JAVA:-

Java code:-

Source code (.java) 🡪 byte code (.class) 🡪 object code

JDK:- JRE(it contains and library and toolkits) and JVM (its virtual machine)

Java boiler code:-

Class classname{

public static void main(String args[]){

block of code

}}

DATA types in java:-

1byte= 8bit

1>boolean 1byte (true or false)

2>byte 1byte (-128 to 127)

3> int 4 bytes(-2 billion to 2 billon)

4>long 8 bytes(-9 quintrillion to 9 quintrillion) e.g 132241413423242L

5>char 2bytes (single character/letter/ASCII) within ‘ ’

6>float 4bytes (.after 7point) e.g .312f

7>double 8bytes (.after 15point) e.g .3123d

8>String varies “string data here”

🡪 Difference between Primitive data types(default data type) and reference data types(user defined data type)

🡪USER input:

import java.util.\*;

class classname{

public static void main(String args[]){

Scanner scannername = new Scanner(System.in);

}}

🡪DATA TYPES INPUT

1.scannername.next() 🡪String

2.scannername.nextInt() 🡪Integer

3.scannername.nextFloat() 🡪Float

4..scannername.nextDouble() 🡪Double

5.scannername.next().charAt(0) 🡪Char

🡪data conversion

Datatype variablename = datatype\_wrapperclass.parseDatatype

e.g. int a= Integer.parseInt(data)

String datatype = Wrapperclass\_data\_type.toString(datatype);

🡪Math Module

1.Math.sqrt(data)

2.Math.min(array)

3.Math.max(array)

4.Math.ceil(data)

5.Math.floor(data)

6.Math.abs(data)

7.Math.pow(base,exponent)

🡪Random value

Import java.util.Random

Class classname{

Public static void main(String args[]){

Random rn = new Random();

rn.nextInt(#it starts with 0 and end-1 so its single data in it)

}}

🡪GUI in java example

import javax.swing.\*; //javax.swing.JOptionPane

class classname{

public static void main(String[] args) {

    String name= JOptionPane.showInputDialog("Enter your name");

    JOptionPane.showMessageDialog(null, "hello"+name);

    int age=Integer.parseInt(JOptionPane.showInputDialog("Enter your age:"));

    JOptionPane.showMessageDialog(null,"YOur age is"+age);

}}

**🡪Switch case in java**

class Switch\_case{

    public static void main(String[] args) {

        String day="Sunday";

        switch (day) {

            case "Sunday":System.out.println("Its sunday");break;

            case "monday":System.out.println("Its monday");break;

            default: System.out.println("No such day");

        }

        char o='a';

        switch(o){

            case 'a':System.out.println("a");break;

            default: System.out.println("Not found");

        }}}

🡪Logical operator in java

These are used to connect expressions

1>and= &&

2>or = ||

3>not = !

🡪loops

1>while (if we don’t know how much iterations will take place)

Initialization

While(condition){

Block of code

steps

}

2>for (if we know how much iterations will take place)

for(intilization,condition,steps){

block of code

}

3>do while 🡪its same as while loop but execute once even condition is not correct

Initialization;

do{

bloack of code;

step;

}while(condition);

🡪nested loop

Outter = row

Inner= column

🡪array: it’s a set of same kind of data which can keep in same variable

Datatype[] arrayname= {data,data,….n data}

e.g.

class array{

    public static void main(String[] args) {

        String[] car= {"mcalaren","BMW"};

        int[] num= {1,2,23};

        //accesing data from above array

        System.out.println(car[1]);

        System.out.println(num[0]);

        //to display all content of array using loops

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

            System.out.print(i);

        }

    }

}

🡪2D array:

It’s an array of array

class array2d {

    public static void main(String[] args) {

        //direct intialization

        String[][] car={{"camero","BMW","ferrari"},

                        {"supra","tesla","audi"},

                        {"mustang","Lamborgini","porshe"}

                    };

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

                System.out.println();

                for(int j=0;j<car[i].length;j++){

                    System.out.print(car[i][j]+"\t");

                }

            }

        //dynammic allocation

        /\*

        String[][] car = new String[3][3];

        car[0][0] = "camero";

        car[0][1] = "BMW";

        car[0][2] = "ferrari";

        car[1][0] = "supra";

        car[1][1] = "tesla";

        car[1][2] = "audi";

        car[2][0] = "mustang";

        car[2][1] = "Lamborghini";

        car[2][2] = "porsche";

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

            for(int j=0;j<car[i].length;j++){

                System.out.print(car[i][j]+" ");

            }

            System.out.println();

        }

    \*/

}

}

🡪String class

class String\_class{

    public static void main(String[] args) {

        String name="aditya das";

        //equal method (totally equal also case sensitive)

        boolean result=name.equals("ADITYADAS");

        System.out.println(result);

        //ignore-eqaul(totally equal not case sensitive)

        boolean result2=name.equalsIgnoreCase("ADITYA");

        System.out.println(result2);

        //lenght of string

        int result3= name.length();

        System.out.println(result3);

        //single letter acces from string

        char result4= name.charAt(2);

        System.out.println(result4);

        //return index of characrter in string

        //by default starts from 0

        int result5= name.indexOf('a');

        System.out.println(result5);

        //to start with specific place just add index value from where to start

        int result6= name.indexOf('a',3);

        System.out.println(result6);

        //to check whethere is empty

        boolean result7= name.isEmpty();

        System.out.println(result7);

        //to convert into uppercase

        String result8= name.toUpperCase();

        System.out.println(result8);

        //to convert into lowercase

        String result9= name.toLowerCase();

        System.out.println(result9);

        //trim empty space

        String result10= name.trim();

        System.out.println(result10);

        //replace character

        String result11= name.replace('a', 'd');

        System.out.println(result11);

        //replace string

        String result12= name.replaceAll("adi", "adiy");

        System.out.println(result12);

    }}

🡪Wrapper class:

--Provides a way to use primitive data type as reference data types

--Reference data type contains usefull method can be used with collection

Primitive data wrapper class

int Integer

boolean Boolean

char Character

double Double

1>autoboxing= converting primitive data to reference data type

2>unboxing= converting reference data to primitive data

E.g

class Wrapper\_class{

    public static void main(String[] args) {

        //wrapper class

        Boolean a=true;

        Character b='a';

        Integer c=2133;

        Double d=12.12d;

        String name="adityadas";

        //we can use many methods using wrapper class

        String result= a.toString();

        System.out.println(result);

    }

}

🡪Arraylist

Its type of array in which we can insert data and deleted in it and reference data type or wrapper class

ArrayList<reference data type> identifier = new ArrayList<reference data type>();

import java.util.ArrayList;

class Arraylist{

    public static void main(String[] args) {

        ArrayList<String> food= new ArrayList<String>();

        //inserting data

        food.add("allu-paratha");

        food.add("Dosa");

        //displaying data

        for(int i=0;i<food.size();i++){

            System.out.println(food.get(i));

        }

        //replace data or set

        food.set(0,"samosa");

        food.set(1,"panner");

        for(int j=0;j<food.size();j++){

            System.out.println(food.get(j));

        }

        //removing data

        food.remove(1);

        for(int k=0;k<food.size();k++){

            System.out.println(food.get(k));

        }

    }

}

🡪2DArraylist

Its type of array of an array in which we can insert data and deleted in it and reference data type or wrapper class

ArrayList<ArrayList<reference data type>> identifier = new ArrayList<ArrayList<reference data type>>();

import java.util.ArrayList;

class Arraylist2D{

    public static void main(String[] args) {

        //mainarray list

        ArrayList<ArrayList<String>> shoplist = new ArrayList<ArrayList<String>>();

        ArrayList<String> chocolateList = new ArrayList<String>();

        //adding items

        chocolateList.add("tiktak");

        chocolateList.add("diary milk");

        ArrayList<String> snacks = new ArrayList<String>();

        //adding items

        snacks.add("tiktak");

        snacks.add("diary milk");

        //adding the both arraylist to main array list

        shoplist.add(chocolateList);

        shoplist.add(snacks);

        //showing all arraylist

        System.out.println(shoplist);

        //get from particular datas from 2daaraylist

        System.out.println(shoplist.get(0));

        //get from particular data from particular arraylist from 2daaraylist

        System.out.println(shoplist.get(0).get(0));

    }

🡪For-each loop (mainly done in refrence datatype like e.g python for I in list)

Its traversing technique to iterate through all the elements(array) and it takes less step in order do it.

For (datatype variable: list-datatype){

System.out.println(varriable);

}

class For\_each{

    public static void main(String[] args) {

        String[] birds= {"pigeon","parrot","owl","crow"};

        for(String i : birds){

            System.out.println(i);

        }

        int[] age= {1,2,3,4};

        for(int j: age){

            System.out.println(j);

        }

    }

}

🡪Method or function:-

Its part of code when its called when we needed it.

Declaration of method

<i> Create a new class for method

<ii> within main class but outside the psvm using static method

//method class

class function\_method{

    void hello(){

        System.out.println("hello world");

    }

}

//main class

class Method{

    public static void main(String[] args) {

        //calling method from outside of class

        //instance of object created here

        function\_method a1 = new function\_method();

        //function calling

        a1.hello();

        //function calling from inside class but outside psvm using static method

        String name= "aditya das";

        hello1(name);

        }

        static void hello1(String name){

            System.out.println("hello\t"+name);

        }

    }

🡪Types of method

1>Overloading-- have within in the same class with same name method but have different parameters

2>Overriding-- have same parameters and have same method name but different class

1>e.g

class Overloading{

    void sum(int a, double b){

        System.out.println(a+b);

    }

    void sum(int a,int b){

        System.out.println(a+b);

    }

    void sum(double a,double b){

        System.out.println(a+b);

    }

    void sum(int a,int b, double c){

        System.out.println(a+b+c);

    }

}

class OverloadingMain{

    public static void main(String[] args) {

        Overloading a1 = new Overloading();

        int a=2,b=3;

        double c=3.234d;

        a1.sum(a, b);

        a1.sum(a, c);

        a1.sum(a, b, c);

    }

}

🡪printf in java:-

System.out.println(“statement %variable\_datatype”,variable/data);

Variable datatype:-

%c = character

%d= integer data type

%f= float value

%b = boolean datatype

Width or limiting data inserted in printf

Syntax

%width/sign datatype\_Character for printf in above

e.g

class Printf{

    public static void main(String[] args) {

        int i=21;

        System.out.printf("\nyour age is:",i);

        System.out.printf("\nyour age is %+d",21);

        System.out.printf("\nyour salary is %.3f",23123.123d);

        System.out.printf("\nyour salary is %,f",23123.123d);

        //width or limiting the string

        String Name = "ADITYADAS IS BRAVE BRO";

        System.out.printf("\nhello %.10S",Name);

        //

    }

}

Finale keyword in gives variable a fixed value which cann’t be change furthere in code.

e.g

class FinaleMain{

    public static void main(String[] args) {

        int v=12;

        System.out.println(v);

        v=131;

        System.out.println(v);

        final int a= 10;

        System.out.println(a);

        //it wil show error

        a=1221;

        System.out.println(a);

    }

}

OOPS in java:-

1.**Object** are real world entities by which tells about real world object with their attributes(methods) and behavior(variables).

2.**class** these are blueprints of object by which we can create number of object using it

Intialization is done for it oops

class Object{

    void hello(){

        System.out.println("hello");

    }

    Void wish(String a){

        return a;

    }

}

class ObjectMain{

public static void main(String[] args) {

    //object instatite

    Object a1 = new Object();

    Object a2 = new Object();

    String name = "ADITYA DAS";

    a1.hello();

    a1.wish(name);

    a2.hello();

    a2.wish(name);

}}

🡪Constructor in java

1.It’s a special method when a object is created or instanstited

2.No need to write code again in java first create the instace of variable and constructor name should match with class name

3.Then use this keyword to tell compiler using the current object

4.If there any method in class rather than Constructor directly use the instance variable from it no need to add arguments in it

class Constructor01{

    int weight;

    int age;

    String name;

    Constructor01(int weight,int age,String name){

        this.weight= weight;

        this.age=age;

        this.name=name;

        System.out.println("hello"+name+"your age is:"+age+"your weight is:"+weight);

    }

    void Yourname(){

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

    }

}

class ConstructorMain{

    public static void main(String args[]){

Constructor01 c1 = new Constructor01(73,19,"ADITYA das");

c1.Yourname();

}}

* Java variable in class and method

In Java, variables are classified into several types based on their scope, lifetime, and how they are declared and initialized. Here are the primary types of variables in Java:

1. Local Variables
2. Instance Variables
3. Class/Static Variables

1.Local variable:- It will only used in a method and its lifespan is same as method in program used.

2.Instance variable:- It declares in class but outside the method.

3.Static/class variable:- It declares in class but outside the method using static keyword can be accesed using the class name.

Static vs Instance variable

* Constructor with different parameter

We can have same concept like method overloading here all rules are same and different parameter

class Overloadded\_Constructor{

    int age;

    int weight;

    String name;

    int salary;

    Overloadded\_Constructor(int age,int weight,String name){

        this.age= age;

        this.weight= weight;

        this.name = name;

    }

    Overloadded\_Constructor(int age,int weight,String name,int salary){

        this.age= age;

        this.weight= weight;

        this.name = name;

        this.salary = salary;

    }

}

class Overloadded\_ConstructorMain{

    public static void main(String[] args) {

        Overloadded\_Constructor s1= new Overloadded\_Constructor(12, 012,"aditya das");

        System.out.println("Here is your all details");

        System.out.println(s1.age);

        System.out.println(s1.name);

        System.out.println(s1.weight);

        System.out.println("New constructor with different parameter");

        Overloadded\_Constructor s2= new Overloadded\_Constructor(12, 012,"aditya das",21221222);

        System.out.println(s2.age);

        System.out.println(s2.name);

        System.out.println(s2.salary);

        System.out.println(s2.weight);

}}

🡪To String method

It’s a special method that all objects inherit that returns a string that “text form represent” can be used both implicitily and explicitly..

class Conversion\_toSting{

    String make="Ford";

    String modal="Mustang";

    String color = "red";

    int year =2021;

    //changing the tostring method

    public String toString(){

        return make+"\n"+modal+"\n"+color+"\n"+year;

    }

}

class Conversion\_toStingMain{

public static void main(String[] args) {

    Conversion\_toSting car = new Conversion\_toSting();

    //tostring in explicity

    System.out.println(car.toString());

    //tostring method in implicity

    System.out.println(car);

}

}

🡪Array of an object

We can declare array in java using objects first we need to declare another class for array using constructor then initialize the class in main class then assign value as per given way then we can get the value using **varaiblename.name** (here variable name is the name for instance of an object)

class Object\_array{

    String name;

Object\_array(String name){

this.name= name;

}

}

class Object\_arrayMain{

    public static void main(String[] args) {

      //declaring array size using the class

        Object\_array[] s1 = new Object\_array[3];

        //assigning value

        Object\_array food1= new Object\_array("pizza");

        Object\_array food2= new Object\_array("burger");

        Object\_array food3= new Object\_array("samosa");

        //it can be two way

        //1

        s1[0]= food1;

        s1[1]= food2;

        s1[2]= food3;

        //2

        Object\_array[] s2=  {food1,food2,food3};

        //accesing the value

        System.out.println(s1[0]); //it will give address

        System.out.println(s1[0].name); //it will return the value

    }

}

🡪Objectpass:-

In this we can pass constructor to other class function in order to show working to gather.

But remember when calling values from one to other place it mark as an reference to that object in order to deal with use .name to gets its value and .id to show its id

class Objectpass{

void park(car c1){

    System.out.println("The"+c1.name+"parked succesfully");

}

}

class car{

    String name;

    car(String name){

        this.name=name;

    }

}

class ObjectpassMain{

    public static void main(String[] args) {

        Objectpass ob1 = new Objectpass();

        car c1 = new car("Supra");

        car c2 =new car("BMW");

        ob1.park(c1);

        ob1.park(c2);

    }

}

* Static keyword:-

In java static keyword is used in both class and object or variable.If we assign a static keyword front of any identifier then it’s one copy can be used in any part of program.

And its value can be called by nameofclass.variable name without need to create the instance of the object.

class Static{

    String name;

    static int i;

    Static(String name){

        this.name = name;

        i++; //increase the age object creates intially it will return 1 cause its start with 0

        System.out.println("your name is"+name+"and your age is"+i);

    }

    static void display(){

        System.out.println("your age is:"+i);

    }

}

class StaticMain{

    public static void main(String[] args) {

        Static c1 = new Static("ADITYA DAS");

        Static a1 = new Static("ADITYA DAS");

        c1.display();

        a1.display();

    }

}

🡪Inhertance:-

When a class inherient all the property of other class is called inheritance.

In java only two types of inheritance can be done:-

**1>single-level[inheritance**

**2>multi-level-inhertance**

In order to inherit the class property we need to use **extends** keyword we use the follow the following syntax:-

Class child\_class\_name extends parent\_name\_class{

Block of code

}

e.g. both single level and multi level inheritance

class Vehical{

    double speed;

    void start(){

        System.out.println("THis is vechcle is stated");

    }

    void stop(){

        System.out.println("This vehical is stopped");

    }

}

class Truck extends Vehical{

void light(){

    System.out.println("showing light");

}}

class Bicycle extends Truck{

}

class InheritanceMain{

public static void main(String[] args) {

 Vehical v1 = new Vehical();

 v1.start();

 v1.stop();

 //single level inheritance

 //truck class inherit the property of Vehical class

 Truck t1 =new Truck();

 t1.light();

 t1.start();

 t1.stop();

 System.out.println(t1.speed);

 //multilevel-inheriance

Bicycle b1 = new Bicycle();

b1.start();

b1.stop();

}}

There is other types of inheritance:-

**3>multiple inheritance**

**4>Hybrid inheritance**

**🡪**Method overriding

It have same method with same parameter but in different class

class Man{

    void voice(){

        System.out.println("blah!");

    }

}

class Dog{

    void voice(){

        System.out.println("yo bro!");

    }

}

class MethodOverdingMain{

    public static void main(String[] args) {

Man m1 =new Man();

Dog d1 = new Dog();

m1.voice();

d1.voice(); }}

🡪Super keyword

This is has same function as this keyword and generally use in inheritance by using it we decrease the code line for constructor cause we already declare the part in other class by using super keyword in the class constructor we are just calling the contructor function of other class to this class.

e.g.

class person{

    String name;

    int age;

    person(String name,int age){

        this.name=name;

        this.age=age;

    }

}

class SuperHero extends person{

    String power;

    SuperHero(String name,int age,String power){

        //already declared in person class so using super keyword here

    super(name,age);

    this.power= power;

    System.out.println("You are"+name+"your age is"+age+"and your power to save this world is"+power);

}

}

class SuperMain{

    public static void main(String[] args) {

 SuperHero s1 =new SuperHero("Iron-man",36,"intelligence");

    }

}

🡪Abstruction:-

It refers to hide some of the function which is done in back of a program whih is irrelavent to user ,cause user need meaningfull output from it.

In java we can achieve abstruction by using abstruct class or abstruct method using using the keyword **abstruct.**

In java abstruct class only contain the method and variable but can’t have body for it , in order give the body the abstruct class treat as parent class and transfer its property to other class.

Abstruct class can’t be initiated but they can have subclass with it.

abstract class Vehical01{

    abstract void go();

    abstract void stop();

}

class Car01 extends Vehical01{

    void go(){

        System.out.println("Vechical is going");

    }

    void stop(){

        System.out.println("car is stopped");

    }

}

class AbstructMain{

    public static void main(String[] args) {

        Car01 c1 =new Car01();

        c1.go();

        c1.stop();

    }

}

🡪Encapsulation / private variable value access:-

Binding of variable and methods in single form,these are the attributes of class will be hidden or private.

Here we can access only through special methods called getters and setters.These all are done in the same class of encapsuled class after the constructor.

Variable in getter and setter are equal to the constructor variable name

1.Getter:- to get value of private variable we use getter method its syntax is

datatype functionname(datatype variable){

return datatype variable;

}

2.Setter:- It helps to set the value from main class to encapsuled class , but a sweeter function should be define in the Encapsuled class.Syntax:

Void functionname(datatype variable){

This.variable = variable; }

class Encapsulation\_car{

    private String make;

    private String modal;

    private int year;

    Encapsulation\_car(String make,String modal,int year){

        this.year=year; //this.set\_make=make;

        this.make=make;//this.set\_model=model;

        this.modal=modal;//this.set\_year=year;

    }

    //in order to take value from above class we will use special attributes or methods which are setter and getters

    //getter method

    String get\_make(){

        return make;

    }

    String get\_modal(){

        return modal;

    }

    int get\_year(){

        return year;

    }

    //setter method

    void set\_make(String make){

        this.make=make;

    }

    void set\_modal(String modal){

        this.modal=modal;

    }

    void set\_year(int year){

        this.year=year;

    }

}

class EncapsulationMain{

    public static void main(String[] args) {

        Encapsulation\_car c1= new Encapsulation\_car("LAmborgini", "Z1", 2020);

        //while we try get the value from above class it will show error cause its private

        // System.out.println(c1.year);

        //by using getter and setter we can access the value from private

        System.out.println(c1.get\_make());

        c1.set\_make("Farrari");

        System.out.println(c1.get\_make());

  }

}

🡪Copy objects

It’s way to copy the data from one onject instance to other object instance. We can easily acheive this by:

1. variable A =variable B
2. variableA.copy(B)

e.g. for all examples

class Encapsulation\_car1{

    private String make;

    private String modal;

    private int year;

    Encapsulation\_car1(String make,String modal,int year){

        this.year=year;

        this.make=make;

        this.modal=modal;

    }

    String get\_make(){

        return make;

    }

    String get\_modal(){

        return modal;

    }

    int get\_year(){

        return year;

    }

    //setter method

    void set\_make(String make){

        this.make=make;

    }

    void set\_modal(String modal){

        this.modal=modal;

    }

    void set\_year(int year){

        this.year=year;

    }

}

class ObjectCopy{

    public static void main(String[] args) {

        Encapsulation\_car1 c1= new Encapsulation\_car1("LAmborgini", "Z1", 2020);

        Encapsulation\_car1 c2= new Encapsulation\_car1("Farrari", "Z12", 2024);

        //c1.copy(c2);

        //c1=c2;

        System.out.println(c1);

        System.out.println(c1.get\_make());

        System.out.println(c1.get\_modal());

        System.out.println(c1.get\_year());

        System.out.println();

        System.out.println(c2);

        System.out.println(c2.get\_make());

        System.out.println(c2.get\_modal());

        System.out.println(c2.get\_year());

    }

}

🡪Interface in java:

It’s a template that can be applied class. Its like inheritance but a class can have more than one one interface.

An interface no need to be in an class and its doesn’t have any body.It only contains the variable and function name in it.

To declare a interface in we use the syntax:

interface interfacename{

function name(datatypes variable);

function name(datatypes variable);

…

}

A class can have ‘n’ of interface with it so to apply on the interface on class we use **implements** keyword

**And then declare interface function in it and also all function should start with public access modifier in order to use the interface.**

interface sum{

    void add(int a,int b);

}

interface subtraction{

    void sub(int a,int b);

}

interface divide\_mul{

    void mul(int a,int b);

    void div(int a,int b);

}

class Airthmatic implements sum,subtraction,divide\_mul{

    public void add(int a,int b){

        System.out.println(a+b);

    }

    public void sub(int a,int b){

        System.out.println(a-b);

    }

    public void mul(int a,int b){

        System.out.println(a\*b);

    }

    public void div(int a,int b){

        System.out.println(a/b);

    }

}

class InterfaceMain{

    public static void main(String[] args) {

        Airthmatic a1 = new Airthmatic();

        a1.sub(3, 2);

        a1.add(2,3);

        a1.mul(2,30);

        a1.div(21, 210);

    }

}

🡪Polymorphism:-

Poly=”many”and Morph=”form”

Means a function can be used in different ways at different types.Its an ability to identify as more than one type.

As we learn before about overriding method its same as but here we use both concept of inheritance and array using for each loop.

class Vehicals{

    void go(){

    }

}

class car extends Vehicals{

void go(){

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

}

}

class Bicycle extends Vehicals{

    void go(){

        System.out.println("Bicycle begains moving");

    }

}

class Boat extends Vehicals{

    void go(){

        System.out.println("Boat begains moving");

    }

}

class PolymorphismMain{

    public static void main(String[] args) {

        car v1= new car();

        Bicycle a1= new Bicycle();

        Boat b1 = new Boat();

        //duck typing like python

        Vehicals[] t1 = {v1,a1,b1};

        for(Vehicals x: t1){

            x.go();

        }

    }}

🡪Dynamic polymorphism:-

Dynamic means after compilation or during runtime of a program and polymorphism means many forms.

Its have same concept as polymorphism but here we take user input indeed to this and we have to also declare that class which give property to other child class in main method.

import java.util.Scanner;

class Vehicals{

    void sound(){

    }

}

class car extends Vehicals{

    void sound(){

        System.out.println("pi pi!");

    }

}

class aeroplane extends Vehicals{

    void sound(){

        System.out.println("whisssssss!");

    }

}

class Dynammic\_polymorphism{

    public static void main(String[] args) {

        //declaring parent class in order to choose according user

        Vehicals a1;

        //object is being intiated for child class here

        car car1 = new car();

        aeroplane aero = new aeroplane();

        Scanner sc =new Scanner(System.in);

        System.out.println("Enter the your choose between car and aeroplane:\nfor aerolplane(press1)\nfor car(press2)");

        int i=sc.nextInt();

        if(i==1){

            a1 =car1;

            a1.sound();

        }

        else if(i==2){

            a1 =aero;

            a1.sound();

        }

        else

         System.out.println("INvalid choose");

    }

}

Exception Handling:-

Exception:- these are problem raised during execution or running time of program.

In java we can handle Exception in form of:-

try{

Risky code goes here

}

catch(Exception\_type e){

Exception solving code or message for exception

}

finally{

clean code;

}

From above either try block executed or catch block but finaly block definitely executed.

1. ArithmeticException: Demonstrated by dividing by zero.

2. ArrayIndexOutOfBoundsException: Demonstrated by accessing an invalid index in an array.

3. ClassCastException: Demonstrated by attempting an invalid cast.

4. IllegalArgumentException: Demonstrated by passing an invalid argument to Thread.sleep.

5. NegativeArraySizeException: Demonstrated by trying to create an array with a negative size.

6. NullPointerException: Demonstrated by trying to access a method on a null object.

7. NumberFormatException: Demonstrated by trying to convert a non-numeric string to an integer.

User level Exception:

We can also define user level exception by creating another class for Exception using **extends Exception** in that other class.

Then we need to use try-catch block in which try block we use in form of

try{

**throw new class\_declared\_for\_exception(arguments);**

}

Then we define the class name in

catch(**User\_defined\_class\_name e**){

code block

}

class AgeRestriction extends Exception{

    int age;

    AgeRestriction(int age){

        this.age=age;

        System.out.println(age+"\tis not possible for vote!");

    }

}

class User\_ExceptionMain{

    public static void main(String[] args) {

        int age=10;

        try{

        if (age<18 || age>100){

            throw new AgeRestriction(age);

        }

    }

    catch(AgeRestriction e){}

    }

}

🡪File Handling java:-

It includes creation of file and read and write operation associate with it.

1.we need to import the java.io.File module which contain whole File operations in it.

2.To do operation on file we need to give the path of the in objectinstance.

e.g. File objectname = new File(“Filepath”);

3.To check whether the file existes objectname.exists() in if-else block statement.

4.To find the absulate position of file we use objectname.getAbsoultePath() in System.out.println()

5.To Delete a file we use e.g. objectname.delete();

//we will import java.io.File which contain all operation which we require

import java.io.File;

class FirstMain{

public static void main(String[] args) {

    //providing the path of the file in here

    File f1= new File("ReadME.txt"); //if its in different location then provide whole path

   //to cheacck whether its existe ot not

   if(f1.exists()){

    System.out.println("Its exists");

    //it help return the path for file

    System.out.println(f1.getAbsolutePath());

   }

   else{

    System.out.println("It doesn't exists");

   }

   //deleting the file

   f1.delete();

}}

1>File writing:-

1.first import java.io.FileWriter

2. throws IOException in after psmv

3.Then creates objectInstance which will open the file

4.To wite in it we use objectname.write(“string goes here”)

5.After doing all writing operations in it then close as it -> objectname.close()

Function or method it are:-

.write(“data”) ; data can be:- string,int,char

import java.io.FileWriter;

class FileWriterMain{

    public static void main(String[] args) {

        try{

            FileWriter f1 =new FileWriter("ReadME.txt");

            //it will write

            f1.write("Hello my name is Aditya");

            //it will add new text

            f1.append("Hello adi0");

            //it will ensure data is being is stored

            f1.flush();

            //it will close

            f1.close();

        }

        catch(Exception e){

            System.out.println("NO such file exists!");

        }

    }

}

2>File reading:-

1.first import java.io.FileWriter

2.Then creates objectInstance which will open the file

3.throws IOException in after psmv

import java.io.\*;

class FileReaderMain{

    public static void main(String[] args) throws IOException {

        try{

            FileReader f1 = new FileReader("ReadMe2.txt");

            int i = f1.read();

            while(i!=-1){

                System.out.println((char)i);

                i=f1.read();

            }

        }

        catch(Exception e){

            System.out.println("Not such file exists!");

        }

    }

}

3>Creating a new file

import java.io.\*;

class Creating\_file{

    public static void main(String[] args) throws IOException {

        File f1 =new File("Creating\_file.txt");

        //creating a new file

        f1.createNewFile();

    }

}

🡪Access Modifier

It adds layer of security to over our program in which we can access the class or methods according to it.

Packages is collection of classes and class is collection of code

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MODIFIER | class | Same Package | Inheritance | Different Package |
| public (default) | yes | Yes | Yes | Yes |
| protected | yes | Yes | Yes | No |
| no modifier or default | yes | Yes | No | No |
| private | yes | No | No | No |

1.Private Access Modifier:-

It can declare in the class with **private** keyword but it can’t be inherent and called in other class or main class.

class Private{

    private int i1=2;

    int i2=3;

    private void sum(){

        System.out.println(i1+i2);

    }

    void sum1(){

        System.out.println(i2);

    }

    //getting private value indirectly

    int i1G(){

        return i1;

    }

}

class PrivateMain{

public static void main(String[] args) {

    Private o1 = new Private();

    //it can be called

    System.out.println(o1.i2);

    //it can't be called

     System.out.println(o1.i1);

    //THis method can't be called

    o1.sum();

    //getting value of i1 indirectly

    System.out.println(o1.i1G());

}

}

2.Default or No Modifier:-

It doesn’t need any keyword to define its access modifier we can call it within the same package or folder.

So you don’t need to use package keyword here

Note:-we can use package keyword one time a code but we can use ‘n’ of times import

Program1:-

class Default{

    String name="ADITYA";

    void display(){

        System.out.println("hello from Defult class");

    }

}

Program2:-

class DefaultMain{

    public static void main(String[] args) {

        Default o1 = new Default();

        System.out.println(o1.name);

        o1.display();

    }

}

3.Protected class:-

It uses **protected** keyword to define its access modifier we can call it within the same package or other package using inheritance.

🡪So you don’t need to use package keyword for same package but if we use different java code for different package then we need to use package keyword.

A)With same class:-

class Protected{

    protected String name="aditya das";

    protected void display(){

        System.out.println("Protected message is here");

    }

}

package AccessModifier;

class ProtectedMain{

    public static void main(String[] args) {

        Protected o1 =new Protected();

        o1.display();

        System.out.println(o1.name);

    }

}

B) with inheritance:-

package Protected;

import AccessModifier.Protected;

class ProtectedMain1 extends Protected{

    public static void main(String[] args) {

        Protected o1 =new ProtectedMain();

        System.out.println(o1.name());

        o1.display();

    }

}

4>Public Modifier:-

It uses **public** keyword to define its access modifier we can call it within the same package or other package without using inheritance.

A)Within the same package

class Protected{

    protected String name="aditya das";

    protected void display(){

        System.out.println("Protected message is here");

    }

}

package AccessModifier;

class ProtectedMain{

    public static void main(String[] args) {

        Protected o1 =new Protected();

        o1.display();

        System.out.println(o1.name);

    }

}

B) with different package without using inheritance:-

Program 1:-

class Protected{

    protected String name="aditya das";

    protected void display(){

        System.out.println("Protected message is here");

    }

}

Program 2:-

package Public;

import AccessModifier.PublicMain;

class PublicMain2{

    PublicMain o1 =new PublicMain();

    o1.display();

}