OOP's

class O

Object and Method

following are Code Resuability Module:

Inheritacne

IS-A relationship

polymorphism(overloading, overriding)

following are Security Module:

Abstration

Data hiding

Encapsulation

Tightly coupled classes.

###

####<Contructor>

#####

1. is block (similiar to method.). having same anme as that class name.

2. Contructor has no RETURN type. cause we are initilizing OBJECT. not even void. so no need

other reason compiler creates constructor sometimes. so compiler cant judge which type to return.

3. Only modifiers for contructor are (Public, protected,default)

4. it executes Automatically when we create OBJECT.

5. Three ways to initilize a object, by using reference variable, using method, and using contructor.

So we can't use refference var for 1000 employe cause it will be same, we cant use each obj to

initialize val then there will be extra lines for initializing object value for each employe.

So onlye way left which is Contructor.

6. constructor are use initilize(means putting value) the object but not to creat obejct. thats the

job of 'new' keyword.

7. Three types of contructor are. Default constructor(no-arg), no argnt(user define), paramiterized

-> Default constructor: Are created by compiler. (Not JVM). ex 1.

-> No arg(user define): means conctructor has been declared by user. ex:2

-> Paramiterized : its has paramiter passes by obj to initilized intance var by constructor. ex3

EX 1:

class Test

{

int i;

//this ex doesn't have any constructor means this is called by default if there is no any

//conctructor declare by user.

public static void main(String []args)

{

Test t=new Test(); //it automatically run contructor

System.out.print(t.i) //this will print 0 thats intz by compiler default contructor

}

}

EX 2:

class Test

{

public test()

{

System.out.print("No argument contructor")

}

public static void main(String []args)

{

Test t=new Test(); //it automatically run contructor

t.test(); // this line is not needed to run contructor, but method.

}

}

EX 3:

class Employee

{

String name; //instace var

int emp\_id; //instace var

public Emplpyee(String n,int id)

{

this.name=n;

this.emp\_id=id;

}

public static void main(String[] args)

{

Employee e1= new Employee("Mehrab",101);

Employee e2= new Emploee("Hossain",102);

S.o.p (e1.name+" "+e1.emp\_id); this will print mehrab, 101

S.o.p (e2.name+" "+e2.emp\_id); this will print hossain, 102

}

}

###

#####<Inheritance> IS-A relationship

######

1.Always check which class extendzing which class and which obj calling which method/property.

2.Constructor does not inherited by child class. Remember

3.Private method does not inherited by child class.Remember

4.So object acquire all properties except above those.

5.even parent class exteds (Object class) which is the parent of all class. but doesn't show.

6.So one class can have only one parent class.thats why java doesn't support multiple,hybrid inhert.

Advantages:

>Code reusability.

>we achieve runtime polymorphism using inheritance that is method overriding.

Disadvantage:

>cause tightly coupledChange parent class means changes every child class.means all behavirour

or variable.

Inheritance are 3 types.

>Single: Means class A single to class B only. means single inheritace. A-->B. ex1

>Multilevel: Means class A to B and class B to c, so A also connect C indierectly. A->B->C

>Hierarchical: means Class A to class B and Class A to class C. A->B and A->C but B nor C can call

each other. cause main method. ex:3

Ex 1: (Single Inheritance)

class A //parent class or super class

{

void showA()

{

Sop("I am show A class");

}

}

//now inheriting A class in B, and inehriing all property and behaiviour from animal class.

class B extends A //child class or subclass.

{

void showB()

{

SOP("I am showing Class B")

}

public static void main(String []args)

{

A ob1= new A(); //creating obj of class A

ob1.showA(); //calling showA method from class A

ob1.showB(); //this is ERROR.cause class A doesnt extend B and dont know showB method.

B ob2= new B();

ob2.showA(); // this will print cause B extends A

ob2.showB(); // this will print cause its own method.

}

}

Ex 2:(Multilevel Inheritance)

class A //parent class or super class

{

void showA()

{

Sop("I am show A class");

}

}

//now inheriting A class in B, and inehriing all property and behaiviour from animal class.

class B extends A //child class or subclass.

{

void showB()

{

SOP("I am showing Class B")

}

//inheriting B class in C

class C extends B

{

void ShowC()

{

SOP("i am showing showC");

}

public static void main(String []args)

{

A ob1= new A(); //creating obj of class A

ob1.showA(); //calling showA method from class A

ob1.showB(); //this is ERROR.cause class A doesnt extend B and dont know showB method.

ob1.ShowC(); ///this is ERROR.cause class A doesnt extend c and dont know showB method

B ob2= new B();

ob2.showA(); // this will print cause B extends A

ob2.showB(); // this will print cause its own method.

ob3.showC(); ///this is ERROR.cause class B doesnt extend C and dont know showB method

C ob3= new C();

ob3.showA(); // this will print cause B extends A

ob3.showB(); // this will print cause its own method.

ob3.showC(); ///this is print cause C extends A and B extends A

}

}

}

EX 3: (Hierarchical)

class A //parent class or super class

{

void showA()

{

Sop("I am show A class");

}

}

//now inheriting A class in B, and inehriing all property and behaiviour from animal class.

class B extends A //child class or subclass.

{

void showB()

{

SOP("I am showing Class B")

}

//inheriting A class in C

class C extends A

{

void ShowC()

{

SOP("i am showing showC");

}

public static void main(String []args)

{

A ob1= new A(); //creating obj of class A

ob1.showA(); //calling showA method from class A

ob1.showB(); //this is ERROR.cause class A doesnt extend B and dont know showB method.

ob1.ShowC(); ///this is ERROR.cause class A doesnt extend c and dont know showB method

B ob2= new B();

ob2.showA(); // this will print cause B extends A

ob2.showB(); // this will print cause its own method.

ob3.showC(); ///this is ERROR.cause class B doesnt extend C and dont know showB method

C ob3= new C();

ob3.showA(); // this will print cause B extends A

ob3.showB(); // this will ERROR cause C extends A not B

ob3.showC(); ///this is print cause C extends A and B extends A

}

}

}

Type of Relationship between Classes

>Inheritance which is IS-A

we use it for code re use, method overrding and interfacing(through abstract class).

>Association which is HAS-A, it has two types

i) Aggeragation

ii) Composition

Association can be one to one, one to many, many to one and many to many.

Association:

Ex :1

class Engine

{

}

// following Car class is not extending so not all the property fo Engine class inherited.

// so its not tightly coupled that makes is association HAS-A relation.

class Car

{

public static void main(STring []args)

{

Engine eg= new Engine(); //Creating object of Engine class

//we can call any property using eg obj above from class Engine.

}

}

Now in between two classes

if

STRONG BOND then its Composition Association

Weak Bond then its Aggregation Association

Ex1: a CAR can function without {Music player} but not without {Engine}

so Car has Aggregation relation with Music player

and Car has COmposition relation with Engine

EX2: a College has a Departments and College has Teacher.

So without few deparmtnet college can run but not without teacher so department are weak

and teacher is Strong

So Association HAS-A is not blood relation and not tightly coupled.

& Inheritance IS-A is blodd and tightly coupled relations

for big project we should alwasy use Association it doesn't need to change everytime we change parent

####

#####<Polymorphism>

######

Poly = Many

Morphism=Form

2 types of poly

i) Compile time poly

Static poly

>we achieve compile time poly through {Method OVERLOADING}

>it handeles by Compiler

ii)Run time poly

Dynamic poly

>we achieve run time poly through {Method OVERRIDING}

>it handles by JVM

Overload: 3 condition

> which method has Same {Name} 2 or more

> which method is from same {CLASS}

> Different Argmt: Any 3 could be.

1/ Number of arg. (one method may have 1 int other have 3 int)

2/Sequence of arg.(one method have int and string other has string and int)

3/type of arg.(one method has int type and other has string type)

if above 3 condition satisfies than its method overload.

Overriding: 3 condition

>Same name

>Different Class

>Same Argument of any 3 could be. (above mentioned)

>inheritance(IS-A)

##Overloading:

Ex 1:

class Test

{

//declaring method

void show()

{

SOP("1");

}

//declaring method of same name & arg

void show(int i)

{

SOP("2");

}

//now main method

public static void main(String[] args)

{

Test t= new Test();

t.show(); //this will error ambugiouty cause both show method are same and {same} arg

//so thats why 3rd condition has to be achieve that is {different} arg.

//now if we change 2nd show method with argm and call it by obj in following

t.show() // now it will call the 1st show method

t.show(5)// now it will call the 2nd show as we are passing value in paramiter

}

}

Important question: and Test Case:

1> Method overloading not possible if we change {Return type} of both methods cause ambiguity

2> One class cant have 2 main method. but if we do it will not show error but print whatever in

1st main method and take 2nd method as another normal method that we can access by calling object

and pass argument value in it.

ex:

class Test

{

public static void main(String[] args)

{

SOP("1");

Test t=new Test();

t.main(20);

}

public static void main(int a)

{

SOP(a);

}

}

So we can overload main method by this system above.

3> Sometimes if we pass different type of arg in method which dont match it doesnt show error and

satisfy overloading.

Automatic promoted diagram:

Byte->short , Short->int, Char->int, Int->(long,float,double),long->(float,double),float->(double)

so if we pass char type in method by obje then it will run and print which covers the automatic

promoted diagram.

passing value chosse methods which has the immediate predecence. Means if we pass String value

by obje for ex "abc" to 2 methods of same name where 1st has OBJECT type and 2nd has String type

then it will print 2nd cause its immediate. but remember object is the mother of all parent

another case, if we pass "abc" to StringBuffer and String arg of 2 methods. then it will print

String method cause stringbuffer is higher precendece and string is immediate.

logic 1:

But if methods of different number of argmunet but {same}type, example int,float and float,int

then if we pass different type of data even its fall under diagram still it will not compile and

satisfy overloadig

Logic 2:

but if we totally change the arg of the 2 methods, ex: String,float and float,int then if we pass

different type of data and fall under diagram then it will be promoted. cuase its has diff arg.

So remember passing values must be under digram precendcy for logic 2.

4> Another way of passing values to method is {VARARG} means method accepts zero to multiple arg.

for ex: 1st method has normal int type arg and 2nd method has (int... a) than we can pass

t.show(10,20,30) means 3 values in that special VARARG argm

and, even if we dont pass any value in paramiter just call method then STILL it will get called

the VARARG method.

##OverRiding:

Again 4 logic to staisfies are

i) Same class

ii) different class

iii) Same Argument: any of it.

: no of arg,

: type of arg,

: Sequence of arg

iV) Inheritance(IS-A)

EX 1: Number of arg

class Test

{

void show()

{

SOP("1");

}

}

//now declaring another class

class XYZ extends Test // have to extends test otherwise wont satisfy overriding method.

{

void show()

{

SOP ("2");

}

// so class test and xyz has same name metho and XYZ methods override Test class method

P O V M(String[] args)

{

Test t= new Test();

t.show(); //this will show Test class show method.

//and now if we create XYZ object and call show.

XYZ obj= new XYZ();

obj.show(); // this will show xyz class show method.

}

}

> So we learn that despite extending Test class if we call show method of Test class by creating

object of XYZ class it DOES NOT show method of test class, but it will show method of itsselft

class.

we can call show method of test class using Test class Object.

EX 2: Type of arg

class Test

{

void show(String a)

{

SOP("1");

}

}

//now declaring another class

class XYZ extends Test // have to extends test otherwise wont satisfy overriding method.

{

void show()

{

SOP ("2");

}

// so class test and xyz has same name method but different type of arg, and its NOT overrriding

// and XYZ does not override Test class method cause different type

P O V M(String[] args)

{

Test t= new Test();

t.show(); //this will show Test class show method.

//and now if we create XYZ object and call show.

XYZ obj= new XYZ();

obj.show();

}

}

Ex 03: Sequence of arg

class Test

{

void show(String n, int i)

{

SOP("1");

}

}

//now declaring another class

class XYZ extends Test // have to extends test otherwise wont satisfy overriding method.

{

void show(String n, int i)

{

SOP ("2");

}

// so class test and xyz has same name method and same paramiter and sequence

// and XYZ methods override Test class method

P O V M(String[] args)

{

Test t= new Test();

t.show(); //this will show Test class show method.

//and now if we create XYZ object and call show.

XYZ obj= new XYZ();

obj.show(); this is show xyz class show method

}

}

Interview Question:

1> Do overriding method {Return} type has to be same ?

Ans: No it may be different. There are mother of all class means object. and it has a diagram of

child class. So the return type of the methods can be different under the hierarchical of

object class. Means to say return type of parent class metod could be Object and returnt type of

child class method could Object class child sub-type for ex: String, Stringbuffer, Stringbuilder,

Numbe, Character, Boolean, void, and .... .. .. .. (this dot dot is a type :P).

returnt type {Number} has its child sub-type returnt type like, Byete, short, Integer, Long,

Float, double. Remember if child class method returnt type is Object and parent class method

returnt type is StringBuffer, means child class extending higher sub-type return type in method.

which will {NOT WORK}

this Concept is called {Covariant} return type.

2> Override and Acces-modifiers:

Remember Child class access modifier has to be same as parent class access modfiers OR

it has to be bigger than the parent class access modifier.

means, SO.

> if no access modifiers declares then its same in both method

> if parent class is Public access modifiers and child class is default

NOT working cause public is bigger than default

So we have to put default in parent class access modifiers and public in child class

> if parent class is default access modifiers and child class is public than it will WORK.

> if parent class is Protected access modifiers and child is public will WORK if we put

private in child class than it will NOT work.

So Access priviliges has to maintain the levels.

3> Overriding and Exception handle:

i/ if parent class DOES NOT throws any exception then child class CAN throws only UNCECKED

exception and CAN NOT through checked exception.

ii/ if parent class method throughs any exception then child class CAN throws same exception or

any child class Exception. even we dont throw any exception in child class still it will

WORK.

iii/ if parent class throws exception of any and child class throws its Parent class exception then

its will give and error. for ex: Parent class has RuntimeExceptin and child class has Exception

itself which is the parent of RuntimeExecption. So this will NOT work

4> Overriding and Abstract Method;

> So if we declare parent class as abstract class. ex: "abstract class Test"

and declare method ex: "abstract void display()"

and if we run just like above example then it will give and ERROR

cause we have to declare.

So we have to declare override method in child class as well. or it will not work.

> we can not make object of abstract class.

So in a nutshell, if we have abstract class then all the abstract methods of that class has

to be declard by the child class indivisually or it will show and error.

5> Overriding and Interface

>so if we declare interface ex:

interface I1

{

void display();

}

then we have to do just as abstract class like. means we have to declare display method of

interface in child class which implemeting Interface class ex: class XYZ implements I1

and remember we can not put void display method in child class just like that. we have to put

access modifiers of public infront of void display or it will show error of weaker privileges.

> and just like abstract class object we cant not make object of interface as well.

6> Invoking overriden method from subclass

we can call parent class method from child class by child class object. even after being override.

just we have to put, "super.method\_name()" in the method of child class.

simple as that.

7> which method CAN NOT be overriden ?

i> Final method can not be overriden, ex: finald void show(){};

ii>private method can not be overriden,ex: private void show(){};

iii>static method can not be overrident,ex: static void show(){};

8> if we use synchronized or scriptfp keyword will it be overriden ?

Yes. it will WORK. ex: synchronized void show(){}; ex:scriptfp void show(){};

Nutshell, we can use these 2 keywords in any method means Parent or child both or any

it will WORK simply.

####

####<Abstraction>

####

Abstraction is {Internal Detailed} hiding means implementation and just highlighting the setup

services that we are offering. EX: car driver can see and use the steering, break, horn, gear, acce,

but driver not need to know how engine works or gear works. So internal fucntionality is being showed

but main services are displayed.

Abstration can be achchived in 2 Ways

i) Abstract Class

ii) Interface

Remember with Abstract(0-100%) abstraction can be possible but with Interface Class(100) surity.

//now lets take an abstract class called Vehicle

class Vehicle

{

int no\_of\_tyre; // here we are not providing detailed of implemantion.

void start(); // we are hiding detailed info of start method

// Remember while we hide detaile info of a method by not typing or giving its body then we

// have to declare its as "abstract void start();"

//above line of "void start()" wont work cause there is not abstract keyword

// So this line will wokr

abstract void start(); // this is the way to declare a method abstract

}

>> if there is a method of abstract class, then that class must has be abstract

but if the class is abstract not necessarryly method must be abstract.

//Lets take a class called Car and extending abstract vehicle class.

class Car extends Vehicle

{

int no\_of\_tyre= 4;

//as we are extending an abstract class then we have to make method body of abstract method of

//of all the method of abstract class which is declared as Abstract.

void start() // so start method were also in abstract class and diff is we re showing implemantion

{

SOP("Car start with a key");

}

}

>>if a regular class extends an Abstract class then all the method of abstract class that is

declared as abstract, has to be implemented in the regular class. or have to declare abstract

as well.

//lets take a another class called Scooter and extendes Abstract class Vehicle.

//so again we have to define abstract method body in the regular class which is scooter.

class Scooter

{

int no\_of\_tyre=2;

//defining abstract method body here but diff is we are showing implementation here.

void start()

{

SOP("Scooter start with a kick")

}

}

>> Fun fact all the 3 classes above is also example of method overriding as it has same method,arg

So in a [NUTSHELL]:

>We are hiding method body implemantion in abstract class. and internal detailes are hidden in

abstraction class.

and the regular classes which are inhering the abstract class has the implemantion and details.

>We CAN NOT make object of abstract class.

####

####<Interface>

###

in above abstraction we learnt that abstract class can be achieved by 2 ways

i> abstract class itself

ii> and Interface.

Lets talks about interface.

Interface are the blueprint of the class it specify what a class must do but not how to do.

so,

> Interface is used to achieve abstraction

> it supports multiple inheritance (Remember java does not support multiple inheritance)

> it can be used to achieve LOOSE COUPLING. (loose coupleming means if we change in one wont change

in other. so its not totally dependent)

Syntax:

interface InterfaceName

{

methods //abstract methods only. but compiler adds keywod public infront of the method.

// ex: public abstract void show(); // or abstract void show(){};

[Remember]: methods in interface are always abstract and public as above line

fields //any field we create in interface, compiler by default adds public static final infron

//of the field name. EX: public static final int a=10;

//its the only to declare variable in intercace

default method // interface update gives permission to put default access specifiers in interface

// only. ex: default show void (){}.

//but normal method cant be declared.

static method. // interface permits to declare method as static method, EX: static void run(){}

private method // in latest update interface permits to put private method also>

// EX:

}

So above declaration of property of interface has to be EXACT same as mentioned or it will show

error. and remember the difference between normal concrete method and abstract/interface method

those curly braces are the difference means method body cant be there. but withe the right

keyword we can have body.

EX 01:

interface I1

{

void show(); // in this line compiler will add by default "public" keyword

}

interface I2

{

void run()

}

class Test implements I1,I2 //this is the way interface support multiple inheritance.

{

//now we have to declare abstract method of interface just like abstract rule

public void show() // so now if u rember child class method access specifier has to be bigger

// or sameas parent class access specifers, so we have to put PUBLIC keyword

// infront of void show.

{

SOP("Show");

}

//same rule goes for interface I2 too.

public void run()

{

SOP("RUN")

}

public static void main(String []args)

{

// remember we cant create interface class object just like abstract

Test t= new Test();

t.show();

t.run();

}

}

####

####<Encapsulation>

####

Encapsulation is java is mechanism of wrapping the data(variable) and code acting on the date(method)

together as a single unit

2 rules to achive Encapsulation

> Declare the variable of a class as PRIVATE

> provide PUBLIC setter and getter methods to modify/initialize and view the varibale

EX 1:

class Employee

{

private int emp\_id; //remember private var can only access by that class itself.

// we have to declare all var in Encapsulation as PRIVATE.

// remember setter method adn getter method has to be always PUBLIC.

public void setEmpid(int id)

{

emp\_id=id;

}

public int getEmpId() //this is returning method thats why its return type is INT.

{

return empId;

}

}

class Company

{

public static void main(String []args)

{

//making emplooyee class object

Employee em=new Employee();

em.setEmpId(101); //setting the value in private emp\_id through setEMP method.

SOP (em.getEmpID()); //displaying the data that we initialize in previous line.

}

}

EX 2:

class Employee

{

private int emp\_id; //remember private var can only access by that class itself.

// we have to declare all var in Encapsulation as PRIVATE.

// remember setter method adn getter method has to be always PUBLIC.

public void setEmpid(int id)

{

if(id<100)

{SOP("Not an enmployee");}

else

{emp\_id=id;}

}

public int getEmpId() //this is returning method thats why its return type is INT.

{

return empId;

}

}

class Company

{

public static void main(String []args)

{

//making emplooyee class object

Employee em=new Employee();

em.setEmpId(101); //setting the value in private emp\_id through setEMP method.

SOP (em.getEmpID()); //displaying the data that we initialize in previous line.

}

}

####

####<this keyword>

####

which reffer to Object we call that variable "refference variable"

so THIS keyword reffer to object.

In following we have discussed 6 uses of THIS keyword

1>this keyword can use to refer current class intance variable.

EX 1:

class Test

{

int i;

void setValues(int x)

{

this.i=x;

//why its not i=x; cause suppose paramiter has same name insteax of 'x'. so we had to write

// i=i then. so local var and intance var will be same and if called, will print 0, cause its

// initial value declares is 0. So if we type this.i=i; then wont have any prob.

}

void show()

{

SOP(i);

}

}

class XYZ

{

P S V M ()

{

Test t=new Test();

t.setValues(10);

t.show();

}

}

2>this keyword can be use to invoke current clas method (implicitly)

EX 02:

class Test

{

void display()

{

SOP("this is display method")

}

void show()

{

display(); //we are here invokin current class method which is display() IN another method of c

//current class which is show, and if we call the show method by main method it will

//print "this is display" method. SO nutshell, this keywod has been added by compiler

//even we dont add it it in the coding

//so adding "this.display()" is good coding.

}

}

class XYZ

{

P S V M ()

{

Test t=new Test();

t.show(); //printing this is display method as this method been invoking in show method bythis.

}

}

3>this() keyword can be used to invoe current class contructor

EX 03:

class ThisDemo

{

//creating contructor of test class

ThisDemo()

{

//but if we wanna invoke parater consctructor below ThisDemo(int a), then just type this(val);

this(10);

SOP("No arg conctrucot")

}

////creating contructor of test class with paramiter

ThisDemo(int a)

{

SOP("arg conctructor")

}

}

P S V M ()

{

Test t1=new Test(); // this will print "no arg contructor"

Test t2=new Test(10); //this will print "arg contructor"

//so now calling after adding this() in paramiter conctructor we can just call ThisDemo(int a)

//with t2 and it will show both of it.

//to reverse the situation of invoking then just call ThisDemo() with t1;

}

}

4> this keywod can be use to pass as an argumetn in the method call.

Ex: 04

class ThisDemo

{

void m1(ThsiDemo td) //passing paramiter in m1 method

{

SOP("I am method m1");

}

void m2()

{

m1(this); //calling m1 method by putting keyword this as argm.

}

P S V M(String[] args)

{

ThisDemo td=new ThisDemo();

td.m2(); //will print "I am method m1";

}

}

5> this keywod can be used to pass as an argument in the contructor call.

Ex: 04

//Lest take a test class and make a constructor in it.

class Test

{

Test(ThisDemo td)

{

SOP ("Test class Constructor");

}

}

//now lets take another ThisDemo class and make method m1

class ThisDemo

{

void m1() //

{

Test t=new Test(this); //creating object of above Test class and calling its constructor

}

P S V M(String[] args)

{

ThisDemo t=new ThisDemo(); //creating object of ThisDemo class

t1.m1(); //calling m1 method which will call Test class constructor and pass arg 'this' to

//call the contructor of the TEST class as we know by creating object contructor

//get called automatically. Simple.

}

Remember one 1 point. carefull when calling or connecting anything as STATIC plays a vital role.

}

6> this keywod can be used to return the current class instance from the method.

EX 06:

class ThisDemo

{

ThisDemo m1() //checkout that return type of m1 method is ThisDemo, means instance of ThisDemo

// class its self

{

return this;

}

public static void main(String[] args)

{

ThisDemo t=new ThisDemo();

t.m1(); //calling m1 method.

}

}

####

#####<Super Keyword>

####

Remember 'this' keyword ? its the 'reference variable' which reffer the object of CURRENT class

property.

Super is also a refference variable.

But,

its a refference variable which is used to refer imeediate parent class OBJECT. so any class,

ex; class A extends class B. So then, Super will reffer object of the class B.

So, this reffers current class object, Super refferes to parent class OBJECT.

remember, inheritance has to be there for super fucntionality.

EX:

class A

{

int a=10;

}

class B extends

{

int a =20; //chechout class A has the same name of int a but diff class.

void show(int a) // taking local variable but same name as class instance var int a.

{

SOP(a); // printing 30 as initialized from ob1.

SOP(this.a); // prinitng 20 as this keyword reffering current class instance var.

SOP(super.a); // printing 10 as super keywod reffering current class parent class instance var.

}

P S V M(String[] args)

{

B ob1= show B();

ob1.show(30); // this will call show method from B class and pring 30 as we passing it.

}

}

uses of SUPER keyword

1> super keyword can be used to reffer parent class intance var.

example is just above ex.

2>super keyword can be used to invoke imeediate parent class method

EX:

class A

{

void m1()

{

SOP("I am in class A");

}

}

class B extends

{

void show()

{

Super.m1(); // so here super is reffereing imeediate parent class method.

}

P S V M(String[] args)

{

B ob1= show B();

ob1.show();

}

}

Above ex goes for one type of ex. but if we add another method of same name in class b as class A

name m1. then its a overriding.

EX:

class A

{

void m1()

{

SOP("I am in class A");

}

}

class B extends

{

void m1()

{

SOP("i am in class B");

}

void show()

{

m1(); //we are calling m1 and oviously it will current class m1. but if we want class 1 method

//m1 then we have to type next line code, which is super.m1();

Super.m1(); // so here super is reffereing imeediate parent class method.

// so above, compiler will not get confuse as ambigious.

}

P S V M(String[] args)

{

B ob1= show B();

ob1.show();

}

}

3> super keywod can be used to invoke immediate parent class constructor.

EX:

class A

{

A() // creating contructor of A class.

{

SOP("I am in class A");

}

}

class B extends

{

B() //creating conttructor of B class.

{

SOP("i am in class B");

}

P S V M(String[] args)

{

B ob1= show B();

// now when we create B class object, its going to call B constructor as weknow , but it will call

// A class contructor too. How ? Compiler automatically does that. which actually fucntion as

// super keyword. even if u dont wirte it.cause we have said it in the beginning that, super

// keyword fucntion while its inheriting something. So this is the reason A class constructor get

// called. and IF you type down, ex: super(); it will remain same in output. where A contructor

// print 'I am in class A' and then B contructor get printed 'I am in class B'

//so if we add super() or not, first parent class contructor get called and then child class.

}

}

####

#####<Final keyword>

####

where we can use Final keywod ?

ans: Variable, Method, and Class

we use final with variable WHEN we want a constant variable, not get changed ever.

we use final with method WHEN we want a method that not get ovverriden.

we use final with class WHEN we want a class not get Inherited.

Finale Variable example;

EX:

class Test

{

P O V M(String [] args)

{

int i=10;

i=i+20; //here are changing value of i, icreamenting 20

SOP (i);

// but if we had declared 'int i' as 'final int i' then above line of increament will show

// error as its final. we use final for constant like pi,g, etc.

}

}

Final method example:

EX: 1

class Demo

{

final void m1() //deaclaring method as final

{

SOP("I am in Demo class");

}

}

class Test extends Demo

{

void m1() //declaring method as same name and arg as m1, means overriding. but will get error

//as final method never gets override, but if there were no final keyword then it

//would have run easily when ever m1 gets called and would print "I am in Test clas"

{

SOP("I am in Test class")

}

P O V M()

{

Test t=new Test();

t.m1;

}

}

//above code will give error as method declare as final but if we dont want error just remove the

// final keyword then its good to go.

3>Final class example.

EX:

final class Demo

{

void m1() //deaclaring method as final

{

SOP("I am in Demo class");

}

}

class Test extends Demo // this will get error as final class CAN NOT be inherited.

{

void m1()

SOP("I am in Test class")

}

P O V M()

{

Test t=new Test();

t.m1;

}

}

// above code wil give error, but if u dont want error then just remove Final keyword from class Demo

// then above code is good to go.

####

####<Static>

####

>Acces Modifiers

(Public private proteceted defaualt)

>NoN access modifiers

(static final abstract synchronized transient volatile strictfp)

So static keyword is a non access modifiers

where we can use STATIC keyword

>variable: (class level variable), but if we use local variable then will get error

>method

>block

>Inner class: (nested class), but not outter class then will get error

so Lets talk about Static Variable.

Ex:

class Test

{

int a =10; //this is instance variable

static int b=10; //this is static variable, its the only way to declare static variable, means

// only class level

void m1()

{

int b=20; //its a local variable but if we add static infront of it, then its static var and

//and it will get error. so 'static int b=20' will throw and error

}

//so we have to remember that class level variable is only static variable. means above code where

//we declar static in b=10; there we ONLY can declare as static

}

Now one important Explaination:

if we declare a static var in 1 class called class TEST and then make another class called DEMO in

following code. then print static variable 'a' typing sop(a);

ans is NO. cause a variable is not recognized in Demo class.

now we, declare Test class int a as simple as that, will it print as typing SOP(Test.a) ?

ans is NO, cause static main method CAN NOT reffer NON STATIC variable a.

Now, if we add, Test word as same as class name in SOP(Test.a) and then compile?

ans is YES, it will print 10;

That is the speciality of static keyword that you dont need to create an object to connect a class

to print out its property if its STATIC.

Ex:

class Test

{

static int= 10;

}

class Demo

{

P S V M (String[] args)

{

SOP(Test.a); //will print 10 directly and no need object to connect with test class

SOP(a) ; //will put error as 'a' varibale is not recognize in DEMO class

}

}

Remember static variable belongs to the class not an OBEJCT>

Why we use staic variable ?

ans : MEMORY efficiency.

EX:

class Employee

{

//creating intance variable

int emp\_ID:

String name;

String company;

//creating constructor of Employee class which is paramiterized constructor

Employee(int emp\_id, String name, String company)

{

//initializing inatance values in above came form emp object below.

this.emp\_id=emp\_id;

String.name=name;

String.company=company;

}

void Display()

{

SOP(emp\_Id+" "+name+" "+company);

}

P S V M()

{

Emplpoyee emp1=new Employee(101,"jansir","Dynamic"); //passing arg in Employee contructor

emp1.Display(); //this will display above line info

Emplpoyee emp1=new Employee(102,"yousha","Dynamic");

emp2.Display(); //this will display imeediate above line code.

//but, now if we see, if we add 1000 of employee, then name and id will be different

//but company name is same and its occupy memory every time we make an object. which

//unnecesarry.

//So, we use static keyword of company to save memory and object can use that static var

//

}

}

above code has no problem except efficiency of MEMORY. so we change it in the folowing

EX:

class Employee

{

//creating intance variable

int emp\_ID:

String name;

static String company; //declaring static variable. checkout its class level as definition.

//creating constructor of Employee class which is paramiterized constructor

Employee(int emp\_id, String name, String company)

{

//initializing inatance values in above came form emp object below.

this.emp\_id=emp\_id;

String.name=name;

//above example we had initialize company name every time

}

void Display()

{

SOP(emp\_Id+" "+name+" "+company); // Dont need to type (Employee.company) as its same class.

}

P S V M()

{

Emplpoyee emp1=new Employee(101,"jansir"); //passing arg in Employee contructor

emp1.Display(); //this will display above line info

Emplpoyee emp1=new Employee(102,"yousha");

emp2.Display(); //this will display imeediate above line code.

//checkout we are not initize company name evertime we declare a object like above example.

}

}

Remember, how every object using compnay name as its not initilize in each of it ?

cause emp1,emp2 in stact memory area, and name,id,company in heap memory area as instance variable

so lets assume 1000 times we dont need to have company name again rather we gonna have it in

somewhere ONLY ONCE.

then we use static variale. Its gonna alocate a memory in 'class variable. which is another type of

memory like heap and stack memory.

so by these process we REDUCE 999 times memory alocation