**1. What Is a Package?**

A package is a namespace that organizes a set of related classes and interfaces. Conceptually you can think of packages as being similar to different folders on your computer. You might keep HTML pages in one folder, images in another, and scripts or applications in yet another. Because software written in the Java programming language can be composed of hundreds or thousands of individual classes, it makes sense to keep things organized by placing related classes and interfaces into packages.

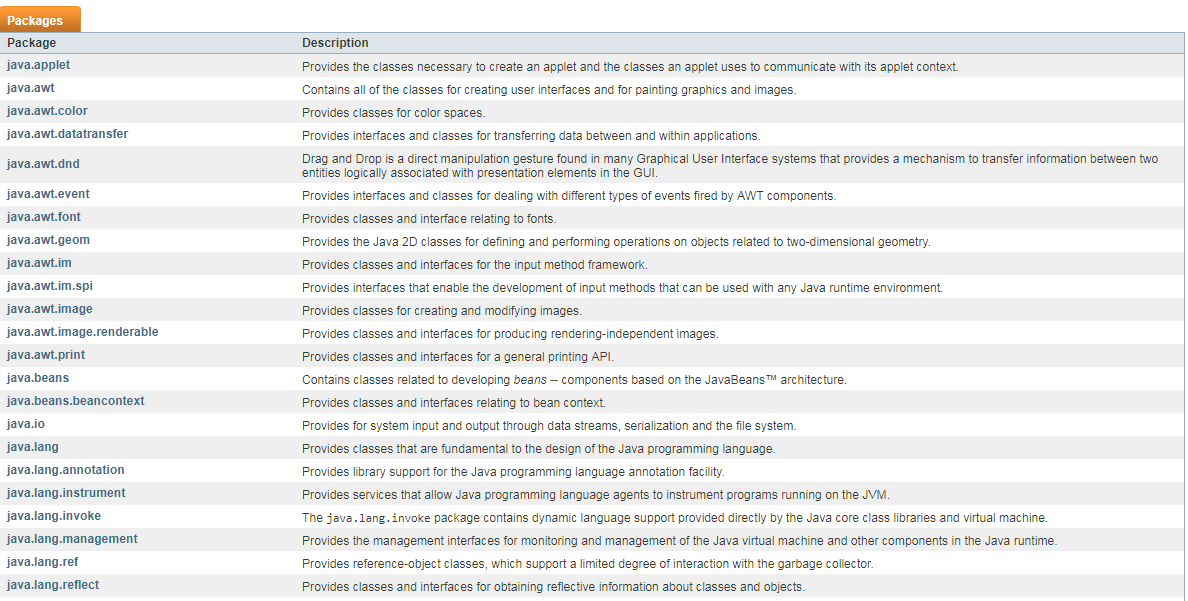
**2. Advantages of using Packages**

In essence, here are the advantages of using packages:

* Packages reduce complexity by facilitating categorization of similar classes.
* Packages provide namespace management. For example, two developers can define the same type name without ending up in a name clash by putting the name in different packages.
* Packages offer access protection (recall the discussion of the default access modifier).

**3. Real-world Examples**

The Java JDK is categorized in various packages:

****

**4. Java Package Naming Conventions**

* A package should be named in lowercase characters. There should be only one English word after each dot.
* The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, like com, edu, gov, mil, net, org. Example:

package org.springframework.core.convert;

package org.hibernate.criterion;

package org.springframework.boot.actuate.audit;

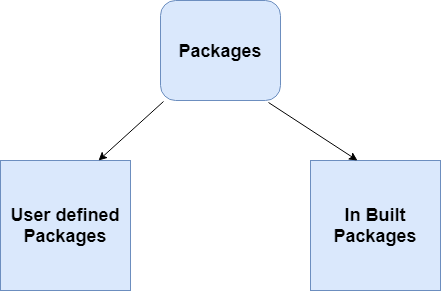
package org.apache.tools.ant.dispatch;

* Package naming convention used by Oracle for the Java core packages. The initial package name representing the domain name must be in lower case.

package java.lang;

package java.util;

**5. Types of packages**

**[](https://2.bp.blogspot.com/-sdaO-KVFHWk/W68GRjJD_OI/AAAAAAAAD_o/VIdzNp25e5gFvLRY3_fWK09meS2BOywpACLcBGAs/s1600/package-types.png)**

**Built-in Packages**

These packages consist of a large number of classes which are a part of Java API. Some of the commonly used built-in packages are:

1. **java.lang**: Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.
2. **java.io**: Contains classed for supporting input/output operations.
3. **java.util**: Contains utility classes which implement data structures like Linked List, Dictionary and support; for Date / Time operations.

**User-defined packages**

Upcoming examples are related user-defined packages.

**6. Defining a Package**

To create a package is quite easy: 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.

This is the general form of the *package* statement:

package pkg;

Here, **pkg** is the name of the package. For example, the following statement creates a package called *MyPackage*:

package MyPackage;

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. The general form of a multileveled *package* statement is shown here:

package pkg1[.pkg2[.pkg3]];

A package hierarchy must be reflected in the file system of your Java development system. For example, a package declared as package *java.awt.image*; needs to be stored in *java\awt\image* in a Windows environment. Be sure to choose your package names carefully. You cannot rename a package without renaming the directory in which the classes are stored.

**7. Finding Packages and CLASSPATH**

As we know that packages are mirrored by directories. This raises an important question: How does the Java run-time system know where to look for packages that you create? The answer has three parts.

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

Second, you can specify a directory path or paths by setting the *CLASSPATH* environmental variable.

Third, you can use the -classpath option with java and javac to specify the path to your classes

**8. Importing Packages**

Java includes the import statement to bring certain classes, or entire packages, into visibility. Once imported, a class can be referred to directly, using only its name. The import statement is a convenience to the programmer and is not technically needed to write a complete Java program. If you are going to refer to a few dozen classes in your application, however, the import statement will save a lot of typing.

In a Java source file, import statements occur immediately following the package statement (if it exists) and before any class definitions. This is the general form of the import statement:

import pkg1 [.pkg2].(classname | \*);

Here, *pkg1* is the name of a top-level package, and *pkg2* is the name of a subordinate package inside the outer package separated by a dot (.).

Example:

import java.util.Date;

import java.io.\*;

* star (\*) - indicates that the Java compiler should import the entire package.

**Static Import**

Java 5 introduced a new feature —**static import** — that can be used to import the static members of the imported package or class. You can use the static members of the imported package or class as if you have defined the static member in the current class.

Example:

import static java.lang.Math.PI;

// class declaration and other members

public double area() {

return PI \* radius \* radius;

}

You can also use wildcard character “\*” to import all static members of a specified package of class.

Always remember that static import only imports static members of the specified package or class.

**9. Working with Packages**

When you want to include a class in a package, you just need to declare it in the class using the *package* statement. Let's create *Circle* class and If you want to include this class in package *com.domain.projectname*, then the following declaration would work:

// Circle.java

package com.domain.projectname;

public class Circle {

//class definition

}

Now, let's say that you want to use this *Circle* class in your *Canvas* class (which is in a different package).

// Canvas.java

package com.domain.projectname.demo;

public class Canvas {

public static void main(String[] args) {

Circle circle = new Circle();

circle.area();

}

}

This code results in the following error message from the compiler: "Circle cannot be resolved to a type". Well, you can remove this error by providing the fully qualified class name, as shown:

// Canvas.java

package com.domain.projectname.demo;

import com.domain.projectname.Circle;

public class Canvas {

public static void main(String[] args) {

Circle circle = new Circle();

circle.area();

}

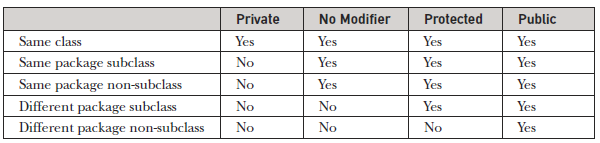
}

**10. Access Protection**

Classes and packages are both means of encapsulating and containing the namespace and scope of variables and methods. Packages act as containers for classes and other subordinate packages. Classes act as containers for data and code. The class is Java’s smallest unit of abstraction. Because of the interplay between classes and packages, Java addresses four categories of visibility for class members:

* Subclasses in the same package
* Non-subclasses in the same package
* Subclasses in different packages
* Classes that are neither in the same package nor subclasses

The three access modifiers, *private*, *public*, and *protected*, provide a variety of ways to produce the many levels of access required by these categories

**[](https://4.bp.blogspot.com/-JcPq6vIfLgQ/W68HyLFvnqI/AAAAAAAAD_4/BCajayzHTPouTTM70i2BhBNxUHJgWX9DgCLcBGAs/s1600/access-protection.PNG)**

The above table applies only to members of classes. A non-nested class has only two possible access levels: *default* and *public*. When a class is declared as public, it is accessible by any other code. If a class has default access, then it can only be accessed by other code within its same package. When a class is *public*, it must be the only public class declared in the file, and the file must have the same name as the class.