DAY-46

-------

refer fig:1

--> INTERFACE is a specification.

refer fig:2

--> Interface promotes standarization using which we can achive polymorphism.

Interface is collection of abstract methods using interface 100% abstraction can be achived becuase inside the interface methods declaration are allowed

SYNTAX:

-------

interface interface\_Name

{

abstract method(); --> method

Constants; --> variables(final)

}

ADVANTAGE OF INTERFACE:

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

1. Using the interface standarization is achived.

eg: pi value

2. It promotes polymorphism

3. 100% abstraction can be achived

4. multiple inheritance can be achived.

NOTE: By default all the data members(variable) present in the interface are 'public static final' in nature.

Also by default all the methods present in the interface is 'public abstract' in nature.

EXAMPLE:1

---------

interface Test

{

float pi = 3.14f;

void fun();

}

compile:

interface Test

{

public static final float pi = 3.14f;

public abstract void fun();

}

RULES OF INTERFACE:

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

RULE:1

--> When a class is implementing the inteface the methods access modifiers should be made as public beacause by default all the abstract method in

interface are public in nature.

EXAMPLE

-------

refer diagram:

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

interface Calculate

{

/\* public abstract \*/void add();

}

class Calculator implements Calculate

{

public void add()

{

int a,b,c;

a=10;

b=20;

c=a+b;

System.out.println(c);

}

}

class Demo

{

public static void main(String[] args)

{

Calculator c = new Calculator();

c.add();

}

}

/\*

public

protected weaker access modifiers can be given

default

private

\*/

OUTPUT:

--------

30

RULE:2

-------

-->If a class implements the interface then it should provide the body for all the methods present in the interface class.

If the class is not in a position to provide the body for all the method then class should be made as anstract.

EXAMPLE:1

----------

interface Calculate

{

void add();

void mul();

}

class Calculator implements Calculate

{

public void add()

{

int a,b,c;

a=10;

b=20;

c=a+b;

System.out.println(c);

}

}

class Demo

{

public static void main(String[] args)

{

Calculator c = new Calculator();

c.add();

}

}

OUTPUT:

------

Calculator is not abstract and does not override abstract method mul() in Calculate

class Calculator implements Calculate

note:

-----

The above probelm can be overcome by making class as bastract and method where class is not in a position to provide the body then it should be made

as abstract.

EXAMPLE:2

----------

interface Calculate

{

void add();

void mul();

}

abstract class Calculator implements Calculate

{

public void add()

{

int a,b,c;

a=10;

b=20;

c=a+b;

System.out.println(c);

}

abstract public void mul();

}

class Demo

{

public static void main(String[] args)

{

//Calculator c = new Calculator();

//c.add();

}

}

RULE:3

-------

An object of abstract class can not be crearted

ex: refer above program

An object of inteface cannot be created

EXAMPLE:

-------

interface Calculate

{

void add();

void mul();

}

class Demo

{

p s v m (---)

{

Calculate c = new Calculate();

}

}

OUTPUT:

-------

ERROR

--> Object can not be instantiated.

RULE:4

------

Though we cannot create the object for interface its refernce can be created.

EXAMPLE:

--------

interface Calculate

{

void add();

void mul();

}

class Demo

{

p s v m (---)

{

//Calculate c = new Calculate();

Calculate c; --> parent refernce

}

}

RULE:5

------

A class and interface cannot be realted using the extends keyword

EXAMPLE:

--------

interface Calculate

{

void add();

}

class Calculator extends Calculate --> use implements instead of extends

{

public void add()

{

int a,b,c;

a=10;

b=20;

c=a+b;

System.out.println(c);

}

}

class Demo

{

public static void main(String[] args)

{

Calculator c = new Calculator();

c.add();

}

}

OUTPUT:

-------

ERROR

RULE:6

------

Using the inteface ref (parent ref) the implemented methods of a class can be accessed and using interface ref polymorphism can be achived

EXAMPLE:

--------

interface Bank

{

void ROI();

}

class SBI implements Bank

{

public void ROI()

{

System.out.println("sbi --> 7.5%");

}

}

class AXIS implements Bank

{

public void ROI()

{

System.out.println("AXIS --> 9.5%");

}

}

class CANARA implements Bank

{

public void ROI()

{

System.out.println("CANARA --> 8.5%");

}

}

class Calculate

{

public void allow(Bank ref)

{

ref.ROI();

}

}

class Demo1

{

public static void main(String[] args)

{

SBI s = new SBI();

AXIS a = new AXIS();

CANARA c = new CANARA();

Calculate c1= new Calculate();

c1.allow(s);

c1.allow(a);

c1.allow(c);

}

}

OUTPUT:

-------

sbi --> 7.5%

AXIS --> 9.5%

CANARA --> 8.5%

RULE 7:

-------

An interface can be implemented by any number of classes.

refer above program.