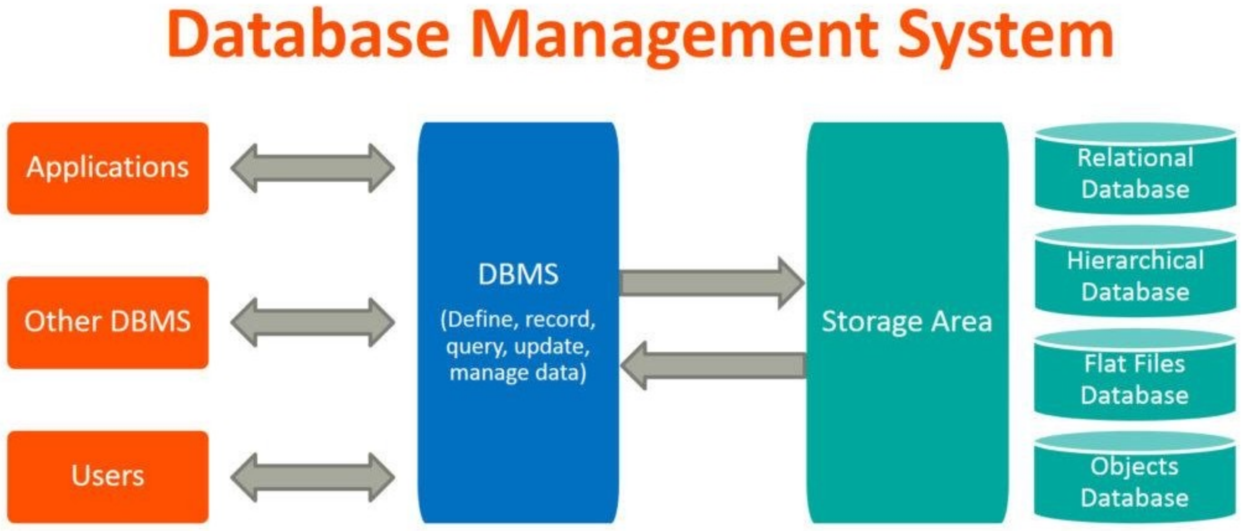
COMP 1618
Lecture 07

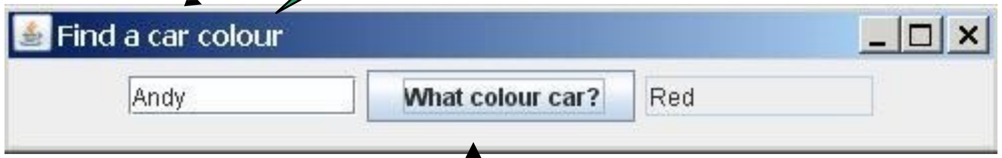
Java Database Programming

Lecture Objectives

- This lecture shows how to write Java programs to interact with Database:
  - Architecture
  - Connectivity
  - SQL query
  - Examples

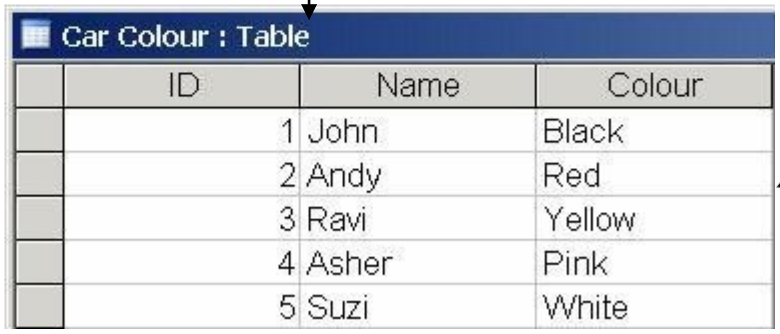


Java application making it easy to access the database



Andy	Red

Application Program e.g. Java



Car Colour : Table

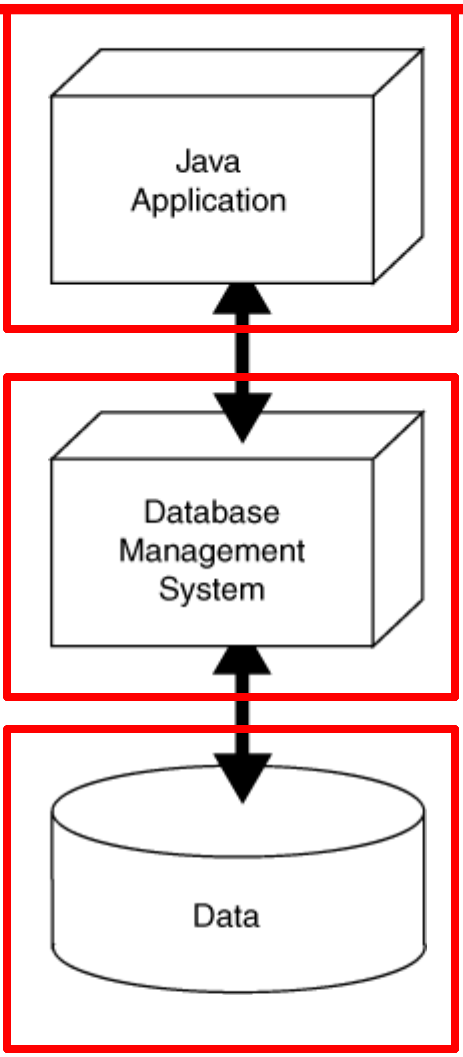
ID	Name	Colour
1	John	Black
2	Andy	Red
3	Ravi	Yellow
4	Asher	Pink
5	Suzi	White

database

Architecture

The Application sends a command to the DBMS

The DBMS executes the command on the Data



The Application displays the result to the user

The DBMS sends the result back to the Application

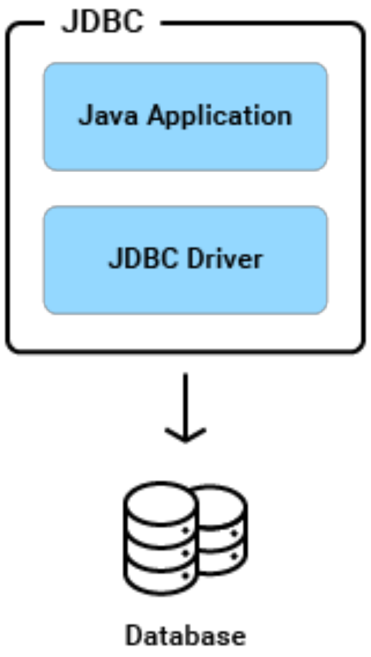
SQL Language

- **SQL** means *structured query language*
  - A standard language for working with database management systems
  - Statements or queries are strings passed from the application to the DBMS using API method calls

JDBC API and Drivers

- The Java Database Connectivity **(JDBC) API** provides universal data access from the Java programming language. Using the JDBC API, you can access any relational databases

- **JDBC drivers** are middleware translating the Java into specific commands that a particular type of database can understand.



Using JDBC in Java applications

- The JDBC API is comprised of two packages:
  - java.sql
  - javax.sql

- Using JDBC in a Java application requires the following steps:
  1. Build a connection to the database
  2. Pass SQL statements to the DBMS
  3. Send SQL results (as a result set)back
  4. Close the connection when finished

the UNIVERSITY of GREENWICH

Developing JDBC Programs

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Statement to load a driver:

Class.forName(“**Use the JDBC Driver Class, see below**");

A driver is a class. For example:

Database	Driver Class	Source
Access	sun.jdbc.odbc.JdbcOdbcDriver	Already in JDK 7
MySQL	com.mysql.jdbc.Driver	Website
Oracle	oracle.jdbc.driver.OracleDriver	Website

The JDBC-ODBC driver for Access is bundled in JDK.

MySQL driver class is in mysqljdbc.jar

Oracle driver class is in classes12.jar

To use the MySQL and Oracle drivers, you have to add mysqljdbc.jar and classes12.jar in the classpath using the following DOS command on Windows:

```
classpath=%classpath%;c:\book\mysqljdbc.jar;c:\book\classes12.jar
```

Establishing connection

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

The static `DriverManager.getConnection` method is used to get a connection to the database

- General format of the simplest version:

```
DriverManager.getConnection(DatabaseURL)
;
```

- General format if *username* and *password* are required:

```
DriverManager.getConnection(DatabaseURL,
                            Username,
                            Password);
```

-

Establishing connection

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Connection connection = DriverManager.getConnection(databaseURL);

Database	**URL Pattern**
Access	dataSource
MySQL	jdbc:mysql://hostname/dbname
Oracle	jdbc:oracle:thin:@hostname:port#:oracleDBSID

Examples:

For Access:

```
Connection connection = DriverManager.getConnection
  (sourceURL, "admin", "");
```

For MySQL:

```
Connection connection = DriverManager.getConnection
  ("jdbc:mysql://localhost/test");
```

For Oracle:

```
Connection connection = DriverManager.getConnection
  ("jdbc:oracle:thin:@liang.armstrong.edu:1521:orcl", "scott", "tiger");
```

Executing statements

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Creating statement:

```
    Statement statement = connection.createStatement();
```

Executing statement (for update, delete, insert):

```
    statement.executeUpdate
      ("create table Temp (col1 char(5), col2
      char(5))");
```

Executing statement (for select):

```
    // Select the columns from the Student table
    ResultSet resultSet = statement.executeQuery
     ("select firstName, mi, lastName from Student where lastName "
       + " = 'Smith'");
```

Executing Statements

- A SQL update statement can be executed using
  `executeUpdate(String sql)`

- For example, the following code executes the SQL statement
  create table Temp (col1 char(5), col2 char(5))

  statement.executeUpdate (**"create table Temp (col1 char(5), col2 char(5))"**);

- A SQL query statement can be executed using **`executeQuery(String sql)`**. The result of the query is returned in **ResultSet**.

The next code executes the SQL query

ResultSet resultSet = statement.executeQuery (**"select firstName, mi, lastName from Student where lastName " + " = 'Smith'"**);

Processing ResultSet

The **ResultSet** maintains a table whose current row can be retrieved. The initial row position is **null**. You can use the **next** method to move to the next row and the various get methods to retrieve values from a current row. For example, the code given below displays all the results from the preceding SQL query.

```
// Iterate through the result and print the student names
    while (resultSet.next())
    System.out.println(resultSet.getString(1) + " " +
resultSet.getString(2) + " " + resultSet.getString(3));
```

The **getString(1)**, **getString(2)**, and **getString(3)** methods retrieve the column values for **firstName**, **mi**, and **lastName**, respectively. Alternatively, you can use **getString("firstName")**, **getString("mi")**, and **getString("lastName")** to retrieve the same three column values. The first execution of the **next()** method sets the current row to the first row in the result set, and subsequent invocations of the **next()** method set the current row to the second row, third row, and so on, to the last row.

Developing JDBC Programs

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Executing statement (for select):

```
// Select the columns from the Student table
ResultSet resultSet = stmt.executeQuery
  ("select firstName, mi, lastName from Student where lastName "
    + " = 'Smith'");
```

Processing ResultSet (for select):

```
// Iterate through the result and print the student names
while (resultSet.next())
System.out.println(resultSet.getString(1) + " "
                                   + resultSet.getString(2)
                                          + "  "
                                             + resultSet.getString(3));
```

Simple JDBC Example

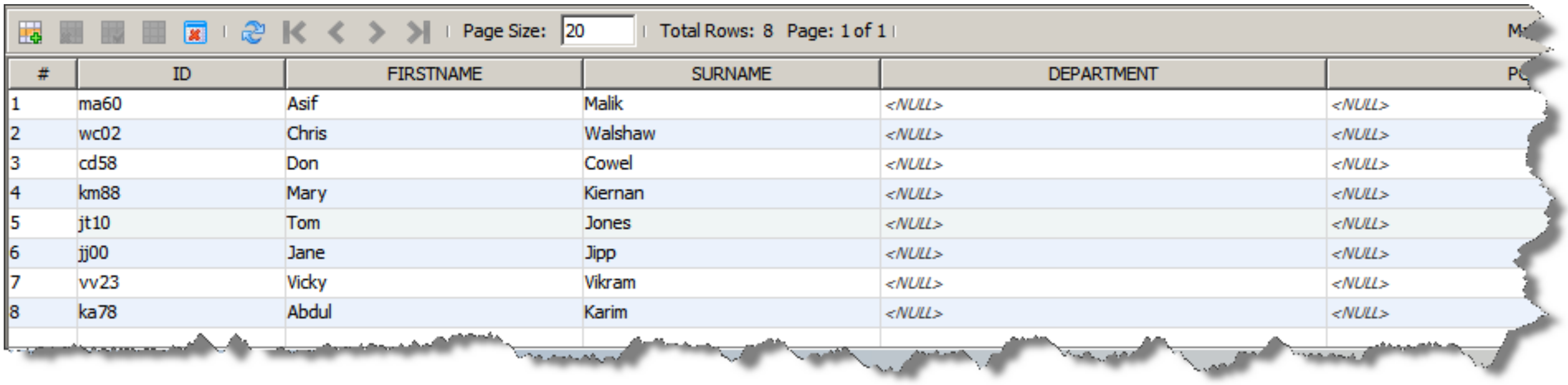
```
1  import java.io.File;
2  import java.sql.*;
3  import static javax.swing.JOptionPane.*;
4  import org.apache.derby.drda.NetworkServerControl;
5
6  public class DBDemo1 {
7
8      public static void main(String[] args) {
9
10         try {
11             NetworkServerControl server = new NetworkServerControl();
12             server.start(null);
13             // Load JDBC driver
14             Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
15             //Establish a connection
16             String sourceURL = "jdbc:derby://localhost:1527/"
17                     + new File("UserDB").getAbsolutePath() + ";";
18             Connection userDB = DriverManager.getConnection(sourceURL, "use", "use");
19             //Create a statement
20             Statement myStatement = userDB.createStatement();
21             if (showConfirmDialog(null, "add Fred Bloggs to database?") == YES_OPTION) {
22                 String writeString = "INSERT INTO Users(Firstname, Surname, Id) VALUES('Fred', 'Bloggs', 'bf01')";
23                 myStatement.executeUpdate(writeString);
24             }
25             // Execute a statement
26             ResultSet results = myStatement.executeQuery("SELECT Firstname, Surname, Id FROM Users ORDER BY Id");
```

Programmatically lines 11 & 12 start the database server. For this the import on line 4 is needed

Simple JDBC Example (cont.)

```java

            // Iterate through the result and print the names
            while (results.next()) {
                System.out.print(results.getString(1) + "  ");
                System.out.print(results.getString(2) + "  ");
                System.out.println(results.getString(3));
            }
            results.close();
            if (showConfirmDialog(null, "delete Fred Bloggs from database?") == YES_OPTION) {
                String deleteString = "DELETE FROM Users WHERE Surname='Bloggs' AND Firstname='Fred'";
                myStatement.executeUpdate(deleteString);
            } else {
                showMessageDialog(null, "OK - not deleted\n\ndo not add Fred Bloggs again\n"
                        + "or DBDemo1 will throw an SQL exception");
            }
            // Close the connection
            userDB.close();
        } // The following exceptions MUST be caught
        catch (SQLException sqle) {
            System.out.println(sqle);
        } catch (ClassNotFoundException cnfe) {
            System.out.println(cnfe);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

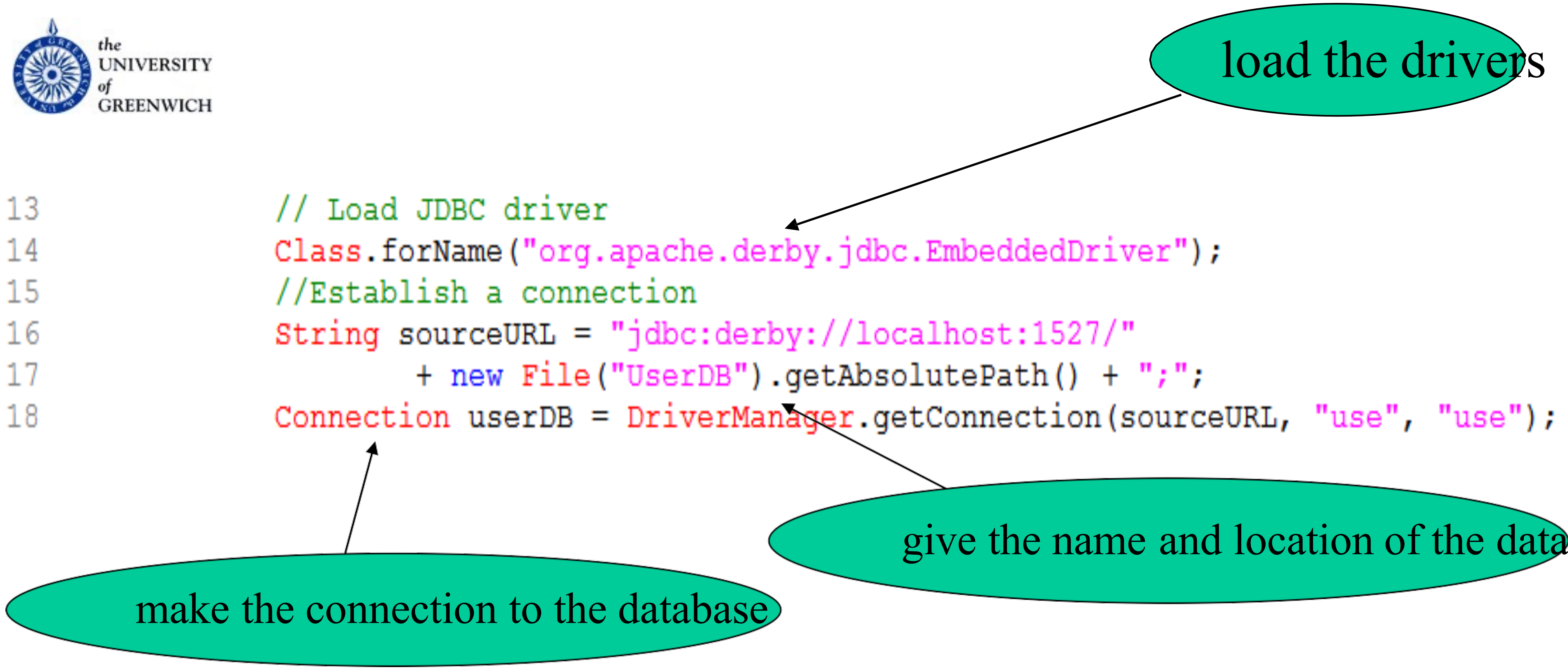
Our Apache Derby database – UserDB.mdb



Page Size: 20 | Total Rows: 8 Page: 1 of 1 | M

#	ID	FIRSTNAME	SURNAME	DEPARTMENT	P
1	ma60	Asif	Malik	<NULL>	<NULL>
2	wc02	Chris	Walshaw	<NULL>	<NULL>
3	cd58	Don	Cowel	<NULL>	<NULL>
4	km88	Mary	Kiernan	<NULL>	<NULL>
5	jt10	Tom	Jones	<NULL>	<NULL>
6	jj00	Jane	Jipp	<NULL>	<NULL>
7	vv23	Vicky	Vikram	<NULL>	<NULL>
8	ka78	Abdul	Karim	<NULL>	<NULL>

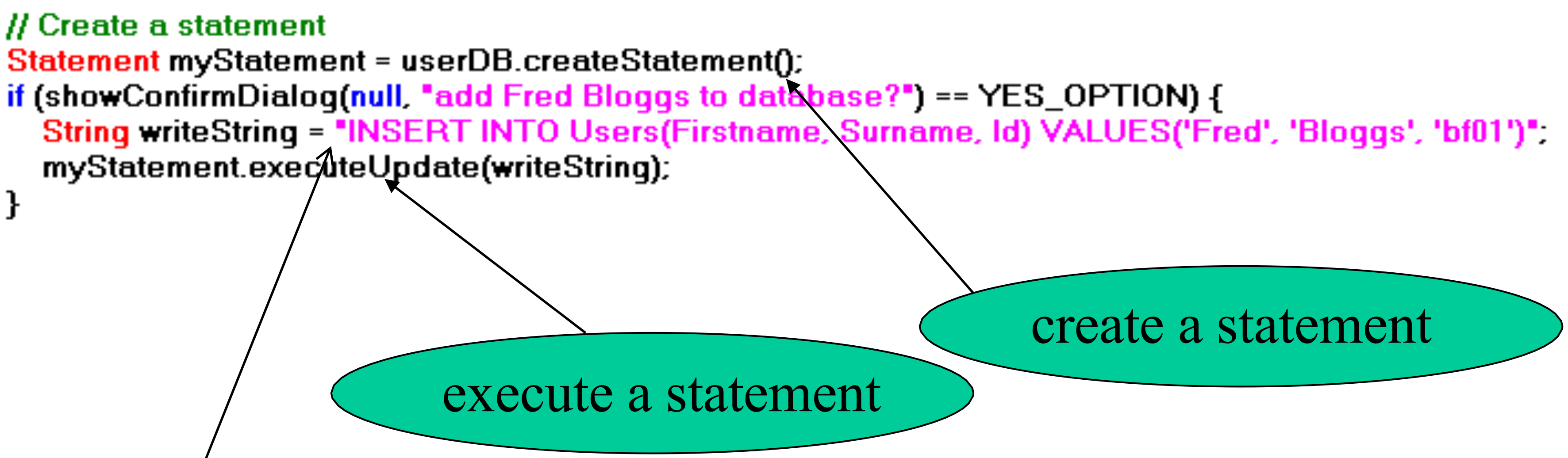
- Apache Derby is a relational database management system developed by the Apache Software Foundation.
- It can be embedded in Java programs and used for online transaction processing.
- It has a 3.5 MB disk-space footprint.


load the drivers
give the name and location of the data
make the connection to the database

```java
            // Load JDBC driver
            Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
            //Establish a connection
            String sourceURL = "jdbc:derby://localhost:1527/"
                    + new File("UserDB").getAbsolutePath() + ";";
            Connection userDB = DriverManager.getConnection(sourceURL, "use", "use");
```

sourceURL and userDB are our own variable names and we can change these as shown here without it affecting the connection to UserDB

In the code DBDemo1



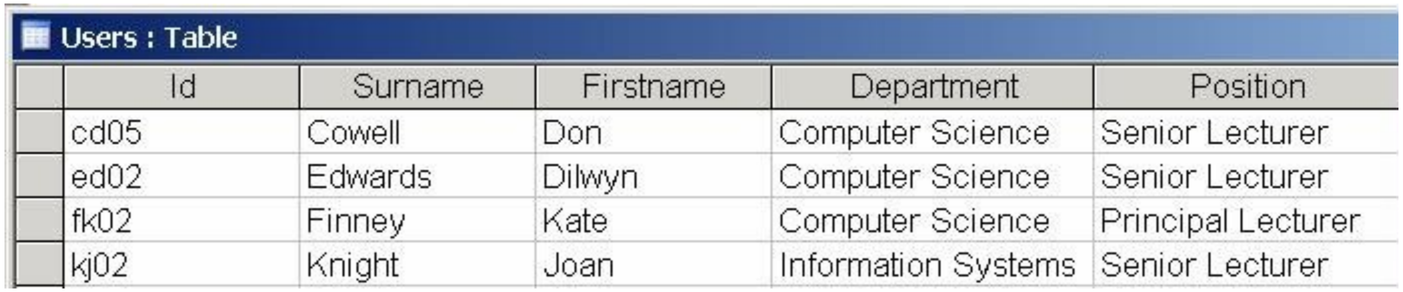
```
// Create a statement
Statement myStatement = userDB.createStatement();
if (showConfirmDialog(null, "add Fred Bloggs to database?") == YES_OPTION) {
   String writeString = "INSERT INTO Users(Firstname, Surname, Id) VALUES('Fred', 'Bloggs', 'bf01')";
   myStatement.executeUpdate(writeString);
}
```

this string represents the SQL for the actions we want to perform with the database
Within the Java we use the createStatement which is a method of the **Connection** class to make an object of the **Statement** class
So that we can use these classes and methods we must use
import java.sql.*;
at the beginning of our program

What did it do?

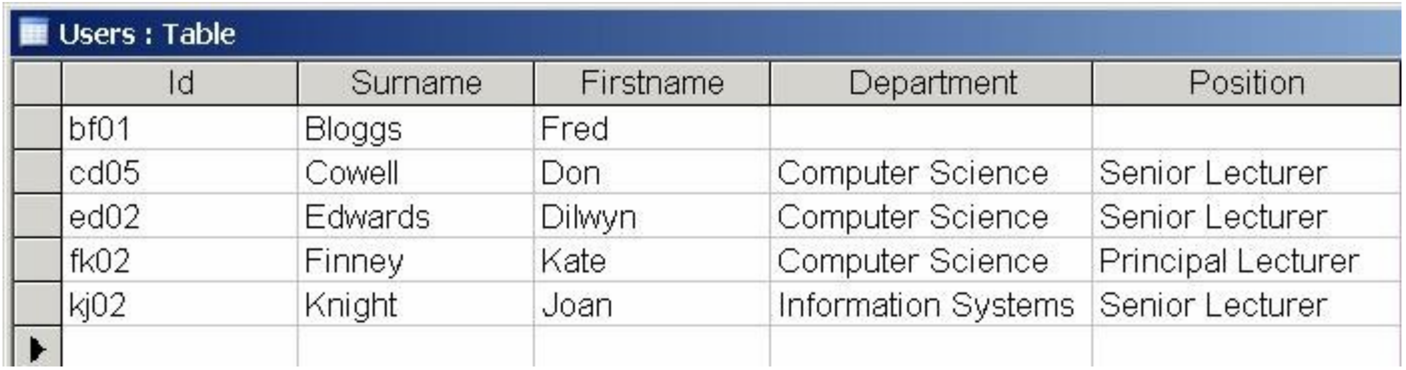
SQL

```
Statement myStatement = userDB.createStatement();
String writeString =
"INSERT INTO Users(Firstname, Surname, Id) VALUES('Fred', 'Bloggs', 'bf01')";
myStatement.executeUpdate(writeString);
```



Users : Table

Id	Surname	Firstname	Department	Position
cd05	Cowell	Don	Computer Science	Senior Lecturer
ed02	Edwards	Dilwyn	Computer Science	Senior Lecturer
fk02	Finney	Kate	Computer Science	Principal Lecturer
kj02	Knight	Joan	Information Systems	Senior Lecturer



Users : Table

Id	Surname	Firstname	Department	Position
bf01	Bloggs	Fred		
cd05	Cowell	Don	Computer Science	Senior Lecturer
ed02	Edwards	Dilwyn	Computer Science	Senior Lecturer
fk02	Finney	Kate	Computer Science	Principal Lecturer
kj02	Knight	Joan	Information Systems	Senior Lecturer

This did not have a 'reply' from the data base as we simply sent it some more data

Catching errors

```
try {
      // Load the JDBC driver

      //Establish a connection

      // Create a statement

}
catch (ClassNotFoundException cnfe)
{
      // if Class.forName string is wrong
   System.out.println(cnfe);
}
catch (SQLException sqle)
{
      // if connection can't be made
   System.out.println(sqle);
}
```

This connects to the Derby database in the same folder as the application. "use" and "use" are the database username and password.

You MUST catch both of the exceptions here in order to compile the code

Executing a database command

get results

SQL

```
ResultSet results = myStatement.executeQuery
   ("SELECT Firstname, Surname, Id FROM Users ORDER BY Id");
```

next and close are methods of the ResultSet class

executeQuery is for SELECT statements.

Note the name of the table is given here - Users

It produces a ResultSet which is like a "Dynaset" in Access – i.e. a transient table which is the results of the query

To check the results of the query we must print out the ordered set

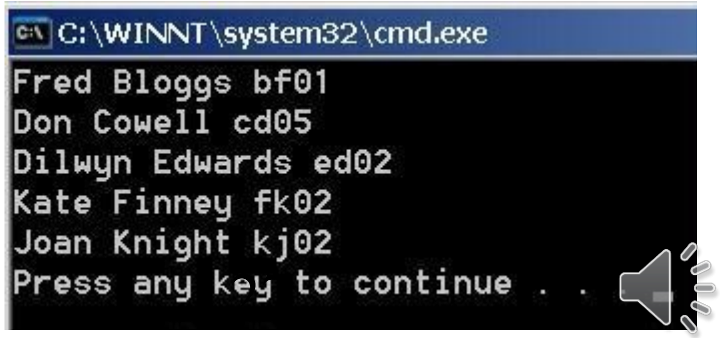
```
while (results.next()) {
   out.print(results.getString(1) + " ");
   out.print(results.getString(2) + " ");
   out.println(results.getString(3));
}
```

getString(1)

getString(3)

Fred Bloggs bf01

getString(2)



```
C:\WINNT\system32\cmd.exe
Fred Bloggs bf01
Don Cowell cd05
Dilwyn Edwards ed02
Kate Finney fk02
Joan Knight kj02
Press any key to continue . . .
```

SQL query

Format for SQL Query:

- *“select <field(s)> from <table(s)> where <condition> order by <field(s)”*

```
SELECT column1, column2 FROM table1, table2 WHERE column2='value';
```

Note: condition - field type must match values specified; e.g.

- Id=‘cd05’ for string type;
- Age>21 for integer type.

Single quote

Format for inserting a record

- **sqlString = “INSERT INTO Users(Firstname, Surname, Id) VALUES('Bob', 'Dolden', 'dr05')”**
- This sort of string manipulation is difficult to get right, especially when the SQL string needs to contain single quotes and the Java String double quotes. A good debugging tip is to print the string out to the console to check it – e.g. we could write:
  - System.out.println(sqlString);

Reading from a database

Use the **executeQuery** method.

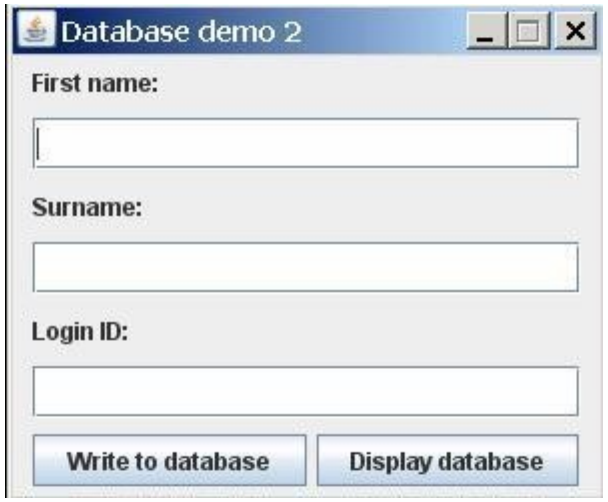
This returns a **ResultSet** which you can process using methods

- **next()** to move to the next row
- **getString(*n*)** to get column *n* from the row
- **getInt(*n*)** to get column *n* as an int
- **getDouble(*n*)** to get column n as a double
- **close()** to close it

…

Now with a GUI DBDemo2

The user of our system may not be a programmer and so we have to have an interface to help them enter data into the database



```java
import java.sql.*;
import javax.swing.*;
import static javax.swing.JOpti
import org.apache.derby.drda.Netwo

public class DBDemo2 extends JFrame implements ActionListener {

    JTextField firstName, surname, loginId;
    JButton writeBtn, displayBtn;
    Connection userDB;
    Statement myStatement;

    public static void main(String[] args) {
        new DBDemo2();
    }

    public DBDemo2() {
        setLayout(new BorderLayout());
        firstName = new JTextField();
        surname = new JTextField();
        loginId = new JTextField();
        writeBtn = new JButton("Write to database")
        displayBtn = new JButton("Display database"
        JPanel middle = new JPanel();
        middle.setLayout(new GridLayout(6, 1, 5, 5)
        middle.add(new JLabel("First name:"));
        middle.add(firstName);
        middle.add(new JLabel("Surname:"));
        middle.add(surname);
        middle.add(new JLabel("Login ID:"));
        middle.add(loginId);
        add("Center", middle);
        JPanel bottom = new JPanel();
        bottom.add(writeBtn);
        bottom.add(displayBtn);
        add("South", bottom);
        add("West", new JPanel());
        add("East", new JPanel());
        writeBtn.addActionListener(this);
        displayBtn.addActionListener(this);
        setSize(300, 250);
        setTitle("Database demo 2");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
```

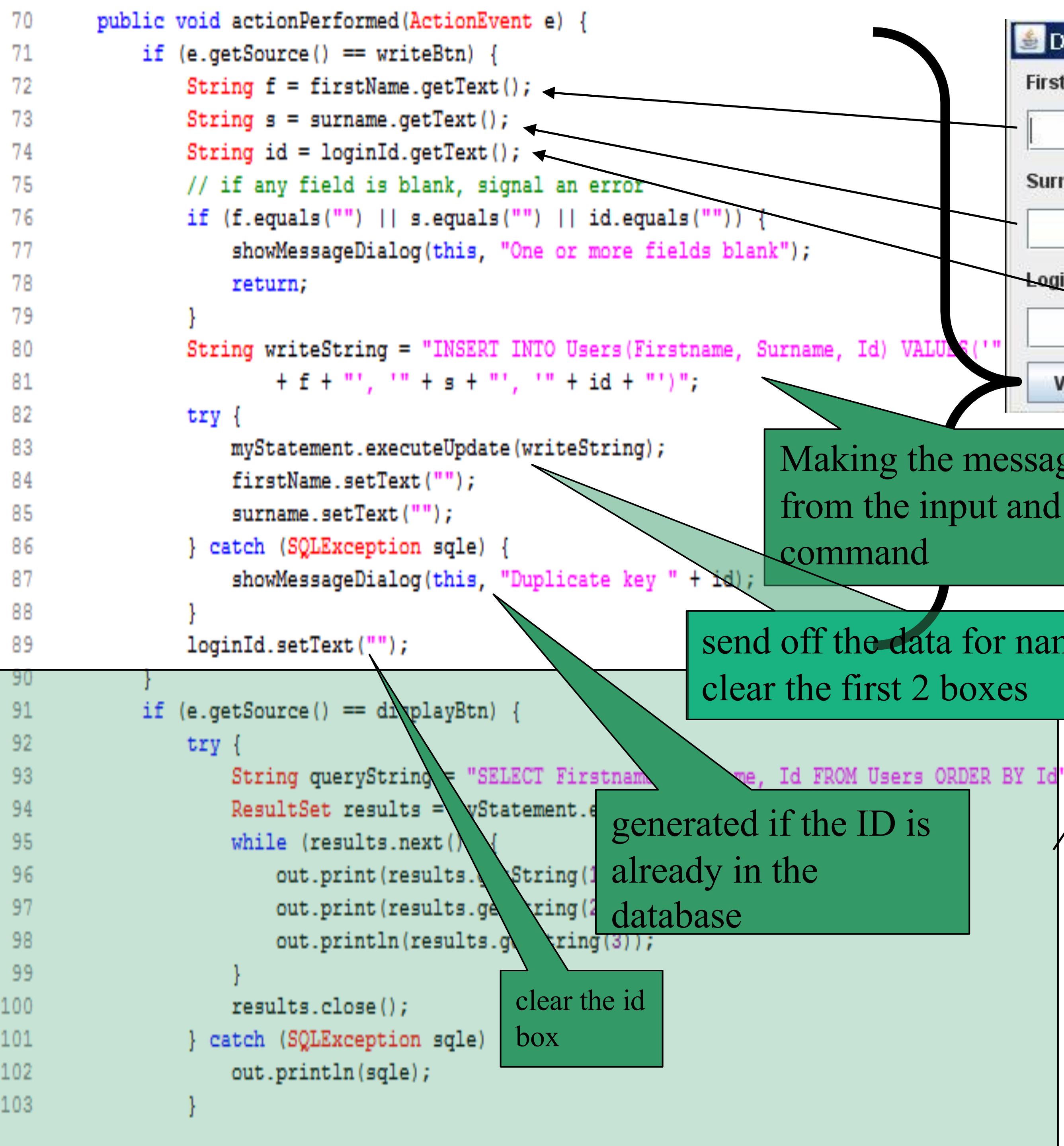
importing the necessary libraries

all this is just setting up the GUI with a nice layout

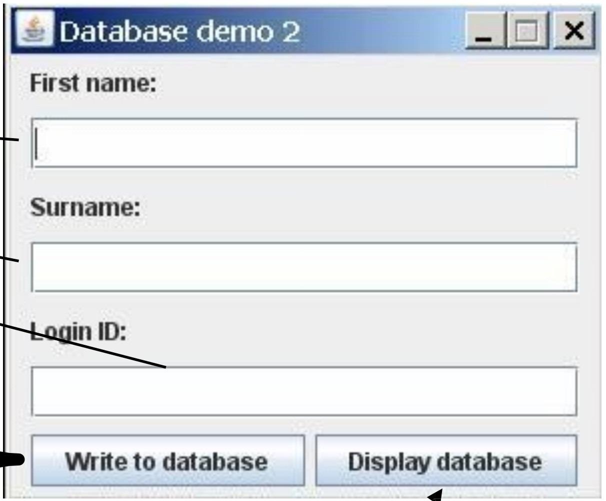
```java
try {
    NetworkServerControl server = new NetworkServerControl();
    server.start(null);
    // Load JDBC driver
    Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
    //Establish a connection
    String sourceURL = "jdbc:derby://localhost:1527/"
            + new File("UserDB").getAbsolutePath() + ";";
    Connection userDB = DriverManager.getConnection(sourceURL, "use", "use");
    myStatement = userDB.createStatement();
} // The following exceptions must be caught
catch (ClassNotFoundException cnfe) {
    out.println(cnfe);
} catch (SQLException sqle) {
    out.println(sqle);
} catch (Exception e) {
    System.out.println(e);
}
```

establishing the connection and associating the statement with the connection

error handling



```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == writeBtn) {
        String f = firstName.getText();
        String s = surname.getText();
        String id = loginId.getText();
        // if any field is blank, signal an error
        if (f.equals("") || s.equals("") || id.equals("")) {
            showMessageDialog(this, "One or more fields blank");
            return;
        }
        String writeString = "INSERT INTO Users(Firstname, Surname, Id) VALUES('"
                + f + "', '" + s + "', '" + id + "')";
        try {
            myStatement.executeUpdate(writeString);
            firstName.setText("");
            surname.setText("");
        } catch (SQLException sqle) {
            showMessageDialog(this, "Duplicate key " + id);
        }
        loginId.setText("");
    }
    if (e.getSource() == displayBtn) {
        try {
            String queryString = "SELECT Firstnam          me, Id FROM Users ORDER BY Id";
            ResultSet results = myStatement.e
            while (results.next()) {
                out.print(results.getString(1
                out.print(results.ge    ring(2
                out.println(results.g    ring(3));
            }
            results.close();
        } catch (SQLException sqle) {
            out.println(sqle);
        }
```

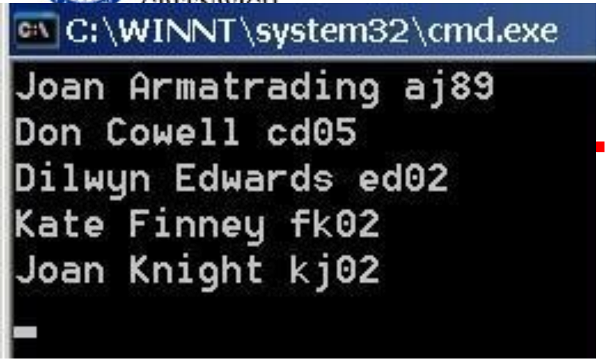


Making the message up from the input and the SQL command

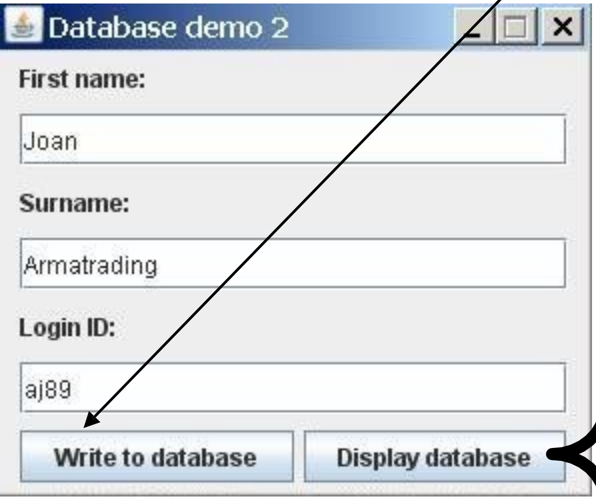
send off the data for names and clear the first 2 boxes

generated if the ID is already in the database

clear the id box



```
C:\WINNT\system32\cmd.exe
Joan Armatrading aj89
Don Cowell cd05
Dilwyn Edwards ed02
Kate Finney fk02
Joan Knight kj02
_
```



Database demo 2

First name: Joan

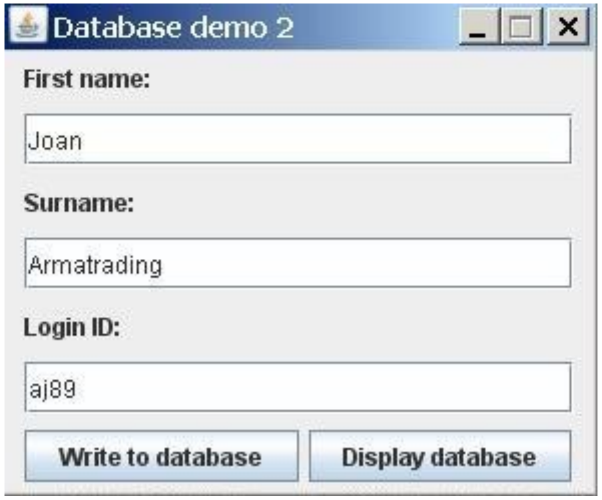
Surname: Armatrading

Login ID: aj89

```java
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == writeBtn) {
            String f = firstName.getText();
            String s = surname.getText();
            String id = loginId.getText();
            // if any field is blank, signal an error
            if (f.equals("") || s.equals("") || id.equals("")) {
                showMessageDialog(this, "One or more fields blank");
                return;
            }
            String writeString = "INSERT INTO Users(Firstname, Surname, Id) VALUES('"
                    + f + "', '" + s + "', '" + id + "')";
            try {
                myStatement.executeUpdate(writeString);
                firstName.setText("");
                surname.setText("");
            } catch (SQLException sqle) {
                showMessageDialog(this, "Duplicate key " + id);
            }
            loginId.setText("");
        }
        if (e.getSource() == displayBtn) {
            try {
                String queryString = "SELECT Firstname, Surname, Id FROM Users ORDER BY Id";
                ResultSet results = myStatement.executeQuery(queryString);
                while (results.next()) {
                    out.print(results.getString(1) + " ");
                    out.print(results.getString(2) + " ");
                    out.println(results.getString(3));
                }
                results.close();
            } catch (SQLException sqle) {
                out.println(sqle);
            }
```

Now with an auxiliary class

- We want to separate our GUI from 'business functions' (as usual).
- The interface looks identical so that the user would not notice any differences

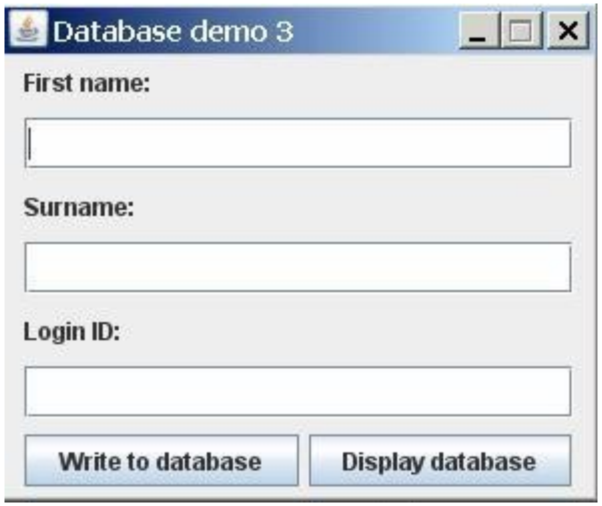


Database demo 2

First name: Joan

Surname: Armatrading

Login ID: aj89

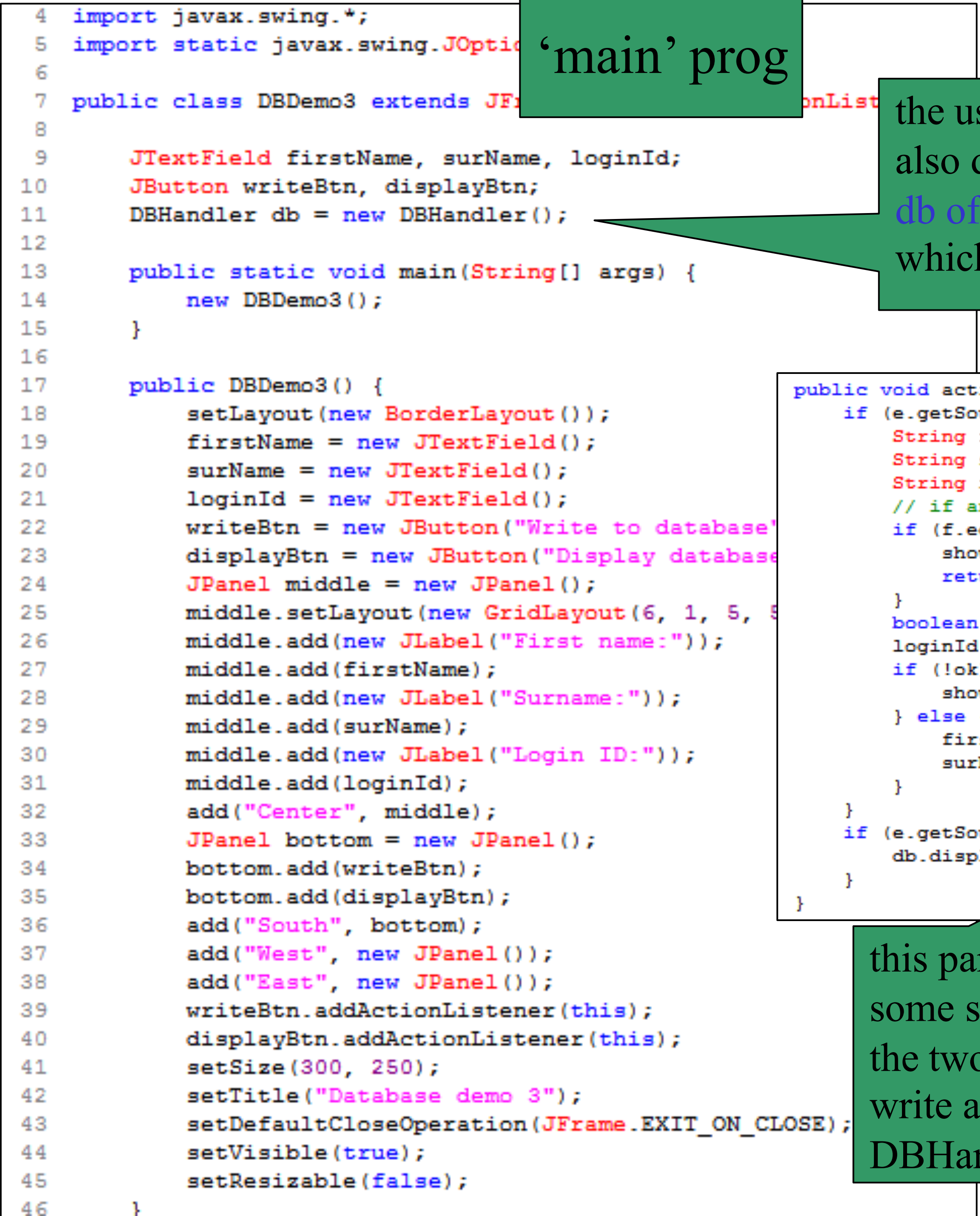


Database demo 3

First name:

Surname:

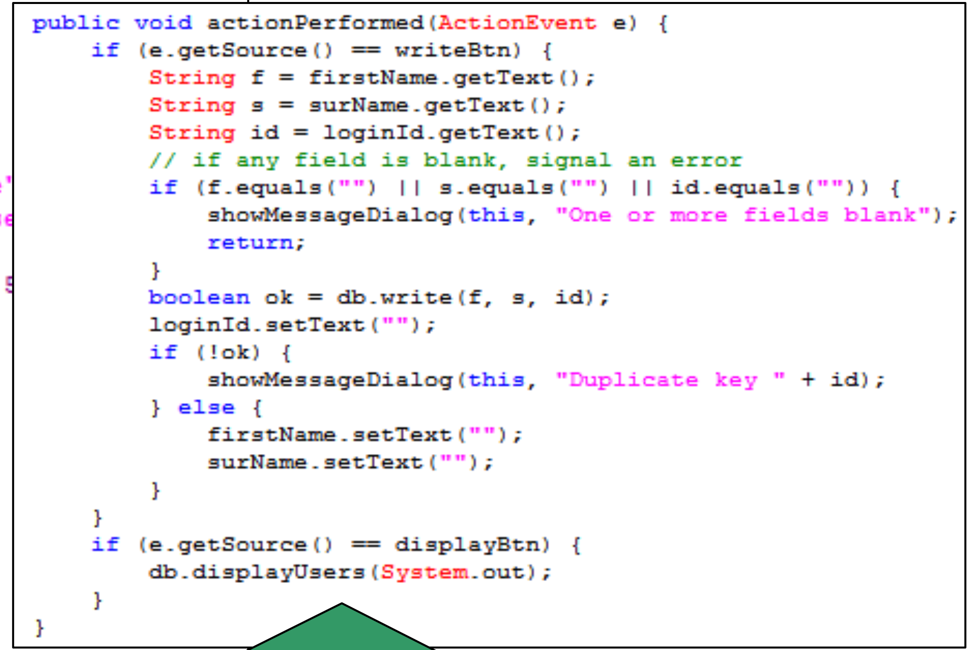
Login ID:



```
import javax.swing.*;
import static javax.swing.JOpti

public class DBDemo3 extends JF                    onList

    JTextField firstName, surName, loginId;
    JButton writeBtn, displayBtn;
    DBHandler db = new DBHandler();

    public static void main(String[] args) {
        new DBDemo3();
    }

    public DBDemo3() {
        setLayout(new BorderLayout());
        firstName = new JTextField();
        surName = new JTextField();
        loginId = new JTextField();
        writeBtn = new JButton("Write to database"
        displayBtn = new JButton("Display database
        JPanel middle = new JPanel();
        middle.setLayout(new GridLayout(6, 1, 5, 5
        middle.add(new JLabel("First name:"));
        middle.add(firstName);
        middle.add(new JLabel("Surname:"));
        middle.add(surName);
        middle.add(new JLabel("Login ID:"));
        middle.add(loginId);
        add("Center", middle);
        JPanel bottom = new JPanel();
        bottom.add(writeBtn);
        bottom.add(displayBtn);
        add("South", bottom);
        add("West", new JPanel());
        add("East", new JPanel());
        writeBtn.addActionListener(this);
        displayBtn.addActionListener(this);
        setSize(300, 250);
        setTitle("Database demo 3");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
        setResizable(false);
    }
```

‘main’ prog

the usual GUI set up of buttons etc
also declaring the object
db of the DBHandler class
which will do the ‘business’



```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == writeBtn) {
        String f = firstName.getText();
        String s = surName.getText();
        String id = loginId.getText();
        // if any field is blank, signal an error
        if (f.equals("") || s.equals("") || id.equals("")) {
            showMessageDialog(this, "One or more fields blank");
            return;
        }
        boolean ok = db.write(f, s, id);
        loginId.setText("");
        if (!ok) {
            showMessageDialog(this, "Duplicate key " + id);
        } else {
            firstName.setText("");
            surName.setText("");
        }
    }
    if (e.getSource() == displayBtn) {
        db.displayUsers(System.out);
    }
}
```

this part handles the input and does
some simple validation and then calls
the two methods
write and displayUsers of the
DBHandler object

```java
import java.io.*;
import static java.lang.System.*;
import java.sql.*;

class DBHandler {

    private Statement myStatement;

    public DBHandler() {
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String sourceURL = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb, *.accdb)};DBQ="
                    + new File("UserDB.accdb").getAbsolutePath() + ";";
            Connection userDB = DriverManager.getConnection(sourceURL, "admin", "");
            myStatement = userDB.createStatement();
        } // The following exceptions must be caught
        catch (ClassNotFoundException cnfe) {
            out.println(cnfe);
        } catch (SQLException sqle) {
            out.println(sqle);
        }
    }

    public boolean write(String f, String s, String id) {
        String writeString =
                "INSERT INTO Users(Firstname, Surname, Id) VALUES('"
                + f + "', '" + s + "', '" + id + "')";
        try {
            myStatement.executeUpdate(writeString);
        } catch (SQLException sqle) {
            return false; // duplicate key
        }
        return true; // inserted OK
    }

    public void displayUsers(PrintStream outS) {
        try {
            String queryString = "SELECT Firstname, Surname, Id FROM Users ORDER BY Id";
            ResultSet results = myStatement.executeQuery(queryString);
            while (results.next()) {
                outS.print(results.getString(1) + " ");
                outS.print(results.getString(2) + " ");
                outS.println(results.getString(3));
            }
            results.close();
        } catch (SQLException sqle) {
            out.println(sqle);
        }
    }
}
```

default constructor need no arguments passing-these are sent when the write method is called

setting up the database Connection and Statement
with the attendant 'catches' incase of error

the two methods write and displayUsers
note that the first is sent the 3 variables and the second is given a PrintStream to use for its results

Summary

- An overall view of Java Database programming
- How to connect to a Database
- How to create, execute, and process SQL queries.
- How to code with auxiliary class