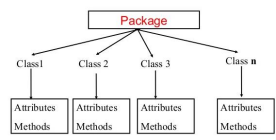
**Core Java/ Topic: Packages**

**Packages**

Packages are used in Java in order to prevent naming conflicts, to control access, to  make searching/locating and usage of classes, interfaces, enumerations and  annotations easier, etc.

A **Package** can be defined as a grouping of related types (classes, interfaces,  enumerations and annotations) providing access protection and namespace  management.



Some of the existing packages in Java are −

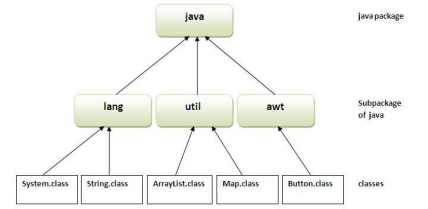
• **java.lang** − bundles the fundamental classes

• **java.util** - The java.util package contains the collections framework, legacy collection  classes, event model, date and time facilities, internationalization, and miscellaneous  utility classes (a string tokenizer, a random-number generator, and a bit array).

• **java.io** − classes for input , output functions are bundled in this package

Page **1** of **10**

**Core Java/ Topic: Packages**

****

**Advantage of Java Package**

1) Java package is used to categorize the classes and interfaces so that they can be  easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

Programmers can define their own packages to bundle group of classes/interfaces, etc.

It is a good practice to group related classes implemented by you so that a programmer  can easily determine that the classes, interfaces, enumerations, and annotations are  related.

Since the package creates a new namespace there won't be any name conflicts with  names in other packages. Using packages, it is easier to provide access control and it is  also easier to locate the related classes.

**Creating a Package**

While creating a package, you should choose a name for the package and include  a **package** statement along with that name at the top of every source file that contains  the classes, interfaces, enumerations, and annotation types that you want to include in  the package.

The package statement should be the first line in the source file. There can be only one  package statement in each source file, and it applies to all types in the file.

Page **2** of **10**

**Core Java/ Topic: Packages**

If a package statement is not used then the class, interfaces, enumerations, and  annotation types will be placed in the current default package.

Simple example of java package

The **package keyword** is used to create a package in java.

1. //save as Simple.java

2. **package** mypack;

3. **public class** Simple{

4. **public static void** main(String args[]){

5. System.out.println("Welcome to package");

6. }

7. }

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

1. javac -d directory javafilename

For **example**

1. javac -d . Simple.java

The -d switch specifies the destination where to put the generated class file. You can use any  directory name like d:/abc (in case of windows) etc. If you want to keep the package within the  same directory, you can use . (dot).

How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

|  |
| --- |
| **To Compile:** javac -d . Simple.java |
| **To Run:** java mypack.Simple |

Output:Welcome to package

|  |
| --- |
| The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder. |

Page **3** of **10**

**Core Java/ Topic: Packages**

**Access Protection**

Java provides many levels of protection to allow fine-grained control over the visibility of  variables and methods within classes, subclasses, and packages.

Classes and packages are both means of encapsulating and containing the name space 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. Table-1 sums up the interactions.

➢ Anything declared **public** can be accessed from anywhere.

Anything declared **private** cannot be seen outside of its class.

➢ When a member does not have an explicit access specification, it is visible to subclasses  as well as to other classes in the same package. This is the **default** access.  ➢ If you want to allow an element to be seen outside your current package, but only to  classes that subclass your class directly, then declare that element **protected**.

**Table-1**

Page **4** of **10**

**Core Java/ Topic: Packages**

**The import Keyword:**

If a class wants to use another class in the same package, the package name need not  be used. Classes in the same package find each other without any special syntax.

**Example**

Here, a class named Boss is added to the payroll package that already contains  Employee. The Boss can then refer to the Employee class without using the payroll  prefix, as demonstrated by the following Boss class.

|  |
| --- |
| package payroll;  public class Boss {   public void payEmployee() {  Employee e=new Employee();   e.mailCheck();   }  } |

What happens if the Employee class is **not** in the payroll package? The Boss class must  then use one of the following techniques for referring to a class in a different package.

• The fully qualified name of the class can be used. For example −

|  |
| --- |
| payroll.Employee |

• The package can be imported using the import keyword and the wild card (\*). For  example −

|  |
| --- |
| import payroll.\*; |

• The class itself can be imported using the import keyword. For example −

|  |
| --- |
| import payroll.Employee; |

**Note** − A class file can contain any number of import statements. The import  statements must appear after the package statement and before the class declaration.

Page **5** of **10**

**Core Java/ Topic: Packages**

**How to access package from another package?**

There are three ways to access the package from outside the package.

1. import package.\*;

2. import package.classname;

3. fully qualified name.

**1) Using packagename.\***

If you use package.\* then all the classes and interfaces of this package will be accessible  but not subpackages.

The import keyword is used to make the classes and interface of another package  accessible to the current package.

//save by A.java

package pack;

public class A{

 public void msg(){System.out.println("Hello");}

}

//save by B.java

package mypack;

import pack.\*;

class B{

 public static void main(String args[]){

 A obj = new A();

 obj.msg();

 }

}

Output:Hello

Page **6** of **10**

**Core Java/ Topic: Packages**

2) Using packagename.classname

If you import package.classname then only declared class of this package will be  accessible.

//save by A.java

package pack;

public class A{

 public void msg(){System.out.println("Hello");}

}

//save by B.java

package mypack;

import pack.A;

class B{

 public static void main(String args[]){

 A obj = new A();

 obj.msg();

 }

}

Output:Hello

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be  accessible. Now there is no need to import. But you need to use fully qualified name  every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql  packages contain Date class.

//save by A.java

package pack;

Page **7** of **10**

**Core Java/ Topic: Packages**

public class A{

 public void msg(){System.out.println("Hello");}

}

//save by B.java

package mypack;

class B{

 public static void main(String args[]){

 pack.A obj = new pack.A();//using fully qualified name

 obj.msg();

 }

}

Output:Hello

**Note: If you import a package, subpackages will not be imported.**

If you import a package, all the classes and interface of that package will be imported  excluding the classes and interfaces of the subpackages. Hence, you need to import the  subpackage as well.

**Note: Sequence of the program must be package then import then class.**

**Predefined packages :**

**java.lang**

**java.lang** is automatically imported into all programs. It contains classes and interfaces that are  fundamental to virtually all of Java programming. It is Java’s most widely used package.  **java.lang** includes the following classes:

Page **8** of **10**

**Core Java/ Topic: Packages**

**java.util:**

This important package contains a large assortment of classes and interfaces that support a  broad range of functionality. For example, **java.util** has classes that generate pseudorandom  numbers, manage date and time, observe events, manipulate sets of bits, tokenize strings, and  handle formatted data. The **java.util** package also contains one of Java’s most powerful  subsystems: the *Collections Framework*. The Collections Framework is a sophisticated hierarchy  of interfaces and classes that provide state-of-the-art technology for managing groups of  objects. It merits close attention by all programmers.

Because **java.util** contains a wide array of functionality, it is quite large. Here is a list of its top level classes:

Page **9** of **10**

**Core Java/ Topic: Packages**

Page **10** of **10**