**Java Full Stack Training - onMobile Client**

**Day 1 - Core Java - 04-09-2019**

**C :**

**#include<stdio.h>**

**void mno() {**

**xyz();**

**}**

**void abc() {**

**mno();**

**}**

**void main() {**

**printf("Welcome to C");**

**abc();**

**}**

**OOPs :**

**object : Any real world entity.**

**Ex:**

**properties(State)----> have ---> fields/variables**

**Person**

**behaviour ------>do/does --- function/methods**

**Bank**

**Animal**

**Car**

**etc**

**class : Blue print of object or template of object or collection of object which have same properties and behaviour or use-defined data type which help to describe the objects.**

**syntax**

**class ClassName {**

**properties**

**behaviour**

**}**

**class Person {**

**String name;**

**int age;**

**void sleeping() {**

**}**

**void teaching() {}**

**}**

**Person p1 = new Person();**

**C + OOPs = C++ + PIEA**

**Java .net**

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

**Java 1.0, 1.2, 1.4, 1.5,1.6,1.7 ,**

**1.8, 1.9, 1.10, 1.11, Java 12**

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

**2 types**

**1. Primitive Data types : It is use to store only value**

**2. Non primitive data types ore reference data type: it is use to store value as well as reference of another data type.**

**Primitive data types**

**8 types**

**1. byte 1 byte**

**2. short 2 byte**

**3. int 4 byte**

**4. long 8**

**5. float 4**

**6. double 8**

**7. char 2**

**8. boolean 1 bit**

**class Demo {**

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

**int a=10;**

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

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

**System.out.println("Value of a "+a);**

**System.out.printf("Value of a = %d\n",a);**

**}**

**}**

**Non Primitive data types**

**1. array**

**2. class (pre-defined or user-defined)**

**3. interface (pre-defined or user-defined)**

**4. enum (pre-defined or user-defined)**

**types casting : converting one data type to another data types is known as type casting.**

**2 types**

**1. Implicit type casting**

**2. explicit type casting**

**----------------------> Implicit ---------------------------------->**

**byte short int long**

**<---------------------Explicit ---------------------<-----------------**

**class Demo {**

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

**byte a = 10; // range of byte value -128 to 127**

**short b = a;**

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

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

**short c = 129;**

**byte d = (byte)c;**

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

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

**}**

**}**

**class Demo {**

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

**int a=10;**

**float b =a;**

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

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

**float c = (float)10.10;**

**int d = (int)c;**

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

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

**}**

**}**

**Operator**

**if statements**

**switch statement**

**looping**

**while loop**

**do while loop**

**for loop**

**for each loop or enhanced loop**

**array : it is a user-defined data types which is use to store same type of values.**

**syntax**

**dataType arrayName[];**

**int abc[];**

**int []abc2**

**int [] abc3;**

**int[] abc4**

**[]int abc5; InValid**

**int []abc2={10,20,30,40,40,50}**

**for enhanced loop**

**for(dataType variableName:arrayName){**

**}**

**class Demo {**

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

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

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

**System.out.println(abc[4]);**

**System.out.println("Using for 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);**

**}**

**}**

**}**

**Creating memory for array in Java**

**dataType []arrayName = new dataType[size];**

**int []abc = new int[10]; 10\*4 = 40 byte memory**

**int abc[10];**

**class Demo {**

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

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

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

**System.out.println(abc[4]);**

**System.out.println("Using for 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);**

**}**

**}**

**}**

**Taking value through keyboards in Java.**

**1. Using Scanner class**

**2. Using DataInputStream class**

**3. BufferedReader class**

**4. Command Line Arguments.**

**Scanner : Scanner is a pre-defined class part of util package. Which contains set of methods which help to scan the value through keyboards.**

**package : It is collection of classes and interfaces.**

**syntax to create the Scanner class object.**

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

**byte : obj.nextByte()**

**short : obj.nextShort();**

**int : obj.nextInt();**

**long: obj.nextLong();**

**float : obj.nextFloat();**

**double : obj.nextDouble();**

**boolean : obj.nextBoolean();**

**String str = obj.next(); //only one word**

**String str = obj.nextLine(); more than one word.**

**Create the Scanner class object in main method, create array id,name,salary variables and create the memory size for those variable using scanner class object. then take id,name,salary for more than one employee and find hra, da and pf where HRA is 10%on salary, DA is 5%salary and PF is 7%salary. Then display all employees id,name,salary(grosssSalary).**

**import java.util.Scanner;**

**class Demo {**

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

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

**System.out.println("How many record do you wan to store?");**

**int n = obj.nextInt();**

**int id[]=new int[n];**

**String name[]=new String[n];**

**float salary[]=new float[n];**

**System.out.println("Enter the records one by one");**

**//Receive the values**

**for(int i=0;i<n;i++) {**

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

**id[i]=obj.nextInt();**

**obj.nextLine(); //use to hold the enter key**

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

**name[i]=obj.nextLine();**

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

**salary[i]=obj.nextFloat();**

**}**

**//Business logic**

**for(int i=0;i<n;i++) {**

**float hra, da,pf;**

**hra = salary[i]\*0.10f;**

**da = salary[i]\*0.05f;**

**pf = salary[i]\*0.07f;**

**salary[i]=salary[i]+hra +da -pf;**

**}**

**//Display Records;**

**for(int i=0;i<n;i++) {**

**System.out.println("id is"+id[i]+" Name is"+name[i]+" Salary is"+salary[i]);**

**}**

**}**

**}**

**OOPs**

**object and class**

**object : any real world entity**

**class : blue print of object or template of object.**

**price, wheel, color etc**

**Car :**

**start(), appliedGear(), moving(), stop()**

**className objectRefeferenceName = new className();**

**Car santro =new Car();**

**Access the class non static methods**

**objectReferencename.methodName();**

**santro.start();**

**objectRefernceName.variableName = value;**

**santro.wheel=4;**

**Types of variables or fields**

**In Java variables are divided into 3 types**

**1. Instance variable**

**a. The variable which declare outside a method including main method is known as Instance variable.**

**b. All instance variable hold default values according to their data types.**

**like**

**int family 0**

**float family 0.0**

**char space (white)**

**boolean false**

**String null**

**c. We can access all instance variable inside same class method directly but the method must be non static.**

**2. Local variable**

**a. The variable which declare inside a method including main method is known as local variable.**

**b. The scope of the variable within that block where it declare.**

**c. The local variable doesn't hold default value we have to initialize.**

**class Car {**

**int wheel;**

**String color;**

**float price;**

**void start() {**

**int temp=0;**

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

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

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

**}**

**void stop() {**

**String msg="Welcome";**

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

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

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

**}**

**}**

**class CarTest {**

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

**Car santro = new Car();**

**santro.start(); santro.stop();**

**}**

**}**

**Create two classes Employee and EmployeeTest**

**Where Employee class must be contains three instance variables id,name,salary and Scanner class objects.**

**read(), calSalary() and display() non static methods.**

**EmployeeTest class must be contains main methods then create the Employee class object and call read() which help to receive the value through keyboards. calSalary() method which help to do calculation on salary.**

**hra, da and pf (local variables).**

**hra = 10%on Salary , da = 5%on salary and pf = 7%onsalary**

**display() method to display id,name,salaray(GrosssSalary)**

**First Employee object all three methods.**

**Second Employee object only two methods (read and 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 = salary \*0.07f;**

**salary = salary+hra+da-pf;**

**}**

**void display() {**

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

**emp2.display();**

**}**

**}**

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

**pts**

**1. Constructor must be same name as the class itself.**

**2. Constructor doesn't contains return type not event void also.**

**3. Constructor no need to call it will call automatically when we create the object of that class.**

**class Employee {**

**Employee() {**

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

**}**

**void display() {**

**System.out.println("display method");**

**}**

**}**

**class EmployeeTest {**

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

**Employee emp1 = new Employee();**

**emp1.display();**

**emp1.Employee();**

**}**

**}**

**In the life of the object if we want to perform task only one time that type of code write inside constructor ex: initialization.**

**In the life of the object if we want to perform task more than one time that type of code write inside a methods.**

**class Employee {**

**int id;**

**Employee() {**

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

**}**

**void display() {**

**System.out.println("display method");**

**}**

**}**

**class EmployeeTest {**

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

**//int a;**

**Employee emp1;**

**new Employee();**

**new Employee().display();**

**new Employee().display();**

**Employee emp2 = new Employee();**

**emp2.display();**

**emp2.display();**

**emp2.display();**

**}**

**}**

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

**Ex :**

**class**

**class Employee {**

**private String name;**

**private float salary;**

**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 EmployeeTest {**

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

**//display();**

**//name="Raj";**

**//salary = 12000;**

**Employee emp = new Employee();**

**//emp.name="Ravi";**

**//emp.salary = -12000;**

**emp.setValue("Ravi",-12000);**

**emp.display();**

**}**

**}**

**JavaBean class : JavaBean is a type of normal class with set of rules and regulation on property and behaviours.**

**Normal class Vs JavaBeanClass**

**class Employee {**

**private int id,age;**

**private String name;**

**//helper method**

**void setValue() {}**

**}**

**public class Employee {**

**private int id;**

**private String name;**

**private int age;**

**//setter and getter**

**//set is use to set the value with term and conditions.**

**//get is use to get the value**

**public void setId(int id) {**

**this.id = id;**

**}**

**public int getId() {**

**return id;**

**}**

**}**

**Pure Encapsulation class ie JavaBean**

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

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

**properties**

**behaviour**

**}**

**class NewClass extends OldClass{**

**properties**

**behaviour //sub class, derived class, 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 EmployeeTest {**

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

**A obj1 = new A();**

**B obj2 = new B();**

**obj1.dis1();**

**obj2.dis2();**

**obj2.dis1();**

**}**

**}**

**Types of Inheritance**

**1. Single Inheritance**

**one super class and one sub class**

**class A {}**

**class B extends A {}**

**2. 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{}**

**3. 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{}**

**4. Multiple Inheritance**

**more than one super class and one sub class**

**class A {}**

**class B {}**

**class C extends A,B {} Java doesn't support this type of inheritance. This type of inheritance java support indirectly using interface.**

**class Employee {**

**id,name,salary**

**Scanner obj**

**read()**

**calSalary()**

**display()**

**}**

**class Manager extends Employee{**

**numberOfEmp**

**readMgr()**

**disMgr();**

**}**

**class Programmer extends Employee{**

**projectName;**

**readPrg()**

**disPrg()**

**}**

**OOPs Relationship**

**1. Manager is a Employee**

**2. Employee/Manager has a Address**

**class Manager extends Employee{**

**}**

**class Employee {**

**Address add = new Address();**

**}**

**class Address {**

**}**

**has a relationship**

**1. Association**

**2.Aggregation**

**3. Composition**

**3. static variable**

**class A {**

**//B obj1 = new B();**

**}**

**class B {**

**A obj2 = new A();**

**A obj3 = new A();**

**}**

**class Employee {**

**Address ladd = new Address();**

**Address padd = new Address();**

**}**

**class Address {**

**}**

**class Student {**

**StudentHistory sh = new StudentHistory();**

**}**

**class StudentHistory {**

**}**

**0-0**

**0-1**

**0-many**

**1-0**

**1-1**

**many-1**

**many-many**

**Day 2 - Core Java - 05-09-2019**

**Polymorphism : One name many forms or many implementation.**

**2 types**

**Compile time Run time**

**Static binding Dynamic binding**

**Early binding late binding**

**Ex: Method Overloading Ex: Method Overriding**

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

**DriverManager.getConnection("url")**

**DriverManager.getConnection("url","userName");**

**DriverManager.getConnection("url","userName","password");**

**class Operation{**

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

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

**}**

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

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

**}**

**void add(String s1, String s2) {**

**System.out.println(s1+s2);**

**}**

**}**

**class Demo {**

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

**Operation op = new Operation();**

**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).**

**Method overriding we can't achieve without inheritance.**

**Annotation : Meta-data : Data about data.**

**Java provided lot of pre-define annotation.**

**Pre-defined annotation as well as user-defined annotation.**

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

**Few or some annotation we can write on class level or method level or constructor level and property level.**

**@Override : This annotation we will write on method level.**

**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("Red");**

**}**

**}**

**class Honda extends Bike {**

**void color() {**

**System.out.println("Black");**

**}**

**}**

**class Tvs extends Bike {**

**@Override**

**void speed() {**

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

**super.speed();**

**}**

**void color() {**

**System.out.println("White");**

**}**

**}**

**class Demo {**

**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();**

**}**

**}**

**abstract : abstract is keyword we can use with method and class 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 returnType methodName(parameterList);**

**abstract void speed();**

**2. abstract class : If class contains one or more abstract method then we have to declare the class as a abstract class.**

**syntax**

**abstract class className {**

**}**

**abstract class Bike {**

**}**

**3. Whichever class extends abstract class that class must be provide the body for all abstract method belongs to that class that class can ignore only if that class itself is a abstract class.**

**4. abstract class we can't create the objects.**

**5. abstract class can contains normal as well as abstract methods.**

**ie zero, 1 or many abstract methods.**

**6. Abstract class can contains default constructor as well as we can write parameterized constructor.**

**abstract class Bike {**

**int wheel;**

**Bike() {**

**wheel = 4;**

**}**

**void speed(){**

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

**}**

**}**

**abstract class Pulsar extends Bike {**

**void color() {**

**System.out.println("Red");**

**}**

**}**

**class Honda extends Bike {**

**@Override**

**void speed() {**

**System.out.println("50km/hr");**

**}**

**void color() {**

**System.out.println("Black");**

**}**

**}**

**class Tvs extends Bike {**

**@Override**

**void speed() {**

**System.out.println("40km/hr");**

**}**

**void color() {**

**System.out.println("White");**

**}**

**}**

**class Demo {**

**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();**

**}**

**}**

**abstract class Bike {**

**int wheel;**

**Bike() {**

**wheel = 4;**

**}**

**void speed(){**

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

**}**

**}**

**abstract class Pulsar extends Bike {**

**void color() {**

**System.out.println("Red");**

**}**

**}**

**class Honda extends Bike {**

**@Override**

**void speed() {**

**System.out.println("50km/hr "+wheel);**

**}**

**void color() {**

**System.out.println("Black");**

**}**

**}**

**class Tvs extends Bike {**

**@Override**

**void speed() {**

**System.out.println("40km/hr");**

**}**

**void color() {**

**System.out.println("White");**

**}**

**}**

**class Demo {**

**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();**

**}**

**}**

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

**1. final variable : To declare the constant value in java we use final keyword with variables.**

**final int A=10;**

**2. final method : If the method is final we can't override that method.**

**3.final class : if the class is a final we can't extends or inherits that class.**

**final and abstract we can't use together.**

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

**1. if the variable are static we can assign the value for those variable using className as well as objects.**

**syntax**

**className.staticVariableName= value;**

**objectReferenceName.staticVariableName = value;**

**2. If the method is static we can call those method with the help of className as well as objectReferenceName;**

**className.staticMethodName();**

**objectReferenceName.staticMethodName();**

**3. Inside non static method we can access static as well as non static variable directly for the same class. But inside static method we can access only static variable directly of the same class.**

**class Outer {**

**int x;**

**static int y;**

**void dis1() {**

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

**System.out.println("x "+x);**

**System.out.println("y "+y);**

**}**

**static void dis2() {**

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

**Outer o = new Outer();**

**System.out.println("x "+o.x);**

**System.out.println("y "+y);**

**}**

**}**

**class Demo {**

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

**Outer out = new Outer();**

**out.x=10;**

**Outer.y=20;**

**out.y=30;**

**out.dis1();**

**Outer.dis2();**

**out.dis2();**

**}**

**}**

**class Outer {**

**int x;**

**static int y;**

**void dis1() {**

**System.out.println("x "+x);**

**System.out.println("y "+y);**

**}**

**static void dis2() {**

**Outer o= new Outer();**

**System.out.println(y+", "+o.x);**

**}**

**}**

**class Demo {**

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

**Outer.dis2();**

**/\*Outer o1 = new Outer();**

**Outer o2 = new Outer();**

**o1.x=10;**

**Outer.y=20;**

**o1.y=30;**

**o2.x=40;**

**Outer.y=50;**

**o2.y=60;**

**o1.dis1();**

**o2.dis1(); \*/**

**}**

**}**

**interface : interface is a type of reference data type which is also known as 100% pure abstract class till JDK 7.**

**syntax**

**interface interfaceName {**

**fields;**

**methods;**

**}**

**by default all fields are public final static**

**by default all methods are public and abstract.**

**interface Abc {**

**public static final int A=0;**

**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 dis2();**

**}**

**interface Mno extends Abc,Xyz{**

**int C =30;**

**void dis3();**

**}**

**Super class /interface Sub class**

**public public**

**protected public**

**protected**

**default(nothing) public**

**protected**

**default**

**private can't override**

**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 Demo1 implements Abc,Xyz {**

**public void dis1() {**

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

**}**

**public void dis2() {**

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

**}**

**}**

**class Demo {**

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

**Demo1 obj1 = new Demo1();**

**obj1.dis1();**

**obj1.dis2();**

**}**

**}**

**interface A {**

**int A1=10;**

**void dis1();**

**}**

**interface B {**

**int A1=20;**

**void dis1();**

**}**

**class D {**

**public void dis1() {**

**System.out.println("D class method coding ");**

**}**

**}**

**class C extends D implements A,B{**

**public void dis1() {**

**System.out.println("A,B and D coding");**

**System.out.println(A.A1 +" - "+B.A1);**

**}**

**}**

**C obj = new C();**

**obj.dis1();**

**this(), this, super, super():**

**class A {**

**int x=10;**

**}**

**class B extends A {**

**int x=20;**

**void dis1() {**

**int x=30;**

**System.out.println("x local variable "+x);**

**System.out.println("x instance variable "+this.x);**

**System.out.println("x super variable "+super.x);**

**}**

**}**

**class Demo {**

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

**B obj = new B();**

**obj.dis1();**

**}**

**}**

**this() : this() is use to do constructor calling or constructor chaining of the same classes.**

**class Employee {**

**int id;**

**String name;**

**float salary;**

**Employee() {**

**this.id = 123;**

**this.name = "Unknown";**

**this.salary = 8000;**

**}**

**Employee(int id) {**

**this();**

**this.id = id;**

**}**

**Employee(int id, String name) {**

**this(id);**

**this.name = name;**

**}**

**Employee(int id, String name, float salary){**

**}**

**void dis() {**

**System.out.println("id "+id+" name "+name+" salary "+salary);**

**}**

**}**

**class Demo {**

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

**Employee emp1 = new Employee(); emp1.dis();**

**Employee emp2 = new Employee(1); emp2.dis();**

**Employee emp3 = new Employee(2,"Ravi"); emp3.dis();**

**}**

**}**

**super() : super() is use to invoke sub class constructor to super constructor chaining.**

**class A {**

**A() {**

**System.out.println("A class constructor");**

**}**

**A(int x) {**

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

**}**

**}**

**class B extends A {**

**B() {**

**super(10);**

**System.out.println("B class constructor");**

**}**

**}**

**class Demo {**

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

**B obj1 = new B();**

**}**

**}**

**class A {**

**A() {**

**this(100);**

**System.out.println("A class constructor");**

**}**

**A(int x) {**

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

**}**

**}**

**class B extends A {**

**B() {**

**this(10);**

**System.out.println("B class constructor");**

**}**

**B(int x) {**

**System.out.println("B(int)");**

**}**

**}**

**class Demo {**

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

**B obj1 = new B();**

**}**

**}**

**abstract class Employee {**

**int id;**

**String name;**

**float salary;**

**Employee() {**

**this.id = 123;**

**this.name = "Unknown";**

**this.salary = 8000;**

**}**

**Employee(int id) {**

**this();**

**this.id = id;**

**}**

**Employee(int id, String name) {**

**this(id);**

**this.name = name;**

**}**

**Employee(int id, String name, float salary){**

**this(id,name);**

**this.salary = salary;**

**}**

**void dis() {**

**System.out.println("id "+id+" name "+name+" salary "+salary);**

**}**

**}**

**class Manager extends Employee {**

**int numberOfEmp;**

**Manager(int id, String name, float salary, int numberOfEmp) {**

**super(id,name,salary);**

**this.numberOfEmp = numberOfEmp;**

**}**

**@Override**

**void dis() {**

**super.dis();**

**System.out.println("Number of Emp "+this.numberOfEmp);**

**}**

**}**

**class Demo {**

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

**Manager mgr = new Manager(1,"Raj",45000,10);**

**mgr.dis();**

**}**

**}**

**init or initialization block and static block**

**class A {**

**A() {**

**System.out.println("A class constructor");**

**}**

**static {**

**System.out.println("static block A class ");**

**}**

**{**

**System.out.println("init bock");**

**}**

**void dis() {**

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

**}**

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

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

**String str[]={};**

**Demo.main(str);**

**}**

**}**

**class Demo {**

**static**

**{**

**System.out.println("main class static method");**

**}**

**{**

**System.out.println("main method init block");**

**}**

**Demo () {**

**System.out.println("Demo class constructor");**

**}**

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

**A obj1 = new A();**

**obj1.dis();**

**obj1.dis();**

**A obj2 = new A();**

**obj2.dis();**

**obj2.dis();**

**}**

**}**

**class A {**

**A() {**

**System.out.println("A class constructor");**

**}**

**static {**

**System.out.println("static block A class ");**

**}**

**{**

**System.out.println("init A bock");**

**}**

**void dis1() {**

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

**}**

**}**

**class B extends A {**

**B() {**

**System.out.println("B class constructor");**

**}**

**static {**

**System.out.println("static block B class ");**

**}**

**{**

**System.out.println("init B bock");**

**}**

**void dis2() {**

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

**}**

**}**

**class Demo {**

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

**new B();**

**}**

**}**

**class A {**

**void dis1() {**

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

**}**

**}**

**class B extends A {**

**void dis1() {**

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

**}**

**void dis2() {**

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

**}**

**}**

**class Demo {**

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

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

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

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

**B obj4 = (B)obj3; obj4.dis1(); obj4.dis2();**

**//B obj4 = new A();**

**}**

**}**

**Super class is type of abstract class**

**abstract class A {**

**abstract void dis1();**

**}**

**class B extends A {**

**void dis1() {**

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

**}**

**void dis2() {**

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

**}**

**}**

**class Demo {**

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

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

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

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

**B obj4 = (B)obj3; obj4.dis1(); obj4.dis2();**

**//B obj4 = new A();**

**}**

**}**

**interface reference**

**interface A {**

**void dis1();**

**}**

**class B implements A {**

**public void dis1() {**

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

**}**

**void dis2() {**

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

**}**

**}**

**class Demo {**

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

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

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

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

**B obj4 = (B)obj3; obj4.dis1(); obj4.dis2();**

**//B obj4 = new A();**

**}**

**}**

**Abstraction :**