

Packages

SMITA SANKHE

Assistant Professor

Department of Computer Engineering

Packages: Introduction

- A **Package** can be defined as a grouping of related types (classes, interfaces, enumerations) providing **access protection** and **name space management**.
- A package may consist of a lot of classes but **only few need to be exposed** as most of them are required internally.

Thus, we can **hide the classes** and prevent programs or other packages from accessing classes which are meant for internal use only. Thus it helps to achieve **data encapsulation**.

Packages: Advantages

- **To prevent naming conflicts.**
- **To achieve reusability**
- **Easier to provide access control and provides access protection**
- **Used to categorize the classes and interfaces so that they can be easily maintained.**
- **it is also easier to locate the related classes and to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.**

Packages: Categories

- Packages are categorized as :

1) Built-in packages

standard packages which come as a part of Java Runtime Environment

2) User-defined packages

packages defined by programmers to bundle group of related classes

1. Built-in-Packages

Package Name	Description
java.lang	Contains language support classes (for e.g classes which defines primitive data types, math operations, etc.) . This package is automatically imported.
java.io	Contains classes for supporting input / output operations.
java.util	Contains utility classes which implement data structures like Linked List, Hash Table, Dictionary, etc and support for Date / Time operations.
java.applet	Contains classes for creating Applets.
java.awt	Contains classes for implementing the components of graphical user interface (like buttons, menus, etc.).

1. Built-in-Packages(contd..)

Package Name	Description
<code>java.net</code>	Contains classes for supporting networking operations, sockets, <code>DNS lookups</code> ,
<code>java.math</code>	multiprecision arithmetics
<code>javax.swing</code>	hierarchy of packages for platform-independent rich <code>GUI</code> components
<code>java.security</code>	key generation, encryption and decryption
<code>java.sql</code>	<code>Java Database Connectivity</code> (JDBC) to access databases

2. User defined 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 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.

- If a package statement is not used then the class, interfaces, enumerations will be placed in the current default package.
- **NOTE:** Good practice to use names of packages with lower case letters to avoid any conflicts with the names of classes, interfaces.

How to execute Packages?

- The compiler will complain if the specified directory does not exist, and it won't create one.
- If the source file is under a package, the compiler will create package structure in the destination directory.
- **To Run:**
 - **set classpath=location of directory eg: set classpath=c:\classes;**
 - **java packagename.filename** (*if in current directory 1st step can be excluded*)

Declaration formats

Import is kept after package statement.

1. **import package.*;**

- all the classes and interfaces of this package will be accessible **but not subpackages.**(so separately u have to import subpackages *java.awt.* and java.awt.event.**)
- If we are using this then in the class using the imported package should do object declaration in this format

i.e `packagename.filenameme objectj=new packagename.filenameme()`

Else if we use simple object declaration an error will be thrown.

2. **import package.classname;**

- then only declared class of this package will be accessible.

3. **fully qualified name.**

- Only declared class of this package will be accessible
- But you need to use fully qualified name every time when you are accessing the class or interface from other class.

Declaration formats

- **Static import:**
- The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly.
- There is no need to qualify it by the class name.
- The import allows the java programmer to access classes of a package without package qualification whereas the static import feature allows to access the static members of a class without the class qualification. The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.(eg given in notes)

Example

Demo.java

```
package pack;  
  
public class Demo  
{  
    public void sum(int num1,int num2)  
    {  
        int result;  
        result=num1+num2;  
        System.out.println("the sum of two  
            numbers is:"+result); }  
}
```

Procedure to run the program:

1. **First** compile the Demo.java as follows:

```
javac -d . Demo.java
```

2. **Secondly** compile the Tester.java in another Command Prompt:

```
javac Tester.java
```

3 Run it : **java Tester**

Tester.java

```
import pack.Demo;  
  
class Tester extends Demo  
{  
    public static void main(String a  
        rgs[])  
    {  
        Tester obj=new Tester();  
        obj.sum(10,20);  
    }  
}
```

Fully Qualified Name

- Let us store the code listing below in a file named “ClassA.java” within subdirectory named “myPackage” within the current directory (say “abc”)

```
package myPackage;  
class ClassA {  
    public void display()  
    { System.out.println("Hello, I am ClassA");  
    }  
}  
class ClassB {  
    // class body  
}
```

```
package secondPackage;  
public class ClassC {  
    public void display()  
    { System.out.println("Hello, I am  
    ClassC");  
    }  
}
```

Fully Qualified Name

- Within the current directory (“abc”) store the following code in a file named “ClassY.java”

```
import myPackage.ClassA;  
import secondPackage.ClassC;  
public class ClassY  
{  
    public static void main(String args[])  
    {  
        ClassA objA = new ClassA();  
        ClassC objC = new ClassC();  
        objA.display();  
        objC.display();  
    }  
}
```

Output

Compile and Run:

```
\abc> javac -d . classA.java
```

```
\abc> javac -d . classC.java
```

```
\abc >javac ClassY. Java
```

```
\abc >java ClassY
```

O/P:

```
Hello, I am ClassA
```

```
Hello, I am ClassC
```

- Package inside the package is called the **subpackage**. . It should be created **to categorize the package further**. The packages that comes lower in the naming hierarchy are called "subpackage"
- This allows packages to be easily managed.
- *The standard of defining package is*
domain.company.package
- *e.g.*
com.javatpoint.bean
org.sssit.dao

Example

```
package importpackage.subpackage;  
public class HelloWorld {  
    public void show(){  
        System.out.println("This is the function of the class HelloWorld!!");  
    }  
}
```

```
import importpackage.subpackage.*;  
class CallPackage{  
    public static void main(String[] args){  
        HelloWorld h2=new HelloWorld();  
        h2.show();  
    }  
}
```


IMPORT

- Able to access only class and other class methods with object of that class itself
- With import we can avoid creating object with fully qualified name
- After importing you could use class for your wish(through instantiation, inheritance, etc)
- If we import packages in our program then we get all the classes define in that package.

- **Does importing all classes in a package make my object file (.class or .jar) larger?**

No, import only tells the compiler where to look for symbols.

- **The star form may increase compilation time—especially if you import several large packages.**
- For this reason it is a good idea to explicitly name the classes that you want to use rather than importing whole packages. However, the star form has absolutely no effect on the run-time performance or size of your classes.
- **Is it less efficient to import all classes than only the classes I need?**

EXTENDS

- Able to access all class members like variables, methods etc with the object of extended class
- You can modify the methods and variables and extend it using overriding
- Extending a class is creating a new class that is a subclass of some other class. This will allow you to add or change functionality of the class you are extending.
- import indicate to call the classes and files,not to use them, But extends indicate to use the parent class in the child class.