**Day 1 : 2/5/2020**

**https://github.com/Kaleakash/ibmbigdatabatch.git**

**C :**

**void mno() {**

**}**

**void xyz() {**

**mno();**

**}**

**void abc() {**

**xyz();**

**}**

**int a=10;**

**void main() {**

**abc();**

**}**

**OOPs :**

**Java:**

**Reflection API :**

**object : any real world entiy**

**properties (state) --have,fields/variables**

**Person**

**Behavior ---do/does 🡪function /methods**

**Bank**

**Animal**

**Car**

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

**C + OOPs = C++**

**Abstraction, Polymorphism, Encapsulation and inheritance.**

**C**

**C++**

**Java .net**

**Bean**

**Oak**

**Nov 1995**

**James Gosling It was belong Sun Microsystem**

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

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

**+**

**Features**

1. **Java is simple : C/C++**
   1. **Pointer, Structure, memory management, virtual keyword, friend, operator overloading, data structure etc.**
2. **Compiler and Interpreter**
3. **Platform independent : write once run anywhere.**
4. **Portable :**
5. **Exception Handling**
6. **Multithreading**

**etc.**

**Java 8 :**

**Syntax class**

**class className { //Pascal Naming rules**

**fields;**

**methods;**

**}**

**Demo.java**

**class Demo {**

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

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

**}**

**}**

**To compile the program**

**javac Demo.java**

**To run the program**

**java Demo**

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

**2 types**

1. **Primitive data types: It is use to hold only value.** 
   1. **byte : 1 byte**
   2. **short : 2**
   3. **int : 4**
   4. **loong : 8**
   5. **float : 4**
   6. **double : 8**
   7. **char : 2**
   8. **boolean : 1 bit**
2. **Non primitive or reference data types: it is use to hold value as well as reference of another data types.**

**class Demo {**

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

**int a=10;**

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

**System.out.print("Welcome");**

**System.out.printf("Welcome\n");**

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

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

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

**}**

**}**

**Type casting: Converting one data type format to another data type format 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 a = 10;**

**short b = a; // type casting**

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

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

**short c = 129; //byte range -128 to 127**

**byte d = (byte)c; //type castint explicit**

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

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

**}**

**}**

**int to float family**

**implicit -🡪**

**int --------------- float**

**🡨------explicit**

**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;**

**float c = 10.10f; // type casting**

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

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

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

**}**

**}**

**Reference data types**

1. **array**
2. **classes (pre-defined or user-defined)**
3. **interfaces (pre-defined or user-defined)**
4. **enum (pre-defined or user-defined)**

**array : array is a type of reference data types which help to store homogeneous elements.**

**Syntax**

**dataType arrayName[];**

**int abc[];**

**int []abc;**

**int [] abc;**

**int[] abc;**

**Valid array declaration**

**If statements**

**Switch statement**

**Looping**

**While loop**

**Do while lop**

**For loop**

**For each loop or 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=2;i<abc.length;i++){**

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

**}**

**System.out.println("for each loop");**

**for(int a:abc) {**

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

**}**

**}**

**}**

**Creating the memory size for the array**

**Syntax**

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

**int []abc = new int[10];**

**int family –0**

**float family -0.0**

**char – space**

**boolean – false**

**String - null**

**Taking the value through keyboards in Java**

1. **Scanner class**
2. **DataInputStream class**
3. **BufferedReader class : io package**
4. **Command line arguments**

**Scanner : It is a pre-defined class which contains set of methods which help scan the value through keyboards.**

**Syntax Scanner class object**

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

**Scanner class part of util package.**

**Package :it is a collection of classes and interfaces.**

**nextByte()**

**nextShort()**

**nextInt()**

**nextFloat()**

**nextDouble()**

**nextBoolean()**

**no method as nextChar()**

**next() or nextLine() for String value.**

**Write a Java program receive employee id, name, salary through keyboards and do simple calculation as hra, da and pf where hra is 10%, da is 5% and pf 7% on salary.**

**Then display id,name,salary(grossSalary)**

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

**import java.util.Scanner;**

**class Demo {**

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

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

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

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

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

**}**

**}**

**import java.util.Scanner;**

**class Demo {**

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

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

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

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

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

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

**String name = obj.nextLine();**

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

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

**}**

**}**

**object :any real world entity**

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

**class Car {**

**String color;**

**float price;**

**int wheel;**

**void start() {**

**System.out.println("Car Start...");**

**}**

**void appliedGear() {**

**}**

**void moving() {**

**}**

**void stop() {**

**}**

**}**

**class CarTest {**

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

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

**//start();**

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

**santro.start();**

**}**

**}**

**class Car {**

**String color;**

**float price;**

**int wheel;**

**void start() {**

**System.out.println("Car Start...");**

**}**

**void appliedGear() {**

**}**

**void moving() {**

**}**

**void stop() {**

**}**

**}**

**class CarTest {**

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

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

**//start();**

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

**santro.start();**

**}**

**}**

**Types of variables or fields**

**3 types**

1. **Instance variable**
2. **Local variable**
3. **Static variable**

**Instance variable**

1. **The variable which declare outside a method including main method but inside a class is known as instance variable.**
2. **All instance variable hold default value depending upon the their data types. Like int family 0, float family 0.0, char space, boolean false and string null.**
3. **All instance variable we can use directly in all methods but the method must be non-static and it must be part of same class.**

**Local variable**

1. **The variable which declare inside a method including main method is known as local variable.**
2. **Local variable doesn’t hold default value we have to initialize.**
3. **The scope of the variable within that block or method where it declare.**

**class Car {**

**String color;**

**float price;**

**int wheel;**

**void start() {**

**int temp=123;**

**System.out.println("Car Start...");**

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

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

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

**}**

**void stop() {**

**String msg="Welcome";**

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

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

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

**}**

**}**

**class CarTest {**

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

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

**//start();**

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

**santro.start();**

**santro.stop();**

**}**

**}**

**Create two classes Employee and EmployeeTest. Inside Employee declare id, name, salary as instance variable. Then write three methods read(), calSalary() and display(). In Employee class create the Scanner class object.**

**In EmployeeTest write main method Then create two employee objects. Using 1st object call read() to receive id, name, salary then call calSalary() to do calculation (hra, da and pf must be local variable in calSalary() method). Then display id, name and salary(salary must be grossSalary).**

**2nd employee call only read and display methods.**

**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);**

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

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

**}**

**}**

**class EmployeeTest {**

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

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

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

**emp1.read();**

**emp2.read();**

**emp1.calSalary();**

**emp1.display();**

**emp2.display();**

**}**

**}**

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

**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 of that class.**

**import java.util.Scanner;**

**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.Employee();**

**emp1.display();**

**}**

**}**

**Constructor Vs Method : In the life of object if we want to execute any task only one time that type of task write in side constructor but in the life of object if we want to do the task more than one time that type of task we have to write in side a methods.**

**Parameterized constructor with this keywords.**

**this is a keyword which refer the current object. One use of this keyword when local variable or parameter variable and instance variable have same name that time local/parameter variable hide the visibility of instance variable but if we want refer the instance variable then we have to use this keyword.**

**this.instanceVariable**

**import java.util.Scanner;**

**class Cal {**

**int a,b;**

**Cal() {**

**a=10;**

**b=20;**

**}**

**Cal(int a, int b){**

**this.a=a;**

**this.b=b;**

**}**

**void setValue(int a, int b) {**

**this.a = a;**

**this.b = b;**

**}**

**void display() {**

**System.out.println(a+" , "+b);**

**}**

**}**

**class EmployeeTest {**

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

**Cal c1 = new Cal(); c1.display();**

**Cal c2 = new Cal(); c2.display();**

**Cal c3 = new Cal(1,2); c3.display();**

**Cal c4 = new Cal();**

**c4.setValue(11,22); c4.display();**

**Cal c5 = new Cal(111,222);**

**c5.setValue(1111,2222);**

**c5.setValue(1212,2121);**

**c5.display();**

**}**

**}**

**Encapsulation:**

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

**Ex :class : properties**

**Behavior**

**import java.util.Scanner;**

**class Employee {**

**String name;**

**float salary;**

**void display() {**

**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.display();**

**emp1.salary=12000;**

**emp1.display();**

**}**

**}**

**Another Encapsulation example**

**import java.util.Scanner;**

**class Employee {**

**private String name;**

**private float salary;**

**void setEmpInfo(String name, float salary) {**

**this.name = name;**

**//this.salary = salary;**

**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[]) {**

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

**//emp1.name="Ravi";**

**//emp1.salary=-12000;**

**emp1.setEmpInfo("Ravi",-12000);**

**emp1.display();**

**}**

**}**

**import java.util.Scanner;**

**class Employee {**

**private String name;**

**private float salary;**

**void setEmpInfo(String name, float salary) {**

**this.name = name;**

**//this.salary = salary;**

**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[]) {**

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

**//emp1.name="Ravi";**

**//emp1.salary=-12000;**

**emp1.setEmpInfo("Ravi",-12000);**

**emp1.display();**

**}**

**}**

**JavaBean class :**

**Rules**

1. **class must be public.**
2. **All variables must be private.**
3. **For every variable we have to write setter and getter methods.**
4. **Setter method is use to set the value and getter method is use to get the value.**
5. **Setter method name must set followed by variableName**

**setId(int id) and getter method name must be get followed by variableName getId()**

**public class Employee {**

**private int id;**

**private String name;**

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

**this.id = id;**

**}**

**public int getId() {**

**return id;**

**}**

**}**

**Java Bean class is known as pure Encapsulation class.**

**Day 2 : 2/6/2020**

**Inheritance: Inheritance is use to acquire or inherits 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**

**behaviors child class.**

**}**

**With the help of sub class object we can access its own properties and behavior as well as super class properties and behavior. But with the help of super class object we can access only its own properties and behavior**

**class A {**

**void dis1() {**

**System.out.println("A class dis1() method");**

**}**

**}**

**class B extends A{**

**void dis2() {**

**System.out.println("B class dis2() 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**

**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 directly connected to super class.**

**class A {}**

**class B extends A{}**

**class C extends A{}**

**class D extends A {}**

1. **Multiple Inheritance**

**class A {}**

**class B {}**

**class C extends A,B {}**

**but Java doesn’t support this type of inheritance. It support indirectly using interface.**

**class A {**

**void dis1() {**

**System.out.println("A class dis1() method");**

**}**

**}**

**class B extends A{**

**void dis2() {**

**System.out.println("B class dis2() method");**

**}**

**}**

**class InheritanceDemo {**

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

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

**obj1.dis1();**

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

**obj2.dis2();**

**obj2.dis1();**

**}**

**}**

**class Employee {**

**id,name,salary**

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

**read(){**

**}**

**calSalary(){**

**hra,da,pf**

**}**

**display() {}**

**}**

**class Programmer extends Employee{**

**projectName;**

**readPgr() {}**

**disPgr() {}**

**}**

**class Manager extends Employee {**

**numberOfemp;**

**readMgr() {}**

**disMgr() {}**

**}**

**class EmployeeTest {**

**Main method**

**/\***

**\*/**

**Programmer**

**Manager**

**}**

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

**Two types**

1. **Compile time polymorphism or static binding or early binding** 
   1. **Method overloading**
2. **Run time polymorphism or dynamic binding or late binding** 
   1. **Method Overriding**

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

**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(float a, float b) {**

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

**}**

**void add(String a, String b) {**

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

**}**

**}**

**class InheritanceDemo {**

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

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

**op.add(1,2);**

**op.add(1,2,3);**

**op.add(10.0f,20.0f);**

**op.add("A","B");**

**}**

**}**

**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 the classes must be use inheritance features.**

**Annotation : Meta-data means data about data. All contains lot of pre-defined annotation those annotation we can use on class level, property/fields levels and methods levels.**

**All annotation start with pre-fix @ followed by annotation name.**

**@Override :This annotation we can use on method level for only those method which you/we are going to override.**

**class Bike {**

**int a=10;**

**void speed() {**

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

**}**

**}**

**class HeroHonda extends Bike {**

**int a=20;**

**void color() {**

**int a =30;**

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

**System.out.println("Local Variable "+a);**

**System.out.println("Instance variable "+this.a);**

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

**}**

**}**

**class Pulsar extends Bike {**

**void color() {**

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

**}**

**@Override**

**void speed() {**

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

**}**

**}**

**class Tvs extends Bike {**

**void color() {**

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

**}**

**@Override**

**void speed() {**

**super.speed();**

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

**}**

**}**

**class InheritanceDemo {**

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

**HeroHonda hh = new HeroHonda();**

**hh.color(); hh.speed();**

**Pulsar pu = new Pulsar();**

**pu.color(); pu.speed();**

**Tvs tv = new Tvs();**

**tv.color(); tv.speed();**

**}**

**}**

**abstract :**

**abstract is a 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 methodName(parameterList);**

**abstract void speed();**

1. **Abstract class :**

**Syntax**

**abstract class className {**

**}**

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

1. **Whichever class extends abstract class that class must be provide the body for all abstract methods mandatory. It can ignore only if that class itself is a abstract class.**
2. **Abstract class we can’t create the object.**
3. **Abstract class can contains zero or 1 or all abstract methods.**
4. **Abstract class can contains default constructor as well as we can write parameterized constructor. ( abstract class can contains instance variables).**

**abstract class Bike {**

**abstract void speed();**

**Bike(){}**

**}**

**abstract class HeroHonda extends Bike {**

**void color() {**

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

**}**

**abstract void mailage();**

**}**

**class SuperHeroHonda extends HeroHonda {**

**void speed(){**

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

**}**

**void mailage() {**

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

**}**

**}**

**class Pulsar extends Bike{**

**void speed() {**

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

**}**

**}**

**class InheritanceDemo {**

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

**//HeroHonda hh = new HeroHonda();**

**//hh.color(); hh.speed();**

**}**

**}**

**Final keyword**

**Final is keyword we can use with variable, method and class.**

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

**final int A=10;**

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

**final void speed() {**

**}**

1. **final class :if the class is a final we can’t inherit or extends.**

**final class Demo {**

**}**

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

**(if the class is inner class then we can use static keyword but not for outer class).**

**class Outer {**

**static int a;**

**static void dis() {}**

**static class Inner {**

**}**

**}**

1. **static variable : if variable is static we can access those variable with help of class Name as well as object name.**
2. **static method :if the method is static we can call those method with the help of className as well as object name.**
3. **Inside a static method we can access only static variable**
4. **Inside a non static method we can access both type of variable ie static as well as non-static.**

**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 obj1 = new Abc();**

**System.out.println("a "+obj1.a);**

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

**}**

**}**

**class InheritanceDemo {**

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

**Abc obj1 = new Abc();**

**obj1.a=10;**

**Abc.b=20;**

**obj1.b=30;**

**obj1.dis1();**

**Abc.dis2();**

**obj1.dis2();**

**}**

**}**

**Interface: interface is a type of reference data type which help to store variable and methods. It is also known as 100% pure abstract class. Where all fields by default public static and final and all methods are public and abstract**

**Syntax**

**interface interfaceName {**

**fields;**

**methods;**

**}**

**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 A {}**

**class Demo extends A implements Abc, Xyz {**

**}**

**class can extends only one class at time.**

**interface can extends more than one interface.**

**class can implements more than one interface.**

**interface can’t extends or implement to class.**

**Super class/interface sub**

**public public**

**protected public**

**protected**

**default public**

**protected**

**default**

**private We 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 A {}**

**class Demo extends A implements Abc, Xyz {**

**public void dis1() {**

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

**}**

**public void dis2(){**

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

**}**

**}**

**class InheritanceDemo {**

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

**Demo obj = new Demo();**

**obj.dis1(); obj.dis2();**

**}**

**}**

**Difference between abstract class and interface.**

1. **Abstract class can contains final or static variable. But in interface only static and final.**
2. **Abstract class can contains normal as well as abstract method but interface contains only abstract methods.**
3. **One class can extends only one abstract class but can implements more than one interface.**
4. **Abstract class can contains default as well as parameterized constructor but interfaces doesn’t contains default constructor.**

**We can’t create the object of interface as well as abstract class.**

**1t Example of down casting with polymorphism**

**class A {**

**void dis1() {**

**System.out.println("dis1 method");**

**}**

**}**

**class B extends A {**

**void dis1() {**

**System.out.println("dis1 method - Override");**

**}**

**void dis2() {**

**System.out.println("dis2 method");**

**}**

**}**

**class InheritanceDemo {**

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

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

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

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

**A obj4 = new B(); obj4.dis1(); //run time polymorphsim**

**//obj4.dis2();**

**B obj5 = (B)obj4; //down casting**

**obj5.dis2(); obj5.dis1();**

**}**

**}**

**2nd Example**

**abstract class A {**

**abstract void dis1();**

**}**

**class B extends A {**

**void dis1() {**

**System.out.println("dis1 method - Override");**

**}**

**void dis2() {**

**System.out.println("dis2 method");**

**}**

**}**

**class InheritanceDemo {**

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

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

**A obj4 = new B(); obj4.dis1(); //run time polymorphsim**

**//obj4.dis2();**

**B obj5 = (B)obj4; //down casting**

**obj5.dis2(); obj5.dis1();**

**}**

**}**

**3rd Example**

**interface A {**

**void dis1();**

**}**

**class B implements A {**

**public void dis1() {**

**System.out.println("dis1 method - Override");**

**}**

**void dis2() {**

**System.out.println("dis2 method");**

**}**

**}**

**class InheritanceDemo {**

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

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

**A obj4 = new B(); obj4.dis1(); //run time polymorphsim**

**//obj4.dis2();**

**B obj5 = (B)obj4; //down casting**

**obj5.dis2(); obj5.dis1();**

**}**

**}**

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

**interface :**

**interface A {**

**int add(int x, int y);**

**}**

**interface B {**

**int sub(int x, int y);**

**}**

**class Server implements A,B{**

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

**return x+y;**

**}**

**public int sub(int x, int y){**

**return x-y;**

**}**

**public void ownMethod() {**

**System.out.println("own ethod");**

**}**

**}**

**class InheritanceDemo {**

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

**Server ss = new Server();**

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

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

**System.out.println(obj1.add(10,20));**

**//System.out.println(obj1.sub(10,20));**

**//System.out.println(obj2.add(10,20));**

**System.out.println(obj2.sub(10,20));**

**}**

**}**

**this() and super()**

**this() It is use to construct same class constructor or constructor chaining for same class.**

**super(): It is use to construct sub class to super class constructor.**

**this() and super() must be use inside a constructor only and it must be the first statement inside a constructor.**

**class Employee {**

**int id;**

**String name;**

**float salary;**

**Employee() {**

**//this(10);**

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

**}**

**Employee(int id) {**

**//this(1,"Ravi");**

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

**}**

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

**System.out.println("Employee(int,string)");**

**}**

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

**System.out.println("Employee(int,string,float)");**

**}**

**}**

**class Manager extends Employee {**

**Manager() {**

**super(10); //by default**

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

**}**

**}**

**class InheritanceDemo {**

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

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

**Manager mgr = new Manager();**

**}**

**}**