Why packages?

Keep your files in order

Keeping your hard drive organized

Imagine if you saved all your files in your desktop folder of your computer. The folder would probably fill up pretty quickly with all sorts of documents, images, sound files, video files, emails, programs, etc, etc.

Now, finding one particular file in this mess would not be easy. Therefore it's quite common to organize your files in folders with hierarchical and descriptive names like:

```
/home/username/music/classical/mozart/
/home/username/music/classical/bach/
/home/username/music/classical/beethoven/
/home/username/music/pop/abba/
/home/username/music/folk/hamza-el-din/
```

Source code files should also be organized

In a larger application there is normally a large number of files. It is common to organize the files according to some logical order like a feature of the application, or a part of the application.

For instance the Java files for the classes etc handling the file system (saving and reading files to and from the file system etc) could be organized and collected in one directory, and the Java files for the graphical user interface of the application could be organized and collected in a different directory.

This has several benefits.

Benefits of using packages

- Classes get a longer, more descriptive name
  - java.lang.ref.Reference
  - javax.naming.Reference
  - javax.xml.crypto.dsig.Reference
- Classes can share name but have different prefix (see above)
- It’s easier for us to find a source code file if the directory names are descriptive
- Packages can provide some sharing of information between classes in the package, while hiding the same information from other classes

Real applications do use packages

If you think packages seem confusing and hard to use, you should know that there are no “real” Java applications without packages, so you must learn how to use them.

The exercises will help you understand how to create and use packages.

Remember, packages are just like relative paths - it’s a help for the compiler and the runtime to find classes used in other classes.

Recap of the package functionality

Using an import statement like this:

```
import org.myproject.text.TextUtilities;
```

Is just a hint to the compiler and the runtime system meaning:

When we say TextUtilities in this class, we mean the class in the package org.myproject.text . You will find it in a directory with this relative path:

```
org/myproject/text/
```

In the source code for TextUtilities there is a package statement:

```
package org.myproject.text; // I'm in this directory/package
```

Package and import are path hints

The compiler must be able to find all Classes which are referred to from a class it is compiling. If the class uses a second class, the import statement tells the compiler the relative path to where the second class is.

The java runtime must also be able to find classes referred to (used) by a class it is working with. The import statement tells java where to find the second class referred to by the class it is working on.

Running the javac and java commands from the directory above your top level package directory always works.

Example directory tree for packages

```
.
└── org
    └── myproject
        ├── main
        │   ├── Main.class
        │   └── Main.java #imports org.myproject.text.TextUtils
        └── text
            ├── TextUtils.class
            └── TextUtils.java
```

```
javac org/myproject/main/Main.java # needs to find TextUtils
java  org.myproject.main.Main      # needs to find TextUtils
```