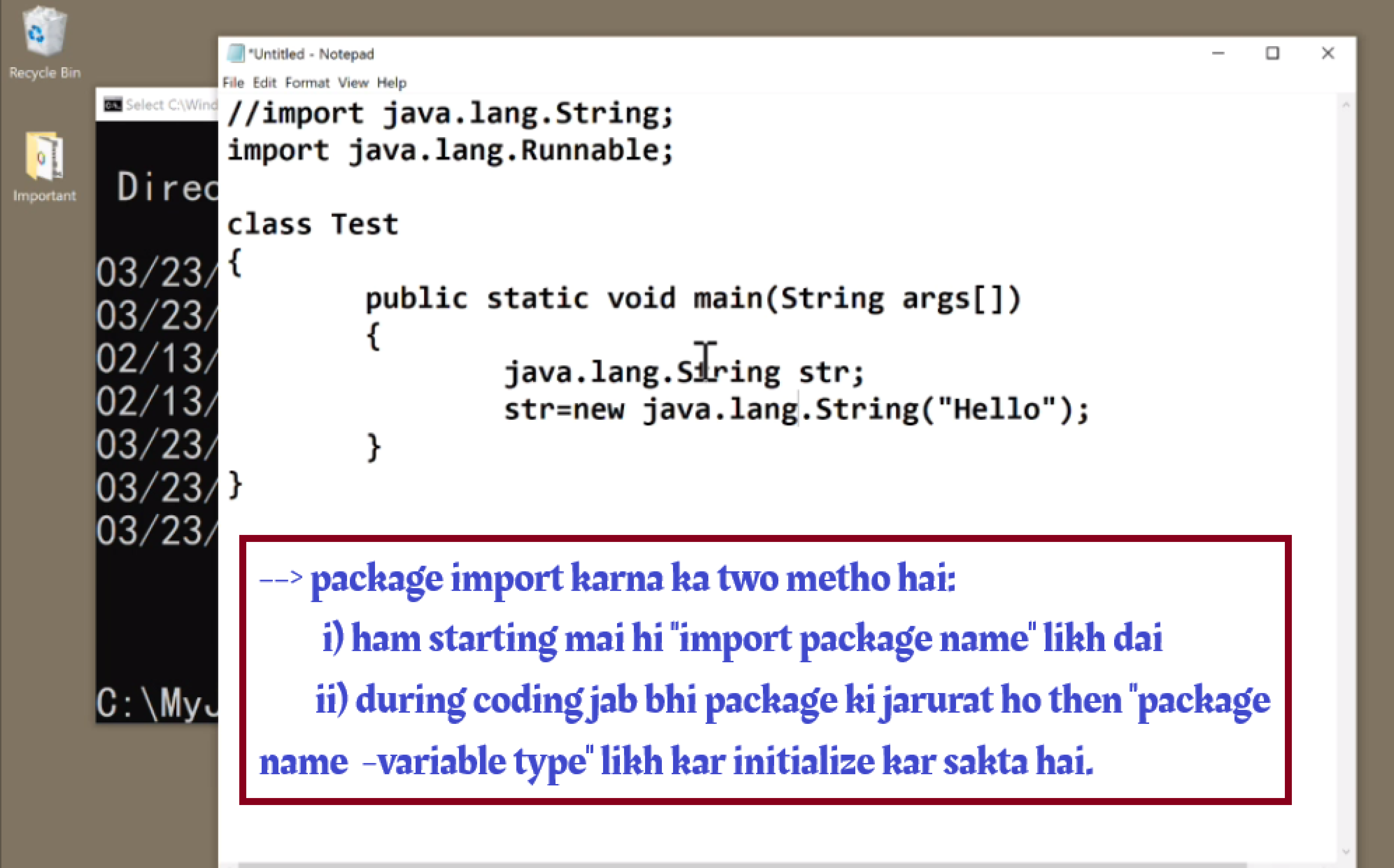


```
 Direc

03/23/
03/23/
02/13/
02/13/
03/23/
03/23/
03/23/


C:\MyJ
```

```java
//import java.lang.String;
import java.lang.Runnable;

class Test
{
        public static void main(String args[])
        {
                java.lang.String str;
                str=new java.lang.String("Hello");
        }
}
```

--> package import karna ka two metho hai:

i) ham starting mai hi "import package name" likh dai

ii) during coding jab bhi package ki jarurat ho then "package name -variable type" likh kar initialize kar sakta hai.

Package Naming Convention

http://www.univ.com

com.univ.academics

University

Student

Faculty

Couse

Book

Library

.academics

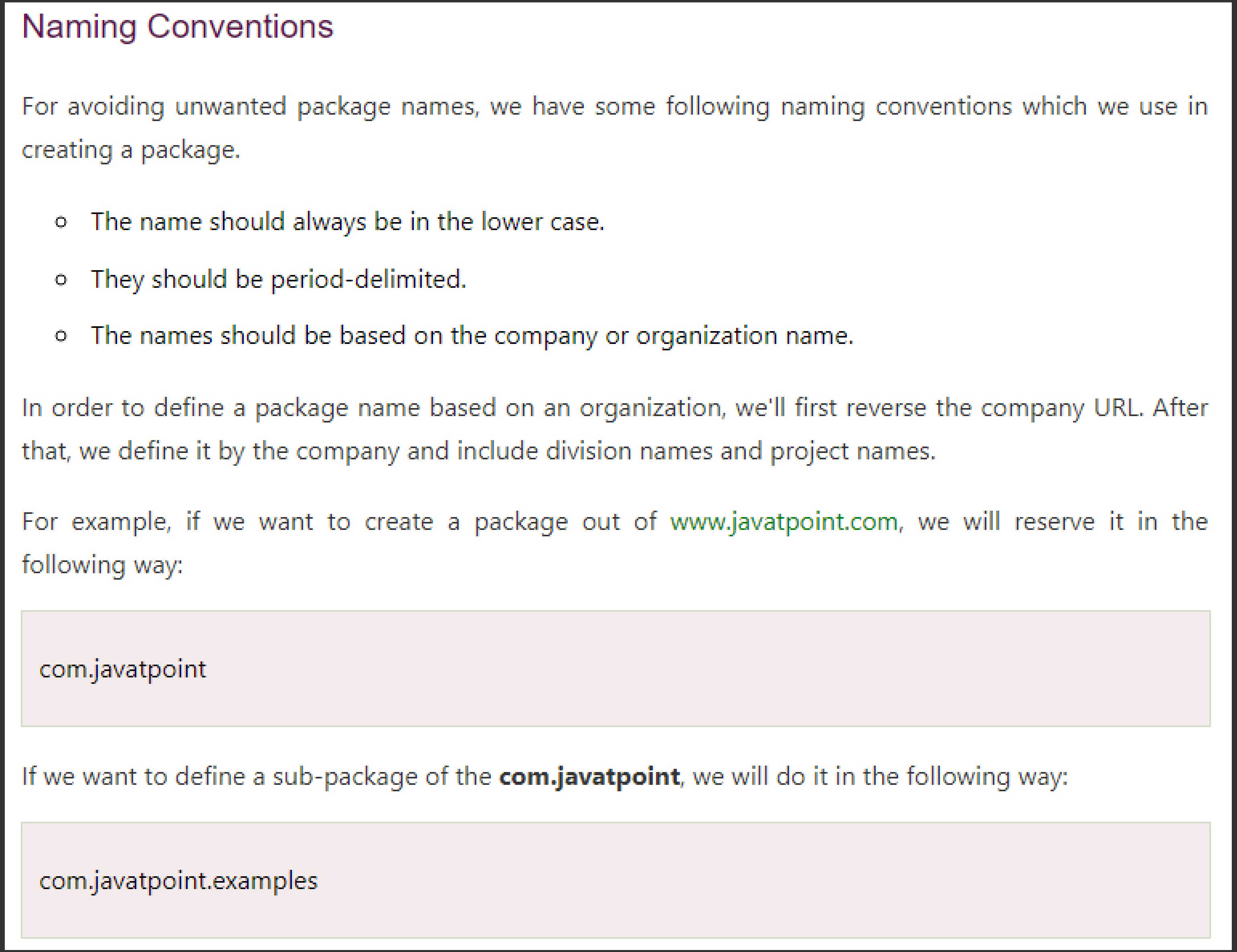
.admissions

.examination

Account

~~Library~~ Account

com.univ.admissions.Account



Naming Conventions

For avoiding unwanted package names, we have some following naming conventions which we use in creating a package.

- The name should always be in the lower case.
- They should be period-delimited.
- The names should be based on the company or organization name.

In order to define a package name based on an organization, we'll first reverse the company URL. After that, we define it by the company and include division names and project names.

For example, if we want to create a package out of www.javatpoint.com, we will reserve it in the following way:

```
com.javatpoint
```

If we want to define a sub-package of the **com.javatpoint**, we will do it in the following way:

```
com.javatpoint.examples
```

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.

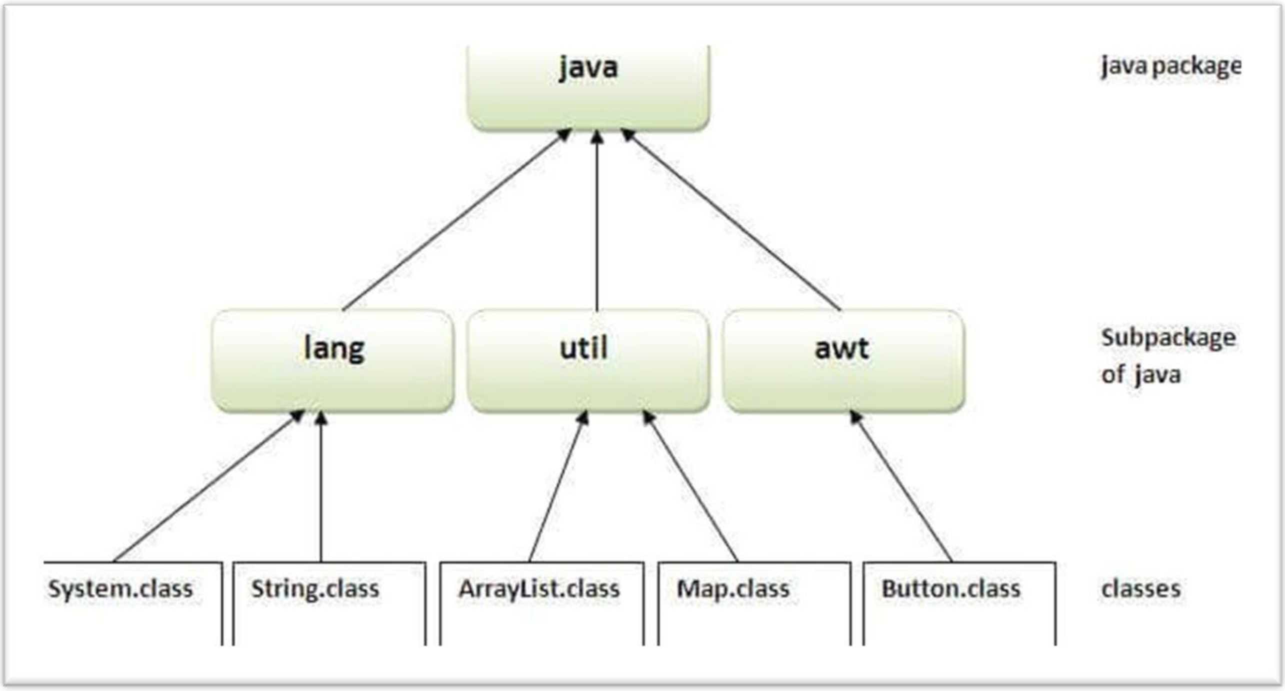
Here, we will have the detailed learning of creating and using user-defined packages.

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.*



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 destination. The . represents the current folder.

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");}
}
```

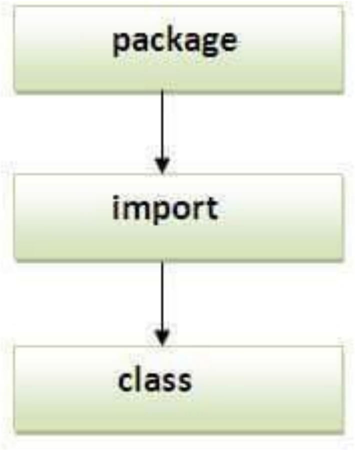
```
//save by B.java
package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```

```
Output:Hello
```

> *Note: If you import a package, subpackages will not be imported.*

If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.

Note: Sequence of the program must be package then import then class.



Subpackage in java

*Package inside the package is called the **subpackage**. It should be created **to categorize the package further**.*

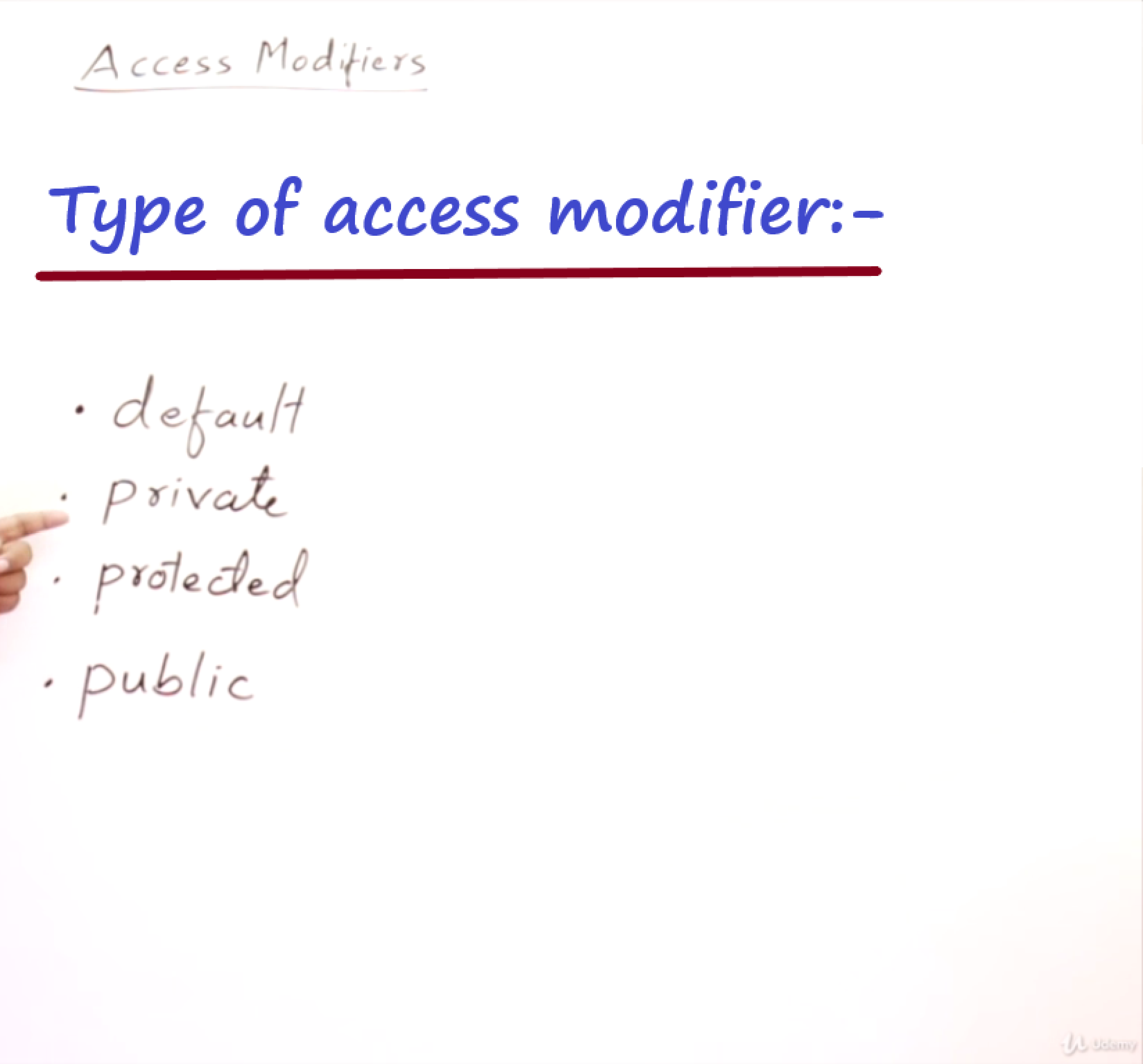
Let's take an example, Sun Microsystem has definded a package named java that contains many classes like System, String, Reader, Writer, Socket etc. These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on. So, Sun has subcategorized the java package into subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.

> *The standard of defining package is domain.company.package e.g. com.javatpoint.bean or org.sssit.dao.*

Example of Subpackage

```
package com.javatpoint.core;
class Simple{
  public static void main(String args[]){
   System.out.println("Hello subpackage");
  }
}
```

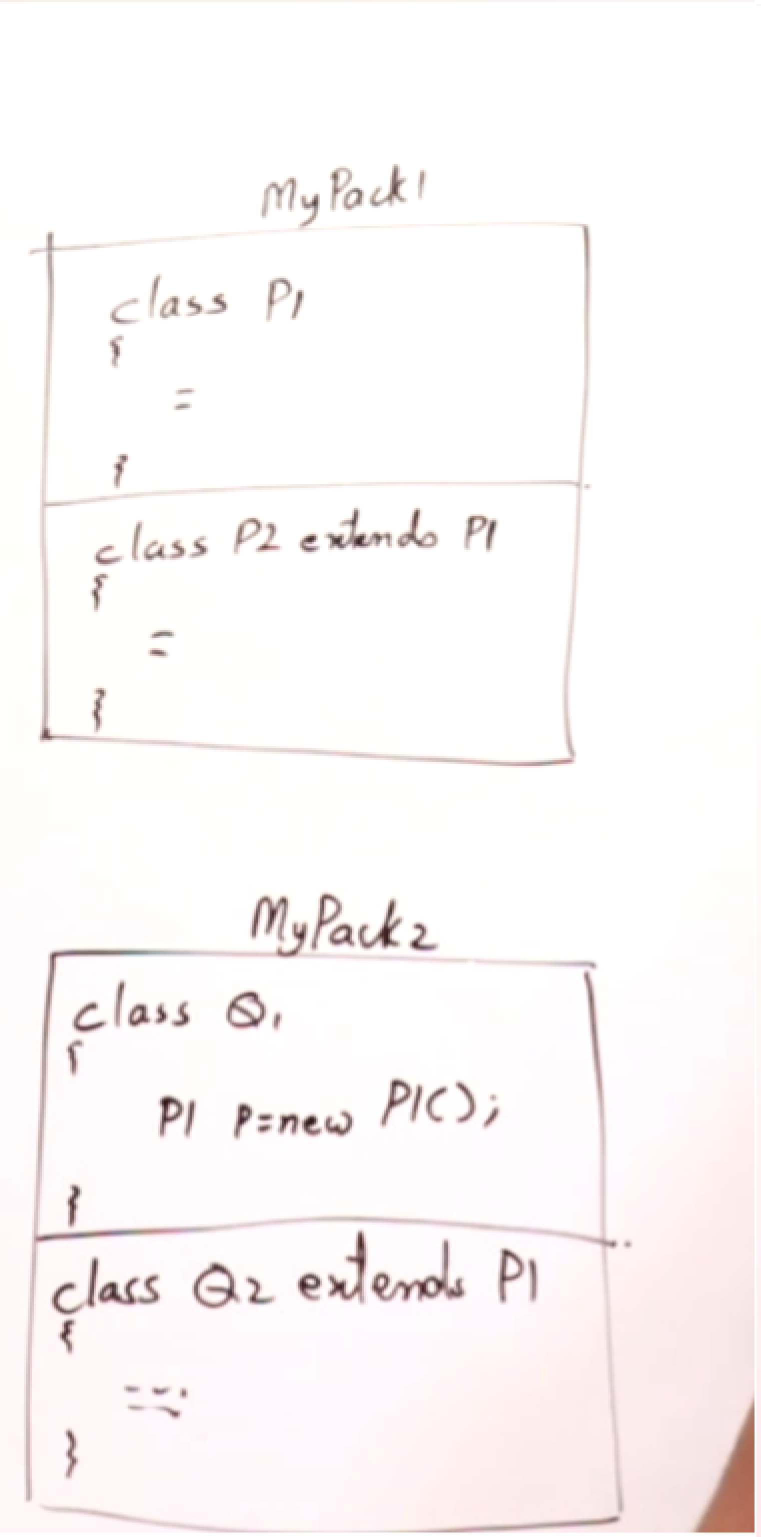
***To Compile:** javac -d . Simple.java*
***To Run:** java com.javatpoint.core.Simple*



Access Modifiers

Type of access modifier:-

- default
- private
- protected
- public

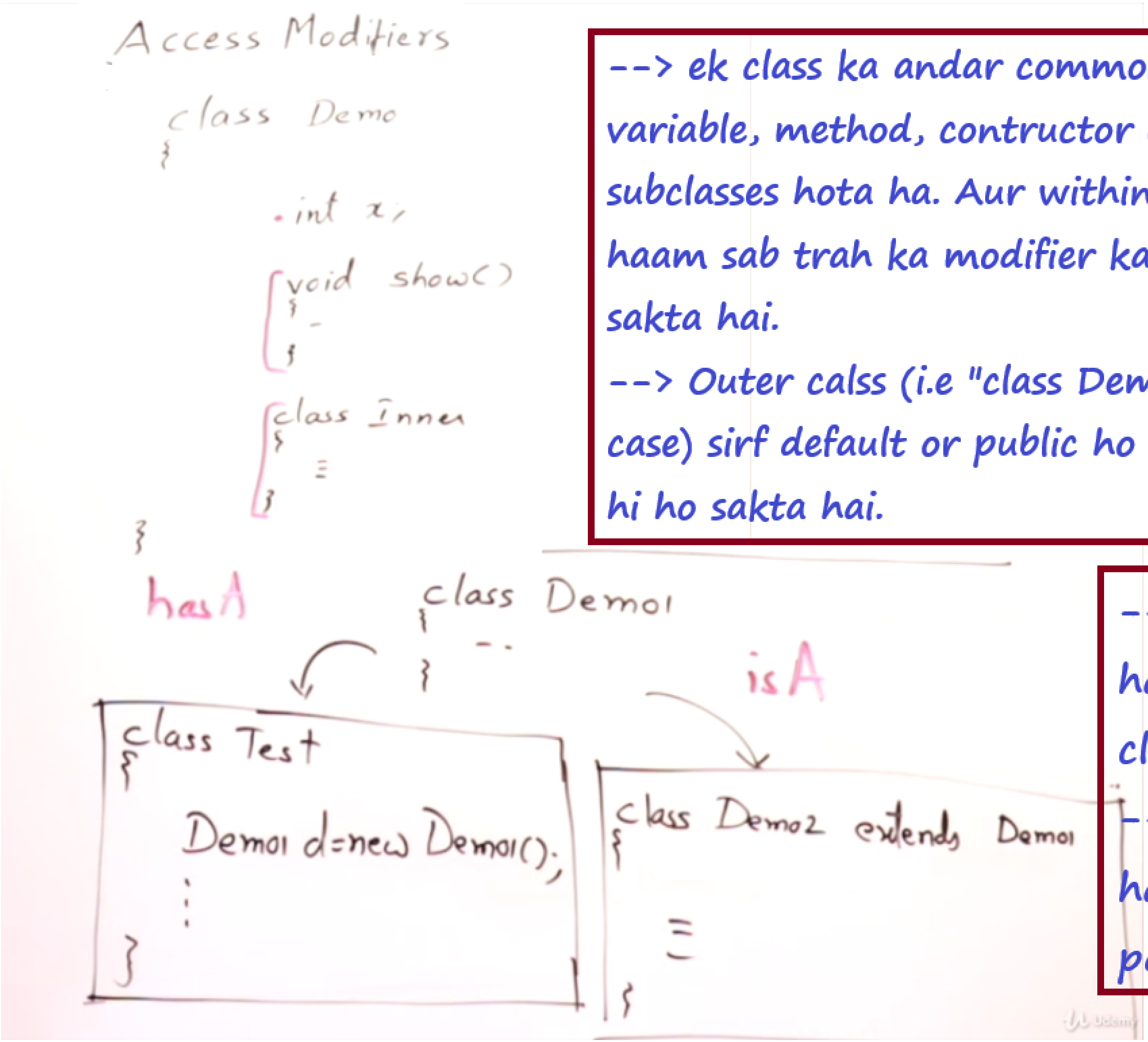


MyPack1

```
class P1
{
    =
}
class P2 extends P1
{
    =
}
```

MyPack2

```
class Q1
{
    P1 p=new P1();
}
class Q2 extends P1
{
    =:
}
```



Access Modifiers

```
class Demo
{
        . int x;
        void show()
        {
          -
        }
        class Inner
        {
          =
        }
}
```

--> ek class ka andar commonly, variable, method, contructor or subclasses hota ha. Aur within class haam sab trah ka modifier ka use kar sakta hai.

--> Outer calss (i.e "class Demo" in this case) sirf default or public ho sakta hai. hi ho sakta hai.

hasA

```
class Demo1
{
   --
}
```

isA

```
class Test
{
    Demo1 d=new Demo1();
    :
}
```

```
class Demo2 extends Demo1
{
    =
}
```

--> "has A" ka mtlb hota hai having a object of class.

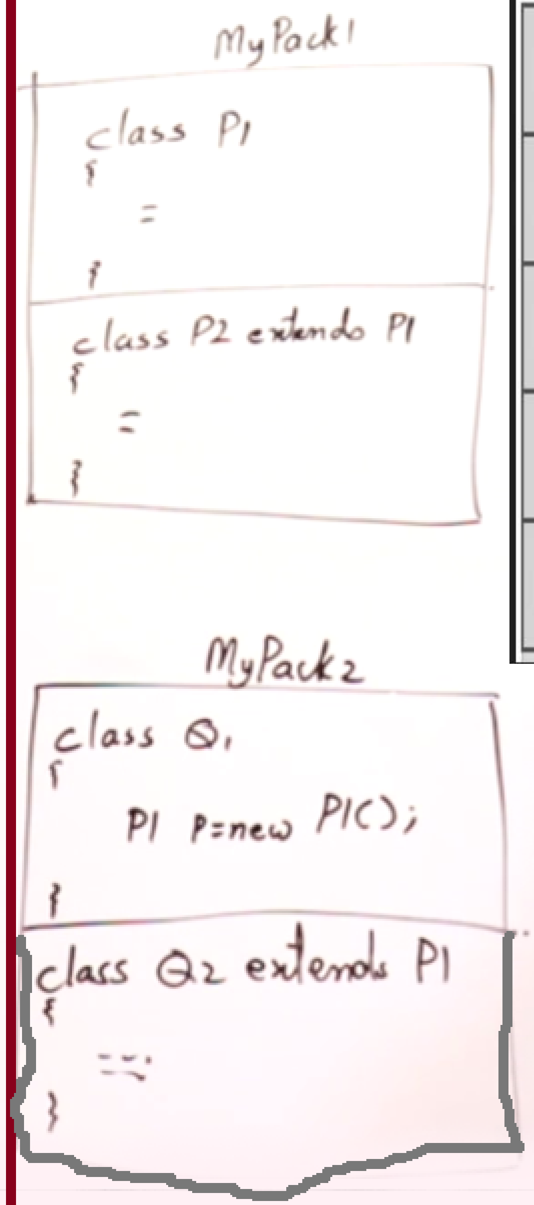
--> "is A" ka mtlb hota hai it is inherting from parent class.

--> class p1 is super class.

--> class p2 is sub-class of class p1 from same package.

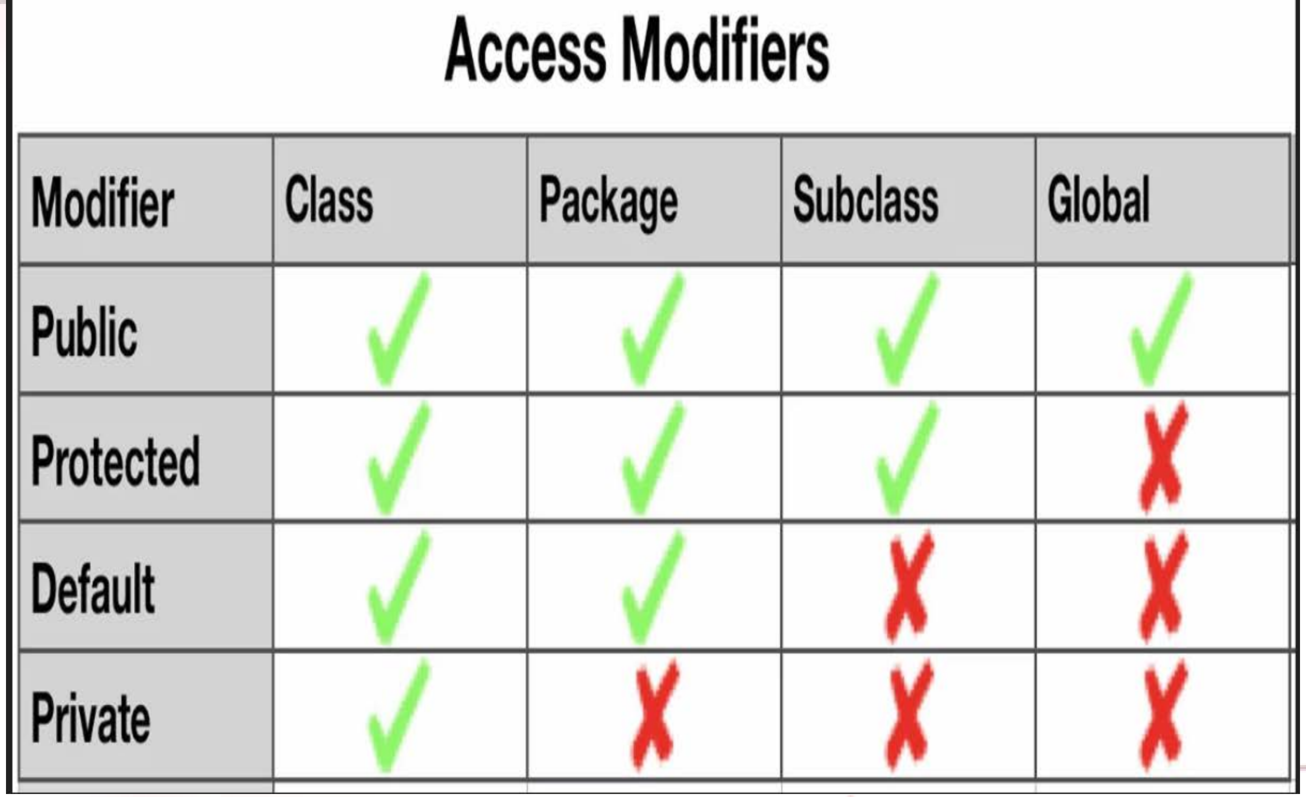
--> class Q1 is non sub-class in different package.

-->class Q2 is sub-class of class p1 from different package.



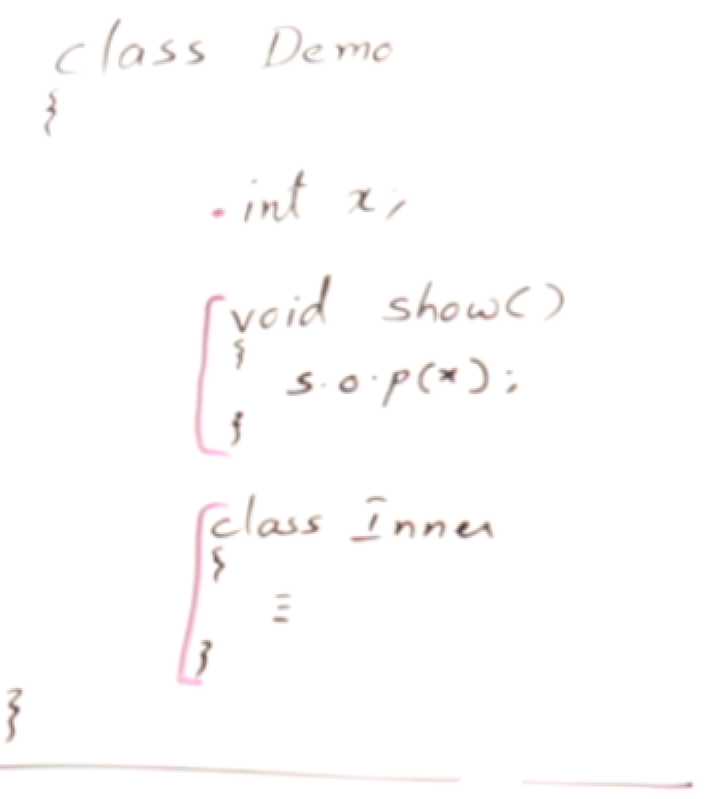
```
class P1
{
   =
}
class P2 extends P1
{
   =
}
```

```
class Q1
{
   P1 p=new P1();
}
class Q2 extends P1
{
   ==
}
```

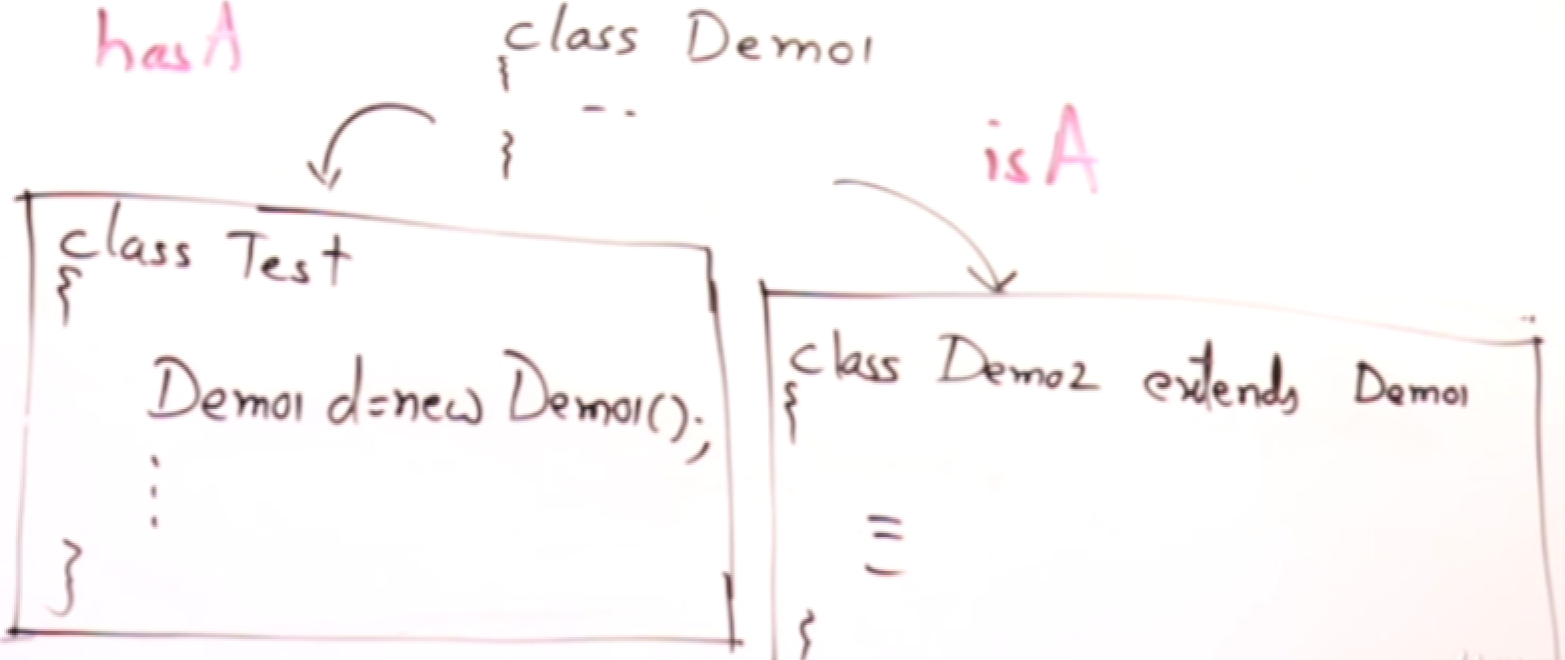


Access Modifiers

Modifier	Class	Package	Subclass	Global
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
Default	✓	✓	✗	✗
Private	✓	✗	✗	✗



```
class Demo
{
    int x;
    void show()
    {
        s.o.p(x);
    }
    class Inner
    {
        =
    }
}
```



```
class Demo1
{
   --
}
```

```
class Test
{
    Demo1 d=new Demo1();
    :
}
```

```
class Demo2 extends Demo1
{
    =
}
```