Day3 14- Feb-2018

Core Java coding Training

In constructor overloading, Int can be converted into Float, short, double.

Because int, Float, short sizes are same.

Low size to high size it will convert, but high size to low size, it will not convert.

Run time Phal morphism

E.g.:- Method overriding:-

Providing specific implementation in child class method which is already available in parent class.

E.g.:-

Parent class and child class should be same.

Method names should be same.

package com.loginclasses;

class parent

{

Void m1(int a)

{

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

}

}

Public class Demo extends parent(

Void m1 (int a)

{

System.out.println(“child method”);

}

public static void main(String[] args) {

//todo

Parent d=new demo (); --‘”left hand side should be parent, right hand side should be child.”

Parent d=new parent (); -- it will not work , need to check it

d.m1(5);

Notes:-

Compile time it will check for – syntax errors

Run time- it checks for memory leakages.

Method overriding Rules:-

1. Method name and parameters should be same in parent class and child class
2. Parameters data types also should be same
3. We can have different return types in parent class and in child class

Differences between overloading and overriding

|  |  |
| --- | --- |
| Method Overloading | Method Overriding |
| 1. Definition | 1. definition |
| 1. takes place with in a class 2. 3. Method name is same, parameters are different | 2.Takes place in other classes  3. all are same |
| 4.Compile time | 4. Run time | |

**Protected Access modifier**

In Every class first line should be

1. package declaration
2. Importing
3. Class

Package com.logout;

Public class Demo {

protected int data=10;

protected void m1(int a)

System.out.println("m1 method");

}

public static void main(String[] args) {

}

}

Calling from above parent package

Package com.login;

Import com.core.demo; - explicit import statement, if we specify the class name

e.g. Import com.core.Logout.\*; - can access all the classes from the package)- Implicit import statement

Public class login extends demo { -- Need to select specific package to import)

public static void main(String[] args) {

//todo

Login d=new Login();

System.out.println(d.data);

d.m1(5);

We are getting parent objects through child objects.

**Abstraction: -** 3rd principle of oops

Hiding the implementation and showing only the functionality is known as abstraction

We can achieve abstraction through

1. Abstract class
2. Interface

One more method is there apart from Instance method and static method. i.e. Abstract method.

Abstract Method

Abstract Void m1 ();

Abstract is keyword

Void is Return type

M1 is Method

Any method which is having Abstract keyword and without method body, is known as abstract method.

Syntax:-

Abstract Void m1 ();

Void m1()

{

}

Abstract class:-

A class which has zero or more than abstract methods, we need to declare class has abstract method.

Package com.logout;

Public abstract class demo { - Need to declare Abstract keyword in the class)

Abstract void m1 ();

public static void main(String[] args) {

// TODO Auto-generated method stub

.. we can declare instance methods, static methods, instance variables, static varisbles.

But abstract class should be declared.

e.g.:-

Public abstract class demo {

Int a =10;

Static int b=10;

Demo()

{

}

Void m2()

{

}

Static void m3()

{

}

Abstract void m1();

Public static void main (String [] args) {

// TODO Auto-generated method stub

--We cannot create object for abstract class---

Demo d= new demo (); -- we cannot create direct object for abstract class, but through child object we can create--

Eg:-

Public abstract class demo {

Int a =10;

Static int b=10;

Demo()

{

}

Void m2()

{

}

Static void m3()

{

}

Abstract void m1();

Abstract void m4();

}

Abstract class child extends demo

{

Void m1()

{

}

Class c extends child

{

Void m4()

{

}

Parent n=new child();

n.m1();

n.m4();

system.out.println(n.a);

Whenever abstract methods are there, we have to provide, implementations for all the methods in child class when we do extending the class. If we don’t want to implement, make the child class is also abstract.

**Interface:-**

Interface is blueprint of a class. We can achieve 100% abstraction through Interface.

Through abstraqct class 0 to 100%.

Interface contains abstract methods, and static variables.

Syntax:-

Interface is a Keyword

A is interface name

Interface A

{

}

Interface contains, abstract methods and static variables.

Interface A

{

Abstract void m1(); ---if you don’t write abstract, compiler will provide

Public Abstract void m1();

Static int a=10; --- if you don’t write static, compiler will provide

Public static final int a=10;

}

Final is keyword, if we declare final, we cannot change the value of variable.

Variables are public, static, final.

Abstracts in interface is achieved by “**implements**” keyword.

“Class print implements test”

Note: - Child class access modifier methods should have higher scope or same scope of parent.

**public** **interface** Interface {

**public** **void** m1();

**public** **void** m2();

**public** **void** m3();

**int** *a*= 10;

**static** String *name*="test";

}

**class** print **implements** Interface

{

Public void m1()

{

System.out.println(“m1”);

}

Public void m2()

{

System.out.println(“m2”);

}

public static void main(String[] args) {

// TODO Auto-generated method stub

Show n= new print();

n.m1();

n.m2();

Mutiple inheritance can do through Interfaces.

**public** **interface** Interface1 {

**public** abstract **void** m1();

**public** **void** m2();

**public** **void** m3();

**int** *a*= 10;

**static** String *name*="test1";

**public** **interface** Interface2 {

**public** abstract **void** m4();

**public** **void** m5();

**public** **void** m6();

**int** *a*= 10;

**static** String *name*="test";

{

Class print implements interface1, interface2

}

Public void m1()

{

System.out.println(“m1”)

}

Public void m1()

{

System.out.println(“m1”)

}

Public void m1()

{

System.out.println(“m1”)

}

Public void m4()

{

System.out.println(“m1”)

}

Note:- if we have same methods in two interfaces, just once we can implement, no need to implement twice.

Note:-

A class implements an interface.

A class extends a class.

An interface extends an interface.

Public Interface show(

Public abstract void m1();

Void m2();

Int a =10;

Static string name=”tets”

Public Interface Read extends show(

Public abstract void m3();

Void m4();

Int a =10;

Static string name=”test”

Class print implements read

Note:-

In classes we cannot extend two classes, but in interfaces, we can extend multiple interfaces, this is called as multiple inheritances.

If we don’t know anything about requirement, we start with interface

Package com.atm;

Interface Atm{

Void Withdraw();

Void deposit();

Void balance();

}

If we know partial requirements, we start with abstract class

Package com.atm;

Abstract class Atm{

Void Withdraw();

{

}

Abstract Void deposit();

Abstract Void balance();

}

Class vf extends ATM

{

}

Abstract class and Interface differences

1. When we know partial implementation we go for abstract

When we don’t know requirements we go for interfaces.

We can achieve 0 to 100% abstraction in abstract class, through interface we can achieve 100%

1. In interface we can use only static variables and abstract methods. Constructor, static methods, and abstracts we can use in abstract class.
2. Compiler will not provide in default keywords in abstract methods.
3. Public and static keywords will provide for interface.
4. We cannot create direct objects in both the ways, we can create through child objects.
5. Through interface we can achieve multiple inheritances.
6. Need to compulsory implement all the abstract methods of interface in class., whereas inn abstract class, not compulsory of implementing all abstract methods in child class.