Inheritance

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system). The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also. Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Advantages of Inheritance:

- Code reusability - public methods of base class can be reused in derived classes
- Data hiding – private data of base class cannot be altered by derived class
- Overriding--With inheritance, we will be able to override the methods of the base class in the derived class

Terms used in Inheritance

Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

Sub Class/Child Class/Derive Class : Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Super Class/Parent Class/Base Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

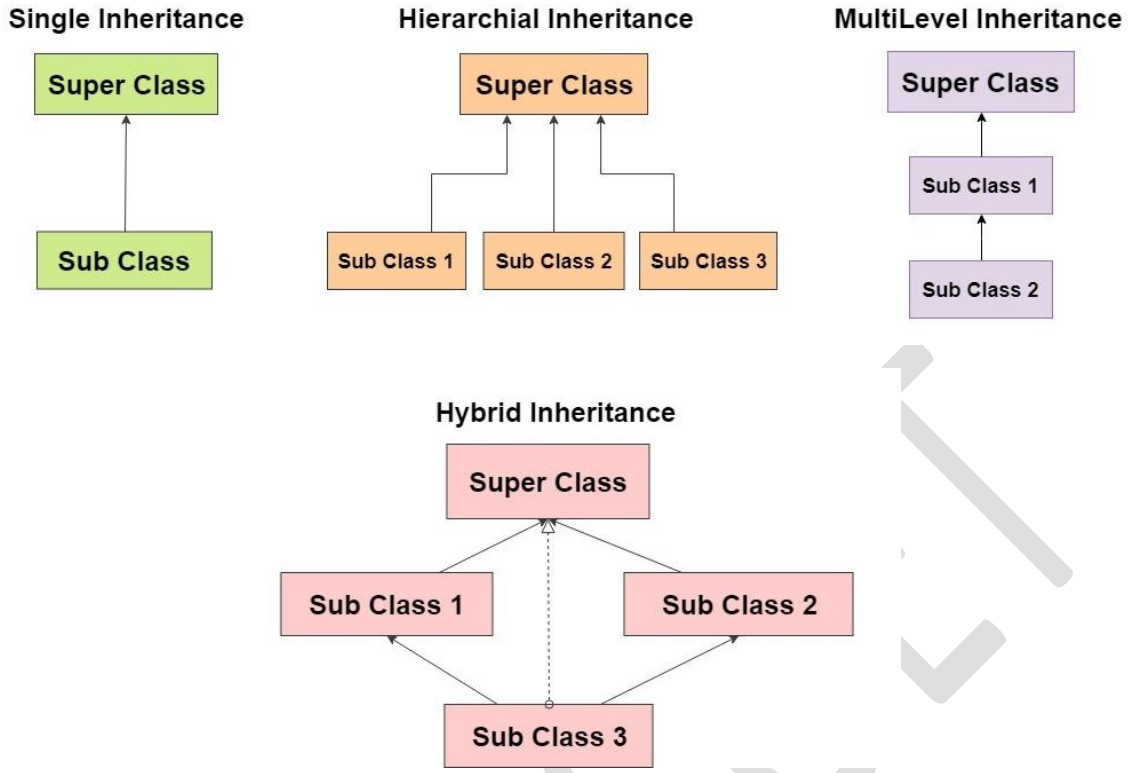
Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
//methods and fields
 }
```

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality. In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Types of Inheritance in java

1. Single-level inheritance
2. Multi-level Inheritance
3. Hierarchical Inheritance
4. Hybrid Inheritance



Single Inheritance

When a class inherits another class, it is known as a single inheritance. In the example given below, Programmer class inherits the Employee class, so there is the single inheritance.

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee

{
 int bonus=10000;}
class Test{
 public static void main(String args[]){
   Programmer p=new Programmer();  //object creation for sub class
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}  } }
```

Output:

```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

Multilevel Inheritance

In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class i.e. a derived class in turn acts as a base class for another class. There is a chain of inheritance.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:
```
weeping...
barking...
eating...
```

Hierarchical Inheritance

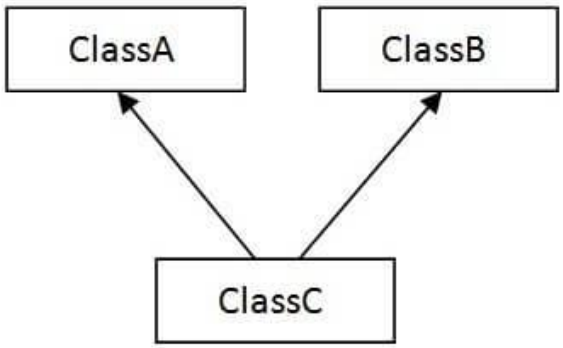
When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.one class serves as a superclass (base class) for more than one sub class.

```
class Animal{
void eat(){System.out.println("eating...");}  }

class Dog extends Animal{
void bark(){System.out.println("barking...");}  }

class Cat extends Animal{
void meow(){System.out.println("meowing...");}  }

class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
}}
```

Output:
```
meowing...
eating...
```

Why multiple inheritance is not supported in java?

- To reduce the complexity and simplify the language, multiple inheritance is not supported in java.
- Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.
- Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.




```
class A{
void msg()
{System.out.println("Hello");}
}
class B{
void msg()
{System.out.println("Welcome");}
}
class C extends A,B{
 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}  }
```

Output:
Compile Time Error

Method overriding:
Child class has the same method as of base class. In such cases child class overrides the parent class method without even touching the source code of the base class. This feature is known as method overriding.

```
public class Base{
public void get () //Base class method
{
System.out.println ("I'm the method of BaseClass");
}}
public class Derived extends Base{
public void get () //Derived Class method
```

```
{
System.out.println ("I'm the method of DerivedClass");
}}
public class Test
{
public static void main (String args []) {
// Base Class reference and object
Base obj1 = new Base ();
// Base Class reference but Derived Class object
Base obj2 = new Derived ();
// Calls the method from Base class
 obj1.get();
//Calls the method from Derived class
\
 obj2.get();
}}
```

Output:

I'm the method of BaseClass
I'm the method of DerivedClass

Rules for Method Overriding:

- Applies only to inherited methods object type (NOT reference variable type) determines which overridden method will be used at runtime
- Overriding methods must have the same return type
- Overriding method must not have more restrictive access modifier
- Abstract methods must be overridden
- Static and final methods cannot be overridden
- Constructors cannot be overridden
- It is also known as Runtime polymorphism.

Important Question for placement

- ***Can we override static method?***
  No, a static method cannot be overridden. It can be proved by runtime polymorphism.

- ***Why can we not override static method?***
  It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

- ***Can we override java main method?***
  No, because the main is a static

Super Keyword in Java

The super keyword in Java is a reference variable which is used to parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

•It can be used to refer immediate parent class instance variable when both parent and child class have member with same name
•It can be used to invoke immediate parent class method when child class has overridden that method.
•super() can be used to invoke immediate parent class constructor.

Use of super with variables:
When both parent and child class have member with same name, we can use super keyword to access member of parent class.

```
import java.io.*;
class Super_Variable
 {
        int b=30;
 }
class Sub_Class extends Super_Variable
{
        int b=12;
        void show()
        {
System.out.println("subclass class variable: "+ b); //print b of sub class
System.out.println("superclass instance variable: "+super.b); //print b of super class
        }
        public static void main (String args[])
    {
                Sub_Class s=new Sub_Class();
                s.show();
        }}
```

Output:
subclass class variable: 12
superclass instance variable: 30

Use of super with methods
The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class (Method Overriding).

```
import java.io.*;
class Anew {
void show(){            //show() in super class
        System.out.println("Super Class show method");
        }}
class Bnew extends Anew {
        void show(){            //show() in super class
                super.show();  // invoke super class display()
                System.out.println("Sub Class show method");
        } }
```

```
class Main {
        public static void main (String args[ ])
    {
                BNew s2=new BNew ();
                s2.show();
        }}
```

Output:

Super Class show method
Sub Class show method

Use of super with constructors:

The super keyword can also be used to invoke the parent class constructor.

Syntax:

```
super();
```

•super() if present, must always be the first statement executed inside a subclass constructor.

•When we invoke a super() statement from within a subclass constructor, we are invoking the immediate super class constructor.

```
import java.io.*;
class A {
        A(){
                System.out.println("Constructor A Revised");
        }}
class B extends A{
        B(){
                System.out.println("Constructor B");}
        B(int a){
                a++;
                System.out.println("Constructor B Revised "+ a );
        }}
class C extends B {
        C()
        { super(11);
                System.out.println("Constructor C Revised");}
        public static void main (String args[]) {
                C a=new C();
        }}
```

Output:

Constructor B Revised 12
Constructor C Revised

Final keyword

Final keyword in java is used to restrict the user. The java final keyword can be used in many context.

Final can be:

1. variable
2. method

3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's firstlearn the basics of final keyword.

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable

There is a final variable speed limit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike9{
final int speedlimit=90;
void run(){
speedlimit=400;}
public static void main(String args[]){
Bike9 obj=new Bike9();8.
obj.run();}
}
```

Output:

Compile Time Error

2) Java final method

If you make any method as final, you cannot override it.

Example of final method

```
class Bike{
final void run(){System.out.println("running");
}}
class Honda extends Bike{
void run(){System.out.println("running safely with 100kmph");}
public static void main(String args[]){
Honda honda= new Honda();
honda.run();
}}
```

Output:

Compile Time Error

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
final class Bike{}
class Honda1 extends Bike{
void run(){
System.out.println("running safely with 100kmph");}
public static void main(String args[]){
Honda1 honda= new Honda();6. honda.run();
```

} }

Can we declare a constructor final?
No, because constructor is never inherited.

Abstract Classes And Methods
Abstract class
A class that is declared as abstract is known as abstract class. It can have abstract and non-abstract methods (method with body). It needs to be extended and its method implemented. It cannot be instantiated(object can not be created)

Syntax:
abstract class_name
{
}
Abstract method

- A method that is declared as abstract and does not have implementation is known as abstract method. The method body will be defined by its subclass.
- Abstract method can never be final and static. Any class that extends an abstract class must implement all the abstract methods declared by the super class.

Note: *A normal class (non-abstract class) cannot have abstract methods.*
Syntax:
abstract return_type functionname (); //No definition

Syntax for abstract class and method:
modifier abstract class_Name
{
//declare fields
//declare methods
abstract dataType methodName();
}
modifier class childClass extends abstract_className
{
dataType methodName()
{
}
}

- **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.

- **Example**
  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();
}
}
```

Output: running safely

Interface in Java

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Uses of interface:

•Since java does not support multiple inheritance in case of class, it can be achieved by using interface.

•It is also used to achieve loose coupling.

•Interfaces are used to implement abstraction.

Defining an Interface

An interface is defined much like a class.

Syntax:

```
accessspecifier interface interfacename
{
return-type method-name1(parameter-list);
return-type method-name2(parameter-list);
type final-varname1 = value;
type final-varname2 = value;
// ...
return-type method-nameN(parameter-list);
type final-varnameN = value;
}
```

When no access specifier is included, then default access results, and the interface is only available to other members of the package in which it is declared. When it is declared as public, the interface can be used by any other code.

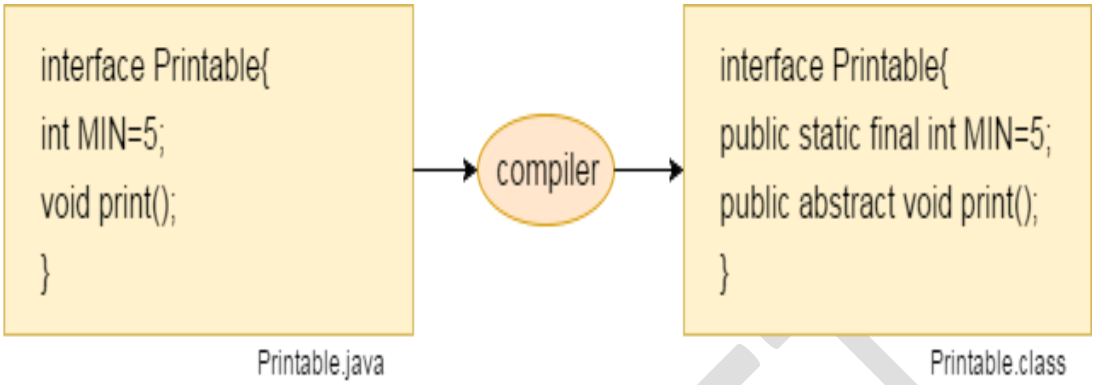
•The java file must have the same name as the interface.

• The methods that are declared have no bodies. They end with a semicolon after the parameter list. They are abstract methods; there can be no default implementation of any method specified within an interface.

•Each class that includes an interface must implement all of the methods.

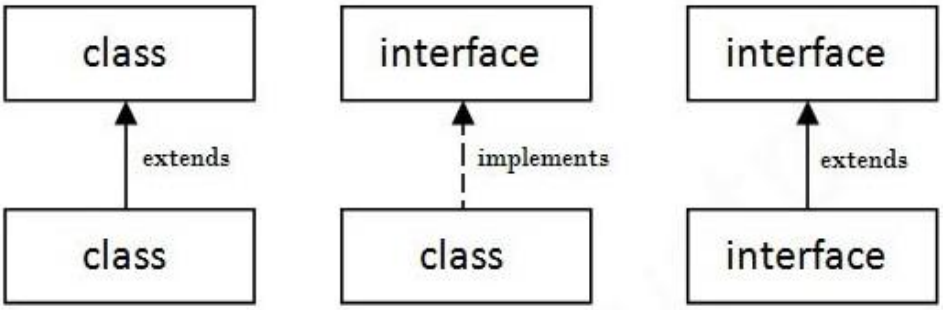
•Variables can be declared inside of interface declarations. They are implicitly final and static, meaning they cannot be changed by the implementing class. They must

also be initialized.
•All methods and variables are implicitly public



The relationship between classes and interfaces

- As shown in the figure given below, a class extends another class, an interface extends another interface, but a class implements an interface.



Example:
In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

```
interface printable{
void print();
}
class A implements printable{
public void print()
{
System.out.println("Hello");
}
public static void main(String args[]){
A obj = new A();
obj.print();  }  }
```

Output:
Hello

Multiple inheritance in Java by interface
If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.
Example:

```
interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}
public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

Difference between abstract class and interface

Abstract class	Interface
Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class
The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".
class members can private, protected, etc.	Members of a Java interface are public by default.

Example: public abstract class Shape{
public abstract void draw();
}
	Example: public interface Drawable{
	void draw();
	}

Abstract class doesn’t provide full abstraction	Interface does provide full abstraction.

What is package:

Packages are used in Java, in-order to avoid name conflicts and to control access of class, interface and enumeration etc. A package can be defined as a group of similar types of classes, interface, enumeration or sub-package. Using package it becomes easier to locate

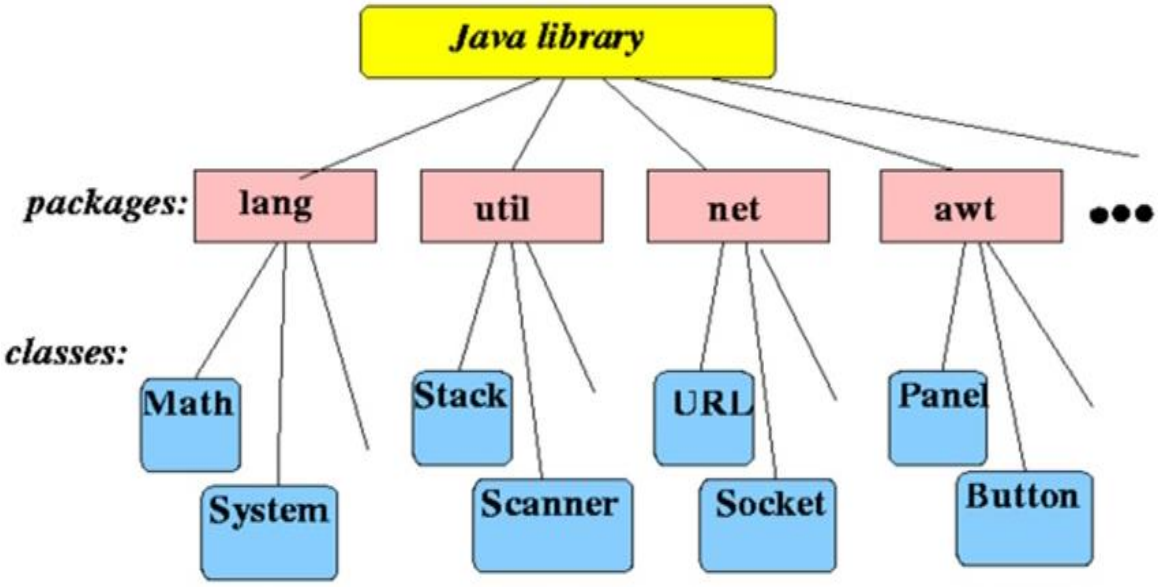
the related classes and it also provides a good structure for projects with hundreds of classes and other files.

Benefits of using package in java:

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.
4) This packages can be provide reusability of code.
5) We can create our own package or extend already available package.

Java Packages Types:

• **Built-in Package:** Existing Java package
Example java.lang, java.util ,java.io etc.
• **User-defined-package:** Java package created by user to categorize their project's classes and interface.



Built-in Packages:

- These packages consist of a large number of classes which are a part of Java API.Some of the commonly used built-in packages are:

1) java.lang: Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.
2) java.io: Contains classed for supporting input / output operations.
3) java.util: Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
4) java.applet: Contains classes for creating Applets.
5) java.awt: Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).
6) java.net: Contain classes for supporting networking operations.
7) java.math − This packages provides classes and interfaces for multiprecision arithmetics.
8) java.sql − This packages provides classes and interfaces for accessing/manipulating the data stored in databases and data sources.

9) java.text − This packages provides classes and interfaces to handle text, dates, numbers, and messages.

All these packages are available in the rt.jar file in the bin folder of your JRE (Java Runtime Environment). Just like normal packages, to use a particular class you need to import its respective package.

User Defined Packages

User-defined packages are those which are developed by users in order to group related classes, interfaces and sub packages. With the help of an example program, let's see how to create packages, compile Java programs inside the packages and execute them.

How to Create a package:

Creating a package in java is quite easy. Simply include a package command followed by name of the package as the first statement in java source file.

```
package mypackage;
public class student
{
Statement;
}
```

The above statement will create a package name mypackage in the project directory.

Java uses file system directories to store packages. For example the .java file for any class you define to be part of mypackage package must be stored in a directory called mypackage.

Additional points about package:

• A package is always defined as a separate folder having the same name as the package name.

• Store all the classes in that package folder.

• All classes of the package which we wish to access outside the package must be declared public.

• All classes within the package must have the package statement as its first line.

• All classes of the package must be compiled before use (So that they are error free)

Example of Java packages:

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

How to compile Java packages:

This is just like compiling a normal java program. If you are not using any IDE, you need to follow the steps given below to successfully compile your packages:

1. java -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 destination. The . represents the current folder.

Including a Class in Java Package

```
package MyPackage;
 public class Compare {
  int num1, num2;
Compare(int n, int m) {
     num1 = n;
     num2 = m;}
public void getmax(){
    if ( num1 > num2 ) {System.out.println("Maximum value of two numbers is " + num1);}
    else { System.out.println("Maximum value of two numbers is " + num2);}}
 public static void main(String args[]) {
 Compare current[] = new Compare[3];
 current[1] = new Compare(5, 10);
 current[2] = new Compare(123, 120);
 for(int i=1; i < 3 ; i++){
 current[i].getmax();} }}
```

Output:
Maximum value of two numbers is 10
Maximum value of two numbers is 123

Creating a class inside package while importing another package

```
package pack;
Public class A{
Public void msg()
{
System.out.println(“HI”);
}
}
Package bigpack;
import pack*;
class B
{
```

```
Public static void main(String args[])
{
A obj=new A();
Obj.msg();
}
}
```

So, the order when we are creating a class inside a package while importing another package is,

- Package Declaration
- Package Import