Packages

Packages: Creating Packages, using Packages, Access protection.

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

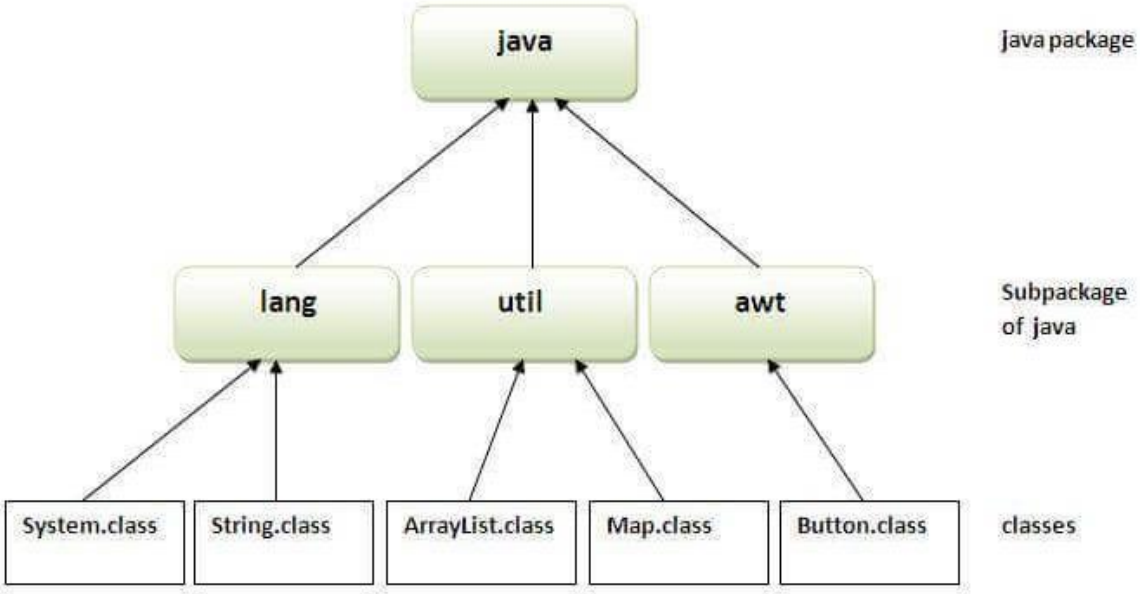
```
import java.util.Scanner
```

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 we have two types of packages in java.

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

1. Java API packages or built-in packages

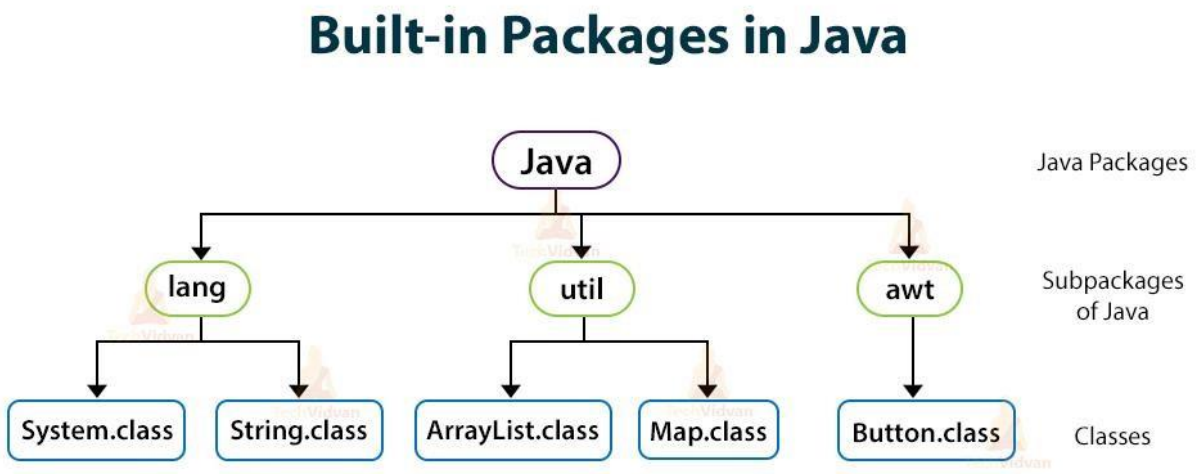
Java provides a large number of classes grouped into different packages based on a particular functionality.

Examples:

java.lang: It contains classes for primitive types, strings, math functions, threads, and exceptions. **java.util:** It contains classes such as vectors, hash tables, dates, Calendars, etc. **java.io:** It has stream classes for Input/Output.

java.awt: Classes for implementing Graphical User Interface – windows, buttons, menus, etc.

java.net: Classes for networking **java. Applet:** Classes for creating and implementing applets



2) User defined package: we have to create new packages.

Simple example of java package

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

```
//save as Simple.java
package mypack;   public
class Simple{
 public static void main(String args[]){
   System.out.println("Welcome to package");
  }
}
```

How to compile java package

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

```
javac –d  directory  javafilename
```

For **example**

```
javac -d . Simple.java
```

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), 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 d represents the current folder.

Example : 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.*;
```

first we have to execute first program. It is having user defined package letmecalculate. It can be execute with folloing commands.
To compile: javac –d . Calculator.java
To run: java letmecalculate. Calculator

Now we have to execute second program. It imported the letmecalculate pakage and it doesn't have its own package. It can be executed by following commands.
To compile: javac Demo.java
To run: java Demo

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.

Example of package that import the packagename.*

```
//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

2) Using packagename.classname

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

Example of package by import package.classname

```
//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.

Example of package by import fully qualified name

```
//save by A.java   package pack;   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

Access Protection in Packages

Access modifiers define the scope of the class and its members (data and methods). For example, private members are accessible within the same class members (methods). Java provides many levels of security that provides the visibility of members (variables and methods) within the classes, subclasses, and packages.

Packages are meant for encapsulating, it works as containers for classes and other subpackages. Class acts as containers for data and methods. There are four categories, provided by Java regarding the visibility of the class members between classes and packages:

1. Subclasses in the same package
2. Non-subclasses in the same package
3. Subclasses in different packages
4. Classes that are neither in the same package nor subclasses

The three main access modifiers *private, public* and *protected* provides a range of ways to access required by these categories.

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Extra topics and examples:

Example : 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.

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.*;
```