```
package client;

import interfaces.*;
import java.rmi.*;
import java.util.*;

/************************************************************************/
/* This is the client that will send tasks to the servers.              */
/* It will                                                              */
/*                                                                      */
/*    1. Get a list of available servers (via LBLookup service)         */
/*    2. Determine the less loaded server (via Executor service)        */
/*    3. Send a task to the less loaded server (via Executor service)   */
/*                                                                      */
/************************************************************************/

// This is an exemple of a task : it prints "hello world!" on the screen
class PrintHello implements Task {
    public PrintHello() {
    }

    public Object run() {
        System.out.println("Hello world !");
        for (long l = 0; l < 100000000; l++);
        return null;
    }
}


// This is the client that searches the best server and send it a task
public class Client {

    // Method called by the VM : instanciates a Client and sends the
    // PrintHello to a server
    public static void main(String[] s) {
        Client cl = new Client();
        cl.executeTask(new PrintHello());
    }


    public Client() {
    }


    // Searches the best server and sends it a task
    // Returns : null in case of problem (no servers or server error)
    //           Object that is the result of the task's calculation
    public Object executeTask (Task task) {
        try {
            Distributor d = (Distributor)
                Naming.lookup(NamingCst.Server + NamingCst.DistributorName);
            return d.forwardTask(task);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

```
/*********************************************************************/
/* This is the server that will forward tasks that are sent by       */
/* the clients.                                                      */
/*                                                                   */
/*                                                                   */
/* IMPLEMENTATION OF THE DISTRIBUTOR INTERFACE                       */
/*********************************************************************/


package server;
import interfaces.*;

import java.rmi.*;
import java.rmi.server.*;
import java.util.*;


public class DServer extends UnicastRemoteObject
                     implements Distributor
{
    private Vector mExecutors;


    DServer() throws RemoteException {
        super();
        mExecutors = new Vector();

        try {
            Naming.rebind(NamingCst.Server +
                          NamingCst.DistributorName, this);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] s) {
        try {
            new DServer();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }


    //-----------------------------------------------------------------
    // Implementation of the Distributor interface

    public Object forwardTask(Task t) throws RemoteException {
        Executor bestServer = null;

        if (mExecutors.size()>0) {
            int bestLoad = 0;
            for (int i = 0; i < mExecutors.size(); i++) {
                Executor e = (Executor) mExecutors.get(i);
                int  load  = e .getLoad();
                if (i == 0 || load < bestLoad) {
                    bestLoad   = load;
                    bestServer = e;
                }
            }
        } else {
            throw new RemoteException("No servers available.");
        }

        return bestServer.execute(t);
    }


    public synchronized void register(Executor e) throws RemoteException {
        mExecutors.add(e);
        System.out.println("DServer: Number of Servers is " + mExecutors.size());
    }
    public synchronized void unregister(Executor e) throws RemoteException {
        mExecutors.remove(e);
        System.out.println("DServer: Number of Servers is " + mExecutors.size());
    }


}
```

```
/*********************************************************************/
/* This is the server that will execute tasks that are sent by       */
/* the clients.                                                      */
/*                                                                   */
/* When it is instanciated, it will register itself with the         */
/* distributor service which maintains a list of                     */
/* Executors.                                                        */
/*                                                                   */
/* IMPLEMENTATION OF THE EXECUTOR INTERFACE                          */
/*********************************************************************/


package server;
import interfaces.*;

import java.rmi.*;
import java.rmi.server.*;


public class Server extends UnicastRemoteObject
                   implements Executor
{

    // Keep track of the number of tasks that are being executed
    private int mLoad;
    private int   mId; // Only for debugging

    // Instanciate -> register with lblookup service
    public Server(int id) throws RemoteException {
        super();
        mLoad = 0;
        mId   = id;

        try {
            Distributor d = (Distributor)
                Naming.lookup(NamingCst.Server + NamingCst.DistributorName);
            d.register(this);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Destruction --> deregister with lblookup service
    protected void finalize () {
        System.out.println("Cleaning up...");
        try {
            Distributor d = (Distributor)
                Naming.lookup(NamingCst.Server + NamingCst.DistributorName);
            d.unregister(this);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }


    // Method called by the VM : instanciate a server
    public static void main(String[] s) {
        System.runFinalizersOnExit(true);

        try {
            Executor server = new Server(0);
            System.out.println("Server running. Press ENTER to stop.");

            // Wait for keystroke
            int a = System.in.read();

            // Makes finalize() get called
            System.exit(0);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Changes the load of the server
    private synchronized void addLoad(int i) {
        mLoad += i;
        System.out.println("Server load is now " + mLoad);
    }


    //-----------------------------------------------------------------
    // Implementation of the Executor interface

    public Object execute(Task task) throws RemoteException {
        addLoad(1);
```



```
        Object ret = task.run();
        addLoad(-1);

        return ret;
    }

    public synchronized int getLoad() {
        return mLoad;
    }
}
```

```
package interfaces;

import java.rmi.*;

public interface Distributor extends Remote {
    public void register(Executor exec)   throws RemoteException;
    public void unregister(Executor exec) throws RemoteException;
    public Object forwardTask(Task t)     throws RemoteException;
}
```

```
package interfaces;

/*****************************************************************/
/* This is the interface that defines the service Executor       */
/* Implemented by Server                                         */
/*****************************************************************/

import java.rmi.*;

public interface Executor extends Remote {

    // Calling this method will execute the task that is passed as parameter
    public Object execute(Task task) throws RemoteException;

    // This will return the number of tasks that are being executed
    public int getLoad() throws RemoteException;
}
```

```
package interfaces;

/*********************************************************************/
/* This is the interface that defines the various constants used   */
/* in this project.                                                */
/*********************************************************************/

public interface NamingCst  {
    static final String Server = "//127.0.0.1/";
    static final String DistributorName = "Distributor";
}
```

```
package interfaces;

import java.rmi.*;

/*********************************************************************/
/* This is the interface that defines a task                         */
/* This interface is used to pass tasks to the Executor              */
/*********************************************************************/

public interface Task extends java.io.Serializable {
    // Causes the taks to perform its duties
    Object run();
}
```