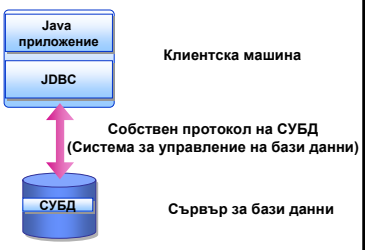
Достъп до бази данни (БД)

1. **JDBC** (Java Database Connectivity) е множество от класове и интерфейси (пакет java.sql), чрез които приложенията четат и обновяват информация в бази данни и други източници на данни.
   - установяват връзка с източника на данни;
   - създават оператор;
   - изпращат операторите за търсене и обновяване към източника на данни;
   - обработват резултатите.

2. **JDBC API** се използват за директно извикване на SQL команди.
   - SQLJ препроцесорът ефективно транслира смесицата от Java/SQL в програмния език Java с извиквания на JDBC;
   - директно преобразува таблици от релационната бази данни в Java класове:
     - таблицата става клас;
     - всеки ред от таблицата става инстанция на този клас;
     - стойността във всеки стълб съответства на атрибут на инстанцията.
3. **JDBC-ODBC Bridge** използва **ODBC** (Open DataBase Connectivity) за достъп до релационни бази данни, като осигурява връзка към почти всички видове бази данни за почти всички платформи.

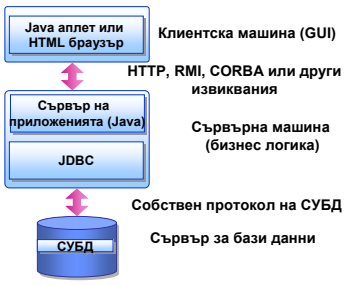
JDBC архитектури

1. Двуслойна архитектура – конфигурация клиент/сървър; изисква JDBC драйвер



Java аплет/приложение комуникира директно с източника на данни. Потребителските команди се изпращат към базата данни и резултатите от операторите се изпращат обратно към потребителя. Източникът на данни може да бъде разположен на друга машина, към която потребителят се свързва през мрежата.

2. Трислойна архитектура

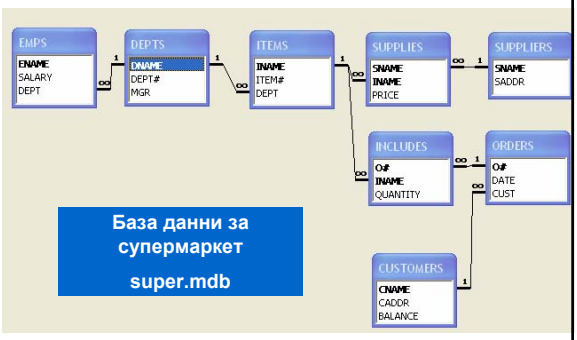


Командите се изпращат към средния слой (бизнес логика), който ги изпраща към източника на данни.

Източникът на данни обработва командите и връща обратно резултатите към средния слой, който ги изпраща към потребителя.

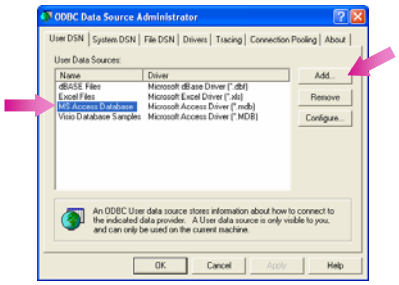
Трислойната архитектура осигурява предимства.

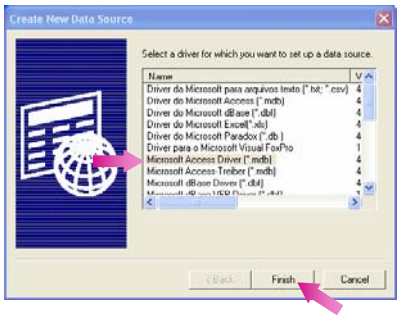
Примерна база данни

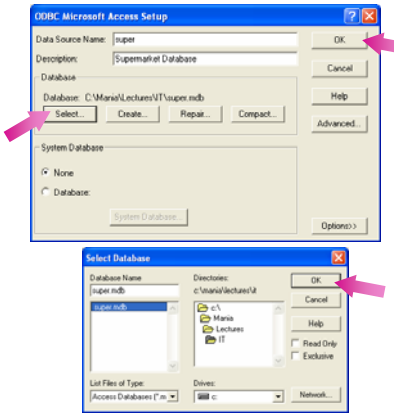


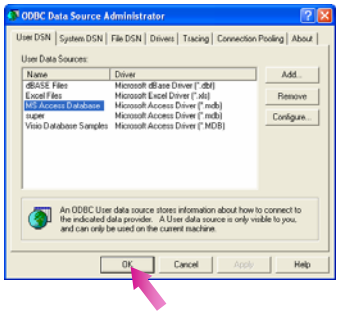
Конфигуриране на базата данни

Start ⇒ Settings ⇒ Control Panel ⇒ Administrative Tools ⇒ Data Sources (ODBC)

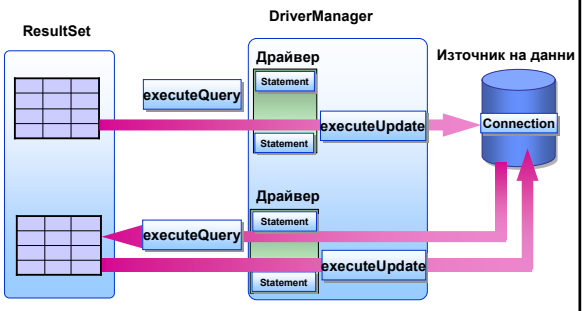








Обектен модел на JDBC



I. Клас DriverManager **– управлява динамично всички обекти на драйвера**

– зарежда драйвера JDBC-ODBC Bridge;

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

– осъществява връзката

```
Connection con = DriverManager.getConnection
        (String url, String user, String password);
```

URL на БД | потребител на БД | потребителска парола

протокол | подпротокол | идентификатор на БД

```
String url = "jdbc:odbc:super";
String user = "";
String password = "";
```

Пример: Тестване на конфигурацията

пакет java.sql

```
import java.sql.*;
class JDBCTest {
  public JDBCTest () {
    try {
      Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch (ClassNotFoundException e) {
      System.err.println ("ClassNotFoundException: " + e.getMessage());
    }
    try {
      Connection con = DriverManager.getConnection("jdbc:odbc:super", "", "");
    }
    catch (SQLException ex) {
      System.err.println ("SQLException: " + ex.getMessage());
    }
  }
}
```

зарежда драйвера

осъществява връзката

II. Statement **обекти – използват се за изпращане на SQL оператори към БД.**

1. Видове Statement **обекти:**

– Statement **– изпълнява прост SQL оператор без параметри;**

– PreparedStatement **(наследник на** Statement**) – изпълнява предварително компилиран SQL оператор с или без входни параметри;**

– CallableStatement **(наследник на** PreparedStatement**) – изпълнява извикване към съхранени процедури на БД.**

2. Създаване на Statement **обекти**

```
Statement s = con.createStatement ();
```

Connection.createStatement създава Statement обект за изпращане на SQL оператори към БД.

3. Изпълнение на оператори, използвайки Statement **обекти**

– executeQuery **– изпълнява дадения SQL оператор** sql**, който връща единствен** ResultSet **обект;**

```
public ResultSet executeQuery (String sql);
```

SQL оператор (обикновено оператор SELECT)

– updateQuery **– изпълнява дадения SQL оператор** sql**, който може да бъде INSERT, UPDATE или DELETE оператор, или SQL оператор, който не връща нищо (напр. като SQL оператор за дефиниране на данни).**

```
public int executeUpdate (String sql);
```

SQL оператор (обикновено INSERT, UPDATE, DELETE или за дефиниране на данни)

4. Клас ResultSet **– съдържа резултатите от изпълнението на SQL заявка – съдържа редове, които удовлетворяват условията на заявката.**

– **метод** getXXX **– връща стойността на дадения стълб в текущия ред;**

```
public XXX getXXX (String columnName);
```

или

```
public XXX getXXX (int columnNumber);
```

където XXX: int, float, …

– **метод** next **– премества курсора на следващия ред в** ResultSet **и той става текущият ред;**

```
public boolean next ();
```

– **курсорът сочи към текущия ред с данни. При първоначалното създаване на** ResultSet **обектът курсорът се позиционира** <u>**преди**</u> **първия ред.**

Пример: Изпълнява SQL SELECT оператор, който връща колекция от редове със стълбове: ENAME от тип String, SALARY от тип double и DEPT от тип String.

```
Statement s = con.createStatement ();          // създава Statement обект
ResultSet rs = s.executeQuery("SELECT * FROM EMPS");   // изпълнява SQL оператор
System.out.println ("Employee Name\tSalary\tDepartment");

while (rs.next()) {
  String ename = rs.getString ("ENAME");
  double salary = rs.getDouble ("SALARY");
  String dept   = rs.getString ("DEPT");          // връща стойностите в стълбовете
  System.out.println(ename + "\t" + salary + "\t" + dept);
}
```

Example: Изпълнява SQL INSERT оператор, който добавя ред в таблицата EMPS.

```
Statement s = con.createStatement ();          // създава Statement обект
s.executeUpdate("INSERT INTO EMPS "+           // изпълнява SQL оператор
        "VALUES ('Mariana Goranova', 5000.0, 'Meat')");
ResultSet rs = s.executeQuery("SELECT * FROM EMPS");   // извлича резултатите
System.out.println ("Employee Name\tSalary\tDepartment");
while (rs.next()) {
  String ename = rs.getString ("ENAME");
  double salary = rs.getDouble ("SALARY");
  String dept   = rs.getString ("DEPT");
  System.out.println(ename + "\t" + salary + "\t" + dept);
}
```

Пример: Изпълнява SQL UPDATE оператор, който обновява ред в таблицата EMPS.

```
Statement s = con.createStatement ();          // създава Statement обект
s.executeUpdate("UPDATE EMPS SET SALARY=4000.0 " +   // изпълнява SQL оператор
          "WHERE ENAME='Mariana Goranova'");
ResultSet rs = s.executeQuery("SELECT * FROM EMPS");   // извлича резултатите
System.out.println ("Employee Name\tSalary\tDepartment");
while (rs.next()) {
  String ename = rs.getString ("ENAME");
  double salary = rs.getDouble ("SALARY");
  String dept   = rs.getString ("DEPT");
  System.out.println(ename + "\t" + salary + "\t" + dept);
}
```

Пример: Изпълнява SQL DELETE оператор, който изтрива ред в таблицата EMPS.

```
Statement s = con.createStatement ();          // създава Statement обект
s.executeUpdate("DELETE * FROM EMPS " +        // изпълнява SQL оператор
          "WHERE ENAME='Mariana Goranova'");
ResultSet rs = s.executeQuery("SELECT * FROM EMPS");   // извлича резултатите
System.out.println ("Employee Name\tSalary\tDepartment");
while (rs.next()) {
  String ename = rs.getString ("ENAME");
  double salary = rs.getDouble ("SALARY");
  String dept   = rs.getString ("DEPT");
  System.out.println(ename + "\t" + salary + "\t" + dept);
}
```

Пример: Определя броя на редовете в ResultSet.

```
Statement s = con.createStatement ();
ResultSet rs = s.executeQuery("SELECT * FROM EMPS");
int count = 0;
while (rs.next()) {
  count++;
}
System.out.println("The supermarket has " + count + " employees");
```

или

```
ResultSet rs = s.executeQuery("SELECT COUNT(*) FROM EMPS");
rs.next();
int count = rs.getInt(1);
System.out.println("The supermarket has " + count + " employees");
```

5. Обект PreparedStatement – съдържа SQL оператор, който вече е компилиран; има входни параметри като въпросителни знаци ("?").

- **създаване на PreparedStatement обекти;**

```
PreparedStatement s = con.prepareStatement ();
```

Connection.prepareStatement създава PreparedStatement обект за изпращане на параметризирани SQL оператори към БД.

- **предаване на входни параметри.**

```
public void setXXX (int parameterIndex, XXX x);
```

Пример: Използва PreparedStatement**, за да създаде** ResultSet**.**

```
PreparedStatement updateItems = con.prepareStatement
  ("UPDATE SUPPLIES SET PRICE=? WHERE SNAME=? AND INAME=?");
updateItems.setDouble (1, 5.5);
updateItems.setString (2, "Metro");
updateItems.setString (3, "Brie");
updateItems.executeUpdate ();

Statement s = con.createStatement ();
ResultSet rs = s.executeQuery("SELECT * FROM SUPPLIES");
System.out.println ("Supplier Name\tItem Name\tPrice");
while (rs.next()) {
  String sname = rs.getString ("SNAME");
  String iname  = rs.getString ("INAME");
  double price  = rs.getDouble ("PRICE");
  System.out.println(sname+ "\t\t" + iname + "\t\t" + price);
}
```

създава PreparedStatement обект

предава входни параметри

изпълнява SQL оператор

6. Обект ResultSetMetaData **– дава информация за типовете и свойствата на стълбовете в** ResultSet **обект.**

– **създава** ResultSetMetaData **обект;**

```
public ResultSetMetaData getMetaData ();
```

ResultSet.getMetaData връща ResultSetMetaData обект

– **извлича броя на стълбовете;**

```
public int getColumnCount ();
```

връща броя на стълбовете в ResultSet обект

– **връща името на стълб с даден номер** column**;**

```
public String getColumnName (int column);
```

Първият стълб е 1

– **връща името на таблицата със стълб с даден номер** column**.**

```
public String getTableName (int column);
```

Първият стълб е 1

Пример: Извлича информация от ResultSet **обект, използвайки** ResultSetMetaData**.**

```
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection ("jdbc:odbc:super", "", "");

Statement s = con.createStatement();
ResultSet set = s.executeQuery ("SELECT * FROM EMPS");

ResultSetMetaData r = set.getMetaData();

System.out.println (r.getTableName (1));

int columns = r.getColumnCount();
for (int i=1; i <= columns; i++)
  System.out.print (r.getColumnName(i) + "\t\t");
System.out.println ();
```

връзка

създава Statement и изпраща заявка

създава ResultSetMetaData обект

връща името на таблицата

връща броя на стълбовете

връща името на стълба

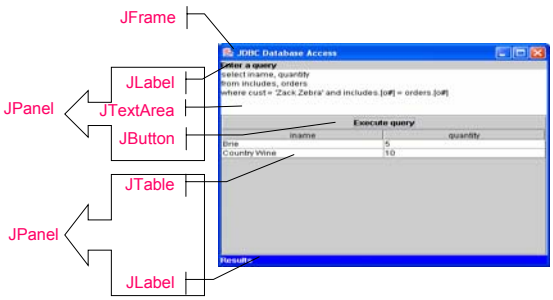
```
int row = 0;
while (set.next())
  row++;
```

изчислява броя на редовете

```
set = s.executeQuery (query);
row = 0;
while (set.next()) {
  for (int i=1; i <= columns; i++)
    System.out.print (set.getString(i)+"\t");
  System.out.println ();
  row++;
}
```

обработва резултатите

Пример: Изпълнява SQL заявки



```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import javax.swing.text.*;
import java.sql.*;
class JDBCwithSwing extends JFrame {
  String url = "jdbc:odbc:super";
  String user = "";
  String password = "";
  Connection con;
  Statement s;
  String query;
  JPanel p1, p2;
  JButton button;
  JLabel l1, l2;
  JTextArea search;
  JTable results;
  JScrollPane sp;
```

```
  public JDBCwithSwing (String title) {
    super (title);
    getContentPane().setLayout (new BorderLayout());
    p1 = new JPanel();
    p1.setLayout (new BorderLayout());
    l1 = new JLabel();
    l1.setText ("Enter a query");
    p1.add (l1, BorderLayout.NORTH);
    search = new JTextArea (5, 40);
    p1.add (search, BorderLayout.CENTER);
    button = new JButton ("Execute query");
    button.addActionListener (new Handler());
    p1.add (button, BorderLayout.SOUTH);
    getContentPane().add (p1, BorderLayout.NORTH);
    p2 = new JPanel();
    p2.setBackground(Color.BLUE);
    l2 = new JLabel();
    l2.setForeground(Color.WHITE);
    p2.setLayout (new BorderLayout());
    p2.add (l2, BorderLayout.NORTH);
    getContentPane().add (p2, BorderLayout.SOUTH);
```

```
    try {
      // зарежда драйвера
      Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
      // установява връзка
      con = DriverManager.getConnection (url, user, password);
      l2.setText("Starting JDBC...");
      s = con.createStatement();
    }
    catch (Exception e) {
      l2.setText (e.getMessage());
    }
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        dispose();
        System.exit(0);
      }
    });
  }
```

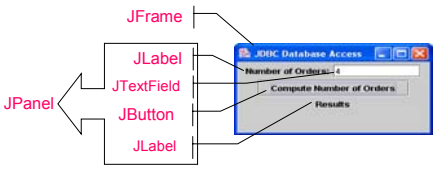
```
  class Handler implements ActionListener {
    public void actionPerformed (ActionEvent e) {
      ResultSet set;
      ResultSetMetaData r;
      l2.setText("");
      if (search.getText().length() == 0) {
        l1.setText("Please, enter a query");
        return;
      }
      query = search.getText();
      try {
        if (query.toUpperCase().startsWith("SELECT")) {
          set = s.executeQuery (query);
        }
        else {
          s.executeUpdate (query);
          return;
        }
        l2.setText ("Results");
        r = set.getMetaData();
```

```
        int columns = r.getColumnCount();
        String [] columnNames = new String[columns];
        for (int i=1; i <= columns; i++)
          columnNames[i-1] = r.getColumnName(i);
        int row = 0;
        while (set.next()) {
          row++;
        }
        Object[][] rows = new Object[row][columns];
        con = DriverManager.getConnection (url, user, password);
        s = con.createStatement();
        set = s.executeQuery (query);
        row=0;
        while (set.next()) {
          for (int i=1; i <= columns; i++)
            rows[row][i-1] = set.getString(i);
          row++;
        }
```

```
        results = new JTable (rows, columnNames);
        sp = new JScrollPane (results);
        results.setPreferredScrollableViewportSize (new Dimension (300, 200));
        getContentPane().add(sp, BorderLayout.CENTER);
        pack();
      }
      catch (Exception ex) {
        l2.setText (ex.getMessage());
      }
    }
  }

  public static void main(String args[]) {
    JDBCwithSwing mainFrame = new JDBCwithSwing
                                        ("JDBC Database Access");
    mainFrame.setSize(400, 300);
    mainFrame.setVisible(true);
  }
}
```

Пример: Просто свързване на данни – използва стойност от източник на данни




```
ResultSet set = s.executeQuery
      ("SELECT COUNT(*) AS NumberOfOrders FROM ORDERS");
set.next();
int n = set.getInt (1);
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import javax.swing.text.*;
import java.sql.*;
class JDBCwithSwing extends JFrame {
  String url = "jdbc:odbc:super";
  String user = "";
  String password = "";
  Connection con;
  Statement s;
  String query;
  JPanel p;
  JButton button;
  JLabel l1, l2;
  JTextField results;
```

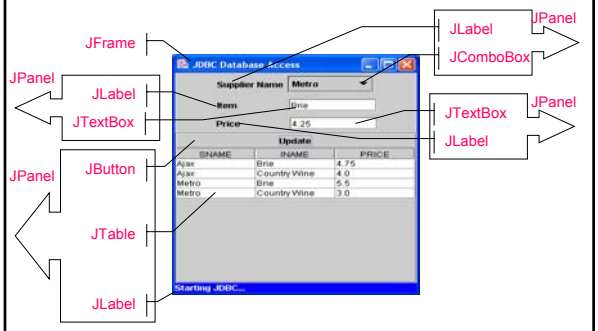
```
public JDBCwithSwing (String title) {
  super (title);
  getContentPane().setLayout(new BorderLayout());
  p = new JPanel();
  l1 = new JLabel();
  getContentPane().add(p);
  l1.setText ("Number of Orders:");
  p.add (l1);
  results = new JTextField (10);
  p.add (results);
  button = new JButton ("Compute Number of Orders");
  button.addActionListener (new Handler());
  p.add (button);
  l2 = new JLabel("");
  p.add (l2);
```

```
  try {
    // зарежда драйвера
    Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
    // установява връзката
    con = DriverManager.getConnection (url, user, password);
    l2.setText("Starting JDBC...");
    s = con.createStatement();
  }
  catch (Exception e) {
    l2.setText (e.getMessage());
  }
  addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
      dispose();
      System.exit(0);
    }
  });
}
```

```
class Handler implements ActionListener {
  public void actionPerformed (ActionEvent e) {
    ResultSet set;
    ResultSetMetaData r;
    query = "SELECT COUNT(*) AS NumberOfOrders FROM ORDERS";
    try {
      set = s.executeQuery (query);
      set.next();
      String n = set.getString(1);
      l2.setText ("Results");
      results.setText(n);
    }
    catch (Exception ex) {
      l2.setText (ex.getMessage());
    }
  }
}
```

```
public static void main(String args[]) {
  JDBCwithSwing mainFrame = new JDBCwithSwing
                                ("JDBC Database Access");
  mainFrame.setSize(250, 150);
  mainFrame.setVisible(true);
}
}
```

Пример: Използва предварително компилиран оператор




```
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import javax.swing.text.*;
import java.sql.*;
class JDBCwithSwing extends JFrame {
  String url = "jdbc:odbc:super";
  String user = "";
  String password = "";
  Connection con;
  Statement s;
  String query;
  String updateString;
  PreparedStatement updateItems;
  JPanel p, p1, p2, p3, p4;
  JButton button;
  JLabel l1, l2, l3, l4;
  JTextField item;
```

```
  JTextField price;
  JComboBox search;
  JTable results;
  JScrollPane sp1;
  JScrollPane sp2;
  Vector list;
  String supplierName;
  String itemSearch;
  double priceNew;
  public JDBCwithSwing (String title) {
    super (title);
    getContentPane().setLayout (new BorderLayout());
    p = new JPanel ();
    p.setLayout (new BorderLayout());
    getContentPane().add (p, BorderLayout.NORTH);
```

```
    p1 = new JPanel();
    l1 = new JLabel("Supplier Name");
    p1.add (l1);
    sp1 = new JScrollPane ();
    list = new Vector();
    list.addElement ("Acme ");
    list.addElement ("Ajax");
    list.addElement ("Metro");
    search = new JComboBox (list);
    search.addActionListener (new Handler1());
    sp1.getViewport ().add (search);
    p1.add (sp1);
    p.add (p1, BorderLayout.NORTH);
    p2 = new JPanel();
    l2 = new JLabel (" Item ");
    p2.add (l2);
    item = new JTextField (10);
    item.addActionListener (new Handler2());
    p2.add (item);
    p.add (p2, BorderLayout.CENTER);
```

```
    p3 = new JPanel();
    l3 = new JLabel (" Price ");
    p3.add (l3);
    price = new JTextField (10);
    price.addActionListener (new Handler3());
    p3.add (price);
    p.add (p3, BorderLayout.SOUTH);
    p4 = new JPanel();
    p4.setBackground (Color.BLUE);
    p4.setLayout (new BorderLayout());
    l4 = new JLabel();
    l4.setForeground(Color.WHITE);
    p4.add (l4, BorderLayout.SOUTH);
    button = new JButton ("Update");
    button.addActionListener (new Handler4());
    p4.add (button, BorderLayout.NORTH);
    getContentPane().add (p4, BorderLayout.CENTER);
```

```
    try {
      // зарежда драйвера
      Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
      // установява връзката
      con = DriverManager.getConnection (url, user, password);
      l4.setText("Starting JDBC...");
    }
    catch (Exception e) {
      l2.setText (e.getMessage());
    }
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        dispose();
        System.exit(0);
      }
    });
  }
```

```
class Handler1 implements ActionListener {
  public void actionPerformed (ActionEvent e) {
    supplierName = (String)search.getSelectedItem();
  }
}
class Handler2 implements ActionListener {
  public void actionPerformed (ActionEvent e) {
    itemSearch = item.getText();
  }
}
class Handler3 implements ActionListener {
  public void actionPerformed (ActionEvent e) {
    priceNew = new Double (price.getText()).doubleValue();
  }
}
```

```
class Handler4 implements ActionListener {
  public void actionPerformed (ActionEvent e) {
    ResultSet set;
    ResultSetMetaData r;
    try {
      updateString = "UPDATE SUPPLIES " +
                     "SET PRICE = ? WHERE SNAME = ? AND INAME = ?";
      updateItems = con.prepareStatement (updateString);
      updateItems.setDouble (1, priceNew);
      updateItems.setString (2, supplierName);
      updateItems.setString (3, itemSearch);
      updateItems.executeUpdate();
      query = "SELECT * FROM SUPPLIES";
      s = con.createStatement ();
      set = s.executeQuery (query);
      r = set.getMetaData();
      int columns = r.getColumnCount();
      String [] columnNames = new String[columns];
      for (int i=1; i <= columns; i++)
        columnNames[i-1] = r.getColumnName(i);
```

```
      int row = 0;
      while (set.next()) {
        row++;
      }
      Object[][] rows = new Object[row][columns];
      set = s.executeQuery (query);
      row=0;
      while (set.next()) {
        for (int i=1; i <= columns; i++)
          rows[row][i-1] = set.getString(i);
        row++;
      }
      results = new JTable (rows, columnNames);
      sp2 = new JScrollPane (results);
      results.setPreferredScrollableViewportSize (new Dimension (300, 200));
      p4.add(sp2, BorderLayout.CENTER);
      pack();
    }
    catch (Exception ex) {
      l4.setText (ex.getMessage());
    }
  }
}
```

```
public static void main(String args[]) {
  JDBCwithSwing mainFrame = new JDBCwithSwing
                                ("JDBC Database Access");
  mainFrame.setSize(400, 300);
  mainFrame.setVisible(true);
 }
}
```