**20-09-2021**

**Version Control System : Version control system that records changes on files or program or application in a projects.**

**3 types**

**Local Version control : RCS : Revision control system**

**Centralized version control : SVN**

**Distributed Version control system : Local repository and remote repository.**

**Git : Git is distributed Sub Version control system which help to manage the source code.**

**Distributed version control system keeps track of software version and allow many developers to work on given project within a maintaining connection of common network.**

**Git is a open source distributed version control system.**

**Create one folder My Repository**

**Please create simple text file or any type of time with some contents.**

**Open terminal in Virtual machine.**

**git --version**

**to make normal folder as a local git repository we have to**

**run the command as**

**git init : after run this command it will create .git folder.**

**This folder consider as a hidden folder in Unix or mac**

**git status :This command is use to check the last command status.**

**git add filename This command is use to add the file from local file system to staging area.**

**Staging area : it is a area generally represent in git directory, that store information about what will go to next commit.**

**It is a intermediate area between os file system and git local repository.**

**git commit –m “First commit”**

**all commands**

**git init**

**git status**

**git add filename**

**or**

**git add . : This command is use to add all files and folder**

**git status**

**git commit –m “message”**

**first this command**

**git config –-global user.email “akash300383@gmail.com”**

**second this command**

**git config –-global user.name “Akash”**

**No do the change in same file**

**Remote repository**

**Github is type of remote repository which help to share the data from one developer to another developer.**

**AWS : code commit**

**Gitlab**

**Azure :**

**So first create the account in github**

**Then create the repository.**

**To link local repository to remote repository we have to run the command as**

**Git remote add origin URL**

**git remote add origin** [**https://github.com/Kaleakash/Java\_FIS\_2021\_Batch.git**](https://github.com/Kaleakash/Java_FIS_2021_Batch.git)

**after connect local to remote**

**we have push local repository code to remote repository**

**before push we have to check default branch**

**git branch**

**master/main**

**git push –u origin master**

**or**

**git push –u origin main**

**git branch :**

**git branch is use to hold more than one commit details.**

**Branch is like a pointer which hold more than one commit.**

**Default branch name is master/main**

**git branch (master/main)**

**to create the user-defined branch**

**git branch branchName**

**command to switch to user-defined branch**

**git checkout branchName**

**To delete the branch**

**git branch –D branchName**

**merge the code**

**git merge branchName : This command is use to merge to code to current branch**

**create folder**

**then create the file**

**then git init**

**then git add .**

**git commit –m “message”**

**Syntax**

git remote add origin https://token@github.com/Kaleakash/test\_info.git

**To Generate token**

****

****

****

****

****

****

git remote add origin <https://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@github.com/Kaleakash/test_info.git>

after added url to origin variable we can push the data to remote repository

git push –u origin master

git remote remove origin

Java Training

**This command is use to download the new remote repository code in local machine**

**git clone https://github.com/Kaleakash/Java\_FIS\_2021\_Batch.git**

**After clone move inside a repository folder.**

**This command is use to pull new update from remote repository to local repository.**

**git pull**

**Java Training**

**What is Java.**

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

**C 1970**

**C++ 1980**

**Java 1990 :**

**Initial name of is Oak. 1991**

**Rename in Nov 1995 to Java.**

**James gosling and Team**

**Java was belong to Sun micro system but not it is a part of Oracle.**

**Version of Java**

**1.0 to 17**

**Java 8 mandatory to develop the application.**

**object : any real world entity.**

**Properties or state 🡪 have-🡪 fields / variables.**

**Person**

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

**Place**

**Bank**

**Animal**

**Car**

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

**Class syntax**

**class classname {**

**fields;**

**methods;**

**}**

**In Java class name must be follow Pascal naming rules.**

**If it contains one world first letter must be upper case. If it contains more than one world each world first letter upper case.**

**Demo.java**

class Demo {

    public static void main(String args[]){

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

        System.out.print("Welcome");

        System.out.printf("Welcome to Java");

    }

}

**Data types : it is a type of data which tell what type of data it hold.**

**2 types**

1. **Primitive data types : it is used to store only value** 
   1. **byte 1 byte**
   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 singe character**
   8. **boolean 1 bit : true or false.**

**Primitive data type example**

class Demo {

    public static void main(String args[]){

        int a=10;

        System.out.println(a);

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

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

    }

}

**Type casting :converting from one data type into another data type is known as type casting.**

**2 types**

1. **implicit type casting**
2. **explicit type casting**

**int family**

**------------------------implicit -------------🡪**

**byte short int long**

**🡨-----------------explicit ---------------------**

**Type casting example**

class Demo {

    public static void main(String args[]){

        byte a =10;

        short b =a;     // implicit type casting

        System.out.println(a);

        System.out.println(b);

        short c = 10;

        // (type)variableName;

        byte d = (byte)c;       // explicit type casting

        System.out.println(c);

        System.out.println(d);

    }

}

**Type casting int and float variables**

class Demo {

    public static void main(String args[]){

        int a=10;

        float b=a;      // implicit type casting

        System.out.println(a);

        System.out.println(b);

        //float c = (float)10.10;

        float c = 10.10f;       // explicit type casting

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

        System.out.println(c);

        System.out.println(d);

    }

}

**Operator :**

**If statement**

**If else**

**Nested if**

**If else if**

**Switch statement**

**Looping : looping is use to execute the statement again and again till conditions becomes false.**

**Initialization : start and end**

**Condition : true**

**Coding**

**Increment or decrement.**

**While loop**

**Do while loop**

**For loop**

**Looping Example**

class Demo {

    public static void main(String args[]){

        // System.out.println("While loop");    // entry loop

        // int i=1,n=10;

        // while(i<=n){

        //     System.out.println(i);

        //     i++;

        // }

        // System.out.println("Do While loop");    // exit loop

        // int i=1,n=10;

        // do{

        //     System.out.println(i);

        //     i++;

        // }while(i<=n);

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

        for(int i=0,n=10;i<=n;i++){

            System.out.println(i);

        }

    }

}

**In java every decimal number by default consider as double.**

1. **Non primitive data types or reference data types: it is used to store value as well as reference of another data types.** 
   1. **Array**
   2. **Class : pre-defined or user-defined**
   3. **Interface : pre-defined or user-defined**
   4. **Enum**

**array : array is used to store more than one value of same types.**

**syntax**

**datatype arrayName[];**

**for each or enhanced loop**

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

**}**

**Array with looping (for loop and enhanced loop)**

class Demo {

    public static void main(String args[]){

        int []abc;

        int []num = {10,20,30,40,50,60};

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

        System.out.println("Value of 1 position in num "+num[1]);

        System.out.println("using for loop");

        for(int i=0;i<6;i++){

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

        }

        System.out.println("for enhanced loop");

        for(int n:num){

            System.out.println(n);

        }

    }

}

**Creating the memory size for the array**

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

**GC()**

class Demo {

    public static void main(String args[]){

        int []abc={10,20,30,40,50};

        int []num = new int[10];

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

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

        System.out.println(abc[0]);

        System.out.println(num[0]);

        System.out.println(num[1]);

        int temp =100;

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

            num[i]=temp;

            temp++;

        }

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

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

        }

    }

}

**Taking the value through keyboards.**

1. **Using Scanner class**
2. **Using DataInputStream**
3. **BufferedReader**
4. **Command line arguments.**

**Scanner is a pre-defined class which provide set of methods which help to scan the value through keyboards.**

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

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

**Scanner pre-defined class part of util package.**

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

**In Java all method follow camel naming rules.**

1. **If method name contains one word it must be in lower case.**
2. **If method name contains more than one world from second word each word first letter upper case.**

**Taking the value through keyboards using Scanner class**

//import java.util.Scanner;

import java.util.\*;

class Demo {

    public static void main(String args[]){

        String msg = "Welcome";

        System.out.println(msg);

        Scanner obj = new Scanner(System.in);

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

        int id = obj.nextInt();

        obj.nextLine();             // it is use to hold enter key

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

        String name= obj.nextLine();

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

        float salary = obj.nextFloat();

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

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

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

    }

}

**Taking the value through keyboards**

//import java.util.Scanner;

import java.util.\*;

class Demo {

    public static void main(String args[]){

        Scanner obj = new Scanner(System.in);

        System.out.println("how many number do you want to store?");

        int n = obj.nextInt();

        int []num = new int[n];

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

        for(int i=0;i<n;i++){

            num[i]=obj.nextInt();

        }

        System.out.println("The number are");

        for(int i=0;i<n;i++){

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

        }

    }

}

**Take array value as id,name,salary and desg**

**If desg is manager we have to give 5000 bonus if desg is developer 3000 else 1500**

//import java.util.Scanner;

import java.util.\*;

class Demo {

    public static void main(String args[]){

        Scanner obj = new Scanner(System.in);

        System.out.println("how many number do you want to store?");

        int n = obj.nextInt();

        int []id=new int[n];

        String []name = new String[n];

        float []salary = new float[n];

        String []desg=new String[n];

        for(int i=0;i<n;i++){

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

            id[i]=obj.nextInt();

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

            name[i]=obj.next();

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

            salary[i]=obj.nextFloat();

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

            desg[i]=obj.next();

        }

        for(int i=0;i<n;i++){

            if(desg[i].equals("Manager")){

                  salary[i]=salary[i]+5000;

            }else if(desg[i].equals("Developer")){

                   salary[i]=salary[i]+3000;

            }else {

                salary[i]=salary[i]+1500;

            }

        }

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

        for(int i=0;i<n;i++){

            System.out.println("id is "+id[i]);

            System.out.println("name is "+name[i]);

            System.out.println("salary is "+salary[i]);

            System.out.println("desg is "+desg[i]);

        }

    }

}

**objet : any real world entity**

**properties or state**

**Car**

**Behaviour**

**Person**

**Bank**

**Animal**

**Customer**

**Employee**

**class : it is user-defined data type which help to create the**

**memory or object.**

**Syntax of class**

**class className {**

**}**

**Syntax to create the object**

**ClassName objectRefName = new ClassName()**

**objectRefName.methodName();**

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 CarTest {

    public static void main(String args[]){

        //System.out.println("Main method");

        //start();

        Car innova = new Car();     // heap memory

        innova.start();

        innova.stop();

    }

}

**Types of variable or fields**

**In Java variables are divided into three types.**

1. **Instance variable**
   1. **The variable which declare outside a method but inside a class is known as instance variable.**
   2. **Instance variable hold default value according to their data types. like int family 0, float family 0.0, char space, String null, boolean false.**
   3. **We can use instance variable directly inside a method but the method must be non-static and it must be part of same class.**
2. **Local variable**
   1. **The variable which declare inside a method is known as local variable.**
   2. **Local variable doesn’t hold default value we have to initialize.**
   3. **Scope of variable within that block where it declare.**
3. **Static variable**

**Instance and local variable example**

class Car {

    int wheel;

    float price;

    String color;

    void start() {

        int temp=100;

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

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

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

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

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

    }

    void stop() {

        String msg="Welcome";

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

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

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

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

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

    }

}

class CarTest {

    public static void main(String args[]){

        Car innova = new Car();     // heap memory

        innova.start();

        innova.stop();

    }

}

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

**Pts**

1. **Constructor have same name as class itself.**
2. **Constructor doesn’t contains return type not even void also.**
3. **Constructor get call automatically when we create the object.**

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

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

**Constructor and method example**

**class Employee {**

**Employee() {**

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

**}**

**void dis() {**

**System.out.println("This is a dis method");**

**}**

**}**

**class Test {**

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

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

**emp.dis();**

**emp.dis();**

**emp.dis();**

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

**emp1.dis();**

**emp1.dis();**

**}**

**}**

**Parameterized method and parametrized constructor**

**class Employee {**

**Employee() {**

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

**}**

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

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

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

**}**

**void dis() {**

**System.out.println("This is a dis method");**

**}**

**void passValue(int id, String name) {**

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

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

**}**

**}**

**class Test {**

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

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

**Employee emp2 = new Employee(100,"Ravi");**

**emp1.dis();**

**emp1.passValue(101,"Ramesh");**

**emp1.passValue(102,"Raju");**

**}**

**}**

**By default Java compiler provide default constructor if we doesn’t write any constructor inside a class. default constructor is always empty constructor. But if we write explicitly empty or parameter then there no default constructor.**

**Example**

class Operation {

    int a,b,sum;

    Operation() {

        a=10;

        b=20;

    }

    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 dis() {

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

    }

}

class Test {

    public static void main(String args[]) {

        Operation op1 = new Operation();                    op1.dis();// sum =0

        Operation op2 = new Operation();    op2.add();      op2.dis();// sum = 30

        Operation op3 = new Operation(1,2); op3.add();      op3.dis();  // sum = 3

        Operation op4 = new Operation();

        op4.setValue(5,6);

        op4.setValue(7,8);

        op4.setValue(9,10);

        op4.add();

        op4.dis();                  // sum

    }

}

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

**class :**

class Employee {

    String name;

    float salary;

    void dis() {

        System.out.println("This is dis method");

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

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

    }

}

class Test {

    public static void main(String args[]) {

        Employee emp = new Employee();

        emp.name = "Raj Deep";

        emp.salary = 12000;

        emp.dis();

    }

}

**When local variable or parameter variable and instance variable have same name then local or parameter variable hide the visibility of instance variable. If we want to refer to instance variable we have to use this.variableName**

**this is keyword which is use to refer the current object.**

class Employee {

    private String name;

    private float salary;

    void setValue(String name, float salary){

        // name = name;         // local variable = local variable

        // salary = salary;

        this.name = name;

        if(salary<0){

            this.salary = 8000;

        }else {

            this.salary = salary;

        }

        //this.salary = salary;

    }

    void dis() {

        System.out.println("This is dis method");

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

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

    }

}

class Test {

    public static void main(String args[]) {

        Employee emp = new Employee();

        //emp.name = "Raj Deep";

        //emp.salary = -12000;

        emp.setValue("Raj",-12000);

        emp.dis();

    }

}

**Create two classes Employee and EmpoyeeTest**

**Employee class must contains id,name,salary as instance variable with private access specifiers**

**It contains three methods**

**read(), calSalary and display()**

**read() method to takes id, name and salary from main method.**

**calSalary() method simple business logic you have to find hra, da and pf(ie local variable) hra is 10% on salary da is 5% on salary and pf is 7% on salary**

**in display you have to display id,name,salary(ie gross salary but don’t declare gross salary variable).**

**EmployeeTest contains main method. Create Scanner class object to receive the value through keyboards and pass the value to read method as a parameter take the value through keyboards.**

import java.util.Scanner;

class Employee {

    private int id;

    private String name;

    private float salary;

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

        this.id = id;

        this.name = name;

        this.salary = salary;

    }

    void calSalary() {

        float hra,da,pf;

        hra = salary\*0.10f;

        da = salary \*0.05f;

        pf = salary \* 0.07f;

        salary = salary + hra +da -pf;

    }

    void dis() {

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

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

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

    }

}

class EmployeeTest {

    public static void main(String args[]){

        Scanner obj = new Scanner(System.in);

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

        int id = obj.nextInt();

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

        String name = obj.next();

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

        float salary = obj.nextFloat();

        Employee emp = new Employee();

        emp.readValue(id,name,salary);

        emp.calSalary();

        emp.dis();

    }

}

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

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

**fields;**

**methods;**

**}**

**class NewClass extends OldClass{**

**fields; // sub class, derived class or child class**

**methods;**

**}**

**To achieve inheritance in java we have to use the extends keywords.**

**With help of sub class object we can access it own as well as super class properties and behaviour.**

class A {

    void dis1() {

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

    }

}

class B extends A{

    void dis2() {

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

    }

}

class EmployeeTest {

    public static void main(String args[]){

        A obj1  = new A();

        B obj2 = new B();

        obj1.dis1();

        obj2.dis2();

        obj2.dis1();

    }

}

**Types of inheritance**

1. **Single inheritance : one super class and one sub class.**

**class A { }**

**class B extends A{}**

1. **Multilevel inheritance : one super class and n number of sub class 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 connect directly to super class.**

**Class A {}**

**Class B extends A{}**

**Class C extends A{}**

**Class D extends A{}**

1. **Multiple inheritance : more than one super class and one sub class**

**Class A {}**

**Class B {}**

**Class C extends A,B{} : but java doesn’t support this type of inheritance using class we have to achieve using interface.**

**OOPs relationship**

1. **is a relationship**
2. **has a relationship**

**Manager/Programmer is a Employee**

**Manager/Employee/Programmer has a Address**

**class Employee {**

**id,name,salary**

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

**}**

**class Manager extends Employee{**

**numberOfEmp**

**}**

**class Programmer extends Employee {**

**projectName;**

**}**

**class Address {**

**city,state and pincode**

**}**

**Has a relationship**

1. **association**
2. **aggregation**
3. **composition**

**class A {**

**B obj1= new B(); 0, 1 or many**

**}**

**class B {**

**A obj2 = new A(); 0, 1 or many**

**}**

**class Manager {**

**Address add = new Address() 1 or many**

**}**

**class Address {**

**}**

**This is a type of association which is known as a week**

**Association. Week association is known as aggregation.**

**class Student {**

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

**}**

**class StudentHistory {**

**}**

**This is a type of association which is known as a strong**

**Association. Strong association is known as composition.**

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

**2 types**

**Compile time run time**

**Static binding dynamic binding**

**Early binding late binding**

**Method overloading method overriding**

**Method overloading : The method have same name different parameter list (type of parameter list or number of parameter list).**

class Abc {

    void add(int x, int y) {

        System.out.println(x+y);

    }

    void add(int x, int y, int z) {

        System.out.println(x+y+z);

    }

    void add(float x, float y) {

            System.out.println(x+y);

    }

    void add(String x, String y) {

            System.out.println(x+y);

    }

}

class EmployeeTest {

    public static void main(String args[]){

        Abc obj = new Abc();

        obj.add(10,20);

        obj.add(1,2,3);

        obj.add(10.10f,20.20f);

        obj.add("1","2");

     }

}

**Hierarchical inheritance**

class Bike {

    void speed() {

        System.out.println("6okm/hr");

    }

}

class Honda extends Bike {

    void color() {

        System.out.println("Red");

    }

}

class Pulsar extends Bike {

    void color() {

        System.out.println("Black");

    }

}

class Test {

    public static void main(String args[]){

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

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

     }

}

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

class Bike {

    void speed() {

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

    }

}

class Honda extends Bike {

    void color() {

        System.out.println("Red");

    }

}

class Pulsar extends Bike {

    void color() {

        System.out.println("Black");

    }

    void speed() {

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

    }

}

class Test {

    public static void main(String args[]){

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

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

     }

}

**Annotation : Annotation means meta-data. Meta-data means data about data. Java provided lot of pre-defined annotation and all annotation start with @ followed by name of the annotation.**

**Some annotation we can use on class level or method level or property level etc.**

**@Override : This annotation is method level.**

**Method override example**

class Bike {

    void speed() {

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

    }

}

class Honda extends Bike {

    void color() {

        System.out.println("Red");

    }

}

class Pulsar extends Bike {

    void color() {

        System.out.println("Black");

    }

    @Override

    void speed() {

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

    }

}

class Test {

    public static void main(String args[]){

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

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

     }

}

**Re-usability and override and merge the code**

class Bike {

    void speed() {

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

    }

}

class Honda extends Bike {              // re-usability

    void color() {

        System.out.println("Red");

    }

}

class Pulsar extends Bike {

    void color() {

        System.out.println("Black");

    }

    @Override                           // override method

    void speed() {

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

    }

}

class Tvs extends Bike {

    void color() {

        System.out.println("Gray");

    }

    @Override

    void speed() {

        super.speed();                              // super class speed method code

        System.out.println("20km/hr");              // merge both class method code

    }

}

class Test {

    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();

     }

}

**abstract keyword**

1. **abstract 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 methods.**

**abstract returnType methodName(parameterList);**

1. **If class contains abstract method then we have to declare the class as abstract class.**

**abstract class className {**

**}**

1. **Which ever class extends abstract class that class must be provide the body for all abstract method belong to that class mandatory. That class can ignore if that class itself is abstract class.**
2. **abstract class we can’t create the object.**
3. **Abstract class can contains normal as well as abstract methods ie 0 or 1 or many.**
4. **Abstract class can contains constructor it may be default or parameterized. (Because it contains instance variable).**

**Static keyword :**

1. **Static keyword we can use with variable and method but not with class (if class is inner class we can use static keyword).**
2. **Static variable : if variable is static we can assign the value for that variable with the help of class name. Even though we can assign the value for that variable using object also.**
3. **Static method : if method is static we can call that method with the help of class name. Even though we can call static method with the help of object also.**
4. **Inside non static method we can access static as well as non static variable directly. But insider static method we can access only static variable directly. So if we want to access we have to create the object.**

**Static 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);

    }

    static void dis2() {

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

        Abc obj = new Abc();

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

        System.out.println("b "+b);

    }

}

class Test {

    public static void main(String args[]){

    Abc.b=10;

    Abc obj1 = new Abc();

    obj1.a=20;

    Abc.dis2();

    obj1.dis2();

    obj1.dis1();

    }

}

**Static is global to all object.**

**So don’t declare your id, name and salary as static. ProjectId, ClientInfo, ManagerId must static.**

**final**

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

1. **final variable : to declare a constant in java we use final keyword with variable.** 
   1. **Syntax final int A=10;**
2. **final method : if method is final we can’t override that method. sub class can use method but can’t override that method.** 
   1. **final void speed() {**

**}**

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

**final class Demo {**

**}**

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

**Example of final keywords**

final class A{

    final void dis1() {

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

    }

}

class B extends A{

    // void dis1() {

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

    // }

}

class Test {

    public static void main(String args[]){

        final int A =10;

        System.out.println(A);

        //A=20;

        B obj1 = new B();

        obj1.dis1();

    }

}

Interface : interface is known as 100% pure abstract class till java7. Interface also known as reference data types.

Syntax for interface

interface interfaceName {

fields;

methods;

}

By default all fields in interface are public, static and final.

By default all methods are public and abstract.

interface A {

public static final int M=10;

public abstract void dis1();

}

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

**Class always implements to interface and class can implements more than one interface.**

**Access specifiers while method overriding**

**Super class / interface sub**

**public public**

**protected public**

**protected**

**default public**

**protected**

**default**

**private can’t override**

interface A {

    int M=10;

    void dis1();

}

interface B {

    int N=20;

    void dis2();

}

interface C extends A,B{        // using interface we can achieve multiple inheritance.

    int O=30;

    void dis3();

}

class D implements A,B {

    public void dis1() {

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

    }

    public void dis2() {

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

    }

}

class Test {

    public static void main(String args[]){

        D obj1 = new D();       obj1.dis1();    obj1.dis2();

    }

}

**Another Example of interface**

interface A {

    int M=10;

    void dis();

}

interface B {

    int M=20;

    void dis();

}

class D implements B,A {

    public void dis() {

        System.out.println("A and B interface method");

        System.out.println(A.M +" "+B.M);

    }

}

class Test {

    public static void main(String args[]){

        D obj1 = new D();

        obj1.dis();

    }

}

**Difference between interface and abstract class.**

1. **Abstract class can contains normal variable as well as final variable. But interface contains only final variable.**
2. **Abstract class can contains normal method as well as abstract methods. But interface contains only abstract methods.**
3. **Abstract class can contains default as well as parameterized constructor but interface doesn’t contains constructor.**
4. **Abstract class can extends only one abstract class but interface can extends more than one interface**
5. **Using abstract class we can achieve partial abstraction but using interface we can achieve 100% pure abstraction.**

**Run time polymorphism**

class A {

    void dis1() {

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

    }

}

class B extends A {

    void dis1() {

        System.out.println("A class method Override by B");

    }

    void dis2() {

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

    }

}

class Test {

    public static void main(String args[]){

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

        B obj2 = new B();

        obj2.dis1();    obj2.dis2();

        //B obj3 = new A();     // super class object and sub class reference   : not possible

        A obj4 = new B();       // sub class object adn super class reference   : possible

                                // with the help of super class reference we can call only those methods

                                // which belong to super class or overrided methods.

        obj4.dis1();    //obj4.dis2();

    }

}

**2nd Example**

abstract class A {

    abstract void dis1();

}

class B extends A {

    void dis1() {

        System.out.println("A class method Override by B");

    }

    void dis2() {

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

    }

}

class Test {

    public static void main(String args[]){

         A obj1 = new B();  // sub class object and super class reference

                            // super class can be abstract class.

         obj1.dis1();

    }

}

**3rd example**

interface A {

    void dis1();

}

class B implements A {

    public void dis1() {

        System.out.println("A interface method Override by B");

    }

    void dis2() {

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

    }

}

class Test {

    public static void main(String args[]){

         A obj1 = new B();  // sub class object and super interface reference

                            // super can be interface reference.

         obj1.dis1();

    }

}

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

**Using interface we can hide the internal logic.**

interface A {

    int add(int x, int y);

}

interface B {

    int sub(int x, int y);

}

class Server implements A,B{

    public int add(int x, int y){

        return x+y;

    }

    public int sub(int x, int y){

        return x-y;

    }

}

class Test {

    public static void main(String args[]){

        Server ss = new Server();

        System.out.println(ss.add(2,3));

        System.out.println(ss.sub(4,3));

        A obj1  = new Server();

        System.out.println(obj1.add(2,3));

        //System.out.println(obj1.sub(4,3));

        B obj2 = new Server();

        //System.out.println(obj2.add(2,3));

        System.out.println(obj2.sub(4,3));

    }

}

**interface is use to provide the specification and class is use to provide the implementation.**

**this() and super()**

**this() parameter is use to invoke same class constructor like constructor chaining**

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

class Employee {

    Employee() {

        this(10);           // one parameter constructor it must first statment insider

                                    // constructor

        System.out.println("()");

    }

    Employee(int id) {

            this(100,"Raj");        // two parameter constructor

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

    }

    Employee(int id, String name){

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

    }

}

class Test {

    public static void main(String args[]){

        Employee emp1 = new Employee();

    }

}

class Employee {

    int id;

    String name;

    Employee() {

            this.id = 123;

            this.name = "Unknown";

    }

    Employee(int id) {

            this();

            this.id = id;

    }

    Employee(int id, String name){

            this(id);

            this.name  = name;

    }

    void dis() {

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

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

    }

}

class Test {

    public static void main(String args[]){

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

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

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

  }

}

**By default every sub class constructor contains super() as a first statement . Which help to invoke sub class constructor to super class constructor. Which always call super class empty constructor if you want to call parameter constructor we can customize it.**

class Employee {

        Employee() {

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

        }

        Employee(int id ){

            System.out.println("Employee object object with one parameter");

        }

}

class Manager extends Employee {

    Manager() {

        super();

        System.out.println("Manager class object");

    }

}

class Test {

    public static void main(String args[]){

    Manager mgr = new Manager();

    }

}

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

**Divided into two types**

1. **User-defined package**
2. **Built in package or pre-defined package.**

**When we create more than one class or interface which have same but different purpose.**

**Package is like a directory or folder.**

**Education**

**School college pg**

**Attendance Attendance Attendance**

**Syntax to create the user-defined package**

**package packageName;**

**package example**

package com;

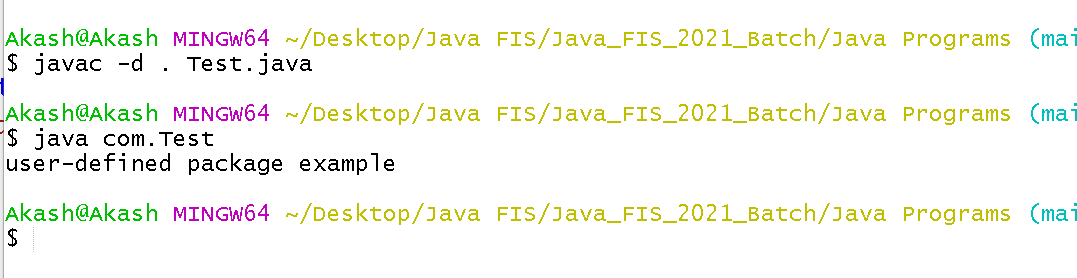
class Test {

    public static void main(String args[]){

    System.out.println("user-defined package example");

    }

}

****

**Access specifiers**

**4 types**

1. **private : we can use with instance variable, static variable, non static method and static method, constructor but not with class as well as local variable. Scope : within a class.**
2. **default (nothing): all. Scope within same package.**
3. **protected : we can use with instance variable, static variable, non static method and static method, constructor but not with: scope within a same package other package if it is sub class**
4. **public : we can access with all except local variable. Scope we can access same package as well as other package.**

**Exception Handling**

**Exception is a object which occurs when unexpected or abnormal condition during the execution of a programs.**

**Java Program**

**Compile run time**

**javac java**

**compile time error run time error**

**syntax error**

**Run time error**

**Error Exception**

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

**Error : The error which generated at the rum time which we can’t handle it. JVM crash, out of memory or software or hardware etc.**

**Exception: It is type of run time error which we can handle it. Divided by zero etc.**

**Object**

**Throwable**

**Exception**

**Checked exception un checked exception**

**RuntimeException**

**IOException ArithmeticException**

**SQLException ArrayIndexOutOfBoundsException**

**InterruptedException NumberFormatException**

**FileNotFoundException NullPointerException**

**Etc etc**

**To handle to both the type of exception java provided 5 keywords.**

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

**try and catch block**

**syntax**

**try {**

**}catch(Exception e) {**

**}**

**This code is use to handle all type of generic exception.**

**try with multiple catch block**

**try {**

**}catch(ArithmeticException e) {**

**}catch(ArrayIndexOutOfBoundsException e) {**

**}**

**finally block : it is type of block which will execute 100% sure if any exception generate or not.**

**try**

**catch catch catch catch finally**

**catch finally catch**

**catch finally**

**finally block mainly use to close the resource like file handling or databases.**

**throw : throw keyword is use to generate or raise the pre-defined or user-defined exception.**

**Syntax**

**throw new Exception();**

**or**

**throw new ExceptionSubClass();**

**main class**

**package** com;

**public** **class** ThrowDemo {

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

**int** a=100;

**int** b=20;

**try** {

**if**(a>b) {

//throw new Exception();

//throw new ArithmeticException();

//int res = 10/0;

//throw new ArithmeticException("a>b");

//throw new MyException();

**throw** **new** MyException("a>b");

}

}**catch**(Exception e) {

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

}

}

}

**Custom Exception class**

**package** com;

**public** **class** MyException **extends** Exception{

**public** MyException() {

}

**public** MyException(String msg) {

**super**(msg); // pass the message to Exception super class

}

}

**throws : throws keyword is use to throw the exception to caller methods.**

**Syntax**

**returnType methodName(parameterList) throws Exception, ExceptionSubClass {**

**}**

**void dis() throws Exception{**

**}**

**Throws Exception**

**package** com;

**public** **class** ThrowsDemo {

**static** **void** dis1() **throws** Exception{

//try {

**int** a=10/0;

//}catch(Exception e) {}

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

}

**static** **void** dis2() **throws** Exception{

//try {

*dis1*();

//}catch(Exception e) {}

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

}

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

**try** {

*dis2*();

}**catch**(Exception e) {}

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

}

}

**24-09-2021**

**Multithreading Overview**

**Program : set of instruction to perform a specific task.**

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

**Process : program in execution or time taken to execute the code.**

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

**Thread also known as light weighted process.**

**By default java is thread base programming language.**

**Inside a main always default thread will execute.**

**Thread t = Thread.currenThread();**

**currentThread is a static method part of Thread class and return type of that method is Thread class reference.**

**t -🡪 Thread[main,5,main]**

**main -🡪 name of the thread**

**5-🡪 priority of the thread min 1 max=10 norm 5**

**main 🡪group of the thread**

**multi tasking :**

1. **process base : C or C++**
2. **thread base : java**

**In java we can create multiple thread using 2 ways**

1. **extends Thread class**
   1. **create normal java class and extends Thread class.**
   2. **create Thread class reference.**
   3. **Using thread class reference you have to call start() method which is part of thread class.**
   4. **Start method internally call run() method which is part of thread class.**
2. **implements Runnable interface**
   1. **create normal java class and implements Runnable interface.**
   2. **Runnable interface contains one method ie run method so we have to override that method mandatory.**
   3. **We have to create the object of Thread class and pass the reference of Runnable interface through constructor.**

**Using Thread class example**

**package** abc;

**class** A **extends** Thread{

@Override

**public** **void** run() {

**for**(**int** i=0;i<10;i++) {

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

}

}

}

**class** B **extends** Thread{

@Override

**public** **void** run() {

**for**(**int** j=0;j<10;j++) {

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

}

}

}

**public** **class** Test {

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

A obj1 = **new** A(); // thread class reference.

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

obj1.start();

obj2.start();

}

}

**Using implements Runnable interface**

**package** xyz;

**class** A **implements** Runnable{

@Override

**public** **void** run() {

**for**(**int** i=0;i<10;i++) {

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

}

}

}

**class** B **implements** Runnable{

@Override

**public** **void** run() {

**for**(**int** j=0;j<10;j++) {

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

}

}

}

**public** **class** Test {

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

Runnable obj1 = **new** A(); // obj1.start();

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

Thread t1 = **new** Thread(obj1);

Thread t2 = **new** Thread(obj2);

t1.start(); // A class override run method

t2.start(); // B class override run method

}

}

**Life cycle of thread**

**sleep()**

**wait()**

**notify()**

**notifyAll()**

**Create-----🡪Runnable --------🡪running --------🡪Destroy**

**obj1 obj1.start() run i=10**

**t1 t1.start(); run j=10**

**setName, getName, sleep method code**

**package** mno;

**class** A **implements** Runnable{

@Override

**public** **void** run() {

Thread t = Thread.*currentThread*();

String name = t.getName();

**for**(**int** i=0;i<10;i++) {

System.***out***.println(name+"="+i);

**try** {

Thread.*sleep*(500);

}**catch**(Exception e) {}

}

}

}

**class** B **implements** Runnable{

@Override

**public** **void** run() {

Thread t = Thread.*currentThread*();

String name = t.getName();

**for**(**int** j=0;j<10;j++) {

System.***out***.println(name+"="+j);

**try** {

Thread.*sleep*(1000);

}**catch**(Exception e) {}

}

}

}

**public** **class** Test {

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

Runnable obj1 = **new** A(); // obj1.start();

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

Thread t1 = **new** Thread(obj1);

Thread t2 = **new** Thread(obj2);

t1.setName("Demo1");

t2.setName("Demo2");

t1.start(); // A class override run method

t2.start(); // B class override run method

}

}

**Creating more than one thread in same memory.**

**Synchronization : synchronization is use to lock or block or allow to use all resource for only one thread at time.**

**To achieve synchronization we have to use synchronized keyword. We can use with method or block also.**

**Example**

**package** booking;

**class** Booking **implements** Runnable{

**int** avl=1;

@Override

**public** **synchronized** **void** run() {

Thread t = Thread.*currentThread*();

String name = t.getName();

**if**(avl==1) {

System.***out***.println(name +" got the ticket");

avl = avl-1;

}**else** {

System.***out***.println(name +" no ticket");

}

}

}

**public** **class** Test {

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

Runnable obj1 = **new** Booking(); //Runnable obj2 = new Booking();

Thread t1 = **new** Thread(obj1); Thread t2 = **new** Thread(obj1);

t1.setName("Raj"); t2.setName("Seeta");

t1.start(); t2.start();

}

}

t4

t1, t2, t3

**Inner thread communication using wait, notify and notifyAll**

**wait() : This method is use to suspend the thread with some condition.**

**notify() : this method is use to resume suspended thread.**

**More than one thread created in same memory and method must be synchronized.**

**Example**

**package** info;

**class** Abc **implements** Runnable{

@Override

**public** **synchronized** **void** run() {

Thread t = Thread.*currentThread*();

String name = t.getName();

**for**(**int** i=0;i<10;i++) {

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

**try** {

Thread.*sleep*(500);

**if**(i==5 && name.equals("Raj")) {

wait();

}

**if**(i==7 && name.equals("Raju")) {

notify();

wait();

}

}**catch** (Exception e) {}

}

}

}

**public** **class** Test {

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

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

Thread t1 = **new** Thread(obj1); t1.setName("Raj");

Thread t2 = **new** Thread(obj1);t2.setName("Raju");

Thread t3 = **new** Thread(obj1);t3.setName("Ramesh");

t1.start(); t2.start(); t3.start();

}

}

**Lang package**

**String class : In java String is pre-defined class.**

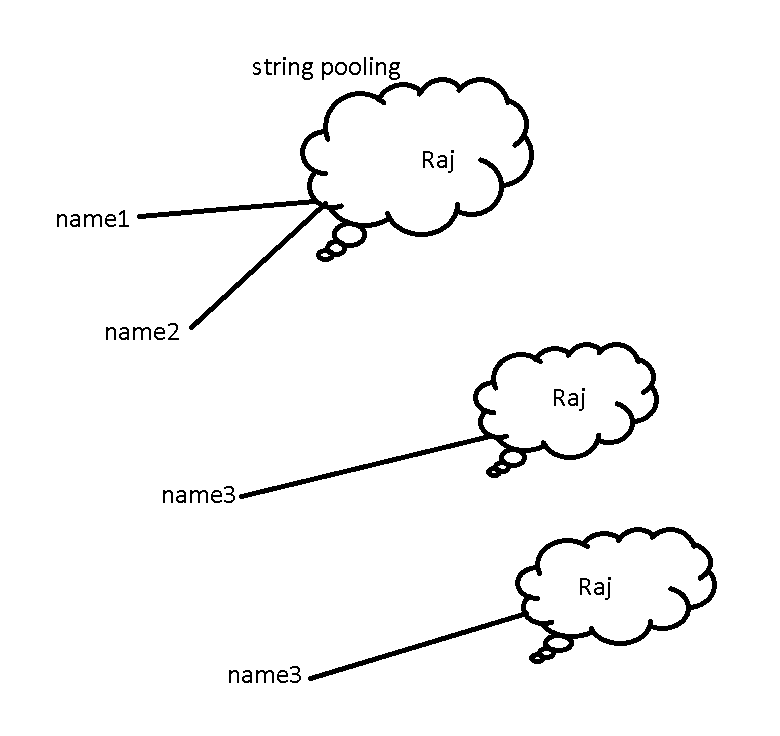
**Syntax to create the String object**

**Using literal style**

**String str1 = “Welcome to Java Training”;**

**Using new keyword.**

**String str2 = new String(“Welcome to Java Training”);**

****

**StringBuffer and StringBuilder**

**Both are type of string classes is known as mutable string classes.**

**StringBuffer method are synchronized so it is thread safe but performance wise slow.**

**StringBuilder not thread safe but performance wise fast.**

**IO package : Input and Output operation**

**In Java we can do the io operation using stream.**

**Stream is known as flow of data or it is a abstraction between source and destination.**

**Io stream operation**

**byte char**

**input output input output**

**InputStream OutputStream Reader Writer**

**All four are abstract classes.**

**DataInputStream DataOutputStream InputStreamReader OutputStreamWriter**

**FileInputStream FileOutputStream FileReader FileWriter**

**BufferedInputStream BufferedOutputStream BufferedReader BufferedWriter**

**ObjectInputStream ObjectOutputStream**

**PrintStream PrintWriter**

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

**System.in**

**System is a pre-defined class which contains three static fields ie in, out and err.**

**PrintStream ps = System.out;**

**PrintStream references always refer to standard output device ie console.**

**InputStream is = System.in;**

**InputStream reference always refer to standard input device it keyword.**

**Byte wise**

**Source :keyboards**

**Destination : console**

**DataInputStream**

**PrintStream**

**package** com;

**import** java.io.DataInputStream;

**import** java.io.PrintStream;

**public** **class** Demo1 {

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

DataInputStream dis = **new** DataInputStream(System.***in***);

PrintStream ps = System.***out***;

ps.println("Enter the name");

String name = dis.~~readLine~~();

ps.println("name is "+name);

}

}

**Byte wise**

**Source :keyboards**

**Destination : file**

**DataInputStream**

**FileOutputStream**

**package** com;

**import** java.io.DataInputStream;

**import** java.io.FileOutputStream;

**public** **class** Demo2 {

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

DataInputStream dis = **new** DataInputStream(System.***in***);

FileOutputStream fos = **new** FileOutputStream("abc.txt");

System.***out***.println("enter the data");

**int** ch;

**while**((ch=dis.read()) != '@' ) {

fos.write(ch);

System.***out***.print((**char**)ch);

}

fos.close();

}

}

**Byte wise**

**Source : file**

**Destination : file**

**FileInputStream**

**FileOutputStream**

**package** com;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**public** **class** Demo3 {

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

FileInputStream fis = **new** FileInputStream("C:\\Users\\91990\\Desktop\\Java FIS\\Java\_FIS\_2021\_Batch\\Java Programs\\Demo.java");

FileOutputStream fos = **new** FileOutputStream("Demo.java");

**int** ch;

**while**((ch=fis.read()) != -1) { // -1 is equal to EOF

fos.write(ch);

}

fis.close();

fos.close();

System.***out***.println("File copied successfully");

}

}

**BufferedInputStream and BufferedOutputStream**

**Buffered is a temporary memory.**

**Input File Buffer**

**Program Buffer**

**Output file**

**If we connect the input or output file with buffer 1000 times improve the performance.**

**package** com;

**import** java.io.BufferedInputStream;

**import** java.io.BufferedOutputStream;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**public** **class** Demo4 {

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

FileInputStream fis = **new** FileInputStream("C:\\Users\\91990\\Desktop\\Java FIS\\Java\_FIS\_2021\_Batch\\Java Programs\\Demo.java");

BufferedInputStream bis = **new** BufferedInputStream(fis);

FileOutputStream fos = **new** FileOutputStream("Demo1.java");

BufferedOutputStream bos = **new** BufferedOutputStream(fos);

**int** ch;

**while**((ch=bis.read()) != -1) { // -1 is equal to EOF

bos.write(ch);

}

bos.flush(); // send the data from buffer output to actual file

fis.close();

fos.close();

System.***out***.println("File copied successfully");

}

}

**Character wise classes**

**Character wise**

**Source : keyword**

**Destination : console**

**package com;**

**import java.io.BufferedReader;**

**import java.io.InputStreamReader;**

**public class Demo5 {**

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

**// InputStreamReader isr = new InputStreamReader(System.in);**

**// BufferedReader br = new BufferedReader(isr);**

**BufferedReader br = new BufferedReader(new InputStreamReader(System.in));**

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

**String name = br.readLine();**

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

**int id = Integer.parseInt(br.readLine()); // string to integer**

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

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

**}**

**}**

**File class : This class is use to find the properties of a**

**Specific file.**

**package com;**

**import java.io.FileReader;**

**import java.io.FileWriter;**

**public class Demo6 {**

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

**FileReader fr = new FileReader("Demo.java");**

**FileWriter fw = new FileWriter("C:\\Users\\91990\\Desktop\\Java FIS\\Demo.txt");**

**int ch;**

**while((ch = fr.read()) != -1) {**

**fw.write(ch);**

**}**

**fr.close();**

**fw.close();**

**System.out.println("File stored...");**

**}**

**}**

**Object serialization**

**Storing the object into external file or storing the state of object in external file or converting object into byte or serializable format is known as object serialization.**

**Object**

**Identity : reference**

**Property : variable**

**Behaviour : methods**

**We are storing their property not reference or behaviour.**

**Which class object storing in external file or converting into byte format that class must be implements Serializable interface.**

**Serializable interface is type of marker interface which doesn’t contains any methods(zero methods).**

**package** com;

**import** java.io.Serializable;

**public** **class** Employee **implements** Serializable{

**private** **int** id;

**private** String name;

**private** **float** salary;

**public** Employee() {

}

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

**this**.id = id;

**this**.name = name;

**this**.salary = salary;

}

**public** **void** display() {

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

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

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

}

}

**package com;**

**import java.io.FileInputStream;**

**import java.io.FileOutputStream;**

**import java.io.ObjectInputStream;**

**import java.io.ObjectOutputStream;**

**public class EmployeeTest {**

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

**Employee emp = new Employee(100, "Raj", 12000);**

**emp.display();**

**FileOutputStream fos = new FileOutputStream("emp.ser");**

**ObjectOutputStream oos = new ObjectOutputStream(fos);**

**oos.writeObject(emp);**

**System.out.println(" Object Serialization done successfully");**

**// FileInputStream fis = new FileInputStream("emp.ser");**

**// ObjectInputStream ois= new ObjectInputStream(fis);**

**// Object obj= ois.readObject();**

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

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

**// System.out.println("Object De-Serialization done successfully");**

**}**

**}**

**Normal class Vs Java Bean class**

**Normal class**

1. **Normal class access specifiers may be public or default.**
2. **In normal java class instance variable may be private or default or public or protected.**
3. **If variable are private then we have to provide helper methods to set the value with condition.**

**JavaBean class**

1. **JavaBean class access specifiers must be public.**
2. **All instance variable must be private.**
3. **For each variable we have to provide setter and getter methods.**
4. **Setter method is use to set the value and getter method is use to get the value.**
5. **Setter method name must be start with set followed by variable name**
6. **Getter method name start with get followed by variable name.**

**public class Employee {**

**private String name;**

**private float salary;**

**public void setName(String name) {**

**this.name = name;**

**}**

**public void setSalary(float salary) {**

**this.salary = salary;**

**}**

**public String getName() {**

**return name;**

**}**

**public float getSalary() {**

**return salary;**

**}**

**}**

**JavaBean class**

**package** bean;

**public** **class** Employee {

**private** **int** id;

**private** String name;

**private** **float** salary;

**public** Employee() {

**super**();

// **TODO** Auto-generated constructor stub

}

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

**super**();

**this**.id = id;

**this**.name = name;

**this**.salary = salary;

}

**public** **int** getId() {

**return** id;

}

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

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **float** getSalary() {

**return** salary;

}

**public** **void** setSalary(**float** salary) {

**this**.salary = salary;

}

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";

}

}

**Main class**

**package** com;

**public** **class** Test {

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

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

emp1.setId(100);

emp1.setName("Raj");

emp1.setSalary(12000);

bean.Employee emp2 = **new** bean.Employee(101, "Ramesh", 14000);

System.***out***.println(" id is "+emp1.getId());

System.***out***.println("name is "+emp1.getName());

System.***out***.println("salary is "+emp1.getSalary());

System.***out***.println(emp2); // internally it call toString() of Object class.

}

}

**Collection Framework**

**Collection framework is like a data structure in Java.**

**int abc; it can hold only one value**

**int num[]; it can hold more than one value of type int**

**class Employee {**

**int id;**

**String name;**

**float salary;**

**}**

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

**emp.id=100;**

**emp.name=”Raj”;**

**emp.salary = 12000;**

**array object**

**Employee employees[]=new Employees[100];**

**Array object is known fixed memory size data structure.**

**Array object can hold only specific type of object.**

**Array object doesn’t provide any pre-defined method to do some operation like add, remove, search etc.**

**Collection framework contains set of collection of classes as well as interfaces which help to store the collection of object or elements. These classes provide set of methods which help to add, remove, search we can do very easily.**

**Collection framework hierarchy**

**java.util.\*;**

**Collection --🡪 interface**

**extends extends extends doesn’textends**

**Set List Queue Map**

**All four are interfaces**

**HashSet Stack PriorityQueue HashMap**

**LinkedHashSet ArrayList LinkedHashMap**

**TreeSet LinkedList TreeMap**

**Vector Hashtable**

**Set : it is use to store collection of Elements or objects. Set doesn’t allow duplicate. In set some elements are maintain order, unorder or sorted.**

**HashSet : display the elements randomly or un-order.**

**LinkedHashSet : This class internally extends HashSet class and it maintain the order.**

**TreeSet : TreeSet internally implements SortedSet interface. So TreeSet display the element in ascending order. By default in TreeSet we can’t store different types of values.**

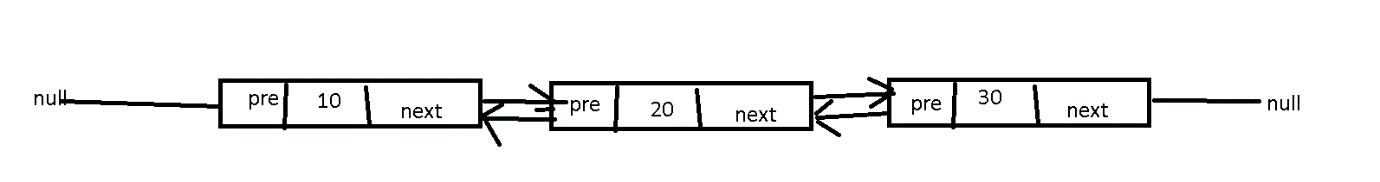
**List : it allow duplicate and maintain the order.**

**Stack : Stack is a type data structure which follow FILO(First In Last Out).method invocation internally follow stack concept.**

**ArrayList : Normal array is known as fixed memory size. It allow to store same type of values. Doesn’t provide any pre-defined methods.**

**But ArrayList by nature allow store any type of values. Doesn’t bother about memory size and it provide lot pre-defined methods which help to some operation on objects.**

**LinkedList : In Java LinkedList internally follow double linkedlist concept**

****

**Vector : vector is known a legacy class. By default in Vector class methods are synchronized. Work is safe but performance slow.**

**Queue : First In First Out. It allow duplicate elements.**

**Queue : First In First Out :**

**PriorityQueue : PriorityQueue First In First Out base upon priority.**

**Map : it store in the information in the form of key-value pairs where key is unique and value may be duplicate.**

**HashMap : un order.**

**Key as well as value can be null.**

**LinkedHashMap : maintain the order**

**TreeMap : ascending order as key value can be null**

**Hashtable : by default method are synchronized**

**It doesn’t allow key and value as null.**

**Retrieve the records from collection of classes one by one.**

1. **For each loop or enhanced loop : it is a type of loop**
2. **Iterator**
3. **ListIterator**
4. **Enumeration : three interfaces.**

**Set family : for each or enhanced loop**

**Iterator**

**Collection utilities classes**

**Arrays : This class provide set of static methods which help to do some operation on array of any types.**

**Collections : This class provide set of static methods which help to do some operation on List reference.**

**Service class : doing business logic on entity.**

**add, update, delete, retrieve etc.**

**please add two more business methods**

**delete employee records using id**

**update employee salary using id**

**Java 8 Features**

**Interface**

**From Java8 interface can contains method with body.**

**But the method must be default or static.**

**interface Bank {**

**void withdraw();**

**void deposit();**

**default void gst(){default logic } we can override**

**static void si(){} we can use it**

**but can’t override.**

**}**

**class Hdfc implements Bank {**

**void withdraw(){}**

**void deposit(){}**

**}**

**class Hsbc implements Bank {**

**void withdraw(){}**

**void deposit(){}**

**}**

**functional interface : The interface contains only one abstract method is known as functional interface. It can contains more than one static as well as default methods.**

**Inner classes**

**Class within another class is known as inner class**

**4 types of inner classes**

1. **Non static inner class**
2. **Static inner class**
3. **Anonymous inner class**
4. **Local inner class**

**Lambda Expression : lambda expression is known as anonymous function. But lambda expression we be use only for functional interface. From Java8 onwards using have we can do functional programming language.**

**Lambda expression by default return value without return keywords.**

**Pre-defined functional interfaces in Java8**

**Java8 introduce function package. It is a sub package of util package. Top most functional interfaces part of function package.**

1. **Function interface : method apply() takes T parameter and return R value.**
2. **Consumer interface : method accept() : Take T parameter and no return type.**
3. **Supplier interface : method get() : doesn’t take any parameter but return T value.**
4. **Predicate interface : method test() : takes T parameter and return boolean value.**

**Java Stream API.**

**A collection framework is an in-memory data structure to hold values and before we start load the data from collection framework. All the values should have been populated. Stream API is a data structure that is computed on-demand. Java Stream API doesn’t store data where collection framework store the data.**

**Stream represents on which one or more operation can be performed. Stream operation or operator are either intermediate or terminator. Intermediate operator return type stream itself. Terminator operator return type is non stream.**

**Source Stream--🡪I1-🡪I2🡪I3-->Terminal Operator**

**Array**

**Collection**

**classes**

**Ant : Build tool**

**Maven : Maven is known as build tool. Build tool is responsible to compile the program, run the program, package the program using jar, war or ear file, creating documentation for the file, downloading dependencies for the project or jar files.**

**Maven is base upon the XML file.**

**Maven contains one configuration file ie pom.xml file**

**Project object model.**

**Maven provided goals which comes under the life cycle of maven.**

**Mvn clean**

**Mvn compile**

**Mvn test**

**Mvn install**

**Mvn run**

**Every goal execute specific task.**

**Gradle**

**Creating the maven project through command prompt**

**mvn archetype:generate**

**Please hit enter key**

**Again enter key**

**It will ask groupId**

**groupId : MyProjects**

**artifactId : MySimpleMavenProject**

**version : 🡪 enter**

**package name : com**

**Then it will ask the configuration Y or N**

**Please enter Y**

**After created project please check the project structure**

**Do some change in App.java program**

**Move inside project using command as**

**Cd projectName (make sure your terminal open in place where pom.xml file present).**

**mvn validate : This command is use to validate the project**

**mvn compile :this command is use to compile the project. It create target folder which contains .class files.**

**mvn clean : it delete target folder.**

**To run the core java project we have to use the command as**

**mvn exec:java -Dexec.mainClass="com.App"**

**MySQL Database**

**Open the terminal**

**sudo mysql –u root –p**

**Simplilearn : password**

**mysql :**

**Create the database**

**create database fisdb : this command is use to create the database**

**use fisdb : this command is use to switch inside existing database.**

**create table employee(id int primary key,name varchar(10), salary float);**

**insert into employee values(1,'Raj',12000);**

**select \* from employee;**

**delete from employee where id =4;**

**update employee set salary = 22000 where id =3;**

**In Java We can connect the Database (Any RDBMS database using 4 ways).**

1. **Using JDBC (Java Database Connectivity)**
2. **Using ORM : Object Relation Mapping** 
   1. **Hibernate**
   2. **JPA**
3. **Spring Data**

**JDBC : Java Database Connectivity : JDBC is a API (Application Programming interfaces) which provide set of classes and interfaces which help to connect the database through Java Application.**

**Steps to connect the database.**

1. **Import the package : java.sql.\* and javax.sql.\*;**
2. **Write a method with exception handling concept ie user-defined or main method. Because JDBC through checked exception. Use try catch or throws.**
3. **Load the Driver : Driver is a pre-defined class provided by vender whose database going to connect. Vendor provide class in the form jar file which help to connect the database.**

**4 types of driver**

1. **Type 1 driver or JDBC ODBC Bridge driver (remove from Java8)**
2. **Type 2 Driver or JDBC native api driver**
3. **Type 3 Driver or jdbc net protocol driver (Data source)**
4. **Type 4 driver or jdbc thin or pure driver**

**Class.forName(“driverName”)**

**Class is a pre-defined class name itself Class part of lang package. Which contains forName() static method which help load the driver or load the any user-defined or pre-defined class.**

**com.mysql.jdbc.Driver : mysql**

**com.mysql.cj.jdbc.Driver : mysql**

**oracle.jdbc.driver.OracleDriver : oracle**

1. **Establish the connection :**

**DriverManager is a pre-defined class contains getConnection static method which takes three parameter ie url, username and password. This method return type is Connection interface reference.**

1. **Create Statement or PreparedStatement : both are interfaces which provide set of methods which help to do the operation on table.**

**Statement stmt = con.createStatement();**

1. **Retrieve : stmt.executeQuery(“select query”): This method return type is ResultSet interface reference.**

**ResultSet rs = stmt.executeQuery(“select \* from employee”);**

1. **DML Operation : stmt.executeUpdate(“DML Operation”): This method return type is int.**
2. **Close the resource properly.**

**bean**

**service : Service class has Dao layer. Depending upon the requirement we can apply business logic before calling dao method or after calling dao method or before/after or no business logic.**

**main**

**dao layer : Data Access Object : writing pure jdbc or orm code. DAO layer not responsible to interact with input or output device or view technologies.**

**bean -🡪 this class always map to table in database.**

**Employee(class) – EMPLOYEE (TABLE)**

**Id ID**

**Name NAME**

**Salary SALARy**

**Properties columns**

**Object record**

**Main(main method) -🡪 interact with view or keyword. This class take value through keyboards set the value to JavaBean class and pass the value to Service layer.**

**PreparedStatement : support parameterized query. The performance wise PreparedStatement is faster than Statement.**

**Main class**

**package** main;

**import** java.util.Scanner;

**import** bean.Employee;

**import** service.EmployeeService;

**public** **class** App {

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

EmployeeService es = **new** EmployeeService();

//es.getAllEmployeeDetails().forEach(e->System.out.println(e));

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

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

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

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

String name= obj.next();

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

**float** salary = obj.nextFloat();

Employee emp= **new** Employee(id, name, salary);

String result = es.storeEmployee(emp);

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

}

}

**Bean class**

**package** bean;

**public** **class** Employee {

**private** **int** id;

**private** String name;

**private** **float** salary;

**public** Employee() {

**super**();

// **TODO** Auto-generated constructor stub

}

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

**super**();

**this**.id = id;

**this**.name = name;

**this**.salary = salary;

}

**public** **int** getId() {

**return** id;

}

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

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **float** getSalary() {

**return** salary;

}

**public** **void** setSalary(**float** salary) {

**this**.salary = salary;

}

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";

}

}

**Service class**

**package** service;

**import** java.util.List;

**import** bean.Employee;

**import** dao.EmployeeDao;

**public** **class** EmployeeService {

EmployeeDao employeeDao = **new** EmployeeDao();

**public** List<Employee> getAllEmployeeDetails() {

// add bonus for all employee base upon desgn or exp

**return** employeeDao.rertieveEmployeeDetails();

}

**public** String storeEmployee(Employee emp) {

**if**(emp.getSalary()<10000) {

**return** "Salary must be > 10000";

}**else** {

**if**(employeeDao.storeEmployeeDetails(emp)>0) {

**return** "Record store successfully";

}**else** {

**return** "Record didn't store";

}

}

}

}

**Dao class**

**package** dao;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**import** java.util.ArrayList;

**import** java.util.List;

**import** bean.Employee;

**public** **class** EmployeeDao {

**public** List<Employee> rertieveEmployeeDetails() {

List<Employee> listOfEmp = **new** ArrayList<>();

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/fisdb", "root", "root@123");

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("select \* from employee");

**while**(rs.next()) {

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

emp.setId(rs.getInt(1));

emp.setName(rs.getString(2));

emp.setSalary(rs.getFloat(3)); // converting each record to java object.

listOfEmp.add(emp); // add the object in list

}

}**catch**(Exception e) {

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

}

**return** listOfEmp;

}

**public** **int** storeEmployeeDetails(Employee emp) {

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/fisdb", "root", "root@123");

PreparedStatement pstmt = con.prepareStatement("insert into employee values(?,?,?)");

pstmt.setInt(1, emp.getId());

pstmt.setString(2, emp.getName());

pstmt.setFloat(3, emp.getSalary());

**return** pstmt.executeUpdate(); // return int if insert return 1

}**catch**(Exception e) {

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

}

**return** 0;

}

}