**Day 1 :**

**20-09-2020**

Programming language

Software

System software OS

Application software

C

C++

Java

Python

C#

JavaScript

Etc

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

object : object is any real world entity.

Properties or state -🡪 have 🡪name, age, height, color,

Person

Behavior -🡪do/does 🡪 teaching, sleeping, taking etc

Place

Bank

Car

Mobile

Account

Employee

Class : blue print of object or template of object.

Class syntax

class ClassName {

fields/ variables

functions / methods

}

class Demo {

public static void main(String args[]) {

System.out.println("welcome to java");

}

}

We have to same the program with ClassName.java

Java 1.0 Java 18

Java 8, 11 and 15 etc

Java

JDK : Java Development kit

JRE : Java run time environment

Variable : Variable is name which hold value and value can change during the execution of program.

a=10

a=30;

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

2 types

1. Primitive data type
2. Non primitive data type or reference data type.

Primitive data type : it is use to store only value.

8 types

1. byte 1 byte
2. short 2 byte
3. int 4 byte
4. long 8 byte

without decimal (number type)

1. float 4 byte
2. double 8 byte

with decimal (number type)

1. char 2 byte : any single character
2. boolean 1 bit : true or false.

datatype variableName;

dataTyep variableName = value;

int a;

int b=100;

**Day 2:**

**22-09-2020**

Operator : Operator is use to do mathematical operation on variable.

Arithmetic Operator : +, -, \*, /, % (remainder)

Conditional operator : >, <, >=, <=, ==, !=

Logical operator : &&, ||, !

&& : both the condition must be true then result is true

|| : any condition must be true then result is true

! : negation

Assignment operator : = int a=10;

a+b=c+d; error

a+b == c+d;

increment and decrement :

variable++ n++ increment the value by one

variable-- n-- decrement the value by one

int a=10;

a=a+1;

or

a++

a=a-1;

or

a--

pre increment and post increment or pre decrement and post decrement

a=10;

a++ post increment a-- post decrement

++a pre increment --a pre decrement

If we use the variable alone then there is no difference between pre and post increment or decrement.

But if we use this variable with increment or decrement in expression or assigning to another variable then pre an post behave differently.

Pre : first increment and then we can use the value

Post : first use then increment the value.

Ternary operator : condition ? true : false;

If statement : it is use to execute set of statement base upon condition.

1. Simple if

If(condition) {

}

1. If else

If(condition) {

}else {

}

1. If else if

If(condition) {

}else if(condition) {

}else if(condition) {

}else {

}

1. Switch statement

Syntax

switch(variableName) {

case value1: block1;

break;

case value2: block2;

break;

case value3: block3;

break;

default : wrong block

break;

}

switch, case, break and default are keywords.

Variablename must be type of int, char or string.

Taking the value through keyboards in Java

1. Using Scanner class
2. DataInputStream
3. BufferedReader
4. Command Line interface

Using Scanner class

We have to create the Scanner class object.

Scanner is a pre-defined class which provided set of methods which help to take the value through keyboards.

Syntax

Scanner obj = new Scanner(System.in);

Scanner is pre-defined class part of utill package. Package is a collection of classes and interfaces.

#include<stdio.h>

printf(“”);

Scanner obj = new Scanner(System.in);

obj.nextByte();

obj.nextShort();

obj.nextInt();

obj.nextLong();

obj.nextFloat();

obj.nextDouble();

obj.nextBoolean();

no nextChar()

String : String is pre-defined class which help to store more than one character.

**Day 3:**

**24-09-2020**

Looping : looping is use to execute set of statement again and again till condition become false.

While loop

Do while loop

For loop

For each or enhanced loop

Initialization : start and end position

Condition : must be true

Body of the loop

Increment or decrement

While loop

Initialization i=1,n=10;

while(i>=n) { entry loop

do the task

increment or decrement , i++ or n--

}

Do while loop

Initialization i=1,n=10;

do{

do the task

increment or decrement , i++ or n--

} while(i<=n); exit loop

For loop

Syntax

1 2 4

for(initialization ; condition ; increment / decrement ){

body for the for loop 3

}

1st block only once then it check the condition . It will execute 2, 3 and 4 till condition become false.

Display numbers in reverse order.

Display only even or odd number

Sum of natural numbers.

Sum of even number or odd number.

array : array is known a reference data type which is use to store more than one value of same type.

int a=10;

a=20;

a=30;

int abc[];

int []abc;

int [] abc;

int[] abc;

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

array store using index position and index position start from zero.

abc[0];

abc[1];

abc[5];

int xyz[]=new int[10]; array created with memory size 10.

xyz[0]; default vale is zero.

xyz[0]=100;

xyz[1]=200;

for each loop

for(datatype variableName: arrayname) {

}

Sum of array elements. Take the value through keyboards.

Sum of array of even or odd elements. Take the value through keyboards.

Store n number of employee details like id, name, salary.

Store n number of employee details like id, name, salary, designation

If desg is Manager 5000 bonus

If des is Develop 3000 bonus

else

1500

int num[][]={{10,20,30},{40,50,60},{70,80,90}}

**Day 4 :**

**27-09-2020**

OOPs (Object Oriented Programming System).

object : object is any real world entity or object is an instance of a class.

Properties or state🡪 have

Person

Behavior -🡪 do/does

Bank

Car

Account

Employee

class : class is blue print of object or class is template of object or class is user defined data type or reference data type which help describe the object.

object creation

className referenceName = new ClassName();

function or methods syntax

returnType methodName(parameterList) {

}

No passing parameter and no return type. Void is keyword we use with method for no return type.

void display() {

System.out.println(“Hello”);

}

Method with passing parameter but no return type

void add(int x, int y) {

int sum = x+y;

System.out.println(sum);

}

Method passing parameter and return value.

String sayHello(String name) {

// logic

return “Welcome user ”+name;

}

// no passing parameter but return value

int getNumber() {

// logic

return 100;

}

User defined Object Example

class Car {

int wheel;

float price;

String color;

void start() {

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

}

void appliedGear() {

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

}

void moving() {

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

}

void stop() {

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

}

}

class App {

public static void main(String args[]) {

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

Car innova = new Car();

innova.start();

innova.stop();

System.out.println("Innova Santro");

Car santro = new Car();

santro.start();

santro.appliedGear();

santro.stop();

}

}

Assignment : Create any user defined object with class.

Types of variable or fields

In Java variable are divided into 3 types

1. Instance variable
   1. The variable which declare inside a class but outside a method is known as instance variable.
   2. Instance variable hold default value base upon their data type like

int family 🡪 0

float family ->0.0

boolean 🡪 false

char 🡪space

String 🡪 null

* 1. Instance variable we can use directly inside all methods but that method must be non static and part of name class.

1. Local variable
   1. The variable which declared 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 where it declared.
2. Static variable

**Local variable and instance variable Example**

class Car {

int wheel;

float price;

String color;

void displayCarInfo() {

int temp=0;

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

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

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

System.out.println("Temp "+temp);

}

void display(){

String msg="Welcome";

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

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

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

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

System.out.println("Temp "+temp);

}

}

class App {

public static void main(String args[]) {

}

}

User defined object with instance variable and behavior

class Car {

int wheel=4;

double price=2000000;

String color="Gray";

void displayCarInfo() {

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

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

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

}

}

class App {

public static void main(String args[]) {

Car innova = new Car();

innova.displayCarInfo();

}

}

Creating more than one object of Car class and setting the value from main method and display the information

class Car {

int wheel;

double price;

String color;

void displayCarInfo(String carDetails) {

System.out.println(carDetails);

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

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

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

}

}

class App {

public static void main(String args[]) {

Car innova = new Car();

innova.wheel = 4;

innova.price = 2500000;

innova.color = "Grey";

innova.displayCarInfo("Innova Car Details");

Car ertiga = new Car();

ertiga.wheel = 4;

ertiga.color="White";

ertiga.price = 1300000;

ertiga.displayCarInfo("Ertiga Car Details");

}

}

Create Product class with instance variable as pid,pname,price and displayProductInfo method.

Inside main method create the two product class object and set the value and display the value.

User defined object with instance property, setter method, business method and display method

class Operation {

int a, b, sum;

void setValue(int x, int y) {

a=x;

b=y;

}

void add(){

sum = a+b;

}

void display() {

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

}

}

class App {

public static void main(String args[]) {

Operation op1 = new Operation();

op1.display();

Operation op2 = new Operation();

op2.add();

op2.display();

Operation op3 = new Operation();

op3.a=100;

op3.b=200;

op3.display();

Operation op4 = new Operation();

op4.a=1;

op4.b=2;

op4.add();

op4.display();

Operation op5 = new Operation();

op5.setValue(10,20);

op5.add();

op5.display();

}

}

**Day 5 :**

**29-09-2022**

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

1. **Constructor have same name as class itself.**
2. **Constructor no need type not even void also.**
3. **Constructor no need to call it will call automatically when we create the memory.**

**// Empty constructor**

**class Employee {**

**Employee() {**

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

**}**

**void display() {**

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

**}**

**}**

**class ConstructorDemo {**

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

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

**//emp1.display();**

**}**

**}**

**If local or parameter variable and instance variable have same name. local or parameter variable hide the visibility of instance variable.**

This keyword. This keyword is use to refer the current object.

If local variable and instance variable have same name if you want to refer to instance variable then we have to use this.instancevariableName;

Constructor with parameter with this keyword

class Employee {

int id;

String name;

float salary;

Employee() {

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

id =123;

name = "Unkown";

salary =8000;

}

/\*Employee(int id1, String name1, float salary1) {

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

id = id1;

name = name1;

salary = salary1;

}\*/

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

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

this.id = id;

this.name = name;

this.salary = salary;

}

void setValue(int id, String name, float salary) {

this.id = id;

this.name = name;

this.salary = salary;

}

void display() {

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

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

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

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

}

}

class ConstructorDemo {

public static void main(String args[]) {

Employee emp1 = new Employee();

Employee emp2 = new Employee();

Employee emp3 = new Employee(1,"Ravi",14000);

Employee emp4 = new Employee();

emp4.setValue(2,"Ajay",18000);

emp4.setValue(3,"Balaji",22000);

emp1.display();

emp2.display();

emp3.display();

emp4.display();

}

}

Encapsulation : binding or wrapping data (variable) and code (methods) in a single unit is known as Encapsulation.

class :

class Employee {

private int id;

private String name;

private float salary;

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

this.id = id;

this.name = name;

if(salary<0){

this.salary = 8000;

}else {

this.salary = salary;

}

}

void setSalary(float salary){

this.salary = salary;

}

void display() {

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

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

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

}

}

class EncapulationDemo {

public static void main(String args[]) {

//display();

//id=100;

Employee emp1 = new Employee(1,"Ravi",12000);

//emp1.id=100;

//emp1.name="Ravi";

//emp1.salary = -12000;

//emp1.setEmployeeInfo(1,"Ravi",12000);

//emp1.setSalary(14000);

emp1.display();

}

}

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

class OldClass { // super class or base class or parent class.

properties

behavior

}

class NewClass extends OldClass{ // sub class or derived class or child class.

property

behavior

}

Inheritance Example

class A {

void dis1() {

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

}

}

class B extends A{

void dis2(){

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

}

}

class InheritanceDemo {

public static void main(String args[]) {

A obj1 = new A();

obj1.dis1();

B obj2 = new B();

obj2.dis2();

obj2.dis1();

}

}

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{ }

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

class A{ }

class B { }

class C extends A,B { } :Wrong in Java.

In Java we can achieve multiple inheritance using Interface.

**Day 6 :**

**06-10-2022**

OOPs relationship

is a relationship

has a relationship

is a relationship we use while using inheritance.

Has a relationship we use. Inside one class object we will create another class object.

class Employee {

id,name,salary

Address add = new Address();

}

class Manager extends Employee{

numberOfEmp

}

class Developer extends Employee{

projectName;

}

class ProjectManager extends Manager{

clientName

}

class Address {

city,state

}

Manager/Developer Is a Employee

Employee as a relationship Address

**Is a relationship and has a relationship**

import java.util.Scanner;

class Employee {

private int id;

private String name;

private float salary;

Scanner sc = new Scanner(System.in);

Address add = new Address();

void readEmp() {

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

id = sc.nextInt();

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

name= sc.next();

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

salary = sc.nextFloat();

}

/\*

calculateSalary hra = 10%, da = 5% and pf = 7%

grossSalary = hra, da, pf

\*/

void disEmp() {

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

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

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

}

}

class Manager extends Employee{

private int numberOfEmp;

void readMgr() {

readEmp(); // calling employee class method

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

numberOfEmp = sc.nextInt();

add.readAdd(); // calling address class method

}

void disMgr() {

disEmp();

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

add.disAdd();

}

}

class Developer extends Employee {

private String projectName;

void readDev() {

readEmp();

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

projectName = sc.next();

add.readAdd();

}

void disDev() {

disEmp();

System.out.println("ProjectName is "+projectName);

add.disAdd();

}

}

class Address {

private String city;

private String state;

Scanner sc = new Scanner(System.in);

void readAdd() {

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

city = sc.next();

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

state = sc.next();

}

void disAdd() {

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

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

}

}

class EmployeeTest {

public static void main(String args[]) {

Manager mgr = new Manager();

Developer dev = new Developer();

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

mgr.readMgr();

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

dev.readDev();

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

mgr.disMgr();

System.out.println("Developer detils");

dev.disDev();

}

}

Has a relationship

1. Association
2. Aggregation
3. Composition

class A {

B obj1 = new B(); zero, 1 or many

}

class B {

A obj2 = new A(); zero, 1 or many

}

class Employee {

Address add = new Address();

}

class Address {

}

Aggregation is a type of association but it is known as weak association.

class Student {

StudentHistory sh = new StudentHistory();

}

class StudentHistory {

}

Composition is a type of association but it is known as strong association.

Polymorphism : One name many forms or many implementation

Compile time polymorphism

: Method overloading or static binding or early binding

Run time polymorphism

: Method overriding or dynamic binding or late binding

Method overloading : the method have same name but different parameter list. Number of parameter or type of parameter must be different. We can achieve method overloading in same class.

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

To achieve overriding we need inheritance.

Method overloading example

class Operation {

void add(int x, int y) {

System.out.println(x+y);

}

void add(float x, float 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 MethodOverloading {

public static void main(String args[]) {

Operation op = new Operation();

op.add(1,2,3);

op.add(1,2);

op.add(10.10f,20.20f);

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

}

}

class Bike {

void speed() {

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

}

}

class Honda extends Bike{

void color() {

System.out.println("Black");

} // re-usability 60

}

class Pulsar extends Bike{

void color() {

System.out.println("Red");

}

void speed() {

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

}

}

class Tvs extends Bike{

void color() {

System.out.println("Black");

}

void speed() {

super.speed();

System.out.println("10km/hr"); // merge the code. 60 + 10

}

}

class MethodOverriding {

public static void main(String args[]) {

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

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

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

}

}

**Day 7 :**

**08-10-2022**

abstract keyword

1. abstract is a keyword we can use with method and class but not with variable.
2. abstract method : the method without body or without curly braces or incomplete method is known as abstract method.

abstract returnType methodName(parameterList);

1. if class contains abstract method that class must be declare as abstract.

abstract class Demo {

}

1. Whichever class extends abstract class that class must be provide the body for all abstract method mandatory. That class can ignore if that class itself is an abstract.
2. Abstract class we can’t create the object.
3. Abstract class can contains normal as well as abstract method. Like it can contains 0 or 1 or many.

abstract class Bike {

void mailage(){

System.out.println("40 km/l");

}

abstract void speed();

}

abstract class Honda extends Bike{

void color() {

System.out.println("Black");

}

void mailage(){

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

}

}

class Activa extends Honda {

void speed() {

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

}

}

class Pulsar extends Bike{

void color() {

System.out.println("Red");

}

void speed() {

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

}

}

class AbstractConcept {

public static void main(String args[]) {

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

Activa aa =new Activa(); aa.speed(); aa.color(); aa.mailage();

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

}

}

final keyword

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

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

final int A=100;

//A=200; Error

1. Final method : if method is final we can’ override that method but we can use it or call it in sub class.
2. Final class : if class is final we can’t extends or inherits.

final class Bike {

final void speed() {

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

}

}

class Honda extends Bike {

/\*void speed() {

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

}\*/

}

class FinalConcept {

public static void main(String args[]) {

final int A=10;

//A=20;

System.out.println(A);

Honda hh = new Honda();

hh.speed();

}

}

static keyword

1. Static keyword we can use with variable and method but not with class. ( we can use static keyword with class but class must be inner or nested class).
2. If variable it static we can assign the value for that variable using class name.
3. If method is static we can call that method with the help of class name.
4. Static variable and static method we can access as well as call through object also.
5. Inside a non static method we can access static as well as non static variable directly. But inside static method we can access only static variable directly.

class Abc {

int a; // instance variable

static int b; // static variable

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");

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

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

}

}

class StaticConcept {

public static void main(String args[]) {

Abc obj1 = new Abc();

obj1.a=10;

Abc.b=20;

obj1.dis1();

Abc.dis2();

obj1.b=30;

obj1.dis2();

}

}

Another Example

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);

}

}



class StaticConcept1 {

public static void main(String args[]) {

Abc obj1 = new Abc();

Abc obj2 =new Abc();

obj1.a=10;

obj1.b=20;

Abc.b=30;

obj2.a=40;

obj2.b=50;

Abc.b=60;

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

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

}

}

interface concept

interface is a type of reference data type. It is also known as 100% pure abstract class.

Syntax of interface

interface interfaceName{

fields;

methods;

}

By default all variable or fields inside an interface are public static and final

By default all methods in interface are public and abstract.

interface Abc {

public static final int A=10;

static final int B=20;

final int C=30;

int D=40;

public abstract void dis1();

abstract void dis2();

void dis3();

}

interface Abc {

int A=10;

void dis1();

}

Like a class one interface can extends another interface But interface can extends more than one interface.

interface Abc {

int A=10;

void dis1();

}

interface Xyz {

int B=20;

v­­oid dis2();

}

interface Mno extends Abc,Xyz{

int C=30;

void dis3();

}

class Demo implements Abc,Xyz{

}

class always implements interface and class can implements more than one interface. Whichever class implements any interface that class must be provide the body for all abstract method belong that interface.

One class can extends only one class

One class can implements more than one interface.

One interface can extends more than one interface.

One interface can’t extends or implements to class.

**Day 8 :**

**11-10-2022**

Rules while overriding with access specifiers

Super class / Interface Sub class

public public

protected public

protected

default (nothing) public

protected

default (nothing)

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 Demo implements Abc,Xyz {

public void dis1() {

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

}

public void dis2() {

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

}

}

class InterfaceDemo {

public static void main(String args[]) {

Demo obj1 = new Demo();

obj1.dis1();

obj1.dis2();

}

}

Difference interface and abstract class.

1. In abstract class it not mandatory all variable must final. But in interface all variable must be final.
2. In abstract class it not mandatory all method must be abstract. But in interface all methods must be abstract.
3. Normal class can extends only one abstract class But normal class can implements more than one interface.
4. Abstract class can contains constructor even we can write parameterized constructor. But interface doesn’t contains default constructor.

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

this keyword, super keyword, this(), super()

class A {

int a=10;

}

class B extends A {

int a=20;

void dis() {

int a=30;

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

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

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

}

}

class ThisAndSuper{

public static void main(String args[]) {

B obj = new B();

obj.dis();

}

}

this() and super()

this() it is use to invoke same class constructor like constructor chaining. It must be first statement inside a constructor.

Super() : it is use to invoke sub class constructor to super class constructor and it must be first statement inside a sub class constructor by default every sub class constructor super() parameter is present.

class A {

A() {

//this(10);

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

}

A(int x) {

this(10,20);

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

}

A(int x, int y) {

this();

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

}

}

class B extends A {

B() {

super(10);

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

}

}

class ThisAndSuperParameter{

public static void main(String args[]) {

//A obj1 = new A();

//A obj2 = new A(10);

//A obj3 = new A(10,20);

B obj4 = new B();

}

}

Run time polymorphism in object in creation.

1st program

class A {

void dis1() {

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

}

}

class B extends A {

void dis1() {

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

}

void dis2() {

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

}

}

class PolymorphismDemo {

public static void main(String args[]) {

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

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

A obj3 = new B(); // creating sub class object and super class reference possible in java

obj3.dis1(); // with the help of this reference we can call only those method belong to super class or overrided methods.

//obj3.dis2();

//B obj4 = new A(); // creating super class object and sub class reference not possible

}

}

2nd example

abstract class A {

abstract void dis1();

}

class B extends A {

void dis1() {

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

}

void dis2() {

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

}

}

class PolymorphismDemo {

public static void main(String args[]) {

A obj = new B(); // sub class object and super class reference super class can be normal class or abstract class.

obj.dis1();

}

}

3rd example

interface A {

void dis1();

}

class B implements A {

public void dis1() {

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

}

void dis2() {

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

}

}

class PolymorphismDemo {

public static void main(String args[]) {

A obj = new B(); // sub class object and super class/interface reference super class can be normal class or abstract class or interface

obj.dis1();

}

}

Abstraction : hiding the internal implementation without knowing background details.

interface Client1 {

public int add(int x, int y);

}

interface Client2 {

public int sub(int x, int y);

}

class Manager implements Client1, Client2 {

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 method");

}

}

class Abstraction {

public static void main(String args[]) {

Manager mgr = new Manager();

System.out.println(mgr.add(100,200));

System.out.println(mgr.sub(100,200));

mgr.ownMethod();

Client1 cl1 = new Manager();

Client2 cl2 = new Manager();

System.out.println(cl1.add(100,200));

System.out.println(cl2.sub(100,200));

}

}

**Day 9 :**

**13-10-2022**

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

**Package mainly divided into two types**

1. **User defined package**
2. **Built in package**

**Package is just like folder or directory. When two or more than classes and interfaces have same name but different purpose.**

**education**

**school college pg**

**Attendance Attendance Attendance**

**Syntax to create package**

**package com;**

**package com.service;**

**Access specifiers : Java provided totally four types of access specifiers**

**Using specifiers we can expose the visibility or accessibility of variable, method and classes in one package to another package.**

**private**

**default (nothing)**

**protected**

**public**

**private :this access specifiers we can use with instance variable, static variable, static and non static method, constructor but we can’t use with class and local variable.**

**Scope : private we can access within a same class.**

**default : we can use with all.**

**Scope : within a same package.**

**protected : this access specifiers we can use with instance variable, static variable, static and non static method, constructor but we can’t use with class and local variable.**

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

**public :this access specifiers we can use with instance variable, static variable, static and non static method, constructor and class but can’t use with local variable.**

**Scope : within a same package as well as other package .**

**Pre defined package : java provided lot of pre defined package and those package part of two packge ie**

**java javax**

**lang swing**

**util servlet**

**io ejb**

**awt sql**

**net**

**sql**

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

**JDK and JRE**

**By default every java program import lang package.**

**So we can use all classes and interfaces part of lang package directly without importing.**

**Exception Handling :**

**Exception : Exception is type of memory or object which occurs when unexpected or abnormal condition happened during the execution of a program is known as exception.**

**Java program**

**javac java**

**compile time error run time error**

**syntax error**

**or typo error**

**run time error divided into two types**

**Error Exception**

**In java Error and Exception are pre defined classes part of lang package.**

**Error : The error which generate at the run time which we can’t handle it is known as Error. Example : JVM crash, out of memory, software and hardware issue.**

**Exception : it is a type of error which generated at the run time which we can handle it. Example divided by zero.**

**By default every java program extends pre defined class ie Object. Object is pre defined class part of lang package.**

**Object**

**extends**

**Throwable**

**extends**

**Errror**

**Exception**

**Checked exception unchecked exception**

**extends**

**IOException RuntimeException**

**SQLException ArrayIndexOutOfBoundsException**

**etc ArithmeticException**

**NullPointerException**

**Etc**

**All un checked exception are sub class of RuntimeExcepiton.**

**But all checked exception directly or indirectly extends Exception class.**

**To handle both the type of exception it can be checked or unchecked java provided five keyword.**

1. **try**
2. **catch**
3. **finally**
4. **throw**
5. **throws**

**syntax of try catch block**

**try {**

**}catch(Exception e) {**

**}**

**Day 10 :**

**13-10-2022**

**Wrapper classes are part of lang package. Which provided set of methods which help to convert one data type to another data. Wrapper classes wrap primitive values and provided extra methods to do some operation on primitive values.**

**Wrapper classes primitive data types**

**Byte byte**

**Short short**

**Integer int**

**Long long**

**Float float**

**Double double**

**Character char**

**Boolean boolean**

**int a=10; primitive consider**

**Integer b = new Integer(a); now b consider as object**

**int c = b.intValue(); help to convert object to primitive**

**float d = b.floatValue(); help to convert objet to primitive**

**Integer.parseInt(str); it will help to convert string to number.**

**In single try catch block if we don’t know what type of exception then we can use as well as if any type of exception generate we want to do common task.**

**Try with multiple catch block.**

**try {**

**}catch(ArithmeticException e) {**

**}catch(ArrayIndexOutOfBoundsException e){**

**}**

**package** com;

**public** **class** TryWithMultipleCatchBlock {

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

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

**int** a=10;

**int** b=1;

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

String str = "10a";

**try** {

**int** res1= a/b;

System.***out***.println("Result "+res1);

**int** res2 = 10/abc[2];

System.***out***.println("Result "+res2);

System.***out***.println(Integer.*parseInt*(str));

}**catch**(ArithmeticException e) {

System.***out***.println(e.toString()); // specific coding

}**catch**(ArrayIndexOutOfBoundsException e) {

System.***out***.println(e.toString()); // specific coding

}**catch**(Exception e) {

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

}

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

}

}

**finally block :**

**try : the code it may be one line code or multi line code we have to keep in a try block it we consider that can generate the exception.**

**catch : this block execute only if any exception generate. No exception no catch block.**

**finally block : this block execute 100% sure if any exception generate or not.**

**try**

**catch catch catch catch finally**

**catch finally catch**

**catch catch**

**finally**

**try{**

**open the file**

**read and write or copy the file**

**}catch(Exception e) {**

**}finally {**

**close the file**

**}**

**Finally block is use to execute the close the resource ie file handling or database connectivity.**

**throw : This keyword is use to generate or raise the pre defined or user defined or custom exception depending upon our conditions.**

**throw new Exception();**

**or**

**throw new ExceptionSubClass()**

**throws : This keyword is use to throw to exception to caller methods. This keyword we use with method signature.**

**void display() throws Exception, ExceptionSubClass {**

**}**

**Day 11 :**

**18-10-2022**

**Multithreading :**

**Program :** set of instruction to perform a specific task is known as program.

**Processor :** Processor is responsible to execute the code.

**Process :** time taken to execute the code or process in execution is known as process.

**Thread :** Thread is small execution of code within a process.

Thread is known as light weighted process means it take less memory or less resource of our machine.

Process is heavy it take more memory and more resources to do same task.

Thread is part of process.

In Java inside a main method by default one thread execute.

Thread is a pre defined class part of lang package which contains pre defined static method that is currentThread which help to provide the details about default thread. This method return type is same class reference.

Thread t = Thread.currentThread();

System.out.println(t); Thread[main,5,main]

main 🡪 name of the thread

5🡪 priority of thread

main 🡪 group of the thread

min 1, max =10, norm 5,

Multithreading : creating and running more than one thread.

Multi tasking : running or creating more than one task at a time.

Using Process base : C or C++

Using Thread base : Java

Process

Thread

Multi tasking using thread base is faster than multi tasking using process base.

In Java we can create more than one thread using

1. extends Thread class
   1. create normal java class and that class must be extends Thread class.
   2. Create the object of that class which extends Thread class. (creating the reference of thread class).
   3. With the help of that object we have to call start() method. Start is a pre defined method part of thread class which help to make the thread ready to run ie Runnable state.
   4. Start() method internally call run() method of Thread class. That run method contains empty body. So if we want to execute any custom code then we have to override run method inside that class the class which extends Thread class.
2. implements Runnable interface
   1. create normal class and that class implements Runnable interface.
   2. Runnable is a pre defined interface part of lang package which contains run abstract method.
   3. When class implements Runnable interface it must be override run method mandatory.
   4. Now we have to create object of user defined class which implements Runnable interface.
   5. Then create the Thread class reference and through constructor pass the reference of that class which class implements Runnable interface.

**Thread pre defined class internally implements Runnable and provided body for run method.**

**annotation :** Annotation is known as meta data. Data about data. Java provide lot of pre defined annotation which we can use on class level, method level, property level. All annotation start with prefix @ followed by annotation name.

**Thread life cycle**

Create ----------🡪 Runnable -----------------🡪Running -----------------🡪Destroy

obj1 obj1.start() run() i=10,j=10

t1 t1.start(); run()

**sleep()**

**wait()**

**notify()**

**notifyAll()**

**isAlive()**