

# Java Package

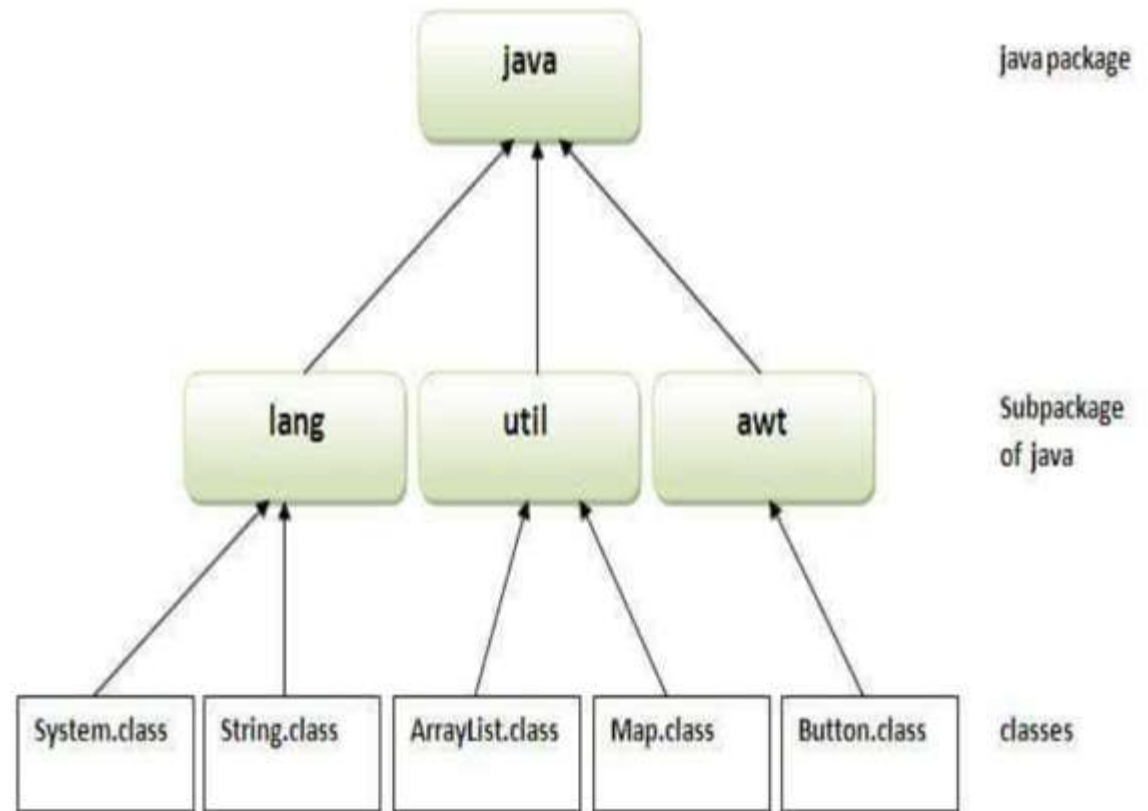
A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

## 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.





# Definig a Package in java

We use the `package` keyword to create or define a package in java programming language.

## Syntax

```
package packageName;
```

 The package statement must be the first statement in the program.

 The package name must be a single word.

## Example

```
package myPackage;

public class DefiningPackage {

    public static void main(String[] args) {

        System.out.println("This class belongs to myPackage.");

    }


}
```

Now, save the above code in a file **DefiningPackage.java**, and compile it using the following command.

```
javac -d . DefiningPackage.java
```

Run the program use the following command.

```
java myPackage.DefiningPackage
```

 When we use IDE like Eclipse, Netbeans, etc. the package structure is created automatically.

# Access protection

In java, the access modifiers define the accessibility of the class and its members. For example, private members are accessible within the same class members only. Java has four access modifiers, and they are default, private, protected, and public.

In java, the package is a container of classes, sub-classes, interfaces, and sub-packages. The class acts as a container of data and methods. So, the access modifier decides the accessibility of class members across the different packages.

Access control for members of class and interface in java

Access Specifier \ Accessibility Location	Same Class	Same Package		Other Package	
		Child class	Non-child class	Child class	Non-child class
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	Yes	No
Default	Yes	Yes	Yes	No	No
Private	Yes	No	No	No	No

# Importing Packages

In java, the `import` keyword used to import built-in and user-defined packages. When a package has imported, we can refer to all the classes of that package using their name directly.

The import statement must be after the package statement, and before any other statement.

## Importing specific class

Using an importing statement, we can import a specific class. The following syntax is employed to import a specific class.

### Syntax

```
import packageName.ClassName;
```

## Importing all the classes

Using an importing statement, we can import all the classes of a package. To import all the classes of the package, we use \* symbol. The following syntax is employed to import all the classes of a package.

### Syntax

```
import packageName.*;
```



# Defining an interface

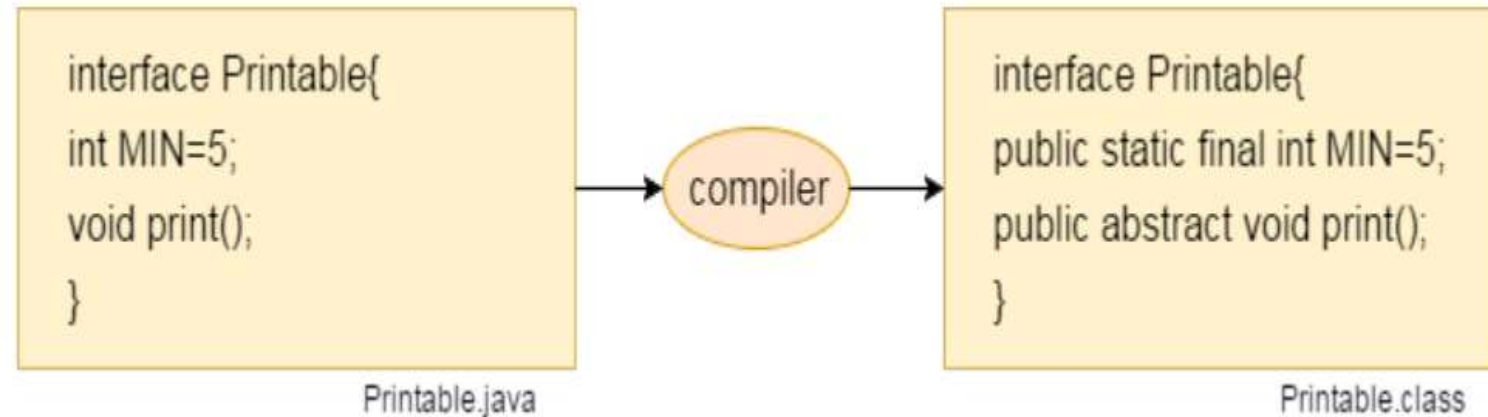
In java, an **interface** is similar to a class, but it contains abstract methods and static final variables only. The interface in Java is another mechanism to achieve abstraction.

## Defining an interface in java

### Syntax:

```
interface <interface_name>{  
  
    // declare constant fields  
    // declare methods that abstract  
    // by default.  
}
```

Interface fields are public, static and final by default, and the methods are public and abstract.



# Implementing an Interface in java

In java, an **interface** is implemented by a class. The class that implements an interface must provide code for all the methods defined in the interface, otherwise, it must be defined as an abstract class.

The class uses a keyword **implements** to implement an interface. A class can implement any number of interfaces. When a class wants to implement more than one interface, we use the **implements** keyword is followed by a comma-separated list of the interfaces implemented by the class.

## Syntax

```
class className implements InterfaceName[
    ...
    body-of-the-class
    ...
]
```

Let's look at an example code to define a class that implements an interface.

```
interface Human {  
  
    void learn(String str);  
    void work();  
  
    int duration = 10;  
  
}
```

```
class Programmer implements Human{  
    public void learn(String str) {  
        System.out.println("Learn using " + str);  
    }  
    public void work() {  
        System.out.println("Develop applications");  
    }  
}
```

```
public class HumanTest {  
  
    public static void main(String[] args) {  
        Programmer trainee = new Programmer();  
        trainee.learn("coding");  
        trainee.work();  
    }  
}
```

In the above code defines an interface **Human** that contains two abstract methods `learn()`, `work()` and one constant duration. The class `Programmer` implements the interface. As it implementing the `Human` interface it must provide the body of all the methods those defined in the `Human` interface.



# Nested Interfaces in java

In java, an interface may be defined inside another interface, and also inside a class. The interface that defined inside another interface or a class is known as nested interface. The nested interface is also referred as inner interface.

- 🔔 The nested interface declared within an interface is public by default.
- 🔔 The nested interface declared within a class can be with any access modifier.
- 🔔 Every nested interface is static by default.

The nested interface cannot be accessed directly. We can only access the nested interface by using outer interface or outer class name followed by dot( . ), followed by the nested interface name.

# Nested interface inside another interface

The nested interface that defined inside another interface must be accessed as

**OuterInterface.InnerInterface**.

```
interface OuterInterface{
    void outerMethod();

    interface InnerInterface{
        void innerMethod();
    }
}

class OnlyOuter implements OuterInterface{
    public void outerMethod() {
        System.out.println("This is OuterInterface method");
    }
}
```

```
class OnlyInner implements OuterInterface.InnerInterface{
    public void innerMethod() {
        System.out.println("This is InnerInterface method");
    }
}

public class NestedInterfaceExample {

    public static void main(String[] args) {
        OnlyOuter obj_1 = new OnlyOuter();
        OnlyInner obj_2 = new OnlyInner();

        obj_1.outerMethod();
        obj_2.innerMethod();
    }
}
```

# Extending an Interface in java

In java, an interface can extend another interface. When an interface wants to extend another interface, it uses the keyword `extends`. The interface that extends another interface has its own members and all the members defined in its parent interface too. The class which implements a child interface needs to provide code for the methods defined in both child and parent interfaces, otherwise, it needs to be defined as abstract class.

- 🔔 An interface can extend another interface.
- 🔔 An interface can not extend multiple interfaces.
- 🔔 An interface can implement neither an interface nor a class.
- 🔔 The class that implements child interface needs to provide code for all the methods defined in both child and parent interfaces.

Let's look at an example code to illustrate extending an interface.

## Example

```
interface ParentInterface{
    void parentMethod();
}

interface ChildInterface extends ParentInterface{
    void childMethod();
}

class ImplementingClass implements ChildInterface{

    public void childMethod() {
        System.out.println("Child Interface method!!");
    }

    public void parentMethod() {
        System.out.println("Parent Interface mehtod!");
    }
}
```

```
public class ExtendingAnInterface {

    public static void main(String[] args) {

        ImplementingClass obj = new ImplementingClass();

        obj.childMethod();
        obj.parentMethod();

    }
}
```

When we run the above program,  
it produce the following output.

```
Child Interface method!!
Parent Interface mehtod!
```