

Notes from Well House Consultants

These notes are written by Well House Consultants and distributed under their Open Training Notes License. If a copy of this license is not supplied at the end of these notes, please visit

*<http://www.wellho.net/net/whcotnl.html>
for details.*

1.1 Well House Consultants

Well House Consultants provides niche training, primarily but not exclusively in Open Source programming languages. We offer public courses at our training centre and private courses at your offices. We also make some of our training notes available under our "Open Training Notes" license, such as we're doing in this document here.

1.2 Open Training Notes License

With an "Open Training Notes License", for which we make no charge, you're allowed to print, use and distribute these notes provided that you retain the complete and unaltered license agreement with them, including our copyright statement. This means that you can learn from the notes, and have others learn from them too.

You are NOT allowed to charge (directly or indirectly) for the copying or distribution of these notes, nor are you allowed to charge for presentations making any use of them.

1.3 Courses presented by the author

If you would like us to attend a course (Java, Perl, Python, PHP, Tcl/Tk, MySQL or Linux) presented by the author of these notes, please see our public course schedule at

<http://www.wellho.net/course/index.html>

If you have a group of 4 or more trainees who require the same course at the same time, it will cost you less to have us run a private course for you. Please visit our onsite training page at

<http://www.wellho.net/course/otc.html>

which will give you details and costing information

1.4 Contact Details

Well House Consultants may be found online at

<http://www.wellho.net>

graham@wellho.net

technical contact

lisa@wellho.net

administration contact

Our full postal address is

404 The Spa

Melksham

Wiltshire

UK SN12 6QL

Phone +44 (0) 1225 708225

Fax +44 (0) 1225 707126

Java Roadmap – Beyond the Fundamentals

There's a huge range of java packages available, and many containers and associated products. Once you've learnt the fundamentals of Java, you'll want to study those extras which are relevant to your application, and that's likely to be only a small proportion of the extras. This module briefly describes some of the more popular extras that are available, to help you see where you should consider further study.

<i>Java releases</i>	4
<i>Java Runtime Environments</i>	5
<i>Other JREs</i>	11
<i>Application Programmer Interfaces (APIs)</i>	12
<i>and also</i>	14
<i>Summary</i>	15

Whatever you're going to be using Java for, you need to know the fundamentals of the language when you write Java. You'll be:

- Learning about a requirement, analysing that requirement, and coming up with a design (a model) as to how it can be implemented in Java.
- Writing classes, either by extending other classes, or from scratch, which will include methods that might be static or refer to particular instances of objects
- Using variables, assignments, conditionals, loops, and calls to other classes (your own, your colleague's, brought in or standard) within your classes to provide functionality which has already been written and is potentially going to be shared between a number of applications.
- Compiling your classes to ensure that the syntax of them is correct, and testing them (using extra, specially written test code if your main job is to write classes for others to call in their programs) by running them within a Java Virtual Machine (JVM), making trial inputs and scrutinising the outputs.
- Laying out your code so that it can be easily followed later, using descriptive variable names and adding comments describing what it is, and the major features of how it works internally. In addition, you'll be writing documentation for your user who may be another programmer who calls your classes, an application user, or a web developer who includes your work within his web site.

To gain the knowledge to perform these tasks that everyone needs to do, you'll need to know about the basic structure of Java – what it is, how the components work together – and understand the fundamentals of the language. You could learn this on one of our fundamentals training courses, or on someone else's course, or perhaps from a book or using Computer Based Training if you find such methods effective for you.

Once you've learnt the basics, you'll also need to learn about a number of Application Programmer Interfaces (API) and Environments so that you can apply your Java to meet your particular requirements. Although you'll need to learn the same basics as everyone else, which APIs and Environments you select will be highly dependent on what you'll be doing with Java. Unless you're going to be a general Java consultant or advisor, you won't need to learn everything.

The initial public and private Java courses that we run for newcomers cover pretty much the same subjects for everyone. We run different courses for those who have not programmed before to those who have, but everyone should end up having knowledge of the same ground. Beyond that initial point, we provide tailored training rather than offering more general advanced training, which would spend a great deal of time covering topics that would not be relevant to particular individuals.

2.1 Java releases

The initial release of Java, in 1995 was release 1.0. I don't think Sun realised quite what they had at the time and what a major tool Java would turn out to be. There were two environments available, stand-alone and applets, and eight packages, or bundles, of APIs.

Release 1.1 of Java expanded Java into many more application areas. The number of packages provided increased from eight to 23. As it expanded, parts of some of the original APIs were deprecated¹ as additional facilities were added that were logically expandible into the new areas that Java was tackling.

Release 1.2 added further packages, many of them now in the Java expanded (javax) package group, up now from 23 to 59 packages, and also with the addition of the Servlet environment.

The Java APIs and Environments required for running on the microprocessor in

¹ declared to be bad practice for new programs

your vacuum cleaner are very different to what's needed in the central computer of one of the "Big Four" banks. As Java grew, so it became increasingly inappropriate for all the packages and environments to be provided in a single bundle.

As from Java 1.3, Sun has restyled Java to "Java 2" and three separate editions have been announced.

- Java 2 Standard Edition (J2SE) is a middle-of-the-road bundle that will suit many applications and users.
- Java 2 Enterprise Edition (J2EE) adds a number of APIs which are going to be associated with enterprise-wide applications, for example the JDBC which is used for accessing relational databases.
- Java 2 Micro Edition (J2ME) for consumer electronics product use such as mobile telephones.

Packages (APIs) are now divided into basic packages (such as `java.lang` and `java.util`), Foundation classes which are particularly concerned with graphics, and Enterprise classes which are of particular use and concern in server-based Enterprise Edition applications.

2.2 Java Runtime Environments

The Java classes that you write will be run within a Java Virtual Machine (JVM); the presence and availability of a JVM is often said to Java Enable a platform of a piece of software. As well as the JVM, you'll need to have standard class files available on the platform on which the code is to be run. For the purpose of this section, we'll consider that the JVM, associated classes, and perhaps certain other enabling software, to be the Java Runtime Environment (JRE).

Users who will be running applications that have already been written won't need a full Java system on their computer, they'll only need an appropriate JRE. In some circumstances, the JRE will be included in another piece of software, but sometimes you will need to download it explicitly, or load it from a CD.

As well as the JRE, Java software developers will need the Software Developer's Kit (SDK) which includes compilers, etc, as well as various JREs. In earlier releases, the SDK was known as the Java Development Kit (JDK).

Let's have a look at various runtime environments:

"Stand Alone" Java programs

With J2SE, Sun provide a program called "Java" which is a Java Virtual Machine that lets you run classes as application programs. This is what you'll initially do when you're learning Java with Well House Consultants – writing a method called "main" in your class, which is where the program starts. When the main method exits, your program has been completed.

```
$ javac Greeting.java
$ java Greeting
Call 01225 708225
$ cat Greeting.java
public class Greeting {

    // Well House Consultants, 2001, "hello" for Java 2 1.3 test

    public static void main(String [] args) {

        System.out.println("Call 01225 708225");

    }

}

$
```

Figure 1 Running public class Greeting

Java Applets

If you are writing Java code to be used in conjunction with the web, you may be using one of a number of environments. Users who visit your Web site may call up a page that's going to include dynamic graphics produced by Java. The place for such graphics to be generated will be within the browser. Such pieces of Java code are known as applets, and there's a Java Virtual Machine that's capable of running applets built into recent versions of both Internet Explorer and Netscape.

An applet doesn't start up and shut down in the same way that a regular stand-alone program does, and it uses an API called the Abstract Windowing Toolkit (AWT) to handle the display and associated inputs.

Here's the source code of a simple applet, a class called "AppOne":

Figure 2 Running public class AppOne

```
import java.applet.*;
import java.awt.*;

public class AppOne extends Applet {
    private int basex, basey;
    Graphics g,goff;
    Image offscreen;
    Dimension app_size;

    public void init() {
        g = this.getGraphics();
        app_size = this.size();
        offscreen = this.createImage(app_size.width,app_size.height);
        goff = offscreen.getGraphics();
        clear();
    }

    /** Respond to mouse clicks */
    public boolean mouseDown(Event e, int x, int y) {
        basex = x; basey = y;
        return true;
    }

    /** Respond to mouse drags */
    public boolean mouseDrag(Event e, int x, int y) {
        shape(basex,basey,x,y);
        g.drawImage(offscreen,0,0,this);
        return true;
    }

    /** convenience method to erase the scribble */
    public void clear() {
        goff.setColor(Color.black);
        goff.fillRect(0, 0, bounds().width, bounds().height);
    }

    void shape(int x1, int y1, int x2, int y2) {
        clear();
        int xlow = x1; int xhigh = x2;
        if (xlow > xhigh) {
            xlow = x2; xhigh = x1; }
        int ylow = y1; int yhigh = y2;
        if (ylow > yhigh) {
            ylow = y2; yhigh = y1; }
        goff.setColor(Color.yellow);
        goff.fillRect(xlow,ylow,xhigh-xlow,yhigh-ylow);
    }
}
```

You'll notice there's no main method, so there will be no way to run it from the command line. Instead, compile it and load the class file onto your Web server, and have your user access it through a Web page.

The Web page will be written in HTML, for example:

```
<html>
<body bgcolor=white>
<H1>Demonstration of a Java Applet</H1><br><br>
<applet code="AppOne.class" width=400 height=400></applet>
<br>
<H4>As well as the Applet, the web page will include other
  elements such as HTML to make it into a complete page.</H4>
</body>
</html>
```

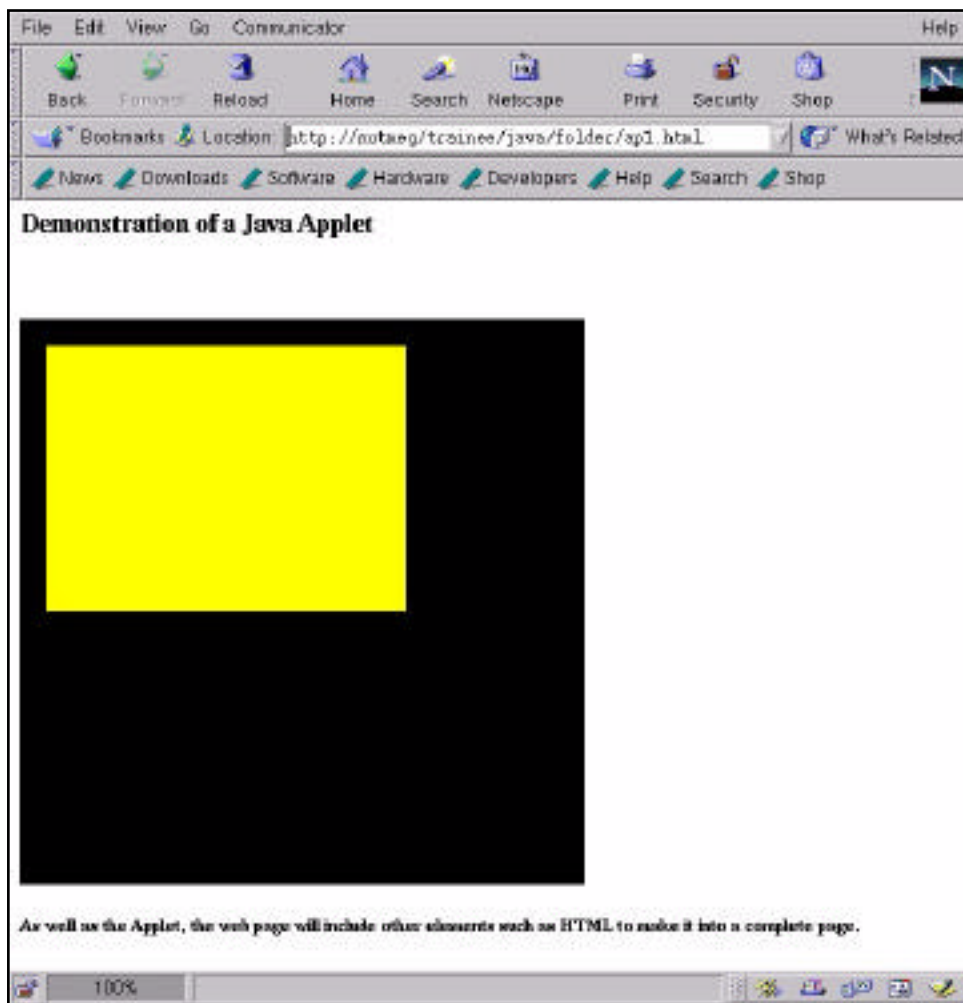


Figure 3 An example of the display produced by AppOne.class

When the Web user calls up this page on his browser, the browser will request the class file from the server as well – much as it would with an embedded image – and will run the class (via certain methods whose names are pre-determined, such as **init**, **paint** and **mouseDrag**) on the JVM within the browser.

As well as the JVM within web browsers, Sun provide a stand-alone JVM called "AppletViewer" which allows you to test an applet without the need to upload the applet and associated Web page to a Web server.

Java servlets

In a web-based application, dynamic graphics are most naturally generated within the browser. You would have major bandwidth and resource problems if you tried to do it any other way. Certain other web applications are most naturally provided on the web server computer; a good example is a Web page that looks up some information in a large database, as there's no way that you want the whole database contents to be sent to the browser for the relevant information to be extracted by an applet.

A piece of Java code run on a Web server in such a situation is possibly a servlet.

When a Web user calls up certain URLs, a Java method is run on the server as the result of running the program that's returned to the browser. The output from the servlet – the thing which contains the method that's run when the page is called – must be in HTML or some other format that the browser understands (the browser needs to be told what format's used, too!).

Here's an example of the source code of a java servlet:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * The simplest servlet.
 *
 */

public class ServOne extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String host = System.getProperty("os.name");

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Simplest Servlet</title>");
        out.println("</head>");
        out.println("<body bgcolor=white>");
        out.println("<h1>Server running the "+host+" Operating System</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Again, no **main** method, so this can't be run from the command line. The **doGet** method is one of the names specified in the Servlet API, and it converts inputs in an **HttpServletRequest** object to outputs using an **HttpServletResponse** object.

The output from running this servlet on our Linux box was:

```
<html>
<head>
<title>Simplest Servlet</title>
</head>
<body bgcolor=white>
<h1>Server running the Linux Operating System
</h1>
</body>
</html>
```

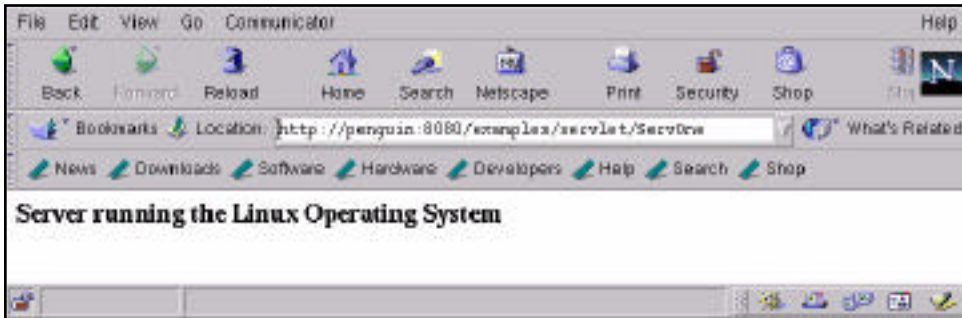


Figure 4 The simplest servlet as interpreted by the browser

The Java Virtual Machine that's running on the web server is most commonly "Tomcat", which is a part of the Jakarta Project at the Apache organisation. Tomcat has largely replaced JServ, which was the previous Apache offering the provided servlet support. For testing purposes, Sun also provide a utility program called `servletrunner` if neither Tomcat nor JServ are easily available to you.

Java Server Pages

If you come from a programming background, you'll be quite happy to run a program to generate a web page. But it does seem an awful lot of work to do if you simply want to enclose one piece of changing data within an otherwise static page.

Java Server Pages (JSP) let you start off with your web page, and then embed Java source code within the web page.

When a user calls up the Web page that includes the Java code, the Web server both compiles and runs that Java code, replacing the Java code in the Web page that it read from the disk with the result of running that Java code before the page is passed onward to the browser.

Here's an example of a piece of code that might be held on the server:

```
<html>
<%@ page session="false"%>

<body bgcolor="white">
<jsp:useBean id='now' scope='page' class='dates.JspCalendar' type="dates.JspCalendar" />
<%
    String osname = System.getProperty("os.name");
%>

<font size=4>
<ul>
<li>Day of week: <jsp:getProperty name="now" property="day"/>
<li>Week Of Year: <jsp:getProperty name="now" property="weekOfYear"/>

<%
```

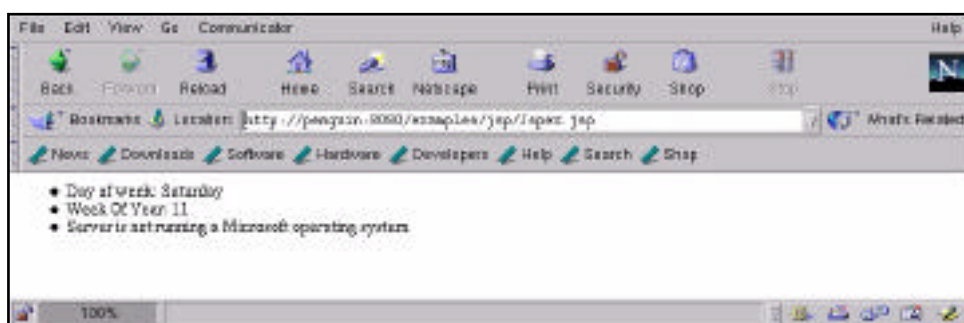
```

    if (osname.startsWith("MS")) {
%>
    <li> Operating System: Microsoft
<%
    } else {
%>
    <li> Server is not running a Microsoft operating system
<%
    }
%>
</ul>
</font>

</body>
</html>

```

Figure 5 Browser displaying piece of code



When it's called up by the browser, that's what the server reads; it's translated into pure HTML for sending on to the browser:

```

<html>
<body bgcolor="white">
<font size=4>
<ul>
<li>Day of week: Saturday
<li>Week Of Year: 11
<li> Server is not running a Microsoft operating system
</ul>
</font>
</body>
</html>

```

You'll notice that the code looks rather different to all our earlier examples; JSPs are implemented internally using servlets,¹ but the code – the classes – are incomplete within the Web page. As the JSP is actually run, a complete servlet class is made up by Tomcat, that class is compiled, and output is cut into the response page as appropriate.

2.3 Other JREs

As well as programs, applets, servlets and JSPs, you may come across a number of other Java runtime environments as you delve in still deeper. The interface to a micro device is likely to be different, and EJB and RMI object servers also provide their own runtime environment.

¹ and so you won't be surprised to learn that Tomcat provides the virtual machine

2.4 Application Programmer Interfaces (APIs)

Every Java application makes use of classes supplied with the Java language – application programmer interfaces – to perform tasks which need to be done across a wide number of applications.

Perhaps the most commonly used API of all is the `java.lang` package. It's used so frequently that, as a special case, you don't even need to tell Java that you're using it. There are other APIs that are used frequently too; for example, `java.io` is used in any applications/JSPs/servlets that read or write local files. Still further APIs are available which apply to only certain types of applications.

Before you can use any API, you need to have some understanding of what role it's there to fulfil. Once you understand that role, you really don't need specialist training on many of the APIs. The `java.math` package, for example, provides you classes which can handle numbers larger than the primitive types in Java, and once you know that (and have a look at the manual), it's clear how to use the classes in question.

Certain other APIs need a few more words of explanation, and yet others are sufficiently complex to merit training in their role and how to use them.

Useful "basic" classes

java.net – a package for handling network communication from a java class. **java.net** includes high-level methods that allow you to collect a file of information from a Web server using its URL in a single call, right through to socket programming methods which allow you to write your own client or server.

java.text – text formatting, date handling and a variety of similar facilities that extend the String handling capabilities of Java beyond what's provided in the basic language itself and classes such as **java.lang.String**.

java.util – a series of classes that provide additional programmer's constructs such as linked lists, stacks and hash tables.

The Java foundation classes

The Java foundation classes provide the graphic user interfaces that Java uses in applets, and may use in stand-alone applications; they're not applicable to server-based environments.

The first API within the Java foundation classes is the Abstract Windowing Toolkit (AWT). It dates from the early days of Java and provides the components necessary to build up and manage a GUI. There are really three main parts to the AWT:

- Components or Widget classes, which allow you to define and manipulate the elements of a GUI.
- Geometry/Window managers, which allow you to control how widgets are laid out in relation to each other.
- Event handlers, which define what's to happen when a slider is dragged, a button pressed, or a window exposed.

If you're looking to produce true graphics, such as filled polygons, circles, lines and the like, then you'll use a series of method calls onto a canvas component in the AWT.

With the expanding application of Java after the early releases, further higher-level GUI facilities were required, and the Swing API (**javax.swing**) was added at release 1.2. The Swing classes provide higher level facilities than the AWT, doing in a single class or API what would have taken a great deal of code under the AWT. As an example of Swing, perhaps the most useful of all the classes is the `JTable`, which lets you draw up a graphic to display a complete table of data, rather than having to use a large number of components and geometry managers under the AWT.

The Threads API

Strictly speaking, threads are a part of the `java.lang` package and always available to you. The number of classes or methods that relate to threads is deceptively small for what is a very powerful facility.

Conventional programming involves writing a piece of code that is executed from the start and carries on through to the end. Jumps to subroutines or functions¹ will be made, and the programmer will be able to say "we're at this point in the code" or "we're there" at any time.

What happens if you have a piece of code that you want to do two things with at the same time? For example, if you're writing a server, you want it to do things when it receives a new connection: you want it to handle the connection made and you want it to monitor for further connections.

The threads API provides an answer to these needs. You can create a whole series of thread objects, each of which is potentially running at the same time, and you can assign separate threads to performing different tasks.

A lot of thought² needs to go into the use of threads; you need to consider how threads are to be created, how they're to work in co-operation with each other on what might be the same data, who's to yield to whom, how (and if) they're to come back together again upon completion of their jobs, and more. Although the API appears to be simple at first, there are complete books on the use of threads in Java.

JDBC

JDBC provides Java Database Connectivity, allowing access to most database systems using Structured Query Language (SQL).

SQL isn't as standard as you might guess (or wish), so classes that use JDBC need to start out by loading an appropriate JDBC driver class. Once they've done so, reasonably portable code can be written provided that you don't use the extended capabilities of one particular database.

Sun maintains a list of available drivers on their Web site; at present it lists about 140 different sources of drivers covering some 70 or so different databases, right through from the obvious Oracle, Sybase and Microsoft SQL Server to some really obscure products.

RMI

Remote Method Invocation (RMI) is a scheme for creating and using remote objects. Using servlets, the `java.net` class, or JDBC that we've already mentioned, it's practical to run a Java class on one computer that will interact with another computer. RMI is another approach – a distributed object system.

Let's take an example in banking. On the bank clerk's desk, his computer is running a Java "teller" class which lets him step through a whole series of options, such as selecting loan interest calculators to demonstrate options to potential customers, and much more.

At some point, the customer asks how a particular type of loan would work for her, and the clerk calls up the appropriate option in his application. Now that customer's account isn't held in full detail on the clerk's system, so RMI within his application contacts a network server and calls for a remote object. All the data concerning the account remains on the server and there's no question of it getting out of step if someone else calls up the same account. But each method call causes traffic – the input parameters and the returned information – to be passed over the network.

RMI is a Java-only remote object system, which means that it can provide more

¹ or methods if you're writing in an Object Oriented language

² and no small amount of learning!

facilities and be more efficient than more general-purpose object brokering systems such as CORBA. If you require language independence so that you can program a client in Java and a server in another language (or vice versa), there is a Corba binding available for Java.

Java Beans

Naming conventions for variables and methods help programmers on most larger projects read and write code more quickly and make fewer mistakes – they're like built-in comments. Java takes this several steps further. If you use certain conventions in the naming of methods that you write (and their calling sequences) then you have what's known as a Java bean.

It's possible to look inside a Java class file and, with an appropriate program, work out the names, calling parameters and return types of the methods it contains. If you've stuck with the bean conventions, software development environments and other pieces of code can usefully work out how to use your methods and apply them automatically (programmers tools).

If you write a class that conforms to the bean rules, that class will be a Java bean. If you wish, you can go further and provide additional related classes (with related names, conventions, etc.) which can be used by programmer's tools in the possession of the programmer who's using your classes.

Enterprise Beans

Enterprise Java Beans (EJBs) are much more than Java beans. Part of the Enterprise Edition, they make use of RMI and JavaBean technologies – they're remote objects on a server.

When writing a client, you have to add the API to your Java install (the `javax.ejb` class), and you need to install and run an EJB-enabled server program on the server system.

The EJB server provides remote object – remote beans – in a container on the server that your clients can access through client code. Database handling is typically done from the EJB server, which provides transaction managements, sessions and an appropriate degree of both co-operation and isolation between client requests.

JNDI

For access to naming services (such as DNS), for Enterprise Java beans, you'll use the Java Naming and Directory Interface (JNDI) API. The naming system can also be used for files, RMI objects, NIS lookups and other naming conversions.

2.5 and also ...

This module has mainly considered the releases, runtime environments and APIs that are associated with Java. Assuming a knowledge of the fundamentals of programming in Java, it's given you an insight into what's on offer as you advance into the more practical application of the language.

There are a number of other advanced topics that you might need to be aware of in Java if you're writing certain types of classes.

Synchronized

The synchronized keyword may be used to describe a Java method¹ if you want to give the programmer manipulating an object a lock on the object in question while that code's being run. It's particularly used in threaded applications, where two transactions being run at the same time on the same object could cause a problem with

¹ blocks of code can also be synchronized to an object

data corruption. For example, you wouldn't want to sell the same theatre seat at the same performance to two different individuals just because they both checked its availability at the same time.

Serializable and transient

While you're using an object, the various variables associated with it are held in the computer memory that's being managed by your JVM. The JVM is quite a clever piece of software, and it's highly unlikely that the object occupies sequential memory locations, or that you could move it around in memory without changing the internal values of some pointers.

So imagine that you want to save an object's variables to a disk file. How are you going to do so? Saving will require the JVM to select which information applies to the particular object, and restoring later won't be easy as it's probable that the object will be physically placed in different memory locations.

If you declare a class to be serializable, then Java retains extra information in the JVM which allows the information to be written to a stream. In other words, converted to a series of bytes that can be later read back and understood. The coding is easy¹ as Java itself is doing the extra work.

Within objects, you may retain temporary variables (perhaps for speed of operation) which don't need to be saved to disk. You may describe such variables as transient, in which case they won't be included in the serialised stream. It's then up to you to re-calculate them as necessary when you restore the object.

As well as saving objects to disk, serialization is used for passing objects around your network; you'll find that it's a required part of RMI and EJB, amongst others.

Please be aware that you shouldn't use the serializable interface for the long-term storage of objects. If you attempt to do so, you're likely to find that versions of classes have changed and you have compatibility problems.

jars

Applications consist of multiple classes, sometimes large numbers of classes. As an aid to distribution, many files (both class and other) can be combined into a single disk file, known as a jar, and then distributed as necessary. Using jar (which stands for Java Archive) files, the number of network connections that have to be made can be reduced, as can the load on the operating system's directory and file structure. The chance of having one or two files get out of step (i.e. be missing or the wrong release) is also reduced if the originator creates and distributes a single jar.

2.6 Summary

It's Sun's stated intent that Java technology be deployed widely into a huge variety of application areas. If you visit their Java Web site, you'll usually find that they're announcing a new environment or API to suit some particular market niche. Some such environments or APIs have little associated training requirement, just read the manual. Others may benefit from specialist training ... the field is so big that no one training company can realistically hope to cover all such topics.

¹ although you must remember that objects used within others must also be serializable



Exercise

License

*These notes are distributed under the **Well House Consultants Open Training Notes License**. Basically, if you distribute it and use it for free, we'll let you have it for free. If you charge for its distribution of use, we'll charge.*

3.1 Open Training Notes License

Training notes distributed under the **Well House Consultants Open Training Notes License** (WHCOTNL) may be reproduced for any purpose PROVIDE THAT:

- This License statement is retained, unaltered (save for additions to the change log) and complete.
- No charge is made for the distribution, nor for the use or application thereof. This means that you can use them to run training sessions or as support material for those sessions, but you cannot then make a charge for those training sessions.
- Alterations to the content of the document are clearly marked as being such, and a log of amendments is added below this notice.
- These notes are provided "as is" with no warranty of fitness for purpose. Whilst every attempt has been made to ensure their accuracy, no liability can be accepted for any errors of the consequences thereof.

Copyright is retained by Well House Consultants Ltd, of 404, The Spa, Melksham, Wiltshire, UK, SN12 6QL - phone number +44 (1) 1225 708225. Email contact - Graham Ellis (graham@wellho.net).

Please send any amendments and corrections to these notes to the Copyright holder - under the spirit of the Open Distribution license, we will incorporate suitable changes into future releases for the use of the community.

If you are charged for this material, or for presentation of a course (Other than by Well House Consultants) using this material, please let us know. It is a violation of the license under which this notes are distributed for such a charge to be made, except by the Copyright Holder.

If you would like Well House Consultants to use this material to present a training course for your organisation, or if you wish to attend a public course is one is available, please contact us or see our web site - <http://www.wellho.net> - for further details.

Change log
Original Version, Well House Consultants, 2004

Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____

License Ends.