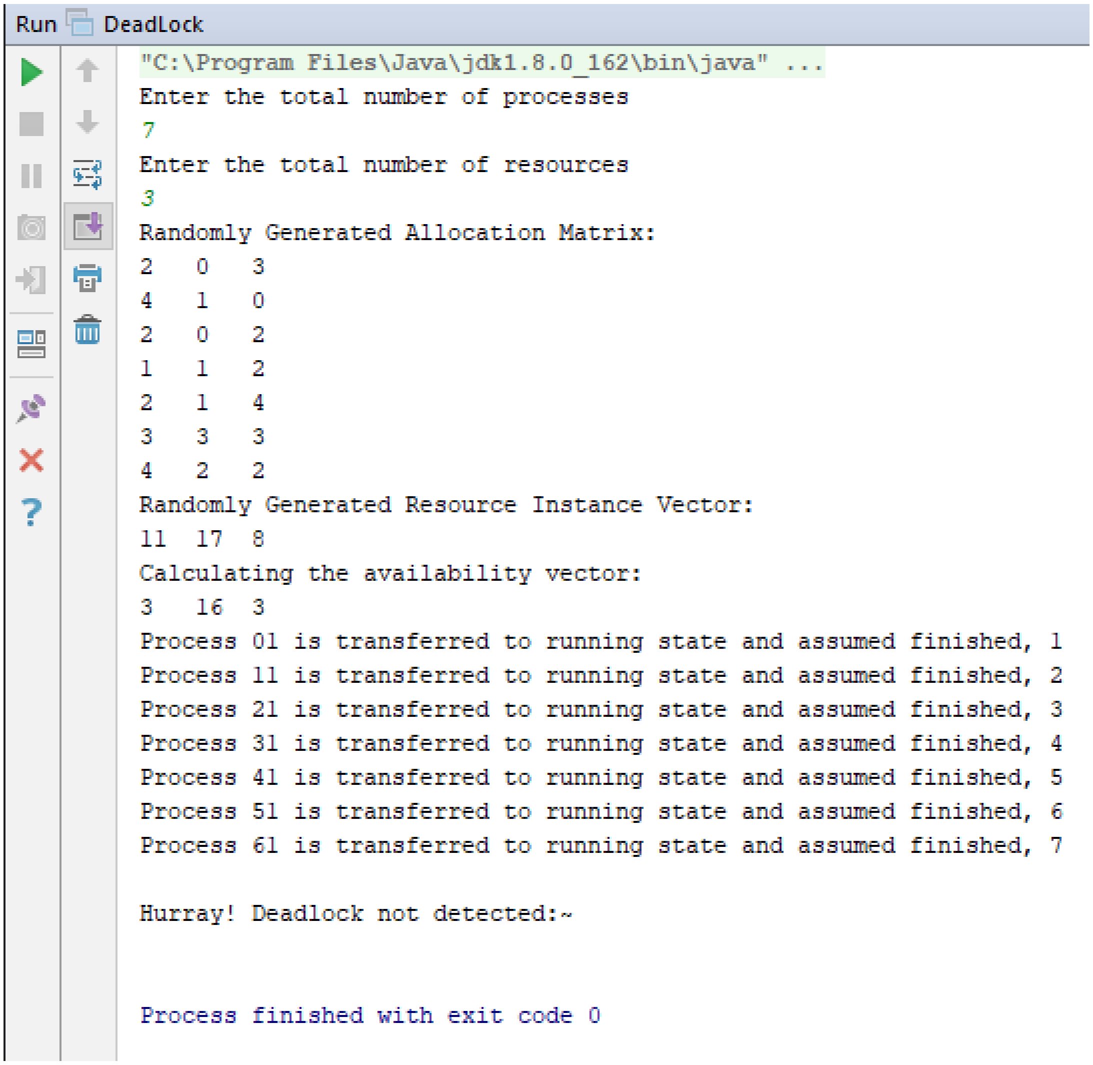
Hold and Wait: A process is holding at least one resource and waiting for resources.

No Preemption: A resource cannot be taken from a process unless the process releases the resource.

Circular Wait: A set of processes are waiting for each other in circular form.

Output of the running program implemented in DeadLock.java:

```
/*
 ** Author: Baftjar Tabaku & Erli Reçi
 ** created on 1/10/2020 inside the package - Exercises.Exercise2
 **
 */

package Exercises.Exercise2;

import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;

public class DeadLock {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Random random = new Random();

        int[][] alloc = new int[10][10];
        int[][] req = new int[10][10];
        int[] ins = new int[10];
        int[] avail = new int[10];
        boolean flag = false;
        int totalProcesses, totalResources, i, j, count;
        int[] tmp = new int[10];
        //Boolean array initialised with false
        boolean[] finish = new boolean[10];
        Arrays.fill(finish, Boolean.FALSE);
//      inputting from user
        //total number of processes
        System.out.println("Enter the total number of processes");
        totalProcesses = sc.nextInt();
        //total number of resources
        System.out.println("Enter the total number of resources");
        totalResources = sc.nextInt();

        //matrix initialised with random values
        System.out.println("Randomly Generated Allocation 
Matrix:");

        for (i = 0; i < totalProcesses; i++) {
            for (j = 0; j < totalResources; j++) {
                alloc[i][j] = random.nextInt(5);
                System.out.print(alloc[i][j] + "\t");
            }
            System.out.println();
        }

        //Randomly Generated Resource Instance Vector
        System.out.println("Randomly Generated Resource Instance 
Vector:");

        for (i = 0; i < totalResources; i++) {
            ins[i] = random.nextInt(10 + totalResources + 
totalProcesses);
            System.out.print(ins[i] + "\t");
        }

        //To calculate the resource availability vector
        System.out.println("\nCalculating the availability 
vector:");
        for (i = 0; i < totalProcesses; i++) {
            for (j = 0; j < totalResources; j++) {
                //sum calculated column wise for allocation matrix
                tmp[i] += alloc[j][i];
            }
        }
        //printing result of availability vector
        for (i = 0; i < totalResources; i++) {
            avail[i] = ins[i] - tmp[i];
            System.out.print(avail[i] + "\t");
        }
        System.out.println();

        //Main logic part
        count = 10;
        while (count > 0) {
            //if finish array has all true's (all processes to
running state)
            //deadlock not detected and loop stops!
            for (i = 0; i < totalProcesses; i++) {
                count = 0;
                //to see whenever proccesses can be allocated to
any blocked  process
                if (finish[i] == false) {
                    for (j = 0; j < totalResources; j++) {

                        if (req[i][j] <= avail[j]) {
                            count++;
                        }
                        flag = false;
                        if (count == totalResources) {

                            for (j = 0; j < totalResources; j++) {
                                avail[j] += alloc[i][j];
                                //Allocated resources are released
and added to available!
                            }
                            finish[i] = true;
                            System.out.println("Process " + i + 1 +
" is transferred to running state and assumed finished, " + (i +
1));
                        } else {
                            flag = true;
                        }
                    }
                    count = 0;
                    for (j = 0; j < totalResources; j++) {
                        if (finish[j]) {
                            count++;
                        }
                    }
                }
            }

            for (i = 0; i < totalProcesses; i++) {
                if (!finish[i]) {
                    System.out.println("Oops! Deadlock detected an
causing process is: proccess " + (i + 1));
                    break;
                }
            }
        }

        if (i - 1 >= 0) {
            i = i - 1;
        }
        if ((finish[i])) {
            System.out.println("\nHurray! Deadlock not
detected:~\n");
        }

    }
}
```



```
"C:\Program Files\Java\jdk1.8.0_162\bin\java" ...
Enter the total number of processes
7
Enter the total number of resources
3
Randomly Generated Allocation Matrix:
2   0   3
4   1   0
2   0   2
1   1   2
2   1   4
3   3   3
4   2   2
Randomly Generated Resource Instance Vector:
11  17  8
Calculating the availability vector:
3   16  3
Process 01 is transferred to running state and assumed finished, 1
Process 11 is transferred to running state and assumed finished, 2
Process 21 is transferred to running state and assumed finished, 3
Process 31 is transferred to running state and assumed finished, 4
Process 41 is transferred to running state and assumed finished, 5
Process 51 is transferred to running state and assumed finished, 6
Process 61 is transferred to running state and assumed finished, 7

Hurray! Deadlock not detected:~


Process finished with exit code 0
```

3. Study of 3 tier client architecture

Objective: To revise concept the standard 3-tier architecture consists of presentation and application logic in the client, application and business logic in a middle tier application server, and data managed by database servers in the 3 tier.

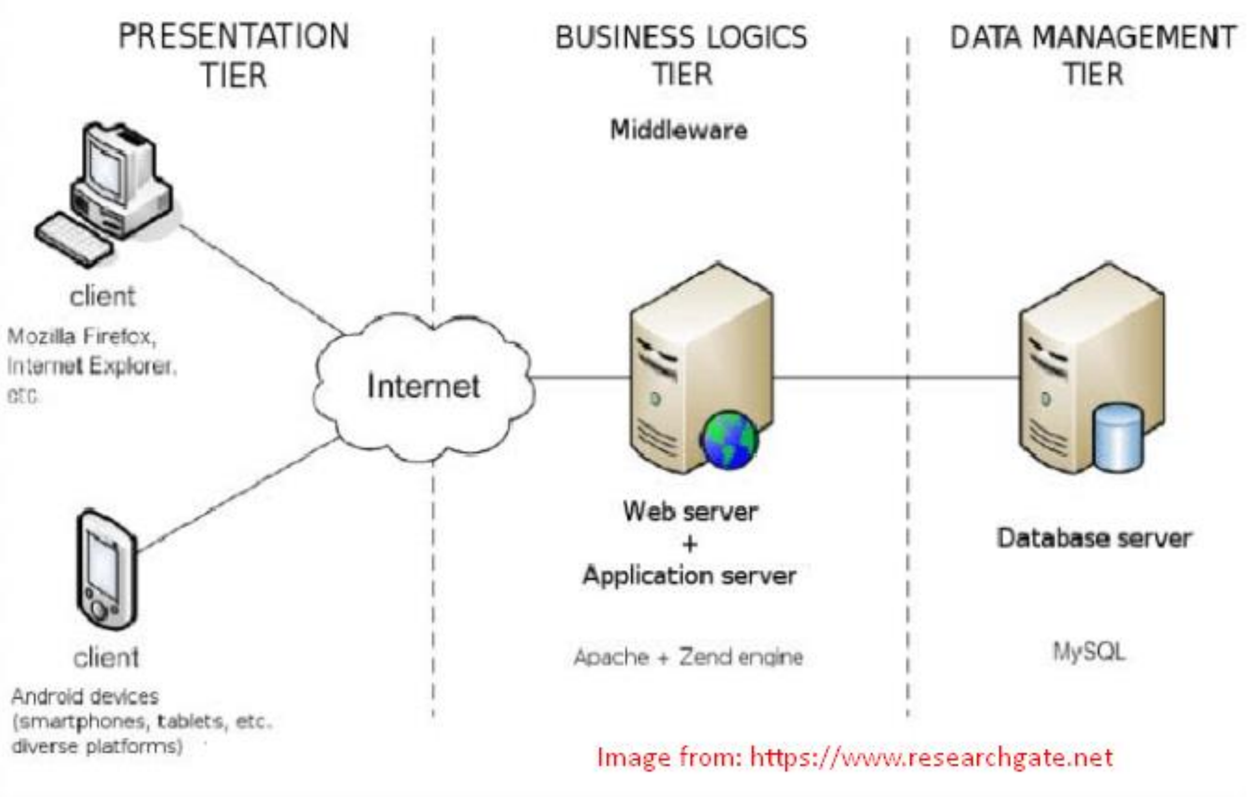
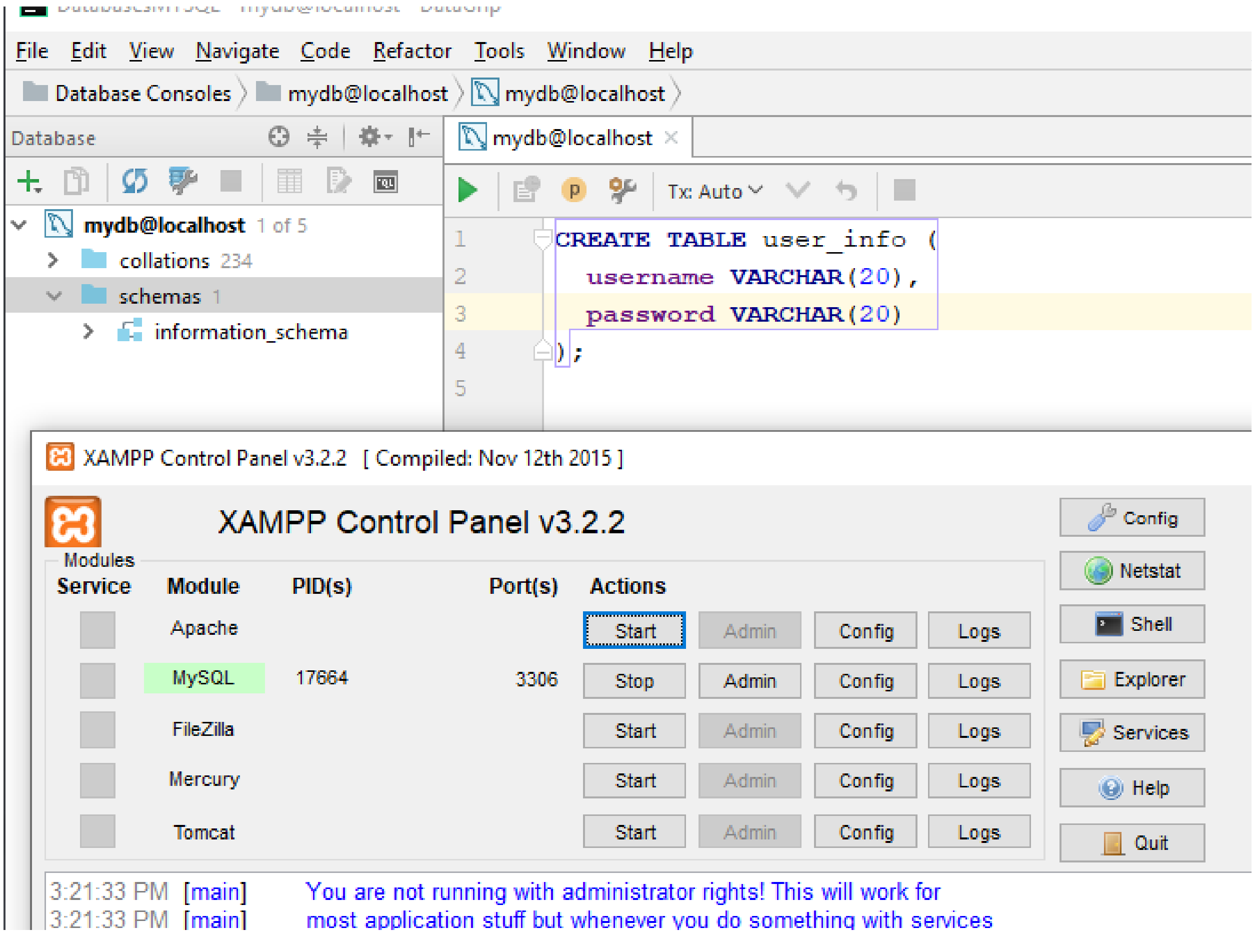

PRESENTATION TIER
BUSINESS LOGICS TIER
DATA MANAGEMENT TIER
Middleware
client
Mozilla Firefox, Internet Explorer, etc.
Internet
client
Android devices (smartphones, tablets, etc. diverse platforms)
Web server + Application server
Apache + Zend engine
Database server
MySQL

Image from: https://www.researchgate.net

The 3Tier architecture provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application. A three-tier architecture is typically composed of a *presentation* tier, a *domain logic* tier, and a *data storage* tier. In our solution will be included 3 java classes, each of them will be connected between them using local host and port number, the client with server and server with database management tier. Proceeding furthermore to the solution we need to create a small database MYSQL model.



```
CREATE TABLE user_info (
  username VARCHAR(20),
  password VARCHAR(20)
);
```

XAMPP Control Panel v3.2.2 [Compiled: Nov 12th 2015]

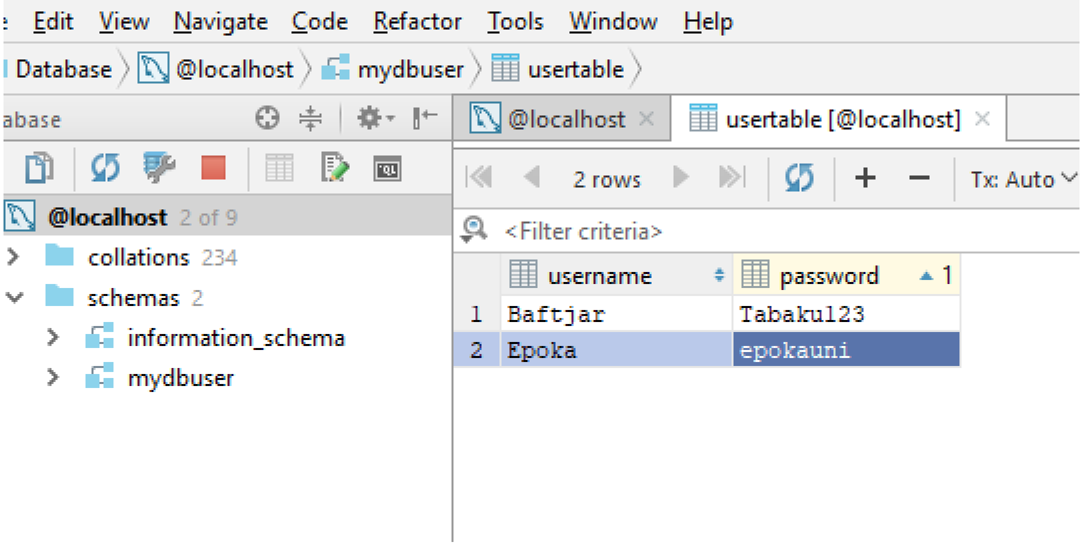
XAMPP Control Panel v3.2.2

Modules

Service	Module	PID(s)	Port(s)	Actions			
	Apache			Start	Admin	Config	Logs
	MySQL	17664	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
3:21:33 PM  [main]    You are not running with administrator rights! This will work for
3:21:33 PM  [main]    most application stuff but whenever you do something with services
```

Creating a simple MYSQL database where the user client will ask the server to login by adding some information like username and password, then server tier will ask the database server tier if the user exists or not in the database , to allow the user to login or give the ‘Welcome!!!’ message. By simulating a small database table with only two records , that presents user and password:



	username	password
1	Baftjar	Tabaku123
2	Epoka	epokauni

First of all is the client, it will get a message, a username, from a user and using ‘Localhost’ it will send it to the server, then server using another local host connection on port 9091 with the server database and himself, using MYSQL database in java it will check if the following user exist and print a positive answer, this will fulfil the 3tier ideology.

ClientTier.java :

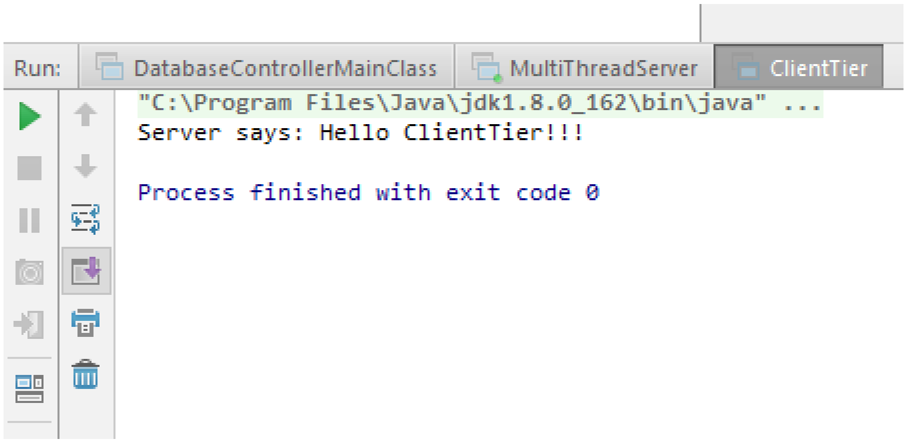
```
/*
 ** Author: Baftjar Tabaku & Erli Reçi
 ** created on 1/10/2020 inside the package - Exercise3_3TierArchitecture
 **
 */

package Exercise3_3TierArchitecture;

import org.omg.CORBA.portable.UnknownException;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;

public class ClientTier {
    public static void main(String[] args) {
        try {

            InetAddress serverAddress = InetAddress.getByName("localhost");
            Socket socket = new Socket(serverAddress, 9090);

            PrintWriter pwOut = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            //Printing message of the server
            System.out.println("Server says: " + input.readLine());
            //After receiving message , reply a message to the server again
            pwOut.println("Artan");

            input.close();
            pwOut.close();
            socket.close();
        } catch (Exception e4) {
            System.out.println("Other exceptions " + e4.toString());
        }
    }

}
```

MultiThreadServer.java, in this part after establishing the connection with the client, it will take input from client, and send it to the database controller java class.

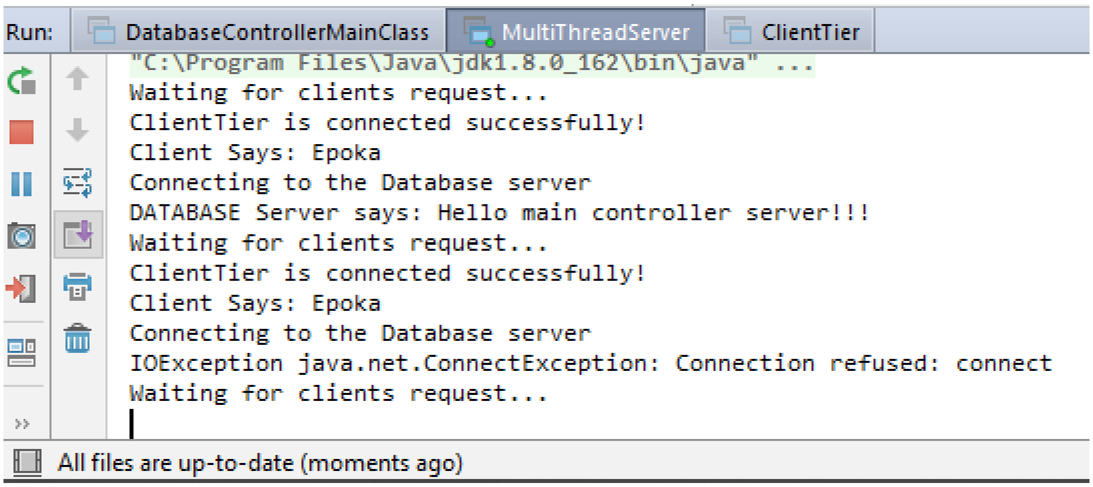
```
/*
 ** Author: Baftjar Tabaku & Erli Reçi
 ** created on 1/10/2020 inside the package - Exercise3_3TierArchitecture
 **
 */

package Exercise3_3TierArchitecture;

import org.omg.CORBA.portable.UnknownException;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;

public class MultiThreadServer {
    public static void main(String[] args) {
        try {

            //We assign a socked of server and an port no. with ipd address
            //in this case 'localhost' ip address
            ServerSocket serverSocket = new ServerSocket(9090);
            while (true) {
                System.out.println("Waiting for clients request...");
                //it accept request from clients
                Socket clientSocket = serverSocket.accept();
                //confirm client connection
                System.out.println("ClientTier is connected successfully!");
                ClientThread clientThread = new ClientThread(clientSocket);
                clientThread.run();

            }
            //avoiding errors when creating the sockets
        } catch (Exception ex) {
            System.out.println("Other exceptions " + ex.toString());
        }
    }
}


//client thread class
class ClientThread extends Thread {
    private Socket socket = null;

    public ClientThread(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try {
            PrintWriter printOut = new PrintWriter(socket.getOutputStream(), true);
            //Server writing to the client
            printOut.println("Hello ClientTier!!!");
            BufferedReader inputBufferedReader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

            //read from client
            String inputFromClient = inputBufferedReader.readLine();
            System.out.println("Client Says: "+inputFromClient);

            //Here it should contact the database class
            checkUser(inputFromClient);

            //closing services after finishing
            inputBufferedReader.close();
            printOut.close();
            socket.close();

        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }

    /*Connect to the database server to check the user information*/
    public void checkUser(String information) {
        System.out.println("Connecting to the Database server");
        try {

            InetAddress serverAddress = InetAddress.getByName("localhost");
            Socket socket = new Socket(serverAddress, 9091);//port of database server is
different
            PrintWriter pwOut = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

            //Printing message of the server
            System.out.println("DATABASE Server says: " + input.readLine());
            //After receiving message , reply a message to the server again, sending our
information to database server
            pwOut.println(information);
            //closing all the services after finishing
            input.close();
            pwOut.close();
            socket.close();

            //avoiding errors when creating the sockets
        } catch (UnknownException e1) {
            System.out.println("Unknown host exception " + e1.toString());
        } catch (IllegalArgumentException e3) {
            System.out.println("Illegal Argument Exception " + e3.toString());
        } catch (IOException e2) {
            System.out.println("IOException " + e2.toString());
        } catch (Exception e4) {
            System.out.println("Other exceptions " + e4.toString());
        }
    }

}
```

DatabaseManagementTier.java, it will be connected to the database and compare the input name with the one that is actually in the Database.

```
/*
 ** Author: Baftjar Tabaku & Erli Reçi
 ** created on 1/10/2020 inside the package - Exercise3_3TierArchitecture
 **
 */

package Exercise3_3TierArchitecture;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.sql.*;

public class DatabaseManagementTier {
    private String JDBC_Driver = "com.mysql.cj.jdbc.Driver";
    private String DATABASE_URL = "jdbc:mysql://localhost:3306/mydbuser";
    private String USER = "root";
    private String PASSWORD = "";
    private boolean isAnUser = false;

    public boolean readDatabase(String accountName) {
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
        try {

//            Set the class name , so register the JDBC driver
            Class.forName(JDBC_Driver);
            connection = DriverManager.getConnection(DATABASE_URL, USER, PASSWORD);
            statement = connection.createStatement();

            /*Difference between JDBC and JPA that manipulate java objects instead of sql
tables(JDBC) */
//            Native sql queries
            String sqlCommand = "Select * from usertable";
//            Result set - all the entries in DB
            resultSet = statement.executeQuery(sqlCommand);

            while (resultSet.next()) {
                String name = resultSet.getString("username");
                //if exist in Database then return true
                if (name.equals(accountName)) {
                    return true;
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                resultSet.close();//we must close the result
                statement.close();
                connection.close();

            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return false;
    }
}

class DatabaseControllerMainClass {
    public static void main(String[] args) throws IOException {
        DatabaseManagementTier dbManagementTier = new DatabaseManagementTier();

        ServerSocket serverSocket = new ServerSocket(9091);
        System.out.println("Database Server Started, waiting for main servers to request a
check of user...");

        Socket socket = serverSocket.accept();
        //Sending message to the client
        PrintWriter pwOut = new PrintWriter(socket.getOutputStream(), true);
        pwOut.println("Hello main controller server!!! ");

        //reading from the client
        BufferedReader bfrInput = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        String clientInput = bfrInput.readLine();
//        check if user exist

        if(dbManagementTier.readDatabase(clientInput)){
System.out.println("User exists!!!");
    }else
        {
            System.out.println("User NOT exists!!!");
        }
        //closing stream and socket
        bfrInput.close();
        pwOut.close();
        socket.close();
    }
}
```

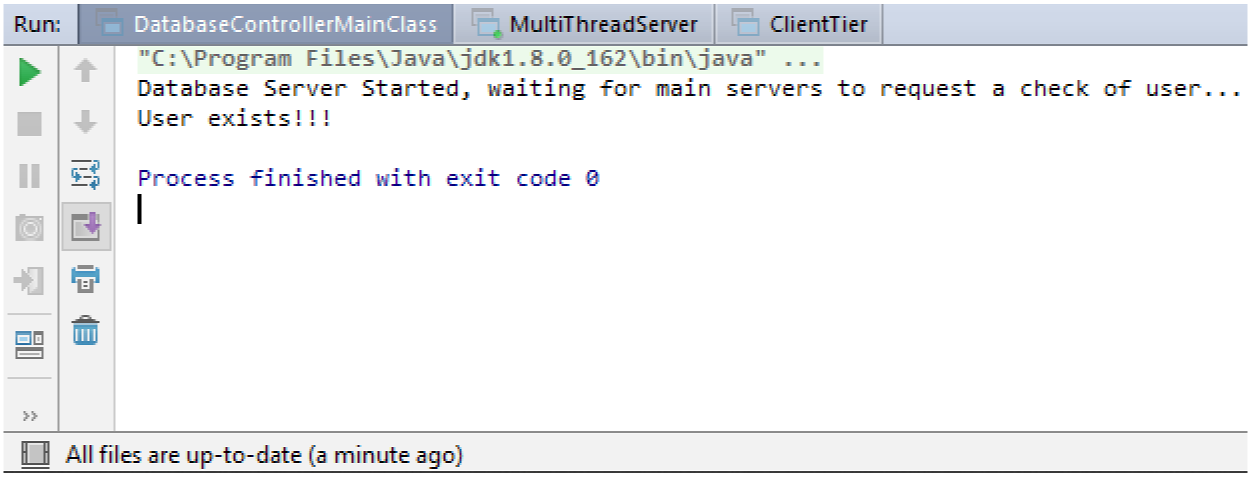


```
"C:\Program Files\Java\jdk1.8.0_162\bin\java" ...
Server says: Hello ClientTier!!!

Process finished with exit code 0
```



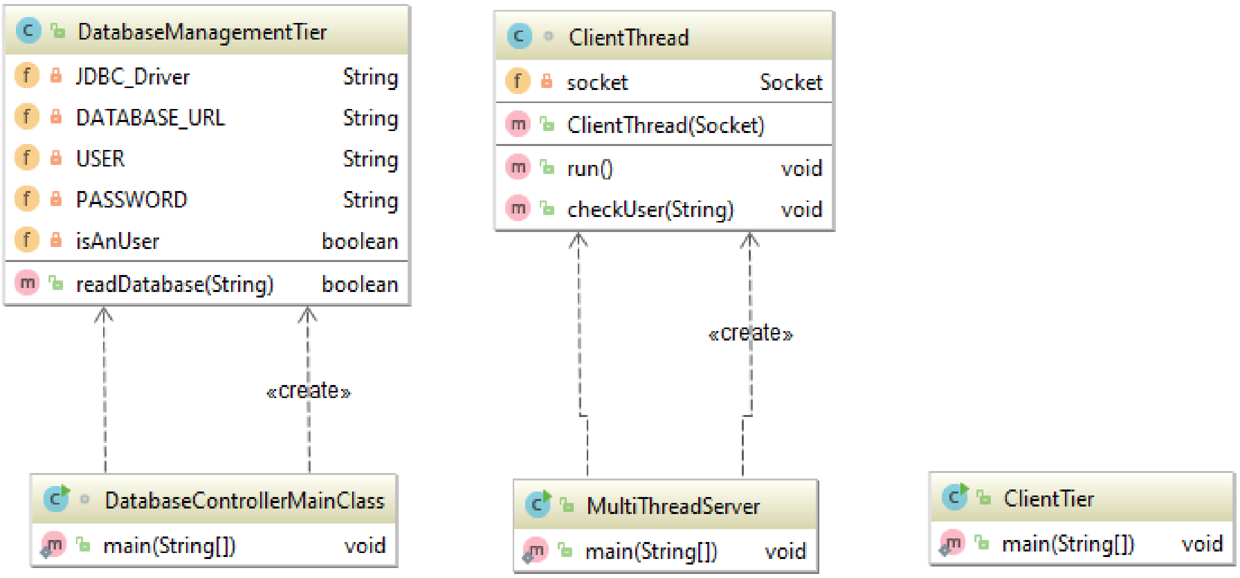
```
"C:\Program Files\Java\jdk1.8.0_162\bin\java" ...
Waiting for clients request...
ClientTier is connected successfully!
Client Says: Epoka
Connecting to the Database server
DATABASE Server says: Hello main controller server!!!
Waiting for clients request...
ClientTier is connected successfully!
Client Says: Epoka
Connecting to the Database server
IOException java.net.ConnectException: Connection refused: connect
Waiting for clients request...
```

All files are up-to-date (moments ago)



```
"C:\Program Files\Java\jdk1.8.0_162\bin\java" ...
Database Server Started, waiting for main servers to request a check of user...
User exists!!!

Process finished with exit code 0
```

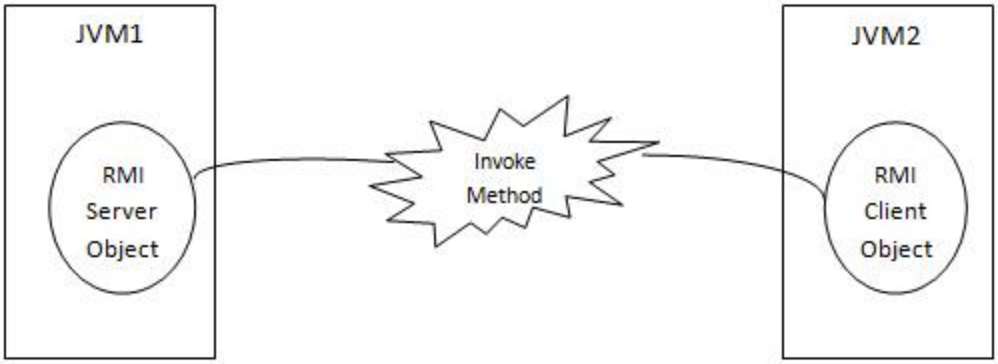
All files are up-to-date (a minute ago)



Java UML diagram representation for the following exercise.

4. Case study on Client and RMI Server

RMI is used to communicate between two running java applications on different JVM (Java Virtual Machines). The main motive behind RMI implementation is to invoke one method running on different JVM. The JVM Application, in which an invoked method is running out, is called RMI Server, where as the invoking application running on the different JVM is called RMI Client.



Remote Obj / RMI Server: It is an object that exposes methods which can be invoked by Remote Client using RMI.

The implementation and elements:

RMIRMI Client: It is an object that invokes remote methods on an RMI Server.

Stub: A stub is proxy that stands for RMI Server on client side and handles remote method invocation on behalf of RMI Client.

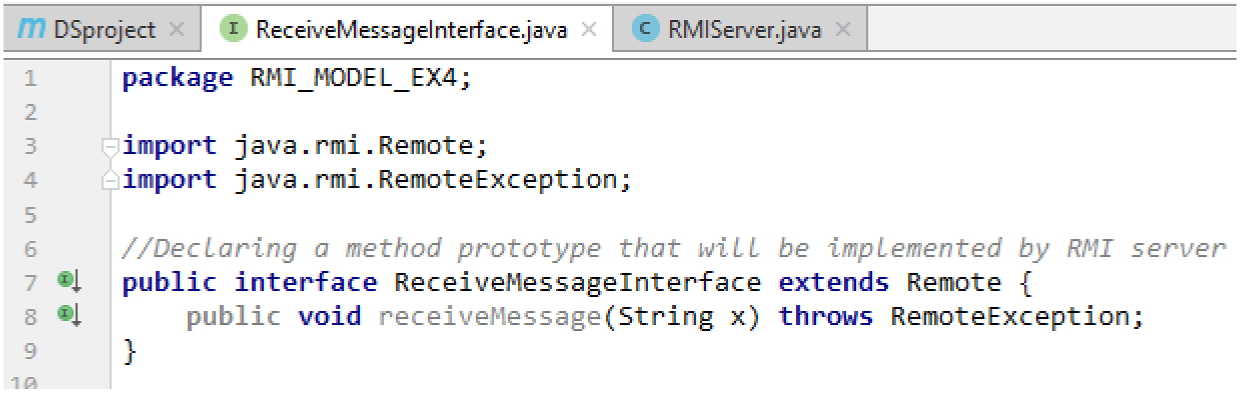
Skeleton: It is a proxy that stands for RMI client on server side and handles remote method invocation on RMI Server on behalf of client.

Registry Service: A Registry Service is an application that provides the facility of registration & lookup of Remote stub.

A Registry Service provides location transparency of Remote Object to RMI Client.

1. Defining the remote interface

The first thing to do is to create an interface which will provide the description of the methods that can be invoked by remote clients. This interface should extend the Remote interface and the method prototype within the interface should throw the RemoteException.



```java
package RMI_MODEL_EX4;

import java.rmi.Remote;
import java.rmi.RemoteException;

//Declaring a method prototype that will be implemented by RMI server
public interface ReceiveMessageInterface extends Remote {
    public void receiveMessage(String x) throws RemoteException;
}
```

```java
/*
 ** Author: Baftjar Tabaku & Erli Reçi
 ** created on 1/12/2020 inside the package - RMI_MODEL_EX4
 **
 */

package RMI_MODEL_EX4;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RecieveMessageInteface extends Remote {
    void receiveMessage(String x) throws RemoteException;

}
```

2. Implementing the ServerRMI.java class

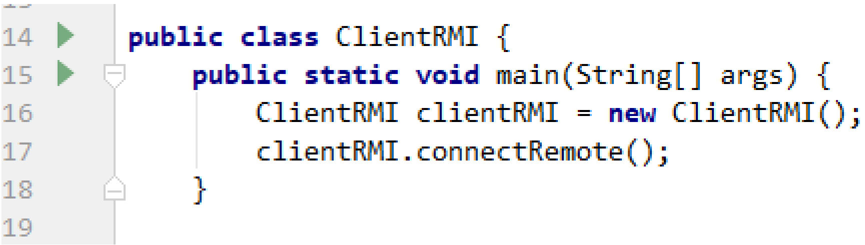
Server is simple and it overrides the interface method, then this method will be called in the main method, a registry will be created on port '4444' and there will be set an name and an object, then according to the possible errors the try and catch are set as following:

```java
/*
 ** Author: Baftjar Tabaku & Erli Reçi
 ** created on 1/12/2020 inside the package - RMI_MODEL_EX4
 **
 */

package RMI_MODEL_EX4;
```

```java
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class ServerRMI extends UnicastRemoteObject implements
RecieveMessageInteface {

    public ServerRMI() throws Exception {
        super();
    }

    @Override
    public void receiveMessage(String x) throws RemoteException {
        System.out.println("Server printing the message: " + x);
    }

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

        Registry reg = LocateRegistry.createRegistry(4444);
        try {
            reg.rebind("hi server", new ServerRMI());
            System.out.println("Server is ready..");

        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Exception"+e);
        }

    }
}
```

3. **Implementing the ClientRMI.java class**

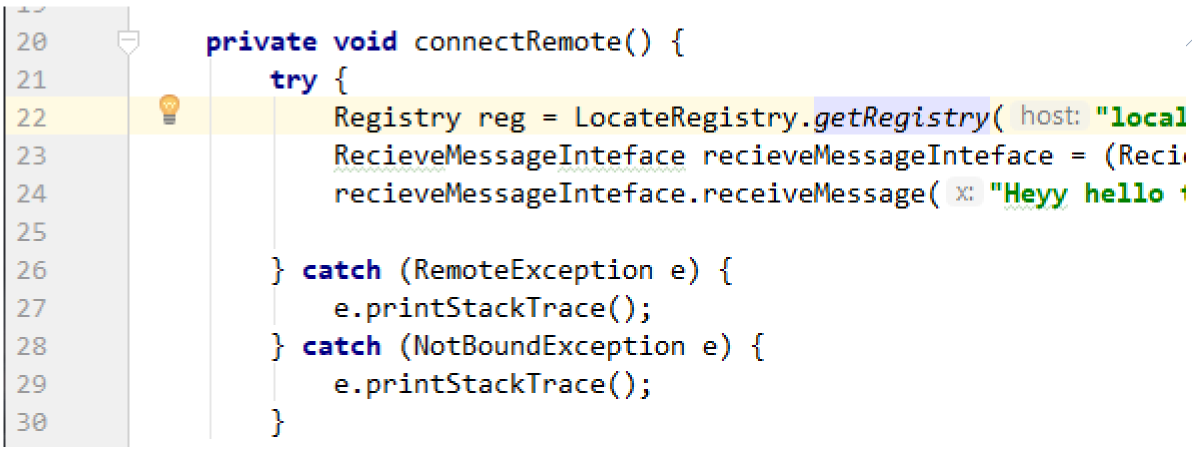
Client will get connected to the server, with the remote connection,



```java
public class ClientRMI {
    public static void main(String[] args) {
        ClientRMI clientRMI = new ClientRMI();
        clientRMI.connectRemote();
    }
```

Then according to the method to connect that would be the

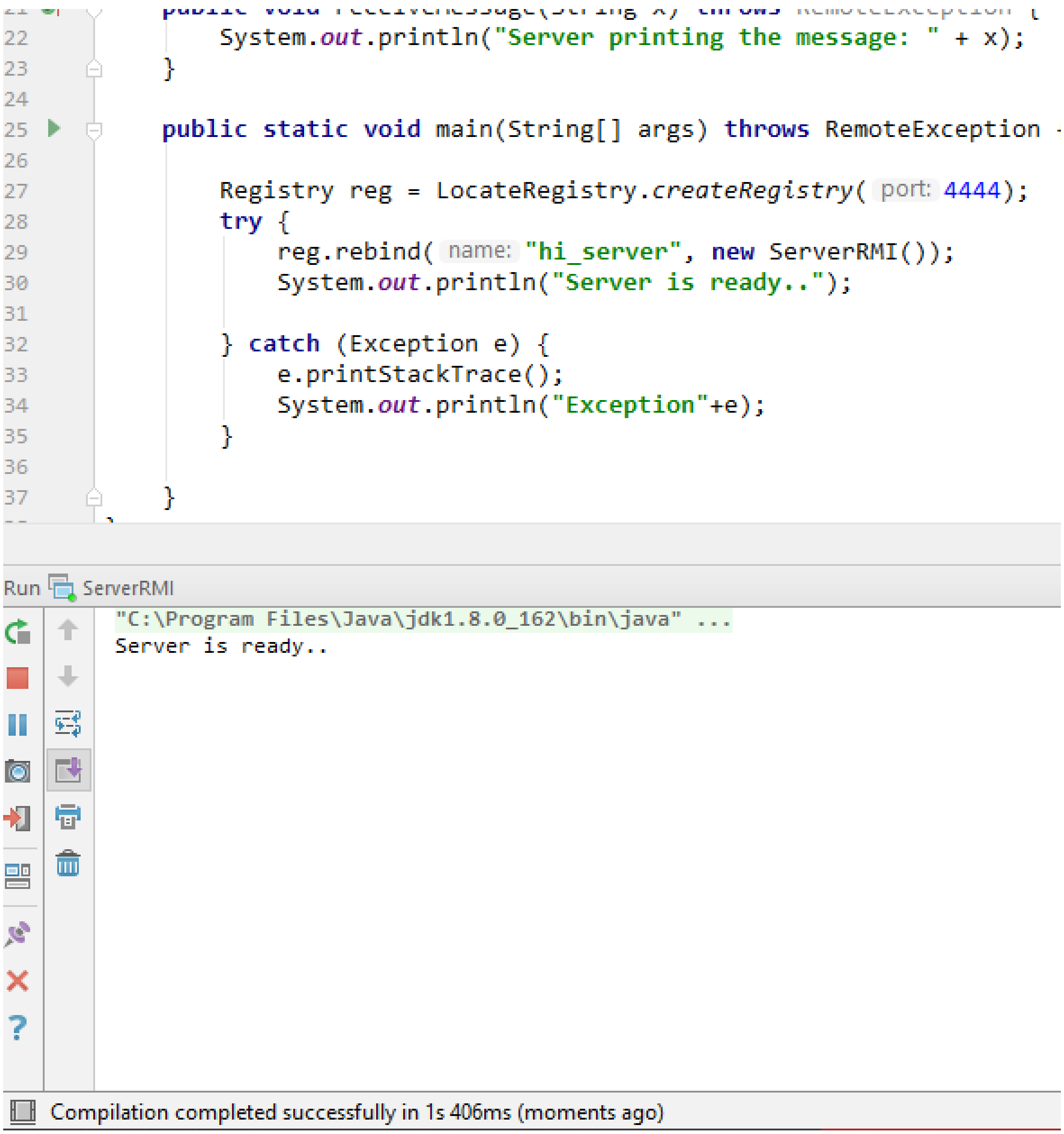
It will realize the full connection according to the current localhost and the port number and it will be connected to the registry, then it will create the interface RMI object and will lookup the current registry in the server according to the name. Then according to the interface created object it will call the method located in the server and print the result as following:



```
    private void connectRemote() {
        try {
            Registry reg = LocateRegistry.getRegistry( host: "local
            RecieveMessageInteface recieveMessageInteface = (Reci
            recieveMessageInteface.receiveMessage( x: "Heyy hello

        } catch (RemoteException e) {
            e.printStackTrace();
        } catch (NotBoundException e) {
            e.printStackTrace();
        }
```

```
/*
 ** Author: Baftjar Tabaku & Erli Reçi
 ** created on 1/12/2020 inside the package - RMI_MODEL_EX4
 **
 */

package RMI_MODEL_EX4;

import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ClientRMI {
    public static void main(String[] args) {
        ClientRMI clientRMI = new ClientRMI();
        clientRMI.connectRemote();
    }

    private void connectRemote() {
        try {
            Registry reg = LocateRegistry.getRegistry("localhost", 4444);
            RecieveMessageInteface recieveMessageInteface =
(RecieveMessageInteface) reg.lookup("hi_server");
            recieveMessageInteface.receiveMessage("Hey hello to all!!!");

        } catch (RemoteException e) {
            e.printStackTrace();
        } catch (NotBoundException e) {
            e.printStackTrace();
        }
    }
}
```

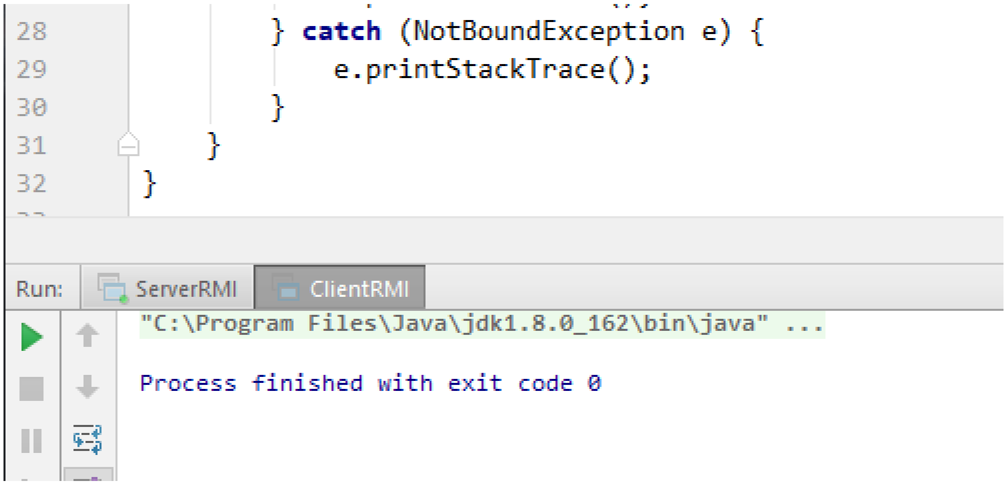
Server running:



```
    public void receiveMessage(String x) throws RemoteException {
        System.out.println("Server printing the message: " + x);
    }

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

        Registry reg = LocateRegistry.createRegistry( port: 4444);
        try {
            reg.rebind( name: "hi_server", new ServerRMI());
            System.out.println("Server is ready..");

        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Exception"+e);
        }

    }
}
```

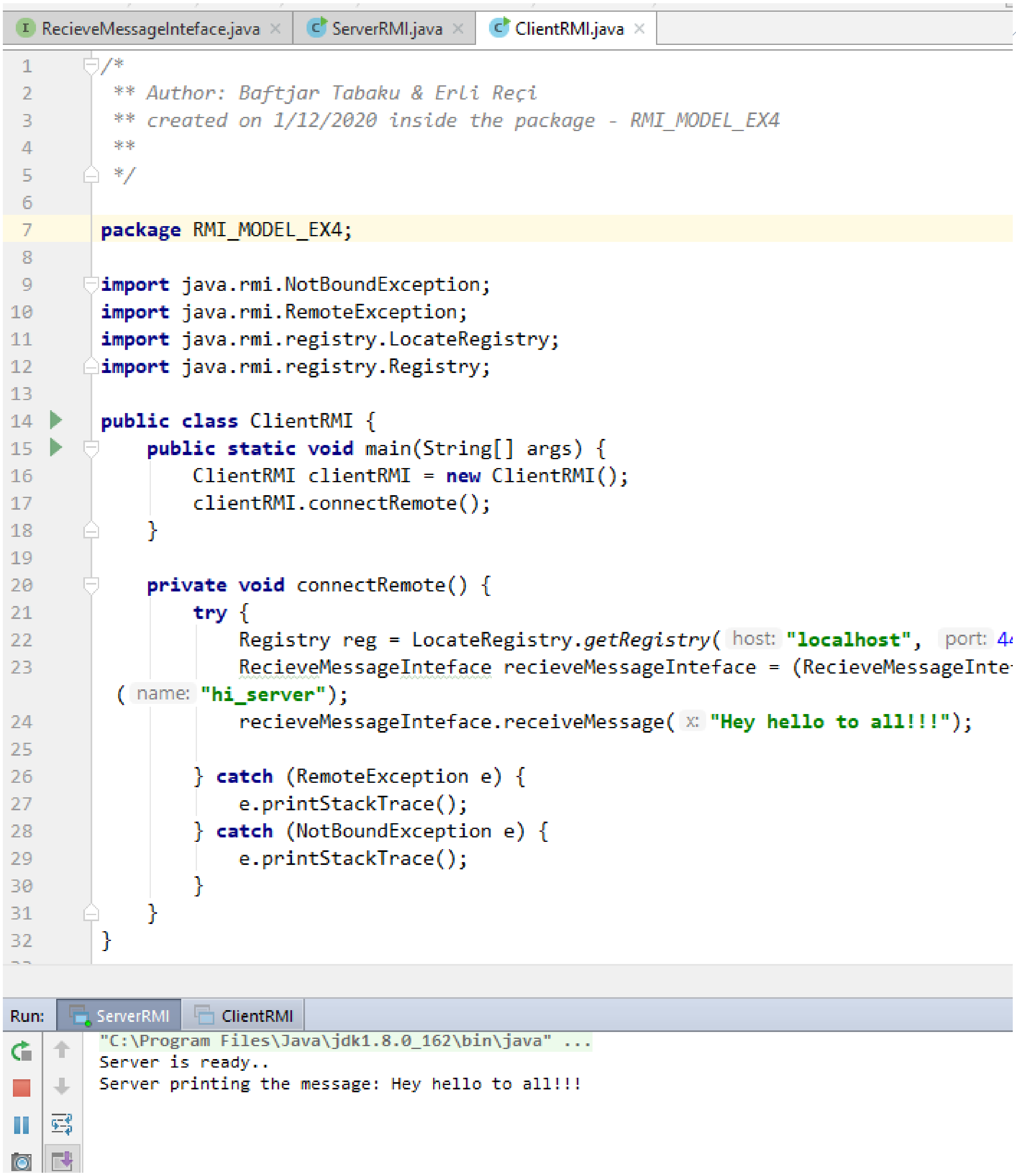
```
"C:\Program Files\Java\jdk1.8.0_162\bin\java" ...
Server is ready..
```

Compilation completed successfully in 1s 406ms (moments ago)



```
        } catch (NotBoundException e) {
            e.printStackTrace();
        }
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_162\bin\java" ...

Process finished with exit code 0
```



```java
/*
 ** Author: Baftjar Tabaku & Erli Reçi
 ** created on 1/12/2020 inside the package - RMI_MODEL_EX4
 **
 */

package RMI_MODEL_EX4;

import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ClientRMI {
    public static void main(String[] args) {
        ClientRMI clientRMI = new ClientRMI();
        clientRMI.connectRemote();
    }

    private void connectRemote() {
        try {
            Registry reg = LocateRegistry.getRegistry( host: "localhost",  port: 4
            RecieveMessageInteface recieveMessageInteface = (RecieveMessageInte
 ( name: "hi_server");
            recieveMessageInteface.receiveMessage( x: "Hey hello to all!!!");

        } catch (RemoteException e) {
            e.printStackTrace();
        } catch (NotBoundException e) {
            e.printStackTrace();
        }
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_162\bin\java" ...
Server is ready..
Server printing the message: Hey hello to all!!!
```

5. WAP to Implement an Election algorithm

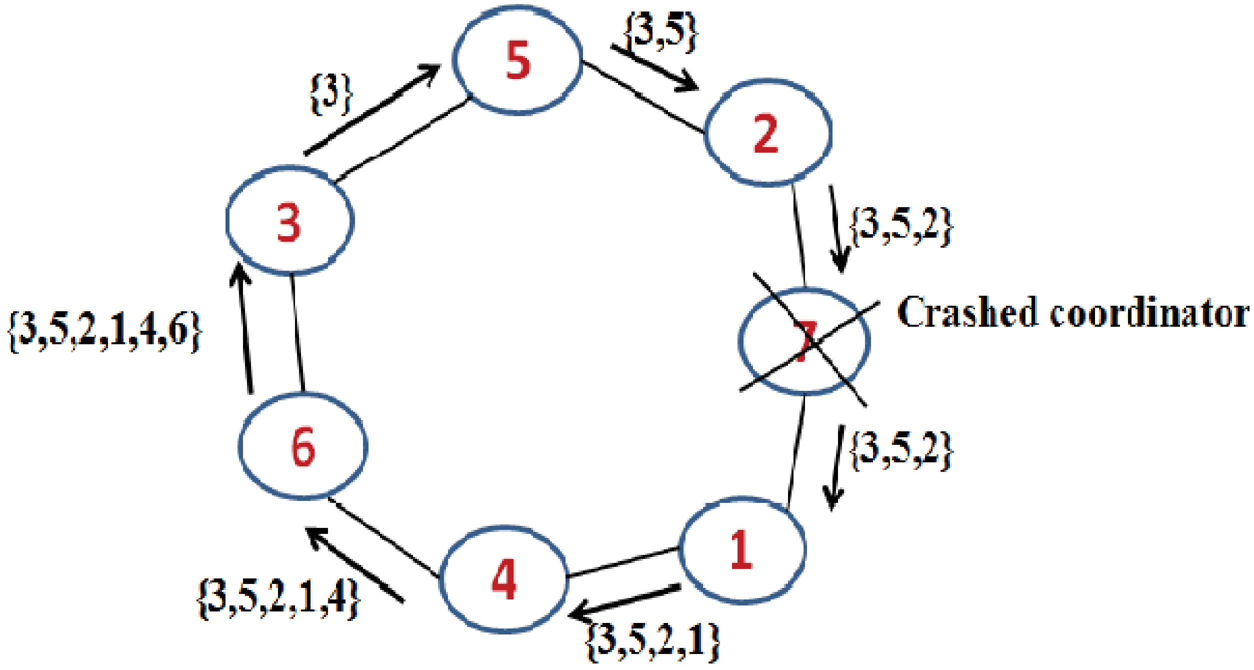
Distributed Algorithm is an algorithm that runs on a distributed system. Distributed system is a collection of independent computers that do not share their memory. Each processor has its own memory and they communicate via communication networks.

Election Algorithms:

Election algorithms choose a process from group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor. Election algorithm basically determines where a new copy of coordinator should be restarted.

Election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number. Then this number is sent to every active process in the distributed system.

ring topology.



The source code implemented in java

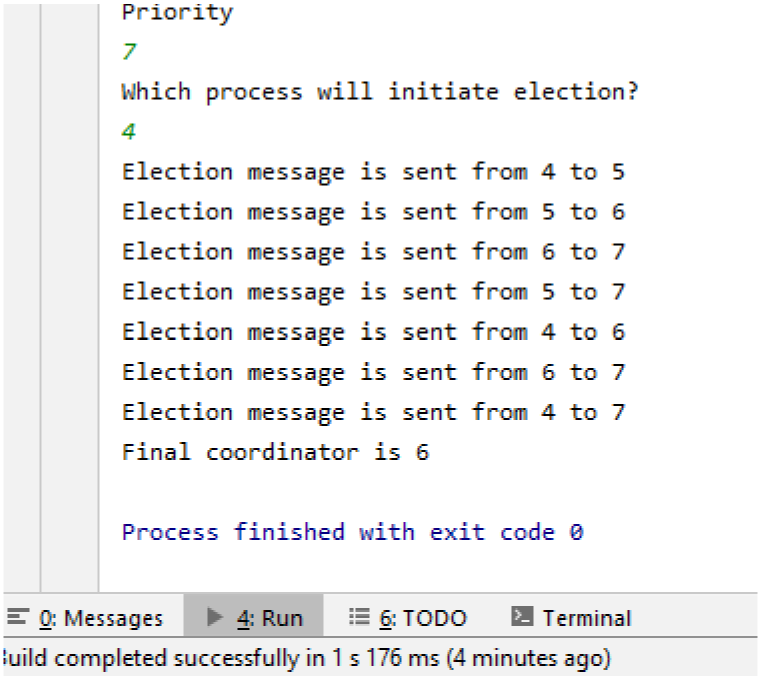
```java
package Exercise5_ELECTION_Algorithm;

import java.io.IOException;
import java.util.Scanner;

/**
 * Project Name: JavaTrainingExercisesProject
 * Created With: IntelliJ IDEA.
 * Author: Baftjar TABAKU & Erli Reçi
 * Date Created: 1/14/2020
 * Time Created: 1:07 PM
 **/
public class ElectionAlgorithm {
    static int n;
    static int pro[] = new int[100];
    static int sta[] = new int[100];
    static int co;

    public static void main(String args[]) throws IOException {
        System.out.println("Enter the number of process");
        Scanner in = new Scanner(System.in);
        n = in.nextInt();

        int i, j, k, l, m;

        for (i = 0; i < n; i++) {
            System.out.println("For process " + (i + 1) + ":");
            System.out.println("Status:");
            sta[i] = in.nextInt();
            System.out.println("Priority");
            pro[i] = in.nextInt();
        }

        System.out.println("Which process will initiate election?");
        int ele = in.nextInt();

        elect(ele);
        System.out.println("Final coordinator is " + co);
    }

    static void elect(int ele) {
        ele = ele - 1;
        co = ele + 1;
        for (int i = 0; i < n; i++) {
            if (pro[ele] < pro[i]) {
                System.out.println("Election message is sent from " + (ele + 1) + "
to " + (i + 1));
                if (sta[i] == 1)
                    elect(i + 1);
            }
        }
    }
}
```

The inputs and outputs according to their order of execution:

```
Enter the number of process
7
For process 1:
Status:
1
Priority
1
For process 2:
Status:
1
Priority
2
For process 3:
Status:
1
Priority
3
For process 4:
Status:
1
Priority
4
For process 5:
Status:
1
Priority
5
For process 6:
Status:
1
Priority
6
```

```
For process 7:
Status:
0
Priority
7
Which process will initiate election?
4
Election message is sent from 4 to 5
Election message is sent from 5 to 6
Election message is sent from 6 to 7
Election message is sent from 5 to 7
Election message is sent from 4 to 6
Election message is sent from 6 to 7
Election message is sent from 4 to 7
Final coordinator is 6
```

Parts of screenshot:



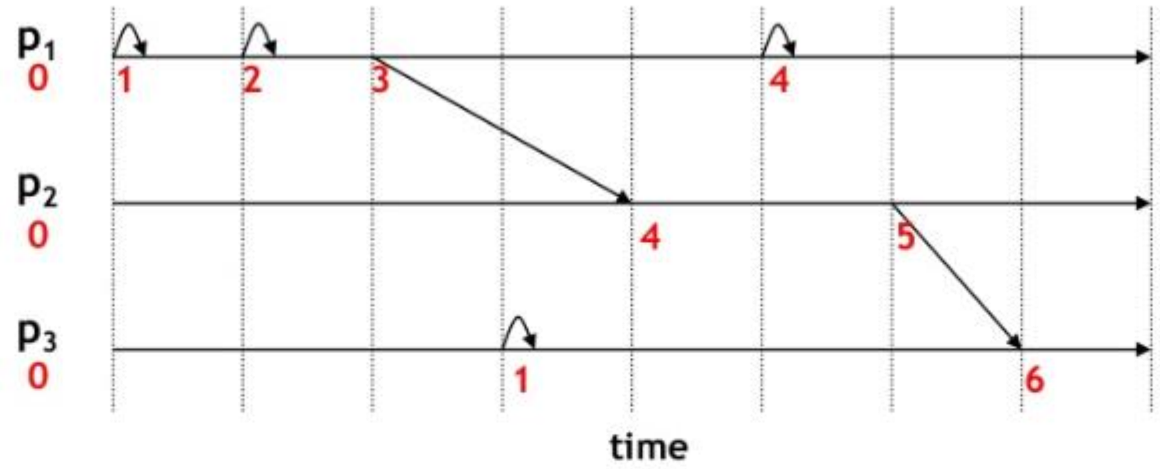
```
Priority
7
Which process will initiate election?
4
Election message is sent from 4 to 5
Election message is sent from 5 to 6
Election message is sent from 6 to 7
Election message is sent from 5 to 7
Election message is sent from 4 to 6
Election message is sent from 6 to 7
Election message is sent from 4 to 7
Final coordinator is 6

Process finished with exit code 0
```

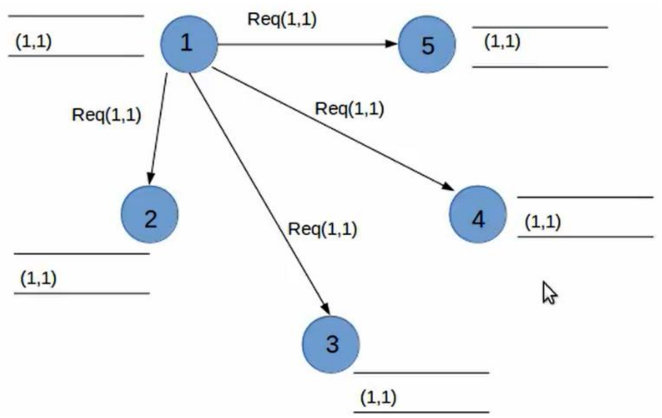
uild completed successfully in 1 s 176 ms (4 minutes ago)

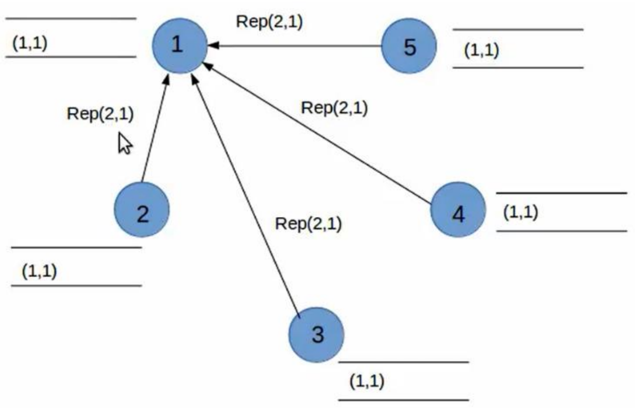
6. S/W (Solid works) Simulation for Clock Synchronization in Distributed Systems using Lamport's Algorithm

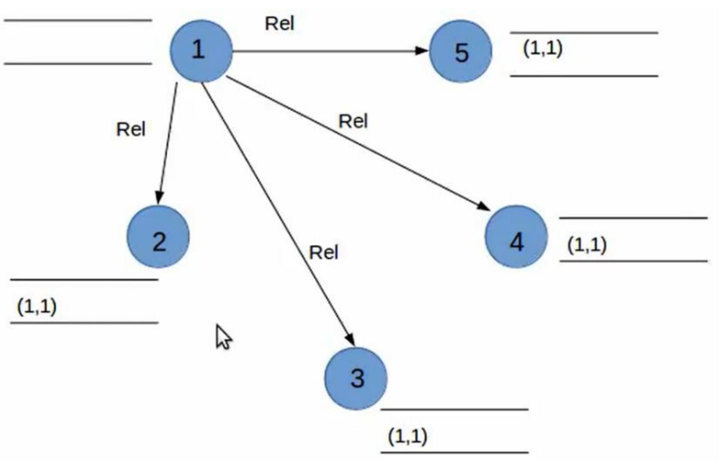
Lamport's Distributed Mutual Exclusion Algorithm is a permission-based algorithm proposed by Lamport as an illustration of his synchronization scheme for distributed systems.

In permission-based timestamp is used to order critical section requests and to resolve any conflict between requests.









The main components of Lamport Algorithm, **Request**, **Reply**, **Release**

Java Code implementation, Important: It was based from ‘tuvtran’ in GitHub

```
package Exercise6_LaportAlgorithm;

/**
 * Project Name: DSproject
 * Created With: IntelliJ IDEA.
 * Author: Baftjar TABAKU & Erli REÇI
 * Date Created: 1/16/2020
 * Time Created: 12:48 PM
 **/
public class Event {
    public int type;
    public long senderId;
    public long receiverId;
    public int localTime;
    public String content;

    public Event(int type, long senderId, long receiverId, String content) {
        this.type = type;
        this.senderId = senderId;
        this.receiverId = receiverId;
        this.content = content;
        this.localTime = 0;
    }

    public Event(int type, long senderId,
                 long receiverId, int localTime, String content) {
        this(type, senderId, receiverId, content);
        this.localTime = localTime;
    }
}


/**
 * Project Name: DSproject
 * Created With: IntelliJ IDEA.
 * Author: Baftjar TABAKU & Erli REÇI
 * Date Created: 1/16/2020
 * Time Created: 12:48 PM
 **/
package Exercise6_LaportAlgorithm;

import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.util.PriorityQueue;
import java.util.Random;

public class LamportClock extends Thread {
    private MulticastSocket sock;
    private InetAddress group;
    private int port;
    private PriorityQueue<Request> clockPQ;

    // local time of a process
```

```java
    private int time;

    // order of the process (viewed from the master's perspective)
    private int order;

    public LamportClock(InetAddress group, int port) throws Exception {
        this.group = group;
        this.port = port;

        // if we don't assign an order to a process
        this.order = -1;

        // set local time to random
        Random rand = new Random();
        // this.time = rand.nextInt(10);
        this.time = 0;

        // initialize the priority queue
        this.clockPQ = new PriorityQueue<Request>();

        sock = new MulticastSocket(port);
        sock.setTimeToLive(2);
        sock.joinGroup(group);
    }

    public LamportClock(InetAddress group, int port, int order) throws Exception {
        this(group, port);
        this.order = order;
    }

    public int getOrder() {
        return this.order;
    }
    public int getTime() {
        return this.time;
    }

    public int localEvent() {
        ++this.time;
        // System.out.println(this.getId() + " performing local event. local time is 
" + this.time);
        return this.time;
    }

    public int receivedEvent(long senderId, int receivedTime) {
        // System.out.println(this.getId() + " received message from "
        //     + senderId + ". local time is " + this.time);
        return this.time;
    }

    public int sendEvent(String msg) throws Exception {
        byte[] data = msg.getBytes();
        DatagramPacket d = new DatagramPacket(data, data.length, group, port);
        sock.send(d);
        return this.time;
    }

    public void updateTime(Event e) throws Exception {
        int type = e.type;
        switch (type) {
            // LOCAL EVENT
            case 0:
                this.localEvent();
                break;
            // SEND EVENT
            case 1: // extract information from the event
                long senderId = e.senderId;
                long receiverId = e.receiverId;
                // increase the time first before sending the message
                e.localTime = ++this.time;
                String content = e.content;

                /** send a message of the following format
                 * SENDER_ID|RECEIVER_ID|LOCAL_TIME
                 */
                String msg = Long.toString(senderId) + "-" +
Long.toString(receiverId)
                        + "-" + e.localTime + "-" + content;
                sendEvent(msg);
                break;
            // RECEIVE EVENT
            case 2:
                // update its logical clock
                this.time = Math.max(e.localTime, this.time) + 1;
                break;

            // REQUEST EVENT
            case 3:
                // update its local clock
                e.localTime = ++this.time;
                // add new request to the priority queue
                clockPQ.add(new Request(this.time, this.getId()));
                String requestContent = "REQUEST-" + this.time + "-" + this.getId();
                sendEvent(requestContent);
                break;
            // REPLY REQUEST EVENT
            case 4:
                // update its local clock
                ++this.time;
                // add new request to the priority queue
                clockPQ.add(new Request(e.localTime, e.senderId));
                break;
            // REPLY EVENT
            case 5:
                e.localTime = ++this.time;
                senderId = e.senderId;
                break;
            // ACK EVENT
            case 6:
                // update its local clock
                ++this.time;
                break;
            // ACK EVENT
            default:
                break;
        }
        printTime(e);
    }

    public void printTime(Event e) {
        String logging = "-------------------------\n";
        logging += "Process " + this.getId() + "\n";
        logging += "Process' local time " + this.getTime() + "\n";
        logging += "\tEvent type: ";

        switch (e.type) {
            case 0:
                logging += "LOCAL EVENT\n";
                break;
            case 1:
                logging += "SEND EVENT\n";
                break;
            case 2:
                logging += "RECEIVE EVENT\n";
                break;
            case 3:
                logging += "REQUEST EVENT\n";
                break;
            case 4:
                logging += "RECEIVE REQUEST EVENT\n";
                break;
            case 5:
                logging += "REPLY EVENT\n";
                break;
            case 6:
                logging += "ACK EVENT\n";
                break;
            default:
                break;
        }

        logging += "\tEvent sender's ID: " + e.senderId + "\n";
        logging += "\tEvent receiver's ID: " + e.receiverId + "\n";
        logging += "\tEvent local time: " + e.localTime + "\n";
        logging += "\tEvent content: " + e.content + "\n";
        logging += "-------------------------\n";

        System.out.print(logging);
    }

    public void run() {
        String greeting = "";
        greeting = "Unique ID " + this.getId() +
                " is initialized with local clock " + this.time;
        if (this.order != -1)
            greeting = "Process " + this.order + " " + greeting;

        System.out.println(greeting);
        try {
            while (true) {
                DatagramPacket d = new DatagramPacket(new byte[256], 256);
                sock.receive(d);
                String s = new String(d.getData());
                // System.out.println(this.getId() + " received " + s);
                String[] meta = s.trim().split("-");

                // if this is a REQUEST event
                if (meta[0].equals("REQUEST")) {
                    int requestTime = Integer.parseInt(meta[1]);
                    long senderId = Long.parseLong(meta[2]);
                    if (this.getId() != senderId) {
                        // create a RECEIVE event for every clock
                        Event e = new Event(4, senderId, this.getId(), requestTime,
"");
                        updateTime(e);
                    }
                    // if this is a REPLY event
                } else if (meta[0].equals("REPLY")) {

                } else {
                    long senderId = Long.parseLong(meta[0]);
                    long receiverId = Long.parseLong(meta[1]);
                    int localTime = Integer.parseInt(meta[2]);
                    String content = "";
                    // if there is a message
                    if (meta.length >= 4)
                        content = meta[3];

                    if (this.getId() == receiverId) {
                        Event e = new Event(2, senderId, receiverId, localTime,
content);
                        updateTime(e);
                    }
                }
            }
        } catch (Exception e) {
            System.err.println("LC Failed: " + e);
            return;
        }
    }
}


/**
 * Project Name: DSproject
 * Created With: IntelliJ IDEA.
 * Author: Baftjar TABAKU & Erli REÇI
 * Date Created: 1/16/2020
 * Time Created: 12:48 PM
 **/
package Exercise6_LaportAlgorithm;

import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.util.PriorityQueue;
import java.util.Random;

public class LamportClock extends Thread {
    private MulticastSocket sock;
    private InetAddress group;
    private int port;
    private PriorityQueue<Request> clockPQ;

    // local time of a process
    private int time;

    // order of the process (viewed from the master's perspective)
    private int order;

    public LamportClock(InetAddress group, int port) throws Exception {
        this.group = group;
        this.port = port;

        // if we don't assign an order to a process
        this.order = -1;

        // set local time to random
        Random rand = new Random();
        // this.time = rand.nextInt(10);
        this.time = 0;

        // initialize the priority queue
        this.clockPQ = new PriorityQueue<Request>();

        sock = new MulticastSocket(port);
        sock.setTimeToLive(2);
        sock.joinGroup(group);
    }

    public LamportClock(InetAddress group, int port, int order) throws Exception {
        this(group, port);
        this.order = order;
    }

    public int getOrder() {
        return this.order;
    }
    public int getTime() {
        return this.time;
    }

    public int localEvent() {
        ++this.time;
        // System.out.println(this.getId() + " performing local event. local time is
" + this.time);
        return this.time;
    }

    public int receivedEvent(long senderId, int receivedTime) {
        // System.out.println(this.getId() + " received message from "
        //     + senderId + ". local time is " + this.time);
        return this.time;
    }

    public int sendEvent(String msg) throws Exception {
        byte[] data = msg.getBytes();
        DatagramPacket d = new DatagramPacket(data, data.length, group, port);
        sock.send(d);
        return this.time;
    }

    public void updateTime(Event e) throws Exception {
        int type = e.type;
        switch (type) {
            // LOCAL EVENT
            case 0:
                this.localEvent();
                break;
            // SEND EVENT
            case 1: // extract information from the event
                long senderId = e.senderId;
                long receiverId = e.receiverId;
                // increase the time first before sending the message
                e.localTime = ++this.time;
                String content = e.content;

                /** send a message of the following format
                 * SENDER_ID|RECEIVER_ID|LOCAL_TIME
                 */
                String msg = Long.toString(senderId) + "-" +
Long.toString(receiverId)
                        + "-" + e.localTime + "-" + content;
                sendEvent(msg);
                break;
            // RECEIVE EVENT
            case 2:
                // update its logical clock
                this.time = Math.max(e.localTime, this.time) + 1;
                break;

            // REQUEST EVENT
            case 3:
                // update its local clock
                e.localTime = ++this.time;
                // add new request to the priority queue
                clockPQ.add(new Request(this.time, this.getId()));
                String requestContent = "REQUEST-" + this.time + "-" + this.getId();
                sendEvent(requestContent);
                break;
            // REPLY REQUEST EVENT
            case 4:
                // update its local clock
                ++this.time;
                // add new request to the priority queue
                clockPQ.add(new Request(e.localTime, e.senderId));
                break;
            // REPLY EVENT
            case 5:
                e.localTime = ++this.time;
                senderId = e.senderId;
                break;
            // ACK EVENT
            case 6:
                // update its local clock
                ++this.time;
                break;
            // ACK EVENT
            default:
                break;
        }
        printTime(e);
    }

    public void printTime(Event e) {
        String logging = "-------------------------\n";
        logging += "Process " + this.getId() + "\n";
        logging += "Process' local time " + this.getTime() + "\n";
        logging += "\tEvent type: ";

        switch (e.type) {
            case 0:
                logging += "LOCAL EVENT\n";
                break;
            case 1:
                logging += "SEND EVENT\n";
                break;
            case 2:
                logging += "RECEIVE EVENT\n";
                break;
            case 3:
                logging += "REQUEST EVENT\n";
                break;
            case 4:
                logging += "RECEIVE REQUEST EVENT\n";
                break;
            case 5:
                logging += "REPLY EVENT\n";
                break;
            case 6:
                logging += "ACK EVENT\n";
                break;
            default:
                break;
        }

        logging += "\tEvent sender's ID: " + e.senderId + "\n";
        logging += "\tEvent receiver's ID: " + e.receiverId + "\n";
        logging += "\tEvent local time: " + e.localTime + "\n";
        logging += "\tEvent content: " + e.content + "\n";
        logging += "-------------------------\n";

        System.out.print(logging);
    }

    public void run() {
        String greeting = "";
        greeting = "Unique ID " + this.getId() +
                " is initialized with local clock " + this.time;
        if (this.order != -1)
            greeting = "Process " + this.order + " " + greeting;

        System.out.println(greeting);
        try {
            while (true) {
                DatagramPacket d = new DatagramPacket(new byte[256], 256);
                sock.receive(d);
                String s = new String(d.getData());
                // System.out.println(this.getId() + " received " + s);

                String[] meta = s.trim().split("-");

                // if this is a REQUEST event
                if (meta[0].equals("REQUEST")) {
                    int requestTime = Integer.parseInt(meta[1]);
                    long senderId = Long.parseLong(meta[2]);
                    if (this.getId() != senderId) {
                        // create a RECEIVE event for every clock
                        Event e = new Event(4, senderId, this.getId(), requestTime,
"");

                        updateTime(e);
                    }
                    // if this is a REPLY event
                } else if (meta[0].equals("REPLY")) {

                } else {
                    long senderId = Long.parseLong(meta[0]);
                    long receiverId = Long.parseLong(meta[1]);
                    int localTime = Integer.parseInt(meta[2]);
                    String content = "";
                    // if there is a message
                    if (meta.length >= 4)
                        content = meta[3];

                    if (this.getId() == receiverId) {
                        Event e = new Event(2, senderId, receiverId, localTime,
content);

                        updateTime(e);
                    }
                }
            }
        } catch (Exception e) {
            System.err.println("LC Failed: " + e);
            return;
        }
    }
}


package Exercise6_LaportAlgorithm;

/**
 * Project Name: DSproject
 * Created With: IntelliJ IDEA.
 * Author: Baftjar TABAKU & Erli REÇI
 * Date Created: 1/16/2020
 * Time Created: 12:47 PM
 **/
public class Request implements Comparable<Request> {

    public int time;
    public long processId;
```

```java
    public Request(int time, long processId) {
        this.time = time;
        this.processId = processId;
    }

    public int getTime() {
        return this.time;
    }

    public long getProcessId() {
        return this.processId;
    }

    @Override
    public int compareTo(Request other) {
        if (this.getTime() == other.getTime())
            return 0;
        else if (this.getTime() > other.getTime())
            return 1;
        else
            return -1;
    }
}
```

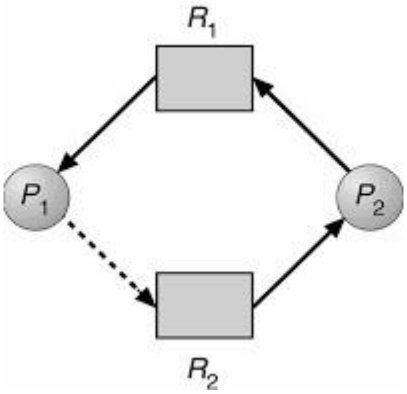
Input & Output example:

```
Enter number of processes

5

Process 0 Unique ID 12 is initialized with local clock 0

Process 1 Unique ID 13 is initialized with local clock 0

Process 2 Unique ID 14 is initialized with local clock 0

Process 3 Unique ID 15 is initialized with local clock 0

Process 4 Unique ID 16 is initialized with local clock 0

LOCAL 1

-------------------------

Process 13

Process' local time 1

        Event type: LOCAL EVENT

        Event sender's ID: 13

        Event receiver's ID: 0

        Event local time: 0

        Event content:
```

```
-------------------------
LOCAL 2
-------------------------
Process 14
Process' local time 1
        Event type: LOCAL EVENT
        Event sender's ID: 14
        Event receiver's ID: 0
        Event local time: 0
        Event content:
-------------------------
SEND 1 0
-------------------------
Process 13
Process' local time 2
        Event type: SEND EVENT
        Event sender's ID: 13
        Event receiver's ID: 12
        Event local time: 2
        Event content:
-------------------------
-------------------------
Process 12
Process' local time 3
        Event type: RECEIVE EVENT
        Event sender's ID: 13
        Event receiver's ID: 12
        Event local time: 2
        Event content:
-------------------------
```

7. Implementation of Banker's Algorithm for avoiding Deadlock

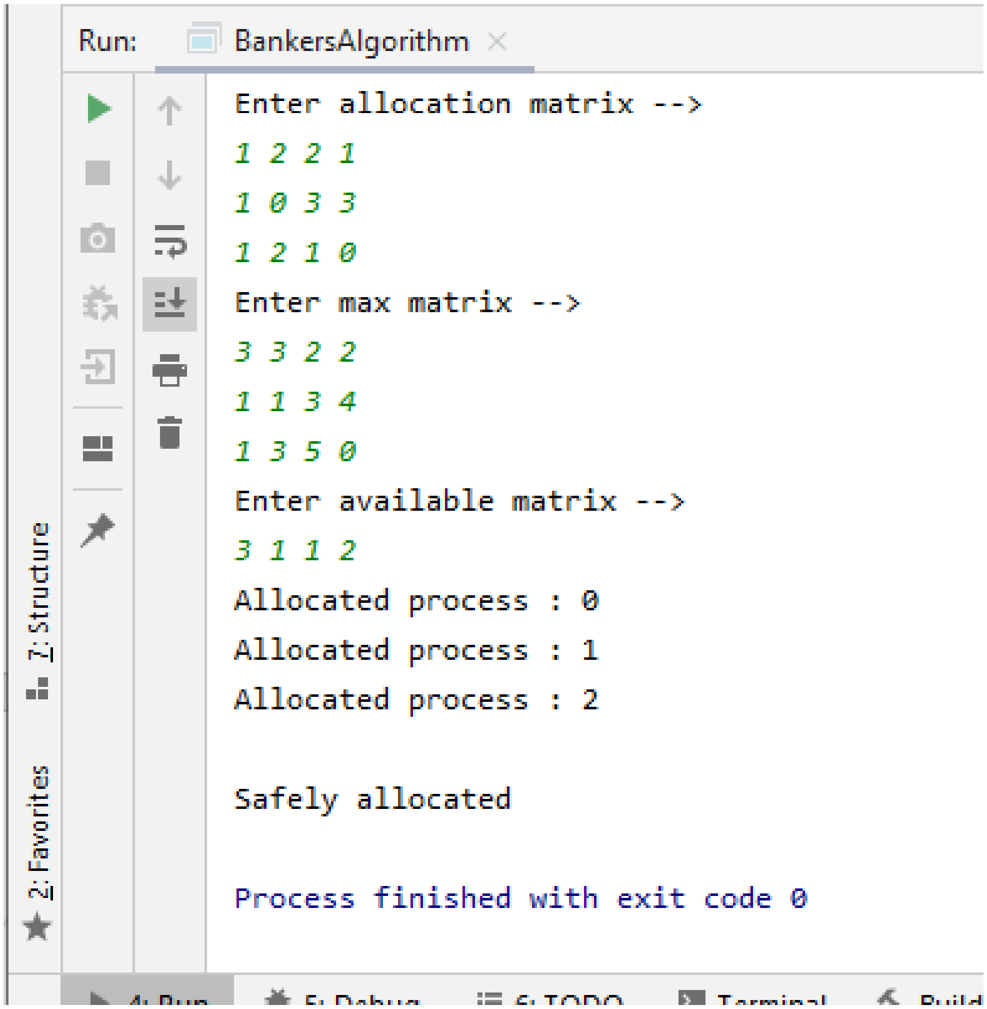
The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.



```
/**
 * Project Name: DSproject
 * Created With: IntelliJ IDEA.
 * Author: Baftjar TABAKU & Erli REÇI
 * Date Created: 1/16/2020
 * Time Created: 2:59 PM
 **/
package Exercise7_BankersAlgorithm;
import java.util.Scanner;

public class BankersAlgorithm {
    private int need[][], allocate[][], max[][], avail[][], np, nr;

    private void input() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter no. of processes and resources : ");
        np = sc.nextInt();  //no. of process
        nr = sc.nextInt();  //no. of resources
        need = new int[np][nr];  //initializing arrays
        max = new int[np][nr];
        allocate = new int[np][nr];
        avail = new int[1][nr];

        System.out.println("Enter allocation matrix -->");
        for (int i = 0; i < np; i++)
            for (int j = 0; j < nr; j++)
                allocate[i][j] = sc.nextInt();  //allocation matrix

        System.out.println("Enter max matrix -->");
        for (int i = 0; i < np; i++)
            for (int j = 0; j < nr; j++)
                max[i][j] = sc.nextInt();  //max matrix

        System.out.println("Enter available matrix -->");
        for (int j = 0; j < nr; j++)
```

```
            avail[0][j] = sc.nextInt();  //available matrix

        sc.close();
    }

    private int[][] calc_need() {
        for (int i = 0; i < np; i++)
            for (int j = 0; j < nr; j++)  //calculating need matrix
                need[i][j] = max[i][j] - allocate[i][j];

        return need;
    }

    private boolean check(int i) {
        //checking if all resources for ith process can be allocated
        for (int j = 0; j < nr; j++)
            if (avail[0][j] < need[i][j])
                return false;

        return true;
    }

    public void isSafe() {
        input();
        calc_need();
        boolean done[] = new boolean[np];
        int j = 0;

        while (j < np) {  //until all process allocated
            boolean allocated = false;
            for (int i = 0; i < np; i++)
                if (!done[i] && check(i)) {  //trying to allocate
                    for (int k = 0; k < nr; k++)
                        avail[0][k] = avail[0][k] - need[i][k] + max[i][k];
                    System.out.println("Allocated process : " + i);
                    allocated = done[i] = true;
                    j++;
                }
            if (!allocated) break;  //if no allocation
        }
        if (j == np)  //if all processes are allocated
            System.out.println("\nSafely allocated");
        else
            System.out.println("All process cant be allocated safely");
    }

    public static void main(String[] args) {
        new BankersAlgorithm().isSafe();
    }
}
```

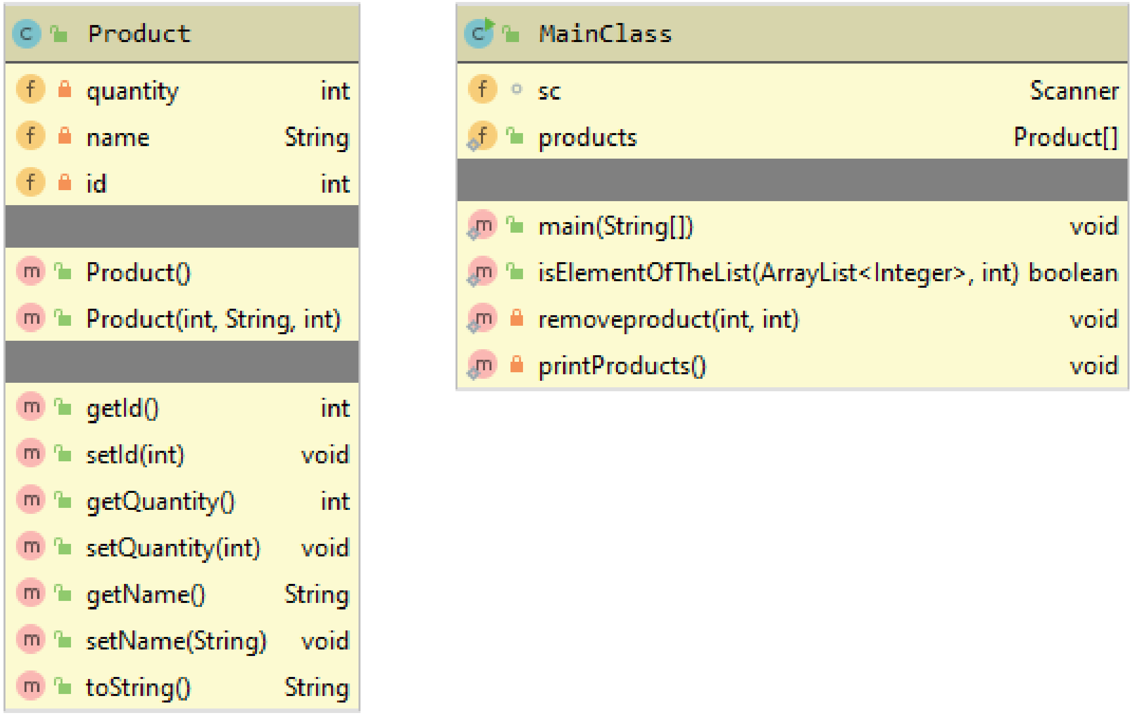
The output and input:



```
Enter allocation matrix -->
1 2 2 1
1 0 3 3
1 2 1 0
Enter max matrix -->
3 3 2 2
1 1 3 4
1 3 5 0
Enter available matrix -->
3 1 1 2
Allocated process : 0
Allocated process : 1
Allocated process : 2

Safely allocated

Process finished with exit code 0
```

8. Case study on Inventory Management

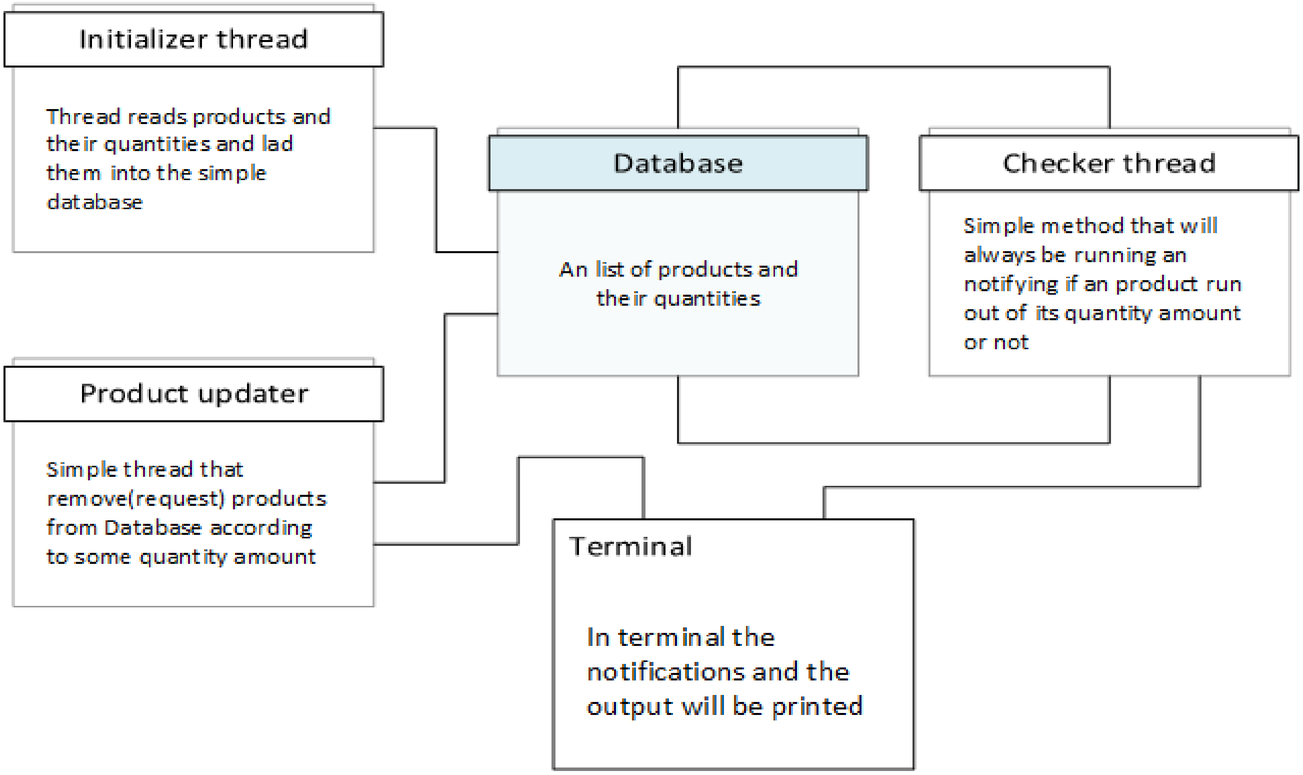
Inventory management system in this case is implemented in java, using threads, their representation is given in the UML diagram:



Product	
quantity	int
name	String
id	int
Product()	
Product(int, String, int)	
getId()	int
setId(int)	void
getQuantity()	int
setQuantity(int)	void
getName()	String
setName(String)	void
toString()	String

MainClass	
sc	Scanner
products	Product[]
main(String[])	void
isElementOfTheList(ArrayList<Integer>, int)	boolean
removeproduct(int, int)	void
printProducts()	void

One thread will take care to load the database, one thread will always be checking it anytime, and informing when a product will run out of quantity.



Thread reads products and their quantities and lad them into the simple database

An list of products and their quantities

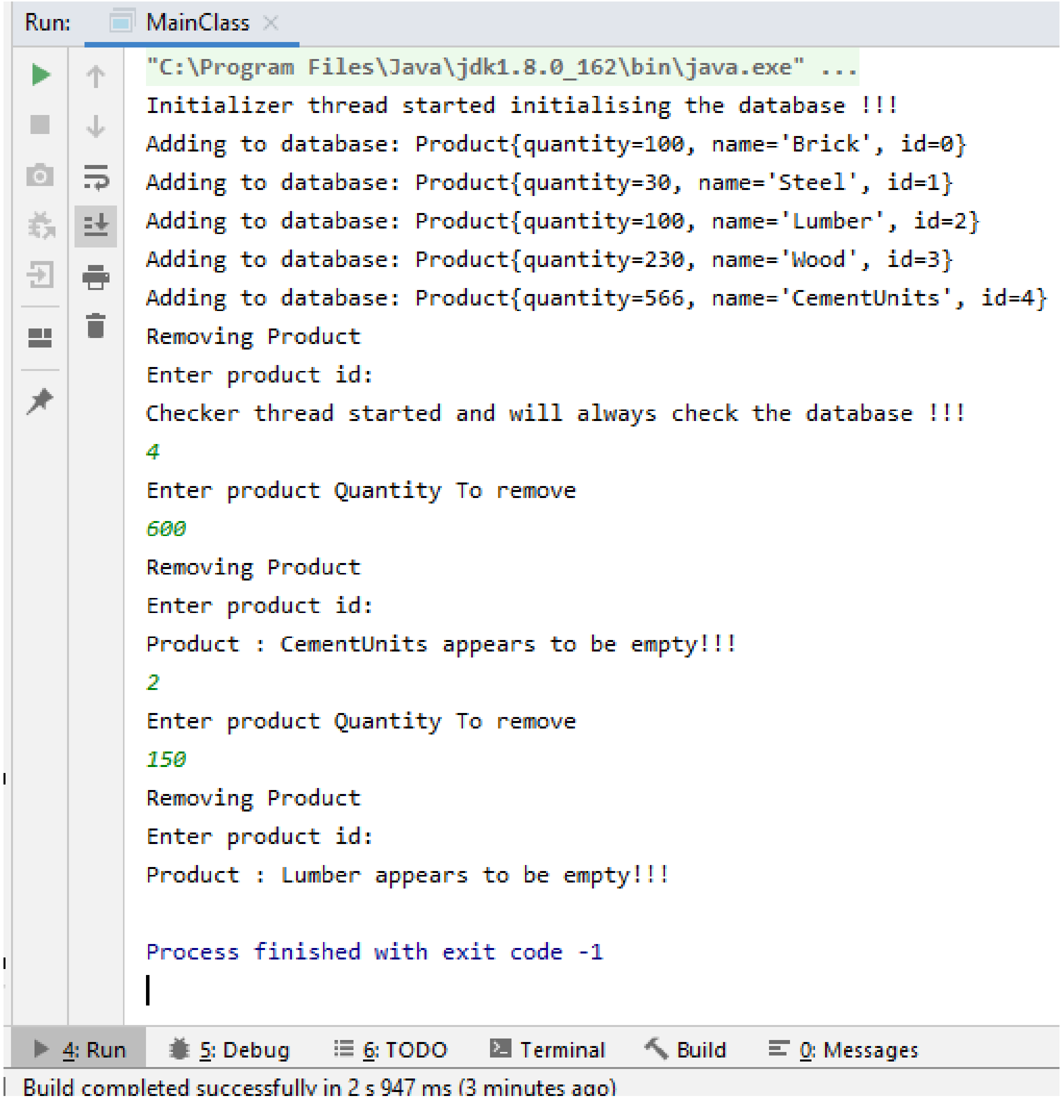
Simple method that will always be running an notifying if an product run out of its quantity amount or not

Simple thread that remove(request) products from Database according to some quantity amount

In terminal the notifications and the output will be printed

```
package Exercise8_CaseStudyInverntoryManagement;
import java.time.chrono.IsoChronology;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
 * Project Name: DSproject
 * Created With: IntelliJ IDEA.
 * Author: Baftjar TABAKU & Erli REÇI
 * Date Created: 1/16/2020
 * Time Created: 3:06 PM
 **/

//Threads
//Adding products

public class MainClass {
    Scanner sc = new Scanner(System.in);
    public static Product[] products;

    //Main calling class
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        //Simply array implementation for an construction shop
        //Products nd anything is pre defined for asy usage,
        String[] productNames = {"Brick", "Steel", "Lumber", "Wood",
"CementUnits"};
        int[] materialQuantity = {100, 30, 100, 230, 566, 360};

        //Initialisation thread
        Thread initialiseThread = new Thread() {
            public void run() {
                products = new Product[productNames.length];
                System.out.println("Initializer thread started
initialising the database !!! ");
                for (int i = 0; i < productNames.length; i++) {
                    products[i] = new Product(materialQuantity[i],
productNames[i], i);
                    System.out.println("Adding to database: " +
products[i].toString());
                }
            }
        };
        initialiseThread.run();

        // Checker thread that will always check the database
        Thread checkerThread = new Thread() {
            public void run() {
                //to avoid notify the same product again and again
                System.out.println("Checker thread started and will always
check the database !!! ");
                ArrayList previousExistingElementsID = new ArrayList();

                while (true) {
                    for (int i = 0; i < productNames.length; i++) {
                        if (products[i].getQuantity() <= 0 &&
!isElementOfTheList(previousExistingElementsID, i)) {
                            System.out.println("Product : " +
products[i].getName() + " appears to be empty!!!");
                            //it may be negative but we set it to 0
                            products[i].setQuantity(0);
                            previousExistingElementsID.add(i);
                        }
                    }
                }
            }
        };
        checkerThread.start();

        while (true) {
            System.out.println("Removing Product");
            System.out.println("Enter product id:");
            int id = scanner.nextInt();
            System.out.println("Enter product Quantity To remove");
            int quantity = scanner.nextInt();
            removeproduct(id, quantity);
            //starting thread to check again

        }

    }

    public static boolean isElementOfTheList(ArrayList<Integer> elements,
int element) {
        for (int i = 0; i < elements.size(); i++) {
            if (elements.get(i) == element) {
                return true;
            }
        }
        return false;
    }

    private static void removeproduct(int id, int quantity) {
        for (int i = 0; i < products.length; i++) {
            if (products[i].getId() == id) {
                products[i].setQuantity(products[i].getQuantity() -
quantity);
            }
        }
    }

    private static void printProducts() {
        for (int i = 0; i < products.length; i++) {
            System.out.println(products[i].toString());
        }
    }
}
```

```java
package Exercise8_CaseStudyInverntoryManagement;

/**
 * Project Name: DSproject
 * Created With: IntelliJ IDEA.
 * Author: Baftjar TABAKU & Erli REÇI
 * Date Created: 1/16/2020
 * Time Created: 3:19 PM
 **/
public class Product {
    private int quantity;
    private String name;
    private int id;

    public Product() {
        quantity = 0;
        name = "";
    }

    public Product(int quantity, String name, int id) {
        this.quantity = quantity;
        this.name = name;
        this.id = id;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Product{" +
                "quantity=" + quantity +
                ", name='" + name + '\'' +
                ", id=" + id +
                '}';
    }
}
```

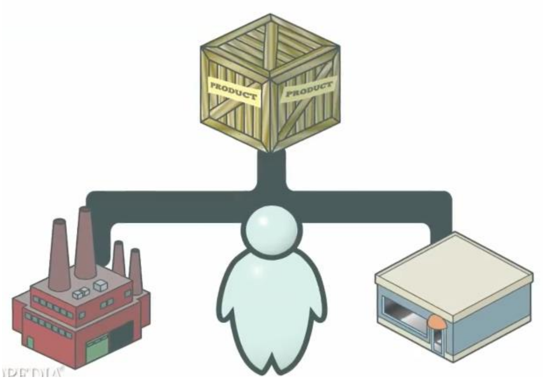


```
"C:\Program Files\Java\jdk1.8.0_162\bin\java.exe" ...
Initializer thread started initialising the database !!!
Adding to database: Product{quantity=100, name='Brick', id=0}
Adding to database: Product{quantity=30, name='Steel', id=1}
Adding to database: Product{quantity=100, name='Lumber', id=2}
Adding to database: Product{quantity=230, name='Wood', id=3}
Adding to database: Product{quantity=566, name='CementUnits', id=4}
Removing Product
Enter product id:
Checker thread started and will always check the database !!!
4
Enter product Quantity To remove
600
Removing Product
Enter product id:
Product : CementUnits appears to be empty!!!
2
Enter product Quantity To remove
150
Removing Product
Enter product id:
Product : Lumber appears to be empty!!!

Process finished with exit code -1
```

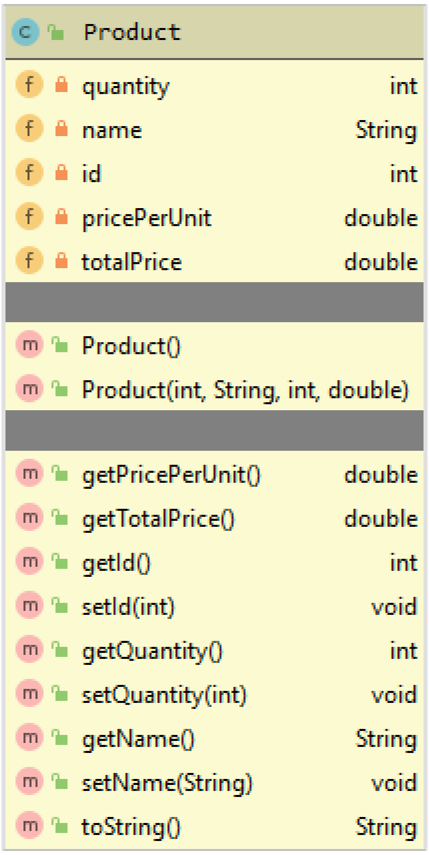
Build completed successfully in 2 s 947 ms (3 minutes ago)

9. Case study on Inventory Management

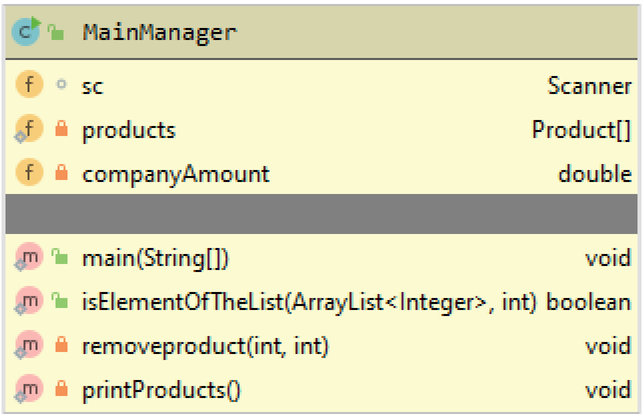
Supply chain management (SCM) seeks to streamline tasks that comprise the supply chain process , combined network of activities and resources involved in moving raw materials, components and finished products from original suppliers to end users. The SCM is based on the idea that every product is the result of the efforts of many entities along the supply chain.



SCM attempts to coordinate the efforts while minimizing costs.



Ⓒ	Product
ⓕ 🔒 quantity	int
ⓕ 🔒 name	String
ⓕ 🔒 id	int
ⓕ 🔒 pricePerUnit	double
ⓕ 🔒 totalPrice	double
ⓜ Product()	
ⓜ Product(int, String, int, double)	
ⓜ getPricePerUnit()	double
ⓜ getTotalPrice()	double
ⓜ getId()	int
ⓜ setId(int)	void
ⓜ getQuantity()	int
ⓜ setQuantity(int)	void
ⓜ getName()	String
ⓜ setName(String)	void
ⓜ toString()	String



Ⓒ	MainManager
ⓕ ○ sc	Scanner
ⓕ 🔒 products	Product[]
ⓕ 🔒 companyAmount	double
ⓜ main(String[])	void
ⓜ isElementOfTheList(ArrayList<Integer>, int)	boolean
ⓜ 🔒 removeproduct(int, int)	void
ⓜ 🔒 printProducts()	void

UML Diagram representation

```
package Exercise9_ChainManagementSystem;

import java.util.ArrayList;
import java.util.Scanner;
/**
 * Project Name: DSproject
 * Created With: IntelliJ IDEA.
 * Author: Baftjar TABAKU & Erli REÇI
 * Date Created: 1/18/2020
 * Time Created: 12:21 AM
 **/
```

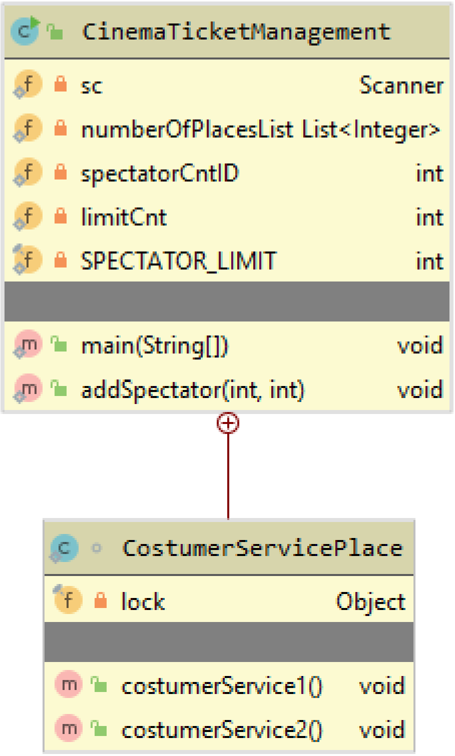
```
public class MainManager {
    Scanner sc = new Scanner(System.in);
    private static Product[] products;
    private double companyAmount = 746905.0; //price in EUR

    //Main calling class
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        //Simply array implementation for an construction shop
        //Products nd anything is pre defined for asy usage,
        String[] productNames = {"Brick", "Steel", "Lumber", "Wood",
"CementUnits"};
        int[] materialQuantity = {100, 30, 100, 230, 566, 360};
        double[] pricePerUnit = {10, 120, 100, 60, 26, 20};

        //Initialisation thread
        Thread initialiseThread = new Thread() {
            public void run() {
                products = new Product[productNames.length];
                System.out.println("Initializer thread started initialising the
database !!! ");
                for (int i = 0; i < productNames.length; i++) {
                    products[i] = new Product(materialQuantity[i],
productNames[i], i, pricePerUnit[i]);
                    System.out.println("Adding to database: " +
products[i].toString());
                }
            }
        };
        initialiseThread.run();

        // Checker thread that will always check the database
        Thread checkerThread = new Thread() {
            public void run() {
                //to avoid notify the same product again and again
                System.out.println("Checker thread started and will always check
the database !!! ");
                ArrayList previousExistingElementsID = new ArrayList();

                while (true) {
                    for (int i = 0; i < productNames.length; i++) {
                        if (products[i].getQuantity() <= 0 &&
!isElementOfTheList(previousExistingElementsID, i)) {
                            System.out.println("Product : " +
products[i].getName() + " appears to be empty!!!");
                            //it may be negative but we set it to 0
                            products[i].setQuantity(0);
                            previousExistingElementsID.add(i);
                        }
                    }
                }
            }
        };
        checkerThread.start();
        //Main function menu BEGIN
        while (true) {
            System.out.println("Removing Product");
            System.out.println("Enter product id:");
            int id = scanner.nextInt();
            System.out.println("Enter product Quantity To remove");
            int quantity = scanner.nextInt();
            removeproduct(id, quantity);
            //starting thread to check again
        }
        //Main function menu END

    }

    public static boolean isElementOfTheList(ArrayList<Integer> elements, int
element) {
        for (int i = 0; i < elements.size(); i++) {
            if (elements.get(i) == element) {
                return true;
            }
        }
        return false;
    }

    private static void removeproduct(int id, int quantity) {
        for (int i = 0; i < products.length; i++) {
            if (products[i].getId() == id) {
                //Add the earning amount
                double earning;
                if (quantity > products[i].getQuantity()) {
                    earning = products[i].getTotalPrice();
                    products[i].setQuantity(0);
                } else {
                    earning = (products[i].getQuantity()-
quantity)*products[i].getPricePerUnit();
                    products[i].setQuantity(products[i].getQuantity() -
quantity);
                }
            }
        }
    }

    private static void printProducts() {
        for (int i = 0; i < products.length; i++) {
            System.out.println(products[i].toString());
        }
    }
}
```

```
package Exercise9_ChainManagementSystem;

/**
 * Project Name: DSproject
 * Created With: IntelliJ IDEA.
```

```java
 * Author: Baftjar TABAKU & Erli REÇI
 * Date Created: 1/16/2020
 * Time Created: 3:19 PM
 **/
public class Product {
    private int quantity;
    private String name;
    private int id;

    public double getPricePerUnit() {
        return pricePerUnit;
    }

    private double pricePerUnit;
    private double totalPrice;

    public Product() {
        quantity = 0;
        name = "";
    }

    public Product(int quantity, String name, int id, double pricePerUnit) {
        this.quantity = quantity;
        this.name = name;
        this.id = id;
        this.pricePerUnit = pricePerUnit;
        totalPrice = getTotalPrice();
    }

    public double getTotalPrice() {
        return pricePerUnit * quantity;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Product{" +
                "quantity=" + quantity +
                ", name='" + name + '\'' +
                ", id=" + id +
                ", pricePerUnit=" + pricePerUnit +
                ", totalPrice=" + totalPrice +
                '}';
    }
}
```

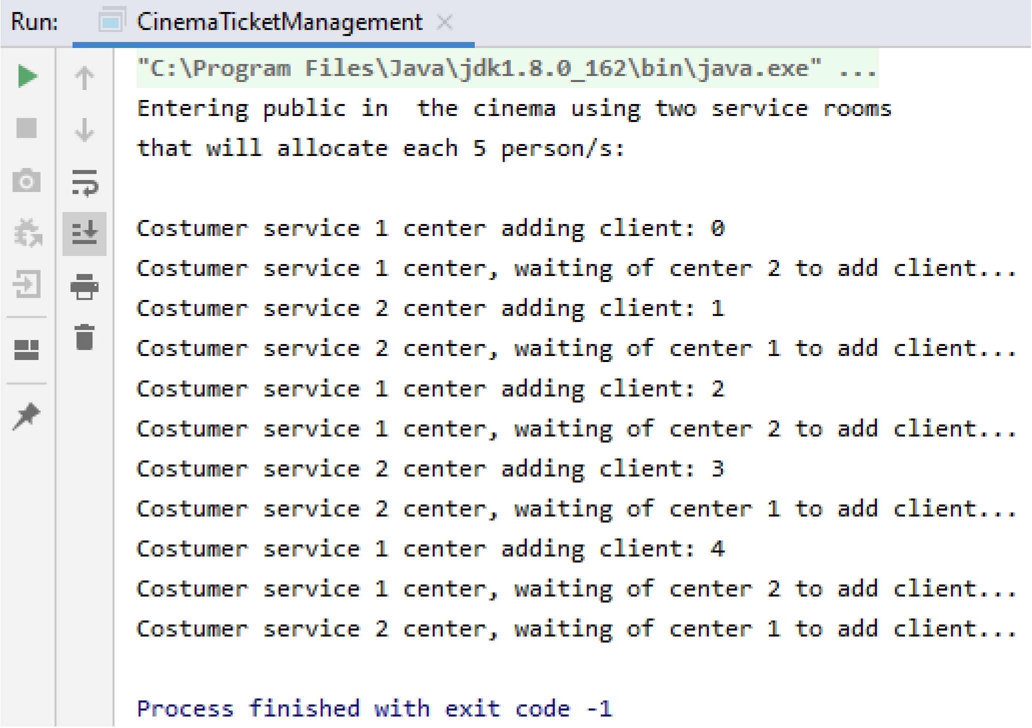
10. Case study on Reservation System

Since this was an advanced topic , the main focus of this solution was to show threads synchronization between them, we included the idea of two costumer centers, center 1 and center 2 which symbolize two threads, selling tickets to the clients and adding them into cinema places (until an certain limit) then for each client they add, they must get synchronized between them to they wont add the same client two times.



```
package Exercise10_ReservationSystem;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
 * Project Name: DSproject
 * Package Name: Exercise10_ReservationSystem
 * Created With: IntelliJ IDEA.
 * Author: Baftjar TABAKU & Erli REÇI
 * Created on: 1/18/2020, Time: 2:17 AM
 **/
public class CinemaTicketManagement {
    private static Scanner sc = new Scanner(System.in);
    private static List<Integer> numberOfPlacesList = new ArrayList<>();//10 rows
with 20 seats 200 persons
    private static int spectatorCntID = 0, limitCnt = 0;
    private static final int SPECTATOR_LIMIT = 5;
//    private static final int ADDING_LIMIT = 1;


    //for synchronisation used an inner class
    static class CostumerServicePlace {
        private final Object lock = new Object();

        //method of client service 1
        public void costumerService1() throws InterruptedException {
            synchronized (lock) {
                while (true) {
                    if (numberOfPlacesList.size() == SPECTATOR_LIMIT || limitCnt
== 1) {
                        System.out.println("Costumer service 1 center, waiting of
center 2 to add client...");
                        lock.wait();
                    } else {
                        System.out.println("Costumer service 1 center adding
client: " + spectatorCntID);
                        numberOfPlacesList.add(spectatorCntID);
                        spectatorCntID++;
                        limitCnt++;
                        lock.notify();
                    }
                    Thread.sleep(1500);
                }
            }
        }

        //method of client service 2
        public void costumerService2() throws InterruptedException {
            synchronized (lock) {
                while (true) {
                    if (numberOfPlacesList.size() == SPECTATOR_LIMIT || limitCnt
== 0) {
                        System.out.println("Costumer service 2 center, waiting of
center 1 to add client...");
                        lock.wait();
                    } else {
                        System.out.println("Costumer service 2 center adding
client: " + spectatorCntID);
                        numberOfPlacesList.add(spectatorCntID);
                        spectatorCntID++;
                        limitCnt -= 1;
                        lock.notify();
                    }
                    Thread.sleep(1500);
                }
            }
        }
    }

    public static void main(String[] args) {
        //Begin of initialization
        System.out.println("Entering public in  the cinema using two service
rooms \n" +
                "that will allocate each " + SPECTATOR_LIMIT + " person/s:\n");

        CostumerServicePlace costumerServicePlaceSync = new
CostumerServicePlace();


        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    costumerServicePlaceSync.costumerService1();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    costumerServicePlaceSync.costumerService2();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        t1.start();
        t2.start();

    }

    public static void addSpectator(int spectatorOrder, int spectatorId) {
        //Adding person on the seat
//          numberOfPlaces[spectatorOrder] = spectatorId;
    }
//Todo printing array with thread if the array of places get updates, but is not
necessary in this case

}
```

The output example:



```
"C:\Program Files\Java\jdk1.8.0_162\bin\java.exe" ...
Entering public in  the cinema using two service rooms
that will allocate each 5 person/s:

Costumer service 1 center adding client: 0
Costumer service 1 center, waiting of center 2 to add client...
Costumer service 2 center adding client: 1
Costumer service 2 center, waiting of center 1 to add client...
Costumer service 1 center adding client: 2
Costumer service 1 center, waiting of center 2 to add client...
Costumer service 2 center adding client: 3
Costumer service 2 center, waiting of center 1 to add client...
Costumer service 1 center adding client: 4
Costumer service 1 center, waiting of center 2 to add client...
Costumer service 2 center, waiting of center 1 to add client...

Process finished with exit code -1
```