DAY 47

------

refer dia:1

RULE 8:

-------

Using the interface ref only the overridden method can be accessed in order to access the speacilized methods downcasting has to be perormed.

refer dia:2

EXAMPLE:

--------

interface Calculate

{

void add();

void sub();

}

class Calculator implements Calculate

{

public void add()

{

int a,b,c;

a=10;

b=20;

c=a+b;

System.out.println(c);

}

public void sub()

{

int a,b,c;

a=20;

b=05;

c=a-b;

System.out.println(c);

}

public void mul()

{

int a,b,c;

a=10;

b=20;

c=a\*b;

System.out.println(c);

}

}

class Rule8

{

public static void main(String[] args)

{

Calculate c;

Calculator cal = new Calculator();

c = cal;

c.add();

c.sub();

//c.mul(); -->

((Calculator)(c)).mul(); // --> downcasting

}

}

OUTPUT:

-------

30

15

200

RULE 9:

-------

A class can be implemented any number of interface

refer dia:3

EXAMPLE:

--------

interface calculate1

{

void add();

}

interface calculate2

{

void sub();

}

class calculator implements calculate1,calculate2

{

public void add()

{

int a,b,c;

a=10;

b=20;

c=a+b;

System.out.println(c);

}

public void sub()

{

int a,b,c;

a=20;

b=05;

c=a-b;

System.out.println(c);

}

}

class Rule9

{

public static void main(String[] args)

{

calculator c = new calculator();

c.add();

c.sub();

}

}

OUTPUT:

-------

30

15

RULE 10:

--------

If two interface contains same method signature and same return type then the implementation class can provide the body for only one method.

refer dia:3

EXAMPLE:

--------

interface calculate1

{

void add();

}

interface calculate2

{

void add();

}

class calculator implements calculate1,calculate2

{

public void add()

{

int a,b,c;

a=10;

b=20;

c=a+b;

System.out.println(c);

}

}

class Rule9

{

public static void main(String[] args)

{

calculator c = new calculator();

c.add();

}

}

OUTPUT:

-------

30

RULE 11:

--------

If two interface conatins same method name and return type and change in parameter then the implementaion class should provide the body for all

the methods.

EXAMPLE:

--------

interface calculate1

{

void add();

}

interface calculate2

{

void add(int a, int b);

}

class calculator implements calculate1,calculate2

{

public void add()

{

int a,b,c;

a=10;

b=20;

c=a+b;

System.out.println(c);

}

public void add(int a, int b)

{

int c;

c=a+b;

System.out.println(c);

}

}

class Rule11

{

public static void main(String[] args)

{

calculator c = new calculator();

c.add();

c.add(10,20);

}

}

OUTPUT:

-------

30

30

RULE 12:

--------

In interface if the method signature is same and the returntype is different then the implementing class can not provide the body.

If implementing class provide the body results in ambiguity problem.

refer dia:4

EXAMPLE:

--------

interface calculate1

{

void add(int x, int y);

}

interface calculate2

{

void add(int a, int b);

}

class calculator implements calculate1,calculate2

{

public void add(int x, int y)

{

int c;

c=x+y;

System.out.println(c);

}

public int add(int a, int b)

{

int c;

c=a+b;

return c;

}

}

class Rule12

{

public static void main(String[] args)

{

calculator c = new calculator();

System.out.println(c.add(5,5));

//c.add(10,20);

}

}

OUTPUT:

-------

error

RULE 13:

--------

If the implementing class is not able to provide the body for all the methods of interface then the implementing class should be made as abstract.

The child class of the implementaing class can provide the body for the remaining methods of interface.

refer dia:5

EXAMPLE:

--------

interface calculate1

{

void add();

void sub();

}

interface calculate2

{

void mul();

}

abstract class calculator implements calculate1,calculate2

{

public void add()

{

int a,b,c;

a=10;

b=20;

c=a+b;

System.out.println(c);

}

public abstract void sub();

public abstract void mul();

}

class juniorCalculator extends calculator

{

public void sub()

{

int a,b,c;

a=20;

b=10;

c=a-b;

System.out.println(c);

}

public void mul()

{

int a,b,c;

a=10;

b=10;

c=a\*b;

System.out.println(c);

}

}

class Rule13

{

public static void main(String[] args)

{

juniorCalculator jc = new juniorCalculator();

jc.add();

jc.sub();

jc.mul();

}

}

OUTPUT:

-------

30

10

100

AMBIGUITY PROBLEMS WITH INTERFACE VARIABLES:

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

refer dia:6

EXAMPLE:

-------

interface calculate1

{

int min = 10;

}

interface calculate2

{

int min = 20;

}

class Test implements calculate1,calculate2

{

public void disp()

{

System.out.println(min);

}

}

class Ambiguity

{

public static void main(String[] args)

{

Test t = new Test();

t.disp();

}

}

OUTPUT:

--------

error: reference to min is ambiguous

System.out.println(min);

^

both variable min in calculate1 and variable min in calculate2 match

NOTE: The above problem can be overcome by accessing the inteface variable with the interface name.

EXAMPLE:

--------

interface calculate1

{

int min = 10;

}

interface calculate2

{

int min = 20;

}

class Test implements calculate1,calculate2

{

public void disp()

{

System.out.println(calculate1.min);

System.out.println(calculate2.min);

}

}

class Ambiguity

{

public static void main(String[] args)

{

Test t = new Test();

t.disp();

}

}

OUTPUT:

-------

10

20

RULE:14

-------

An interface can extend any number of interface.

refer dia: 7

EXAMPLE:

--------

interface calculate1

{

void add();

}

interface calculate2

{

void sub();

}

interface calculate3 extends calculate1,calculate2

{

void mul();

}

class Test implements calculate3

{

public void add()

{

int a,b,c;

a=10;

b=20;

c=a+b;

System.out.println(c);

}

public void sub()

{

int a,b,c;

a=20;

b=05;

c=a-b;

System.out.println(c);

}

public void mul()

{

int a,b,c;

a=20;

b=05;

c=a\*b;

System.out.println(c);

}

}

class Rule14

{

public static void main(String[] args)

{

Test t = new Test();

t.add();

t.sub();

t.mul();

}

}

OUTPUT:

-------

30

15

100