**Tema 6 – Conectarea la baza de date MYSQL**

**Muntean Andrei 30227**

**1. Introducere**

Scopul principal al programului este de a simula operaţiunile privind o administrare de stoc folosind Swing API, în special cele privind manipularea unui eveniment.

Programul ar trebui să pună în aplicare următoarele acţiuni:

- Administrarea stocului

- Adăuga produs

- Şterge produs

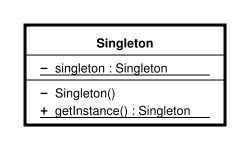
- Modificare titlu produs sau brand

- Cautare dupa titlu

- Listarea produse

Pentru punerea în aplicare a acestei aplicatii am putea folosi modelul Singleton. Modelul **Singleton** este un design pattern, care se utilizeaza pentru a restricţiona [instanţieri](http://translate.google.com/translate?hl=ru&prev=_t&sl=en&tl=ro&u=http://en.wikipedia.org/wiki/Instantiation_%2528computer_science%2529) a unei clase la un [obiect](http://translate.google.com/translate?hl=ru&prev=_t&sl=en&tl=ro&u=http://en.wikipedia.org/wiki/Object-oriented_programming) . Acest lucru este util atunci când avem nevoie ca exact un obiect sa coordoneze acţiunile în cadrul sistemului. Uneori se generalizeaza la sisteme care operează mai eficient atunci când numai unul sau câteva obiecte există. Se consideră, de asemenea, un [anti-pattern](http://en.wikipedia.org/wiki/Anti-pattern) , deoarece este adesea folosit ca un [eufemism](http://translate.google.com/translate?hl=ru&prev=_t&sl=en&tl=ro&u=http://en.wikipedia.org/wiki/Euphemism) pentru [variabila globala](http://translate.google.com/translate?hl=ru&prev=_t&sl=en&tl=ro&u=http://en.wikipedia.org/wiki/Global_variable) .

**Class Diagram :**



[Design-pattern](http://en.wikipedia.org/wiki/Anti-pattern) Singleton este implementat prin crearea unei [clase](http://translate.google.com/translate?hl=ru&prev=_t&sl=en&tl=ro&u=http://en.wikipedia.org/wiki/Class_%2528computer_science%2529) cu o metodă care creează o nouă instanţă a clasei în cazul în care nu există. În cazul în care o instanţă există deja, acesta returnează pur şi simplu o referinţă la acel obiect. Pentru a se asigura că obiectul nu poate fi instantiat in orice alt mod, [constructorul](http://translate.google.com/translate?hl=ru&prev=_t&sl=en&tl=ro&u=http://en.wikipedia.org/wiki/Constructor_%2528computer_science%2529) se face fie private sau protected. Modelul Singleton trebuie să fie construit cu grijă în aplicaţii [multi-threaded](http://translate.google.com/translate?hl=ru&prev=_t&sl=en&tl=ro&u=http://en.wikipedia.org/wiki/Thread_%2528computer_programming%2529). Dacă două “fire” sunt executate în acelaşi timp, atunci când un singleton nu există încă, amândoi trebuie să verifice o instanţă de singleton şi apoi doar unul singur ar trebui să creeze unul nou. În cazul în care limbajul de programare are capacitati de prelucrare concomitenta, metoda ar trebui să fie construita astfel încât să execute o operaţie reciproc exclusiva.

Soluţia clasică la această problemă este de a folosi [excludere reciprocă](http://translate.google.com/translate?hl=ru&prev=_t&sl=en&tl=ro&u=http://en.wikipedia.org/wiki/Mutual_exclusion) cu privire la clasa care indică faptul că obiectul este instantiat.

1. **Implementarea si analiza problemei**

Pentru implementarea proiectului avem nevoie de o baza de date si mai trebuie sa cunoastem cum se face interconectarea intre aplicatia java si baza de date.

În acestă aplicaţie am structurat proiectul, după cum urmează: am mai multe pachete, unul din care e pachet responsabil de baza de date,unde avem o clasa pentru conexiunile la baza de date, utilizând limbajul SQL prin conexiune JDBC.

Clasa are un constructor static numit getInstance public () – aici se testeaza daca instanţă este “instantieted” sau nu . Operaţiunile pentru a adăuga, şterge şi actualiza sunt realizate folosind sintaxa SQL - ADD, DELETE şi UPDATE.

În continuare, proiectul are unele interfeţe în designul său.

Avem o interfata principala de unde un utilizator poate alege operaţiunile sale în funcţie de statutul său. Administratorul poate efectua una dintre operaţiunile disponibile: adăugarea un produs, sterge produs sau actualizarea unui produs. Un utilizator obişnuit poate sa listeze produsele din baza de date si sa faca “search” dupa titlul produsului. Deci, avem, de asemenea, interfeţe pentru administrator şi pentru utilizatori, de asemenea.

Clasele din package “**admin\_interf**” :

*AddProduct* – raspunde de adaugarea produselor in baza de date

*DeleteProduct* – raspunde de stergerea produselor din baza de date

*UpdateProduct* – raspunde de modificarea proprietatilor unui produs din baza de date

*Admin\_Page* – panelul administratorului, de unde se pot accessa toate comenzile de mai sus

Clasele din package “**Database**” :

*Connection* – raspunde de legatura aplicatiei cu baza de date si gestionarea query-urilor ce se pot efectua din aplicatie

package Database;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import javax.swing.\*;

**public class Connections**

{

public Connection connection;

**public static Connections instance=null;**

void showError(String error)

{

JOptionPane.showMessageDialog(null,error);

}

**private Connections()**

{

instance=this;

try

{

Class.forName("com.mysql.jdbc.Driver").newInstance();

}

catch (Exception e)

{

showError("Unable to find and load driver");

System.exit(1);

}

try

{

connection = DriverManager.getConnection("jdbc:mysql://localhost/test?user=root&password=root");

System.out.println("Connected to database");

}

catch(SQLException e)

{

showError("DATABASE ERROR!!!");

}

}

**public static Connections getInstance()**

{

if (instance==null)

instance=new Connections();

return instance;

}

**public void add(String t,String a)**

{

try

{

Statement stmt = connection.createStatement();

stmt.executeUpdate("INSERT into products " +"(title, Brand) "+"values ('"+t+"', '"+a+"')") ;

showError("The product was added succesfully!");

}

catch (SQLException ex)

{

showError("Failure in adding the product to database!!!");

}

}

**public void delete\_title(String del)**

{

try

{

Statement stmt = connection.createStatement();

stmt.executeUpdate(" DELETE from products where title ='" +del+ "' limit 30");

showError("The product title has been deleted!");

}

catch (SQLException ex)

{

showError("Failed to delete!");

}

}

**public void delete\_Brand(String del)**

{

try

{

Statement stmt = connection.createStatement();

stmt.executeUpdate("DELETE from products where Brand ='" +del+ "' limit 30");

showError("The product title has been deleted!");

}

catch (SQLException ex)

{

showError("Failed to delete!");

}

}

**public void update\_title(String init, String fin)**

{

try

{

Statement stmt = connection.createStatement();

stmt.executeUpdate(" Update products set " +

" title='"+fin+"' " +

" where title='"+init+"'");

showError("The product's title has been updated!");

}

catch (SQLException ex)

{

showError("Failed to update!");

}

}

**public void update\_Brand(String init, String fin)**

{

try

{

Statement stmt = connection.createStatement();

stmt.executeUpdate(" Update products set " +

" brand='"+fin+"' " +

" where brand='"+init+"'");

showError("The product's brand has been updated!");

}

catch (SQLException ex)

{

showError("Failed to update!!");

}

}

**public int log\_in(String user, String password)**

{

int i=0;

try {

Statement statement = connection.createStatement();

ResultSet rs = statement.executeQuery("SELECT user,password FROM admin"

+ " where user='"+user+"'" + " and password='"+password+"'" );

while (rs.next())

{

i=i+1;

}

}

catch (SQLException ex)

{

showError("DATABASE ERROR!!!");

}

return i;

}

}

Clasele din package “**Products**” :

*Main* – clasa ce raspunde de interfata :

package products;

**public class Main**

{

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

{

start\_interf.Start st;

st = start\_interf.Start.getInstance();

st.setVisible(true);

}

}

*TableModel – clasa responsabila de logica tabelelor si query-urilor*

package products;

import javax.swing.\*;

import javax.swing.table.\*;

import javax.swing.event.\*;

import java.io.\*;

import java.util.\*;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import javax.swing.\*;

**public class** TableModel extends AbstractTableModel

{

public Vector data;

public Vector columnNames ;

//INTITIALIZE THE TABLE

public void initVectors\_admin()

{

data = new Vector();

columnNames = new Vector();

columnNames.addElement("ID");

columnNames.addElement("Title");

columnNames.addElement("Brand");

Object[][] data = {

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null}};

}

//INTITIALIZE THE TABLE

**public void initVectors\_user()**

{

data = new Vector();

columnNames = new Vector();

columnNames.addElement("Title");

columnNames.addElement("Brand");

Object[][] data = {

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null},

{null, null,null}};

}

**public void** setData\_admin()

{

Database.Connections con;

con=Database.Connections.getInstance();

try

{

Statement statement = con.connection.createStatement();

ResultSet rs = statement.executeQuery("SELECT id,title,Brand FROM products");

while (rs.next())

{

String id=(String)(rs.getString(1));

String title=(String)(rs.getString(2));

String Brand=(String)(rs.getString(3));

data.addElement(id);

data.addElement(title);

data.addElement(Brand);

}

}

catch (SQLException excep)

{

showError("DATABASE ERROR!!!");

}

}

**public void** setData\_user()

{

Database.Connections con;

con=Database.Connections.getInstance();

try

{

Statement statement = con.connection.createStatement();

ResultSet rs = statement.executeQuery("SELECT title,Brand FROM products");

while (rs.next())

{

String title=(String)(rs.getString(1));

String Brand=(String)(rs.getString(2));

data.addElement(title);

data.addElement(Brand);

}

}

catch (SQLException excep)

{

showError("DATABASE ERROR!!!");

}

}

**public int** search\_Brand(String a)

{

Database.Connections con;

con=Database.Connections.getInstance();

int i=0;

try

{

Statement statement = con.connection.createStatement();

ResultSet rs = statement.executeQuery("SELECT title,Brand FROM products"+ " where Brand='"+a+"'");

while (rs.next())

{

String title=(String)(rs.getString(1));

String Brand=(String)(rs.getString(2));

data.addElement(title);

data.addElement(Brand);

i=i+1;

}

}

catch (SQLException excep)

{

showError("DATABASE ERROR!!!");

}

return i;

}

**public int** search\_title(String t)

{

Database.Connections con;

con=Database.Connections.getInstance();

int i=0;

try

{

Statement statement = con.connection.createStatement();

ResultSet rs = statement.executeQuery("SELECT title,Brand FROM products"+ " where title='"+t+"'");

while (rs.next())

{

String title=(String)(rs.getString(1));

String Brand=(String)(rs.getString(2));

data.addElement(title);

data.addElement(Brand);

i=i+1;

}

}

catch (SQLException excep)

{

showError("DATABASE ERROR!!!");

}

return i;

}

void showError(String error)

{

JOptionPane.showMessageDialog(null,error);

}

**public int** getRowCount() {

return data.size()/ getColumnCount();

}

**public int** getColumnCount(){

return columnNames.size();

}

public String getColumnName(int columnIndex) {

String colName = "";

if (columnIndex <= getColumnCount())

colName = (String)columnNames.elementAt(columnIndex);

return colName;

}

**public Class** getColumnClass(int columnIndex){

return String.class;

}

public boolean isCellEditable(int rowIndex, int columnIndex) {

return false;

}

public Object getValueAt(int rowIndex, int columnIndex) {

return (String)data.elementAt

( (rowIndex \* getColumnCount()) + columnIndex);

}

**public void** setValueAt(Object aValue, int rowIndex, int columnIndex) {

return;

}

}

Clasele din package “**start\_interf**” :

*Start* – raspunde de interfata principala GUI

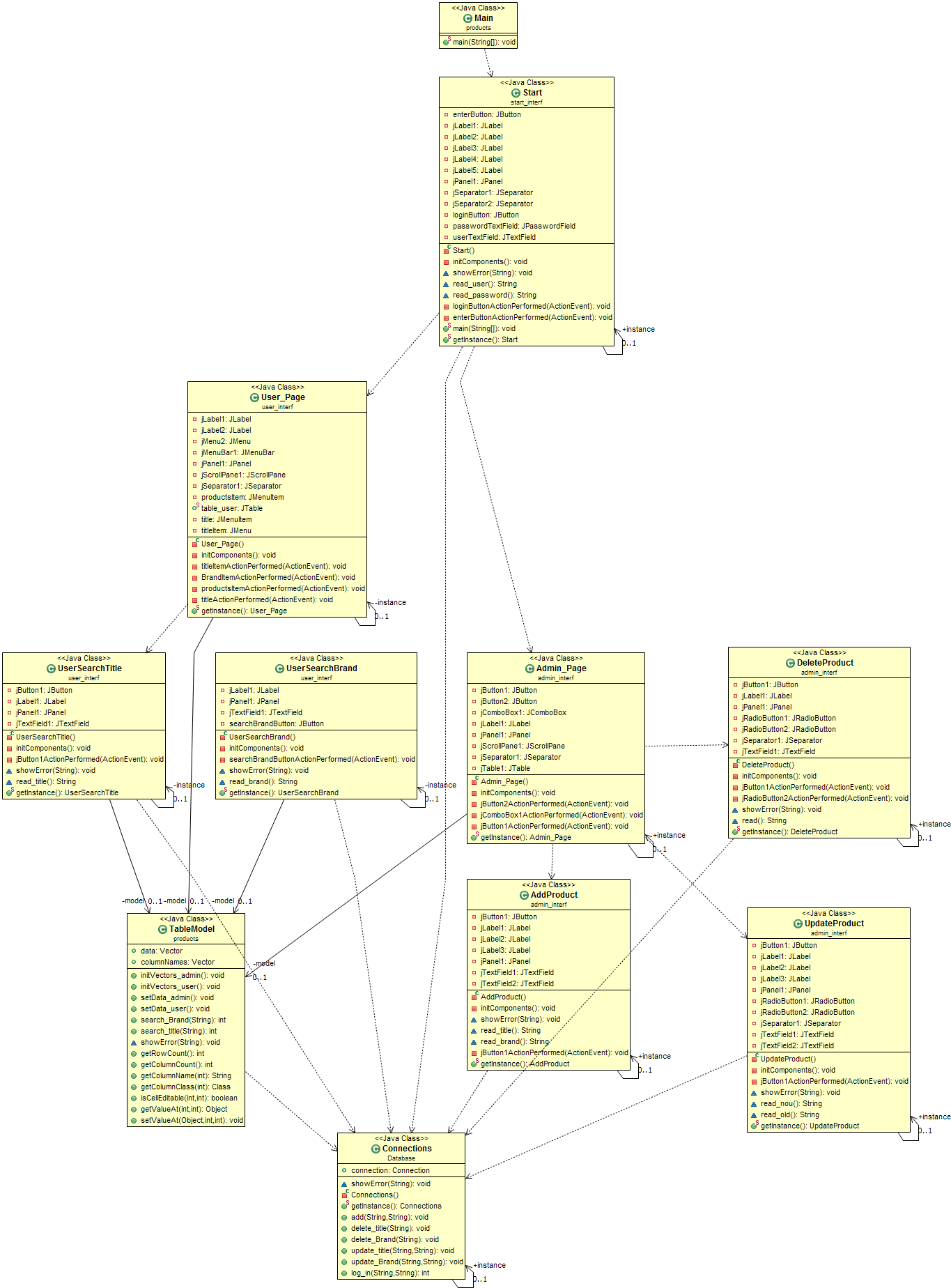
Clasele din package “**user\_interf**” :

*User\_Page – raspunde de pagina utilizatorului simplu*

*UserSearchBrand – raspunde de cautarea dupa brand*

*UserSearchTitle – raspunde de cautarea dupa titlu*

1. **Diagrama UML**

****

**4. Baza de date**

O **bază de date**, uneori numită şi „bancă de date”, reprezintă o modalitate de stocare a unor [informaţii](http://ro.wikipedia.org/wiki/Informa%C8%9Bie) şi [date](http://ro.wikipedia.org/wiki/Dat%C4%83) pe un suport extern (un dispozitiv de stocare), cu posibilitatea extinderii uşoare şi a regăsirii rapide a acestora. La prima vedere sarcina poate părea banală. Totuşi, în condiţiile în care este vorba de a lucra cu milioane de elemente, fiecare putând consta din mari cantităţi de date care trebuie accesate simultan prin [Internet](http://ro.wikipedia.org/wiki/Internet) de către mii de utilizatori răspândiţi pe întreg globul; şi în condiţiile când disponibilitatea aplicaţiei şi datelor trebuie să fie permanentă (de ex. pentru a nu pierde ocazia de a încheia afaceri), soluţiile bune nu sunt de loc simple.

De obicei o bază de date este memorată într-unul sau mai multe [fişiere](http://ro.wikipedia.org/wiki/Fi%C8%99ier). Bazele de date sunt manipulate cu ajutorul [sistemelor de gestiune a bazelor de date](http://ro.wikipedia.org/wiki/Sistem_de_gestiune_a_bazelor_de_date).

Cel mai răspândit tip de baze de date este cel relaţional, în care datele sunt memorate în [tabele](http://ro.wikipedia.org/wiki/Tabel). Pe lânga tabele, o bază de date relaţională mai poate conţine: indecşi, proceduri stocate, declanşatori, utilizatori şi grupuri de utilizatori, tipuri de date, mecanisme de securitate şi de gestiune a tranzacţiilor etc.

In ceea ce priveste partea de baze de date pentru punerea în aplicare am folosit MySQL Server

Am lucrat cu serverul Apache şi am pus în aplicare o bază de date care conţine date necesare pentru a rula corect aplicatia mea: există un tabel - produse şi altul admin.

In tabelul Produse avem 3 campuri id,title,brand. Campul ID se incrementeaza de fiecare data cand e adugat in nou produs in stoc.Campurile title si brand raspund de specificatiile produsului.Tabelul Admin e destinat pentru administratorul de stoc si contine doua campuri – user si password.

Produse

ADMIN

user

password

id

title

brand

In proiect am folosit libraria MySQL Connector pentru a face conexiune la baza de date.

MySQL Connector este un driver nativ Java, care converteşte JDBC (Java Database Connectivity) în protocolul de reţea utilizat de baze de date MySQL.

Aceasta permite dezvoltatorilor care lucreaza cu limbajul de programare Java sa construiasca cu uşurinţă programe şi applet-uri care interacţionează cu MySQL şi sa interconecteze toate datele corporative, chiar şi într-un mediu eterogen.

**5. Testarea**

Pentru a testa aplicatia avem nevoie de MySQL si Apache+PHP pentru a vedea cum se modifica baza de date si in caz de apar anumite erori sa le putem vedea mai usor.

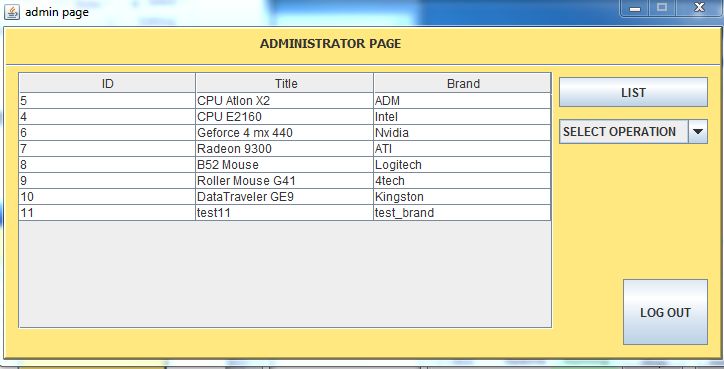
La pornirea aplicatiei noi trebuie sa alegem o actiune. Sau intram in “stoc” ca administrator sau ca un simplu utilizator.



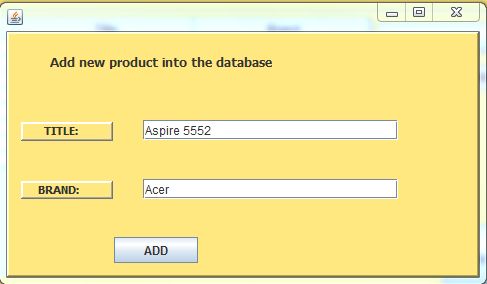
In cazul in care vrem sa ne logam ca administrator trebuie sa introducem username si parola, daca nu am introdus corect date sau nu avem access la baza de date programul ne va afisa o eroare



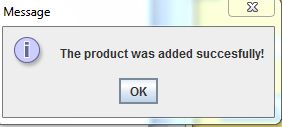
In cazul in care am introdus corect parola si usernameul, noi avem access in panelul administratorului



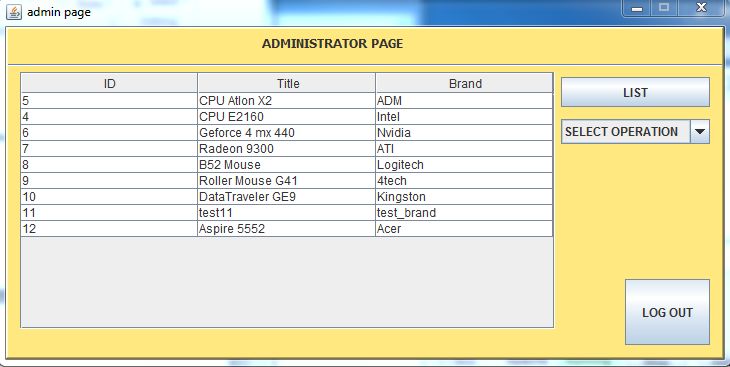
Aici avem optiunea LIST care ne arata continul tabelului “Products”, si mai avem 3 operatii ADD,DELETE si UPDATE. Spre exemplu vrem sa adaugam un produs in baza de date atunci apasam ADD din “select operation” si introducem datele produsul care va fi adugat :



Apasand butonul ADD noi adugam produsul in baza de date, si ne apare un mesaj de avertizare



Pentru a trece iar in meniul principal se utilizeaza butonul LOG OUT sau LIST



Daca vrem sa accesam interfata unui utilizator simplu (client) atunci in interfata principala apasam butonul ENTER si ne apare o fereseatra unde putem face “LIST” sa cauta produsul in baza de date dupa “titlu” in meniul SEARCH.

