Day 1

Agile

Git Overview

Java Basic Programming

Java OOPs Concept

Exception Handling

Packages

Lang, file handling, collection framework.

Data Structure using Java

Maven tool

Git Git is a distributed version control source code management tool.

SVN

Project : Java, .net, php, angular, react js

1%

10%

20%

A person Login module

B person Application module

C person Customer module

D person Employee module

SVN

1st Person push

Remote folder

2nd Person push

3rd Person push

Please download the git software

Install the software.

**git –version**

To make folder as a git repository we have to write the command as

**git init**

**git status :** This command is use to check the status of repository

git add filename : This command is use to send the file from file system to staging area.

git commit –m “some message”

This command use to pass the file from staging area to local repository.

**Git :** git is a open source distributed version control software tool which help to push or send the data to remote repository.

**git hub :**

AWS : code commit a to z.

Azure

Google cloud

git init

git add filename.txt

git commit –m “message”;

link local repository to remote repository

git remote add origin URL

**git push –u origin HEAD**

**Please create New folder**

**Open the git bash terminal**

**Git clone** [**https://github.com/Kaleakash/java\_oops\_phase1.git**](https://github.com/Kaleakash/java_oops_phase1.git)

**We will take the break…**

**Java : Java is a pure object oriented and platform independent programming language.**

**C : 1970**

**C with class or C++ : 1980**

**1990 :**

**Initial name of the Java is Oak :**

**Nov 1995 rename from oak to Java.**

**James gosling and Team**

**It was belong to sun micro system**

**Now it is a part of Oracle.**

**Version**

**1.0, 1.2……………..1.8,………………………………..16**

**Java 1.7**

**Java 8 Features 100%**

**Java 9**

**Java 11**

**Java 15**

**Java 8 Version**

**Simple Java program**

**Syntax of class**

**class ClassName { //**

**variables/fields;**

**methods/functions;**

**}**

**className must be follow Pascal Naming rules.**

1. **If class contains 1 world first letter upper case.**
2. **If class contains more than one world each world first letter uppers.**

**class Demo {**

**public static void main(String args[]) {**

**System.out.println("Welcome to Java...");**

**}**

**}**

**Save the program className.java**

**Demo.java**

**To compile the program open command prompt and refer to java program directory**

**javac Demo.java (javac className.java)**

**after compile successfully**

**java Demo (java classname)**

**Day 2 : 04-04-2021**

**Created 3 folder**

**Manager**

**In manager folder created Java class**

**Compiled and run the program**

**git init**

**git add .**

**git commit –m “java program created”**

git remote add origin https://github.com/Kaleakash/java\_projects.git

**HEAD last commit**

**git push –u origin master**

**git push –u origin main**

**git push –u origin HEAD (last commit in branch)**

**git push –u origin**

**origin means given the name for that URL (remote URL).**

**Ajay**

**git clone URL**

**given the instruction to Ajay write another java program with A.java**

**Vijay**

**git clone URL**

**given the instruction to Vijay write B.java program**

Branch : branch is just like a movable pointer which hold more than one commit details.

By default In git default branch created with name master/main

To check branch

git branch

Syntax to create the branch

git branch branchname

move from one branch to another branch

git checkout branchname

how to delete the branch

git branch –D branchName

1 way

Git branch branchname

Git checkout branchName

2 way

Git checkout –b branchName (created branch and switch to new branch)

Ajay Developer

Created branch using command

Git checkout –b Ajay

Created Ajay.java program

Compile and run

Git add .

Git commit –m “file created”

Git push –u origin HEAD

Then in remote branch (manager merge the Ajay from to main branch)

Then in Ajay developer come to main branch and delete ajay branch.

Pull in main/master branch from remote repository

Git pull ( from remote branch)

If you want to do any changes create the branch and push or delete if anything go wrong

Git push –u origin HEAD (From user – defined branch)

class Demo {

int a; //Ajay

int b; //Vijay

}

**Conflicts**

**Switch from one remote repository to another repository**

**Clone vs fetch command**

**Java Notes**

**class Demo {**

**public static void main(String args[]) {**

**System.out.println("Welcome to Java...");**

**}**

**}**

**Data types**

2 types

1. Primitive types : it is use to store only value

8 types

1. byte 1 byte
2. short 2
3. int 4
4. long 8 : without decimal
5. float 4
6. double 8 :with decimal
7. char 2 :single character
8. boolean 1 bit : true or false.

Data Type example

class Demo {

public static void main(String args[]) {

int a=10;

boolean b = true;

System.out.println(a);

System.out.println("The value of a "+a);

System.out.println("The value of b "+b);

}

}

Type casting

Converting from one data type to another data is known as type casting.

1. Implicit
2. Explicit

Int family

------------🡪 Implicit --------------------------🡪

byte short int long

🡨------------------Explicit ---------------------------------

class Demo {

public static void main(String args[]) {

// byte range -128 to 127

byte a=10;

short b=a; //implicit type casting

System.out.println(a);

System.out.println(b);

short c =129; //size 2

byte d = (byte)c; // (type)variableName; Explicit type casting

System.out.println(c);

System.out.println(d);

}

}

int to float family

by default every decimal number in java double consider. The memory size for double is 8 byte.

class Demo {

public static void main(String args[]) {

//implicit type casting

int a=10;

float b=a; //implicit type casting

System.out.println(a);

System.out.println(b);

//float c=10.10f;

float c = (float)10.10;

//double c = 10.10;

int d = (int)c; // explicit type casting

System.out.println(c);

System.out.println(d);

}

}

Byte casting char and int

class Demo {

public static void main(String args[]) {

int a=72;

char b=(char)a; // explicit type casting : char to int

System.out.println(a);

System.out.println(b);

char c ='A';

int d = c; //implicit type casting : int to char

System.out.println(c);

System.out.println(d);

}

}

Operator :

Arithmetic Operator : +, -, \*, /, %

Conditional Operator : >, >=,<, <=, ==, !=

Assignment operator : =

Logical operator : &&, ||, !

&& : both the condition must be true then only result true else false.

|| : if any condition true then result is true.

Increment and Decrement : ++, --

Ternary operator : condition ? true: false;

class Demo {

public static void main(String args[]) {

/\*int a=1;

int b=0;

int res = a&&b;\*/ // C/C++ but not in java

boolean a = true;

boolean b = false;

boolean res1 = a&&b;

boolean res2 = a||b;

System.out.println(res1);

System.out.println(res2);

}

}

Increment and decrement

Increment by 1

Decrement by 1

a++ : post increment

++a; : pre increment

If we are assigning the value to another variable or using inside a expression pre and post work differently.

++a: first increment and assign

a++ : first assign and increment

class Demo {

public static void main(String args[]) {

int a=10;

System.out.println(a);

a++; // increment by 1

System.out.println(a);

++a; // increment by 1

System.out.println(a);

}

}

class Demo {

public static void main(String args[]) {

/\*int a=10;

int b=a;

System.out.println(b);

b = a++; // increment by 1

System.out.println(b);

b = ++a; // increment by 1

System.out.println(b); \*/

int x = 3;

int y = ++x + x++; // 4 + 4

System.out.println(y);

System.out.println(x);

}

}

class Demo {

public static void main(String args[]) {

int a=10;

int b=50;

//int res =a>b?a:b;

String res = a>b?"a is big":"b is big";

System.out.println(res);

}

}

If statement

1. simple if
2. If else
3. Nested if
4. If else if
5. Switch statement

Looping

1. While loop
2. Do while loop
3. For loop
4. For each loop or enhanced loop

Loop is use to iterate or execute statement again and again till the condition become false.

Starting 3 types

Initialization start and end position

Condition true

Increment / decrement increment or decrement.

while loop

initialization

while(condition) {

body of the loop

increment / decrement

}

Do while loop

initialization

do{

body of the loop

increment / decrement

} while(condition)

For loop

1 2 4

for(initialization;condition;increment/decrement) {

3

}

class Demo {

public static void main(String args[]) {

/\*int i=1;

int n=10;

while(i<=n) { variableName

System.out.println(i +" , "+n);

i++; // n--;

}

System.out.println("Finish");

}\*/

/\*int i=1;

int n=10;

do {

System.out.println(i +" , "+n);

i++; // n--;

}while(i>=n);\*/

for(int i=0;i<=10;++i) {

System.out.println("i="+i);

}

}

}

While loop : we have to do the logic till the condition becomes false.

Do while : if we want to do one task without checking conditions like menu driven application

For loop : fixed iteration.

Continue and break

class Demo {

public static void main(String args[]) {

//break and condition.

for(int i=0;i<=10;i++) {

//if(i==5) break; // stop the loop

if(i==5) continue; // skip next line statement part of loop

System.out.println(i);

}

}

}

1. Non primitive types or Reference data types

It is use to store value as well as reference of another data types.

array :

class

interface

enum

array : it is user defined data types which is use to store the same type of values. It is also know as reference or non-primitive data types.

syntax

datatype arrayName[];

int abc[]; //java

int abc[10]; //C – C++ but not in java

array initialization

int abc[]={10,20,30,40,50,60};

array values store using index position start with zero.

class Demo {

public static void main(String args[]) {

int a;

int abc[];

int mno[]={};

int xyz[]={10,20,30,40,50,60};

System.out.println(xyz.length);

System.out.println(mno.length);

//System.out.println(abc.length);

System.out.println(mno);

System.out.println(xyz);

//System.out.println(mno[0]);

System.out.println(xyz[0]);

}

}

For each loop or enhanced loop

Syntax

for(datatype variableName : arrayName) {

}

Display array using for loop and enhanced loop

class Demo {

public static void main(String args[]) {

int abc[]={100,200,300,400,500,600};

System.out.println(abc[0]);

System.out.println("using for loop");

for(int i=2;i<5;i=i+1) {

System.out.println(abc[i]);

}

System.out.println("Using for each loop or enhaced loop");

for(float a : abc) { // retrieve the element begining to end one by one

System.out.println(a);

}

}

}

Creating memory size for the array

Syntax

datatype arrayName[]=new dataType[size];

int abc[]=new int[10];

In Java new is a keyword which help to create the memory at run time (dynamic(.

Default values for array created using new keyword

Int family : 0

Float family : 0.0

Char : space (white space)

Boolean : false

String : null

Day 3 : 10-04-2021

Branch creation

Git branch branchName

Git checkout branchName

Or

Git checkout –b branchName

git clone URL

first time if you want to download the project from git repository we have to use

git clone URL

git clone URL

If you want to get the new updated data from existing git repository we have use git pull

Git pull : pull all files from repository and add the local repository.

git pull

Vs

git fetch : pull all files from repository but doesn’t add the local repository.

git merge

**Java OOPs Concept**

object : object is any real world entity.

Properties or State have variables / fields

Person

Behaviour do/does functions / methods.

Place

Accno, typeOfAccount

Bank

Withdraw, deposit, transferAmount

Animal

Wheel, name, price

Car   
 start(), appliedGear(), moving, stop()

Etc

Concept.

class : blue print of object or template of object or collection of objects which have same properties and behaviour or class is user defined data types which help to create the objet.

Syntax to create the object

ClassName objectReferenceName = new ClassName();

**Simple object example**

class Car {

int wheel;

String color;

float price;

void start() {

System.out.println("Car Started...");

}

void appliedGear() {

System.out.println("Applied Gear...");

}

void moving() {

System.out.println("Car is moving...");

}

void stop() {

System.out.println("Car Stop...");

}

}

class App {

public static void main(String args[]) {

Car innova = new Car(); // memory created (heap memory)

innova.start();

innova.stop();

}

}

Type of variable or fields

In Java variable or fields are divided into 3 types

1. Instance variable
   1. The variable which declare outside method but inside a class is known as instance variable.
   2. The instance variable hold default value according to their data types

Like int family -0, float family 0.0, char –space, boolean false, string null

* 1. Instance variable we can access directly in all non-static method within a same class.

1. Local variable :
   1. The variable which declare inside a method is known as local variable.
   2. The local variable doesn’t hold default value.
   3. The scope of variable within that method where it declare.
2. Static variable

**Instance variable and local variable example**

class Car {

int wheel;

String color;

float price;

void start() {

int a=100;

System.out.println("Car Started...");

System.out.println("wheel "+wheel);

System.out.println("Price "+color);

System.out.println("Price "+price);

System.out.println("a "+a);

}

void stop() {

String msg="Welcome";

System.out.println("Car Stop...");

System.out.println("wheel "+wheel);

System.out.println("Price "+color);

System.out.println("Price "+price);

System.out.println("msg "+msg);

}

}

class App {

public static void main(String args[]) {

Car innova = new Car();

innova.start();

innova.stop();

}

}

Creating multiple objects

Example

class Abc{

int a;

void dis() {

System.out.println("a = "+a);

}

}

class App {

public static void main(String args[]) {

Abc obj1 = new Abc();

Abc obj2 = new Abc();

obj1.dis();

obj2.dis();

obj1.a=10; // assign the value for instance variable through object.

obj2.a=20;

obj1.dis();

obj2.dis();

}

}

Create Employee class with three instance variable id,name,salary

And display methods.

Create EmployeeTest class which contains main method and

Create two employee class object set the value and display the values.

Id-int

name –string

salary – float / double

**constructor :** constructor is a type of special method which help to create the object.

Pts

1. Constructor have same name as class itself.
2. Constructor doesn’t contains return type not even void also.
3. Constructor no need to call it will call automatically when we create the object.

Method syntax or functions syntax

Method must be part of class

Syntax

returnType methodName(parameterList) {

method body

}

In the life of the object if we want to perform any task only one time that type of task only one time that type of task we have to write inside a constructor.

If we want to perform more than one time that type of code we have to write inside a methods.

Contructor

class Abc{

Abc() {

System.out.println("Object created....");

}

void dis() {

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

}

}

class App {

public static void main(String args[]) {

Abc obj1=new Abc();

obj1.dis();

obj1.dis();

obj1.dis();

}

}

**Parameterized constructor.**

class Abc{

int a,b,sum;

Abc() {

a=10;

b=20;

}

Abc(int x, int y) {

a=x;

b=y;

}

void setValue(int x, int y) {

a =x;

b =y;

}

void add() {

sum = a+b;

}

void display() {

System.out.println("Sum is "+sum);

}

}

class App {

public static void main(String args[]) {

Abc obj1 = new Abc(); obj1.add(); obj1.display();

Abc obj2 = new Abc(); obj2.add(); obj2.display();

Abc obj3 = new Abc(100,200); obj3.add(); obj3.display();

Abc obj4 = new Abc(10,20); obj4.add(); obj4.display();

Abc obj5 = new Abc();

obj5.setValue(1,2);

obj5.setValue(4,5);

obj5.add();

obj5.setValue(7,8);

obj5.display();

}

}

Encapsulation :

Binding or wrapping data (variable ) and code (functions/methods) in a single

Unit is known as Encapsulation.

Example :

class Employee {

variables ;

methods ;

}

If variable is a private we can’t access those variable directly as well as through object.

Outside a class we can’t access.

this keyword

When local variable or parameter and instance variable have same name. Then local variable hide the visibility of instance variable.

If we want to access instance variable then we have to use this.varaibleName

Encapsulation Example

class Employee {

private String name;

private float salary;

//helper method

void setValue(String name, float salary) {

this.name = name;

if(salary<0) {

this.salary = 8000;

}else {

this.salary = salary;

}

}

void display() {

System.out.println("Name is "+name);

System.out.println("Salary is "+salary);

}

}

class App {

public static void main(String args[]) {

Employee emp1 = new Employee();

//emp1.name = "Ramesh";

//emp1.salary = -12000;

emp1.setValue("Ramesh",-12000);

emp1.display();

}

}

JavaBean class

Normal class Vs JavaBean class

Normal class may or may be public

Variable may be or may not be private

If variable is private we have to provide helper method to set the value and method name may be anything but meaningful name.

class Employee {

String name;

float salary;

}

JavaClass must be public

Variable must be private

For every variable we have to provide setter and getter method

Setter method to set the value and getter method is use to get the value.

public class Employee {

private String name;

private float salary;

public setName(String name) {

this.name = name;

}

public String getName() {

return this.name;

}

public setSalary(float salary) {

this.salary = salary;

}

public float getSalary() {

return this.salary;

}

}

static

static keyword we can use with variable and method but not with class.

(outer class we can’t use static keyword).

(if class inner class we can use static keyword).

Static variable and methods.

1. If variable is a static we can assign the value for that variable with help of className as well as object.
2. If method is a static we can call that method with the help of className as well objects.
3. Inside a non static method we can access static as well as non static variable directly.
4. Inside a static method we can access only static variable directly.

Static example

class Abc {

int a;

static int b;

void dis1() {

System.out.println("Non Static method");

System.out.println("a = "+this.a);

System.out.println("b = "+b);

}

static void dis2() {

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

Abc obj = new Abc();

System.out.println("a = "+obj.a);

System.out.println("b = "+b);

}

}

class App {

public static void main(String args[]) {

//a=10;

Abc obj1 = new Abc();

obj1.a=100;

Abc.b=200;

obj1.b=300;

//Abc.a=400;

obj1.dis1();

Abc.dis2();

obj1.dis2();

}

}

Difference between Heap memory and Static memory

Every class only one static memory.

Static is global to all objects.

class Abc {

int a;

static int b;

void dis1() {

System.out.println("a = "+a);

System.out.println("b = "+b);

}

}

class App {

public static void main(String args[]) {

Abc obj1 = new Abc();

Abc obj2 = new Abc();

obj1.a=10;

obj1.b=20;

Abc.b=30;

obj2.a=40;

obj2.b=50;

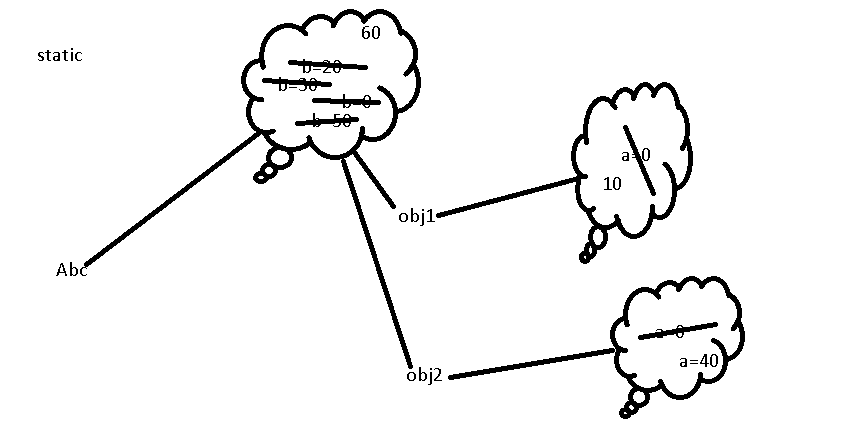
Abc.b=60;

obj1.dis1(); //a =10 , b=60

obj2.dis1(); //a=40 , b=60

}

}



**Day 4 : 11/04/2021**

Inheritance, Abstraction, Interface, final , packages, access specifiers.

super, this, this(), super()

Inheritance is use to inherit or acquire the properties and behaviour of old class to new class.

class OldClass { super class, base class, parent class,

variables;

methods;

}

class NewClass extends OldClass{ sub, derived class, child class

variables;

methods;

}

Inheritance Example

class A {

void dis1() {

System.out.println("A class method");

}

}

class B extends A{

void dis2() {

System.out.println("B class method");

}

}

class InheritanceDemo {

public static void main(String args[]) {

A obj1 = new A();

obj1.dis1();

B obj2 = new B();

obj2.dis2();

obj2.dis1();

}

}

Type of Inheritance

1. Single Inheritance : One super class and one sub class.

class A {}

class B extends A {}

1. Multilevel Inheritance : One super class and n number of sub classes connected one by one

class A {}

class B extends A {}

class C extends B{}

class D extends C{}

1. Hierarchical Inheritance : one super class and n number of sub classes connected directly to super class.

class A {}

class B extends A {}

class C extends A{}

class D extends A {}

1. Multiple Inheritance : more than one super class and one sub class.

class A {}

class B {}

class C extends A,B {} But Java doesn’t support this type of inheritance.

This type of Inheritance java support indirectly using interface.

Inheritance uni-directional not bi-directional

class A {}

class B {}

1st option

class A extends B{}

class B

2nd option

class A {}

class B extends A {}

OOPs relationship

is a

super class must be generics

class Employee{

}

class Manager extends Employee{

}

class Programmer extends Employee{

}

class SrManager extends Manager{

}

class Tester extends Employee{

}

**Inheritance Demo**

class Employee {

private int id;

private String name;

private float salary;

public void setEmployee(int id, String name, float salary) {

this.id = id;

this.name = name;

this.salary = salary;

}

public void displayEmployee() {

System.out.println("Id is "+this.id);

System.out.println("Name is "+this.name);

System.out.println("Salary is "+this.salary);

}

}

class Manager extends Employee{

private int numberOfEmp;

public void setManager(int numberOfEmp) {

this.numberOfEmp = numberOfEmp;

}

public void displayManager() {

System.out.println("Number of Employee "+numberOfEmp);

}

}

class Programmer {

}

class InheritanceDemo {

public static void main(String args[]) {

Employee emp1 = new Employee();

emp1.setEmployee(100,"Ravi",12000);

emp1.displayEmployee();

Manager mgr1 = new Manager();

mgr1.setEmployee(101,"Ajay",45000);

mgr1.setManager(12);

mgr1.displayEmployee();

mgr1.displayManager();

}

}

**Polymorphism**

One name many forms or many implementation

Compile time Run time

Static binding dynamic binding

Early binding late binding

Method Overloading Method Overriding

**Method Overloading : T**he method have same name but different parameter list (number of parameter or type of parameter must be differ)

Compile time polymorphism

class Abc {

void add(int a, int b) {

System.out.println(a+b);

}

void add(int x, int y, int z) {

System.out.println(x+y+z);

}

void add(float a, float b) {

System.out.println(a+b);

}

void add(String s1, String s2){

System.out.println(s1+s2);

}

}

class PolymorphismDemo {

public static void main(String args[]) {

Abc obj = new Abc();

obj.add(10,20); //compile time binding

obj.add(10,20,30);

obj.add(10.10f,20.20f);

obj.add("10","20");

}

}

**Method Overriding** : The method have same name and same method signature (number of parameter list type of parameter list and return type must be same).

To achieve method overriding classes must be in inheritance.

(super class and sub class).

**Annotation :** annotation is known as meta-data (data about data).

Java provided lot of pre-defined annotation.

All annotation start with @ followed by annotation names

**@Override** : This annotation we have to write in sub class method. If method overriding Then at compile time we won’t get any error. If method not overriding we will get the error.

Method Override

class Bike {

void speed() {

System.out.println("60km/hr");

}

}

class Honda extends Bike {

void color() {

System.out.println("Black");

}

}

class Pulsar extends Bike {

@Override

void speed() { // method overriding super class method

System.out.println("90km/hr");

}

void color() {

System.out.println("Red");

}

}

class PolymorphismDemo {

public static void main(String args[]) {

//Honda hh = new Honda(); hh.color(); hh.speed();

Pulsar pu = new Pulsar(); pu.color(); pu.speed();

}

}

super() method invocation

class Bike {

void speed() {

System.out.println("60km/hr");

}

}

class Honda extends Bike {

void color() {

System.out.println("Black");

}

}

class Pulsar extends Bike {

@Override

void speed() { // method overriding super class method

System.out.println("90km/hr");

}

void color() {

System.out.println("Red");

}

}

class Tvs extends Bike {

@Override

void speed() {

super.speed(); //merge both class code 60km/hr

System.out.println("20km/hr");

}

void color() {

System.out.println("White");

}

}

class PolymorphismDemo {

public static void main(String args[]) {

Honda hh = new Honda(); hh.color(); hh.speed();

Pulsar pu = new Pulsar(); pu.color(); pu.speed();

Tvs t = new Tvs(); t.color(); t.speed();

}

}

Multilevel super.methodName() concept

class Bike {

void speed() {

System.out.println("60km/hr");

}

}

class Pulsar extends Bike {

@Override

void speed() {

super.speed();

System.out.println("90km/hr");

}

}

class SuperPulsar extends Pulsar {

@Override

void speed() {

super.speed();

System.out.println("150km/hr");

}

}

class PolymorphismDemo {

public static void main(String args[]) {

SuperPulsar sp = new SuperPulsar();

sp.speed();

}

}

**abstract :** abstract is keyword we can use with method and class but not with variable.

1. Abstract method : The method without body or without curly braces or incomplete method is known as abstract method.

abstract returnType methdoName(parameterList);

abstract void speed();

1. If class contains one or more than one abstract method then we have to declare the class as abstract class.

Syntax

abstract class className {

}

1. Whichever class extends abstract class that class must be provide the body for all abstract method belong to that class. That class can ignore if that class itself is a abstract class.
2. Abstract class we can’t create the object.
3. Abstract class can contains normal as well as abstract methods means it can contains zero, 1 or all methods.
4. Abstract class can contains constructor.

(to assign the default value for the instance variable).

Final

final is a keyword we can use with variable, method and class.

1. Final variable : to declare constant in java we use final variable

final int A=10;

So we can’t change the value.

1. final method : if method is final we can’t override that method.

final void speed() {

}

Sub class can’t override final method can use it but can’t override.

1. final class : if class is final we can’t inherit that class.

final class Demo {

}

Demo class can’t extends or inherits

Final and abstract we can’t use together.

final and abstract variable, method and class not possible.

**interface :** It is type of reference data type also known as 100% pure abstract class (till Java7).

Syntax interface.

Till Java7 version .

interface interfaceName {

fields;

methods;

}

By default all fields are public static and final

By default all method are public and abstract.

interface Abc {

public static final int A=10;

static final int B=20;

final int C=30;

int D=40;

public abstract void dis1();

abstract void dis2();

void dis3();

}

interface Abc{

int A=10;

void dis1();

}

Interface Xyz {

int B=20;

void dis1();

}

interface Mno extends Abc,Xyz{ // multiple inheritance.

int C=30;

void dis3();

}

Like a class one interface extends another interface but class can extends only one class but interface can extends more than one interface.

Class can implements one or more than one interface .

class Demo implements Abc,Xyz {

}

Whichever class implements any interface that class must be provide the body for all abstract method belong that interface.

That class can ignore only if that class itself is a abstract class.

**Access specifiers overriding rules (method)**

Super class /interface sub

public public

default (nothing) public

default

protected public

protected

default

private can’t override.

1. class extends class (only one)
2. class implements interface (many)
3. interface X class
4. interface extends interface (many)

Interface Vs Abstract class

1. variable level : abstract class can contains any type of variable (static, non static and final). But interface contains onlypublic, static and final.
2. Method level : interface contains only abstract and public methods. Abstract class can contains normal as well as abstract with public , private, protected and default methods.
3. Constructor level : interface doesn’t contains constructor but abstract class can contains default as well as we can write parameterized constructor.
4. class can (Abstract / Normal) extends only one abstract class but can implements more than one interface.

Abstraction : hiding internal implementation without knowing background details is know as abstraction.

float checkBalance(int accno) {

}

So using abstraction class we can achieve partial abstraction.

But using interface we can achieve 100% abstraction.

interface Abc {

int A=10;

void dis1();

}

interface Xyz {

int B=20;

void dis2();

}

interface Mno extends Abc,Xyz {

int C=30;

void dis3();

}

class Demo implements Mno {

public void dis1() {

System.out.println("Abc interface method");

}

public void dis2() {

System.out.println("Xyz interface method");

}

public void dis3() {

System.out.println("Mno interface method");

}

}

class PolymorphismDemo {

public static void main(String args[]) {

Demo obj = new Demo();

obj.dis1();

obj.dis2();

obj.dis3();

}

}

**Polymorphism using object creation**

class A {

void dis1() {

System.out.println("A class method");

}

}

class B extends A {

void dis1() {

System.out.println("A class overrid method");

}

void dis2() {

System.out.println("B class method");

}

}

class PolymorphismDemo {

public static void main(String args[]) {

A obj1 =new A(); obj1.dis1();

B obj2 = new B(); obj2.dis1(); obj2.dis2();

//B obj3 = new A(); //creating super class object sub class reference not possible

A obj4 = new B(); //sub class object and super class reference possible. With the help of super class reference

// we can call only those methods which belong to super class or overrided methods.

// super class may be normal class or abstract class or interface

obj4.dis1();

//obj4.dis2();

}

}

interface A {

void dis1();

}

class B implements A {

public void dis1() {

System.out.println("A class overrid method");

}

}

class PolymorphismDemo {

public static void main(String args[]) {

A obj1 = new B(); // sub class object and interface reference possible if class implementat that interface.

obj1.dis1();

}

}

Day 5 : 17-04-2021

1st

class Dog{

public static void main(String [] args) { //int size = 12; //String name = "Goofy" Dog dog = new Dog(); dog.bark(9); }

void bark(int n){ System.out.println("barking "+n+" times") }}

Multiple constructor real time example

Java 8 functions.

**Access specifiers : Access specifiers or modified help use to show the visibility .**

**public**

**protected**

**default (nothing)**

**private**

**private : we can use with instance variable, static variable, constructor, method (non static as well as static) but not with local variable and class.**

**Scope : (within a same class)**

**default : we can use with all**

**scope : (within in a same package).**

**protected : we can use with instance variable, static variable, constructor, method (non static as well as static) but not with local variable and class.**

**scope : within same package other package if it is a sub class**

**public : we can use with instance variable, static variable, constructor, method (non static as well as static), public but not for local variable.**

**scope : same package as well as other package.**