Packages & accessibility modifiers

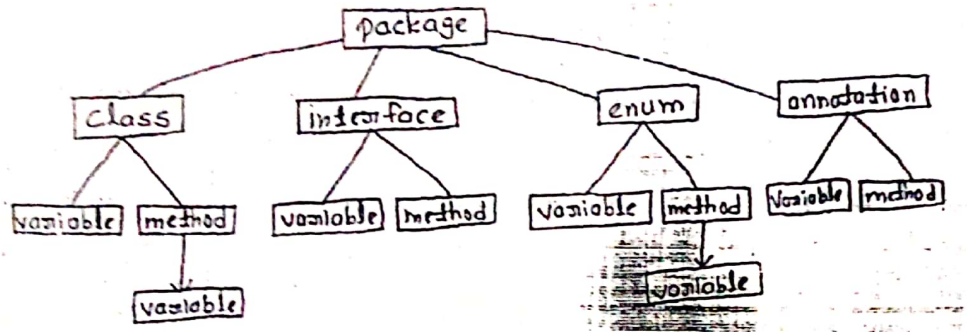
1) What is a package & sub package
2) Creating package & subpackage
3) -d option
4) package & sub package creation as per project standards
5) use of import statement
6) differences between import P1.*;
   import P1.A;
7) importing parent & sub packages
8) Diff ways of setting classpath
9) Accessing

NON-package class member → from non-packaged class member
NON-package class member → from packaged class member

20/09/17 ~~Classes & Types of Classes~~

Java Programming Elements & their purpose

Java supports 7 programming elements, for representing real world object in the programming world, all the 7 programming elements are organized as shown in below diagram:-



From the above diagram we can conclude, we have 7 different concepts we must learn to understand OOP and to create real world object in programming world by storing its values and by implementing its operations.

1. Package and types of packages and their purpose.
2. class and types of classes and their purpose.
3. variable and types of variables and their purpose.
4. block and types of blocks and thier purpose.
5. constraunctor and types of constructor and their purpose.
6. methods and types of methods and their purpose.
7. this keyword and its forms and their purpose.
8. sharing object to other methods, its modification effect

---

9. pass by value and different test cases.
10. setter method, getter methods, mutator, accessor methods.
11. mutable and immutable objects.
12. Factory method, Factory class, factory design pattern.
13. Singleton design pattern class.
14. Reading runtime values from keyboard.
15. OOP block diagram, steps to create real world object in programming world, developing a project to create real world object in programming world using all above OOP concept.

Complete list of Java Programming Elements

For supporting object oriented programming concepts and implementing real world objects in programming world, Java supports 7 programming elements, They are :-

1. package	→ 2	different packages.
2. class	→ 6	different types of class.
3. variable	→ 4	different types of variables.
4. method	→ 2	different types of method.
5. constructor	→ 3	different types of constructor.
6. block	→ 3	different types of block.
7. inner class	→ 4	different types of inner class
	+	
	= 24	

Learning OOP is nothing but learning above 7 programming elements and its subtypes and ~~their purpose and~~ their creation syntax, and their purpose in object creation.

1. Package & Types of packages.

a*. What is package ?

b* Need of package or Why package ?

c*. Types of packages ?

d*. Syntax to create package.

e* Saving, Compiling and executing packaged class.

f*. Structure of a class with package and subpackage ?

g* Naming convention of a pakage as per project standards ? and different standard package names we use in project.

h*. Need of -d option of javac command and importance of a classpath in executing packaged class ?

i*. Different in setting class path in below options

1. Using java command options -cp/-classpath
2. Using cmd command set classpath
3. Using environment variable classpath.

~~import and static~~

* How many ways we can create package?

Java file → (1) Manually (new → folder)
(2) Programmatically (-d)
→ class file

import and static import statements

j * What is import and use of import ?

k * What is the meaning of fully qualified class name and what is the difference between FQCN and import statement.

l * What is difference between
import p1.*; and import p1.example;

m * Compilers algorithm in finding classes ?

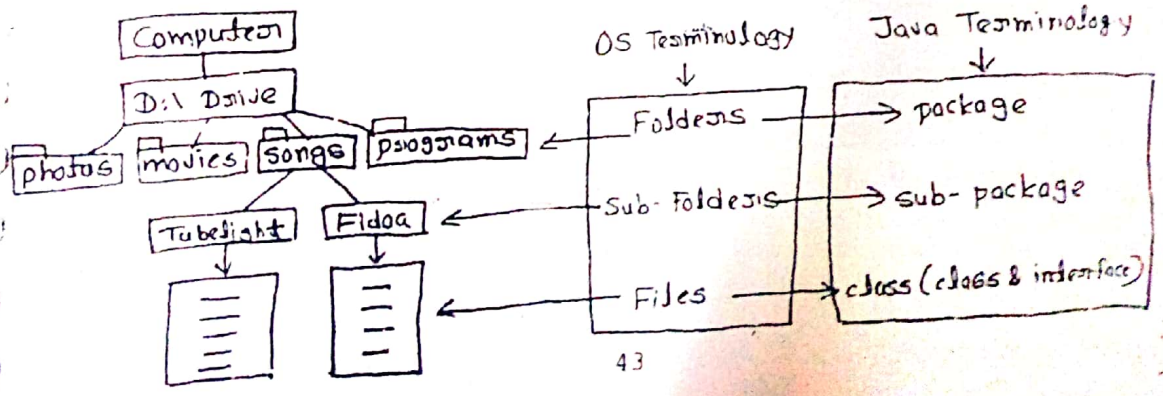
n * Difference in accessing packaged class and subpackaged class ?

o * Interview and OCJP questions on package and import ?

p * Need of Java 5 new feature "static import", its creation syntax, rules, sample programs and FAQs ?

a & b: What is package and why package ?

A package is a folder i.e linked with a class, it is used for organizing same functionality related classes and interfaces as one group and also used for separating one functionality multiple classes from other functionality group of classes.
And also it is used for seperating new classes from existing classes when both have same name.
For Example :- In computer we generally store different type of files as separate groups by using folders and directory concept as shown below. This folder wise organization is nothing but package concept in java and files are nothing but classes and interface in java.



- Like in computers, in projects we will create different types of classes and interfaces based on project functionality
- All these classes and interfaces we must create as seperate groups by using package and sub-package.

For example:-

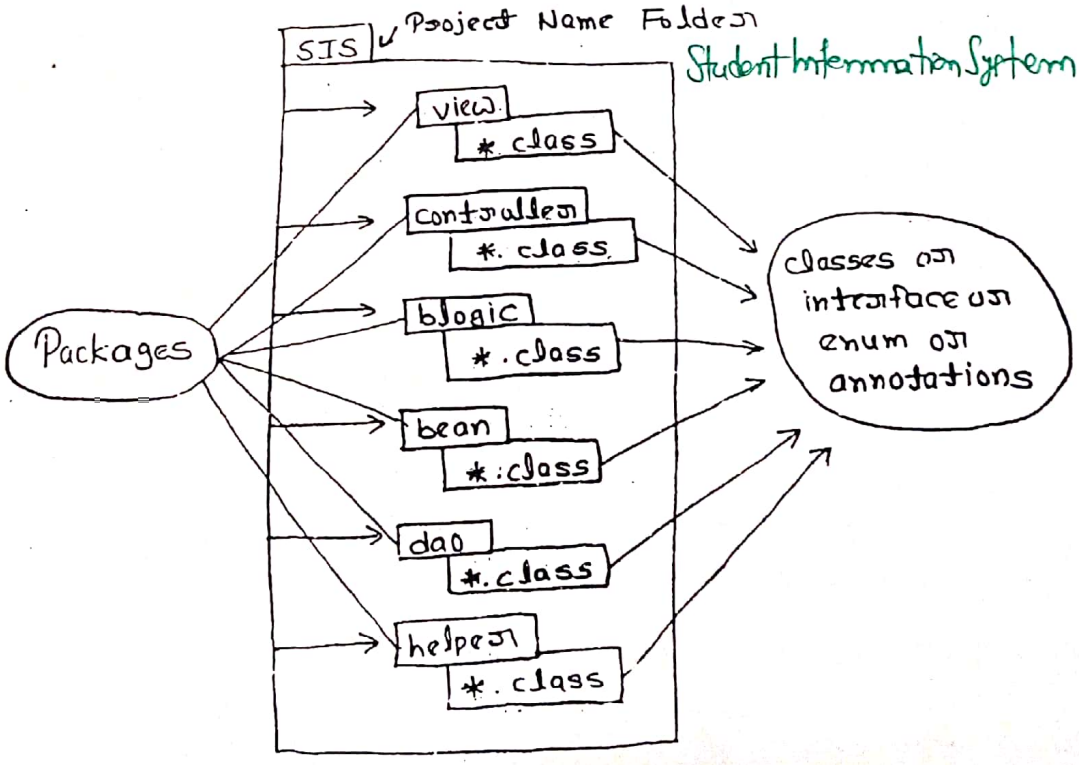
→ We will have DB interaction classes (Model)/(dao)
→ We will have controllers classes (controller)
→ We will have view classes (view)
→ We will data storeing classes (Bean/Domain)
→ We will have reusable/helpers/util classes (helpers)

to store all these classes as seperate groups we will create folders/packages with the same functinality names.

That means we will create packages with the names

model, controllers, view, bean, helpers, blogic,....

↑ package names in projects.



26/08/17

c.) Types of package.

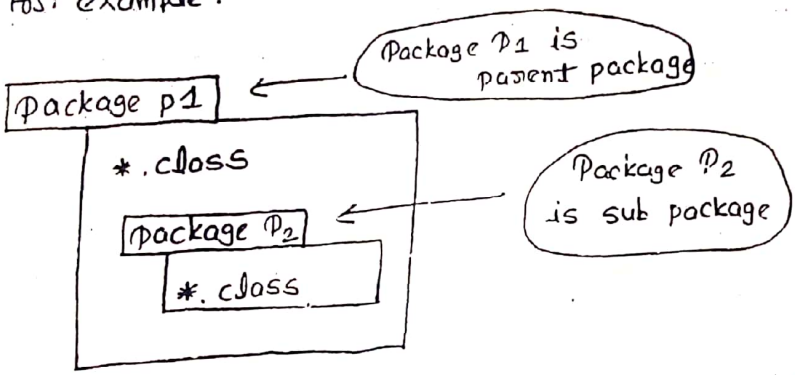
Java supports two types of package creation

1.) Parent package.

2.) Sub package.

The main package is nothing but parent package, A package created inside other package is called as sub-package.

For example :-



Parent package is used for representing main functionality of the project classes.

We will use sub-package for representing sub-functionality of the parent package functionality.

For example, in Java we have

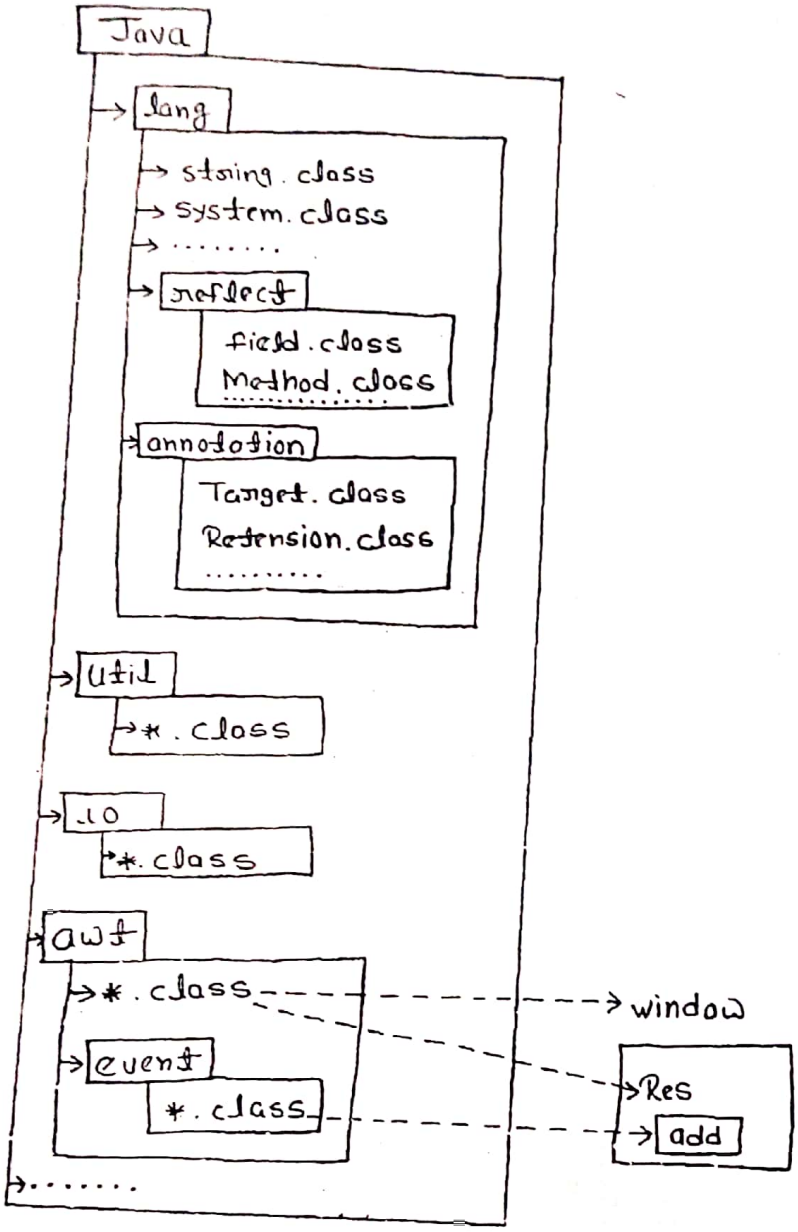
parent package java and

sub package lang, util, io, awt, sql.....

for specifying sub functionalities of Java language.

Note :- Inside sub package we are allowed to create some more sub packages.

For e.g. Inside java.lang subpackage we have subpackages annotation, reflect..... to represent lang level sub-functionality.



awt programming has 2 functionality

(i) Components creation logic

(ii) Components Event handing logic.

To create window and its component like label, textfield, button etc.... we will use awt package classes.

To handle components event and to execute the respective background logic, we will use awt.event sub-package classes.

Note :- According to Ameerpet, we have 2 types of packages

(i.) pre-defined package.

(ii.) user-defined package.

This answer is dirty answer or stupid answer, because these are not types these words are prepared based on who created and when created. The package (parent or sub) which is already created by SUN or by other team member is called pre-defined (parent or sub) package.

For example, java, java.lang, java.io, java.util .... are pre-defined (parent and sub) packages given by SUN as part of Java API.

For example service, service.dao, service.impl are user defined (parent and sub) packages.

The package created newly by our self is called user defined package.

Summary :-

Q. How many types of packages java support ?

- Java supports 2 types of packages : (i.) parent package and (ii.) sub package, the main folder we created for storing the main functionality classes is called parent package. The sub-folder created inside main folder for storing sub-functionality classes is called sub-package.
- The parent package and sub-package can be a predefined package or can be a user defined package.

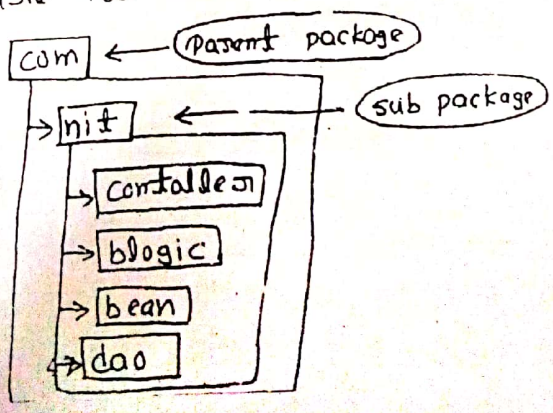
Q. What is the folder structure for the below package statements :-

com.nit.controllers

com.nit.blogic

com.nit.bean

com.nit.dao



d.) Syntax to create package.

- For organizing classes as groups we must create our own packages (user defined packages).
- To create either predefined or user-defined, parent or sub packages in java language we will use a "package" keyword.
- Syntax:-

parent package creation

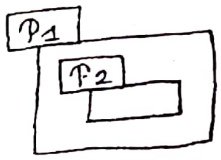
```
package packageName;
```

Example: package p1;
creates a folder with name p1

sub package creation

```
package parentpackageName.subpackageName;
```

Example:- package p1.p2;
Creates two folders with the name p1, p2



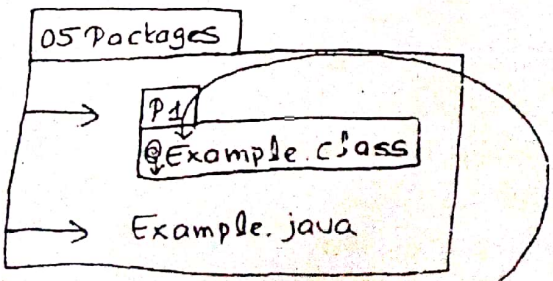
Note:- if parent package p1 is already existed, above statement will create only subpackage p2 inside p1 folder.

e.) Procedure to save, compile and execute package java file class

29/08/17

1. You can save java file in current directory.
2. Open cmd prompt → change path to current working directory.
3. Run javac command as below:-

```
javac -d . Example.java
```



4. Run java command as below
java p1.Example

5. We must develop a java file package with a class as shown below :-

```
//Example.java
package p1;
class Example {
    public static void main (String [] args) {
        System.out.println ("Hello World");
    }
}
```

Save above program with the name "Example.java" inside current working directory "05 packages" folder.

05 Packages
Example.java

Wrong Compilation and Execution

- We should not compile and execute packaged class with a wrong syntax as :-

```
javac Example.java
java Example
```
X Wrong Syntax

- if we compile packaged class only by using javac command package folder is not created, Example.class file is directly saved in current working directory, as shown below

05 packages
Example.java
Example.class

- if we compiled packaged class directly by using javac command, we don't get any error, but if execute packaged class directly by using java command we will get error: "could not find or load main class Example"
- We get this error because inside Example.class file the name of the class is not Example, rather it name is p1.Example.

- At the time of compilation compile software will attach package name to class name. Then to execute packaged class we must use class name as packagename.classname as java p1.Example
- For the above command we will get error again "could not find or load main class p1.Example" because the p1 folder is not available.
- The meaning of java Example command is:
  (i.) search for Example.class file in current working directory.
  (ii.) if Example.class file is available. verify the class name is Example.

  Note:- given name and name of class in .class file must be same.

  (iii.) if the class name is different, JVM will throw error "Could not find or load main class"
  If class name is same as mentioned after java command, JVM will start execution with main method.

- The meaning of p1.Example command is :-
  (i.) Search for package folder with the name p1 in CWD
  (ii.) if p1 is not available throw error "cfoLmc", if p1 is available search for Example.class file in p1 package.
  (iii.) If Example.class file is not available, JVM will throw error "cnflmc"
  If Example.class is available, JVM will verify whether the class name is p1.Example or not.
  (iv.) If class name is different, JVM will throw error "cnfoLmc"
  If class name is same as the name mentioned after java command, JVM will start execution with main method.

- So, to execute a packaged class we must place its class file inside its packaged folder, then we should run the command

```
java packagename.classname
```

For example:-

```
java p1.Example
```

Que:- Will package folder created automatically by using the keyword "package" ?

Ans:- No, package keyword will not create package folder automatically at the time of compilation. We must run javac command with its option "-d" as shown below:-

```
javac -d . Example.java
```

Summary:- developing, saving, compiling and executing packaged class and its structure

1. // Example.java

```
package p1
class Example{
    p s v main(String[] args){
        Sopln("H,W!");
    }
}
```

2. 05 packages
   - Example.java
   - p1
     - Example.class

save.

3. 05package>javac -d . Example.java
4. 05package> java p1.Example
5. O/p: H,W!

30/08/17

h.) Need of -d option with javac command.

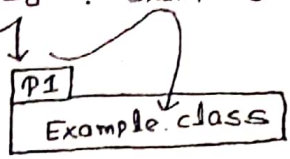
-d option will creates folder for the package statement with the given name and it will move the generated .class file automatically into this package folder

```
package p1;
class Example{

}
```

case#1: javac Example.java
↳ Example.class
here package folder is not created.

case #2: javac -d . Example.java



here with -d option package folder is created with the given name p1, and then Example.class file is moved into p1 folder.

Here . represent the folder ~~where or must start~~ path where this package folder must be saved along with its .class file

here . means CWD, so the package p1 folder is saved in CWD along its .class file.

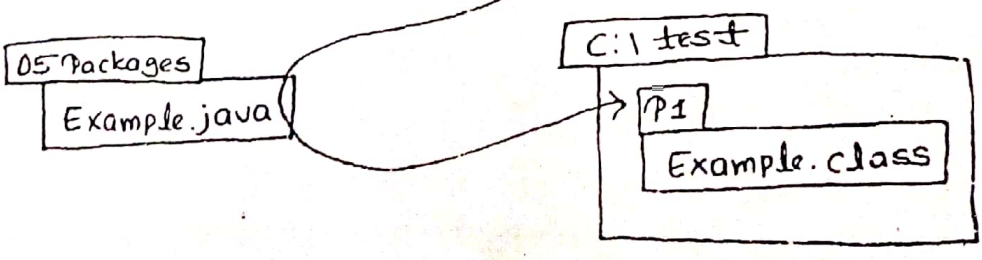
Syntax to copy package into different folder :-

different folder :- C:\test

step 1: Create the folder test in C drive.

step 2: ~~with~~ in the command prompt, in 05Package folder path run below command

javac -d C:\test Example.java



Executing above .class from current directory 05package

05package>java p1.Example

Error: Could not find or load

When we place package folder in different directory, We can not execute this package class from current working directory. JVM throw error, because JVM will search this package p1 only in CWD (05 package), it can't search in other directory.

Solution :- To run classes from different folder we must set classpath by storing package's parent directory path.

This means to run p1.Example class from current working directory 05package, we must set classpath by storing C:\test, as shown below :-

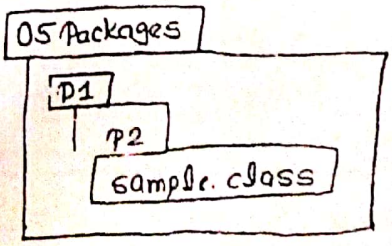
```
05Package > javac -d C:\test Example.java
05 Package > java p1.Example
Error: Could not find or load main class p1.Example
05Package > set classpath = C:\test
05 package > java p1.Example
Hello, World !
```

Note :- In classpath, we should not include package name, if we include package name, than JVM search for package inside package, it is not available it will throw error.

Que :- Develop a class with sub-package, compile and execute by saving in current directory and by saving in different directory.

Ans :- ⇒ Open Edit plus
⇒ write below code :-

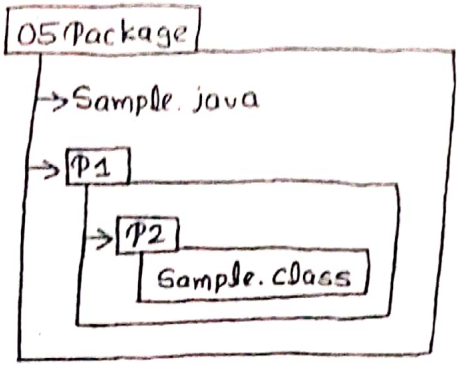
```
package p1.p2;
class sample{
    public static v m(SI args){
        S.o.p.ln("Helloworld");
    }
}
```



⇒ Save with the name Sample.java in CWD.

⇒ Compile and Execute by saving package in CWD.

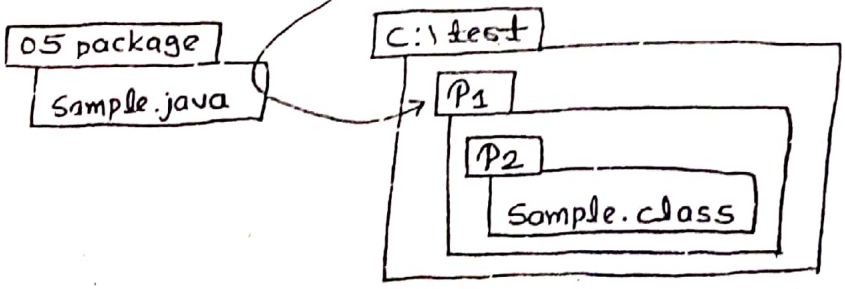
05package > javac -d . Sample.java




```
05 package > java p1.p2.Sample
Hello World
```

⇒ Compile and Execute by saving package in different directory (C:\test)

```
05 package > javac -d C:\test Sample.java
```




```
05 package > set classpath = C:\test
05 package > java p1.p2.Sample
Output :- Hello World.
```

Note:- If parent package folder already existed, only sub-package folder and its .class file are created directly inside this parent package folder

Q. What will happen for the below commands execution:-

X 05 package > javac -d C:\test Example.java

31/08/17

1.) Different ways of setting classpath and difference between them.

* Java 9 onwards --class-path

We can set classpath in 4 ways :-

(1.) By using java command option -cp

(2.) By using java command option -classpath

(3.) By using temporary setting in command prompt
set classpath=

(4.) By using permanent setting in environment variable window.

If we set classpath by using -cp/-classpath, this classpath setting value is available only for this cmd line.
If we run java program in the next cmd line, we will get Error: c f o l m c.
If we want to maintain this classpath setup value for next cmd lines we must set classpath using set classpath option.

If we set classpath by using set classpath= option, this classpath setup value is available for all cmd lines, but only in this cmd prompt window.
If we close this cmd window, classpath setup values lost.

If we set classpath by in Environment variable window, this classpath setup values are available permanently to all cmd windows, in all cmd lines even after system restart.

Usage:-

```
05Package>java p1.Example
Error: could not find or load main class p1.Example
```

```
05 package>java -cp C:\test p1.Example
Hello World! from the class p1.Example
```

← Not available for next cmd line.

```
05package > java p1.Example
Error: c n f o l m c p1.cla.Example
```

```
05package > set classpath = C:\test
05pakcage > java p1.Example
Hello World! from the class p1.Example
05package > java p1.Example
Hello World! from the class p1.Example.
```

Need of . in classpath :- The . represents CWD path, By Using . ~~compiler~~ a placed in classpath, compiler and JVM will find, compile and execute classes from CWD.

If we donot place . in classpath, the classes placed in CWD are not found, compiled and executed by compiler and JVM.

> Note:- If we do not create classpath variable in our system, the default value of classpath is . means CWD path.
> Then compiler and JVM automatically find, compile and execute classes from CWD.

If we explicitly create classpath, we must place . in classpath value, else classes in CWD are not executed.

Refer page no. 221 (volume-1B) for more details on classpath and . in classpath execution cases.

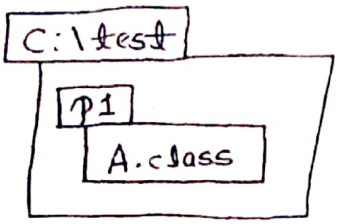
Role of ; in classpath :- The character ; will act as

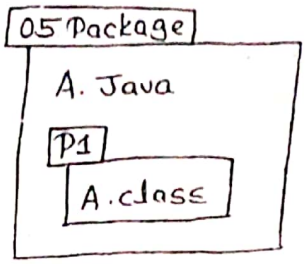
(i) Seperator when we use in between two paths.

(ii) . (CWD) when we use at end of classpath value or when we use multiple time as ;;

E.g. set classpath = C:\test; D:\abc
set classpath = C:\test;
set classpath = C:\test;; D:\abc
set classpath = Hari;; Krishna

Create a class called A in package p1, compile and save in C:\test and also in CWD. Display message loaded from C:\test when you compile and save in C:\test folder and Display message loaded from CWD when you compile and save in CWD.
With this program test all above classpath cases





01|09|17

Shocking point on "-d" option :-

We can use -d option for compiling normal java files, it means we can use -d option to compile a java file even though it doesn't have package statement.

The basic need of -d option is saving .classfile in the given folders path.

If the java file contains package statement,
-d option will create package folders, generate .class file in this package folders and then saves package folders in the given directory path.

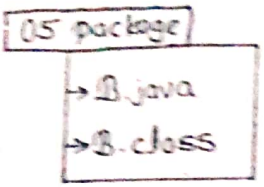
If the java file doesn't contains package statement,
-d option will save .class file directly inside the given folders path.

For example,

```
// B.java
class B {
    public s v m(string[] args){
        System.out.println("Hi");
    }
}
```

```
05 package > javac -d . B.java
   B.class is directly saved in CWD
05 package > javac -d C:\test B.java
   B class is saved in C:\test folder
```

Case #1 :-




```
05package > javac -d . B.java
05package > java B
Hi (.class file is loaded from and
        executed from CWD)
```

Case #2 :-


```
05package > java -d C:\test B.java
05package > java B
Error: could not find or load main class
```

If we save .class file in different directory, then to execute this class, we must set classpath.

Here, B.class file is saved in other directory, so, classpath is mandatory.

```
05package > set classpath=C:\test
05package > java B
Hi (.class file is loaded and executed
        from C:\test)
```

Que:- Ameerpet style of creating package is wrong, Why?

- In Ameerpet, while learning packages chapter we will save all java files inside one directory, and then later we will compile these java files by using -d option. then, Now package folders are created, .class files are generated and saved inside these package folders.
- Saving java files inside project folder directly leads to two problems:-
  (i) All hundreds of different-functionality java files are saved in single folder. So that finding a particular functionality java file to modify is very tough.
  (ii) And moreover we can not create 2 classes java files with a same name.
- Solutions to the Above two problems is:-
  We must create package folder not only for .class files but also for .java files.

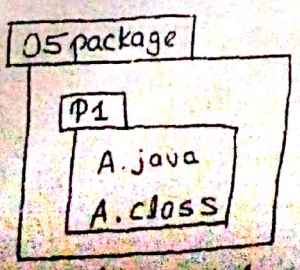
Project Style of package Creation

We must create package folder manually for java files, programatically for .class files

Procedure:-

1. Create a folder with packagename p1.
2. Save source file in this package folder.
3. then compile these java file without using -d option.
4. then .class file is directly created inside package folder (p1)

05pakage>javac p1\A.java



5. 05 package > java p1.A

In this approach also, we have a problem, that is separating .class file from .java is very tough for sending project executable file to client.

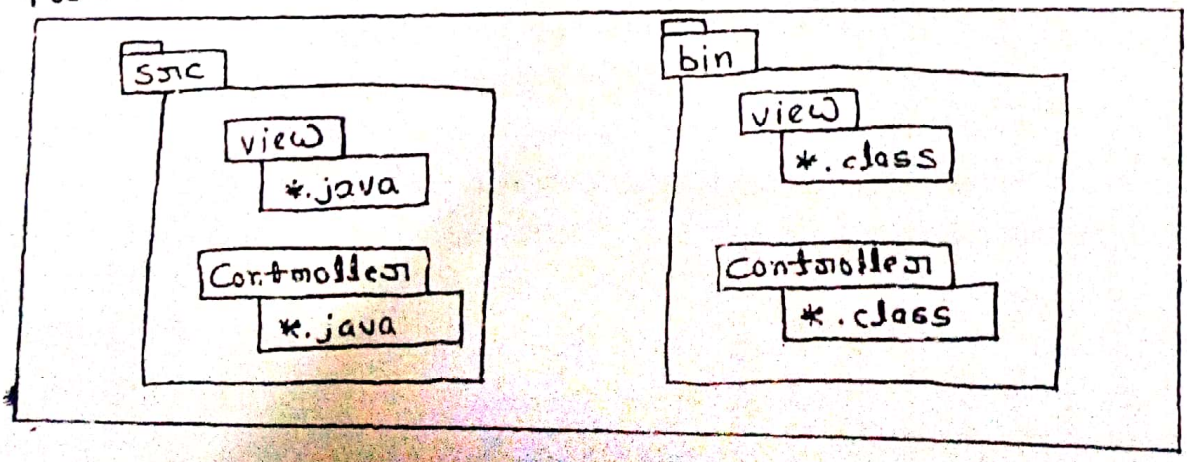
So the right approach is we must save java files and class files with package folders in seperate directories as shown below

Procedure:-

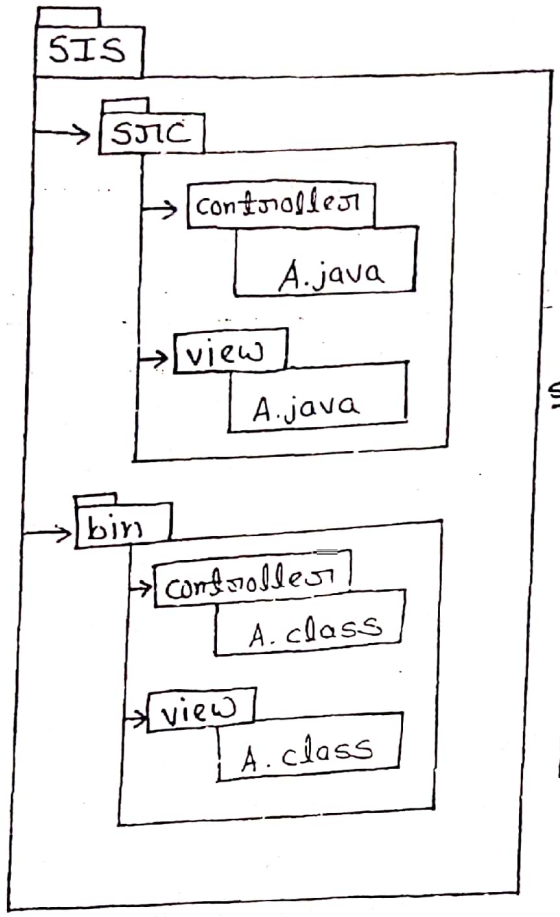
1. Create project folder with two ~~seperate~~ sub-directories "src" and "bin"
2. Inside src folder create package folders with its java files.
3. ~~Compile~~ Compile the java file with -d option by given bin folder path.
4. Then -d option will create package folders inside bin directory with the .class files generated.
5. Then change directory path to bin folder and execute class files
   (or) send bin folder all packages with its class file to client.

This is the right procedure of creating packaged java files and class files.

Not only we are following, eclipse also follows same procedure if we use eclipse IDE for java file creation, it automates this packages creation for java file and class files seperately in src and bin folders.



Example :-



SIS> javac -d .\bin src\view\A.java

SIS>javac -d .\bin src\controllers\A.java

SIS > cd bin

bin> java view.A
↳Load view's class

bin>java controllers.A
↳Load Controllers's class

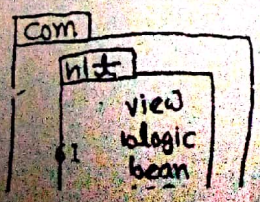
02/09/17

g.) Naming convention of a package as per projects standards.

1. Inside package name all characters should be small letters. And package name should be as short as possible.
2. Package name should represent a group name generally; functionality name like view, blogic, dao.....
3. In project we will create package name with as below

Company web domain reverse. functionality

E.g. com.nit.view
com.nit.blogic
com.nit.bean
com.nit.dao



Package statement Rule :-

In a single java file only one package statement is allowed also the pakage statement must be first statement in the java file.

Example.java

```
class Example{}
package p1;
```

✗ CE : Package statement must be first statement in java file.

Example.java

```
package com.nit.view;
package com.nit.blogic;
package com.nit.bean;
class Example{
}
```

→ ✗ Compile time Error

Reason:- A class must exhibit only one functionality.

Accessing One package class from another packaged class :-

A.java

```
Package p1;
public class A{
(R1) public void m1(){
(R2)    S.O.pl.n ("Hi");
    }
}
```

Test.java

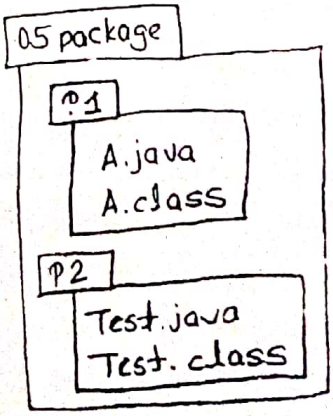
```
Package p2;
class Test{
  P.S.V.main(S[] args){
    p1.A a1 = new p1.A();
(R3) a1.m1();
  }
}
```

05 package > javac p1\A.java

05 package > javac p2\Test.java

05 package > java p2.Test

Op: Hi



If you want to access one package class from another package class, we must follow below three rules:—

R1:- Class should be public.

R2:- The method or variable we want to access should be public.

R3:- In another class, we must access this class by using fully qualified name. Means we must access class with its package name as p1.A.

Run above program with the below 4 cases:-

Case 1:- Call m1() method directly, without class A reference, as shown below:—

```
m1(); X CE: not find symbol m1
```

If we access method directory without reference then compiler searches m1() method definition in same class i.e. Test class. It can not search in class A. If you want to search m1() in class A we must invoke m1() with its class A object reference. As shown in above program.

Case 2:- Remove package name in accessing class A. means replace `p1.A a1 = new p1.A();` as `A a1 = new A(); X CE: can not find symbol class A`

If we place class name directly, compiler searches class A definition in current method, in current class, in current package folder. But not in another packages.
If you want to access class from another package we must access it using package name as shown in above program.

Case 3 & Case 4 :-

Remove public to class A and method m1() then we will get compile time error, "class A is not public", "method m1() is not public".

If we want to access a class and or its members from different package, it should be public. As shown in above program.

Introduction to import statements

1. Import is a package concept related keyword.
2. When we want to access a class from one package to another package, we must use import statement in the accessing class java file. For example in previous program we must use import statement in Test class to access class A from the package p1 to package p2.

Test.java

```
package p2;
import p1.*;
class Test {
    p.s v. main (string[] args) {
    A a1 = new A();
    a1.m1();
    }
}
```

3. import statement must be placed after package statement and before class declaration.
4. import statement will not import (c and p) classes from one package to another package, rather import statement will just give info to compiler in which package the class is available.

04/09/17

Compiler Adding changes to our class with respect to packages and import statement

```
package p2;
import p1.*;
class Test {
  P S V m(S[] args){
   A a1 = new A();
   Sopln (a1.x);
}
```

```
class p2.Test{
  P S V main(java.long S[] args){
    p1.A a1 = new p1.A();
    java.long.System o p ln (a1.x);
  }
}
```

1. If we create a class with a package and import statements, at the time of compilation compiler software will remove package and import statements and attaches package name to class name and then finally save the modified compiled byte code inside .class file. as shown above ↑
2. Compiler will attach package statement package name to class name and constructor name.
3. Compiler will attach import statement to the classes names those are accessing from this package or this imported package

For more details, read bytecode by using javap command.

Syntax :- `javap -verbose classname`

Example :- 05packge>javap -verbose p2.Test

Que:- What is the name of a package class ?

An :- Fully qualified name it means package_name.class_name

For example :- p1.A, p2.test,
java.long.String
java.lang.System
java.sql.connection

Example :— If we need to pass classname as input to another program, we must pass name with fully qualified name. as shown below:-

Enter class name: A ✗ java.lang.CNFE

Enter class name: p1.A ✓

1. Difference between import p1.* and import p1.A

- When we need to access class A from p1 package to p2 package, we can use either import p1.* or import p1.A
- There are 3 differences between these two statement :-

import p1.*	import p1.A
1. Using p1.* we can access all public classes available in the package p1.	1. Using p1.A, we can access only class A in the package p1.
2. p1.* has less priority, it means if class A available in both p1 and p2 package, class A is loaded and executed from current package p2.	2. p1.A has First priority, it means class A is loaded and executed from package p1 even though class A is available in current package p2.
3. With p1.* we can create class A in current java file.	3. With p1.A we can not create class A in current java file.
4. With p1.* compiler doesn't search for .class/.java file in package p1, until we use class name.	4. With p1.A, compiler will immediately search A.class/ A.java file in package p1 if not available, it will throw error.

For programming point on above table, Refer page No. 222 & 223 in volume 1(B)

06/09/17

m. Compiler algorithm in finding classes

import p1.*	import p1.A;
1. Searching class A {} in current method main method.	1. Searching class A {} in current method i.e. main method.
2. Searching class A {} in current class Test{} body.	2. searching class A {} in current class Test {} body.
3. Searching class A {}, in current java file Test.java	3. Searching class A {} in current java file Test.java If class A {} is available compiler will throw error for import p1.A statement
4. Searching A class {}, in current java file package p2.	4. Searching A class {} file in imported package p1.
5. Searching A.java file in current current java file package p2.	5. Searching A.java file in imported package p1.
6. Searching A class {} file in imported package p1.	
7. Search in A.java file in imported package p1.	

if here also not available, then compiler ~~with~~ throw error could not find symbol class A

Here the error thrown for the statement
A a1 = new A();

Here the error thrown for the statement
import p1.A;

Note:- If we remove import p1.A; from this java file, compiler will search A.class then A.java files in current package p2.

Auto Compilation

In a project, we no need to compile every class java file. When we compile main method class java file all classes java files those are accessing from this main method class are automatically compiled by compiler software.

The phenomina or algorithm, compiling internally using classes java file automatically by compiler software, is called auto compilation.

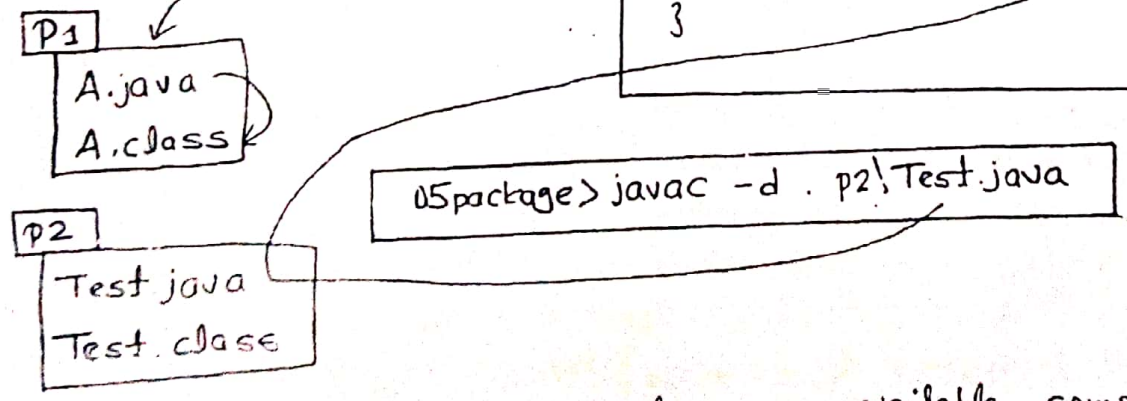
For example. Observe below program

A.java

```
package p1;
public class A {
    public void m1(){
        SOPln("p1.A.m1");
    }
}
```

Test.java

```
package p2;
import p1.*;
class Test {
    p.s.v main(S[] args){
        A a1 = new A();
        a1.m1();
    }
}
```



If both .class and .java files are available, compiler will the time stamp of .java and .class file.

If .java file time stamp > .class time stamp
then recompile

else
Compiler will directly uses existing .class file.

Autocompilation algorithm all steps:- [Page No. 222]

When we are using a class from another class, No need to compile that class first, Compiler automatically compiles that class.

For example assume we are calling Example class method from sample class method we can compile sample class directly without compiling Example class.

Compiler follows below procedure to compile Example class:-

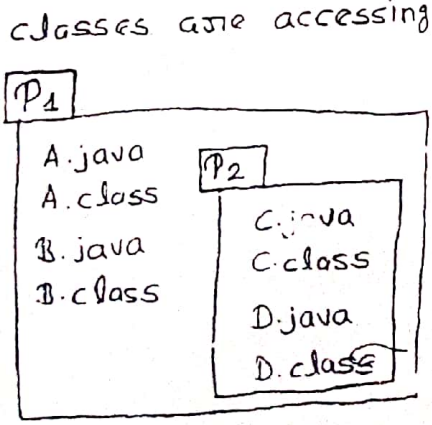
1. First it searches that Example class definition in Sample .java, if not found
2. It searches for Example.class in Sample class package, if not found
3. It seaches for Example.java in Sample class package, if not found
4. It searches for Example.class in imported packages, if not found
5. It searches for Example.java in imported packages, if not found
6. Then compiler terminate Sample.java file compilation by throwing CE: cannot find symbol.
7. If Example.java is found, it searches for ~~exap~~ Example class definition in Example.java file. If it is found, compiler compile entire java file, it means it also compile other class definitions and generate those class's .class files. Else terminates Sample.java file compilation with above compilation error.
8. If Example.class is found, it also searches for Example.java. If not found, compiler uses Example .class file directly.

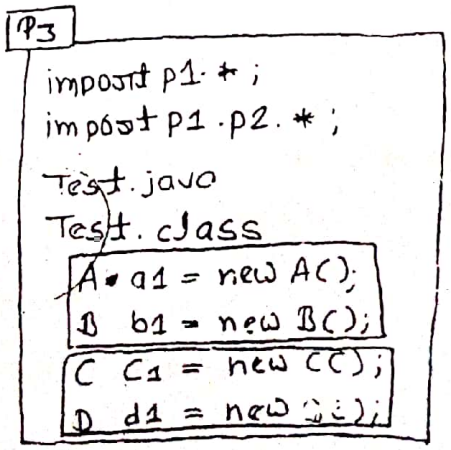
9. If Example.java is also available, it checks modified time of both files, if Example.java file modified date is greater than Example.class modified date, it compiles Example.java again for generating Example.class with its latest changed java code.

Test all above cases/points using below two programs.
[Refer Volume - 1B, Page No. 222 to 223] 09/09/17

Importing Subpackage and accessing sub-package classes from our own class from different package

- For accessing classes from sub-package we must place import statement for subpackage classes.
- If we write only parent package import statements, we can access only the classes those are available in parent package.
- By adding sub-package import statement, we can not access classes from parent package, we can access classes only from subpackage.
- If we want to access classes from both, parent package and sub-package, we must add import statement for both parent package and sub-package.
- Consider below diagram and identify from which package classes are accessing from which import statement :-





70

A.java

```
package p1;
public class A {
    p.s.v.m(s[] args)
}
```

B.java

```
package p1;
public class B {
}
```

C.java

```
package p1.p2;
public class C {
}
```

D.java

```
package p1.p2;
public class D {
}
```

Test.java

```
package p3;
import p1.*;
import p1.p2.*;
class Test {
    p.s.v.m(s[] args){
        A a1 = new A();
        B b1 = new B();
        C c1 = new C();
        D d1 = new D();
    }
}
```

- In above project we should not write main method in every class, its not compile time error or run time error, It is a wrong design, because the classes A, B, C, D must be executed from class Test
- So we should not define main method in these 4 classes.
- We should define all 4 classes public, then only we can access them from Test class

- To compile and execute above project classes, we need to compile and execute Test.java file, As per Auto Compilation algorithm the classes A,B,C,D are Automatically Compiled.

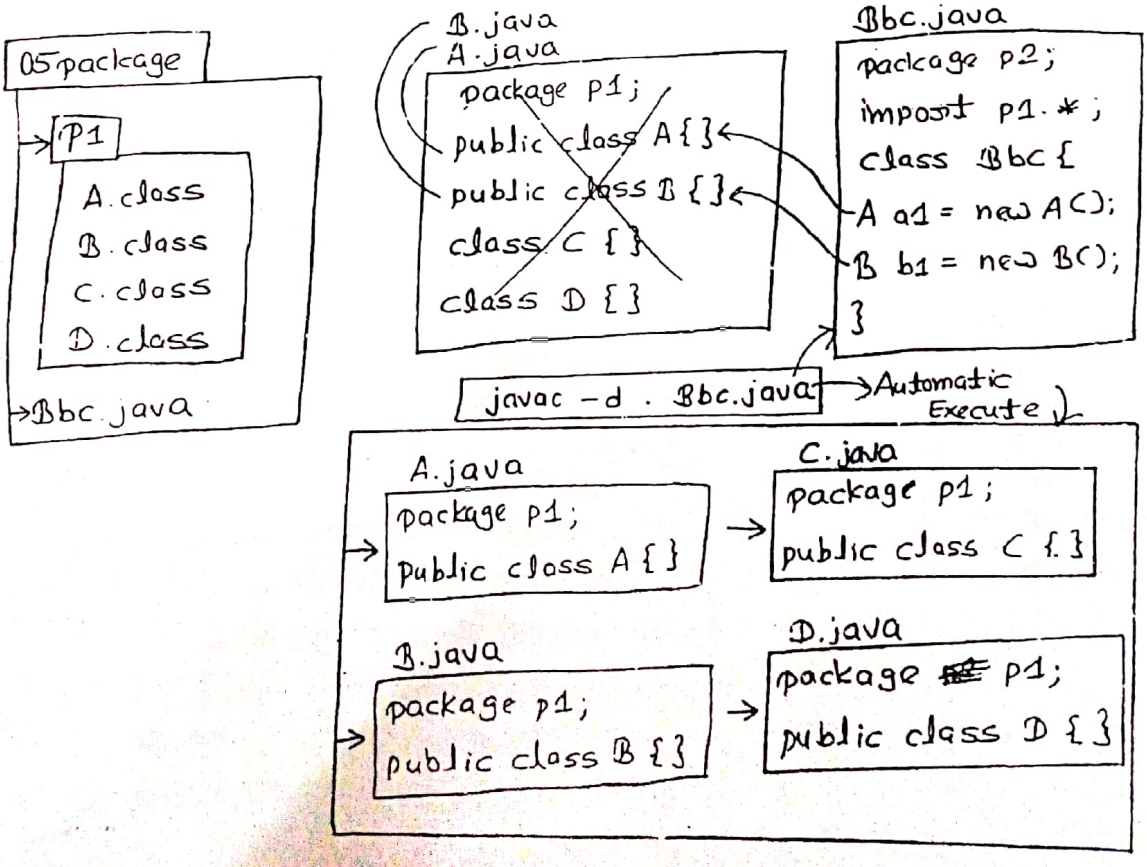
Use below commands to compile and execute test class.

```
05package> javac p3\Test.java
05package> java p3.Test
```

Q. How can we place multiple classes in single package folder ?

Ans:- We have 2 approaches to place multiple classes in single package

1. Declare all classes in single java file
2. Declare Every class in separate java file with the same package statements.



- If we save all classes in single java file, all class files are created in single package but the problem in this approach is we can not access these classes in another package classes, because these classes are not public.
- To access these classes in another packaged classes, we should declare classes as public.
- If we declare class as public, classname and java-file name must be same, then in a single java file we can not declare multiple public classes. Hence it is not a good approach creating multiple classes in single java file.
- Hence to reuse a class, it means to access a class from different package classes and also to achieve auto-compilation we must create each class in seperate java file by declaring class as public. If we want to place .class files of all these in a single package we must place same package name statment in all java files.

> Rule:— To get auto compilation of these java files we must save java file also inside single package.

Q. Why public class name and Java file name must be same and why non public class name and Java file name no need to be same ?

⇒ For 2 reasons, public class name and java file name should be same:-

1. For easy Identifying in which java file the class is defined either for reading or modifying source code.
2. Also for achieving auto-compilation.

These 2 rules are not applicable for non-public class, because non-public classes are accessible to some programmers who defined them, so he/she knows in which java file class is defined, hence non-public class name and java file name no need to be same.

- Either class is public or non public we must create java filename and class name must be same and more over we must create one class per java file for autocompilation.

Que:- What is the output from the below program ?

```
public class A {
    public void m1(){
        sopln("Hi");
    }
}
```

```
package p1;
class B {
    p s v main(s[] args){
        A a1 = new A();
        a1.m1();
    }
}
```

Ans:- Compilation Error: could not find symbol class A.

Different Combinations to access one class from another class.

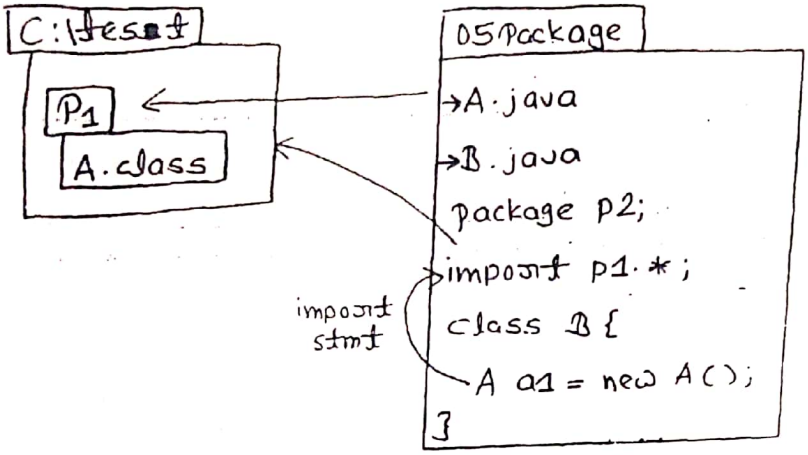
Non Packaged class —X→ Packaged class

Non Packaged class —✓→ Non packaged class

Packaged class —✓→ packaged class

Packaged class —✓→ Non packaged class

Q. From Compiler point of view, what is the difference between import statement and classpath ?
What is the output of below program :-




```
05package > javac -d C:\test A.java
05 package > javac -d C:\test B.java
```

Ans:- 1. Compiler uses class path to find package and ofcourse also uses for finding classes.

2. Compiler uses import statement to find classes those we are accessing in our class.

12/09/17

Q. How can we access parent package classes in the same parent package other classes and parent package classes in its sub-package classes and sub-package classes in other sub-package classes and in parent package classes ?

Ans:- If we want to access a class from other classes of same package we no need to use import statement. we can access the class directly. So parent package classes we can access directly by their name inside other classes in the same parent package.

For Example:-

A.java

```
Package p1;
public class A {
}
```

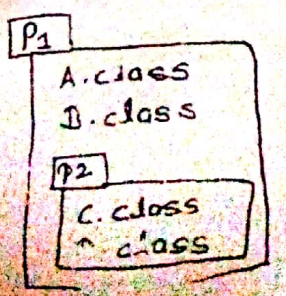
B.java

```
package p1;
Public class B {
    A a1 = new A();
}
```

← Import statement is not used

If we want to access a class from other package classes this class should be public and we must use import statement of this class package in another java file in which we want to access.

Hence if we want to access parent package classes in subpackage classes or if we want to access sub-package classes in other sub-package classes or in its parent package classes we must import statement of its package and also the class should be public.



A.java

```
package p1;
import p1.p2.*;
class A {
    C c1 = new C();
}
```

D.java

```
package p1.p2;
import p1.*;
class D {
    B b1 = new B();
}
```

B.java

```
package p1;
public class B {

}
```

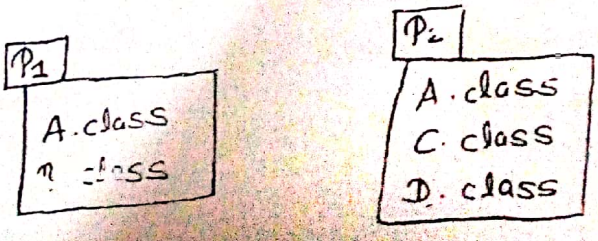
C.java

```
package p1.p2;
import public class C {

}
```

Q. We can access classes of one package from another package in two ways using fully qualified name and by using import statement. Then when to use import statement and when to use FQN ?

Ans:- We will use FQN only if there is any name conflict- that means if a class is defined in more than one imported packages we must use FQN (Fully qualified Name) to differentiate this class from the imported packages. Otherwise, means if there is no name conflict, we always access classes by using Import statement. Consider below Example, we access class by using FQN and other classes by using Import statement.



P3

```
//Test.java
package p3;
import p1.*;
import p2.*;
class Test {
    psvm main(s[]args){
    //A a1 = new A();  ---> CE: Ambiguous
      B b1 = new B();
      C c1 = new C();
      D d1 = new D();
      p1.A a1 = new p1.A();
      p2.A a2 = new p2.A();
      a1.m1();
      a2.m1();
    }
}
```

Pending Topics :-

1. Static import
2. Working with jar files

~~Refer next chapter~~.

Refer volume 1B