DAY 3 :

package day3;

class Employeee

{

int empid;

String name;

float sal;

public Employeee() {

System.***out***.println("default constructor");

}

public Employeee(int empid,String name,float sal)

{

this.empid=empid;

this.name=name;

this.sal=sal;

}

void display()

{

System.***out***.println(empid);

System.***out***.println(name);

System.***out***.println(sal);

}

}

class Developer extends Employeee{

public Developer()

{

System.***out***.println("developer obj is created");

}

void calpay()

{

this.sal=this.sal+1500;

}

}

public class task1 {

public static void main(String args[])

{

Developer dev = new Developer();

dev.sal=5000;

dev.calpay();

System.***out***.println(dev.sal);

}

}

The above code when the obj is created , the obj first execute the default constructor in the parent class and then execute the child class constructor however when the method is call from the same obj of the dev class the associated method from its own class(dev) is executed .

**OP:**

default constructor

developer obj is created

6500.0

Over loading

package day3;

class Employeee

{

int empid;

String name;

float sal;

public Employeee()

{

this.empid=empid;

this.name=name;

this.sal=sal;

System.***out***.println("employee constructor");

}

void display()

{

System.***out***.println(empid);

System.***out***.println(name);

System.***out***.println(sal);

}

void calpay()

{

this.sal=this.sal+1500;

}

void fun()

{

System.***out***.println("this is parent class function ");

}

void fun(int a)

{

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

}

}

class Developer extends Employeee{

public Developer()

{

System.***out***.println("developer obj is created");

}

void calpay()

{

this.sal=this.sal+1500;

}

}

public class task1 {

public static void main(String args[])

{

Developer dev = new Developer();

dev.sal=5000;

dev.calpay();

System.***out***.println(dev.sal);

dev.fun(); // the child class dontn't have the fun method so it go to the parent class method and exe that method.

dev.fun(12); // overloading methos in the employee class

}

}

OP:

this is parent class function

12

* CONDITION FOR OVERRIDING:  
  > min two class

> classes involved should in inheritance

> the child class should have at least one method with implementation .

> the methos have the same signature of the parent class

**Condition for overloading**

* At least Two method with the same name should be define with in the same class
* Those methos either

1. Argument length is change
2. Argument datatype is change

package day3;

class Employe

{

int empid;

String name;

float sal;

public Employe()

{

System.***out***.println("employee constructor");

}

public Employe(int empid , String name, float sal)

{

this.empid=empid;

this.name=name;

this.sal=sal;

}

}

class Developer extends Employe{

public Developer()

{

System.***out***.println("developer obj is created");

}

void calpay()

{

this.sal=this.sal+1500;

}

}

public class task2 {

public static void main(String args[])

{

Employe emp = new Employe(12,"hari",12000);

Developer dev = new Developer(13,"haran",13000);

}

}

OP:

The constructor Developer(int, String, int) is undefined

package day3;

class Employe

{

int empid;

String name;

float sal;

public Employe()

{

System.***out***.println("employee constructor");

}

public Employe(int empid , String name, float sal)

{

this.empid=empid;

this.name=name;

this.sal=sal;

System.***out***.println("emp parameter");

}

}

class Develop extends Employe{

public Develop(int empid,String name,float sal)

{

this.empid=empid;

this.name=name;

this.sal=sal;

System.***out***.println("developer constructor");

}

void calpay()

{

this.sal=this.sal+1500;

}

}

public class task2 {

public static void main(String args[])

{

Employe emp = new Employe(12,"hari",12000);

Develop dev = new Develop(13,"haran",13000);

}

}

package day3;

class Employe

{

int empid;

String name;

float sal;

public Employe()

{

System.***out***.println("employee constructor");

}

public Employe(int empid , String name, float sal)

{

this.empid=empid;

this.name=name;

this.sal=sal;

System.***out***.println("emp parameter");

}

}

class Develop extends Employe{

public Develop(int empid,String name,float sal)

{

System.***out***.println("developer constructor");

}

void calpay()

{

this.sal=this.sal+1500;

}

}

public class task2 {

public static void main(String args[])

{

Employe emp = new Employe(12,"hari",12000);

Develop dev = new Develop(13,"haran",13000);

System.***out***.println(dev.name);

}

}

**OP**

The dev name is NULL because the priority is in the child class first . the name data is wont be set.

**Super class constructor**

package day3;

class Employe

{

int empid;

String name;

float sal;

public Employe()

{

System.***out***.println("employee constructor");

}

public Employe(int empid , String name, float sal)

{

this.empid=empid;

this.name=name;

this.sal=sal;

System.***out***.println("emp parameter");

}

}

class Develop extends Employe{

public Develop(int empid,String name,float sal)

{

super(empid,name,sal);

System.***out***.println("developer constructor");

}

void calpay()

{

this.sal=this.sal+1500;

}

}

public class task2 {

public static void main(String args[])

{

Employe emp = new Employe(12,"hari",12000);

Develop dev = new Develop(13,"haran",13000);

System.***out***.println(dev.name);

}

}

package day3;

class Employe

{

int empid;

String name,company="info";

float sal;

public Employe()

{

System.***out***.println("employee constructor");

}

public Employe(int empid , String name, float sal)

{

this.empid=empid;

this.name=name;

this.sal=sal;

System.***out***.println("emp parameter");

}

void display()

{

System.***out***.println(this.empid);

System.***out***.println(this.name);

System.***out***.println(this.sal);

}

}

class Develop extends Employe{

String lang = "string";

String des="developer"+super.company;

public Develop(int empid,String name,float sal)

{

super(empid,name,sal);

this.lang=lang;

System.***out***.println("developer constructor");

}

void calpay()

{

this.sal=this.sal+1500;

}

void display()

{

super.display();

System.***out***.println(des);

}

}

public class task2 {

public static void main(String args[])

{

Employe emp = new Employe(12,"hari",12000);

Develop dev = new Develop(13,"haran",13000);

System.***out***.println(dev.name);

emp.display();

dev.display();

}

}

TYPE CASTING

* UPPER CASTING (CHILD-PARENT)
* DOWN CASTING(PARENT-CHILD)
* package day3;
* class third
* {
* void fun()
* {
* System.***out***.println("third fun");
* }
* void fun3()
* {
* System.***out***.println("third fun3");
* }
* }
* class second extends third
* {
* void fun()
* {
* System.***out***.println("second fun");
* }
* void fun2()
* {
* System.***out***.println("second fun2");
* }
* }
* public class task4 {
* public static void main(String args[])
* {
* third th = new second();
* second s = new second();
* s.fun();
* s.fun2();
* s.fun3();
* th.fun();
* th.fun3();
* //th.fun2(); cannot acces by the third class because the third(parent class) cannot access the methos of the child class
* second s1= (second)th; // downcast
* s1.fun3();
* s.fun();



* }
* }

Abstract Class :  
package day3;

abstract class Emp

{

int id;

String name,role;

float sal;

public Emp(int id,String name,String role,Float sal)

{

this.id=id;

this.name=name;

this.role=role;

this.sal=sal;

}

void display()

{

System.***out***.println(this.id+" "+this.name+" "+this.role+" "+this.sal);

}

void calpay()

{

this.sal=this.sal+(this.sal\*0.05f);

}

}

class Develop1 extends Emp{

public Develop1 (int id, String name, String role, Float sal) {

super(id, name, role, sal);

}

}

class Hr extends Emp{

public Hr(int id, String name, String role, Float sal) {

super(id, name, role, sal);

}

}

class Accountant extends Emp{

public Accountant(int id, String name, String roll, Float sal) {

super(id, name, roll, sal);

}

}

public class TASK5 {

public static void main(String args[])

{

Develop1 emp1 =new Develop1(55,"hari","Developer",90000.0f);

Hr emp2 = new Hr(66,"kumar","Hr",100000.0f);

Accountant emp3 = new Accountant(77,"manoj", "Accountant",70000.0f);

emp1.calpay();

emp2.calpay();

emp3.calpay();

emp1.display();

}

}

Above program abstract class Emp holds the all the common methods that can be used by all its child classes however the seems to be know need of creating an object for the Emp class itself.

package day3;

abstract class Emp

{

int id;

String name,role;

float sal;

public Emp(int id,String name,String role,Float sal)

{

this.id=id;

this.name=name;

this.role=role;

this.sal=sal;

}

void display()

{

System.***out***.println(this.id+" "+this.name+" "+this.role+" "+this.sal);

}

/\*void calpay()

{

this.sal=this.sal+(this.sal\*0.05f);

}\*/

abstract void calculatePay();

}

class Develop1 extends Emp{

public Develop1 (int id, String name, String role, Float sal) {

super(id, name, role, sal);

}

}

class Hr extends Emp{

public Hr(int id, String name, String role, Float sal) {

super(id, name, role, sal);

}

}

class Accountant extends Emp{

public Accountant(int id, String name, String roll, Float sal) {

super(id, name, roll, sal);

}

}

public class TASK5 {

public static void main(String args[])

{

Develop1 emp1 =new Develop1(55,"hari","Developer",90000.0f);

Hr emp2 = new Hr(66,"kumar","Hr",100000.0f);

Accountant emp3 = new Accountant(77,"manoj", "Accountant",70000.0f);

emp1.calculatePay();

emp1.display();

}

}

OP

The type Develop1 must implement the inherited abstract method Emp.calculatePay()

We access the calculatePay by using emp1 so this is not allowed because the method must implement the child classes ..

Abstract methods :

package day3;

abstract class Emp

{

int id;

String name,role;

float sal;

public Emp(int id,String name,String role,Float sal)

{

this.id=id;

this.name=name;

this.role=role;

this.sal=sal;

}

void display()

{

System.***out***.println(this.id+" "+this.name+" "+this.role+" "+this.sal);

}

/\*void calpay()

{

this.sal=this.sal+(this.sal\*0.05f);

}\*/

abstract void calculatePay();

}

class Develop1 extends Emp{

public Develop1 (int id, String name, String role, Float sal) {

super(id, name, role, sal);

}

void calculatePay() {

this.sal=this.sal+1000;

}

}

class Hr extends Emp{

public Hr(int id, String name, String role, Float sal) {

super(id, name, role, sal);

}

void calculatePay() {

this.sal=this.sal+2000;

}

}

class Accountant extends Emp{

public Accountant(int id, String name, String roll, Float sal) {

super(id, name, roll, sal);

}

void calculatePay() {

this.sal=this.sal+3000;

}

}

public class TASK5 {

public static void main(String args[])

{

Develop1 emp1 =new Develop1(55,"hari","Developer",90000.0f);

Hr emp2 = new Hr(66,"kumar","Hr",100000.0f);

Accountant emp3 = new Accountant(77,"manoj", "Accountant",70000.0f);

emp1.calculatePay();

emp1.display();

System.***out***.println(emp1.sal);

}

}

package day3;

class Emp

{

int id;

String name,role;

float sal;

public Emp(int id,String name,String role,Float sal)

{

this.id=id;

this.name=name;

this.role=role;

this.sal=sal;

}

void display()

{

System.***out***.println(this.id+" "+this.name+" "+this.role+" "+this.sal);

}

/\*void calpay()

{

this.sal=this.sal+(this.sal\*0.05f);

}\*/

abstract void calculatePay();

}

class Develop1 extends Emp{

public Develop1 (int id, String name, String role, Float sal) {

super(id, name, role, sal);

}

void calculatePay() {

this.sal=this.sal+1000;

}

}

class Hr extends Emp{

public Hr(int id, String name, String role, Float sal) {

super(id, name, role, sal);

}

void calculatePay() {

this.sal=this.sal+2000;

}

}

class Accountant extends Emp{

public Accountant(int id, String name, String roll, Float sal) {

super(id, name, roll, sal);

}

void calculatePay() {

this.sal=this.sal+3000;

}

}

public class TASK5 {

public static void main(String args[])

{

Develop1 emp1 =new Develop1(55,"hari","Developer",90000.0f);

Hr emp2 = new Hr(66,"kumar","Hr",100000.0f);

Accountant emp3 = new Accountant(77,"manoj", "Accountant",70000.0f);

emp1.calculatePay();

emp1.display();

System.***out***.println(emp1.sal);

}

}

OP

The type Emp must be an abstract class to define abstract methods

The abstract method calculatePay in type Emp can only be defined by an abstract class

package day3;

interface Paymant

{

void funt()

{

System.out.println("this is an interface function");

}

}

class Emp

{

int id;

String name,role;

float sal;

public Emp(int id,String name,String role,Float sal)

{

this.id=id;

this.name=name;

this.role=role;

this.sal=sal;

}

void display()

{

System.***out***.println(this.id+" "+this.name+" "+this.role+" "+this.sal);

}

/\*void calpay()

{

this.sal=this.sal+(this.sal\*0.05f);

}\*/

abstract void calculatePay();

}

class Develop1 extends Emp implements Payment{

public Develop1 (int id, String name, String role, Float sal) {

super(id, name, role, sal);

}

void calculatePay() {

this.sal=this.sal+1000;

}

}

class Hr extends Emp implements Payment{

public Hr(int id, String name, String role, Float sal) {

super(id, name, role, sal);

}

void calculatePay() {

this.sal=this.sal+2000;

}

}

class Accountant extends Emp implements Payment{

public Accountant(int id, String name, String roll, Float sal) {

super(id, name, roll, sal);

}

void calculatePay() {

this.sal=this.sal+3000;

}

}

public class TASK5 {

public static void main(String args[])

{

Develop1 emp1 =new Develop1(55,"hari","Developer",90000.0f);

Hr emp2 = new Hr(66,"kumar","Hr",100000.0f);

Accountant emp3 = new Accountant(77,"manoj", "Accountant",70000.0f);

emp1.calculatePay();

emp1.display();

System.***out***.println(emp1.sal);

emp1.funt();

}

}

OP

The method funt() is undefined for the type Develop1

Because the funt is an cancrit

Any concrete method in interface should always be define with default..

Any variable inside the interface is final or default

Interface :

* A inter face can extends another interface
* class can extends implement any number of interfaces
* Every var in interface if by default or final
* All the concreate methods in the interface should be default
* All static methods of interface are access only by the interface name and not the inherited class name.