**ĐỀ ÔN THI CUỐI KÌ MÔN LẬP TRÌNH MẠNG**

***I. RMI***

***1. Phần chung cho cách viết RMI***

**a) Server**

**public** **interface** Echo **extends** Remote{

**public** String echo (String s) **throws** RemoteException;

}

**public** **class** EchoImp **extends** UnicastRemoteObject **implements** Echo{

**private** **static** **final** **long** *serialVersionUID* = 1L;

**protected** EchoImp() **throws** RemoteException {

**super**();

}

**public** String echo(String s) **throws** RemoteException {

**return** "Xin chao: " + s;

}

}

**public** **class** Server {

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

Registry r = LocateRegistry.*createRegistry*(12345);

Echo e = **new** EchoImp();

r.rebind("echo", e);

System.*out*.println("Server dang chay!");

}

}

**b) Client**

**public** **class** Client {

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

Registry reg = LocateRegistry.*getRegistry*("127.0.0.1", 12345);

Echo e = (Echo) reg.lookup("echo");

System.*out*.println(e.echo("LTM"));

}

}

***2. Phần tra cứu thông tin***

**a) Server**

**public** **class** SinhVien **implements** Serializable{

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** String name;

**private** **int** mssv;

**private** **int** age;

**private** **double** score;

**public** SinhVien(String name, **int** mssv, **int** age, **double** score) {

**this**.name = name;

**this**.mssv = mssv;

**this**.age = age;

**this**.score = score;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getMssv() {

**return** mssv;

}

**public** **void** setMssv(**int** mssv) {

**this**.mssv = mssv;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** **double** getScore() {

**return** score;

}

**public** **void** setScore(**double** score) {

**this**.score = score;

}

**public** String toString(){

**return** "Ten: " + name +", mssv: " + mssv +", tuoi: "+ age + ", diem: "+score;

}

}

**public** **class** SinhVienDAO **extends** UnicastRemoteObject **implements** ISinhVien {

Connection con;

**private** **static** **final** **long** *serialVersionUID* = 1L;

**protected** SinhVienDAO() **throws** RemoteException {

**try** {

Class.*forName*("sun.jdbc.obdc.JdbcOdbcDriver");

con = DriverManager.*getConnection*("jdbc:odbc:Student");

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**public** ArrayList<SinhVien> findByName(String name) **throws** RemoteException {

ArrayList<SinhVien> li = **new** ArrayList<>();

Statement sta;

**try** {

sta = con.createStatement();

String sql = "select \* from student where Name = '" + name + "'";

ResultSet re = sta.executeQuery(sql);

**while** (re.next()) {

li.add(**new** SinhVien(re.getString("Name"), re.getInt("Mssv"), re

.getInt("Age"), re.getDouble("Score")));

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** li;

}

**public** ArrayList<SinhVien> findByAge(**int** age) **throws** RemoteException {

ArrayList<SinhVien> li = **new** ArrayList<>();

Statement sta;

**try** {

sta = con.createStatement();

String sql = "select \* from student where Age = " + age;

ResultSet re = sta.executeQuery(sql);

**while** (re.next()) {

li.add(**new** SinhVien(re.getString("Name"), re.getInt("Mssv"), re

.getInt("Age"), re.getDouble("Score")));

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** li;

}

**public** ArrayList<SinhVien> findMoreThanScore(**double** score)

**throws** RemoteException {

ArrayList<SinhVien> li = **new** ArrayList<>();

Statement sta;

**try** {

sta = con.createStatement();

String sql = "select \* from student where Score > " + score;

ResultSet re = sta.executeQuery(sql);

**while** (re.next()) {

li.add(**new** SinhVien(re.getString("Name"), re.getInt("Mssv"), re

.getInt("Age"), re.getDouble("Score")));

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** li;

}

}

**public** **interface** ISinhVien **extends** Remote{

**public** ArrayList<SinhVien> findByName(String name) **throws** RemoteException;

**public** ArrayList<SinhVien> findByAge(**int** age) **throws** RemoteException;

**public** ArrayList<SinhVien> findMoreThanScore(**double** score) **throws** RemoteException;

}

**public** **class** Server {

**public** Server() **throws** RemoteException {

@SuppressWarnings("resource")

Scanner sc = **new** Scanner(System.*in*);

List<SinhVien> list;

SinhVienDAO dao = **new** SinhVienDAO();

**while** (**true**) {

String command = sc.nextLine();

StringTokenizer token = **new** StringTokenizer(command, "\t");

String request = token.nextToken();

**switch** (request) {

**case** "findByName":

list = dao.findByName(token.nextToken());

**for** (SinhVien sv : list) {

System.*out*.println(sv);

}

**break**;

**case** "findByAge":

list = dao.findByAge(Integer.*parseInt*(token.nextToken()));

**for** (SinhVien sv : list) {

System.*out*.println(sv);

}

**break**;

**case** "findByScore":

list = dao.findMoreThanScore(Double.*parseDouble*(token

.nextToken()));

**for** (SinhVien sv : list) {

System.*out*.println(sv);

}

**break**;

**default**:

System.*out*.println("Nhap lenh sai vui long nhap lai");

**break**;

}

}

}

**public** **static** **void** main(String[] args) {

Random r = **new** Random();

System.*out*.println(r.nextInt(24));

}

}

**public** **class** Server1 {

**public** Server1() **throws** RemoteException {

Registry regis = LocateRegistry.*createRegistry*(12345);

ISinhVien sv = **new** SinhVienDAO();

regis.rebind("sv", sv);

}

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

**new** Server1();

}

}

**b) Client**

**public** **class** Client {

**public** Client() **throws** RemoteException, NotBoundException {

Registry rs = LocateRegistry.*getRegistry*(12345);

ISinhVien sv = (ISinhVien) rs.lookup("sv");

System.*out*.println("ban hay nhap lenh");

@SuppressWarnings("resource")

Scanner sc = **new** Scanner(System.*in*);

List<SinhVien> list;

**while** (**true**) {

String command = sc.nextLine();

StringTokenizer token = **new** StringTokenizer(command, "[\t ]");

String request = token.nextToken();

**switch** (request) {

**case** "findByName":

list = sv.findByName(token.nextToken());

print(list);

**break**;

**case** "findByAge":

list = sv.findByAge(Integer.*parseInt*(token.nextToken()));

print(list);

**break**;

**case** "findByScore":

list = sv.findMoreThanScore(Double.*parseDouble*(token.nextToken()));

print(list);

**break**;

**default**:

System.*out*.println("Nhap lenh sai vui long nhap lai");

**break**;

}

}

}

**private** **void** print(List<SinhVien> list) {

**for** (SinhVien sv1 : list) {

System.*out*.println((SinhVien)sv1);

}

}

***3. Phần copy bằng mảng***

**a) Server**

**public** **interface** Copy **extends** Remote{

**public** **void** copy(String s) **throws** RemoteException;

**public** **byte**[] sendData() **throws** RemoteException;

**public** **void** closeFile() **throws** RemoteException;

}

**public** **class** CopyIm **extends** UnicastRemoteObject **implements** Copy {

BufferedInputStream bis;

**protected** CopyIm() **throws** RemoteException {

**super**();

}

**private** **static** **final** **long** *serialVersionUID* = -329587034347766967L;

**public** **void** copy(String s) **throws** RemoteException {

**try** {

bis = **new** BufferedInputStream(**new** FileInputStream(s));

} **catch** (FileNotFoundException e) {

**throw** **new** RemoteException();

}

}

**public** **byte**[] sendData() **throws** RemoteException {

**byte**[] d = **new** **byte**[1024];

**int** i;

**try** {

i = bis.read(d);

**if**(i==-1) **return** **null**;

**byte**[] con = **new** **byte**[i];

System.*arraycopy*(d, 0, con, 0, i);

**return** con;

} **catch** (IOException e) {

**return** **null**;

}

}

**public** **void** closeFile() **throws** RemoteException {

**try** {

bis.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

**public** **class** Server {

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

Registry rg = LocateRegistry.*createRegistry*(12346);

Copy c = **new** CopyIm();

rg.rebind("Copy", c);

}

}

**b) Client**

**public** **class** Client{

**public** **static** **void** main(String[] args) **throws** NotBoundException, IOException {

Registry r = LocateRegistry.*createRegistry*(12346);

Copy c = (Copy) r.lookup("Copy");

BufferedOutputStream bos = **new** BufferedOutputStream(**new** FileOutputStream("d:\\test\\1.txt"));

c.copy("d:\\test\\2.txt");

**byte**[] data;

**while**((data = c.sendData())!= **null**){

bos.write(data);

}

c.closeFile();

bos.close();

}

}

***4. Phần copy bằng file***

**a) Server**

**public** **interface** Copy **extends** Remote{

**public** **int** openFile(String filename) **throws** RemoteException;

**public** **byte**[] readFile (**int** index) **throws** RemoteException;

**void** closeFile(**int** index) **throws** RemoteException;

}

**public** **class** CopyIm **extends** UnicastRemoteObject **implements** Copy {

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** ArrayList<BufferedInputStream> fileList = **new** ArrayList<>();

**protected** CopyIm() **throws** RemoteException {

**super**();

}

**public** **int** openFile(String file) **throws** RemoteException {

BufferedInputStream bis;

**try** {

bis = **new** BufferedInputStream(**new** FileInputStream(file));

fileList.add(bis);

} **catch** (FileNotFoundException e) {

**return** -1;

}

**return** fileList.size()-1;

}

**public** **byte**[] readFile(**int** index) **throws** RemoteException {

**byte**[] data = **new** **byte**[1024];

**int** c;

**try** {

**if**((c=fileList.get(index).read(data))!=-1){

**if**(c==1024) **return** data;

**else**{

**byte**[] tmp =**null**;

System.*arraycopy*(data, 0, tmp, 0, c);

**return** tmp;

}

}

} **catch** (IOException e) {

e.printStackTrace();

}

**return** **null**;

}

**public** **void** closeFile(**int** index) **throws** RemoteException {

**try** {

fileList.get(index).close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**public** **class** Server {

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

Registry reg = LocateRegistry.*createRegistry*(12346);

Copy copy = **new** CopyIm();

reg.rebind("copy", copy);

}

}

**b) Client**

**public** **class** Client {

**public** **static** **void** main(String[] args) **throws** RemoteException, NotBoundException, IOException {

Registry reg = LocateRegistry.*getRegistry*("127.0.0.1", 12346);

Copy copy = (Copy) reg.lookup("copy");

BufferedOutputStream bos = **new** BufferedOutputStream(**new** FileOutputStream(**new** File("d:\\test\\1.txt")));

**int** index = copy.openFile("d:\\test\\2.txt");

**byte**[] data;

**while**((data = copy.readFile(index))!=**null**){

bos.write(data);

}

bos.close();

copy.closeFile(index);

}

}

***II. TCP***

***1. Phần copy dùng DIS, DOS***

**public** **class** Server {

**public** **static** String *SERVER\_PACK* = "d:\\test\\server";

**public** **static** **int** *PORT* = 1236;

DataInputStream dis;

DataOutputStream dos;

**public** Server() **throws** IOException{

@SuppressWarnings("resource")

ServerSocket ss = **new** ServerSocket(*PORT*);

System.*out*.println("Waiting...");

Socket socket = ss.accept();

System.*out*.println("Connect.");

File s = **new** File(*SERVER\_PACK*);

dos = **new** DataOutputStream(**new** BufferedOutputStream(socket.getOutputStream()));

copy(s);

dos.writeInt(0);

dos.flush();

dos.close();

dis.close();

}

**public** **void** copy(File f) **throws** IOException {

**if**(f.isFile()){

dis = **new** DataInputStream(**new** BufferedInputStream(**new** FileInputStream(f)));

dos.writeInt(1);

dos.flush();

dos.writeLong(f.length());

dos.flush();

dos.writeUTF(f.getPath().substring(*SERVER\_PACK*.length(), f.getPath().length()));

dos.flush();

**byte**[] buffer = **new** **byte**[1024];

**int** i;

**while**((i = dis.read(buffer))!= -1){

dos.write(buffer, 0, i);

dos.flush();

}

}**else**{

dos.writeInt(2);

dos.flush();

dos.writeUTF(f.getPath().substring(*SERVER\_PACK*.length(), f.getPath().length()));

dos.flush();

**for** (File f1 : f.listFiles()){

copy(f1);

}

}

}

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

**new** Server();

}

}

**public** **class** Client {

**public** **static** String *SERVER\_PACK* ="d:\\test\\client";

**public** **static** **int** *SERVER\_PORT* = 1236;

DataInputStream dis;

BufferedOutputStream bos;

**public** Client() **throws** IOException{

@SuppressWarnings("resource")

Socket socket = **new** Socket("127.0.0.1", *SERVER\_PORT*);

System.*out*.println("Connected.");

File d = **new** File(*SERVER\_PACK*);

**if**(!d.exists()) d.mkdirs();

dis = **new** DataInputStream(**new** BufferedInputStream(socket.getInputStream()));

**long** size;

String name;

**long** byteReaded = 0;

**long** byteMustRead;

**byte**[] buffer;

**int** i;

**while**(**true**){

byteReaded = 0;

**if**((i = dis.readInt()) == 1){

size = dis.readLong();

name = dis.readUTF();

byteMustRead = (size-byteReaded) > 20480 ? 20480 : size-byteReaded;

System.*out*.println(byteMustRead);

buffer = **new** **byte**[(**int**) byteMustRead];

bos = **new** BufferedOutputStream(**new** FileOutputStream(*SERVER\_PACK* + name));

System.*out*.println(*SERVER\_PACK* + name);

**while**(byteMustRead != 0){

i = dis.read(buffer);

bos.write(buffer, 0, i);

byteReaded += i;

byteMustRead = (size - byteReaded) > 20480 ? 20480 : size - byteReaded;

}

bos.flush();

bos.close();

}**else** **if**(i == 2){

name = dis.readUTF();

File f = **new** File(*SERVER\_PACK* + name);

f.mkdirs();

}**else**{

dis.close();

**break**;

}

}

}

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

**new** Client();

}

}

**public** **class** Test {

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

File f = **new** File("d:\\test\\1.mp3");

FileInputStream fis = **new** FileInputStream(f);

FileOutputStream fos = **new** FileOutputStream("d:\\test\\2.mp3");

**long** len = f.length();

**long** byteReaded = 0;

**int** byteMustRead = (len - byteReaded > 1024) ? 1024 : (**int**) (len-byteReaded);

**byte**[] buffer = **new** **byte**[byteMustRead];

**int** i = 0;

**while**(byteMustRead != 0){

i = fis.read(buffer);

fos.write(buffer, 0, i);

byteReaded += i;

byteMustRead = (len-byteReaded > 1024) ? 1024 : (**int**) (len - byteReaded);

buffer = **new** **byte**[byteMustRead];

}

fis.close();

fos.flush();

fos.close();

}

}

***2. Phần copy dùng BIS, DOS***

**public** **class** Server {

**public** **static** String *SERVER\_PATH* = "E:\\may ao\\ubuntu 01(12.10)\\abc.txt";

**public** **static** **int** *PORT* = 1495;

DataOutputStream dos;

BufferedInputStream bis;

**public** Server() **throws** IOException{

@SuppressWarnings("resource")

ServerSocket ss = **new** ServerSocket(*PORT*);

System.*out*.println("Waitting...");

Socket socket = ss.accept();

System.*out*.println("Connected.");

File f = **new** File(*SERVER\_PATH*);

dos = **new** DataOutputStream(**new** BufferedOutputStream(socket.getOutputStream()));

bis = **new** BufferedInputStream(**new** FileInputStream(f));

**long** size = f.length();

dos.writeLong(size);

dos.flush();

dos.writeUTF(f.getName());

dos.flush();

**long** byteReaded = 0;

**int** byteMustRead = (size-byteReaded)> 10240 ? 10240 :(**int**) (size-byteReaded);

**byte**[] buffer = **new** **byte**[byteMustRead];

**int** i;

**while**(byteMustRead != 0){

i = bis.read(buffer);

dos.write(buffer, 0, i);

dos.flush();

byteReaded += i;

byteMustRead = (size-byteReaded)> 10240 ? 10240 :(**int**) (size-byteReaded);

buffer = **new** **byte**[byteMustRead];

}

bis.close();

dos.flush();

dos.close();

socket.close();

}

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

**new** Server();

}

}

**public** **class** Client {

**public** **static** String *CLIENT\_PATH* = "d:\\test\\client";

**public** **static** **int** *PORT* = 1495;

DataInputStream dis;

BufferedOutputStream bos;

**public** Client() **throws** IOException{

@SuppressWarnings("resource")

Socket socket = **new** Socket("127.0.0.1", *PORT*);

System.*out*.println("Ket noi thanh cong.");

dis = **new** DataInputStream(**new** BufferedInputStream(socket.getInputStream()));

**long** size = dis.readLong();

String name = dis.readUTF();

bos = **new** BufferedOutputStream(**new** FileOutputStream(*CLIENT\_PATH* +"\\" + name));

**long** byteReaded = 0;

**int** byteMustRead = (size-byteReaded) > 10240 ? 10240 : (**int**) (size-byteReaded);

**byte**[] buffer = **new** **byte**[byteMustRead];

**int** i;

**while**(byteMustRead != 0){

i = dis.read(buffer);

bos.write(buffer, 0, i);

byteReaded += i;

byteMustRead = (size-byteReaded)> 10240 ? 10240 : (**int**) (size-byteReaded);

buffer = **new** **byte**[byteMustRead];

}

bos.flush();

bos.close();

dis.close();

}

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

**new** Client();

}

}

***3. Phần upload***

**public** **class** Server {

**public** **static** **final** String *SERVER\_PTH* = "D:\\server";

**public** Server() **throws** IOException {

//mở cổng server

@SuppressWarnings("resource")

ServerSocket ss = **new** ServerSocket(1111);

System.*out*.println("cho ket noi");

//bat dau ket noi

Socket socket = ss.accept();

System.*out*.println("ket noi thanh cong");

//mở tream lay du lieu

DataInputStream stream = **new** DataInputStream(socket.getInputStream());

// lay ten file tu stream

String path = stream.readUTF();

//lay kich thuoc

**long** size = stream.readLong();

//tạo file tren server

File f = **new** File(*SERVER\_PTH* + "\\" + path);

//tạo tream lưu du lieu

BufferedOutputStream bos = **new** BufferedOutputStream(

**new** FileOutputStream(f));

// sô lượng byte đã ghi vào file

**long** byteReaded = 0;

// số lượng byte cần đ�?c trong lần tiếp theo

**int** byteMustRead =(**int**) ((size - byteReaded) > 20480 ? 20480 : size - byteReaded);

//tạo mang byte để lấy dữ liệu

**byte** [] buffer;

//bắt đầu lấy dữ liệu và lưu vào file

**while**(byteMustRead!=0){

buffer= **new** **byte**[byteMustRead];

//d�?c dữ liệu

**int** i=stream.read(buffer);

// ghi dữ liẹu

bos.write(buffer, 0, i);

// cập nhật lại số lượng byte đã đ�?c

byteReaded+=i;

// số lượng byte cần đ�?c trong lần tiếp theo

byteMustRead =(**int**) ((size - byteReaded) > 20480 ? 20480 : size - byteReaded);

}

// đóng file

bos.flush();

bos.close();

}

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

**new** Server();

}

}

**public** **class** Client {

**public** Client() **throws** UnknownHostException, IOException {

// kết nối tới server

@SuppressWarnings("resource")

Socket socket = **new** Socket("localhost", 1111);

// đưa dữ liệu và stream trong socket

DataOutputStream stream = **new** DataOutputStream(socket.getOutputStream());

@SuppressWarnings("resource")

Scanner s = **new** Scanner(System.*in*);

// lấy dòng comment từ bàn phím

String line = s.nextLine();

StringTokenizer tk = **new** StringTokenizer(line, " ");

String data[] = **new** String[3];

**int** count = 0;

// phân tích dòng comment là lấy thông tin cho vào mang data

**while** (tk.hasMoreTokens()) {

data[count++] = tk.nextToken();

**if** (count == 3)

**break**;

}

// nếu comment là copy thì thực hiện

**if** (data[0].equals("copy")) {

// lấy đư�?ng dẫn file nguồn

String source = data[1];

String des = data[2];

// tạo file nguồn

File f = **new** File(source);

**if** (des == **null**)

des = f.getName();

// kiểm tra file ton tai

**if** (f.exists()) {

// gửi tên file cho server

stream.writeUTF(des);

stream.flush();

// gửi dung lượng file gửi qua server

stream.writeLong(f.length());

stream.flush();

BufferedInputStream bis = **new** BufferedInputStream(

**new** FileInputStream(f));

**byte**[] buffer = **new** **byte**[20480];

**int** i = -1;

// gửi dữ liệu của file server

**while** ((i = bis.read(buffer)) != -1) {

stream.write(buffer, 0, i);

stream.flush();

}

// đóng file

bis.close();

}}

}

// }

**public** **static** **void** main(String[] args) **throws** UnknownHostException,

IOException {

**new** Client();

}}

***4. Phần download***

**public** **class** Server {

**public** **static** **final** String *SERVER\_PTH* = "e:\\server";

**public** Server() **throws** IOException {

//mở cổng server

@SuppressWarnings("resource")

ServerSocket ss = **new** ServerSocket(1111);

System.*out*.println("cho ket noi");

//bat dau ket noi

**while**(**true**){

Socket socket = ss.accept();

System.*out*.println("ket noi thanh cong");

//mở tream lay du lieu

DataOutputStream dos = **new** DataOutputStream(socket.getOutputStream());

DataInputStream stream = **new** DataInputStream(socket.getInputStream());

// lay ten file tu stream

String path = stream.readUTF();

System.*out*.println("Da lay duoc ten file");

File file = **new** File(*SERVER\_PTH* + "\\" + path);

dos.writeLong(file.length());

dos.flush();

System.*out*.println("Da gui size file");

DataInputStream bis = **new** DataInputStream(

**new** FileInputStream(file));

**byte**[] buffer = **new** **byte**[20480];

**int** a = -1;

**while** ((a = bis.read(buffer)) != -1) {

dos.write(buffer, 0, a);

dos.flush();

}

dos.close();

bis.close();System.*out*.println("Da gui file xuong");

}

}

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

**new** Server();

}

}

**public** **class** Client {

**static** **final** String *CLIENT\_PATH* = "E:\\client";

// public Client() {

**public** **static** **void** client() {

**try** {

@SuppressWarnings("resource")

Socket socket = **new** Socket("127.0.0.1", 1111);

DataOutputStream dos;

DataInputStream dis;

**while** (**true**) {

dos = **new** DataOutputStream(socket.getOutputStream());

dis = **new** DataInputStream(socket.getInputStream());

@SuppressWarnings("resource")

Scanner sc = **new** Scanner(System.*in*);

System.*out*.println("Nhap command");

String line = sc.nextLine();

**if**(line.equals("exit")){

**break**;

}

StringTokenizer st = **new** StringTokenizer(line, " ");

String[] data = **new** String[3];

**int** count = 0;

**while** (st.hasMoreTokens()) {

data[count++] = st.nextToken();

**if** (count == 3)

**break**;

}

// if (data[0].equals("download")) {

String source = data[1];

String dest = data[2];

**if** (dest == **null**) {

dest = source;

}

dos.writeUTF(source);

System.*out*.println("Da gui ten file");

**long** size = dis.readLong();

System.*out*.println("DA doc sizefile");

File file = **new** File(*CLIENT\_PATH*+"\\"+dest);

BufferedOutputStream bos = **new** BufferedOutputStream(**new** FileOutputStream(file));

**long** byteReaded = 0;

**int** byteMustRead = (**int**)((size - byteReaded) > 20480 ? 20480 : size - byteReaded);

**byte**[] buffer;

**int** a;

**while**(byteMustRead != 0){

buffer = **new** **byte**[byteMustRead];

a = dis.read(buffer);

bos.write(buffer, 0, a);

bos.flush();

byteReaded += a;

byteMustRead = (**int**)((size - byteReaded) > 20480 ? 20480 : size - byteReaded);

}

bos.close();

dis.close();

System.*out*.println("Download XONG");

}

// }

} **catch** (UnknownHostException u) {

u.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

}

@SuppressWarnings("static-access")

**public** **static** **void** main(String[] args) {

Client c = **new** Client();

c.*client*();

}

}

***5. Tra cứu thông tin***

**public** **class** SinhVien {

String name;

**int** mssv;

**int** old;

**double** score;

**public** SinhVien(String name, **int** mssv, **int** old, **double** score) {

**this**.name = name;

**this**.mssv = mssv;

**this**.old = old;

**this**.score = score;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getMssv() {

**return** mssv;

}

**public** **void** setMssv(**int** mssv) {

**this**.mssv = mssv;

}

**public** **int** getOld() {

**return** old;

}

**public** **void** setOld(**int** old) {

**this**.old = old;

}

**public** **double** getScore() {

**return** score;

}

**public** **void** setScore(**double** score) {

**this**.score = score;

}

**public** String toString(){

**return** name + "-" + mssv + "-" + old +"-" + score;

}

}

**public** **class** SinhVienDAO {

**static** ArrayList<SinhVien> *list* = **new** ArrayList<>();

**static** {

*list*.add(**new** SinhVien("aaa", 1234, 20, 10));

*list*.add(**new** SinhVien("bbb", 1235, 21, 9));

*list*.add(**new** SinhVien("ccc", 1236, 22, 9));

*list*.add(**new** SinhVien("ddd", 1237, 20, 5));

*list*.add(**new** SinhVien("aaa", 1238, 19, 10));

*list*.add(**new** SinhVien("ddd", 1239, 21, 9));

*list*.add(**new** SinhVien("eee", 1230, 21, 9));

}

**public** **static** ArrayList<SinhVien> findByName(String name) {

ArrayList<SinhVien> re = **new** ArrayList<>();

**for** (**int** i = 0; i < *list*.size(); i++) {

**if** (*list*.get(i).getName().equalsIgnoreCase(name))

re.add(*list*.get(i));

}

**return** re;

}

**public** **static** ArrayList<SinhVien> findByAge(**int** old) {

ArrayList<SinhVien> re = **new** ArrayList<>();

**for** (**int** i = 0; i < *list*.size(); i++) {

**if** (*list*.get(i).getOld() == old)

re.add(*list*.get(i));

}

**return** re;

}

**public** **static** ArrayList<SinhVien> findByScore(**double** score) {

System.*out*.println(score);

ArrayList<SinhVien> re = **new** ArrayList<>();

**for**(**int** i = 0; i < *list*.size(); i++){

**if**(Math.*abs*(*list*.get(i).getScore())< 0.1)

re.add(*list*.get(i));

}

**return** re;

}

}

**public** **class** Server {

Socket socket;

BufferedReader br;

PrintWriter pw;

**public** Server() **throws** IOException{

@SuppressWarnings("resource")

ServerSocket ss = **new** ServerSocket(12346);

System.*out*.println("Waiting...");

socket = ss.accept();

System.*out*.println("Connected.");

**while**(**true**){

br = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

pw = **new** PrintWriter(socket.getOutputStream(), **true**);

process(br, pw);

br.close();

pw.close();

socket.close();

}

}

**private** **void** process(BufferedReader br, PrintWriter pw) **throws** IOException {

String commmand;

ArrayList<SinhVien> list = **null**;

**while**(**true**){

commmand = br.readLine();

StringTokenizer token = **new** StringTokenizer(commmand, " ");

String action = token.nextToken();

**if** (action.equalsIgnoreCase("findbyname")) {

list=SinhVienDAO.*findByName*(token.nextToken());

} **else** **if** (action.equalsIgnoreCase("findbyage")) {

list=SinhVienDAO.*findByAge*(Integer.*parseInt*(token.nextToken()));

} **else** **if** (action.equalsIgnoreCase("findbyscore")) {

list=SinhVienDAO.*findByScore*(Double.*parseDouble*(token.nextToken()));

}**else** **if**(action.equalsIgnoreCase("exit"))

**break**;

pw.println(list.size());

**for** (SinhVien sinhVien : list){

pw.println(sinhVien.toString());

}

}

}

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

**new** Server();

}

}

**public** **class** Client {

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

Socket socket = **new** Socket("localhost", 12346);

System.*out*.println("Connected.");

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

BufferedReader in = **new** BufferedReader(**new** InputStreamReader(System.*in*));

PrintWriter pw = **new** PrintWriter(socket.getOutputStream(), **true**);

PrintWriter out = **new** PrintWriter(System.*out*, **true**);

String command;

**while** (**true**){

command = in.readLine();

pw.println(command);

**if**(command.equalsIgnoreCase("exit")) **break**;

command = br.readLine();

**int** count = Integer.*parseInt*(command);

**for**(**int** i = 0; i< count; i++){

command = br.readLine();

out.print(command);

}

}

in.close();

out.close();

pw.close();

br.close();

socket.close();

}

}

**public** **class** EchoServer **extends** Thread{

Socket socket;

**public** EchoServer(Socket socket){

**this**.socket = socket;

}

**public** **void** run(){

**super**.run();

System.*out*.println("Connected.");

**try** {

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

PrintWriter pw = **new** PrintWriter(socket.getOutputStream(), **true**);

String command;

**while**(**true**){

command = br.readLine();

**if**(command.equalsIgnoreCase("exit")) **break**;

pw.println("echo: " + command);

}

pw.close();

br.close();

socket.close();

} **catch** (IOException e) {

}

}

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

@SuppressWarnings("resource")

ServerSocket ss = **new** ServerSocket(12346);

System.*out*.println("Waiting...");

**while**(**true**){

EchoServer es = **new** EchoServer(ss.accept());

es.start();

}

}

}

**public** **class** EchoClient {

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

Socket socket = **new** Socket("localhost", 12346);

System.*out*.println("Connected.");

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

BufferedReader in = **new** BufferedReader(**new** InputStreamReader(System.*in*));

PrintWriter pw = **new** PrintWriter(socket.getOutputStream(), **true**);

PrintWriter out = **new** PrintWriter(System.*out*, **true**);

String command;

**while**(**true**){

command = in.readLine();

pw.println(command);

**if**(command.equalsIgnoreCase("exit")) **break**;

command = br.readLine();

out.println(command);

}

in.close();

out.close();

pw.close();

br.close();

socket.close();

}

}

***6. Phần đăng nhập POP3***

SinhVien và SinhVienDAO như 5

**public** **class** User {

String name;

String pass;

**public** User(String name, String pass) {

**this**.name = name;

**this**.pass = pass;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getPass() {

**return** pass;

}

**public** **void** setPass(String pass) {

**this**.pass = pass;

}

}

**public** **class** UserDAO {

**static** ArrayList<User> *list* = **new** ArrayList<>();

**static** {

*list*.add(**new** User("abc", "123"));

*list*.add(**new** User("teo", "123"));

*list*.add(**new** User("ti", "123"));

*list*.add(**new** User("suu", "123"));

*list*.add(**new** User("dan", "123"));

}

**public** **static** **boolean** check(String user){

**for**(User u : *list*){

**if**(u.getName().equals(user))

**return** **true**;

}

**return** **false**;

}

**public** **static** **boolean** check(String user, String pass){

**for**(User u : *list*){

**if**((u.getName().equals(user)) && (u.getPass().equals(pass)))

**return** **true**;

}

**return** **false**;

}

}

**public** **class** Server {

Socket socket;

BufferedReader br;

PrintWriter pw;

String user = "", pass = "";

**public** Server() **throws** IOException{

@SuppressWarnings("resource")

ServerSocket ss = **new** ServerSocket(1234);

System.*out*.println("Waiting...");

socket = ss.accept();

br = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

pw = **new** PrintWriter(socket.getOutputStream(), **true**);

System.*out*.println(br.readLine());

pw.println("Ban hay dang nhap");

processLogin(br, pw);

}

**private** **void** processLogin(BufferedReader br, PrintWriter pw) **throws** IOException {

String command, action, data;

StringTokenizer token;

**while**(**true**){

command = br.readLine();

token = **new** StringTokenizer(command);

action = token.nextToken();

**if**(action.equalsIgnoreCase("user")){

data = token.nextToken();

**if**(UserDAO.*check*(data)){

pw.println("OK");

}**else** {

pw.println("User khong tan thanh");

}

**this**.user = data;

}

**if** (action.equalsIgnoreCase("pass")) {

data = token.nextToken();

**if**(UserDAO.*check*(**this**.user, data)){

pw.println("Dang nhap thanh cong");

**return**;

}**else**{

pw.println("Dang nhap khong thanh cong");

}

**this**.pass = data;

}

}

}

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

**new** Server();

}

}

**public** **class** Client {

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

@SuppressWarnings("resource")

Socket socket = **new** Socket("localhost", 1234);

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

BufferedReader in = **new** BufferedReader(**new** InputStreamReader(System.*in*));

PrintWriter pw = **new** PrintWriter(socket.getOutputStream(), **true**);

PrintWriter out = **new** PrintWriter(System.*out*, **true**);

System.*out*.println("Connected.");

pw.println("Connected.");

System.*out*.println(br.read());

String data;

**while**(**true**){

data = in.readLine();

pw.println(data);

data = br.readLine();

out.println(data);

}

}

}

**public** **class** Server2 {

Socket socket;

BufferedReader br;

PrintWriter pw;

String user = "", pass = "";

**public** Server2() **throws** IOException{

@SuppressWarnings("resource")

ServerSocket ss = **new** ServerSocket(1234);

System.*out*.println("Waiting...");

socket = ss.accept();

System.*out*.println("Connected.");

**while**(**true**){

br = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

pw = **new** PrintWriter(socket.getOutputStream(), **true**);

pw.println("Hay dang nhap");

processLogin(br, pw);

process(br, pw);

br.close();

pw.close();

socket.close();

}

}

**private** **void** process(BufferedReader br, PrintWriter pw) **throws** IOException {

String command;

StringTokenizer token;

String action, data;

**while**(**true**){

command = br.readLine();

token = **new** StringTokenizer(command);

action = token.nextToken();

**if**(action.equalsIgnoreCase("user")){

data = token.nextToken();

**this**.user = data;

**if**(UserDAO.*check*(**this**.user)){

pw.println("OK");

}**else**{

pw.println("User khong ton tai");

}

**continue**;

}**else** **if**(action.equalsIgnoreCase("pass")){

data = token.nextToken();

**this**.pass = data;

**if**(UserDAO.*check*(**this**.user, **this**.pass)){

pw.println("Dang nhap thanh cong");

**return**;

}**else**{

pw.println("Dang nhap khong thanh cong");

}

}**else** **continue**;

}

}

**private** **void** processLogin(BufferedReader br, PrintWriter pw) **throws** IOException {

String command;

ArrayList<SinhVien> list = **null**;

**while**(**true**){

command = br.readLine();

StringTokenizer token = **new** StringTokenizer(command, " ");

String action = token.nextToken();

**if**(action.equalsIgnoreCase("findbyname")){

list = SinhVienDao.*findByName*(token.nextToken());

}**else** **if**(action.equalsIgnoreCase("findbyage")){

list = SinhVienDao.*findByAge*(Integer.*parseInt*(token.nextToken()));

}**else** **if**(action.equalsIgnoreCase("findbyscore")){

list = SinhVienDao.*findByScore*(Double.*parseDouble*(token.nextToken()));

}**else** **if**(action.equalsIgnoreCase("exit")) **break**;

pw.println(list.size());

**for**(SinhVien sinhVien : list){

pw.println(sinhVien.toString());

}

}

}

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

**new** Server2();

}

}

**public** **class** Client2 {

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

Socket socket = **new** Socket("localhost", 1234);

System.*out*.println("Connected.");

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

BufferedReader in = **new** BufferedReader(**new** InputStreamReader(System.*in*));

PrintWriter pw = **new** PrintWriter(socket.getOutputStream(), **true**);

PrintWriter out = **new** PrintWriter(System.*out*, **true**);

String command, data;

out.println(br.readLine());

**while**(**true**){

command = in.readLine();

pw.println(command);

data = br.readLine();

out.println(data);

**if**(data.equalsIgnoreCase("Dang nhap thanh cong")) **break**;

}

**while**(**true**){

command = in.readLine();

pw.println(command);

**if**(command.equalsIgnoreCase("exit")) **break**;

command = br.readLine();

**int** count = Integer.*parseInt*(command);

**for**(**int** i = 0; i < count; i++){

command = br.readLine();

out.println(command);

}

}

in.close();

out.close();

pw.close();

br.close();

socket.close();

}

}

***III. CSDL***

***1. Load CSDL***

**public** **class** CSDL {

**public** **static** **void** main(String[] args) {

**try** {

Class.*forName*("sun.jdbc.odbc.JdbcOdbcDriver");

String database ="jdbc:odbc:SV";

Connection conn;

conn = DriverManager.*getConnection*(database);

String sql = "SELECT \* from Table1";

Statement s = conn.createStatement();

ResultSet rs = s.executeQuery(sql);

**while**(rs.next()){

System.*out*.print(rs.getString(1)+ " ");

System.*out*.print(rs.getString(2)+ " ");

System.*out*.println(rs.getString(3));

}

} **catch** (Exception e) {

System.*out*.println(e.getMessage()+"\n Class not found Exception.");

}

}

}

***2. Tra cứu thông tin***

public class Access {

private static final String DB = "newstar";

public static void main(String[] args) {

try {

// Buoc 1: Load driver

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

// Buoc 2: Tao ket noi xuong CSDL: URL (jdbc:dbms:database

Connection conn = DriverManager.getConnection("jdbc:odbc:" + DB);

// Buoc 3: Tao Statement

Statement stmt = conn.createStatement();

// Buoc 4: Execute

ResultSet rs = stmt.executeQuery("SELECT \* FROM Student");

// Buoc 5: Fetch data

while(rs.next()) {

String sid = rs.getString("SID");

String fn = rs.getString("FirstName");

String ln = rs.getString("LastName");

String cl = rs.getString("Class");

System.out.println("[" + sid + "]." + fn + " " + ln + ". Class:" + cl);

}

} catch (ClassNotFoundException e) {

e.printStackTrace();

} catch (SQLException e) {

e.printStackTrace();

}}}

public class StudentProcedureODBC {

public static boolean dropProcedure(String procedure) {

Connection con = DBConnection.DEFAULT.getConnection();

if (con == null) return false;

String drop = "DROP PROCEDURE IF EXISTS " + procedure;

Statement st = null;

try {

st = con.createStatement();

st.execute(drop);

return true;

} catch (SQLException e) {

MyLogger.singleton.log(StudentProcedureODBC.class.getName(), e.getMessage());

e.printStackTrace();

return false;

}

}

/\*\*

CREATE PROC procProductsAddItem(inProductName VARCHAR(40), inSupplierID LONG, inCategoryID LONG) " & \_

"AS INSERT INTO Products (ProductName, SupplierID, CategoryID) " & \_

"Values (inProductName, inSupplierID, inCategoryID);"

"CREATE PROC procProductsUpdateItem(inProductID LONG, " & \_

" inProductName VARCHAR(40)) " & \_

"AS UPDATE Products SET ProductName = inProductName " & \_

" WHERE ProductID = inProductID;"

\* @return

\*/

public static boolean createInsertProcedure() {

Connection con = DBConnection.DEFAULT.getConnection();

if (con == null) return false;

dropProcedure("insert\_student");

//CREATE PROCEDURE Sales\_By\_Country [Beginning Date] DateTime, [Ending Date] DateTime;

String insert\_pro = "CREATE PROCEDURE insert\_student(id CHAR(3), IN fn VARCHAR(30), IN ln VARCHAR(10), IN bd DATE, IN bp VARCHAR(30)) ";

insert\_pro += " AS ";

insert\_pro += " INSERT INTO student(sid, first\_name, last\_name, birthday, birthplace) ";

insert\_pro += " VALUES(id, fn, ln, bd, bp); ";

try {

Statement st = con.createStatement();

st.executeUpdate(insert\_pro);

return true;

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentProcedureODBC.class.getName(), e.getMessage());

return false;

}  
}

public static void main(String[] args) {

createInsertProcedure();

//dropProcedure("insert\_student");

}}

public class StudentJDBCDAO {

public static Student findStudentsBySID(String sid) {

Connection con = DBConnection.DEFAULT.getConnection();

if (con == null) return null;

try {

String sql = "SELECT first\_name, last\_name, birthday, birthplace FROM student WHERE sid = ?";

PreparedStatement st = con.prepareStatement(sql);

st.setString(1, sid);

ResultSet rs = st.executeQuery();

if(rs.next()) {

String fn = rs.getString(0);

String ln = rs.getString(1);

Date bd = rs.getDate(2);

String bp = rs.getString(3);

Student s = new Student(sid, fn, ln, bd, bp);

return s;

} else {

return null;

}

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentJDBCDAO.class.getName(), e.getMessage());

return null;

}

}

public static List<Student> findStudentsByField(String field, String value) {

Connection con = DBConnection.DEFAULT.getConnection();

List<Student> list = new ArrayList<Student>();

if (con == null) return list;

try {

String sql = "SELECT sid, first\_name, last\_name, birthday, birthplace FROM student WHERE ? = ?";

PreparedStatement st = con.prepareStatement(sql);

st.setString(1, field);

st.setString(2, value);

ResultSet rs = st.executeQuery();

while(rs.next()) {

String sid = rs.getString(0);

String fn = rs.getString(1);

String ln = rs.getString(2);

Date bd = rs.getDate(3);

String bp = rs.getString(4);

Student s = new Student(sid, fn, ln, bd, bp);

list.add(s);

}

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentJDBCDAO.class.getName(), e.getMessage());

}

return list;

}

public static List<Student> getAllStudents() {

Connection con = DBConnection.DEFAULT.getConnection();

List<Student> list = new ArrayList<Student>();

if (con == null) return list;

try {

String sql = "SELECT sid, first\_name, last\_name, birthday, birthplace FROM student";

Statement st = con.createStatement();

ResultSet rs = st.executeQuery(sql);

while(rs.next()) {

String sid = rs.getString(0);

String fn = rs.getString(1);

String ln = rs.getString(2);

Date bd = rs.getDate(3);

String bp = rs.getString(4);

Student s = new Student(sid, fn, ln, bd, bp);

list.add(s);

}

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentJDBCDAO.class.getName(), e.getMessage());

}

return list;

}

public static boolean insertStudent(Student student) {

Connection con = DBConnection.DEFAULT.getConnection();

if (con == null) return false;

try {

String sql = "INSERT INTO student VALUES(?, ?, ?, ?, ?)";

PreparedStatement st = con.prepareStatement(sql);

st.setString(1, student.getSid());

st.setString(2, student.getFirstName());

st.setString(3, student.getLastName());

st.setDate(4, new java.sql.Date(student.getBirthday().getTime()));

st.setString(5, student.getBirthplace());

System.out.println(st.toString());

int rs = st.executeUpdate();

System.out.println("RS:" + rs);

return (rs > 0);

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentJDBCDAO.class.getName(), e.getMessage());

return false;

}}

public static boolean updateStudent(Student student) {

Connection con = DBConnection.DEFAULT.getConnection();

if (con == null) return false;

try {

String sql = "UPDATE student SET first\_name = ?, last\_name = ?, birthday = ?, birthplace = ? WHERE sid = ?";

PreparedStatement st = con.prepareStatement(sql);

st.setString(1, student.getSid());

st.setString(2, student.getFirstName());

st.setString(3, student.getLastName());

st.setDate(4, new java.sql.Date(student.getBirthday().getTime()));

st.setString(5, student.getBirthplace());

int rs = st.executeUpdate();

return (rs > 0);

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentJDBCDAO.class.getName(), e.getMessage());

return false;

}

}

public static boolean deleteStudent(Student student) {

Connection con = DBConnection.DEFAULT.getConnection();

if (con == null) return false;

try {

String sql = "DELETE FROM student WHERE sid = ?";

PreparedStatement st = con.prepareStatement(sql);

st.setString(1, student.getSid());

int rs = st.executeUpdate();

return (rs > 0);

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentJDBCDAO.class.getName(), e.getMessage());

return false;

}

}

public static boolean deleteAllStudent() {

Connection con = DBConnection.DEFAULT.getConnection();

if (con == null) return false;

try {

String sql = "DELETE FROM student";

Statement st = con.createStatement();

int rs = st.executeUpdate(sql);

return (rs > 0);

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentJDBCDAO.class.getName(), e.getMessage());

return false;

}

}

public static void main(String[] args) {

Student s1 = new Student("001", "Nguyen Van", "A", new java.util.Date(), "Can Tho");

Student s2 = new Student("002", "Nguyen Van", "B", new java.util.Date(), "Tp. HCM");

Student s3 = new Student("002", "Nguyen Van", "C", new java.util.Date(), "Tp. HCM");

insertStudent(s1);

insertStudent(s2);

updateStudent(s3);

deleteStudent(s3);

deleteAllStudent();

}}

public class StudentProcedureDAO {

public static Student findStudentsBySID(String sid) {

Connection con = DBConnection.DEFAULT.getConnection();

if (con == null) return null;

try {

String call = "CALL find\_student\_by\_sid(?)";

CallableStatement st = con.prepareCall(call);

st.setString(1, sid);

ResultSet rs = st.executeQuery();

if(rs.next()) {

String fn = rs.getString(0);

String ln = rs.getString(1);

Date bd = rs.getDate(2);

String bp = rs.getString(3);

Student s = new Student(sid, fn, ln, bd, bp);

return s;

} else {

return null;

}

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentProcedureDAO.class.getName(), e.getMessage());

return null;

}

}

public static List<Student> findStudentsByField(String field, String value) {

Connection con = DBConnection.DEFAULT.getConnection();

List<Student> list = new ArrayList<Student>();

if (con == null) return list;

try {

String call = "CALL find\_student\_by\_field(?, ?)";

CallableStatement st = con.prepareCall(call);

st.setString(1, field);

st.setString(2, value);

ResultSet rs = st.executeQuery();

while(rs.next()) {

String sid = rs.getString(0);

String fn = rs.getString(1);

String ln = rs.getString(2);

Date bd = rs.getDate(3);

String bp = rs.getString(4);

Student s = new Student(sid, fn, ln, bd, bp);

list.add(s);

}

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentProcedureDAO.class.getName(), e.getMessage());

}

return list;

}

public static List<Student> getAllStudents() {

Connection con = DBConnection.DEFAULT.getConnection();

List<Student> list = new ArrayList<Student>();

if (con == null) return list;

try {

String call = "CALL get\_all\_students()";

CallableStatement st = con.prepareCall(call);

ResultSet rs = st.executeQuery();

while(rs.next()) {

String sid = rs.getString(0);

String fn = rs.getString(1);

String ln = rs.getString(2);

Date bd = rs.getDate(3);

String bp = rs.getString(4);

Student s = new Student(sid, fn, ln, bd, bp);

list.add(s);

}

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentProcedureDAO.class.getName(), e.getMessage());

}

return list;

}

public static boolean insertStudent(Student student) {

Connection con = DBConnection.DEFAULT.getConnection();

if (con == null) return false;

try {

String call = "CALL insert\_student(?, ?, ?, ?, ?)";

CallableStatement st = con.prepareCall(call);

st.setString(1, student.getSid());

st.setString(2, student.getFirstName());

st.setString(3, student.getLastName());

st.setDate(4, new java.sql.Date(student.getBirthday().getTime()));

st.setString(5, student.getBirthplace());

System.out.println(st.toString());

int rs = st.executeUpdate();

System.out.println("RS:" + rs);

return (rs > 0);

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentProcedureDAO.class.getName(), e.getMessage());

return false;

}

}

public static boolean updateStudent(Student student) {

Connection con = DBConnection.DEFAULT.getConnection();

if (con == null) return false;

try {

String call = "CALL update\_student(?, ?, ?, ?, ?)";

CallableStatement st = con.prepareCall(call);

st.setString(1, student.getSid());

st.setString(2, student.getFirstName());

st.setString(3, student.getLastName());

st.setDate(4, new java.sql.Date(student.getBirthday().getTime()));

st.setString(5, student.getBirthplace());

int rs = st.executeUpdate();

return (rs > 0);

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentProcedureDAO.class.getName(), e.getMessage());

return false;

}

}

public static boolean deleteStudent(Student student) {

Connection con = DBConnection.DEFAULT.getConnection();

if (con == null) return false;

try {

String call = "CALL delete\_student(?)";

CallableStatement st = con.prepareCall(call);

st.setString(1, student.getSid());

int rs = st.executeUpdate();

return (rs > 0);

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentProcedureDAO.class.getName(), e.getMessage());

return false;

}

}

public static boolean deleteAllStudent() {

Connection con = DBConnection.DEFAULT.getConnection();

if (con == null) return false;

try {

String call = "CALL delete\_all\_student()";

CallableStatement st = con.prepareCall(call);

int rs = st.executeUpdate();

return (rs > 0);

} catch (SQLException e) {

e.printStackTrace();

MyLogger.singleton.log(StudentProcedureDAO.class.getName(), e.getMessage());

return false;

}

}

public static void main(String[] args) {

Student s1 = new Student("001", "Nguyen Van", "A", new java.util.Date(), "Can Tho");

Student s2 = new Student("002", "Nguyen Van", "B", new java.util.Date(), "Tp. HCM");

Student s3 = new Student("002", "Nguyen Van", "C", new java.util.Date(), "Tp. HCM");

deleteAllStudent();

insertStudent(s1);

insertStudent(s2);

updateStudent(s3);

deleteStudent(s3);

}}

public class Student {

private String sid;

private String firstName;

private String lastName;

private Date birthday;

private String birthplace;

public Student(String sid, String firstName, String lastName,

Date birthday, String birthplace) {

this.sid = sid;

this.firstName = firstName;

this.lastName = lastName;

this.birthday = birthday;

this.birthplace = birthplace;

}

public String getSid() {

return sid;

}

public void setSid(String sid) {

this.sid = sid;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public Date getBirthday() {

return birthday;

}

public void setBirthday(Date birthday) {

this.birthday = birthday;

}

public String getBirthplace() {

return birthplace;

}

public void setBirthplace(String birthplace) {

this.birthplace = birthplace;

}

}

public class IllegalDateException extends Exception {

private static final long serialVersionUID = 1L;

}

public class Constant {

public static final String MSSQL\_DRIVER = "com.microsoft.sqlserver.jdbc.SQLServerDriver";

public static final String MSSQL\_URL = "jdbc:sqlserver://localhost:1433;databaseName=hungvuongitc;";

public static final String MSSQL\_USER = "sa";

public static final String MSSQL\_PASS = "123456";

public static final String MYSQL\_DRIVER = "com.mysql.jdbc.Driver";

public static final String MYSQL\_URL = "jdbc:mysql://localhost/student";

public static final String MYSQL\_USER = "newstar";

public static final String MYSQL\_PASS = "123456";

public static final String PG\_DRIVER = "org.postgresql.Driver";

public static final String PG\_URL = "jdbc:postgresql://localhost/hungvuongitc";

public static final String PG\_USER = "postgres";

public static final String PG\_PASS = "123456";

public static final String ODBC\_DRIVER = "sun.jdbc.odbc.JdbcOdbcDriver";

public static final String ODBC\_URL = "jdbc:odbc:hungvuongitc";

// MS-Access Data Type

public static final String BIGBINARY = "LONGBINARY";

public static final String BINARY = "BINARY";

public static final String BIT = "BIT";

public static final String COUNTER = "COUNTER";

public static final String CURRENCY = "CURRENCY";

public static final String LONGTEXT = "LONGTEXT";

public static final String MEMO = "LONGTEXT";

public static final String NUMBER\_SINGLE = "SINGLE";

public static final String NUMBER\_DOUBLE = "DOUBLE";

public static final String NUMBER\_BYTE = "UNSIGNED BYTE";

public static final String NUMBER\_INTEGER = "SHORT";

public static final String NUMBER\_LONG = "LONG";

public static final String NUMERIC = "NUMERIC";

public static final String OLE = "LONGBINARY";

public static final String TEXT = "VARCHAR";

public static final String VARBINARY = "VARBINARY";

}

public class DateUtils {

public static Date getDate(int day, int month, int year) {

Calendar cal = new GregorianCalendar();

cal.set(Calendar.DAY\_OF\_MONTH, day);

cal.set(Calendar.MONTH, month);

cal.set(Calendar.YEAR, year);

return cal.getTime();

}

public static Date getDate(String date, String delimeter) throws Exception {

StringTokenizer token = new StringTokenizer(date, delimeter);

String dd = token.nextToken();

String mm = token.nextToken();

String yy = token.nextToken();

try {

int d = Integer.parseInt(dd);

if (d > 31) throw new IllegalDateException();

int m = Integer.parseInt(mm);

if (m > 12) throw new IllegalDateException();

int y = Integer.parseInt(yy);

if (y < 0) throw new IllegalDateException();

return getDate(d, m, y);

} catch (Exception e) {

throw e;

}

}

public static String toString(Date date) {

return "19/09/2012";

}

}

public class MyLogger {

public static final MyLogger singleton = new MyLogger("log\_file.txt");

private FileHandler fh;

private LogManager lm;

public MyLogger(String file) {

lm = LogManager.getLogManager();

try {

fh = new FileHandler(file);

fh.setFormatter(new SimpleFormatter());

} catch (SecurityException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

public void log(String className, String msg) {

try {

Logger logger = lm.getLogger(className);

if (logger == null) {

logger = Logger.getLogger(className);

logger.setLevel(Level.SEVERE);

logger.addHandler(fh);

lm.addLogger(logger);

}

logger.severe(msg);

} catch (Exception e) {

System.out.println("Exception thrown: " + e);

e.printStackTrace();

}}}