Lecture - Packages

Packages and import statements

Java uses packages to form libraries of classes. A package is a group of classes that have been placed in a directory or folder, and that can be used in any program that includes an import statement that names the package

The import statement must be located at the beginning of the program file: Only blank lines, comments, and package statements may precede it. The program can be in a different directory from the package.

Import statements

We have already used import statements to include some predefined packages in Java, such as `Scanner` from the `java.util` package

```
import java.util.Scanner;
```

It is possible to make all the classes in a package available instead of just one class:

```
import java.util.*;
```

Note that there is no additional overhead for importing the entire package

Drawbacks of using `*`:

- Worse readability of code due to lack of info of which package is used
- Possibly longer compilation time
- Larger possibility of conflict of package names with other packages

The package `java.lang` is special. It is imported automatically, and so no `import` statement is needed. It contains the classes that are fundamental to Java programming. It includes classes such as `Math`, `String`, `System` and the wrapper classes.

Package statements

To make a package, group all the classes together in a single directory (folder), and add the following package statement to the beginning of each java file for those classes:

```
package package_name;
```

Only the `.class` files must be in the directory or folder; the .java files are optional

Only blank lines and comments may go before the package statement.

If there are both import and package statements, the package statement must go before any import statements.

Pakage names and directories

A package name is the path name for the directory or subdirectories that contain the package classes

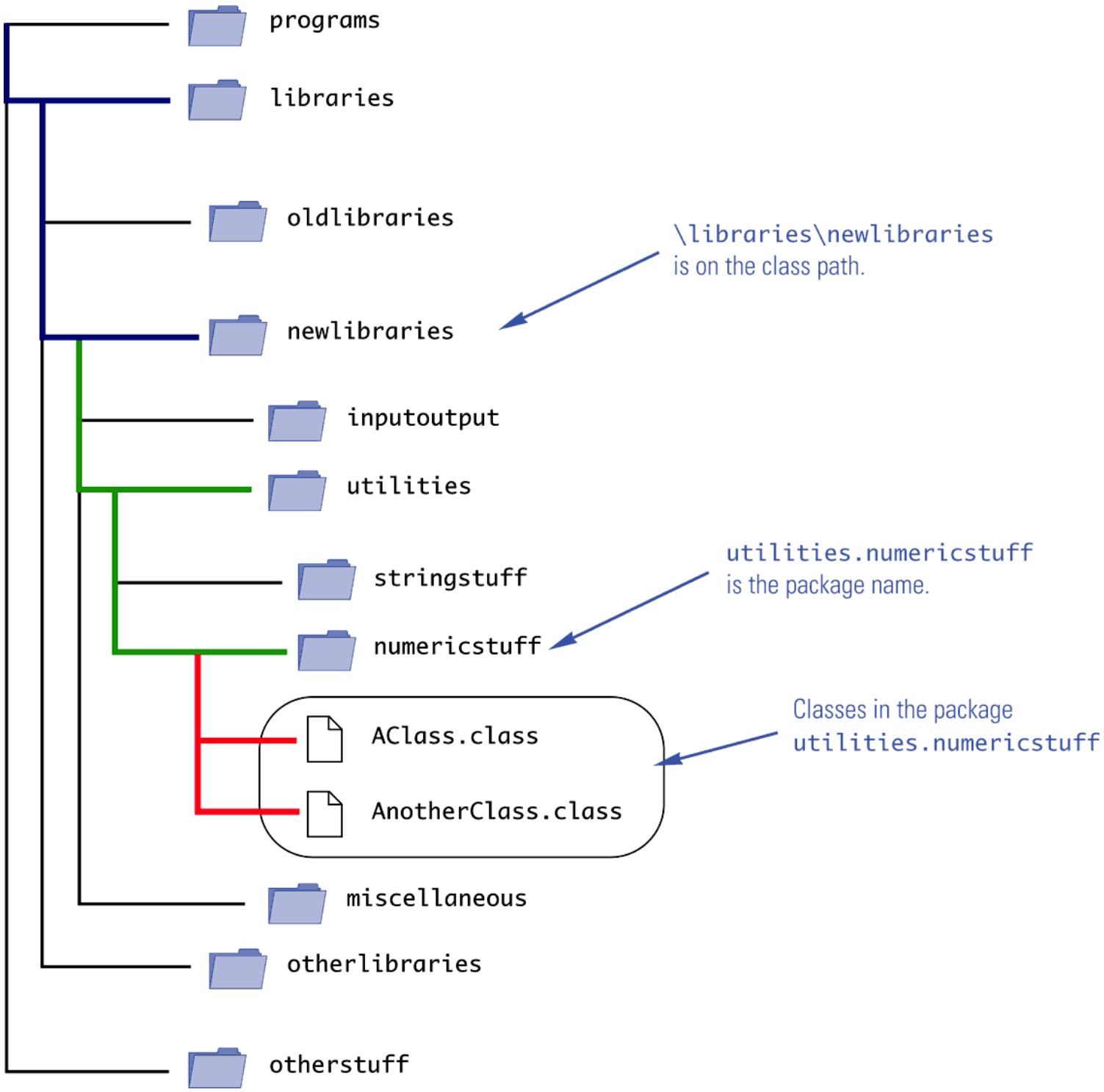
Java needs two things to find the directory for a package:

1. The name of the package and
2. The value of the CLASSPATH variable.

The CLASSPATH environment variable is similar to the PATH variable, and is set in the same way for a given operating system.

The CLASSPATH variable is set equal to the list of directories (including the current directory, ".") in which Java will look for packages on a particular computer.

Java searches this list of directories in order, and uses the first directory on the list in which the package is found.



⚠ Pitfall: Subdirectories are not automatically imported.

When a packages is stored in a subdirectory of the directory containing another package, importing the enclosing package does not import the subdirectory package.

In the above example, `import utilities.numericstuff.*;` imports the `utitilities.numericstuff` package only.
The statements

```
import utilities.numericstuff.*;
import utilities.numericstuff.statistical.*;
```

import both the `utilities.numericstuff` and `utilities.numericstuff.statsistical` packages.

The default package

All classes in the current directory belong to an unnamed package called the *default package*.

As long as the current directory (`.`) is part of the CLASSPATH variable, all the classes in the default package are automatically available to a program.

Pitfall: Not including the current directory in your class path

If the CLASSPATH variable is set, the current directory must be included as one of the alternatives. Otherwise, Java may not even be able to find the .class files for the program itself.

If the CLASSPATH variable is not set, then all the class files for a program must be put in the current directory.

Specifying a class path when you compile

The class path can be manually specified when a class is compiled. Just add `-classpath` followed by the desired class path to the `javac` line. This will compile the class, overriding any previous CLASSPATH setting.

You should use the same `-classpath` option again when the class is run.

Name clashes

In addition to keeping class libraries organized, packages provide a way to deal with name clashes: a situation in which two classes have the same name.

Different programmers writing different packages may use the same name for one or more of their classes. For example, they may have classes with generic names like like `test`, or `search`, or `updater`.

This ambiguity can be resolved by using the fully qualified name (i.e., precede the class name by its package name) to distinguish between each class:

```
    package_name.ClassName;
```

If the fully qualified name is used *everywhere*, it is no longer necessary to import the class (because it includes the package name already).

```
// Note: no  import java.util.Scanner;

class Main {
    static public void main (String[] args) {
        java.util.Scanner keyboard = new java.util.Scanner(System.in);
        System.out.println("Type a line.  I will echo it");
        System.out.println(keyboard.nextLine());
    }
}
```

Example: A community management system

This is a possible layout for some of the classes we have used so far, as part of a system to manage a community.

Click the rectangle at the top left of the code window to see the class hierarchy.