



# Classes 5/5

---



# Outline

---

- Packages and Import **statements**
- The Package **java.lang**
- Package Names and Directories
- The Default Package
- Specifying a Class Path When You Compile



## - 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



## -- The `package` Statement

---

- To make a package, group all the classes together into a single directory (folder), and add the following package statement to the beginning of each class file:

`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 precede the package statement
- If there are both import and package statements, the package statement must precede any import statements



## - The Package `java.lang`

---

- The package `java.lang` contains the classes that are fundamental to Java programming
  - It is imported automatically, so no import statement is needed
  - Classes made available by `java.lang` include `Math`, `String`, and the wrapper classes



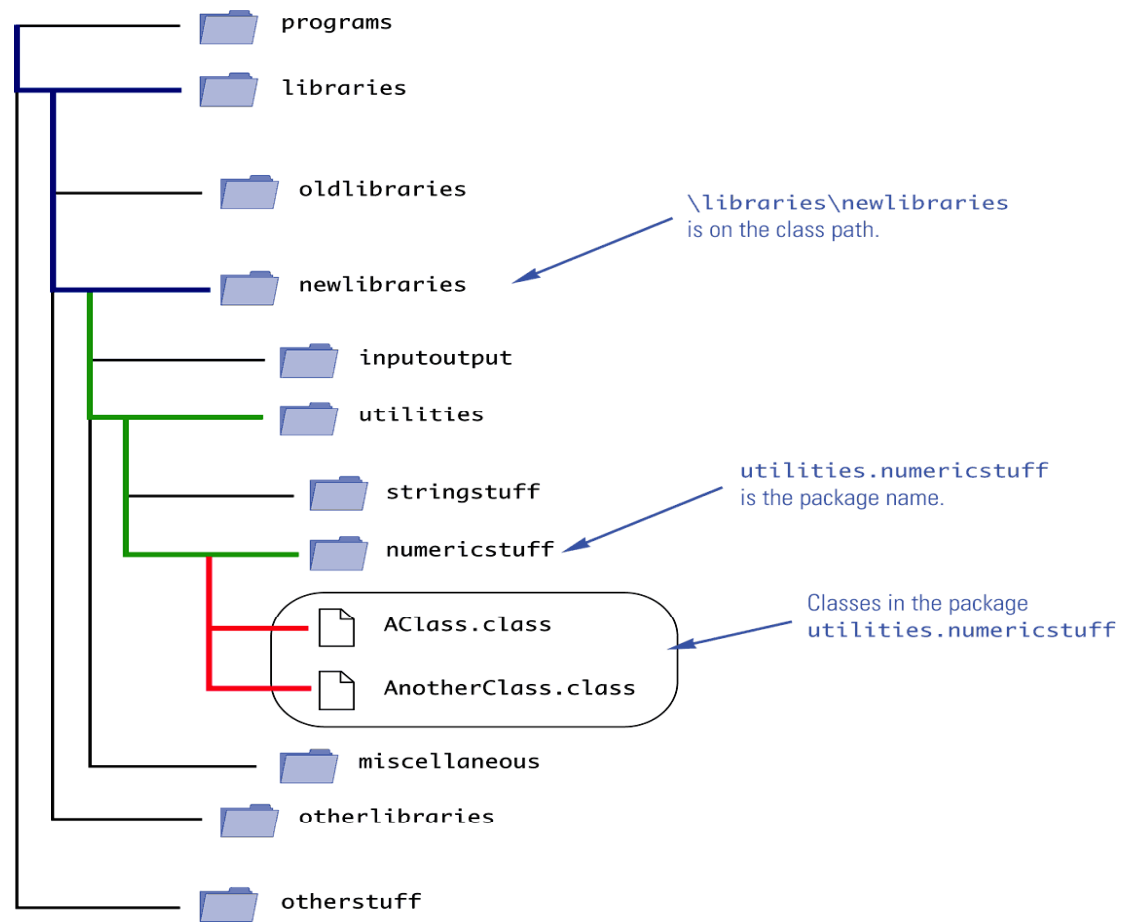
## - Package Names and Directories

---

- 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: the name of the package and 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

# -- A Package Name

Display 5.14 A Package Name





## Pitfall: Subdirectories Are Not Automatically Imported

---

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

- The import statement:

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

imports the `utilities.numericstuff` package only

- The import statements:

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

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



## - The Default Package

---

- All the 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

---

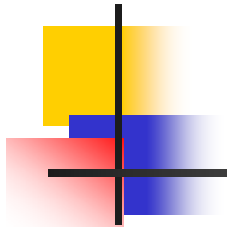
- If the **CLASSPATH** variable is set, the current directory must be included as one of the alternatives
  - Otherwise, Java may not even be able to find the **.class** files for the program itself
- If the **CLASSPATH** variable is not set, then all the class files for a program must be put in the 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
  - This will compile the class, overriding any previous `CLASSPATH` setting
- You should use the `-classpath` option again when the class is run



THE END