MODULE 5p - Packages

DIRECTORY STRUCTURE

A simple strategy for organising one's Java programs is to set up a directory java in your home directory and group Java source files into various subdirectories of the java directory:

```
/....
  |
  +--fhk1
       |
       +--java
            |
            +--funnies
            |
            +--jiffies
            |
            +--tests
```

For anything serious (especially Remote Method Invocation) the local arrangements make it expedient for any user-set-up packages to be set up in subdirectories which are arranged thus:

```
/....
  |
  +--fhk1
       |
       +--public_html
              |
              +--java
                   |
                   +--classes
                         |
                         +--fhk1
                              |
                              +--box
                              |
                              +--parcel
                              |
```

```
                                +--sack
```

Here the frivolous subdirectories box, parcel and sack are the names of packages, collections of Java class files.

Set up this directory structure NOW down as far as:

```
    /..../<your-id>/public_html/java/classes
```

There is no need to go beyond classes just yet.

A FIRST PACKAGE

The source files for your packages can usefully be in one of your subdirectories of java (such as jiffies in the first structure shown above). Starting from your home directory, change to the jiffies directory:

```
$ cd java/jiffies
```

Now key the following source code (with fhk1 replaced by your own identifier) into the file Missive.java

```
    package fhk1.sack;

    public class Missive
     { public String jack()
        { return "JACK";
        }

       protected String jill()
        { return "JILL";
        }
     }
```

Note the package statement at the beginning. This directs that the associated class file is to go in the package fhk1.sack (and the directories fhk1 and sack will be set up automatically first time given that they don't already exist).

COMPILE THE SOURCE CODE

To compile the source code in Missive.java AND ensure that the resultant class file ends up in the correct place go:

```
$ javac -d ~/public_html/java/classes Missive.java
```

Assuming that the directory ~/public_html/java/classes is empty (just before this command is given) the command notices:

1. (after -d) the destination ~/public_html/java/classes

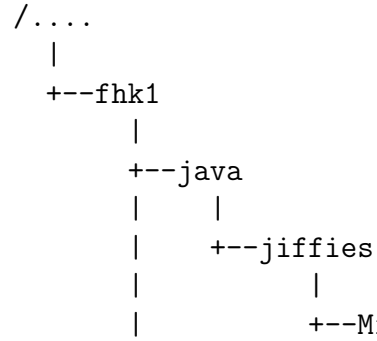
2. (in Missive.java) the package fhk1.sack

The latter is translated into fhk1/sack and a directory fhk1 is created in ~/public_html/java/classes and another directory sack is created in this new directory.

The command then compiles Missive.java and puts Missive.class into the overall directory ~/public_html/java/classes/fhk1/sack

If a second similar command is given to set up another class file in the same package, the command would note that fhk1/sack already exists and would merely compile the new class file and then put it in the directory ~/public_html/java/classes/fhk1/sack alongside the Missive.class file.

The file structure at this stage should be:



```
          |
          +--public_html
                  |
                  +--java
                       |
                       +--classes
                             |
                             +--fhk1
                                  |
                                  +--sack
                                       |
                                       +--Missive.class
```

The package has only one class file in it at the moment.

USING THE PACKAGE

Still in the directory .../java/jiffies key the following source code (again replacing fhk1 with your own identifier) into the file PackTest.java

```
import fhk1.sack.Missive;

public class PackTest
 { public static void main(String[] args)
    { Missive ape = new Missive();
      System.out.println(ape.jack());
    }
 }
```

The import statement gives only the latter part of the path leading to class Missive. The earlier part of the path is supplied by a Unix variable CLASSPATH which includes ~/public_html/java/classes Equipped with CLASSPATH and the import statement Java can find Missive.

Compile PackTest.java locally:

```
$ javac PackTest.java
```

The compiled class file is said to be in an anonymous package (which is effectively the directory jiffies) and can now be run:

```
$ java PackTest
```

The ape object invokes the method jack to yield the output:

```
JACK
```

THE EFFECT OF THE VISIBILITY MODIFIER protected

In class Missive the method jack is heralded by the modifier public which means that within any instance of Missive the method jack is visible even if the instance is in a different package.

Now change the System.out.println statement in PackTest.java to:

```
    System.out.println(ape.jill());
```

An attempt to recompile PackTest.java will now fail:

```
$ javac PackTest.java
```

The error message includes a complaint about 'No method matching jill() found in class fhk1.sack.Missive.'

This is because the method jill() is heralded by the visibility modifier protected which means that within an instance of Missive which is in a different package the method jill() is not visible.

The rules specify that the method jill() WILL be visible as an inherited method in an instance of a sub-class of Missive provided that the sub-class is in the same package as that in which the instance is created.

In the present case, modify the source PackTest.java thus:

```
    import fhk1.sack.Missive;

    public class PackTest
     { public static void main(String[] args)
        { SubMissive ape = new SubMissive();
          System.out.println(ape.jilly);
```

```
    }
}

class SubMissive extends Missive
{       String jilly = this.jill();
}
```

Here ape is an instance of SubMissive which contains a data field jilly and two methods jack() and jill() inherited from the parent class Missive. The String data field jilly is initialised via the inherited method jill() by the call this.jill() which should return JILL.

Compile the new version of PackTest.java:

```
$ javac PackTest.java
```

There should be no complaints. Now run the compiled version:

```
$ java PackTest
```

This should yield:

```
JILL
```

Note that the data field jilly has a blank visibility modifier (the blanks are not essential!). This default case forces jilly to have 'package' visibility (but note 'package' cannot be used explicitly as this keyword is used in package statements to mean something else).

PackTest.java would readily compile if jilly were (unnecessarily) made MORE visible by heralding it with protected (or even public) but if it were heralded with private then PackTest.java would not compile.

KEEPING ALL CLASSES IN THE SAME PACKAGE

For completeness it is worth verifying the rule that says that there would have been no trouble with protected jill() had class Missive been in the same package as class PackTest so, as an experiment, delete the package statement from Missive.java so the source file is now:

```
public class Missive
```

```
    { public String jack()
       { return "JACK";
       }

      protected String jill()
       { return "JILL";
       }
    }
```

Now recompile locally:

```
$ javac Missive.java
```

Next remove the import statement from PackTest.java and restore the remainder of the source file to its original form:

```
    public class PackTest
     { public static void main(String[] args)
        { Missive ape = new Missive();
          System.out.println(ape.jill());
        }
     }
```

Now recompile PackTest.java:

```
$ javac PackTest.java
```

The compiled version will run happily:

```
$ java PackTest
```

This yields:

```
JILL
```

FINAL VERSION

Alternatively, the two classes could be brought together into a single

source file:

```
    public class PackTest
     { public static void main(String[] args)
        { Missive ape = new Missive();
          System.out.println(ape.jill());
        }
     }

    class Missive
     { public String jack()
        { return "JACK";
        }

       protected String jill()
        { return "JILL";
        }
     }
```

Notice that only one class in a source file may be declared public and that has to be the class whose name is given to the file itself. In this case the file is PackTest.java so class PackTest is declared public and class Missive is not.

Compile again:

```
$ javac PackTest.java
```

Note that in both this and the previous experiment, the method jill() could have been given the default package visibility. Indeed that would be the approved style.

If the method jack() were accessed via ape it too would need at least package visibility. If it were use only in class Missive it could have private visibility.

Run this final version:

```
$ java PackTest
```

The result, as before, is:

```
JILL
```