

Packages and Interfaces

Packages

Packages are containers for classes.

The package is both a naming and a visibility control mechanism

They are used to keep the class namespace compartmentalized.

Used to avoid collision of class names.

Packages are stored in a hierarchical manner

You can define classes inside a package that are not accessible by code outside that package.

Defining a Package

To create a package, simply include a package command as the first statement in a Java source file.

Any classes declared within that file will belong to the specified package.

The `package` statement defines a namespace in which classes are stored.

If you omit the package statement, the class names are put into the default package, which has no name.

the default package is inadequate for real applications.

This is the general form of the package statement:

```
package pkg;
```

`pkg` is the name of the package

Java uses file system directories to store packages. For example, the .class files for any classes you declare to be part of `MyPackage` must be stored in a directory called `MyPackage`.

directory name must match the package name exactly.

We can create a hierarchy of packages. To do so, simply separate each package name from the one above it by use of a period. `package pkg1[.pkg2[.pkg3]];`

A package hierarchy must be reflected in the file system of Java development system.

Eg. package `java.awt.image;` needs to be stored in `java\awt\image` in a Windows environment

Creating a package

```
1 //create a package mypack
2 package mypack;
3 class MyPackageClass {
4     public static void main(String[] args) {
5         System.out.println("This is my package!");
6     }
7 }
```

Store it in source file `MyPackageClass.java`

Compile the package using javac

Output will be `MyPackageClass.class` and it should be stored in subdirectory `/mypack`

```
C:\java> javac -d . MyPackageClass.java
```

This forces the compiler to create the "mypack" package.

The `-d` keyword specifies the destination for where to save the class file.

The directories corresponding to packages will be created automatically.

You can use any directory name, like `c:/user (windows)`, or,

use the dot sign `"."`, to keep the package within the same directory

Creating a subpackage

```
1 //Create a subpackage inside mypack
2 package mypack.subpack;
3 class MySubPackageClass {
4     public static void main(String[] args) {
5         System.out.println("This is my sub-package!");
6     }
7 }
```

Store it in source file `MySubPackageClass.java`

Compile the package using javac

Output, `MySubPackageClass.class` should be stored in subdirectory `/mypack/subpack`

```
C:\java> javac -d . MySubPackageClass.java
```

PC > Windows (C:) > java > mypack > subpack

Name	Date modified	Type
 MySubPackageClass.class	16-10-2020 11:41	CLASS File

This forces the compiler to create the "mypack.subpack" package.

The `-d` keyword specifies the destination for where to save the class file.

The directories corresponding to packages will be created automatically.

The above statement will create the directories `/mypack/subpack`

Running a Package

```
C:\java>java mypack.MyPackageClass
```

```
This is my package!
```

Run by using 'java' command followed by
`packagename.classname`

```
C:\java>java mypack.subpack.MySubPackageClass
```

```
This is my sub-package!
```

Run by using 'java' command followed by
`packagename.subpkgname.classname`

PATH and CLASSPATH

These are two environment variables present in all operating systems.

You can run the JDK just fine without setting the **PATH** variable

However, you should set the **PATH** variable if you want to be able to run the executables (javac, java, javadoc, and so on) from any directory without having to type the full path of the command.

PATH defines a set of directories where the operating system looks for the executables corresponding to commands you type-in at the prompt.

If you do not set the **PATH** variable, you need to specify the full path to the executable every time you run it.

Finding Packages and CLASSPATH

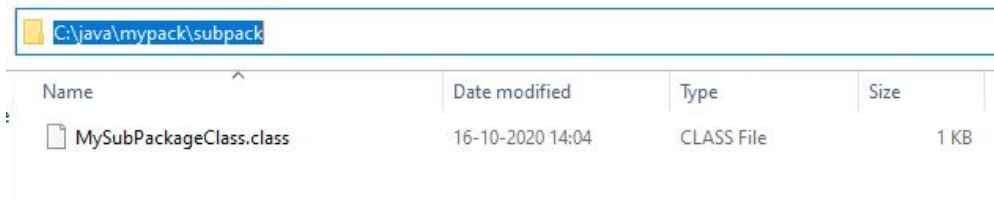
How does the Java run-time system know where to look for packages that we create?

1) by default, the Java run-time system uses the current working directory as its starting point. Thus, if our package is in a subdirectory of the current directory, it will be found.

ie., When we try to execute a class in a package, make sure that the package directory is situated in current directory.

Eg. in order to execute `mypack.subpack.MySubPackageClass`, make sure that `/mypack/subpack` is situated in current directory.

```
C:\java>java mypack.subpack.MySubPackageClass  
This is my sub-package!
```



C:\java\mypack\subpack			
Name	Date modified	Type	Size
MySubPackageClass.class	16-10-2020 14:04	CLASS File	1 KB

Finding Packages and CLASSPATH

How does the Java run-time system know where to look for packages that we create?

2) you can use the `-classpath` option or `-cp` with java and javac to specify the path to your classes.

If you specify classpath like this, it will be possible to run the class from anywhere.

```
C:\>java -classpath c:\java mypack.subpack.MySubPackageClass
This is my sub-package!

C:\>java mypack.subpack.MySubPackageClass
Error: Could not find or load main class mypack.subpack.MySubPackageClass
Caused by: java.lang.ClassNotFoundException: mypack.subpack.MySubPackageClass
```

```
C:\> java -classpath pathtoclass classnamewithpackage
```

Finding Packages and CLASSPATH

How does the Java run-time system know where to look for packages that we create?

3) we can specify a directory path or paths by setting the CLASSPATH environment variable.

In windows, goto System Properties->Advanced Tab->Environment Variables and create a new user variable with name = CLASSPATH and value =<<your classpath>>

Then,you will be able to execute any package class from anywhere.

```
C:\>java mypack.subpack.MySubPackageClass  
This is my sub-package!
```

```
C:\>_
```

