

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

Packages

Java applications often comprise a very large number of classes. In order to provide for management of all these classes, they are arranged into groupings known as packages, with each package containing a logical group of related classes.

<i>Overview</i>	4
<i>Package directory structure</i>	6
<i>Importing classes from a package</i>	10
<i>Introduction to standard packages</i>	11

2.1 Overview

As we start to develop more complex programs, we'll start to use more and more classes. Some classes will be from standard libraries; others will be classes which we develop ourselves.

After we get to the point of having classes all over the place, we'll want to provide some sort of organisation.

Let's return to our card decks.

We already have a class of object for a card pack, but we might also develop other classes and objects for a hand, and sub-objects for individual cards. Immediately, the number of different classes starts to multiply!!

The solution Java provides is to **package** classes.

Let's take the example *axford.java* that we studied when we first looked at classes:

```
// Well House Consultants2004.
/** overloaded constructor*/
public class axford
{
    public static void main(String[] args)
    {

        System.out.println("Regular pack");
        pack_4 my_pack = new pack_4();
        System.out.println(my_pack.ncards);
        my_pack.print();

        System.out.println("Double pack, 5 jokers");
        my_pack = new pack_4(2,5);
        System.out.println(my_pack.ncards);
        my_pack.print();

        System.out.println("Short pack, 24 cards removed");
        my_pack = new pack_4(1,0,6);
        System.out.println(my_pack.ncards);
        my_pack.print();
    }
}
```

This worked well enough, but the *pack_4* class was left cluttering up the current directory. Let's put it into a package we'll call:

the *packagecard_pkg*

the *classpack*

Here are the changes we'll make to the **main** method:

```
seal% diff axford.java sopworth.java
5c5
< public class axford
---
> public class sopworth
```

Within the *card_pkg* package call the method *pack* (rather than call the method *pack_4*) each time we create an instance of class *card_pkg.pack*.

```

11c11
< pack_4 my_pack = new pack_4();
---
> card_pkg.pack my_pack = new card_pkg.pack();
16c16
< my_pack = new pack_4(2,5);
---
> my_pack = new card_pkg.pack(2,5);
21c21
< my_pack = new pack_4(1,0,6);
---
> my_pack = new card_pkg.pack(1,0,6);

```

We don't need to make any other changes since any references to *my_pack* such as in:

```

System.out.println(my_pack.ncards);
my_pack.print();

```

are now very clearly referring to a *pack* type object in the *card_pkg* package.

We now need to amend the *pack_4* class that was used by the *axford.java* main method.

Firstly, it needs to be renamed from the *pack_4* class used by *axford.java* into the *pack* class that we have chosen to use in *sopworth.java*.

Secondly, we need to tell the compiler that this class is in the package called *card_pkg*.

Thirdly, we need to ensure that all variables and methods which are going to be used external to the package are declared as **public**. (If we fail to do this, such variables will be available only to other classes within the package. By default, all classes are assumed to be in the default package. This happened with both *axford* and *pack_4* so we did not need the public declaration).

Here is our instruction to the Java compiler to include this class in the *card_pkg* package:

```

seal% diff ../pack_4.java pack.java
2a3,4
> package card_pkg;
>

```

The first of many renamings from *pack_4* to *pack* (we won't report on them all!):

```

5c7
< public class pack_4
---
> public class pack

```

The making public of the *ncards* instance variable, and of the constructors, so that they can be seen outside the package:

```
11c13
<   int ncards;
---
>   public int ncards;
15c17
<   pack_4(int ndecks, int njokers, int shorten)
---
>   public pack(int ndecks, int njokers, int shorten)
```

2.2 Package directory structure

One of the purposes of packaging classes was to allow us to structure our directories.

Java packages should be stored in subdirectories of the same name as the package. Remember that, already, the class file¹ must be stored in a file of the same name as the class so that our *pack* class within the *card_pkg* package must be stored in the file:

```
card_pkg/pack.java
```

relative to the calling class/method.

In our example (and on the author's system) the files pertinent to the current example are:

```
/export/home/graham/JF/sopworth.class
```

```
/export/home/graham/JF/card_pkg/pack.class
```

Complete listings on the *.java* files follow for reference ...

```
/export/home/graham/JF/sopworth.java
```

```
// Well House Consultants2004.

/** Packaged card pack*/

public class sopworth
{
    public static void main(String[] args)
    {

        System.out.println("Regular pack");
        card_pkg.pack my_pack = new card_pkg.pack();
        System.out.println(my_pack.ncards);
        my_pack.print();

        System.out.println("Double pack, 5 jokers");
        my_pack = new card_pkg.pack(2,5);
        System.out.println(my_pack.ncards);
        my_pack.print();

        System.out.println("Short pack, 24 cards removed");
        my_pack = new card_pkg.pack(1,0,6);
        System.out.println(my_pack.ncards);
        my_pack.print();

    }
}
```

¹ byte code

```
/export/home/graham/JF/card_pkg/pack.java

// Well House Consultants2004.

package card_pkg;

/** Card pack / hand dealer*/
public class pack
{

// Instance Variables
int [] cards;
public int ncards;

// Constructors
public pack(int ndecks, int njokers, int shorten)
{
    ncards = (((13 - shorten) * 4 ) * ndecks) + njokers ;
    cards = new int [ncards];
    int n_at = 0;
    for (int k=0;k<njokers;k++)
        cards[n_at++]=0;
    for (int i=0;i<ndecks;i++) {
        for (int k=shorten;k<13;k++) {
            for (int j=0;j<4;j++) {
                cards[n_at++]=j+4*k+1;
            }
        }
    }
}

public pack()
{
    pack temp = new pack(1,0,0);
    ncards = temp.ncards;
    cards = temp.cards;
}

public pack(int ndecks)
{
    pack temp = new pack(ndecks,0,0);
    ncards = temp.ncards;
    cards = temp.cards;
}

public pack(int ndecks,int njokers)
{
    pack temp = new pack(ndecks,njokers,0);
    ncards = temp.ncards;
    cards = temp.cards;
}
}
```

```
// method to access data
public void print()
{
    for (int k=0;k<ncards;k++)
    {
        System.out.print(" " + cards[k]);
        if ((k+1) % 10 == 0 || k == ncards-1)
            System.out.println();
    }
}
}
```

Exercise

Take your weather station class and create a package for it.

- Update your test harness so that it now references the class within the package

Our example answer is trombone (test harness)

Our example answer is serpent (package)

Our example answer is rattle (class)

Should run identically to the previous example:

Sample

```
seal% java trombone
Stations defined:

Tiree
Ronaldsway
Channel light vessel automatic
Greenwich automatic
Total Weather Station Count: 4

Please give name of interestRonaldsway
Matched Ronaldsway
seal%
```

File structure should look like:

```
seal% ls -l trombone* serpent
-rw-r--r--  1 graham  wellho 1432 Feb  7 03:09 trombone.class
-rw-r--r--  1 graham  wellho 1441 Feb  7 03:09 trombone.java

serpent:
total 8
-rw-r--r--  1 graham  wellho 1107 Feb  7 03:08 rattle.class
-rw-r--r--  1 graham  wellho 1870 Feb  7 03:07 rattle.java
seal%
```

For Advanced Students

Update your advanced test harness as well.

2.3 Importing classes from a package

We can use classes and methods from a package. All we have to do is to give the package name as we create new instances or reference methods where there is no instance variable to tell Java which package and class to use.

But perhaps we would prefer not to have to state in which package the method is to be found. Perhaps we would like to have Java find the methods for us, or to tell Java to use all the methods within a package without explicitly naming them all the time.

Yes we can.

We can use **import** statements to pull packages and classes into our methods. For example, to bring in all classes in the *card_pkg* package:

```
import card_pkg.*;
```

We can then reduce the code we need from:

```
card_pkg.pack my_pack = new card_pkg.pack();
```

to:

```
pack_pkg my_pack = new pack_pkg();
```

The pack class is unaltered. The new main method:

```
// Well House Consultants2004.

/** Packaged card pack*/

import card_pkg.*;

public class sherston
{
    public static void main(String[] args)
    {

        System.out.println("Regular pack");
        pack_pkg my_pack = new pack_pkg();
        System.out.println(my_pack.ncards);
        my_pack.print();

        System.out.println("Double pack, 5 jokers");
        my_pack = new pack_pkg(2,5);
        System.out.println(my_pack.ncards);
        my_pack.print();

        System.out.println("Short pack, 24 cards removed");
        my_pack = new pack_pkg(1,0,6);
        System.out.println(my_pack.ncards);
        my_pack.print();

    }
}
```

2.4 Introduction to standard packages

There are various sorts of packages you may wish to use:

- Packages unique to an application
- Packages shared by closely linked applications
- Packages shared more widely
- Commercial packages
- Packages supplied as a standard part of Java
- Packages so fundamental to Java that they are always needed

The package that we used in our earlier section may well be unique to one application, or at most, be shared between a few closely linked applications.

In that circumstance, it is fine being located in a subdirectory of the directory which includes the application. In other circumstances, though ... with more widely shared.

The CLASSPATH environment variable (CLASSDIR in Java 1.0) is used to tell Java where to look for the packages you call up – specify a colon or semicolon separated list.

Setting CLASSPATH is operating system (and shell) dependent. Here are some examples:

- Unix, C shell:

```
setenv CLASSPATH ./home/java/lib:/usr/local/lib/java
```

- Unix, Bourne or korn shells:

```
CLASSPATH=./home/java/lib:/usr/local/lib/java
```

- Windows 98, NT, 2000, XP, etc.:

```
set CLASSPATH = .;c:\java\lib;c:\lib\java
```

In these examples, you might expect to find all your personal packages in the current directory, the workstation-wide packages in */home/java/lib*, and the site-wide packages in */usr/local/lib/java*.

As you learn more about Java, you'll see how classes and packages can also be held in *.zip* and *.jar* files, compressed and bundled and be automatically extracted using the CLASSPATH.

There are a considerable number of standard packages available. Java started with eight standard packages in release 1.0, and that grew to nearly 150 in Java 2 1.4, comprising thousands of classes and tens of thousands of methods. All these standard packages are available for you to import..

Their use ranges from applet support¹ through maths functions to the windowing toolkit and network operation support.

As you continue to learn about Java, you'll find that much of that learning is about standard packages, and the classes and methods they contain. You may well have already seen examples that include:

```
import java.lang.Math;
import java.lang.System;
```

although all of the classes in *java.lang* are automatically available to you anyway.

The class we provided for you to read from the keyboard includes:

```
import java.io.*;
```

¹ rather than the applications we have been writing so far

Exercise

Import the package that you wrote in the previous exercise so that there is no longer the need to specify the package name explicitly at all references.

Our example answer is flageolet

Sorry folks ... for all that work, *flageolet* will look exactly the same!

For Advanced Students

Study the `java.text.DecimalFormat` class provided with Java 1.1. Extend the answer above to print out the temperature at the selected weather station to two decimal places, no matter what accuracy was used to set it up.

Our example answer is glockenspiel

Sample

```
seal% java glockenspiel
Stations defined:
Tiree
Ronaldsway
Channel light vessel automatic
Greenwich automatic
Total Weather Station Count: 4
Please give name of interestTiree
Matched Tiree
Temperature 31.5
Let's format that:
Temperature 31.50 degrees ← Note
seal% java glockenspiel
Stations defined:
Tiree
Ronaldsway
Channel light vessel automatic
Greenwich automatic
Total Weather Station Count: 4
Please give name of interestRonaldsway
Matched Ronaldsway
Temperature -17.6
Let's format that:
Temperature 17.60 deg below ← Note
seal%
```

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 _____

Updated by: _____ on _____

License Ends.