DAY-39

--------

WHAT IS RUNTIME POLYMORPHISM AND VIRTUAL POLYMORPHISM ?

--> METHOD OVERLOADING --> VIRTUAL POLYMORPHISM --> STATIC POLYMORPHISM --> EARLY BINDING.

--> METHOD OVERRIDING --> RUNTIME POLYMORPHISM --> DYNAMIC POLYMORPHISM --> LATE BINDING.

// VIRTUAL POLYMORPHISM USING THE METHOD OVERLOADING.

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

class Plane

{

String name;

int wheel;

void takeOff()

{

System.out.println("plane is taking off");

}

void fly()

{

System.out.println("plane is flying");

}

void land()

{

System.out.println("plane is landing");

}

}

class CargoPlane extends Plane

{

void takeOff()

{

System.out.println("Cargoplane is taking off");

}

void fly()

{

System.out.println("Cargoplane is flying with the low speed");

}

void CarryCargo()

{

System.out.println("Cargoplane is carrying the goods");

}

}

class PassengerPlane extends Plane

{

void fly()

{

System.out.println("Passengerplane is flying with the medium speed");

}

void CarryPassenger()

{

System.out.println("Passengerplane is carrying the people");

}

}

class FighterPlane extends Plane

{

void fly()

{

System.out.println("Fighterlane is flying with the high speed");

}

void CarryArms()

{

System.out.println("Fighterplane is carrying the weapons");

}

}

class Airport

{

void allow(CargoPlane ref)

{

ref.takeOff();

ref.fly();

ref.land();

}

void allow(PassengerPlane ref)

{

ref.takeOff();

ref.fly();

ref.land();

}

void allow(FighterPlane ref)

{

ref.takeOff();

ref.fly();

ref.land();

}

}

class Demo

{

public static void main(String[] args)

{

CargoPlane cp = new CargoPlane();

PassengerPlane pp = new PassengerPlane();

FighterPlane fp = new FighterPlane();

Airport a = new Airport();

a.allow(cp);

a.allow(pp);

a.allow(fp);

}

}

// RUNTIME POLYMORPHISM USING THE METHOD OVERRIDING.

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

class Plane

{

String name;

int wheel;

void takeOff()

{

System.out.println("plane is taking off");

}

void fly()

{

System.out.println("plane is flying");

}

void land()

{

System.out.println("plane is landing");

}

}

class CargoPlane extends Plane

{

void takeOff()

{

System.out.println("Cargoplane is taking off");

}

void fly()

{

System.out.println("Cargoplane is flying with the low speed");

}

void CarryCargo()

{

System.out.println("Cargoplane is carrying the goods");

}

}

class PassengerPlane extends Plane

{

void fly()

{

System.out.println("Passengerplane is flying with the medium speed");

}

void CarryPassenger()

{

System.out.println("Passengerplane is carrying the people");

}

}

class FighterPlane extends Plane

{

void fly()

{

System.out.println("Fighterlane is flying with the high speed");

}

void CarryArms()

{

System.out.println("Fighterplane is carrying the weapons");

}

}

class Airport

{

void allow(Plane ref)

{

ref.takeOff();

ref.fly();

ref.land();

}

}

class Demo1

{

public static void main(String[] args)

{

CargoPlane cp = new CargoPlane();

PassengerPlane pp = new PassengerPlane();

FighterPlane fp = new FighterPlane();

Airport a = new Airport();

a.allow(cp);

a.allow(pp);

a.allow(fp);

}

}

RULES OF METHOD OVERRIDING

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

RULE 1:

-------

In method overriding the method signature should be same. IE The method name and argument type should be same in both parent and child class.

eg:

---

public void test(int a)

{

---------

-----------

-----

}

--> test(int a) // method signature

RULE 2:

--------

eg: w.r.t 1.4 version of java

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

class A

{

void fun1()

{

----

----

}

}

class B extends A

{

void fun1()

{

-----

}

}

eg: w.r.t 1.5 version of java

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

class A

{

A fun1()

{

System.out.println("inside the parent class");

return null;

}

}

class B extends A

{

B fun1()

{

System.out.println("inside the child class");

return null;

}

public static void main(String[] args)

{

A a = new A();

a.fun1();

B b = new B();

b.fun1();

}

}

--> Till 1.4 version the return type of both the methods in parent and child class should be same.

From the 1.5 version of java co-variant return types are allowed.

ie, return type of both the method in parent and chid class need not to be same.

example:

---------

class A

{

Object fun1()

{

System.out.println("inside the parent class");

return null;

}

}

class B extends A

{

String fun1()

{

System.out.println("inside the child class");

return null;

}

public static void main(String[] args)

{

A a = new A();

a.fun1();

B b = new B();

b.fun1();

}

}

example:

---------

class A

{

double fun1()

{

double a= 22.5;

System.out.println(a);

return a;

}

}

class B extends A

{

float fun1()

{

int b=29.5f;

System.out.println(b);

return b;

}

public static void main(String[] args)

{

A a = new A();

a.fun1();

B b = new B();

b.fun1();

}

}

output:

----------

compilation error

RULE:3

-------

In method overriding the child class overrinding method can not have a weaker access modifier when compared with overriden method of the parent class.

REFER THE DIAGRAM

example:

----------

class A

{

protected void fun1()

{

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

}

}

class B extends A

{

public 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();

}

}

NOTE: Private members will not participate in inheritance and hence private methods can not be overriden.

example:

---------

class A

{

private 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();

}

}

NOTE: In the above program fun1() method of the child class is not a overriden method it is a independent instance method of child class.

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());

}

}

RULE 4:

-------

METHOD OVERRIDING W.R.T STATIC METHODS:

non-static to static method overriding and static to non-static method overriding is not permited in java.

example:

-------

class A

{

void fun1()

{

System.out.println("inside the parent class");

}

}

class B extends A

{

static void fun1()

{

System.out.println("inside the child class");

}

public static void main(String[] args)

{

//A a = new A();

//a.fun1();

B b = new B();

b.fun1();

}

}

NOTE : STATIC memebers can not be overriding.

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