Day 1

20-09-2022

Software

Application software : This type of software mainly use to develop the application.

System software : Operating system : it is to run the machine. OS window, unix, linux etc.

Programming language :

C 1970

C++ 1980

Java 1990

Python

C#

JavaScript

Go lang

Etc

What is Java ?

Java is pure object oriented and platform independent programming language.

object : object is any real world entity.

Properties or state --🡪 have -🡪 name, age, weight, color, height etc.

Person

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

Bank

Place

Car

Customer

Employee

class : blue print of object or template of object.

syntax of class

class ClassName {

property or variable or fields

behavior – function / methods

}

IDE

class

Java software

JDK : Java Development kit : which help to develop the application

JRE : Java run time environment :which help to run the application.

javac ProgramName.java : to compile the program

java classname : it is use to run the program

Day 2

22-09-2022

Variables : variable is a name which hold value and value can change during the execution of a program.

a=10;

name =”Raj Deep”;

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

2 types

1. Primitive data types : it is use to store only value.
2. Non primitive data types or reference data type: it use to store value as well as reference of another value.

Primitive data types :

8 types

1. byte 1 byte : -128 to 127 range
2. short 2 byte
3. int 4 byte
4. long 8 byte : without decimal
5. float 4 byte
6. double 8 byte : with decimal
7. char 2 byte : any single character a to z or symbol or number
8. boolean 1 bit : true or false

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

1. Arithmetic operator : +, -, \*, /, %(remainder)
2. Conditional operator : >, >=, <, <=, ==, !=
3. Logical operator : && (and), || (or) , !(negation)

&& if all condition is true then result is true

|| if any condition is true then result is true

1. Assignment operator : =

int a=10;

int b = 40+60;

int c+d = 100+400; Error

1. Increment and decrement

int a=10;

a = a+1;

or

a++ increment the value by one

a= a-1;

or

a-- decrement the value by one

a=10;

c=30;

value a 10;

value c 30;

a = c; assigning the value of c in a variable

a == c; comparing the value of a and c

a = c+d; error

a+b == c+d comparing

Day 3

27-09-2022

Conditional statement

1. If statement

if(condition) {

true block

}

1. If else

if(condition) {

}else {

}

1. If else if
2. Switch statement

**If and else example**

class IfStatement {

public static void main(String args[]) {

int age = 25;

if(age>21) {

System.out.println("You can vote!");

}else {

System.out.println("You Can't vote");

}

System.out.println("Finish");

}

}

If else if program

class IfElseIfStatement {

public static void main(String args[]) {

int m1=70, m2=90,m3=50;

int total = m1+m2+m3;

int avg = total/3;

if(avg>90) {

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

}else if(avg>75) {

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

}else if(avg >65){

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

}else {

System.out.println("C Grade");

}

System.out.println("Thank You!");

}

}

switch : in switch user can take decision which block want to execute.

Syntax

switch(variableName){

case label1: block1;

break;

case label2: block2;

break;

case label3: block3;

break;

default : defaultblock

break;

}

switch, case, break and default are keywords

in Switch we can use variable of type int or char or String etc.

class SwitchStatement {

public static void main(String args[]) {

int choice = 5;

switch(choice) {

case 1: System.out.println("First block");

System.out.println("First block finish");

break;

case 2: System.out.println("Second block");

System.out.println("Second block finish");

break;

case 3: System.out.println("Third block");

System.out.println("Third block finish");

break;

default : System.out.println("Wrong choice");

break;

}

System.out.println("Finish");

}

}

Taking the value through keyboards in Java

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

Scanner : Scanner is pre-defined class part of util package which provided lot of pre defined methods which help to take the value through keywords.

Syntax to create the Scanner class object

Scanner sc =new Scanner(System.in);

sc.nextByte();

sc.nextShort()

sc.nextInt()

sc.nextLong();

sc.nextFloat();

sc.nextDouble();

sc.nextBoolean();

sc.nextChar() no methods

Scanner class with switch statement

import java.util.\*;

class InputValue {

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

/\*System.out.println("Enter the value of a");

int a = sc.nextInt();

System.out.println("Enter the value of b");

int b = sc.nextInt();

int sum = a+b;

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

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

int choice,a,b,sum,sub;

System.out.println("1:Add 2:Sub");

System.out.println("Plz enter your choice");

choice = sc.nextInt();

switch(choice) {

case 1: System.out.println("Enter the value of a");

a = sc.nextInt();

System.out.println("Enter the value of b");

b = sc.nextInt();

sum = a+b;

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

break;

case 2: System.out.println("Enter the value of a");

a = sc.nextInt();

System.out.println("Enter the value of b");

b = sc.nextInt();

sub = a-b;

System.out.println("Sub of two number is "+sub);

break;

default : System.out.println("Wrong choice");

break;

}

System.out.println("Finish");

}

}

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

While loop

Do while loop

For loop

Initialization : start and end position

Condition : true

Body of the loop

Increment or decrement the value

While loop

Initialization

while(condition) { entry loop

do the task

increment or decrement

}

Do while loop

Initialization

do {

do the task

increment or decrement

}while(); exit loop

For loop

1 2 4

for(initialization;condition; increment/ decrement) {

body of for loop 3

}

Initialization only once

Check the condition if condition is true then body of the loop and increment or decrement and then again check the condition till condition become false. – 2,3,4

Day 4

29-09-2022

Reference data types : it is use to store the value as well as reference of another data type.

array

class : user defined or pre-defined class

interface : user defined or pre defined interface

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

int a=10;

syntax to declare the array

datatype arrayName[];

int abc[]; // array declaration

int xyz[]={10,20,30,40,50,60}; // array declaration with memory creation array index position start from zero.

System.out.println(a);

System.out.println(xyz);

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

System.out.println(xyz[1]);

System.out.println(xyz[2]);

Array example

class ArrayDemo {

public static void main(String args[]) {

int a=10;

a=20;

int abc[];

int xyz[]={10,20,30,40,50,100,12,34,54,34,23,56,78,90,45,67,100,10,200,300,400};

System.out.println(a);

//System.out.println(xyz);

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

System.out.println(xyz[1]);

System.out.println(xyz[2]);

System.out.println("Size of the array "+xyz.length);

System.out.println("Retrieve the elements from array using loop");

int searchNumber = 10;

int flag = 0;

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

//System.out.println(xyz[i]);

if(searchNumber ==xyz[i]){

flag++;

//break;

}

}

if(flag>0){

System.out.println("Element is present and it present number of times are "+flag);

}else {

System.out.println("Element not present");

}

}

}

Syntax for array memory creation

datatype arrayName[]=new DataType[size];

int abc[]=new int[10]; default 0

float xyz[]=new float[20]; float 0.0

class ArrayMemoryCreation {

public static void main(String args[]) {

int abc[]={100,200,300,400};

int num[]=new int[10];

System.out.println("Size of array is "+num.length);

System.out.println("Size of array is "+abc.length);

System.out.println("Value in zero index position "+num[0]);

System.out.println("Value in 2 index position "+num[2]);

System.out.println("Value in 9 index position "+num[9]);

//System.out.println("Value in 10 index position "+num[10]);

num[0]=100;

num[1]=200;

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

System.out.println(num[1]);

}

}

import java.util.\*;

class ArrayMemoryCreation {

public static void main(String args[]) {

/\*int abc[]={100,200,300,400};

int num[]=new int[10];

System.out.println("Size of array is "+num.length);

System.out.println("Size of array is "+abc.length);

System.out.println("Value in zero index position "+num[0]);

System.out.println("Value in 2 index position "+num[2]);

System.out.println("Value in 9 index position "+num[9]);

//System.out.println("Value in 10 index position "+num[10]);

num[0]=100;

num[1]=200;

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

System.out.println(num[1]);\*/

Scanner obj = new Scanner(System.in);

System.out.println("How many number do you want store");

int n = obj.nextInt();

int num[]=new int[n];

System.out.println("Plz enter number one by one");

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

num[i]=obj.nextInt();

}

int sum=0;

System.out.println("All elements are ");

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

//System.out.println(num[i]);

sum = sum + num[i]; // sum = 0+1, sum = 1+10

}

System.out.println("Sum of all numbers are "+sum);

}

}

String : String is a pre-defined or also known as reference data types.

Syntax to create the String class object.

String name = “Raj Deep”;

String msg = “Welcome to Java Training”;

import java.util.\*;

class StringDemo {

public static void main(String args[]) {

/\*String name = "Raj Deep";

String msg = "Welcome to Java Training";

System.out.println(name);

System.out.println(msg);

Scanner obj = new Scanner(System.in);

System.out.println("Plz enter your name");

//String fname = obj.next(); // it is use to scan the value through keyword, only one word.

String fname = obj.nextLine(); // more than one word till hit enter key

System.out.println("your name is "+fname);\*/

Scanner obj = new Scanner(System.in);

System.out.println("How many names do you want to store");

int n = obj.nextInt();

String names[]=new String[n];

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

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

names[i]=obj.next();

}

System.out.println("All names are");

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

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

}

}

}

int num[][]={{1,2,3},{4,5,6},{7,8,9}}

num[0][0] -🡪1

num[0][1] 🡪2

num[0][2] 🡪3

num[1][0] 🡪 4

int marks [][]=new int[][]

1. Create the Java Application to store more than one students details like sid, sname,age.
2. Create another Java application to store more than one students details like sid, sname,age, marks(array of array).

Day 5

06-10-2022

OOPs : Object Oriented Programming system

object : object is any real world entity.

Properties or state -🡪 have -🡪 variables / fields

Person

Behavior -🡪 do/does -🡪 function / methods

Bank

Animal

Car

Customer

Employee

class : class is blue print of object or template of object or it is a user defined data type which help to create the object.

Method or function : it is use to write set of instruction to perform a specific task.

Method or function syntax

returnType methodName(parameterList) {

}

No return type and no passing parameter

void info() {

// coding …

}

Passing parameter but no return type

void add(int x, int y) {

int sum = x+y;

System.out.println(sum);

}

Passing parameter and return value

String sayHello(String name) {

// coding

return “Welcome user “+name;

}

int addNumber(int x, int y) {

int sum = a+b;

return sum;

}

User defined class with properties and behavior

class App {

public static void main(String args[]) {

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

Car innova = new Car(); // memory created...

innova.start("Innova");

innova.stop();

//System.out.println("Santro Car Details");

Car santro = new Car();

santro.start("Santro");

santro.appliedGear();

santro.stop();

}

}

class Car {

int wheel;

float price;

String color;

void start(String name) {

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

}

void appliedGear() {

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

}

void moving() {

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

}

void stop() {

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

}

}

Types of variable or fields

In Java variables are divided into 3 types

1. Instance variable
   1. The variable which declared inside a class but outside method is known as instance variable.
   2. The instance variable hold default value according to their data types example int family 0, float 0.0, boolean false, string null etc.
   3. Instance variable we can access in all method but method must be part of same class and method must be non static.
2. 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 variable within that block where it declared.
3. Static variable

class App {

public static void main(String args[]) {

Car ertiga = new Car();

//ertiga.carDetails();

ertiga.wheel = 4;

ertiga.color = "Gray";

ertiga.price = 1400000;

ertiga.carDetails("Ertiga Car Details");

Car innova = new Car();

innova.wheel = 4;

innova.color = "White";

innova.price = 2200000;

innova.carDetails("Innova Car Details");

}

}

class Car {

int wheel;

double price;

String color;

void carDetails(String name) {

int temp;

System.out.println(name);

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

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

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

}

}

Another example

class App {

public static void main(String args[]) {

int a=1;

System.out.println(a);

Car ertiga = new Car();

//ertiga.carDetails();

ertiga.wheel = 4;

ertiga.color = "Gray";

ertiga.price = 1400000;

ertiga.carDetails("Ertiga Car Details");

Car innova = new Car();

innova.wheel = 4;

innova.color = "White";

innova.price = 2200000;

innova.carDetails("Innova Car Details");

}

}

class Car {

int wheel;

double price;

String color;

void carDetails(String name) {

int temp=100;

System.out.println(name);

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

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

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

System.out.println(temp);

}

void display() {

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

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

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

//System.out.println(temp);

}

}

Day 6

11-10-2022

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

While creating a constructor we have to follow few steps.

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 memory.

In the life of the object if we want to perform any task only one time that type of task we have to write inside a constructor. Ex : instance variable initialization.

In the life of the object if we want to perform any task more than one time that type of task we have to write inside a method.

class Abc {

Abc() {

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

}

void display() {

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

}

}

class ConstructorDemo {

public static void main(String args[]) {

Abc obj1 = new Abc();

obj1.display();

obj1.display();

Abc obj2 = new Abc();

obj2.display();

obj2.display();

obj2.display();

}

}

Parameterized constructor example

class Operation {

int a, b,sum; //a=0,b=0,sum=0

Operation() {

a=1;

b=2;

}

Operation(int x, int y) {

a=x;

b=y;

}

void setValue(int x, int y) {

a=x;

b=y;

}

void add() {

sum = a+b;

}

void display() {

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

}

}

class ConstructorDemo1 {

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.add(); op3.display();

Operation op4 = new Operation(10,20); op4.add(); op4.display();

Operation op5 = new Operation(100,200); op5.add(); op5.display();

Operation op6 = new Operation(); op6.setValue(11,22); op6.setValue(111,222); op6.setValue(1111,2222); op6.add(); op6.display();

}

}

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

Example : class

If local variable and instance variable have same name then local variable hide the visibility of instance variable. So if we want to refer to instance variable this we have to use this.instancevaraible.

this is a keyword which refer to current object.

class Employee {

private int id;

private String name;

private float salary;

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

id = id1;

name = name1;

//salary = salary1;

if( salary1<0 ) {

salary = 8000;

}else {

salary = salary1;

}

}\*/

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

this.id = id;

this.name = name;

if(salary<0) {

this.salary = 8000;

}else {

this.salary = salary;

}

}

void display() {

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

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

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

}

}

class Encapsulation {

public static void main(String args[]) {

Employee emp1 = new Employee();

Employee emp2 = new Employee();

//emp1.id = 100;

//emp1.name = "Ravi";

//emp1.salary = -12000;

emp1.setValue(100,"Ravi",-12000);

emp2.setValue(101,"Ramesh",10000);

emp1.display();

emp2.display();

}

}

Another Example of Encapsulation

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 Encapsulation1 {

public static void main(String args[]) {

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

Employee emp2 = new Employee(2,"Ramesh",14000);

emp1.display();

emp2.display();

emp1.setSalary(15000);

emp1.display();

emp2.display();

}

}

Day 7

13-10-2022

**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 child class or derived class.

properties

behavior

}

Simple example of Inheritance

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 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 { } But this type of inheritance java doesn’t support. In Java one class can extends only one class at time. This type of inheritance java support using interface concept.

OOPs relationship

Two type of relationship

1. Is a relationship : generally this type of relationship we can achieve using inheritance.
2. Has a relationship : inside one class we have to create another class object.

class Employee {

id,name,salary

}

class Manager extends Employee{

numberOfEmp;

Address add = new Address();

}

class Programmer extends Employee {

projectName;

}

class ProjectManager extends Manager {

clientInfo;

}

class Address {

city,state

}

Manager/Programmer Is a Employee

ProjectManager is a Manager

Employee has a Address

Has a relationship

Association

Aggregation

Composition

Association : inside one class we have to create the object of another class to make association.

class A {

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

}

class B {

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

}

Aggregation : it is type o association which is known as weak association ie aggregation

class Employee {

Address padd = new Address();

Address ladd = new Address();

}

class Address {

}

Composition : it is a type of association which is known as strong association ie composition

class Student {

StudentHistory sh = new StudentHistory();

}

class StudentHistory {

}

Day 8

19-10-2022

Polymorphism :One name many forms or many implementation

2 types

1. Compile time polymorphism

Static binding or early binding

Method overloading : The method have same name but different parameter list ie type of parameter list as well as number of parameter list must be different.

Example

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(double x, double y) {

System.out.println(x+y);

}

void add(String x, String y) {

System.out.println(x+y);

}

}

class MethodOverloading{

public static void main(String args[]) {

Operation op = new Operation();

op.add(1,2,3);

op.add(1,2);

op.add(10.10,20.20);

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

}

}

Find area of triangle, circle and rectangle. Method name must be area using method overloading.

1. Run time polymorphism

Dynamic binding or late binding

Method Overriding : The method have same name and same signature (number of parameter list, type of parameter list and return type must be same). If we want to achieve method overriding then we have to use inheritance.

Method overriding example

class Bike {

void speed() {

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

}

}

class Honda extends Bike { // re-usability

void color() {

System.out.println("Gray");

}

}

class Pulsar extends Bike {

void speed() { // override the code don't like

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

}

void color() {

System.out.println("Black");

}

}

class Tvs extends Bike {

void color() {

System.out.println("Red");

}

void speed() {

super.speed(); // merge the code super class and its own code.

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

}

}

class MethodOverriding {

public static void main(String args[]) {

Honda hh = new Honda();

System.out.println("Honda bike details");

hh.color();

hh.speed();

Pulsar pu = new Pulsar();

System.out.println("Pulsar bike details");

pu.color();

pu.speed();

Tvs tv = new Tvs();

System.out.println("Tvs bike details");

tv.color();

tv.speed();

}

}

In same example add mileage method,

Same example try to do with Bank, Hdfc, Hsbc, Sbi

Abstract keyword

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

abstract returnType methodName(parameterList);

abstract void speed();

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

abstract class Demo {

}

1. Whichever class extends abstract class that class must be provide the body for all abstract method belong that class mandatory. That class can ignore only if that class itself is an abstract class.
2. Abstract class we can’t create the object.
3. Abstract class can contains normal as well as abstract methods. It can contains zero or 1 or may abstract method. It is not mandatory abstract class contains only abstract methods.
4. Abstract class without abstract method use we can’t create the object of that class.

abstract class Bike {

abstract void speed();

}

class Honda extends Bike{

void color() {

System.out.println("Gray");

}

void speed() {

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

}

}

class Pulsar extends Bike {

void color() {

System.out.println("Black");

}

void speed() {

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

}

}

class AbstractKeyword {

public static void main(String args[]) {

Pulsar pu = new Pulsar();

pu.speed();

Honda hh = new Honda();

hh.speed();

}

}

Day 9

20-10-2022

final keyword :

1. final keyword we can use with variable, method and class.
2. Final variable : if we want to declare constant value means we don’t want to change the value

final int A=10; final variable declare in upper case.

A=20; // we can’t change the value.

1. final method : if method is final sub class can’t override that method. But we can call that method but can’t override.

final void speed() {

System.out.println(“60km/hr”);

}

1. final class : if class is final we can’t inherits or extends that class.

final class Sample {

}

Example

final class Bike {

final void speed() {

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

}

}

class Honda extends Bike {

/\*void speed() {

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

}\*/

}

class FinalDemo {

public static void main(String args[]) {

int a=10;

a=20;

final int B=20;

//B =30;

System.out.println(a);

System.out.println(B);

Honda hh = new Honda();

hh.speed();

}

}

Static keyword

1. static keyword we can use with variable and method but not with class.
2. Static variable: if variable is static we can access that variable or assign the value for that variable with help of class name.
3. Static method : if method is static we can call that method with help of class name.
4. We can assign or access static variable with help of object also.
5. We can call static method through object also.

class Abc {

int a; // instance variable

static int b; // static variable

void dis1() {

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

}

static void dis2() {

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

}

}

class StaticDemo {

public static void main(String args[]) {

Abc obj1 = new Abc();

obj1.a=100;

Abc.b=200;

obj1.b=300;

obj1.dis1();

Abc.dis2();

obj1.dis2();

}

}

In Java we can get two types of memory

First one is static memory

Second one is instance memory or heap memory.

For every Java class we will get only one static memory and for that class how many object we create that many instance or heap memory get created.

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 StaticDemo1 {

public static void main(String args[]) {

Abc obj1 = new Abc();

Abc obj2 = new Abc();

obj1.a=10;

Abc.b=20;

obj1.b=30;

obj2.a=40;

Abc.b=50;

obj2.b=60;

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

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

}

}

Employee

id,name,salary instance variable

clientId,MgrId,companyName : static variable

static is like a global to all object.

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

Interface contains only abstract methods.

Syntax to declare the interface

interface interfaceName {

fields or variable;

methods;

}

In interface by default all variables are public static final.

In interface by default all methods are public and abstract.

interface Abc {

public static final int A=10;

public abstract void dis1();

}

We can declare same interface another way

interface Abc {

int A=10;

void dis1();

}

interface Xyz {

int B=20;

void dis2();

}

interface Mno extends Abc, Xyz{

int C=30;

void dis3();

}

Like a class one interface can extends more than one interface But interface can extends more than one interface. Class can extends only one class. Using this interface we can achieve multiple inheritance. Abc and Xyz are super interface and Mno is sub interface.

Mno interface contains 3 methods ie dis1(), dis2() and dis3() but all method are incomplete.

class Demo implements Abc, Xyz {

}

Class always implements interface class can implements more than one interface.

Whichever class implements any interface it may be one or more than one that class must be provide the body for all those methods belongs that interfaces.

Class to class : extends only one.

Interface to interface : extends more than one

Class to interface : implements more than one

Interface to class : interface can’t extends or implements to class.

While overriding with access specifiers with method we have to follow some rules

Super class / super interface Sub class

public public

protected public

protected

default (nothing) public

protected

default (noting)

private we can’t override that method

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 obj = new Demo();

obj.dis1();

obj.dis2();

}

}

Difference between interface and abstract class.

1. abstract class can contains final variable But interface contains only final variable.
2. Abstract class can contains abstract method. It may contains one or more but interface contains only abstract methods.
3. Normal class can extends only one abstract class but it can implements more than one interface.
4. Using abstract class we can achieve partial abstraction but using interface we can achieve 100% pure abstraction

Abstraction means hiding the internal implementation without knowing background details.

Day 10

25-10-2022

this keyword

super keyword

this()

super()

if local variable and instance variable have same name then local variable hide the visibility of instance variable so if we want to refer to instance variable then we have to use this.instancevariablename.

if sub class contains same variable which one available in super class then sub class variable hide the visibility of super class variable. So if we want to refer to super class variable then we have to use super.superclassvariableName.

class A {

int n=10;

}

class B extends A {

int n=20;

void dis1() {

int n=30;

System.out.println(" local variable n "+n); // local variable

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

System.out.println(" Super class variable n "+super.n); // super class variable

}

}

class SuperAndThis {

public static void main(String args[]) {

B obj = new B();

obj.dis1();

}

}

this() and super() : this() parameter is use to call same class constructor or constructor chaining or calling same class constructor. this() parameter must be use inside a constructor and it must be first statement inside a constructor. this() by default not available.

super() parameter is use to call sub class constructor to super class constructor or constructor chaining between sub class to super class. Super() parameter must be use inside constructor and it must be first statement inside a sub class constructor. By default every sub class constructor super() available.

This parameter example

class A {

A() {

this(100);

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

}

A(int x) {

this(100,200);

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

}

A(int x, int y) {

this("Ravi");

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

}

A(String str) {

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

}

}

class ThisParameter{

public static void main(String args[]) {

A obj1 = new A();

//A obj2 = new A(100);

//A obj3 = new A(100,200);

}

}

Super parameter example

This and super parameter example

abstract class Employee {

private int id;

private String name;

private float salary;

Employee() {

id = 123;

name = "Unknown";

salary = 8000; // we set these information with big logic.

}

Employee(int id) {

this();

this.id = id;

}

Employee(int id, String name) {

this(id);

this.name = name;

}

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

this(id,name);

this.salary = salary;

}

void 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;

Manager() { // by default super() present it will call super class empty constructor

numberOfEmp =2;

}

Manager(int id) {

super(id); // calling super class with one parameter as id

numberOfEmp =2;

}

Manager(int id, String name) {

super(id,name); // calling super class with two parameter as id and name

numberOfEmp =2;

}

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

super(id,name,salary); // calling super class with three parameter id ,name and salary

numberOfEmp =2;

}

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

super(id,name,salary);

this.numberOfEmp = numberOfEmp;

}

void disMgr() {

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

}

}

class SuperAndThisParameter {

public static void main(String args[]) {

Manager mgr1 = new Manager(); mgr1.disEmp(); mgr1.disMgr();

Manager mgr2 = new Manager(100); mgr2.disEmp(); mgr2.disMgr();

Manager mgr3 = new Manager(101,"Ajay"); mgr3.disEmp(); mgr3.disMgr();

Manager mgr4 = new Manager(102,"Vijay",45000); mgr4.disEmp(); mgr4.disMgr();

Manager mgr5 = new Manager(103,"Mahesh",55000,10); mgr5.disEmp(); mgr5.disMgr();

}

}

Day 11

27-10-2022

Package : package is a collection of classes and interfaces. Package is just like a directory or folder when two or more than one classes and interfaces which have same name but different purpose. Then using package we can avoid conflicts.

Package mainly divided into two types

1. User defined package
2. Pre defined package

User defined package

education

school college

Attendance.java Attendance.java

Syntax to create the package

package packagename;

package com;

package education.school;

Access Specifiers or Access Modifiers

Using access specifiers we can expose the visibility of variable, method and class within same package as well as other packages.

Java provided 4 types of access specifiers

1. private
2. default (nothing)
3. protected
4. public

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

scope : we can access or use within a same class. Outside class we can’t access.

default : We can use default access specifiers with all

scope : we can access within same package. Other package we can’t access.

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

Scope : same package as well as other package if it is a sub class.

public : This access specifiers we can use with instance variable, static variable, non static method static method, constructor, class but local variable.

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

Day 12

01-11-2022

**Exception Handling** : Exception is object or memory which occurs when unexpected or abnormal condition occurs during the execution of a program. To handle that expected thing using some technique is known as exception handling.

**Java Program**

javac java

java compiler java interpreter

compile time error run time error

syntax error

typo error

class Demo {

int a;

display() {

}

}

Run time error

Error Exception

Error and Exception both are pre-defined classes part of lang package. By default every java program imported lang package.

**Error** : it is type of error which generate at the run time which we can’t handle it.

Example : JVM crash , out of memory , software or hardware etc.

**Exception** : it is type of error which generate at the run time which we can handle it.

Divided by zero, array index out of bounds exception.

By default every java program it may be pre-defined or user defined internally extends Object class. This class also part of lang package. By default every java import lang package.

Object -🡪 top most class

Throwable : 🡪 this class internally extends Object. class.

Error Exception

Error and Exception classes internally extends Throwable.

Checked Exception Un checked Exception

IOException RuntimeException

SQLException ArrayIndexOutOfBoundsException

ArithmeticException

NumberFormatException

Etc

All unchecked exception internally extends RuntimeException class.

But all checked exception directly or indirectly extends Exception class.

To handle both checked as well as unchecked exception java provided totally 5 keywords.

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

try and catch block : single catch block

try {

try block

}catch(Exception e) {

Catch block

}

The code it may be one line code or more than one line code which make problem we need to keep in try block. If any exception generate in try block then catch block will ready to catch that exception.

Try with multiple catch block:

Try with single catch block is ready to handle any type of exception.

If any exception generate we want to do common task then then we can use try with single catch block.

If we want to do specific task or set of code want to execute then we have to use try with multiple catch block base upon the exception. But in this option we must know about name of exception classes.

In Java Integer is a type of wrapper class which help to convert string to number.

Integer.parseInt(str); : it is use to convert string to int.

Finally block : This block execute 100% sure if any exception generate or not.

File handing or connecting to database

try{

Open the file

Read or write or copy data from a file

}catch(Exception e) {

}finally {

Close the file

}

Finally block is use to close the resources property if any exception generate or not.

throw and throws

throw : throw keyword is use to generate or raise the pre defined or user defined exception base upon conditions.

Syntax

**throw new Exception();**

or

**throw new ExceptionSubClass();**

To create our own or custom exception we need to create normal java class and that class must be extends Exception class and we need to write two constructor

First one is empty and second one is parameter constructor with string parameter.

throws : this keyword is use to throw or pass the exception to caller method. This keyword we need to use with methods signature

syntax

returnType methodName() throws Exception {

}

void display() throws Exception {

}

Checked exception

CheckecException it check twice that is compile time as well as run time.

Checked exception we need to handle it mandatory otherwise we can’t compile programs.

Checked exception we can handle using try-catch or throws keyword.

Unchecked exception like ArithmeticException, ArrayIndexOutOfBoundsException

Some extends we can avoid it.

Unchecked exception it check only at run time .

Multi threading

Program : set of instruction to perform a specific task.

Processor : Processor is responsible to execute the code.

Process : time taken to execute the code

Thread : Thread is known as small execution of code within a process.

Thread also known as light weighted process.

Thread take less memory or less resource to do some task.

Process take more memory or more resource to do same task.

Thread is a part of process.

By default C and C++ is known as process base programming language

By default java is known as thread base programming language.

By default in Java inside a main method default thread will execute.

Thread t = Thread.currentThread();

Thread is pre defined class part of lang package which contains currentThread static method.

This method provide the details about current thread.

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

Thread 🡪 thread

main 🡪 name of the thread

5 🡪 priority of the thread

main 🡪 group of the thread

Day 13

15-11-2022

Multi tasking

1. Using process base
2. Using thread base

Creating user defined threads

In Java we can create two ways

1. Extends Thread class
2. Implements Runnable interface.

Extends Thread class

1. We need to create user defined class and that class must be extends Thread class.
2. Then we need to create the thread class reference(the class extends Thread class we have to create that object of that class).
3. With help of reference we need to call start(). This method is part of thread class which help to make ready to run.
4. Start() method internally call run() method of thread class. Run method part of thread class contains empty body.
5. So if we want to execute any custom code then we need to override run method inside that class which class extends Thread class.

Implements runnable interface

1. We need to create user defined class and that class must be implements Runnable interface.
2. Runnable interface contains run() method. And it is an abstract which we need to override mandatory.
3. When class implements Runnable interface we need to override run() method mandatory.
4. In main class we need to create Thread class and pass the object of that class which implements through constructor.
5. With help of thread reference call start() and start() method call run method.

Life cycle of thread

Create ----------🡪 runnable ------------------🡪 running -----------------🡪destroy

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

t1 t1.start(); run()

isAlive() : to check thread is running or not

sleep() : to make thread to pause few second

wait() : suspend the thread

notify() : resume only one thread

notifyAll() : resume more than one thread