DAY-40

-------

METHOD HIDING:

// EXAMPLE FOR METHOD OVERRIDING

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

class Parent

{

void fun1()

{

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

}

}

class child extends Parent

{

void fun1()

{

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

}

}

class Demo

{

public static void main(String[] args)

{

Parent p = new Parent();

p.fun1();

child c = new child();

c.fun1();

p = new child(); // upcasting

p.fun1();

}

}

OUTPUT:

--------

inside the parent method

inside the child method

inside the child method

// EXAMPLE FOR METHOD HIDING

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

class Parent

{

static void fun1()

{

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

}

}

class child extends Parent

{

static void fun1()

{

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

}

}

class Demo

{

public static void main(String[] args)

{

Parent p = new Parent();

p.fun1();

child c = new child();

c.fun1();

p = new child(); // upcasting

p.fun1();

}

}

OUTPUT:

--------

inside the parent method

inside the child method

inside the parent method

METHOD HIDING : It is a concept in java where both methods in the parent and child class are static and with the same signature.

During the method hiding the method call is resolved based on reference.

DIFFERENCE B/W METHOD OVERRIDING AND METHOD HIDING

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

METHOD OVERRIDING:

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

1.Both the methods present in the parent and child class should be instance(non-static) in nature.

2.method call gets resolved by jvm based on object creation.

3.it is also called as RUNTIME POLYMORPHISM --> DYNAMIC POLYMORPHISM --> LATE BINDING.

METHOD HIDING:

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

1.Both the methods present in the parent and child class should be static in nature.

2.method call gets resolved by compiler based on refernce.

3.it is also called as VIRTUAL POLYMORPHISM --> STATIC POLYMORPHISM --> EARLY BINDING.

DIFFERENCE B/W METHOD OVERRIDING AND METHOD OVERLOADING

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

METHOD OVERRIDING:

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

1.method overriding refers to the method which is inherited by the child class from the parent class and modified it.

2.method signature should be same

3.return type can be different from 1.5 version onwards but return type of child class method should be the child of parent class method return type

4.access modifiers privelage can not be reduced

5.actions performed based on object creation

METHOD OVERLOADING:

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

1.METHOD overloading refers to multiple methods within a same class with the name.

2.method signature need not be same.

3.return type can be anything

4.access modifiers can be anything.

5.actions performed based on reference

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

RELATIONSHIPS IN JAVA

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

As a part of java application development we have to use entities as per the requirement

relationships are used to provide the optimization over memory utilization,code reusablity,execution time,ease of shareablity.

TYPES OF RELATIONSHIPS IN JAVA:

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

1.IS-A relationship

2.HAS-A relationship

IS-A : by using this we can achive the concpet of inheritance by doing so we can perform 'code reusablity'.

HAS-A : It defines associations b/w the entites to improve the communication b/w the entites for better data navigation.

NOTE: has-A relationships is of two types:

1. COMPOSITION

2. AGGREGATION

WHAT IS THE DIFFERENCE B/W COMPOSITION AND AGGREGATION ?

-->Aggregation referes to weak relationship whereas compostion refers to strong relationship.

arggregation will have less dependency with enclosed object where as compostion will have more dependency with enclosed object

aggregation members can be accessed when with out enclosed object

compostion members can not be accessed without enclosed object

aggregate object is independent in nature

compostion objects are dependent in nature

EXAMPLE:

--------

class os

{

String name;

int size;

public os(String name, int size)

{

this.name=name;

this.size=size;

}

public String getName()

{

return name;

}

public int getSize()

{

return size;

}

}

class charger

{

String brand;

String color;

public charger(String brand, String color)

{

this.brand=brand;

this.color=color;

}

public String getBrand()

{

return brand;

}

public String getColor()

{

return color;

}

}

class mobile

{

os o = new os("android",125);

public void hasA(charger c)

{

System.out.println(c.getBrand());

System.out.println(c.getColor());

}

}

class Demo1

{

public static void main(String[] args)

{

mobile m = new mobile();

charger c1 = new charger("samsung","white");

//System.out.println(m.o.getName());

//System.out.println(m.o.getSize());

m.hasA(c1);

m=null;

//System.out.println(o.getName());

//System.out.println(o.getSize());

System.out.println(c1.getBrand());

System.out.println(c1.getColor());

}

}

OUTPUT:

-------

samsung

white

samsung

white