Introduction to Java

By Sanjana Bandara

Content

- Object Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation
- OOPs Misc

Abstract

01. Abstract class

Abstract class in Java

- A class which is declared with the abstract keyword is known as an abstract class in Java.

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (100%)

Abstract class in Java

- A class which is declared as abstract is known as an **abstract class**.
- It can have abstract and non-abstract methods.
- It needs to be extended and its method implemented.
- It cannot be instantiated.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

- abstract class A{}

Abstract Method in Java

- A method which is declared as abstract and does not have implementation is known as an abstract method.
- Rule: If there is an abstract method in a class, that class must be abstract.
- abstract void display(); //no method body and abstract

Example of Abstract class that has an abstract method

```
abstract class Bike{
  abstract void run();  }
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
} }
```

02. Interface in Java

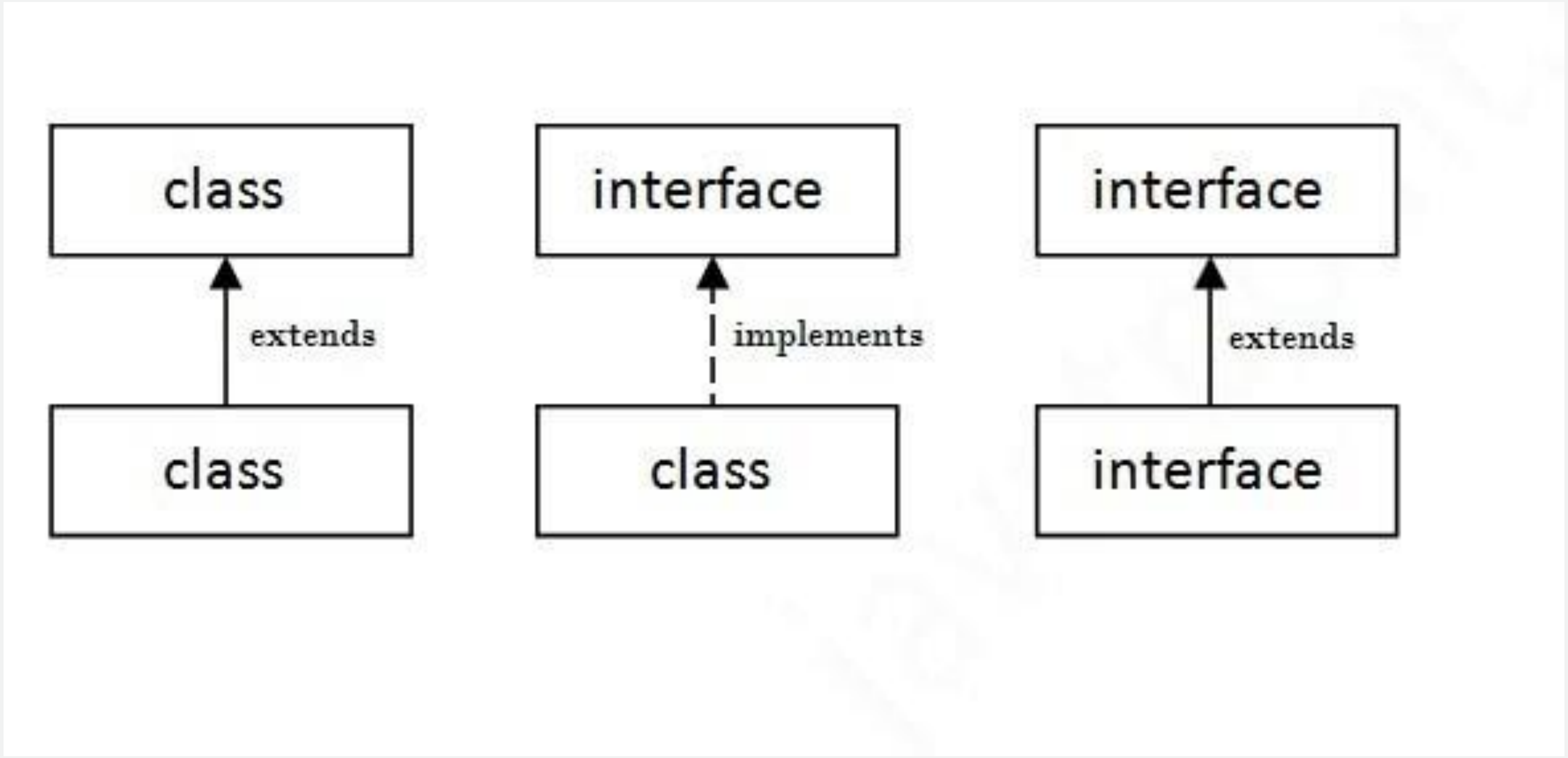
Interface in Java

- An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.
- The interface in Java is *a mechanism to achieve abstraction*.
- Interfaces can have abstract methods and variables. It cannot have a method body.
- **Represents the IS-A relationship**.
- interface <interface_name>{}

Why use Java interface?

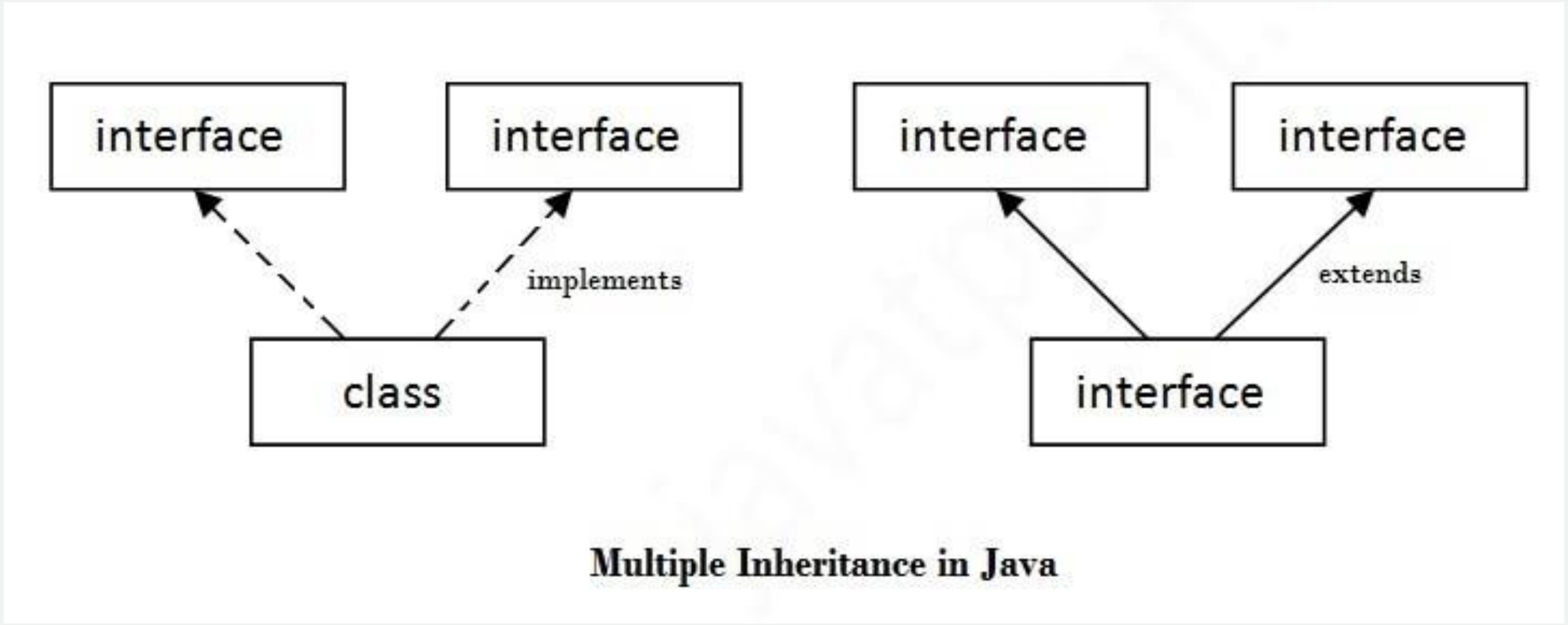
- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Relationship between classes and interfaces



Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Nested Interface in Java

Note: An interface can have another interface which is known as a nested interface.

```
interface printable{
 void print();
 interface MessagePrintable{
   void msg();
 }
}
```

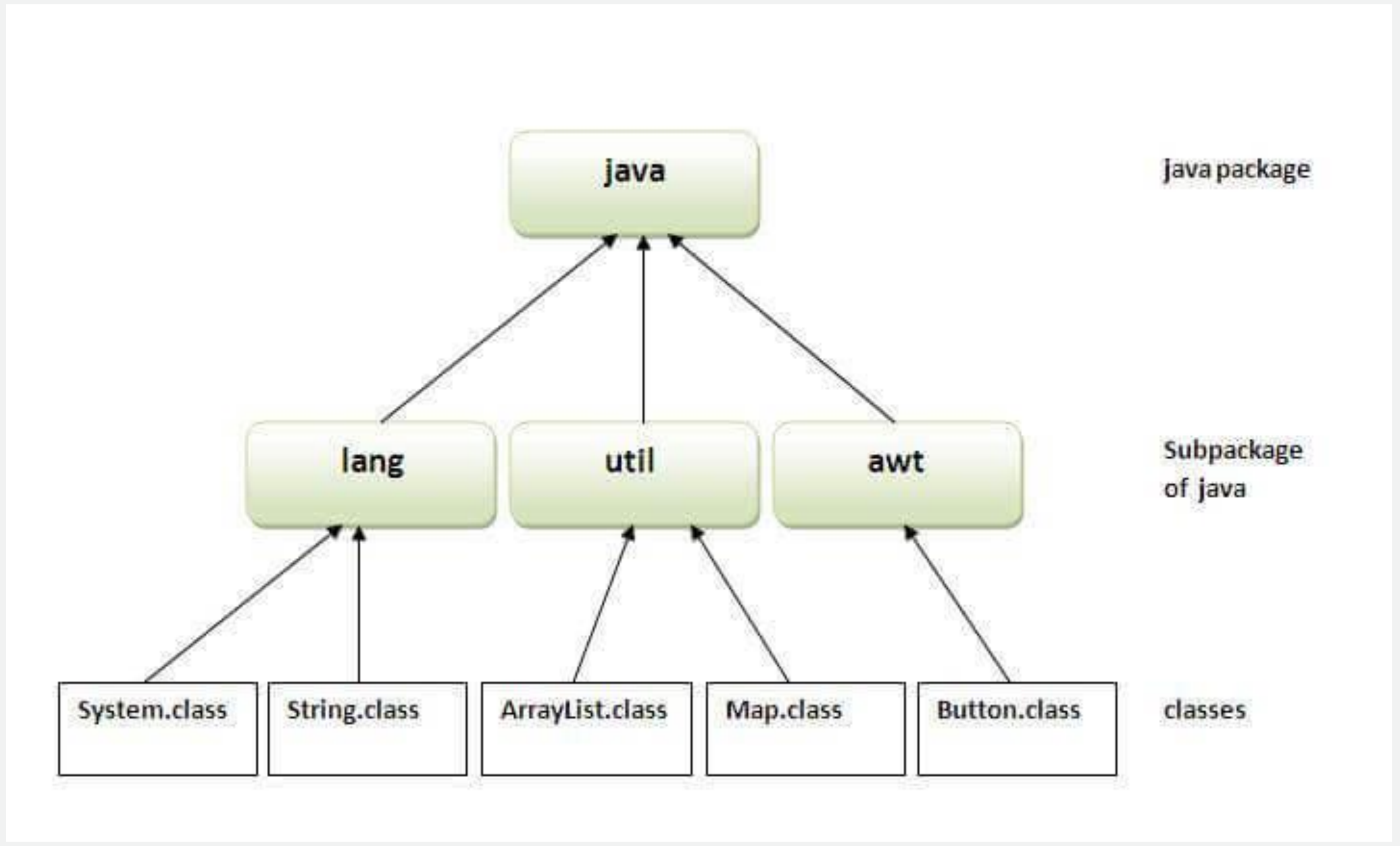
Encapsulation

01. Package

Java Packaae

- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- There are two types - built-in package and user-defined package.
- 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.



Packages in Java

How to compile java package - If you are not using any IDE, you need to follow the **syntax** given below:

```
javac -d . java_file_name.extension
```

How to run java package program

```
java package_name.java_file_name
```

```
//save as Simple.java

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

How to access package from another package?

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

- import package*;
- import package.classname;
- fully qualified name

1) Using 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();  } }
```

2) Using packagename.classname

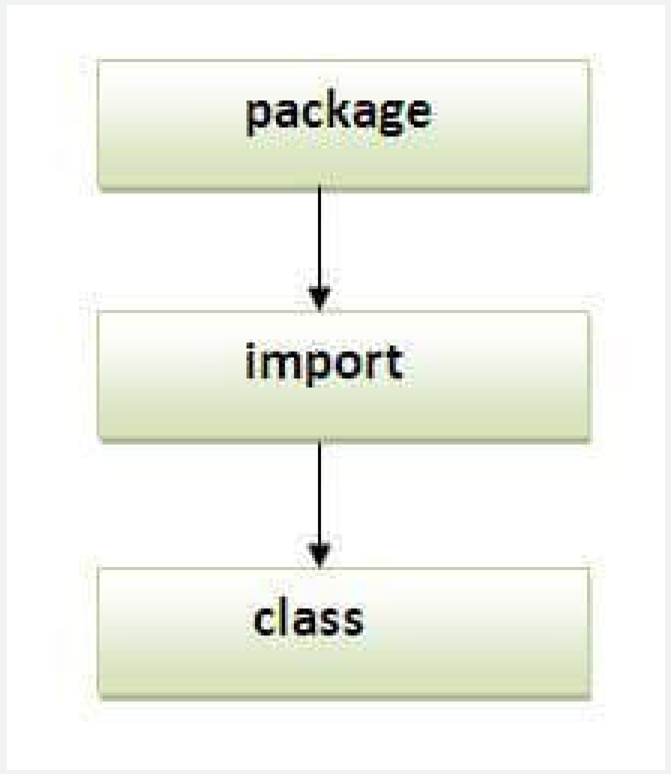
If you import package.classname then only declared class of this package will be accessible.

```
//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();
  }
}
```

3) Using fully qualified name

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.
Note: If you import a package, subpackages will not be imported.



```
//save by A.java

package pack;
public class A{
  public void msg(){System.out.println("Hello")
;}
}
```

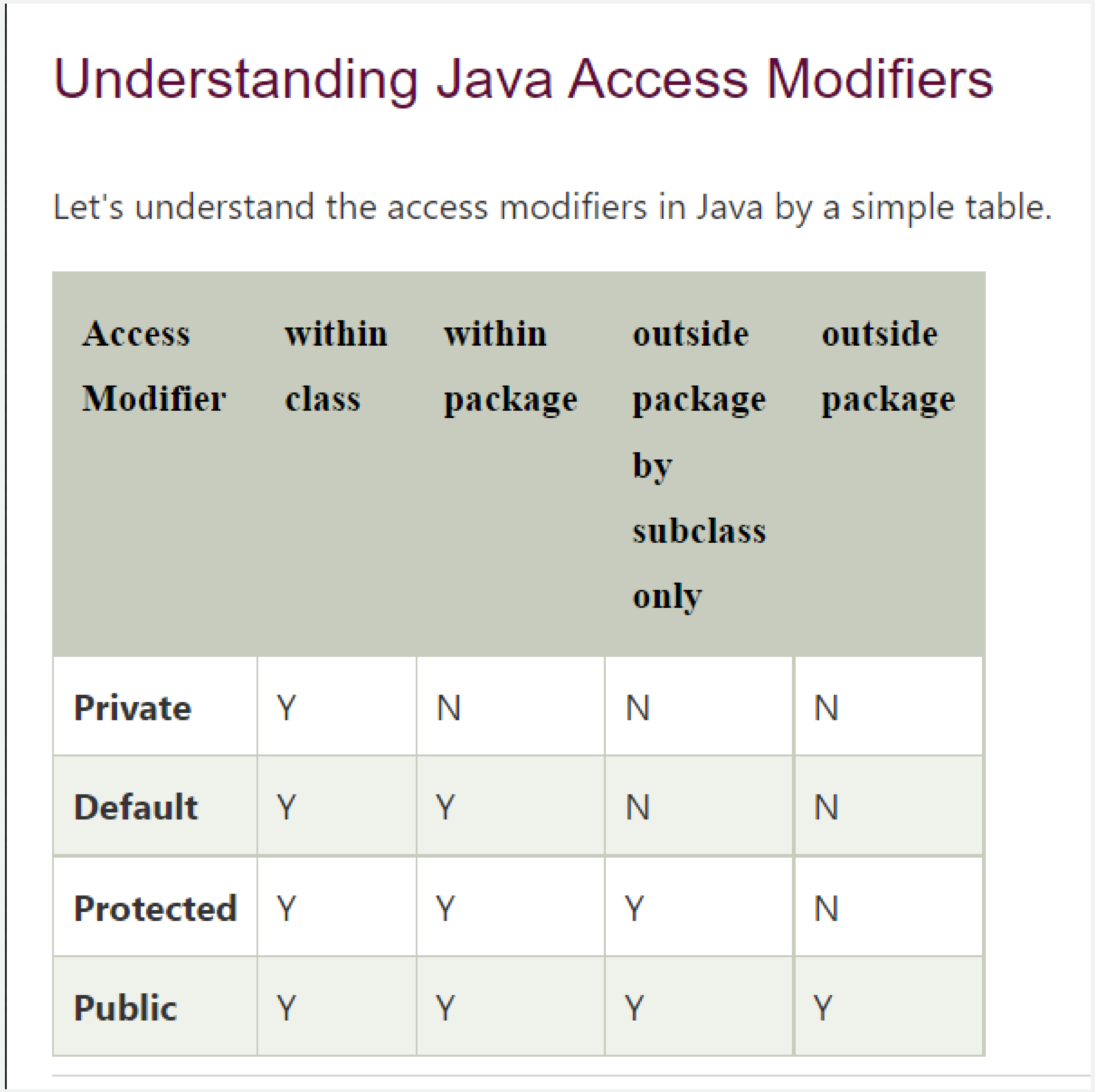
```
//save by B.java

package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully quali
fied name
   obj.msg();
  }
}
```

02. Access Modifiers

Access Modifiers in Java

- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.
- We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

    - **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
    - **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
    - **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
    - **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.



Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

01) Private

- The private access modifier is accessible only within the class.

Role of Private Constructor
If you make any class constructor private, you cannot create the instance of that class from outside the class.

```
class A{
private A(){}//private constructor
void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
}
```

02) Default

- If you don't use any modifier, it is treated as **default** by default.
- The default modifier is accessible only within package.
- It cannot be accessed from outside the package.
- It provides more accessibility than private. But, it is more restrictive than protected, and public.

```
//save by A.java

package pack;
class A{
  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();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

03) Protected

- The **protected access modifier** is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.
- It provides more accessibility than the default modifer.

```
//save by A.java

package pack;
public class A{
protected void msg(){System.out.println("Hell
o");}
}
```

```
//save by B.java

package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

04) Public

- The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

```
//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();
  }
}
```

Java Access Modifiers with Method Overriding

- The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

```
class A{
protected void msg(){System.out.println("Hello java");}
}

public class Simple extends A{
void msg(){System.out.println("Hello java");}//C.T.Error
 public static void main(String args[]){
   Simple obj=new Simple();
   obj.msg();
   }
}
```

03. Encapsulation

Encapsulation in Java

Encapsulation in Java is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The **Java Bean** class is the example of a fully encapsulated class.

Advantages of Encapsulation

- can make the class **read-only or write-only**.
- Control over class
- control over the data
- data hiding
- easy to text