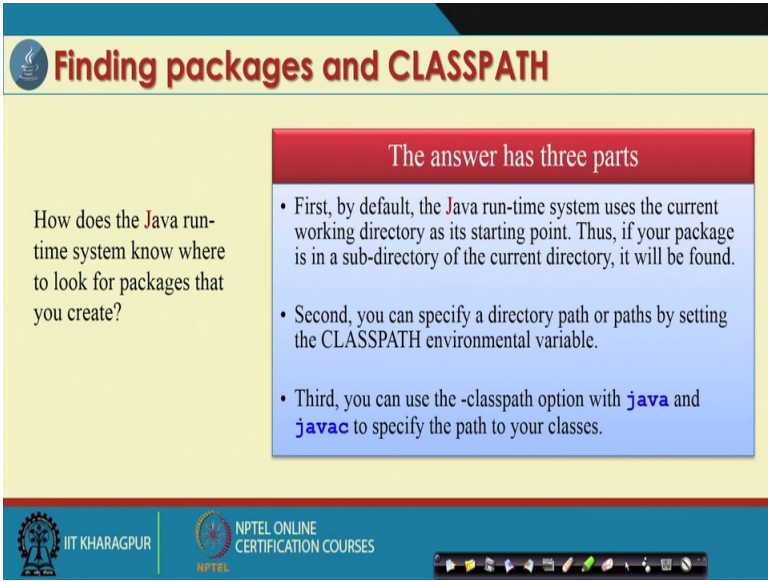


Programming in Java
Prof. Debasis Samanta
Department of Computer Science Engineering
Indian Institute of Technology, Kharagpur

Lecture – 18
Packages – II

So we are discussing Packages in Java. In the last module we have introduced the concept of packages; that means, how the build in of a API packages are there in Java JDK and then how a user can define their own package. Now today we will discuss much more information about Java packages. This is regarding access and then maintaining and then specification. So, today basically discuss mainly we will discuss about how a package can be accessed, and so for the access is concerned, how the different access specification works in case of packages it is there.

(Refer Slide Time: 00:58)



Finding packages and CLASSPATH

How does the Java run-time system know where to look for packages that you create?

The answer has three parts

- First, by default, the Java run-time system uses the current working directory as its starting point. Thus, if your package is in a sub-directory of the current directory, it will be found.
- Second, you can specify a directory path or paths by setting the CLASSPATH environmental variable.
- Third, you can use the -classpath option with **java** and **javac** to specify the path to your classes.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, first thing is that how, whenever you are running one program, how your runtime environment runtimes manager, Java runtime inward interpreter can understand that which package is located where. In order to give this information, it is better is that you can set the 6

classpath. Classpath is basically a system variable, environment variable in any operating system windows, in Unix or in MacOS Mac OS, whatever it is there, there is a way of setting the path.

So, the classpath is one variable, the environment variable is where you have to set it. So, classpath basically should tell where your packages are located. So, whenever you refer to a package, the Java runtime interpreter will consult this classpath name and then get the exact location to locate it. So, this is the 3, there are 3 ways; in fact, by which you can specify the explicitly you can specify the location of a package. So, the first is that it is a default of course that it considers that all the packages are under your working directory.

So, Java runtime interpreter, if you do not mention any classpath in your environment very well. We will try to find by default any directory which has the same name as the package name that you are mentioned in your import statement and that directory if it is there under your working directory.

So, these are the first method very trivial method of course. So, if you maintain one package under a working directory of your program where you want to run it, then it will work. The second method is basically setting the classpath environment variable, this is the way and this is the more sophisticated way I should say because all that is in you may not maintain all the package directory under your working directory.

You can maintain here and there and here. So that is fine, but if you can explicitly mention that this is the name of the package and this is the classpath it will work for you to access this package from anywhere, it is not necessary that all package should be under your working directory. And the second third is explicitly you can call a particular package by the Javac or command. So, the Javac suppose you want to run one program which is there in say xy; z package which is under abc.

So, you should write at javacabc.xyz.the name of the class file that is there. So, it will run it. So, this is the Java and javac, next we explicitly mention there. So, these are the three ways it is there, so you can use it.

(Refer Slide Time: 04:08)

Finding packages and CLASSPATH: Example

Consider the following package specification:

```
package MyPack;
```

For a program to find *MyPack*, one of three things must be true.

- The program can be executed from a directory immediately above *MyPack*
- The CLASSPATH must be set to include the path to *MyPack*
- The -classpath option must specify the path to *MyPack* when the program is run via *java*.

When the second two options are used, the class path must not include *MyPack*, itself. It must simply specify the path to *MyPack*.

Example: In a Windows environment, if the path to *MyPack* is
C:\MyPrograms\Java\MyP
then the class path to *MyPack* is
C:\MyPrograms\Java

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

.Now, here again as an example, so that I can discuss about that suppose *MyPack* is the user-defined package is there and you can mention that this my pack is under your working directory, immediately after your working directory or you can use the 2 other method classpath setting or explicitly mentioning while you are invoking *Javac* or *Java* that this is the location.

Now, if you mention the classpath set; So then *MyPack* and then you just mention the classpath as C My programs under Java and then name of the directory. So, it is basically to mention that in which root directory or parent directory under which the path that only needs to be mentioned in your class. So, basically, Java runtime interpreter will consult this classpath in one variable and check that whether in the according to that path your package is there or not. If it is not there, it will that package does not found or no class file available.

(Refer Slide Time: 05:14)

Finding packages and CLASSPATH: Example

```
package MyPack;
class Balance {
    String name;
    double bal;
    Balance(String n, double b) {
        name = n;
        bal = b;
    }
    void show() {
        if (bal < 0)
            System.out.print("Account is dead");
        else
            System.out.print(name + ": $ " + bal);
    }
}
```

```
class AccountBalance {
    public static void main(String args[]) {
        Balance current[] = new Balance[3];
        current[0] = new Balance("K. J. Fielding", 123.23);
        current[1] = new Balance("Will Tell", 157.02);
        current[2] = new Balance("Tom Jackson", -12.33);
        for (int i=0; i<3; i++)
            current[i].show();
    }
}
```

MyPack
import a-y. Balance

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this kind of command it will be there. And here is an example of a very simple example that we can consider it. Here we have declared a 1 class, the name of the class is balanced and then it is declared in package MyPack. So, we are basically creating 1 package, the name of the package is MyPack and in this pack package the class is class balance and one thing that it is not here although, but you should consider that this as I already told you that all the class which belongs to a package should be declared as a public.

So, here this is required, so public. So, public access specification it is required, if you do not do then you will not be able to use it. And here again, this is if it is a public and all the access, specification by virtue of default access specification is also public. So, these are all basically it is a public class, it is a public class and in this under this public class, we decide we declare name as a data type string balance, ball is a type float and there is a method the this is also public method is a constructor and as it is public this is also public.

So, it is a public constructor and it also has a void method, show method also public like. So, this is a typical look of a class basically it will create an object of type class balance, containing all these fields and the methods. So, is basically the package the class is there, once this package is created, we can use it, you can use this class file in any program and this is the one class you can define, it is basically this my pack can be created under this

directory where this, this class is defined. So, this is the class file under a working directory and this is the file in a package.

Now, this is the typical look of the program is basically this is the class file account balance main class because the main method is there and it will be able to access all the members which are there without any error because this is a class all methods are class. So, it is accessible to any class from anywhere it is like this because of the package access specification. So, this basically gives a simple example that a class can be created can be stored in a package and that package, that class can be utilized in any other Java programs as if it is your own defined class like.

So, this is the concept and then classpath if we say the classpath of this MyPack, then that there is an import statement that you have to do it. So, here; obviously, the import statement is missing. So, import MyPack in this case, import MyPack and then use it, so you can use it like that on ok. And this import MyPack once it is mentioned, it will consult that Java runtime interpreter will consult that this MyPack is a package it located where, so it is like this; otherwise the second way it is that instead of this one, we can write import and then full setting x.y. This one and then we can write balance. That means, this means that this is an x is a root directory, under this y directory under that directory the MyPack and then under that directory, this balance file is there.

So, this is also one way by which you can specifically tell other than the classpath setting is there. So, these are the way that you can use it to define the different way so that the location of a particular class belongs to a package can be specified.

(Refer Slide Time: 09:25)

Finding packages and CLASSPATH: Example

Call this file *AccountBalance.java* and put it in a directory called *MyPack*.

Then, try executing the *AccountBalance* class, using the following command line:
`java MyPack.AccountBalance`

Next, compile the file. Make sure that the resulting *.class* file is also in the *MyPack* directory.

Remember, you will need to be in the directory above *MyPack* when you execute this command.

As explained, *AccountBalance* is now part of the package *MyPack*. This means that it cannot be executed by itself. That is, you cannot use this command line:
`java AccountBalance`

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, again, in a nutshell, we have to create your own program. The name of the program as you have discussed earlier that `account balance.java` and put it in a directory and that directory before the `MyPack` directory whatever it is there. If you put into the same directory as the package also it will work no issue.

Then we have to compile the file so that, that class file is there and then try to execute the account balance class file using the command whatever it is there, and then you can get it. So, it is the basic idea it is there. Now, here you see in this case, in this example that I have discussed here account balance is your program.

(Refer Slide Time: 10:12)

Finding packages and CLASSPATH: Example

```
package MyPack;
class Balance {
    String name;
    double bal;
    Balance(String n, double b) {
        name = n;
        bal = b;
    }
    void show() {
        if (bal < 0)
            System.out.print("Account is dead");
        else
            System.out.print(name + ": $ " + bal);
    }
}
```

```
class AccountBalance {
    public static void main(String args[]) {
        Balance current[] = new Balance[3];
        current[0] = new Balance("K. J. Fielding", 123.23);
        current[1] = new Balance("Will Tell", 157.02);
        current[2] = new Balance("Tom Jackson", -12.33);
        for (int i=0; i<3; i++)
            current[i].show();
    }
}
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, it is basically if you see the program here. This is the account balance is the user-defined one class, we have created here and this is the package class, but they can be put into together in the same class directory, so no need to specify explicitly the location of the import statement is not required.

If you want to put all those things in the same directory, actually no import location is there, but if they are in a different directory and then, then import statement is a must. Now, here fine, so here again ok, so there containing this discussion. So, account balance the class that we have created as an application Java program, we stored into the same as a package directory MyPack and then compile it, it will work absolutely no problem no need to this one.

Now, see here account balance the class that you have created is now part of the MyPack. That means, any other program can use it, but again be careful. The main class can be used in any import statement only once. If you want to access one class which is in the package having the main class, in addition to the main class in your own application then again it is a compilation error. So, you cannot do that you should not do that. So, usually it is good practice that the package should contain only the non-main classes; that is what that thing it is there and you can use it like that ok.

So, this is the concept that we have discussed it. Now regarding importing a package, we have to consider a few more cases, a few more steps to be considered very carefully which we are going to discuss now.

(Refer Slide Time: 11:52)

Importing a package

This is the general form of the import statement:

```
import pkg1 [.pkg2].(<classname> | *);
```

Here, **pkg1** is the name of a top-level package
pkg2 is the name of a sub-ordinate package inside the outer package separated by a dot (.).

There is no practical limit on the depth of a package hierarchy, except that imposed by the file system.

Finally, you specify either an explicit class name or a star (*), which indicates that the Java compiler should import the entire package.

Example:

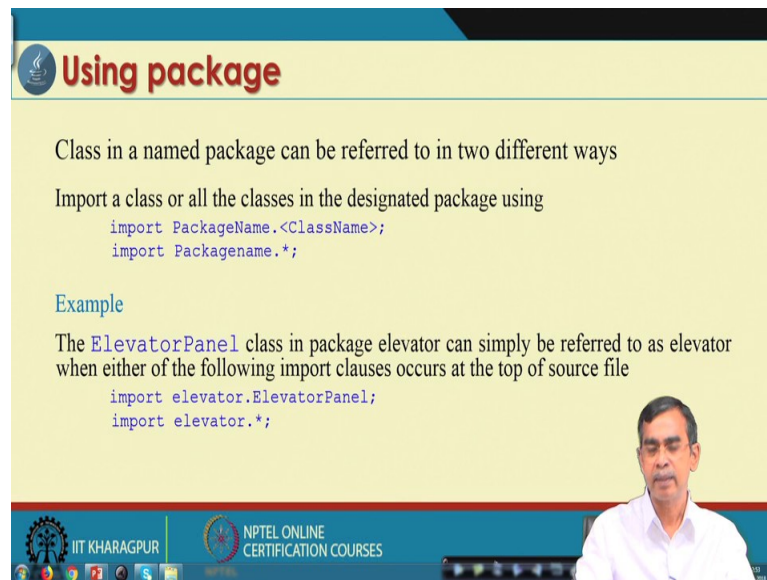
```
import java.util.Date;  
import java.io.*;
```

So, for the import statement is concerned, there are two ways that import can be explained. So, import only the name of the package, it is possible if you decide that package in a classpath setting. If we do not do then you use the.comma.a statement that.dot.explicitly specifying the location of the particular package it is there.

In this example, as we can see, so if package 1 is the name of top level and then package 2 is the sub package and under this package a particular class that you want to access it, then you can write it using import package 1,.package 2 then.class name. So, is a.common.the periods actually can be used there?

So, either. as a particular class name explicitly or.star all classes belong to that package whatever it is there; otherwise here this is also one way if you can specify as an example right. So, this is basically is an illustration of this concept is basically import Java.util.date this means that I want to use that date class in my class now and this is also basically I want to import all classes which belong to.io packages that are there in JDK. So, these are the basically util and io are the two API packages and we can access it ok. So, this is the import statement is very simple that import statement is there we have to specify it.

(Refer Slide Time: 13:28)



Using package

Class in a named package can be referred to in two different ways

Import a class or all the classes in the designated package using

```
import PackageName.<ClassName>;  
import PackageName.*;
```

Example

The `ElevatorPanel` class in package `elevator` can simply be referred to as `elevator` when either of the following import clauses occurs at the top of source file

```
import elevator.ElevatorPanel;  
import elevator.*;
```

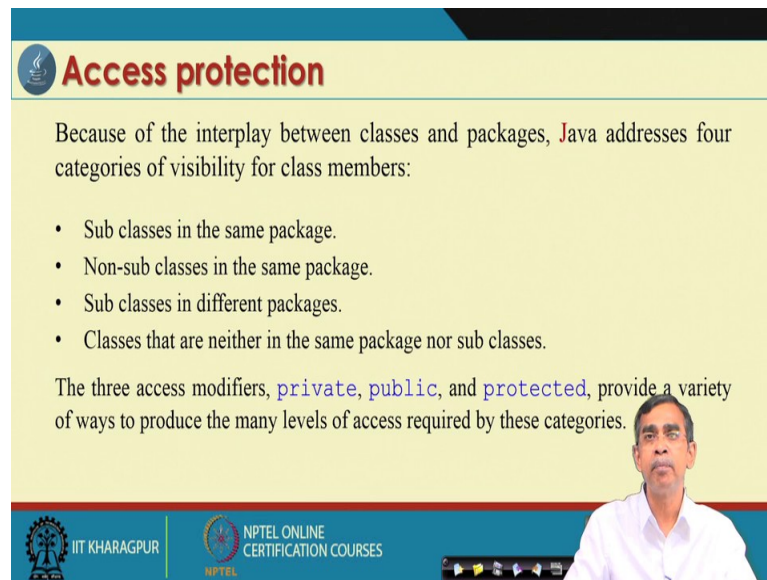
The slide features a speaker overlay in the bottom right corner. The footer includes the IIT KHARAGPUR logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'.

And also we can mention using your own package you can just import package name. the class name the same concept, whenever you build a package whether it is an API package or it is a user-defined package. There is no discrimination, you can use you can access, in the same way, it is API or it is a user-defined package.

So, the concept is similar is generic rather no difference concept have to be followed. Now, so we have discussed package, maintenance and then how we can access it. Now I just want to discuss the access protection for a package this is very important and needs to need to be learned also very carefully, because if it is not maintained properly then either is a compilation error, you will not be able to build your program successfully, so many concepts are there.

So, access protection, already we have discussed while we are discussing the information hiding by virtue of 3 4 access specifier default public private protected. Now all these access modifiers are equally applicable to the package classes also. So, we will discuss how the access protection in the context of packages is.

(Refer Slide Time: 14:50)



Access protection

Because of the interplay between classes and packages, Java addresses four categories of visibility for class members:

- Sub classes in the same package.
- Non-sub classes in the same package.
- Sub classes in different packages.
- Classes that are neither in the same package nor sub classes.

The three access modifiers, `private`, `public`, and `protected`, provide a variety of ways to produce the many levels of access required by these categories.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, so here is basically the idea about that, because a package can play in between the different classes. So, it is basically interplay between classes.

So, Java should have a very good visibility control and naming convention equally. So, subclasses, as you see Java, addresses 4 categories of visibility for class members. It basically visibility for subclasses, which are in the same package; that means, a class is declared and a subclass is basically inherited class is also declared, and if they belong to the same package, this is the visibility one type of is there. The no non subclasses in the same package that is; obviously, the package is there, there are many classes not necessarily subclasses of other classes like this.

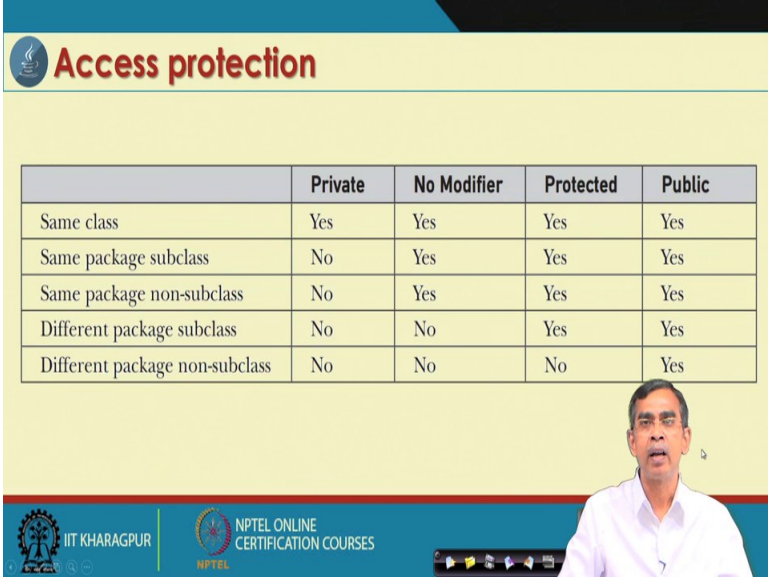
Subclasses in a different package; one class belong to say package p 1; whereas, from the class which belongs to p 1, there may be a subclass in say q in other package q another subclass of the class belong to p is in another sub packages r like this one. So, subclasses in different categories, so how the visibility can be controlled and then classes, those are neither in the same packages nor in the subclasses are there.

So, these are the 4 different ways, the visibility needs to be controlled and obviously, for this visibility, things are there, access specifier public private protected it is applicable. Now, the default is not a useful access specification so far the public is concerned. This is because the default is applicable to all the classes either in the same program file or in

the same packages is there. So, usually people avoid a default access specification, so, far the package is concerned.

So, if we use that default access specification, then you only limit to that package into that a limit to that classes into that directory that packages only. So, all classes belong to the same package can use the default classes that are there. But anyway, so we will discuss more critically all this access specification namely default private public and protected obviously we can follow some example to understand the concept it is there.

(Refer Slide Time: 17:22)



	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

Now, again the table that I want to place it so that all 4 categories of visibility that can be including the same class as you know if it is the same class any method can access any member belonging to the same class. So, visibility is basically true for all members to belong to the same class. So, it is the same class visibility is true for whether the access specification you specify, it does not matter. Now if it is the same package subclass; that means, a subclass which belongs to the same, except the private member, all other access spaces does not have any limitation.

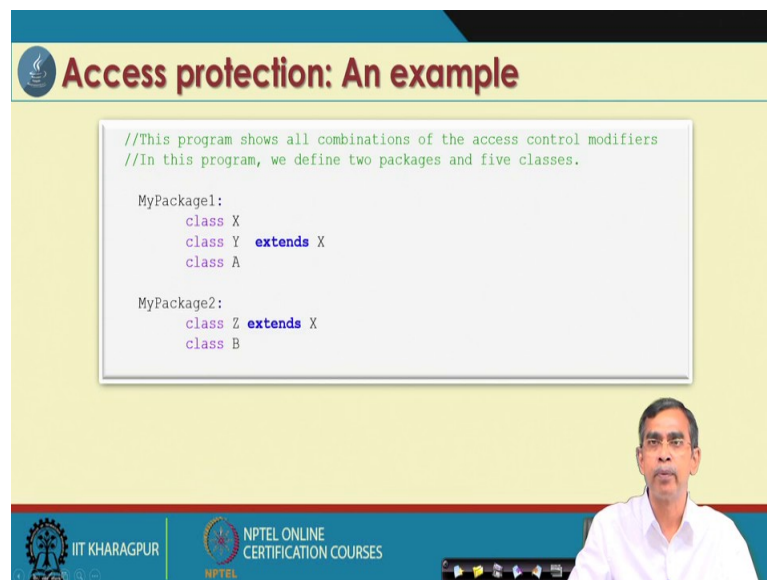
So, if you specify the private then no subclass method can access the private member. However, if it is a default it can no modifier mean default and then protected and the public can be accessible to the subclass. So, this is obvious, the same package non subclass. That means, it is not a subclass, but belong to the same package, so private cannot access, default cannot default can access, protected can access and then public;

obviously, it can access. So, if it is the same package, but not subclass then it can access it.

Now, different package subclass, so a subclass, but that subclass if it belongs to the different directories different package. So, private no way, a default will also no way, but protected it can, because it is a subclass and then public it can because it is public and then different package in subclass. So, private cannot access, default cannot access, protected cannot access; whereas, public it can access. So this is the table, it is a similar table that we have discussed in the context of the information accessing it is there.

So, this is the way you can think about it and then it will work for you now ok.

(Refer Slide Time: 19:28)



The slide is titled "Access protection: An example" and features a code block with the following content:

```
//This program shows all combinations of the access control modifiers
//In this program, we define two packages and five classes.

MyPackage1:
    class X
    class Y extends X
    class A

MyPackage2:
    class Z extends X
    class B
```

The slide also includes a video feed of a presenter in the bottom right corner and logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom.

So, after having this kind of summary of different access modification it is better to discuss about using some example. Now, this is the one example although little bit complex example because it is really difficult to maintain one simple example to illustrate all the features regarding the access protection in packages. Now even I have tried it to make it as simple as possible here. We consider is 2 packages, MyPackage 1 and MyPackage 2, they are different is not that in some same directory a subdirectory whatever it is there.

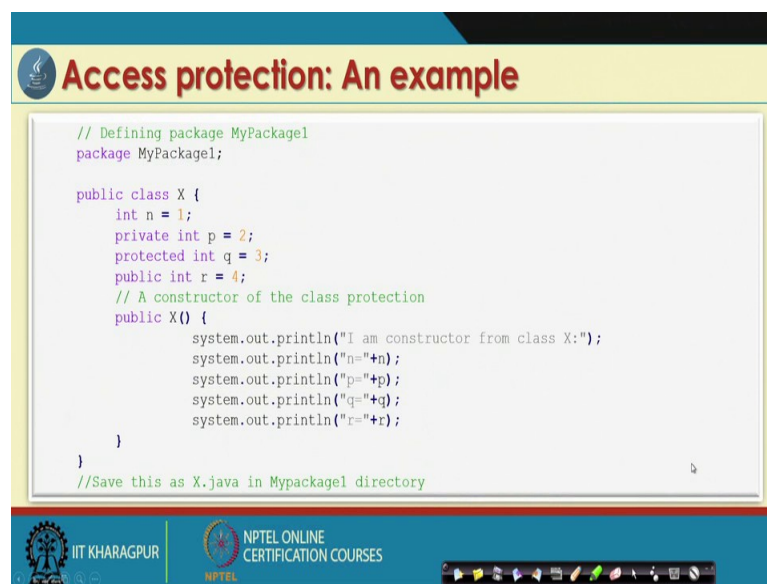
Let us consider 2 different subdirectories are there MyPackage 1 and the MyPackage 2. In this MyPackage 1, we plan 3 classes, class X, class Y, class A, which belongs to the

MyPackage 1 directory and class Y is a basically subclass of class X and as I told you if all their member of a package, they should be declared as public. So, here these are all public class, I have not mentioned explicitly a public class.

Similarly, here the two classes which belong to another package MyPackage 2 also public and Z is basically a derived class subclass subclass of class X. X is again super class belongs to the package 1. Fine, so this is the organization of the 2 packages, the 2 packages contain 5 different classes actually X, Y, Z, A, and B, out of which Y and Z are the subclasses. Now having this organization, let us have certain simple what is called a structure of the 4, 5 different classes 1 by 1.

Let us first have the idea about the first class X, which belong to the plus package 1.

(Refer Slide Time: 21:22)



The slide is titled "Access protection: An example" and displays the following Java code:

```
// Defining package MyPackage1
package MyPackage1;

public class X {
    int n = 1;
    private int p = 2;
    protected int q = 3;
    public int r = 4;
    // A constructor of the class protection
    public X() {
        system.out.println("I am constructor from class X:");
        system.out.println("n="+n);
        system.out.println("p="+p);
        system.out.println("q="+q);
        system.out.println("r="+r);
    }
}

//Save this as X.java in Mypackage1 directory
```

The slide footer includes the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the NPTEL logo.

So, we are we are defining we are the creating or building the class X belong to the package MyPackage 1. Here just note it, the class X contains 4 members including its own constructor and class X is public as I told you, it should be declared as public. Now, here int n, n is an integer type variable which is declared as public because the public is a default.

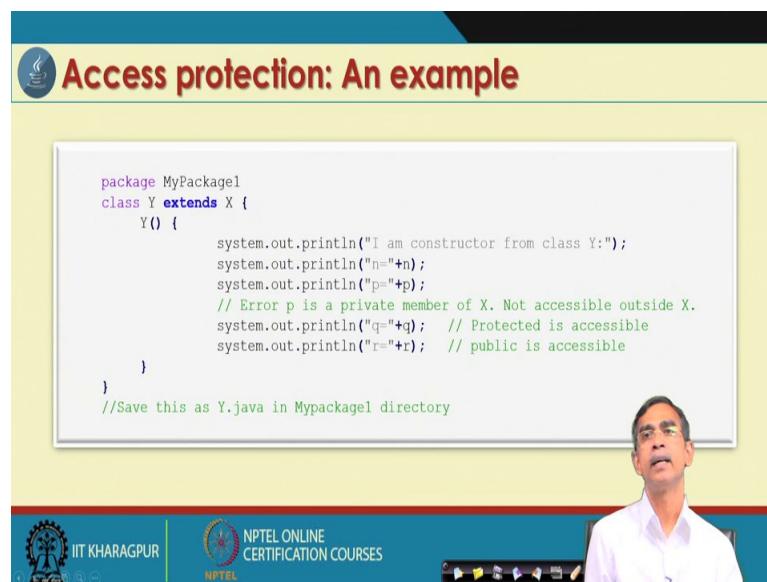
Now, p is a private member, q is a protected member, r is a public member again. So, default int is basically a default access specifier by default as if public. So, it will be public-private p is a private, q is protected and r is public. So, 4 members are declared

with 4 different access specification. Just np, qr we have to remember, that default means it is a public, private, protected and r is public. Now, if I ask you, r is public; this means that any class whether it is a subclass of this x or it is any other class belongs to any other r is accessible.

So, regarding r it does not a problem. .Now, here is the problem regarding either, default access specification in this case n and then p q all these things. So, n is a public, so no issue, the default is no issue, but again there is an issue, of course, we start discussing. Now let us save this class as Java file x.java and then compile it and then save it and this public X is basically constructor which will basically call this and you see a so far this constructor is concerned it is a method actually automatic method and this constructor will attempt to access all the members that mean n, p, q, r in this case.

So, far the accessing of this is concerned as it belongs to the same class, all these are ok. So, no compilation error it will put whenever you want to compile X.java file. So, this is the class X file, and I hope you have understood it clearly. Now let us come to another class file and this is the subclass of class X.

(Refer Slide Time: 23:42)



Access protection: An example

```
package MyPackagel
class Y extends X {
    Y() {
        system.out.println("I am constructor from class Y:");
        system.out.println("n="+n);
        system.out.println("p="+p);
        // Error p is a private member of X. Not accessible outside X.
        system.out.println("q="+q); // Protected is accessible
        system.out.println("r="+r); // public is accessible
    }
}
//Save this as Y.java in Mypackagel directory
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, Y extends X, now so far this class is concerned. You see it does not have any own member of its own, so no issue, but it will try to access the member which basically belongs to its superclass X.

.Now, here, here you see .Now, here is basically the constructor is basically by using the constructor I want to discuss it as there is a default, so it is public. So, you can say public it is and this is also public constructor know ok. So, this basically X, so derive it is basically is a derived class subclass. This means that all public members will be accessible and then all protective member will be accessible, but any private will not be able to access, this is the concept right and as this class Y belongs to the same directory as the class X is apart from this one. So all that default member there the default member is n is also accessible.

Now, so far this statement is concerned this is ok because it is default access in the same file accessible. .Now, here p that is declared as a private p is declared there as a um protected right, so it is also a private. So, private is you know p is declared let us see, what is the p is declared anyway. So, here let me just remind it how we have declared the p it is there.

So, here exactly if you see p is declared as a private and then q is declared as a protected and r is private that we have to remember somehow to understand this concept it is there. Now let us come here, so here we can see so n is the default, so it is accessible. .Now, here p is declared as a private, so there is a problem because private cannot be accessed by any other class this one, so that is why this statement will give you a compilation error if you put it common then only the compilation successful; otherwise it is there ok.

And then here if you see the q, q is a protected and as a Y is a derived class of this one, so this is and the r is a public, so it is accessible. Now in order to run this program successfully, this statement should be commented. So, that you can run it, if you do not comment it, it will give you a compilation error and compilation error will report the compilation at this point here only ok. So, this is the idea about the derived class subclass of class X, which belong to the MyPackage one.

Now, let us come to another class that we have already planned, which belongs to the package MyPackage 1 is the class A here.

(Refer Slide Time: 26:50)

Access protection: An example

```
// Defining package MyPackage1
package MyPackage1
public class X {
    int n = 1;
    private int p = 2;
    protected int q = 3;
    public int r = 4;
    // A constructor of the class protection
}
//Save this as X.java in MyPackage1 directory

//Save this as A.java in MyPackage1 directory
class A {
    // class with default protection
    A() {
        // default constructor with default access
        X x = new X() // create an object of class X
        System.out.println("Same package constructor ...");
        System.out.println("n from A"+x.n);
        // Default variable is accessible in the same package
        System.out.println("p from A"+x.p); // Error
        System.out.println("q from A"+x.q); // Error protection
        System.out.println("r from A"+x.r); // OK: public
    }
}
```

So, here is the declaration of class A here ok. So, this class A belongs to this MyPackage 1, we have included whatever the other main classes are there anyway. So, class A, we have also do not declare any other explicit member of its own, it has the only constructor and our objective is that whether this class a can access any other members belong to this class, belong to any other class either X, Y into the package or not as, Y does not have any member, so no question, it will consider only any members because that belongs to X.

Now, here look at this one, so we create an object X of class X; that means, we want to access any members that belong to the class X or not. Now, so far this one it is not an error, because it will work because n is defined as a default access specification by default it can be accessible to any class belong to the same directory, as this class A is belonged the same directory, it is possible. Now, here see this p, p is basically is a private and we want to access a private member from other class A here. So, this will report an error. So, it should be commented to successfully right.

And q is a protected, protected cannot be X less by a non subclass, as A is a non subclass, so it is an error and then r is a public member, so it is ok. So, this will work for you. So, here, in order to run this, so this and these 2 are to be commented. So, that it can successfully run it ok. So, this is the class 3 classes that we have discussed with the 3

different modifications it is there. Now let us come to another package called the MyPackage 2.

(Refer Slide Time: 28:40)

```
// Defining package MyPackage1
package MyPackage1
public class X {
    int n = 1;
    private int p = 2;
    protected int q = 3;
    public int r = 4;
    // A constructor of the class protection
}
//Save this as X.java in MyPackage1 directory

//Save this as Z.java in MyPackage2 directory
package MyPackage2;
class Z extends MyPackage1.X {
    Z() {
        System.out.println("I am constructor from class Z:");
        System.out.println("n from Z"); // Error:
        // Default is not accessible outside its package.
        System.out.println("p from Z"); // Error : private of X
        System.out.println("q from Z");
        // Protected member is accessible by inheritance
        System.out.println("r from Z");
        // public is accessible
    }
}
```

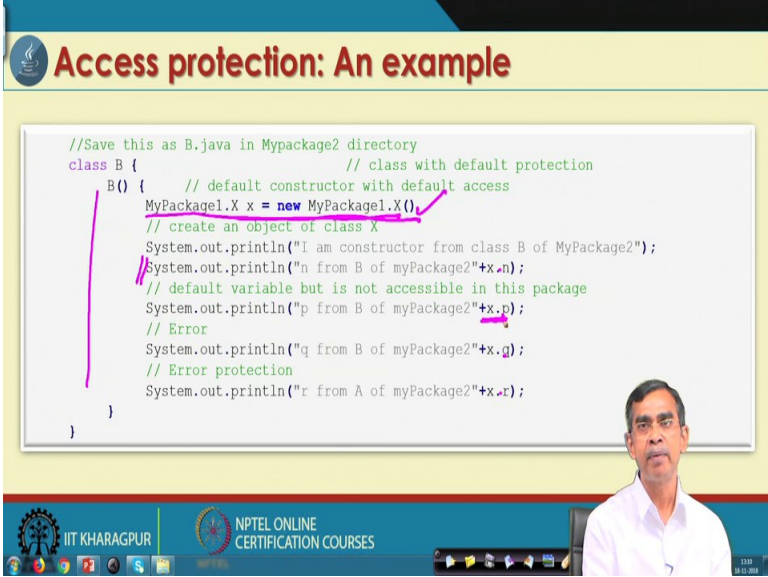
And in the MyPackage 2, as I discussed it has 2 classes the class Z and class B, the class Z is basically is a subclass of class X.

But it is in a different directory. Now let us see what will happen, so for this class Z is concerned. Now, so it is a Z, it does not have any members, so fine only the constructor. .Now, here this is the default accession, default class member cannot be accessed outside any other class from the class which is there. So, if it belongs to the class MyPackage 1 and this class Z belongs to MyPackage 2, it cannot access this default class the that. So, this is why, it should give an error, so it should be commented.

Next is that private. As you know private member cannot be accessible other than the class member itself. So, private p cannot be accessed from the class Z, which is in the other directory or in other packages, so it should be commented as well. Now q is a protected as it is a derived a subclass of x. So, a protected member is accessible by inheritance, so this is correct. Now the public is accessible, so for the Z class, which belong to another package is concerned. So, it can access other than these 2, it can access other all other things are there. I hope you have understood this concept.

Now, this is the Z class, now let us come to the discussion of another class which belongs to the B, it is not a subclass of this one is a derived class, so a class B.

(Refer Slide Time: 30:20)



The slide is titled "Access protection: An example". It displays a Java code snippet for a class `B`. The code is annotated with pink lines and text explaining access protection rules. The code is as follows:

```
//Save this as B.java in MyPackage2 directory
class B { // class with default protection
    B() { // default constructor with default access
        MyPackage1.X x = new MyPackage1.X();
        // create an object of class X
        System.out.println("I am constructor from class B of MyPackage2");
        System.out.println("n from B of myPackage2"+x.n);
        // default variable but is not accessible in this package
        System.out.println("p from B of myPackage2"+x.p);
        // Error
        System.out.println("q from B of myPackage2"+x.q);
        // Error protection
        System.out.println("r from A of myPackage2"+x.r);
    }
}
```

The annotations highlight the following issues:

- `x.n`: Access to a default (package-private) member from another package is not allowed, resulting in an error.
- `x.p`: Access to a private member from another package is not allowed, resulting in an error.
- `x.q`: Access to a protected member from another package is not allowed, resulting in an error.
- `x.r`: Access to a public member from another package is allowed.

The slide also features the IIT Kharagpur logo and the text "NPTEL ONLINE CERTIFICATION COURSES" at the bottom.

.Now, here B again, we do not have any members. So, it is b method is here only, .Now, here. So, B is basically here we MyPackage p 1, I am explicitly specifying that I want to create an object. So, X.X new package I create a X and for this X object, I want to create n p I want to access n p q r, where n is default in the class.

Now, here as it is in the different directory as you know x.n is not accessible. So, the default variable is not accessible in this program, so it basically is an error. So, it should be commented and then here X.p, p is a private member cannot be accessed by any other class belongs to anywhere. So, it is there it is should be right and then q also protected, it cannot be accessed anywhere. So, it should be a commented; however, r is public, so r can be accessed from anywhere.

So, we have discussed the different access modification, if it is placed for some class in one package. And then how the distance classes belong to some other package has the limitation, so for the accessing of the member is concerned. Now then fine we can have a demo of this right.

(Refer Slide Time: 31:43)

Access protection: An example

```
//This is the demo of MyPackage1
package MyPackage1;
public class Demo1{
    public static void main(String args[])
    {
        X x1 = new X();
        Y y1 = new Y();
        A a1 = new A();
    }
}
```

```
//This is the demo of MyPackage2
package MyPackage2;
public class Demo2{
    public static void main(String args[])
    {
        Z z2 = new Z();
        B a2 = new B();
    }
}
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And then the demo is basically you can run this one from the MyPackage one and create the object then, you will see whether these objects are created successfully or not. Whatever the comment I have mentioned here, if we put it then this statement will be correct.

But if you uncomment all the statements, then it will report the errors where the access specification is violated by the rule of the access protection. So, the thing is for your understanding, if you run this program and then you will be able to understand more the things clearly. Now, this is regarding the packages. So, for the MyPackage one concern and one demo program that you can think about it. So, we will discuss the demo in details, when we will discuss it. Now another demo program that belongs to the MyPackage also you can run and you can create it and then you can find it.

So, it will create the 2 things and you can. Now, here I want to mention one more thing is that, so here in this demo program I just a limit the accessing of all the package that those are there, but in that case see suppose I can use import MyPackage 2 and here also import MyPackage 1 then all the things that are there and here also we can include both the things are there.

So, I can create 1 class file 1 demo file, so it is a demo 3, which basically import MyPackage 1 and MyPackage 2 together and then run all the program and then you will see exactly what will happen.

Basically, all these things will be there and then the report or that error will be due at the time of compilation. So, all the class file if you are not able to successfully compile, so you cannot build the package and once we are not able to put the class file prior, I mean after the compilation. So, no way of creating this one if you can successively create it then demo program will successfully run for you. So, this is the thing that you should consider on the way whatever it is there. So, this is all about the packages I want to convey it to you and that is the concept of the packages that you should have in your hold actually.

(Refer Slide Time: 33:51)

? Question to think...

- How multiple inheritance is possible in Java?
- How polymorphism is implementable in Java?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE

Our next discussion is basically inheritance related much more. We have discussed the simple inheritance and we have also mentioned at the time of that discussion that multiple inheritance is not allowed in java. But in some situation, you know multiple inheritance is not avoidable. It is obviously, erroneous or gives a lot of ambiguities while debugging your program, but it is also sometimes not avoidable and it makes the programming easy if we can do it.

Now, the question is that; although technically the multiple inheritances is not possible in java, if it is required can I do that that how the multiple inheritance is possible. So, we will discuss that concept. And then polymorphism is also a very important concept, polymorphism means the same thing, but in different places are there. Now, this multiple

inheritance and polymorphism are the 2 concepts which can be achieved by means of another concept interfaces.

So, in our next module, we will discuss multiple inheritances and polymorphism in the discussion of the interface. So, the interface is our next topics are to be discussed in our next module.

Thanks for your attention.