**Java Notes :**

**Day 1**

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

Java

J2SE J2EE J2ME

JavaSE JavaEE JavaME

JSE JEE JME

Java Standard Java Enterprise Java Micro

Edition Edition Edition

Core Java

Basic Programming

OOPs

Exception handling

Packages

Lang, io, util, awt, swing, jdbc, rmi etc

Java Software :

Version of Java 1.0, 1.1,1.2,1.4,1.5, 1.6, 1.7,Java 8 , 9,10, 11, 12

JDK : Java Development Kit

JRE : Java Run time Environment :

**Java Virtual Machine**

Class syntax

class className {

variables

methods /functions

}

class Test {

public static void main(String args[]) {

System.out.println(“Welcome to Java”);

}

}

**Data Types :** Data type is a type of data which tells what type of value it can hold.

2 types

1. Primitive data types
   1. byte 1 byte
   2. short 2
   3. int 4
   4. long 8
   5. float 4
   6. double 8
   7. boolean 1 bit
   8. char 2 byte
2. Non primitive data types or reference data types.

**Types Casting :**Converting from one data type to another data types

2 types

1. Implicit type casting
2. Explicit type casting

-------🡪implicit type casting -----🡪

byte short int long

1 2 4 8

🡨--------explicit type casting ---------

class Test {

public static void main(String args[]) {

byte a = 10;

short b =a; //i

System.out.println(a);

System.out.println(b);

short c = 127;

byte d = (byte)c; //E

System.out.println(c);

System.out.println(d);

}

}

int float

4 4

class Test {

public static void main(String args[]) {

int a=10;

float b = a; //Im

System.out.println(a);

System.out.println(b);

float c = 10.0f;

int d =(int)c;

System.out.println(c);

System.out.println(d);

}

}

**Reference Data types** : It is use to store value as well as reference of another data types.

4 types

1. array
2. class
3. interface
4. enum

**array :** array is use to store the same type of values.

int abc[10]; C/C++

int abc[];

int []abc;

int [] abc;

int[] abc;

looping :

1. while loop
2. do while loop
3. for loop
4. for each loop

class Test {

public static void main(String args[]) {

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

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

System.out.println(abc[5]);

System.out.println(abc.length);

}

}

while loop

initialization start and end

condition

body of the loop

class Test {

public static void main(String args[]) {

int i=0,n=10;

while(i<=n) {

System.out.println(i);

i++;

}

}

}

class Test {

public static void main(String args[]) {

int i=0,n=10;

do{

System.out.println(i);

i++;

}while(i<=n);

}

}

class Test {

public static void main(String args[]) {

int i,n;

for(i=0,n=10;i<=n;i++){

System.out.println(i);

}

}

}

For each loop

Syntax

for(dataType variableName:arrayName) {

}

class Test {

public static void main(String args[]) {

int []abc={10,20,30,40,50,60,70};

System.out.println("Normal loop");

for(int i=0;i<abc.length;i++){

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

}

System.out.println("For each loop");

for(int a:abc){

System.out.println(a);

}

int b;

//System.out.println(b);

}

}

**Creating memory size to array**

dataType []arrayName=new dataType[size];

int abc[]=new int[10];

class Test {

public static void main(String args[]) {

int abc1[]={10,20,30,40,50};

int []abc=new int[10];

System.out.println(abc.length);

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

System.out.println(abc[9]);

for(int i=0;i<abc.length;i++) {

abc[i]=100+i;

}

for(int i=0;i<abc.length;i++){

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

}

}

}

**Taking value through keyboards**

1. using Scanner class
2. using DataInputStream class
3. using BufferedReader class
4. command line arguments.

Scanner is a pre-defined class which provided set of methods which help to scan the value through keyboards. Syntax to create the Scanner class object.

**Scanner obj = new Scanner(System.in);**

Scanner class is a part of **util** package.

**import java.util.Scanner;**

**or**

**import java.util.\*;**

nextByte()

nextShort()

nextInt();

nextLong()

nextBoolean()

nextFloat()

nextDouble()

nextChar() no methods

next() or nextLine() to receive String value.

import java.util.Scanner;

class Test {

public static void main(String args[]) {

Scanner obj = new Scanner(System.in);

System.out.println("Enter the id");

int id = obj.nextInt();

obj.nextLine(); //hold the enter key

System.out.println("Enter the name");

String name = obj.nextLine();

System.out.println("Enter the salary");

float salary = obj.nextFloat();

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

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

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

}

}

**OOPs**

**object and class**

object : any real world entity

state or properties,have,**variables/fields**

Person

Behavior , do/does , **function /methods**

Bank

Animal

Car

Employee

**class :** class is a blueprint of object or template of objet or user-defined data types which help to describe the objects.

class Car {

int wheel;

String color;

float price;

void start() {

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

}

void appliedGear() {

}

void moving() {

}

void stop() {

}

}

class CarTest {

public static void main(String args[]) {

System.out.println("Main Method");

Car santro =new Car();

santro.start();

}

}

**Types of variables /fields**

3 types

1. instance variables
   1. The variables which declare inside a class but outside a method including main method is known as instance variables.
   2. Instance variables hold default value according to their data types. int family 0, float 0.0, char space, Boolean false, String null.
   3. Instance variable we can use directly in all methods within a same class but method must be **non static**.
2. local variables
   1. The variable which declare inside a method including main method is known as local variables.
   2. Local variable doesn’t hold default value. We have to initialize.
   3. Scope of the local variable within that block where it declare.
3. static variables

Employee

id,name,salary -🡪instance variables

Scanner object

read()

calSalary()

hra 10%, da 5% and pf 7% local variables

display()

id,name,salary(grossSalary)

EmployeeTest

Main method

Employee object create

Call read, calSalary, display

import java.util.Scanner;

class Employee {

int id;

String name;

float salary;

Scanner obj = new Scanner(System.in);

void read() {

System.out.println("Enter the id");

id = obj.nextInt();

System.out.println("Enter the name");

name = obj.next();

System.out.println("Enter the salary");

salary = obj.nextFloat();

}

void calSalary() {

float hra, da,pf;

hra = salary\*0.10f;

da = salary\*0.05f;

pf = 0.07f\*salary;

salary = salary+hra+da-pf;

}

void display(){

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

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

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

}

}

class EmployeeTest {

public static void main(String args[]){

Employee emp1 = new Employee();

emp1.read();

emp1.calSalary();

emp1.display();

Employee emp2 = new Employee();

emp2.read();

emp2.display();

}

}

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

**Pts**

1. Constructor must have same name as the 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.

class Employee {

Employee() {

System.out.println("constructor called...");

}

void display(){

System.out.println("method");

}

}

class EmployeeTest {

public static void main(String args[]){

Employee emp1 = new Employee();

emp1.display();

emp1.display();

emp1.display();

}

}

class Employee {

int id;

Employee() {

System.out.println("constructor called...");

}

void display(){

System.out.println("method "+id);

}

}

class EmployeeTest {

public static void main(String args[]){

Employee emp1,emp2,emp3;

new Employee();

new Employee().display();

emp1 = new Employee();

emp1.id=100;

emp2 = new Employee();

emp2.id=200;

emp3 = emp1;

emp1.display(); emp2.display(); emp3.display();

emp1=null;

//emp1.display();

emp2.display(); emp3.display();

emp3 = null;

}

}

**Parameterized constructor with this keyword**

class Employee {

int id;

String name;

Employee() {

id = 123;

name="Unknown";

}

Employee(int id, String name) {

this.id =id;

this.name = name;

}

void setValue(int id, String name) {

this.id = id;

this.name = name;

}

void display() {

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

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

}

}

class EmployeeTest {

public static void main(String args[]){

Employee emp1 = new Employee(); emp1.display();

Employee emp2 = new Employee(1,"Ravi"); emp2.display();

Employee emp3 = new Employee();

emp3.setValue(2,"Ajay");

emp3.setValue(3,"Seeta");

emp3.display();

}

}

**Encapsulation: :** Binding or wrapping data(variables) and code(functions/methods) in a single unit is known as Encapsulation.

Ex : class

**JavaBean class:**

**Normal class Vs JavaBean class**

class Person {

id,name,salary

}

public class Person {

private int id;

private String name;

private float salary;

//setter and getter

public void setId(int id) {

this.id = id;

}

public int getId() {

return id;

}

}

**Day 2**

**Inheritance:** Inheritance is use to inherits or acquire the properties and behavior of old class to new class.

class OldClass { super class, base class

properties parent class

behavior

}

class NewClass extends OldClass {

properties sub class, derived class,

behavior child class

}

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();

}

}

**Types of Inheritance**

1. Single Inheritance

class A {}

class B extends A {}

1. Multilevel Inheritance

class A {}

class B extends A{}

class C extends B {}

class D extends C {}

1. Hierarchical Inheritance

class A {}

class B extends A {}

class C extends A {}

1. Multiple Inheritance

class A {}

class B {}

class C extends A,B {} wrong in Java.

Java doesn’t support multiple inheritance. It support using **interface**.

**OOPs Relationship**

1. Is a relationship
2. Has a relationship

Manager **is a** Employee

Employee **has a** Address

class Employee {

id,name,salary

Address add=new Address();

}

class Manager extends Employee{

}

class Developer extends Employee{

}

class Address {

city, state

}

Has a relationship

1. Association
2. Aggregation
3. Composition

class A {

B obj1 = new B(); 0,1, many

}

class B {

A obj2 = new A(); 0, 1, many

}

class Manager {

Address add =new Address();

}

class Developer {

Address add = new Address();

}

class Address {

}

class Student {

StudentHistory sh = new StudentHistory()

}

class StudentHistory {

}

class Employee {

id,name,salary

Scanner obj =new Scanner();

Address add = new Address();

read(), calSalary(),display()

}

class Manager extends Employee{

numberOfEmp;

readMgr()

add.readAdd();

disMgr()

add.disAdd();

}

class Programmer extends Employee {

projectName;

readPgr()

disPgr()

}

class Address {

city

state

Scanner obj =new Scanner(System.in);

readAdd()

disAdd()

}

**Inheritance Example with is a and has a relationship**

import java.util.Scanner;

class Employee {

int id;

String name;

float salary;

Scanner obj = new Scanner(System.in);

Address add = new Address();

void read() {

System.out.println("Enter the id");

id = obj.nextInt();

System.out.println("Enter the name");

name = obj.next();

System.out.println("Enter the salary");

salary = obj.nextFloat();

}

void calSalary() {

float hra, da,pf;

hra = salary\*0.10f;

da = salary\*0.05f;

pf = 0.07f\*salary;

salary = salary+hra+da-pf;

}

void display(){

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

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

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

}

}

class Manager extends Employee {

int numberOfEmp;

void readMgr() {

read();

System.out.println("Enter the number of Emp");

numberOfEmp = obj.nextInt();

add.readAdd();

}

void disMgr() {

display();

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

add.disAdd();

}

}

class Programmer extends Employee {

String projectName;

void readPgr() {

read();

System.out.println("Enter the projectName");

projectName = obj.next();

add.readAdd();

}

void disPgr() {

display();

System.out.println("Project Name is "+projectName);

add.disAdd();

}

}

class Address {

String city;

String state;

Scanner obj = new Scanner(System.in);

void readAdd() {

System.out.println("Enter the city");

city = obj.next();

System.out.println("Enter the state");

state = obj.next();

}

void disAdd() {

System.out.println("City is "+city);

System.out.println("State is "+state);

}

}

class EmployeeTest {

public static void main(String args[]){

/\*Employee emp1 = new Employee();

emp1.read();

emp1.calSalary();

emp1.display();

Employee emp2 = new Employee();

emp2.read();

emp2.display();\*/

System.out.println("Enter the manager details");

Manager mgr = new Manager();

mgr.readMgr();

mgr.disMgr();

System.out.println("Enter the developer details");

Programmer prg = new Programmer();

prg.readPgr();

prg.disPgr();

}

}

**Polymorphism** One name many forms or many implementations.

2 types

1. Compile time polymorphism
   1. Method Overloading (static binding or early binding)
2. Run time polymorphism
   1. Method Overriding (dynamic binding or late binding)

Method Overloading :The method have same name but different parameter list(number of parameter list as well as type of parameter list must be different).

class Operation {

void add(int a, int b) {

System.out.println(a+b);

}

void add(int a, int b,int c) {

System.out.println(a+b+c);

}

void add(String a, String b) {

System.out.println(a+b);

}

}

class PolyDemo {

public static void main(String args[]){

Operation op = new Operation();

op.add(10,20);

op.add(1,2,3);

op.add("Raj","Deep");

}

}

**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 more than one class with inheritance is require.

class Bike {

void speed() {

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

}

}

class Pulsar extends Bike {

void speed() {

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

}

void color() {

System.out.println("Black");

}

}

class Honda extends Bike {

void color() {

System.out.println("Red");

}

}

class Tvs extends Bike {

void speed() {

super.speed(); //merge super class coding

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

}

void color() {

System.out.println("White");

}

}

class TvsSuper extends Tvs {

void speed() {

super.speed(); //merge super class coding

System.out.println("100km/hr");

}

void color() {

System.out.println("White");

}

}

class PolyDemo {

public static void main(String args[]){

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

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

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

TvsSuper ts =new TvsSuper(); ts.speed();

}

}

**Annotation :** annotation is a meta-data. Meta-data means data about data.

In Java all annotation start with pre-fix **@** followed by annotation name.

Some annotation we can use on class level, method level, property level.

int a;

**@Override :**

class Bike {

void speed() {

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

}

}

class Pulsar extends Bike {

@Override

void speed() {

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

}

void color() {

System.out.println("Black");

}

}

class PolyDemo {

public static void main(String args[]){

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

}

}

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

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

**Syntax**

abstract return type methodName(parameterList);

abstract void speed();

1. Abstract class: if class contains abstract method then we have to declare the class as a abstract class.
2. Abstract class can contains normal as well as abstract methods.(0, 1 or all).
3. Which ever class extends abstract class that class must be provide the body for all abstract method belong to that class. That class can ignore only it that class itself is abstract class.
4. Abstract class we can’t create the object.
5. Abstract class can contains default constructor as well as we can write parameterized constructor.

abstract class Bike {

abstract void speed();

}

abstract class Pulsar extends Bike {

void color() {

System.out.println("Black");

}

}

class PolyDemo {

public static void main(String args[]){

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

}

}

**final :** final keyword we can use with variable, method and class.

**Final variables :** to declare the constant value in java we use final keyword with variables.

**final int A=10;**

**final method** : If method is a final we can’t override that method.

**final class:** If class is final we can’t extends that class.

class Employee {

final int id;

final String name;

Employee() {

id =100;

name="Ravi";

}

void dis() {

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

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

}

}

class PolyDemo {

public static void main(String args[]){

Employee emp1 = new Employee();

emp1.dis();

}

}

class Employee {

final int id;

final String name;

Employee() {

id =100;

name="Ravi";

}

void setInfo(final int abc) {

System.out.println(abc);

//abc=100;

System.out.println(abc);

}

void dis() {

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

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

}

}

class PolyDemo {

public static void main(String args[]){

Employee emp1 = new Employee();

emp1.dis();

emp1.setInfo(123);

}

}

**static** : static keyword we can use with variable, method but not with class. (if class is inner class then we can use static keyword for inner class).

class Outer {

static int a;

static void dis() {}

static class Inner {

}

}

class Abc {

int a;

static int b;

void dis1() {

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

System.out.println("a "+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 StaticDemo {

public static void main(String args[]) {

Abc obj1 = new Abc();

obj1.dis1();

Abc.dis2();

obj1.a=100;

Abc.b=200;

obj1.dis2();

obj1.b=300;

}

}

**Static concept**

class Abc {

int a;

static int b;

void dis1() {

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

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

}

static Abc obj;

static void dis2() {

obj = new Abc();

System.out.println("static method "+obj.a);

}

}

class StaticDemo {

public static void main(String args[]) {

Abc.dis2();

/\*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(); obj2.dis1();\*/

}

}

**Interface :** Interface is a type of reference data types which is use to store constants variables and abstract methods. Interface also known as 100% pure abstract class.

Syntax

interface interfaceName {

variables;

methods;

}

By default all variables inside a interface are **public** **static** and **final**

By default all methods inside a interface are **public** and **abstract**

interface Abc {

int A=10;

void dis();

}

interface Xyz {

int B=20;

void dis1();

}

interface Mno extends Abc,Xyz{

int C=40;

void dis2();

}

class Demo **implements** Abc, Xyz {

}

**Super Sub**

(Class/Abstract class

Interface)

public public

protected public

protected

default(nothing) public

protected

default

private Can’t Override

interface Abc {

int A=10;

void dis();

}

interface Xyz {

int B=20;

void dis1();

}

interface Mno extends Abc,Xyz{

int C=40;

void dis2();

}

class Demo implements Abc, Xyz {

public void dis() {

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

}

public void dis1() {

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

}

}

class InterfaceDemo {

public static void main(String args[]) {

Demo obj =new Demo();

obj.dis();

obj.dis1();

}

}

**Difference between abstract class and interfaces**

1. Interface contains only final variables but abstract class can contains normal as well as final variables.
2. Interfaces contains only abstract methods but abstract can contains normal as well as abstract methods.
3. Interface doesn’t contains default constructor as well as we can’t write parameterized constructor but abstract class can contains default as well as parameterized constructor
4. Any class(normal/abstract) can extends only one abstract/normal class but can implements more than one interfaces.

**Runtime polymorphism using classes**

Creating sub class object and super class reference.

1st Example

class A {

void dis1() {

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

}

}

class B extends A {

void dis1() {

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

}

void dis2() {

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

}

}

class InterfaceDemo {

public static void main(String args[]) {

A obj1 = new A(); obj1.dis1(); //obj1.dis2();

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

A obj3 = new B(); obj3.dis1(); //obj3.dis2();

//B obj4 = new A();

}

}

2nd example

Sub class object and super class reference (where super class is abstract class)

abstract class A {

abstract void dis1();

}

class B extends A {

void dis1() {

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

}

void dis2() {

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

}

}

class InterfaceDemo {

public static void main(String args[]) {

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

A obj3 = new B(); obj3.dis1(); //obj3.dis2();

//B obj4 = new A();

}

}

3rd Example

Sub class object and interface reference.

interface A {

void dis1();

}

class B implements A {

public void dis1() {

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

}

void dis2() {

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

}

}

class InterfaceDemo {

public static void main(String args[]) {

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

A obj3 = new B(); obj3.dis1(); //obj3.dis2();

//B obj4 = new A();

}

}

**Abstraction :** hiding the internal implementation without knowing background details.

interface ATeam {

int add(int a, int b);

}

interface BTeam {

int sub(int a, int b);

}

class Server implements ATeam,BTeam{

public int add(int a, int b){

return a+b;

}

public int sub(int a, int b){

return a-b;

}

public void ownMethod() {

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

}

}

class InterfaceDemo {

public static void main(String args[]) {

Server ss = new Server();

ATeam tt1 = new Server();

System.out.println(tt1.add(10,20));

//System.out.println(tt1.sub(10,20));

BTeam tt2 = new Server();

//System.out.println(tt2.add(10,20));

System.out.println(tt2.sub(10,20));

}

}

**this, super, this() and super()**

class A {

int n=10;

}

class B extends A {

int n=20;

void dis1() {

int n = 30;

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

System.out.println("instance n "+this.n);

System.out.println("super n "+super.n);

}

}

class InterfaceDemo {

public static void main(String args[]) {

B obj = new B();

obj.dis1();

}

}

**Constructor chaining using this()**

class A {

A() {

this(10);

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

}

A(int x) {

this(1,2);

System.out.println("A(x)");

}

A(int x, int y) {

System.out.println("A(x,y)");

}

}

class InterfaceDemo {

public static void main(String args[]) {

A obj1 = new A();

}

}