DAY-52

------

ACCESS\_MODIFIERS

---------------------------------

These are the keywords in object oriented language that set the accessibility of classes,methods and other members.

In Java having a keyword before the class,method or variable is considered as 'modifiers'.

NOTE:In java there is not concept of access\_specifiers.

Access\_specifiers are available in c,c++ languages.In java Acess\_modifiers are available.

ACCESS\_MODIFIERS IN JAVA:

-------------------------

1. public

2. protected

3. default

4. private

5. final

6. abstract

7. static

8. synchronized

9. native

10. strictfp

11. transient

12. volatile

ACCESS\_MODIFERIS availble for user-difiend classes:

---------------------------------------------------

1. public

2. default

3. final

4. abstract

5. strictfp

ACCESS\_MODIFERIS availble for inner-classes:

---------------------------------------------------

1. public

2. default

3. final

4. abstract

5. strictfp

6. private

7. protected

8. static

PUBLIC/DEFAULT access\_modifers w.r.t classes:

----------------------------------------------

--> If a UDC is declared as 'public' then we can access that class from anywhere.

EXAMPLE:

--------

package pack1;

public class A

{

public void fun1()

{

System.out.println("inside pack1");

}

}

package pack2;

import pack1.A;

class B

{

public static void main(String[] args)

{

A a = new A();

a.fun1();

System.out.println("inside pack2 of class B");

}

}

OUTPUT:

-------

inside pack1

inside pack2 of class B

NOTE: If we want to use methods of a class in another package then we must make the UDC as 'public'.

if UDC is public in nature then members of that class can be accessed within the same package and also from differnt packages.

EXAMPLE:

--------

class Demo

{

public void fun1()

{

System.out.println("inside class Demo");

}

}

class Demo1

{

public static void main(String[] args)

{

Demo d = new Demo();

d.fun1();

System.out.println("inside class Demo1");

}

}

OUTPUT:

-------

inside class Demo

inside class Demo1

NOTE: If we want to use methods of a class in the same package then we must make the UDC as 'default'.

if UDC is default in nature then members of that class can be accessed within the same package and can not be accessed from differnt packages.

FINAL modifiers w.r.t UDC: [refer day-39 notes]

------------------------------------------------

FINAL KEYWORD:

--------------

This keyword can be used or applied on :

1. on class

2. on method

3. on variable

final keyword on class:

------------------------

If a class is made as final then that class will not participate in inheritance.

example:

----------

final class A

{

void fun1()

{

System.out.println("INSIDE PARENT CLASS");

}

}

class B extends A

{

void fun1()

{

System.out.println("INSIDE child CLASS");

}

public static void main(String[] args)

{

A a = new A();

a.fun1();

B b = new B();

b.fun1();

}

}

final keyword on method:

------------------------

The methods which are made as final can be inherited to the child class but can not be overridden.

example:

----------

class A

{

final void fun1()

{

System.out.println("INSIDE PARENT CLASS");

}

}

class B extends A

{

void fun1()

{

System.out.println("INSIDE child CLASS");

}

public static void main(String[] args)

{

//A a = new A();

//a.fun1();

B b = new B();

b.fun1();

//b.fun1();

}

}

final keyword on variable:

------------------------

If the variable is made as final then the value of the variable can not be alterd in other words it will become static in nature.

example:

----------

class A

{

final int speed = 300;

int fun1()

{

speed = 250;

//System.out.println(spped);

return speed;

}

}

class B

{

public static void main(String[] args)

{

A a = new A();

System.out.println(a.fun1());

}

}

ABSTRACT [non-access\_modifiers] w.r.t. methods :

------------------------------------------------

abstract modifier is only applicable for classes and methods and it is not applicable for variables

(refer class notes(abstract concept))

NOTE: If any method present in the class is abstract in nature then the class should be prefixed with abstract keyword.

NOTE: Illegal combinations with abstarct keyword:

1. final

2. synchronized[multithreading]

3. private

4. native

5. static

6. strictfp

strictfp [strict floating point] ACCESS\_MODIFIER:

--------------------------------------------------

--> This modifier is applicable for classes and methods and not applicable for variables.

eg: 10/3 --> 3.33333

the result w.r.t floating point is always machine dependent

ie, based on 16 bit 32 bit 64bit we will get different floating point.

To strandarize the result and to achive platform independent strictfp is used w.r.t floating point calculations.

strictfp w.r.t methods:

------------------------

--> If strictfp is used on methods then all the floating points in the methods should fallow "IEE754" format. To achive standarization and platform

independence

strictfp w.r.t classes:

------------------------

--> If a UDC is declared as strictfp then every floating point within the class is calculated for concrete methods as per IEE754 format.

Note: abstract and strictfp is a illegal combination

EXAMPLE:

--------

abstract strictfp class Demo2

{

abstract strictfp void fun1()

{

}

public void fun100()

{

}

abstract void method1();

abstract void method2();

}

OUTPUT:

-------

illegal combination

METHOD LEVEL AND VARIABLE LEVEL ACCESS MODIFIERS:

--------------------------------------------------

1. public methods and variables:

--> public members can be accessed from any where but, its correspondingg class should public.

2. default members:

--> default members can be accessed only within the current package.

3. private members:

--> if a members is private we can use only within the current UDC.

4. protected members:

--> can be accessed within the current package but, only in child classes of outside package.

EXAMPLE:

--------

package p1;

public class sample

{

protected void fun1()

{

System.out.println("inisde pack p1 and class sample and method fun1");

}

}

package p2;

import p1.sample;

class sample2 extends sample

{

public static void main(String[] args)

{

//sample s = new sample();

//s.fun1();

//sample s = new sample2();

//s.fun1();

sample2 s2 = new sample2();

s2.fun1();

System.out.println("inside sample2 and pack p2 ");

}

}

OUTPUT:

-------

inisde pack p1 and class sample and method fun1

inside sample2 and pack p2

[REFER DIA for visibility of modidfiers]