

Java SE JDBC Introduction

abelski

Introduction

What is JDBC?

- The JDBC API includes all it takes in order to access a relational database.
- The JDBC API includes two packages:

`java.sql`

Includes the required APIs to access and manipulate data stored in a database.

`javax.sql`

Includes the required APIs to access server side data sources.

What is JDBC?

- The JDBC API mainly includes the interfaces that needs to be implemented by drivers used to connect with data base servers. Database vendors provide their implementation for these interfaces.

JDBC Drivers

- The JDBC drivers implementations are categorized into four types:

Type 1 Drivers

Implement the JDBC API as a mapping to another data access API (such as ODBC). These drivers are usually limited for a specific platform for which the data access API was implemented. In addition, these drivers are usually slower, usually lack the support for multi threading and usually lack the support for most JDBC API newest features.

Type 2 Drivers

Written partly in Java and partly in native code. For that reason, they are developed for specific platforms and have a limited portability.

JDBC Drivers

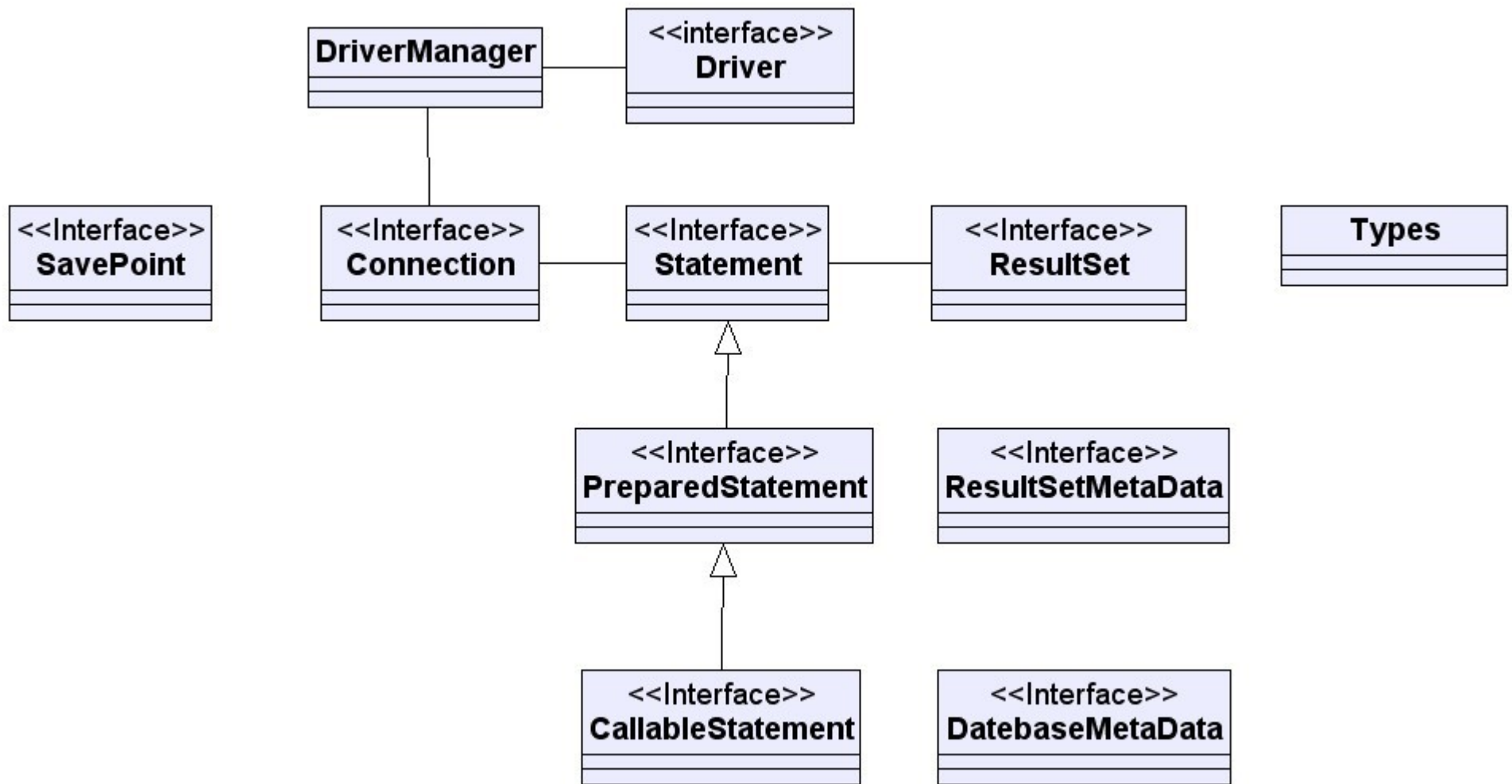
Type 3 Drivers

Include a 100% pure Java client and communicate with a middle-ware server that connects with the database using the database dependent protocol.

Type 4 Drivers

Written 100% in Java. Platform independent. Communicate directly with the data source using standard Java sockets.

The java.sql UML Diagram



Connecting to a Database

- Connecting with a database is possible via one of the following ways:
 - Using The DriverManager Class
 - Using DataSource Implementation
- Using the DataSource implementation we will enjoy the following benefits:
 - An Improved Portability
 - Simpler Maintenance for Our Code

Connecting to a Database

- In order to connect with a database using the DriverManager class we need to follow these steps:

Registering the appropriate JDBC driver

Registering the required JDBC driver is done by calling the `DriverManager.registerDriver()` method and sending a reference to the object that can be used as a driver to it. Alternatively, instantiating the driver class will cause its registration as well (according to the specification the driver default constructor should include a call to `DriverManager.registerDriver()` method to register that driver).

Calling the `getConnection()` method

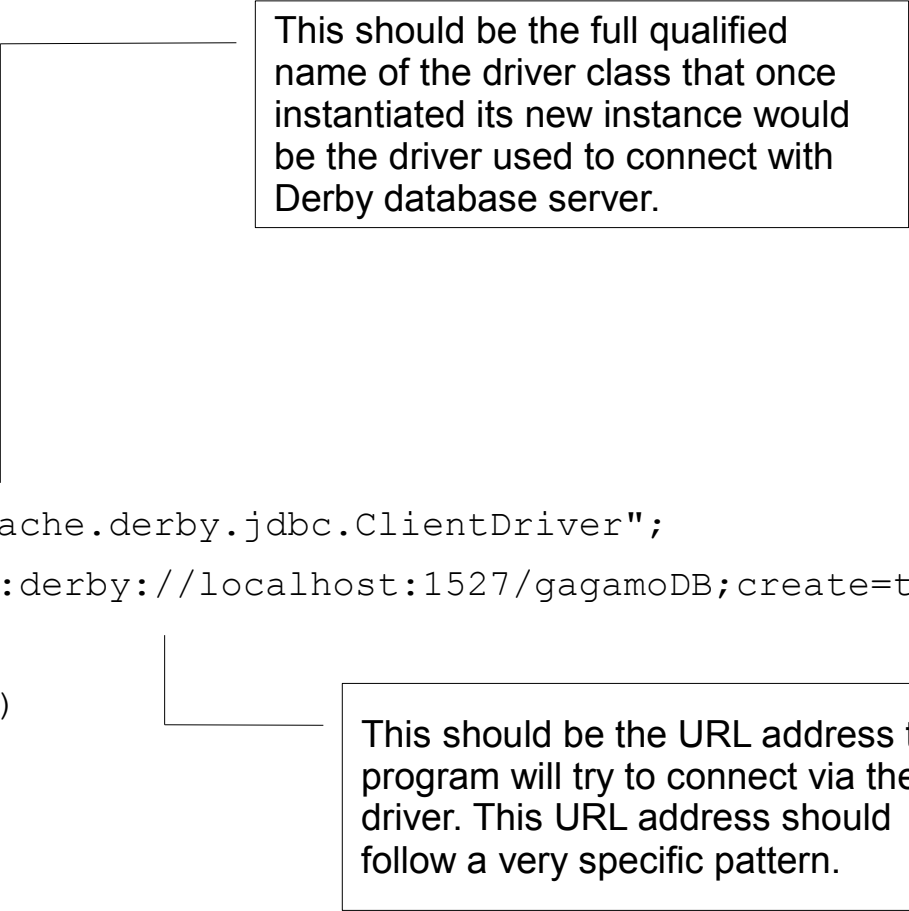
Once the required driver is already registered, calling the `getConnection()` method will indirectly call the `connect` method on that driver (an object instantiated from the class that implements `Driver` specifically for the database we work with).

Code Sample

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DerbySimpleCode
{
    public static String driver = "org.apache.derby.jdbc.ClientDriver";
    public static String protocol = "jdbc:derby://localhost:1527/gagamoDB;create=true";

    public static void main(String[] args)
    {
        Connection connection = null;
        Statement statement = null;
        ResultSet rs = null;
    }
}
```



The diagram consists of two L-shaped lines. The first line starts at the right side of the code line `import java.sql.DriverManager;` and extends horizontally to the right, then vertically down to a text box. The second line starts at the right side of the code line `protocol = "jdbc:derby://localhost:1527/gagamoDB;create=true";` and extends horizontally to the right, then vertically down to another text box.

This should be the full qualified name of the driver class that once instantiated its new instance would be the driver used to connect with Derby database server.

This should be the URL address this program will try to connect via the driver. This URL address should follow a very specific pattern.

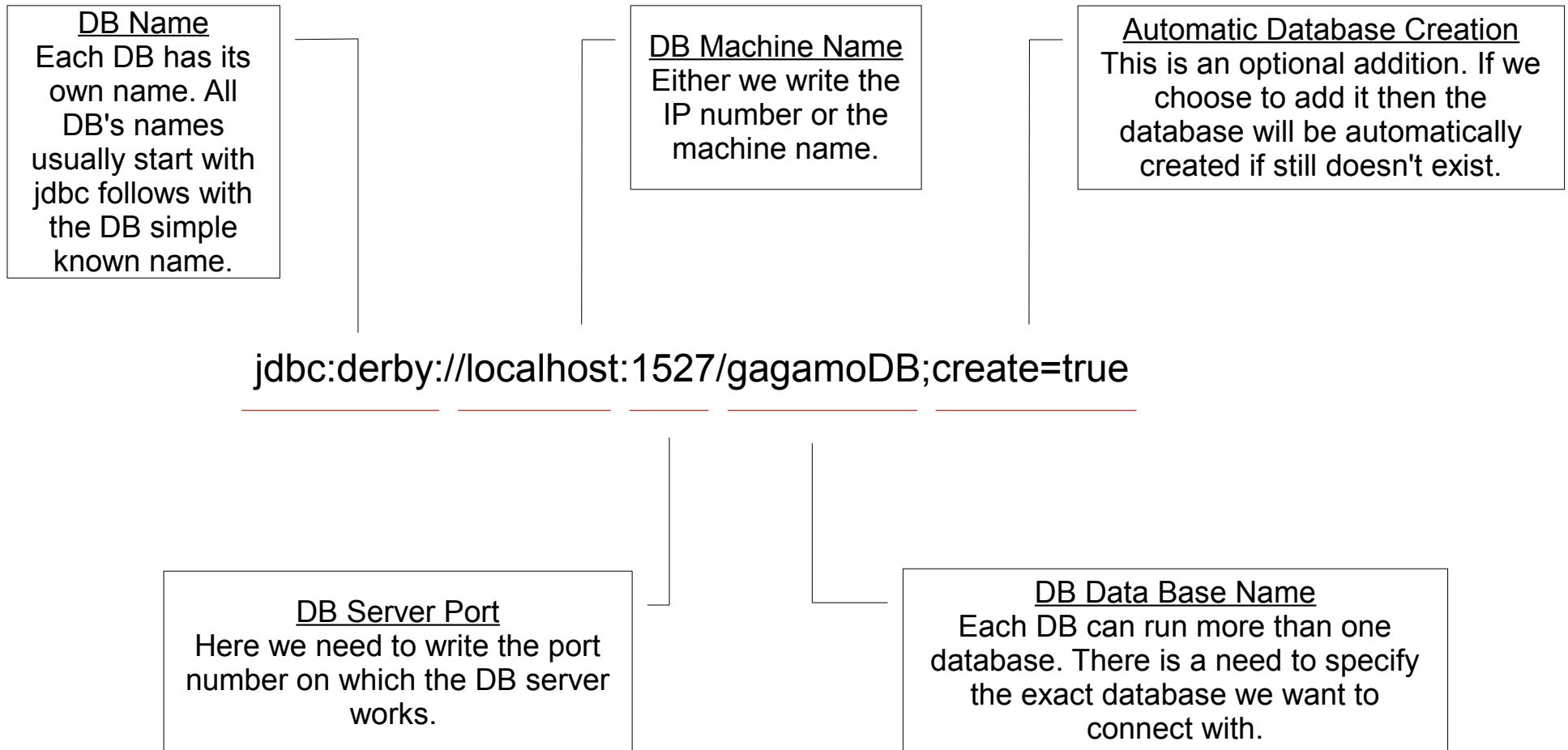
Code Sample

```
try
{
    connection = null;
    //Instantiating the driver class will indirectly register
    //this driver as an available driver for DriverManager
    Class.forName(driver);
    //Getting a connection by calling getConnection
    connection = DriverManager.getConnection(protocol);
    statement = connection.createStatement();
    statement.execute("create table inventory(id int, fee double)");
    statement.execute("insert into inventory values (100212,2.5)");
    statement.execute("insert into inventory values (100213,1.2)");
    statement.execute("insert into inventory values (100214,4.2)");
}
```

Code Sample

```
rs = statement.executeQuery(
    "SELECT id,fee FROM inventory ORDER BY id");
while(rs.next())
{
    System.out.println("id="+rs.getInt("id")
        +" fee="+rs.getDouble("fee"));
}
statement.execute("DROP TABLE inventory");
}
catch(Exception e) { e.printStackTrace(); }
finally
{
    if(statement!=null) try{statement.close();}catch(Exception e){}
    if(connection!=null) try{connection.close();}catch(Exception e){}
    if(rs!=null) try{rs.close();}catch(Exception e){}
}
}
```

Database URL Pattern



Java SE JDBC Introduction

abelski

12/05/10

© 2008 Haim Michael. All Rights Reserved.

1

Introduction

What is JDBC?

- The JDBC API includes all it takes in order to access a relational database.
- The JDBC API includes two packages:

[java.sql](#)

Includes the required APIs to access and manipulate data stored in a database.

[javax.sql](#)

Includes the required APIs to access server side data sources.

What is JDBC?

- The JDBC API mainly includes the interfaces that needs to be implemented by drivers used to connect with data base servers. Database vendors provide their implementation for these interfaces.

JDBC Drivers

- The JDBC drivers implementations are categorized into four types:

Type 1 Drivers

Implement the JDBC API as a mapping to another data access API (such as ODBC). These drivers are usually limited for a specific platform for which the data access API was implemented. In addition, these drivers are usually slower, usually lack the support for multi threading and usually lack the support for most JDBC API newest features.

Type 2 Drivers

Written partly in Java and partly in native code. For that reason, they are developed for specific platforms and have a limited portability.

JDBC Drivers

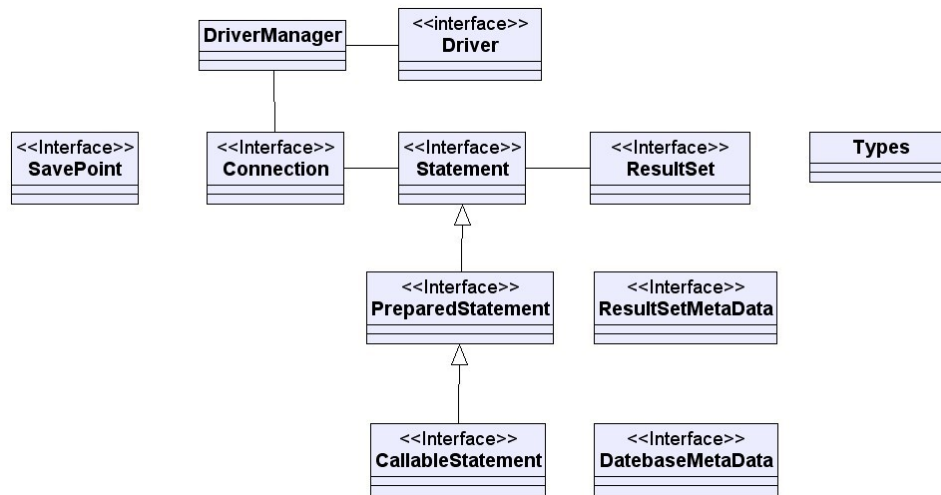
Type 3 Drivers

Include a 100% pure Java client and communicate with a middle-ware server that connects with the database using the database dependent protocol.

Type 4 Drivers

Written 100% in Java. Platform independent. Communicate directly with the data source using standard Java sockets.

The java.sql UML Diagram



12/05/10

© 2008 Haim Michael. All Rights Reserved.

7

Connecting to a Database

- Connecting with a database is possible via one of the following ways:
 - Using The DriverManager Class
 - Using DataSource Implementation
- Using the DataSource implementation we will enjoy the following benefits:
 - An Improved Portability
 - Simpler Maintenance for Our Code

Connecting to a Database

- In order to connect with a database using the DriverManager class we need to follow these steps:

Registering the appropriate JDBC driver

Registering the required JDBC driver is done by calling the DriverManager.registerDriver() method and sending a reference to the object that can be used as a driver to it. Alternatively, instantiating the driver class will cause its registration as well (according to the specification the driver default constructor should include a call to DriverManager.registerDriver() method to register that driver).

Calling the getConnection() method

Once the required driver is already registered, calling the getConnection() method will indirectly call the connect method on that driver (an object instantiated from the class that implements Driver specifically for the database we work with).

12/05/10

© 2008 Haim Michael. All Rights Reserved.

9

The alternative of using a DataSource is presented in “Java 2 SE JNDI Introduction” course.

Code Sample

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DerbySimpleCode
{
    public static String driver = "org.apache.derby.jdbc.ClientDriver";
    public static String protocol = "jdbc:derby://localhost:1527/gagamoDB;create=true";

    public static void main(String[] args)
    {
        Connection connection = null;
        Statement statement = null;
        ResultSet rs = null;
    }
}
```

This should be the full qualified name of the driver class that once instantiated its new instance would be the driver used to connect with Derby database server.

This should be the URL address this program will try to connect via the driver. This URL address should follow a very specific pattern.

12/05/10

© 2008 Haim Michael. All Rights Reserved.

10

Starting with JDK 1.6, the Apache Derby database is automatically installed when installing the JDK.

The Apache Derby database default installation folder is [c:\Program Files\Sun\JavaDB](#)

The Derby database can be used in a client-server mode, which is the common way for writing Java code that communicates with a database. The database server is running on one hand and the Java code (via the JDBC driver) acts as the client. Another possibility Derby database allows us is running it as part of our code (embedded mode). This option is unique for Derby database and it won't be discussed in this course.

Running this sample requires opening two command lines. In one of them there is a need to run the Derby database server.

To do so there is a need to type the following command:
`java org.apache.derby.drda.NetworkServerControl start`
In order to enable this command to work you need to update the classpath environment with `derby.jar` and `derbynet.jar` located in [c:\Program Files\Sun\JavaDB\lib](#).

Code Sample

```
try
{
    connection = null;
    //Instantiating the driver class will indirectly register
    //this driver as an available driver for DriverManager
    Class.forName(driver);
    //Getting a connection by calling getConnection
    connection = DriverManager.getConnection(protocol);
    statement = connection.createStatement();
    statement.execute("create table inventory(id int, fee double)");
    statement.execute("insert into inventory values (100212,2.5)");
    statement.execute("insert into inventory values (100213,1.2)");
    statement.execute("insert into inventory values (100214,4.2)");
}
```

12/05/10

© 2008 Haim Michael. All Rights Reserved.

11

One option through which the class path can be updated with these two jar files is by typing the following command:

```
D:\>set classpath=c:\Program
Files\Sun\JavaDB\lib\derby.jar;c:\Program
Files\Sun\JavaDB\lib\derbynet.jar;%classpath%
```

Once the Derby database server is running in one command line window it is possible to run our sample in another one. The `org.apache.derby.jdbc.ClientDriver` class can be found within the `derbyclient.jar` file. In order to be capable to use the `org.apache.derby.jdbc.ClientDriver` we must add `derbyclient.jar` file to our Java environment. We can do it by typing the the following command: `D:\>set classpath=c:\Program Files\Sun\JavaDB\lib\derbyclient.jar;%classpath%`

Code Sample

```
rs = statement.executeQuery(
    "SELECT id,fee FROM inventory ORDER BY id");
while(rs.next())
{
    System.out.println("id="+rs.getInt("id")
        +" fee="+rs.getDouble("fee"));
}
statement.execute("DROP TABLE inventory");
}
catch(Exception e) { e.printStackTrace(); }
finally
{
    if(statement!=null) try{statement.close();}catch(Exception e){}
    if(connection!=null) try{connection.close();}catch(Exception e){}
    if(rs!=null) try{rs.close();}catch(Exception e){}
}
}
```

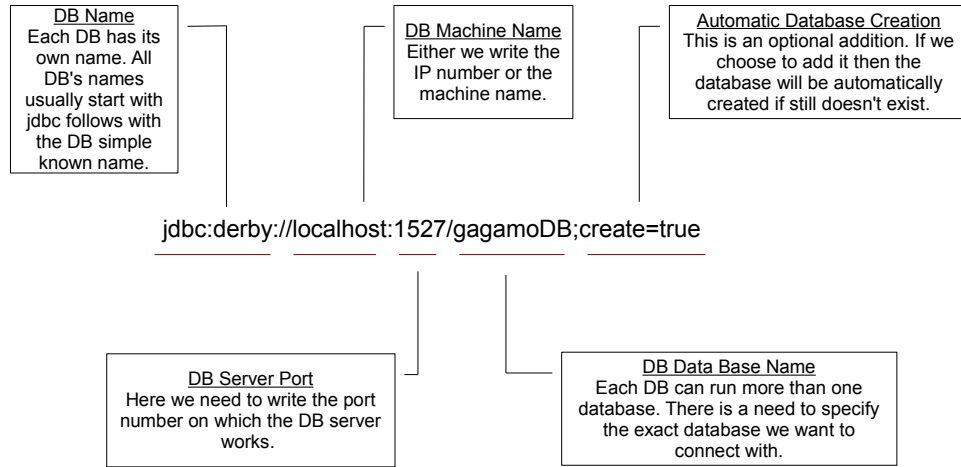
12/05/10

© 2008 Haim Michael. All Rights Reserved.

12

This code includes one catch block only in order to have it on two slides (instead of three). Practically, when writing such code it isn't recommended to use one catch block only that tries to catch the simple Exception type. Instead, it is recommended to place catch blocks that refer specifically to each one of the exception types that might be thrown and which we should handle.

Database URL Pattern



12/05/10

© 2008 Haim Michael. All Rights Reserved.

13