# Packages in java and how to use them

BY CHAITANYA SINGH | FILED UNDER: OOPS CONCEPT

A package as the name suggests is a pack(group) of classes, interfaces and other packages. In java we use packages to organize our classes and interfaces. We have two **types of packages in Java**: built-in packages and the packages we can create (also known as user defined package). In this guide we will learn what are packages, what are user-defined packages in java and how to use them.

In java we have several built-in packages, for example when we need user input, we import a package like this:

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

Here:
→ **java** is a top level package
→ **util** is a sub package
→ and **Scanner** is a class which is present in the sub package **util**.

Before we see how to create a user-defined package in java, lets see the advantages of using a package.

## Advantages of using a package in Java

These are the reasons why you should use packages in Java:

- **Reusability**: While developing a project in java, we often feel that there are few things that we are writing again and again in our code. Using packages, you can create such things in form of classes inside a package and whenever you need to perform that same task, just import that package and use the class.
- **Better Organization**: Again, in large java projects where we have several hundreds of classes, it is always required to group the similar types of classes in a meaningful package name so that you can organize your project better and when you need something you can quickly locate it and use it, which improves the efficiency.
- **Name Conflicts**: We can define two classes with the same name in different packages so to avoid name collision, we can use packages

## Types of packages in Java

As mentioned in the beginning of this guide that we have two types of packages in java.

1) User defined package: The package we create is called user-defined package.
2) Built-in package: The already defined package like java.io.*, java.lang.* etc are known as built-in packages.

We have already discussed built-in packages, lets discuss user-defined packages with the help of examples.

## Example 1: Java packages

I have created a class `Calculator` inside a package name `letmecalculate`. To create a class inside a package, declare the package name in the first statement in your program. A class can have only one package declaration.
Calculator.java file created inside a package `letmecalculate`

```
package letmecalculate;

public class Calculator {
   public int add(int a, int b){
         return a+b;
   }
   public static void main(String args[]){
         Calculator obj = new Calculator();
         System.out.println(obj.add(10, 20));
   }
}
```

Now lets see how to use this package in another program.

```
import letmecalculate.Calculator;
public class Demo{
   public static void main(String args[]){
         Calculator obj = new Calculator();
         System.out.println(obj.add(100, 200));
   }
}
```

To use the class Calculator, I have imported the package `letmecalculate`. In the above program I have imported the package as `letmecalculate.Calculator`, this only imports the Calculator class. However if you have several classes inside package `letmecalculate` then you can import the package like this, to use all the classes of this package.

```
import letmecalculate.*;
```

## Example 2: Creating a class inside package while importing another package

As we have seen that both package declaration and package import should be the first statement in your java program. Lets see what should be the order

when we are creating a class inside a package while importing another package.

```
//Declaring a package
package anotherpackage;
//importing a package
import letmecalculate.Calculator;
public class Example{
   public static void main(String args[]){
        Calculator obj = new Calculator();
        System.out.println(obj.add(100, 200));
   }
}
```

So the order in this case should be:
→ package declaration
→ package import

## Example 3: Using fully qualified name while importing a class

You can use fully qualified name to avoid the import statement. Lets see an example to understand this:
**Calculator.java**

```
package letmecalculate;
public class Calculator {
   public int add(int a, int b){
        return a+b;
   }
   public static void main(String args[]){
        Calculator obj = new Calculator();
        System.out.println(obj.add(10, 20));
   }
}
```

**Example.java**

```
//Declaring a package
package anotherpackage;
public class Example{
   public static void main(String args[]){
        //Using fully qualified name instead of import
        letmecalculate.Calculator obj =
                new letmecalculate.Calculator();
        System.out.println(obj.add(100, 200));
   }
}
```

In the Example class, instead of importing the package, I have used the full qualified name such as `package_name.class_name` to create the object of it. You may also want to read: static import in Java

# Sub packages in Java

A package inside another package is known as sub package. For example If I create a package inside letmecalculate package then that will be called sub package.

Lets say I have created another package inside `letmecalculate` and the sub package name is `multiply`. So if I create a class in this subpackage it should have this package declaration in the beginning:

```
package letmecalculate.multiply;
```

Multiplication.java

```
package letmecalculate.multiply;
public class Multiplication {
        int product(int a, int b){
                return a*b;
        }
}
```

Now if I need to use this Multiplication class I have to either import the package like this:

```
import letmecalculate.multiply;
```

or I can use fully qualified name like this:

```
letmecalculate.multiply.Multiplication obj =
     new letmecalculate.multiply.Multiplication();
```

# Points to remember:

1. Sometimes class name conflict may occur. For example: Lets say we have two packages **abcpackage** and **xyzpackage** and both the packages have a class with the same name, let it be `JavaExample.java`. Now suppose a class import both these packages like this:

```
import abcpackage.*;
import xyzpackage.*;
```

This will throw compilation error. To avoid such errors you need to use the fully qualified name method that I have shown above. For example

```
abcpackage.JavaExample obj = new abcpackage.JavaExample();
xyzpackage.JavaExample obj2 = new xyzpackage.JavaExample();
```

This way you can avoid the import package statements and avoid that name conflict error.

2. I have already discussed this above, let me mention it again here. If we create a class inside a package while importing another package then the

package declaration should be the first statement, followed by package import. For example:

```
package abcpackage;
import xyzpackage.*;
```

3. A class can have only one package declaration but it can have more than one package import statements. For example:

```
package abcpackage; //This should be one
import xyzpackage;
import anotherpackage;
import anything;
```

4. The wild card import like package.* should be used carefully when working with subpackages. For example: Lets say: we have a package **abc** and inside that package we have another package **foo**, now **foo** is a subpackage.

classes inside abc are: Example1, Example 2, Example 3
classes inside foo are: Demo1, Demo2

So if I import the package **abc** using wildcard like this:

```
import abc.*;
```

Then it will only import classes Example1, Example2 and Example3 but it will not import the classes of sub package.

To import the classes of subpackage you need to import like this:

```
import abc.foo.*;
```

This will import Demo1 and Demo2 but it will not import the Example1, Example2 and Example3.

So to import all the classes present in package and subpackage, we need to use two import statements like this:

```
import abc.*;
import abc.foo.*;
```