1 Java

Java is just another programming language, that's similar to C/C++ syntax (and C#[1]). It is an interpreted language that has been applied to a variety of domains—such as client side programming via Applets and Swing applications, and server side via Java Enterprise Edition.

Java still remains relatively popular for database/server type applications. Many would argue that the glory days of Java on the client side (Applets/Applications) is long gone...

1.1 Installation

If you're using Windows, you probably already have a way to execute Java programs. It is called the JavaVM, and is usually included with Internet Explorer. If you want to compile programs though, you need to download the Java SDK. You can find it at `http://java.sun.com`.

After you install the SDK, you should setup your `$JAVA_HOME` and `$PATH` variables. The `$JAVA_HOME` needs to point to the directory where you installed the SDK, typically something like `c:\j2sdk`, and `$PATH` should point to the 'bin' directory, typically `c:\j2sdk\bin`

2 JDBC

Java uses JDBC[2] to access databases of various sorts. It is a portable database interface: all databases do the same things anyway in the same way—so why not create a library that abstracts minor differences between databases?

In this document, we will develop a web-site using JSP, using Microsoft Access at first, and then quickly (without hassle) switch to using Oracle.

3 JSP

JSP stands for Java Server Pages. It is a form of a Servlet. A Servlet is a piece of Java code that is setup to process requests (ie: it acts like a server—but not a full fledged server[3].)

JSP is sort of like ASP, or PHP, only it uses Java.

3.1 Installation: Tomcat

You need a "Servlet Container" in order to run (& develop) Servlets and JSP pages. There are many such "containers". The first (developed by Sun) and most standard one (and generally relatively fast) is Tomcat. Now managed by the Apache Group; i.e.: you can download it for free from `http://www.apache.org/`. It is a sub-project of "Jakarta" (so

[1]C# came *after* Java.

[2]Which *does not* stand for Java Database Connectivity—in a similar way as ODBC *does* stand for Open Database Connectivity

[3]Well, you can implement full fledged servers of all kinds using the Servlet API

click on Jakarta link). After downloading the binary distribution (get Tomcat version 5), you unzip it into something like: `C:\jakarta-tomcat`.

To start Tomcat, you run: `C:\jakarta-tomcat\startup`, and to stop it, you run: `C:\jakarta-tomcat\shutdown`. There are other ways of starting it up (you can even have it run as a Windows service); you can read up on various configuration options in the documentation, which (now that you've installed it and it's running), can be found on `http://localhost:8080/tomcat-docs/` or alternatively on the Jakarta website.

4 Microsoft Access

Yes!, you read it right. Microsoft Access. Why are we studying it? Because it's always there... and sometimes it's sorta useful as a quick proof-of-concept thing. Anyway, don't worry if you don't have Microsoft Access... since you already have it: it's built into Windows.

In Windows: go to your control panel (maybe also "Advanced Tools") and then double click on "Data Sources (ODBC)". This is a central Windows thing that allows you to configure ODBC interfaces. Once there, pick the "System DSN" tab, and click "Add..." From the list of drivers that come up, select "Microsoft Access Driver (*.mdb)" and click "Finish" button.

Click on the "Create" button to crate a new database; and place it some directory like `C:\Temp`. Name it `gbookdsn`, and give it any description you like. Click ok, etc., and congratulations, you've just setup a Microsoft Access database!

4.1 Adding Tables, etc.

If you have Microsoft Access, just double click on the database file you've created, and add tables via the usual Microsoft Access GUI interface. On the other hand, if you're a bit under-funded, and haven't purchased a copy of Microsoft Access, then here's what you can do (besides "borrowing" a copy): download `dbconnect.zip`[4], then open the file, read the comments, setup appropriate DSN and just connect. Basically this is a client program that gives you a cheap version of the MySQL console, except it's for Microsoft Access (well, technically any ODBC capable database).

Whichever way you do it, you can create the table such as (you can just type it into that `dbconnect.zip` tool linked above):

```
--- this is for Microsoft Access
CREATE TABLE GBOOK (
   GBOOKID COUNTER,
   NAME VARCHAR(64),
   EMAIL VARCHAR(64),
   POSTDATE DATETIME,
   MESSAGE TEXT,
   PRIMARY KEY(GBOOKID)
);
```

[4]`http://www.theparticle.com/files/programs/util/dbconnect.zip`

5 Architecture

Just as before, we will break up the whole application into components, so that changing things is relatively easy. For example, for our GuestBook, we will have a "value" class such as:

```
package gbook;

import java.util.*;

public class GBook {
    public int gbookid;
    public String name;
    public String email;
    public String message;
    public Date postdate;
}
```

Obviously we could've made it nice, etc., but I'm a bit in a rush to get these notes typed up before the class.

5.1 Database Utility Class

Also, just as in our previous notes, we will put the database code into a separate class, in this case, named `DbHelper`. The source code is roughly[5]:

```
package gbook;

import java.sql.*;
import java.util.*;

public class DbHelper {

    // gets a db connection
    public static Connection getConnection() throws SQLException {
        Connection conn = null;
        try{
            // Microsoft Access PART
            // load db driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
            // connect with a Data Source Name (DSN) of a database.
            conn = DriverManager.getConnection("jdbc:odbc:gbookdsn");
        }catch(ClassNotFoundException e){
            e.printStackTrace();
        }catch(InstantiationException e){
```

[5]The actual source code contains comments on how to get Oracle to work.

```
        e.printStackTrace();
    }catch(IllegalAccessException e){
        e.printStackTrace();
    }
    return conn;
}

//add a guestbook record.
public static void addGuestBook(GBook o){
    try{
        Connection connection = DbHelper.getConnection();
        Statement statement = connection.createStatement();

        // Microsoft Access INSERT
        String command = "INSERT INTO GBOOK" +
            "(NAME,EMAIL,POSTDATE,MESSAGE) VALUES ('"+o.name+"',"+
                "'"+o.email+"',NOW(),'"+o.message+"')";

        statement.execute(command);
        statement.close();
        connection.close();
    }catch(SQLException e){
        e.printStackTrace();
    }
}

// get all guest book entries
public static Vector getGuestBook(){
    Vector v = new Vector();
    try{
        Connection connection = getConnection();

        Statement statement = connection.createStatement();

        // Microsoft Access INSERT
        String command = "SELECT * FROM GBOOK ORDER BY POSTDATE DESC";

        if(statement.execute(command)){
            ResultSet rs = statement.getResultSet();
            while(rs.next()) {
                GBook g = new GBook();
                g.gbookid = rs.getInt("GBOOKID");
                g.name = rs.getString("NAME");
                g.email = rs.getString("EMAIL");
                g.message = rs.getString("MESSAGE");
                g.postdate = rs.getDate("POSTDATE");
                v.addElement(g);
            }
        }
        statement.close();
        connection.close();
    }catch(SQLException e){
        e.printStackTrace();
    }
    return v;
  }
}
```

6 JSP Guest Book

The JSP code is pretty much a word-for-word translation of the PHP guest book you've already seen:

```
<%@ page import="gbook.GBook,gbook.DbHelper"%>
<html>
<head>
<title>Guest Book!</title>
<meta HTTP-EQUIV="Pragma" content="no-cache">
<meta HTTP-EQUIV="Cache-Control" content="no-cache">
</head>
<body>
<center>
<h3>Welcome to the Guest Book Page!</h3>

<p>[<a href="gbook.zip">donwload source code</a>]</p>

<%
    java.util.Enumeration enum = DbHelper.getGuestBook().elements();
    if(enum.hasMoreElements()){
%><table border="1" width="60%"
    bgcolor="#DDDDDD"
cellspacing="1" cellpadding="5">
<%
        while (enum.hasMoreElements()){
            GBook g = (GBook)enum.nextElement();
%><tr>
<td width="20%" bgcolor="#EEEEEE"
valign="top">
<nobr><%=g.name%></nobr><br>
<small><%=g.email%></small><br>
<small><%=g.postdate%></small>
</td>
<td width="80%" bgcolor="#EEEEEE"
valign="top">
 <%=g.message%>
</td>
</tr><%
        }
%></table><%
    }else{
        %><h3 style="color:red">Sorry,
            database Error.</h3><%
}
%>

<p>&nbsp;</p>

<h3>Sign this Guest Book!</h3>
<form action="index_action.jsp" method="POST">
<table border="0" width="40%" bgcolor="#DDDDDD">
 <tr>
  <td style="width:10%" align="right">Name:</td>
  <td style="width:90%">
   <input type="text" name="name" style="width:100%">
  </td>
 </tr>
 <tr>
  <td style="width:10%" align="right">
   <nobr>E-Mail:</nobr>
  </td>
  <td style="width:90%">
   <input type="text" name="email" style="width:100%">
  </td>
 </tr>
 <tr>
  <td style="width:100%" colspan="2">
   <textarea cols="60" rows="6"
   style="width:100%" name="message"></textarea>
  </td>
 </tr>
 <tr>
  <td style="width:100%" colspan="2" align="right">
   <input type="submit" value="Sign!">
  </td>
 </tr>
```

```
</table>
</form>

<p>&nbsp;</p>
<p>&copy; 2004, CIS45</p>

</center>
</body>
</html>
```

Notice that right now we use the `<%` and `%>` instead of the PHP ones (and also that we have Java instead of PHP code).

Another important thing to notice is that our JSP code has no database code at all. All we do is display results we got from some other class. There is a way to make this even cleaner using JSP tag libraries (i.e.: not have any Java code at all, just special tags that do all these things), but we don't have time to go into that. Tomcat documentation (which you should already have installed) has everything you need to build tag libraries yourself.

The submission code (the "action") is just:

```
<%@ page import="gbook.GBook,gbook.DbHelper"%><%
    GBook g = new GBook();
    g.name = request.getParameter("name");
    g.email = request.getParameter("email");
    g.message = request.getParameter("message");
    DbHelper.addGuestBook(g);
    response.sendRedirect("index.jsp");
%>
```

Note that it doesn't do any error checking, etc., and in fact has a *very serious bug*[6], which we will discuss when we deal with security a bit later in the course.

7 Setting it up

To get the sample application working, you need to place it in the `webapps`[7] directory, and then point your web-browser (after starting Tomcat of course) to `http://localhost:8080/gbook/`

8 Oracle

You don't need much to get it to run under Oracle. You already have most of the code. From the Oracle installation directory, you need to copy the JDBC "thin" drivers, and place them in the `/WEB-INF/lib` directory, as a JAR file[8].

You then need to change the connection code in `DbHelper`, to that of Oracle:

[6]What happens when someone enters a quotation character as part of their name or message?

[7]You'd have: `C:\jakarta-tomcat\webapps\gbook` or something similar.

[8]You may need to rename it to `.jar`, since it is likely a `.zip`

```
String driverName = "oracle.jdbc.driver.OracleDriver";
Class.forName(driverName).newInstance();

// Create a connection to the database
String serverName = "hostname";
String portNumber = "1521";
String sid = "sid_oracle_instance";
String url = "jdbc:oracle:thin:@" + serverName +
        ":" + portNumber + ":" + sid;
String username = "user";
String password = "password";
conn = java.sql.DriverManager.getConnection(url,username,password);
conn.setAutoCommit(true);
```

Basically what the above does is use the Oracle "thin" driver[9] to connect to a an Oracle instance running on a certain machine on a certain port, using a certain username, and certain password.

The "AutoCommit" part is so that if you disconnect without calling "commit", all your things don't get lost, but get auto-committed. By default, if you forget to "commit" everything that you've done would get un-done.

8.1 AutoNumber Issue

One of the tougher porting issues involves the fact that Oracle doesn't have `AUTO_INCREMENT` (like MySQL) nor does it have an `IDENTITY` (like Microsoft SQL Server), and it doesn't have `COUNTER` (like Microsoft Access[10]). What Oracle does have is "Sequences"[11]).

A sequence is like an incrementing variable that's maintained by the database. It can be used by stored procedures, returned, queried, etc. To create a sequence, you do:

```
CREATE SEQUENCE gbook_sequence ;
```

After that, you can select from it via a plain regular select statement:

```
SELECT gbook_sequence.NEXTVAL FROM DUAL
```

The above just gets a new sequence value (well, the "next" value). You can also get the current value:

```
SELECT gbook_sequence.CURRVAL FROM DUAL
```

Anyway, what all this means is that you need to use this sequence value whenever you're inserting records. So for example, instead of our usual insert statement, you'd actually do something like:

[9] There are 2 types of drivers: the "thin" and the non-thin OCI driver. If you need high performance and don't mind installing an Oracle client driver on the user's machine, then you can use the OCI driver.

[10] I don't know why Microsoft Access doesn't support the same SQL instructions as SQL Server—seems kind of stupid to have two different standards at a single company.

[11] Because of this, you have to use a slightly different table definition—the one that doesn't include 'auto numbers'.

```
INSERT INTO
GBOOK (GBOOKID,NAME,EMAIL,POSTDATE,MESSAGE) VALUES (
    gbook_sequence.CURRVAL,
    'John Doe',
    'jdoe@yahoo.com',
    SYSDATE,
    'And this is a message!'
);
```

Also notice that we use `SYSDATE` instead of the 'usual' `NOW()`.

9 Conclusion

And basically that's it. Once you write an application, changing the database shouldn't be too hard. Even between vastly different databases, like Microsoft Access and Oracle.