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

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

**C :**

**void mno() {**

**}**

**void xyz() {**

**mno();**

**}**

**void abc() {**

**xyz();**

**}**

**int a=10;**

**void main() {**

**abc();**

**}**

**OOPs :**

**Java:**

**Reflection API :**

**object : any real world entiy**

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

**Person**

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

**Bank**

**Animal**

**Car**

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

**C + OOPs = C++**

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

**C**

**C++**

**Java .net**

**Bean**

**Oak**

**Nov 1995**

**James Gosling It was belong Sun Microsystem**

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

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

**+**

**Features**

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

**etc.**

**Java 8 :**

**Syntax class**

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

**fields;**

**methods;**

**}**

**Demo.java**

**class Demo {**

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

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

**}**

**}**

**To compile the program**

**javac Demo.java**

**To run the program**

**java Demo**

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

**2 types**

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

**class Demo {**

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

**int a=10;**

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

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

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

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

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

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

**}**

**}**

**Type casting: Converting one data type format to another data type format is known as type casting.**

1. **Implicit**
2. **Explicit**

**int family**

**-🡪Implicit -🡪**

**byte short int long**

**🡨-Explicit ---**

**class Demo {**

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

**byte a = 10;**

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

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

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

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

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

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

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

**}**

**}**

**int to float family**

**implicit -🡪**

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

**🡨------explicit**

**class Demo {**

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

**int a=10;**

**float b =a;**

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

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

**//float c = (float)10.10;**

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

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

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

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

**}**

**}**

**Reference data types**

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

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

**Syntax**

**dataType arrayName[];**

**int abc[];**

**int []abc;**

**int [] abc;**

**int[] abc;**

**Valid array declaration**

**If statements**

**Switch statement**

**Looping**

**While loop**

**Do while lop**

**For loop**

**For each loop or enhanced loop**

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

**}**

**class Demo {**

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

**int []abc={10,20,30,40,50};**

**System.out.println(abc[0]);**

**System.out.println(abc[4]);**

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

**for(int i=2;i<abc.length;i++){**

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

**}**

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

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

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

**}**

**}**

**}**

**Creating the memory size for the array**

**Syntax**

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

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

**int family –0**

**float family -0.0**

**char – space**

**boolean – false**

**String - null**

**Taking the value through keyboards in Java**

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

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

**Syntax Scanner class object**

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

**Scanner class part of util package.**

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

**nextByte()**

**nextShort()**

**nextInt()**

**nextFloat()**

**nextDouble()**

**nextBoolean()**

**no method as nextChar()**

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

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

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

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

**import java.util.Scanner;**

**class Demo {**

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

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

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

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

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

**}**

**}**

**import java.util.Scanner;**

**class Demo {**

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

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

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

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

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

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

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

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

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

**}**

**}**

**object :any real world entity**

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

**class Car {**

**String color;**

**float price;**

**int wheel;**

**void start() {**

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

**}**

**void appliedGear() {**

**}**

**void moving() {**

**}**

**void stop() {**

**}**

**}**

**class CarTest {**

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

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

**//start();**

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

**santro.start();**

**}**

**}**

**class Car {**

**String color;**

**float price;**

**int wheel;**

**void start() {**

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

**}**

**void appliedGear() {**

**}**

**void moving() {**

**}**

**void stop() {**

**}**

**}**

**class CarTest {**

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

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

**//start();**

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

**santro.start();**

**}**

**}**

**Types of variables or fields**

**3 types**

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

**Instance variable**

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

**Local variable**

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

**class Car {**

**String color;**

**float price;**

**int wheel;**

**void start() {**

**int temp=123;**

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

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

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

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

**}**

**void stop() {**

**String msg="Welcome";**

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

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

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

**}**

**}**

**class CarTest {**

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

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

**//start();**

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

**santro.start();**

**santro.stop();**

**}**

**}**

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

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

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

**import java.util.Scanner;**

**class Employee {**

**int id;**

**String name;**

**float salary;**

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

**void read() {**

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

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

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

**name = obj.next();**

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

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

**}**

**void calSalary() {**

**float hra,da,pf;**

**hra = salary\*0.10f;**

**da = salary\*0.05f;**

**pf = salary\*0.07f;**

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

**}**

**void display() {**

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

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

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

**}**

**}**

**class EmployeeTest {**

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

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

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

**emp1.read();**

**emp2.read();**

**emp1.calSalary();**

**emp1.display();**

**emp2.display();**

**}**

**}**

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

**Pts**

1. **Constructor have same name as class itself.**
2. **Constructor doesn’t contains return type not even void also.**
3. **Constructor no need to call it will call automatically when we create the object of that class.**

**import java.util.Scanner;**

**class Employee {**

**Employee() {**

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

**}**

**void display() {**

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

**}**

**}**

**class EmployeeTest {**

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

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

**//emp1.Employee();**

**emp1.display();**

**}**

**}**

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

**Parameterized constructor with this keywords.**

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

**this.instanceVariable**

**import java.util.Scanner;**

**class Cal {**

**int a,b;**

**Cal() {**

**a=10;**

**b=20;**

**}**

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

**this.a=a;**

**this.b=b;**

**}**

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

**this.a = a;**

**this.b = b;**

**}**

**void display() {**

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

**}**

**}**

**class EmployeeTest {**

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

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

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

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

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

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

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

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

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

**c5.display();**

**}**

**}**

**Encapsulation:**

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

**Ex :class : properties**

**Behavior**

**import java.util.Scanner;**

**class Employee {**

**String name;**

**float salary;**

**void display() {**

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

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

**}**

**}**

**class EmployeeTest {**

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

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

**emp1.display();**

**emp1.salary=12000;**

**emp1.display();**

**}**

**}**

**Another Encapsulation example**

**import java.util.Scanner;**

**class Employee {**

**private String name;**

**private float salary;**

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

**this.name = name;**

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

**if(salary<0) {**

**this.salary =8000;**

**}else {**

**this.salary = salary;**

**}**

**}**

**void display() {**

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

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

**}**

**}**

**class EmployeeTest {**

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

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

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

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

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

**emp1.display();**

**}**

**}**

**import java.util.Scanner;**

**class Employee {**

**private String name;**

**private float salary;**

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

**this.name = name;**

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

**if(salary<0) {**

**this.salary =8000;**

**}else {**

**this.salary = salary;**

**}**

**}**

**void display() {**

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

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

**}**

**}**

**class EmployeeTest {**

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

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

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

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

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

**emp1.display();**

**}**

**}**

**JavaBean class :**

**Rules**

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

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

**public class Employee {**

**private int id;**

**private String name;**

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

**this.id = id;**

**}**

**public int getId() {**

**return id;**

**}**

**}**

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

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

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

**class OldClass { super class, base class**

**properties parent class**

**behavior**

**}**

**class NewClass extends OldClass{**

**properties sub class, derived class**

**behaviors child class.**

**}**

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

**class A {**

**void dis1() {**

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

**}**

**}**

**class B extends A{**

**void dis2() {**

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

**}**

**}**

**class InheritanceDemo {**

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

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

**obj1.dis1();**

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

**obj2.dis2();**

**obj2.dis1();**

**}**

**}**

**Types of Inheritance**

1. **Single Inheritance**

**One super class and one sub class**

**class A {}**

**class B extends A {}**

1. **Multilevel inheritance**

**One super class and n number of sub classes connected one by one**

**class A {}**

**class B extends A{}**

**class C extends B{}**

**class D extends C {}**

1. **Hierarchical Inheritance**

**One super class and n number of sub classes directly connected to super class.**

**class A {}**

**class B extends A{}**

**class C extends A{}**

**class D extends A {}**

1. **Multiple Inheritance**

**class A {}**

**class B {}**

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

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

**class A {**

**void dis1() {**

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

**}**

**}**

**class B extends A{**

**void dis2() {**

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

**}**

**}**

**class InheritanceDemo {**

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

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

**obj1.dis1();**

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

**obj2.dis2();**

**obj2.dis1();**

**}**

**}**

**class Employee {**

**id,name,salary**

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

**read(){**

**}**

**calSalary(){**

**hra,da,pf**

**}**

**display() {}**

**}**

**class Programmer extends Employee{**

**projectName;**

**readPgr() {}**

**disPgr() {}**

**}**

**class Manager extends Employee {**

**numberOfemp;**

**readMgr() {}**

**disMgr() {}**

**}**

**class EmployeeTest {**

**Main method**

**/\***

**\*/**

**Programmer**

**Manager**

**}**

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

**Two types**

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

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

**class Operation {**

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

**System.out.println(x+y);**

**}**

**void add(int x, int y, int z) {**

**System.out.println(x+y+z);**

**}**

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

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

**}**

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

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

**}**

**}**

**class InheritanceDemo {**

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

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

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

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

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

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

**}**

**}**

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

**To achieve method overriding the classes must be use inheritance features.**

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

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

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

**class Bike {**

**int a=10;**

**void speed() {**

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

**}**

**}**

**class HeroHonda extends Bike {**

**int a=20;**

**void color() {**

**int a =30;**

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

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

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

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

**}**

**}**

**class Pulsar extends Bike {**

**void color() {**

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

**}**

**@Override**

**void speed() {**

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

**}**

**}**

**class Tvs extends Bike {**

**void color() {**

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

**}**

**@Override**

**void speed() {**

**super.speed();**

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

**}**

**}**

**class InheritanceDemo {**

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

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

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

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

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

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

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

**}**

**}**

**abstract :**

**abstract is a keyword we can use with method and class but not with variable.**

1. **Abstract method : The method without body or without curly braces or incomplete method is known as abstract method.**

**abstract returnType methodName(parameterList);**

**abstract void speed();**

1. **Abstract class :**

**Syntax**

**abstract class className {**

**}**

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

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

**abstract class Bike {**

**abstract void speed();**

**Bike(){}**

**}**

**abstract class HeroHonda extends Bike {**

**void color() {**

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

**}**

**abstract void mailage();**

**}**

**class SuperHeroHonda extends HeroHonda {**

**void speed(){**

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

**}**

**void mailage() {**

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

**}**

**}**

**class Pulsar extends Bike{**

**void speed() {**

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

**}**

**}**

**class InheritanceDemo {**

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

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

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

**}**

**}**

**Final keyword**

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

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

**final int A=10;**

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

**final void speed() {**

**}**

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

**final class Demo {**

**}**

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

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

**class Outer {**

**static int a;**

**static void dis() {}**

**static class Inner {**

**}**

**}**

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

**class Abc {**

**int a;**

**static int b;**

**void dis1() {**

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

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

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

**}**

**static void dis2() {**

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

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

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

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

**}**

**}**

**class InheritanceDemo {**

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

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

**obj1.a=10;**

**Abc.b=20;**

**obj1.b=30;**

**obj1.dis1();**

**Abc.dis2();**

**obj1.dis2();**

**}**

**}**

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

**Syntax**

**interface interfaceName {**

**fields;**

**methods;**

**}**

**interface Abc {**

**int A=10;**

**void dis1();**

**}**

**interface Xyz {**

**int B=20;**

**void dis2();**

**}**

**interface Mno extends Abc, Xyz{**

**int C=30;**

**void dis3();**

**}**

**class A {}**

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

**}**

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

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

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

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

**Super class/interface sub**

**public public**

**protected public**

**protected**

**default public**

**protected**

**default**

**private We can’t**

**override**

**interface Abc {**

**int A=10;**

**void dis1();**

**}**

**interface Xyz {**

**int B=20;**

**void dis2();**

**}**

**interface Mno extends Abc, Xyz{**

**int C=30;**

**void dis3();**

**}**

**class A {}**

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

**public void dis1() {**

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

**}**

**public void dis2(){**

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

**}**

**}**

**class InheritanceDemo {**

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

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

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

**}**

**}**

**Difference between abstract class and interface.**

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

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

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

**class A {**

**void dis1() {**

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

**}**

**}**

**class B extends A {**

**void dis1() {**

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

**}**

**void dis2() {**

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

**}**

**}**

**class InheritanceDemo {**

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

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

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

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

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

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

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

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

**}**

**}**

**2nd Example**

**abstract class A {**

**abstract void dis1();**

**}**

**class B extends A {**

**void dis1() {**

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

**}**

**void dis2() {**

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

**}**

**}**

**class InheritanceDemo {**

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

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

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

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

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

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

**}**

**}**

**3rd Example**

**interface A {**

**void dis1();**

**}**

**class B implements A {**

**public void dis1() {**

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

**}**

**void dis2() {**

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

**}**

**}**

**class InheritanceDemo {**

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

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

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

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

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

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

**}**

**}**

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

**interface :**

**interface A {**

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

**}**

**interface B {**

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

**}**

**class Server implements A,B{**

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

**return x+y;**

**}**

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

**return x-y;**

**}**

**public void ownMethod() {**

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

**}**

**}**

**class InheritanceDemo {**

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

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

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

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

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

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

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

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

**}**

**}**

**this() and super()**

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

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

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

**class Employee {**

**int id;**

**String name;**

**float salary;**

**Employee() {**

**//this(10);**

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

**}**

**Employee(int id) {**

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

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

**}**

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

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

**}**

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

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

**}**

**}**

**class Manager extends Employee {**

**Manager() {**

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

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

**}**

**}**

**class InheritanceDemo {**

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

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

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

**}**

**}**

**Dy 3 : 2/7/2020**

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

**2 types**

1. **User-defined package**
2. **Pre-defined or built-in package**

**Education**

**school college**

**Attendance Attendance**

**Date : util**

**Date : sql**

**Syntax**

**package com;**

**class Demo {**

**}**

**Eclipse for JEE**

**IDE :**

**Access specifiers**

1. **public : instance variable, static variable, non static and static method, constructor and class but not with local variable. (In one file or editor if we are writing more than one class only one class we can use public).**

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

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

**scope :only same package or within a same package.**

1. **Protected : same as public not with class.**

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

1. **private : instance variable, static variable, non static and static method, constructor but not with class and local variable.**

**Scope : within a same class.**

**Pre-defined packages**

**Exception Handling**

**java javax-🡪 root packages**

**lang: language swing**

**io :inputoutpu net**

**util: utility rmi**

**awt:abstract window toolkit sql**

**net :networking servlet**

**rmi : remote method**

**invocation ejb Enterprise**

**Java Bean**

**sql: jms JEE**

**structure query language**

**Java Messaging Service**

**etc etc**

**lang, io, util (collection framework) and sql**

**Servlet and JSP**

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

**By default every java program extends Object class.**

**String**

**System**

**Super API (Application Programming Interface).**

**Exception Handling :**

**javap java.packageName.className/interfaceName**

**Exception is a object or memory when we do unexpected or abnormal condition occurs during the execution of program.**

**Web Service :**

**JNI :Java Native Interface**

**Java**

**Compile Run**

**javac java**

**Compile time run time**

**error error**

**syntax**

**error or**

**typo error**

**run time error**

**Error Exception 🡪pre-defined classes**

**Error :the error which generate at the time which we can’t handle it ex :Jvm crash, out of memory etc.**

**Exception : It is a type of error which we can handle it ex :ArthmeticException /0**

**Exception**

**Checked exception Unchecked exception**

**IOException RuntimeException**

**SQLException Pre-defined class**

**ArithmeticException**

**ArrayIndexOutOfBounds**

**NumberFormat**

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

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

**try {**

**}catch(Exception e){**

**}**

**class ExpDemo{**

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

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

**int a=10;**

**int b =1;**

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

**try{**

**int res = a/b; // new ArithmeticException();**

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

**res = 10/abc[5];**

**}**

**catch(ArithmeticException e){**

**System.out.println("I Take Care!");**

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

**}catch(ArrayIndexOutOfBoundsException e) {**

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

**}catch(Exception e) {**

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

**}**

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

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

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

**}**

**}**

**finally :It is type of block which will execute compulsory if exception generate or not.**

**try**

**catch catch catch catch finally**

**catch finally catch**

**finally**

**class ExpDemo{**

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

**try {**

**int res = 10/1;**

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

**System.out.println("No Exception");**

**}catch(Exception e){**

**System.out.println("Catch block");**

**}finally {**

**System.out.println("finally block");**

**}**

**System.out.println("Normal Statement");**

**}**

**}**

**class ExpDemo{**

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

**try {**

**int res = 10/0;**

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

**System.out.println("No Exception");**

**}finally {**

**System.out.println("finally block");**

**}**

**System.out.println("Normal Statement");**

**}**

**}**

**class ExpDemo{**

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

**try {**

**System.exit(0);**

**int res = 10/0;**

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

**System.out.println("No Exception");**

**}finally {**

**System.out.println("finally block");**

**}**

**System.out.println("Normal Statement");**

**}**

**}**

**class ExpDemo{**

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

**try {**

**System.exit(0);**

**int res = 10/0;**

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

**System.out.println("No Exception");**

**}finally {**

**System.out.println("finally block");**

**}**

**System.out.println("Normal Statement");**

**}**

**}**

**throw : this keyword is use to throw the exception base upon the conditions or raise the exception.**

**Syntax**

**throw new Exception();**

**or**

**throw new ExceptionSubClass();**

**class MyException extends Exception {**

**MyException() {}**

**MyException(String str) {**

**super(str);**

**}**

**}**

**class ExpDemo{**

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

**try {**

**int a=10;**

**int b=5;**

**if(a>b) {**

**//int res =10/0;**

**//throw new Exception();**

**//throw new ArithmeticException();**

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

**//throw new MyException();**

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

**}**

**}catch(Exception e ) {**

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

**}**

**}**

**}**

**throws :**

**throws keyword is use to throw the exception to caller method.**

**Syntax**

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

**}**

**class ExpDemo{**

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

**//try{**

**int a=10/0;**

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

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

**}**

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

**//try{**

**dis1();**

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

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

**}**

**}**

**class ExpDemo{**

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

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

**//try{**

**Thread.sleep(2000);**

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

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

**}**

**}**

**class Employee {**

**int id;**

**String name;**

**Employee() {**

**id=123;**

**name="Abc"; // from file, from database with condition**

**}**

**Employee(int id) {**

**this();**

**this.id=id;**

**//name = "Abc";**

**}**

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

**this(id);**

**//this.id = id;**

**this.name = name;**

**}**

**void dis() {**

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

**}**

**}**

**class ThisDemo {**

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

**Employee e1 = new Employee(); e1.dis();**

**Employee e2 = new Employee(1); e2.dis();**

**Employee e3 = new Employee(2,"Ravi"); e3.dis();**

**}**

**}**

**Multithreading :**

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

**Process : time taken to execute the code.**

**Processor : it help to execute the code.**

**Thread : Small execution of a code with a process or thread is light weighted process.**

**public static Thread currentThread() {**

**//coding**

**return t;**

**}**

**class ThreadDemo {**

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

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

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

**/\***

**main -- name of the thread**

**5 --priority of the thread**

**main -- group of the thread**

**\*/**

**t.setName("Demo thread");**

**t.setPriority(1); //min 1, max 10, norm 5**

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

**t.setPriority(Thread.MAX\_PRIORITY);**

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

**String name = t.getName();**

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

**}**

**}**

**Multitasking :**

**Process based**

**Thread based**

**2 ways**

1. **extends Thread class**
2. **implements Runnable interface**

**Using extends Thread class**

**class A extends Thread{**

**@Override**

**public void run() { //running state**

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

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

**}**

**} //i = 10destroy**

**}**

**class B extends Thread{**

**@Override**

**public void run() {**

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

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

**}**

**}**

**}**

**class ThreadDemo {**

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

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

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

**obj1.start(); // Runnable state --> ready to run**

**obj2.start();**

**}**

**}**

**Using implements Runnable interface**

**class A implements Runnable{**

**@Override**

**public void run() { //running state**

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

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

**}**

**} //i = 10destroy**

**}**

**class B implements Runnable{**

**@Override**

**public void run() {**

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

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

**}**

**}**

**}**

**class ThreadDemo {**

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

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

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

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

**Thread t2 = new Thread(obj2); // Thread class reference**

**t1.start(); // Runnable state --> ready to run**

**t2.start();**

**}**

**}**

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

**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);**

**}**

**}**

**}**

**class ThreadDemo {**

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

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

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

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

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

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

**Thread t3 = new Thread(obj1);**

**t1.setName("1st Thread");**

**t2.setName("2nd Thread");**

**t3.setName("3rd Thread");**

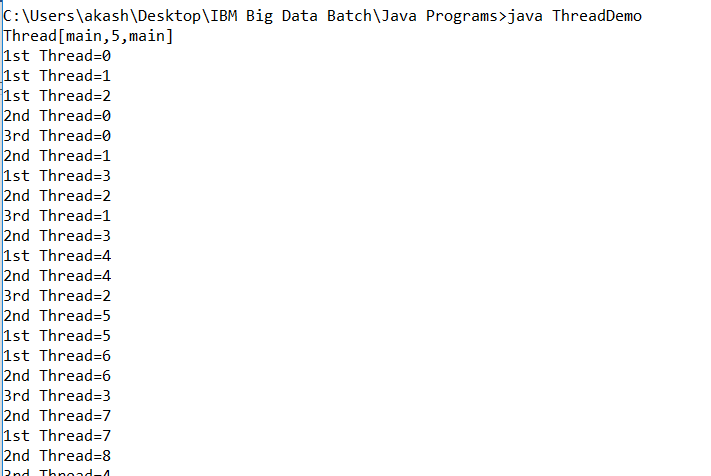
**t1.start();**

**t2.start();**

**t3.start();**

**}**

**}**

****

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

**synchronized keyword we can use with method or within a method we can use block**

**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+" sorry no ticket");**

**}**

**}**

**}**

**class ThreadDemo {**

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

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

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

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

**Thread t3 = new Thread(obj1);**

**t1.setName("Raj");**

**t2.setName("Seeta");**

**t3.setName("Reeta");**

**t1.start();**

**t2.start();**

**t3.start();**

**}**

**}**

**Day 4: 2/10/2020**

**IO package**

**Input and Output**

**Stream: flow of data or it is an abstraction between source and destination.**

**Stream**

**byte char**

**Input Output Input Output**

**InputStream Reader Writer**

**OutputStream**

**All 4 types of abstract classes**

**DataInputStream InputStreamReader**

**FileInputStream FileReader**

**BufferedInputStream BufferedReader**

**ObjectInputStream**

**DataOutputStream OutputStreamWriter**

**FileOutputStream FileWriter**

**BufferedOutputStream BufferedWriter**

**ObjectOutputStream PrintWriter**

**PrintStream**

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

**InputStream obj = System.in;**

**PrintStream ps = System.out;**

**ps.println(“Welcome to Jav”);**

**import java.io.\*;**

**class IoDemo {**

**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);\*/**

**/\***

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

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

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

**int ch;**

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

**fos.write(ch);**

**}**

**fos.close();**

**\*/**

**FileInputStream fis = new FileInputStream("abc.txt");**

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

**int ch;**

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

**fos.write(ch);**

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

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

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

**}**

**fis.close();**

**fos.close();**

**}**

**}**

**import java.io.\*;**

**class IoDemo {**

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

**FileInputStream fis = new FileInputStream("abc.txt");**

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

**FileOutputStream fos = new FileOutputStream("xyz1.txt",true);**

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

**int ch;**

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

**if(ch=='o') {**

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

**bis.skip(4);**

**}else {**

**bos.write(ch);**

**}**

**}**

**bos.flush();**

**fis.close();**

**fos.close();**

**}**

**}**

**Object Serialization: Storing the object state in external file or converting object into byte format is known as Object Serialization.**

**Object**

**State**

**Behavior**

**Identity**

**Class must be implements Serializable interface**

**import java.io.\*;**

**class Employee implements Serializable{**

**int id;**

**String name;**

**float salary;**

**Employee() {}**

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

**this.id = id;**

**this.name = name;**

**this.salary = salary;**

**}**

**void display() {**

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

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

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

**}**

**}**

**class IoDemo {**

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

**Employee emp = new Employee(1,"Ravi",12000);**

**emp.display();**

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

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

**oos.writeObject(emp);**

**System.out.println("Serializable done successfully....");**

**FileInputStream fis = new FileInputStream("emp.txt");**

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

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

**Employee e = (Employee)obj;**

**e.display();**

**}**

**}**

**Collection Framework:**

**Collection framework contains set of collection of classes and interfaces which help to store the collection of object or elements of any type as well as provided lot of pre-defined method which help to store, retrieve, search etc.**

**int a;**

**int abc[];**

**class Employee {**

**int id;**

**String name;**

**float salary;**

**}**

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

**Array object**

**Employee e[]=new Employee[100];**

**e[0]=new Employee();**

**e[0].id=100;**

**e[1]=new Employee();**

**Collection --🡪 interface**

**extends extends but map doesn’t extends**

**Set List Queue Map 🡪interfaces**

**Doesn’t Maintain FIFO key-value pairs**

**Allow Oder where key is**

**Duplicate Allow unique**

**Order, Duplicate and value**

**UnOder may be**

**Sorted duplicate**

**HashSet Stack PriorityQueue**

**LinkedHashSet ArrayList**

**TreeSet LinkedList**

**Vector**

**HashMap**

**LinkedHashMap**

**TreeMap**

**Hashtable**

**HashSet**

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

**class A{}**

**class B{}**

**class CollectionDemo {**

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

**HashSet hs = new HashSet();**

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

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

**int a=10;**

**Integer b = new Integer(a); //converting primitive to object**

**hs.add(b);**

**hs.add(20); //auto-boxing : converting primtive to object**

**hs.add(10.20);**

**hs.add("Ravi");**

**hs.add(new A());**

**hs.add(new B());**

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

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

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

**hs.remove(20);**

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

**System.out.println(hs.contains("Ravi"));**

**System.out.println(hs.contains("Raj"));**

**}**

**}**

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

**class A{}**

**class B{}**

**class CollectionDemo {**

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

**TreeSet hs = new TreeSet();**

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

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

**int a=10;**

**Integer b = new Integer(a); //converting primitive to object**

**hs.add(b);**

**hs.add(20); //auto-boxing : converting primtive to object**

**hs.add(50);**

**hs.add(30);**

**hs.add(60);**

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

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

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

**hs.remove(20);**

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

**}**

**}**

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

**class A{}**

**class B{}**

**class CollectionDemo {**

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

**Stack ss = new Stack();**

**ss.push(100);**

**ss.push(200);**

**ss.push(300);**

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

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

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

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

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

**}**

**}**

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

**class A{}**

**class B{}**

**class CollectionDemo {**

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

**ArrayList al = new ArrayList();**

**al.add(10);**

**al.add(20);**

**al.add(30);**

**al.add(40);**

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

**System.out.println(al.get(1));**

**al.add(1,100);**

**System.out.println(al.get(1));**

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

**al.remove(1);**

**al.remove(new Integer(40));**

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

**}**

**}**

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

**class A{}**

**class B{}**

**class CollectionDemo {**

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

**LinkedList al = new LinkedList();**

**al.add(10);**

**al.add(20);**

**al.add(30);**

**al.add(40);**

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

**System.out.println(al.get(1));**

**al.add(1,100);**

**System.out.println(al.get(1));**

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

**al.remove(1);**

**al.remove(new Integer(40));**

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

**}**

**}**

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

**class A{}**

**class B{}**

**class CollectionDemo {**

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

**PriorityQueue pq = new PriorityQueue();**

**pq.add(3);**

**pq.add(1);**

**pq.add(4);**

**pq.add(2);**

**pq.add(5);**

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

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

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

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

**}**

**}**

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

**class A{}**

**class B{}**

**class CollectionDemo {**

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

**HashMap hm = new HashMap();**

**hm.put(2,"Ravi");**

**hm.put(1,"Ajay");**

**hm.put(5,"Dinesh");**

**hm.put(6,"Vishal");**

**hm.put(2,"Raj");**

**hm.put(3,"Vishal");**

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

**System.out.println(hm.get(2));**

**}**

**}**

**Retrieve the elements from collection classes one by one**

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

**class A{}**

**class B{}**

**class CollectionDemo {**

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

**Set ss = new HashSet();**

**ss.add(10);**

**ss.add(20);**

**ss.add(30);**

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

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

**for(Object obj:ss) {**

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

**}**

**System.out.println("Using Iterator interface");**

**Iterator ii = ss.iterator();**

**while(ii.hasNext()) {**

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

**}**

**}**

**}**

**Collection Framework with Generics**

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

**class CollectionDemo {**

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

**/\*List ll = new ArrayList();**

**ll.add(10); // auto-boxing : primitive to object**

**ll.add(10.10f);**

**ll.add("Ravi");**

**ll.add(20);**

**Object obj = ll.get(0);**

**Integer i = (Integer)obj;**

**int n = i.intValue(); //primitive value**

**System.out.println(n);\*/**

**//List<Type> ll = new ArrayList<Type>();**

**List<Integer> ll = new ArrayList<Integer>();**

**ll.add(10);**

**ll.add(20);**

**ll.add(30);**

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

**int n = ll.get(0); //auto-unboxing object to primitive**

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

**}**

**}**

**UML : Unified modeling language :**

**C/C++ :**

**Flow chart :**

**OOPs Relationship**

1. **Manager Is a Employee**
2. **Has a**

**class Employee {**

**id**

**name**

**salary**

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

**}**

**class Manager extends Employee{**

**}**

**class Address {**

**}**

**Has a**

1. **Association**
2. **Aggregation**
3. **Composition**

**class A {**

**B obj1 = new B(); 0, 1, infinity**

**}**

**class B {**

**A obj2 = new A();0, 1, infinity**

**}**

**class Employee {**

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

**}**

**class Address {**

**}**

**Class Student {**

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

**}**

**Class StudentHistory {**

**}**

**Day 5 : 2/11/2020**

**Connect**

**Username**

**Password**

**create table employee(id number(10) primary key, name varchar(10),salary number(10,2));**

**desc employee**

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

**JDBC : Java Database Connectivity :**

**JDBC is a API (Application Programming Interfaces) which provided lot of pre-defined classes and interfaces which help store and retrieve records from any database like Oracle, MySQL, db2, sql server etc.**

**Steps to connect the database through JDBC**

1. **Import the package ie sql package**

**java.sql.\*;**

**javax.sql.\*;**

1. **Write a method may be main method or user-defined with exception handling concept ie try-catch or throws.**
2. **Load the Driver :Driver is pre-defined class provide by vendor whose database going to connect through Java technologies.**

**Types of Driver**

**4 types**

1. **Type 1 : jdbc odbc bridge driver (java 8 type 1 driver doesn’t support).**
2. **Type 2 : jdbc native api driver**
3. **Type 3 : jdbc net protocol driver**
4. **Type 4 : jdbc pure or thin driver**

**External jar is required for the Type 4 Driver.**

**Type 4 driver Name for Oracle database.**

**oracle.jdbc.driver.OracleDriver**

**Class.forName(“DriverName”);**

**Build tool name**

**Maven or gradle**

1. **Establish the connection**

**DriverManager is a pre-defined class which contain pre-defined static method ie getConnection(url,username,password);**

**The getConnection() method return type is Connection interface reference.**

**Connection con = DriverManger.getConnection(url,username,password);**

1. **Create Statement :**

**Types of Statement**

* 1. **Statement**
  2. **PreparedStatement**
  3. **CallableStatement**

**All three are interfaces provided set of method which help to do operation of specific tables.**

1. **Statement stmt = con.createStatement();**
2. **A. Retrieve**

**ResultSet rs = stmt.executeQuery(“select clause”);**

**B. Insert /Delete /Update**

**int res =stmt.executeUpdate(“insert/delete/update query”);**

**if(res>0) {**

**}else {**

**}**

**Statement : Whenever we execute any query using Statement. The query will compile on java side, send to database, then execute on database side and give the acknowledgement to Java side my be success or failure.**

**PreparedStatement :query compile only once on java side and execute n number of times on database.**

**PreparedStatement support parameterized query concept.**

**Java**

**J2SE J2EE J2ME**

**JavaSE JavaEE JavaME**

**JSE JEE JME**

**Java Java Java**

**Standard Enterprise Micro**

**Edition Edition Edition**

**Basic**

**Java**

**OOPS**

**Exception**

**Handling**

**Multithreading**

**Packages**

**Io, lang,**

**Util, awt**

**Swing**

**Net, rmi**

**Sql etc**

**AWT/Swing -🡪GUI**

**JDBC**

**Oracle/MySQL**

**https://**[**www.google.com**](http://www.google.com) **-🡪URL**

**req(http/https)---🡪**

**Client Server**

**🡨---res(http/https)-----**

**HTML/HTML5**

**CSS/CSS3**

**JavaScript**

**Client side**

**Validation using**

**JavaScript or HTMl5 features**

**JEE**

**Servlet/JSP/EJB**

**Java Server Pages**

**Enterprise Java**

**Bean**

**Asp.net**

**Php**

**Python**

**Server**

**Types of Server :**

**Container : It is also known as engine. Container is responsible to execute Servlet, JSP and EJB Application. Load the class, creating the objects, calling life cycle methods and destroy the objects.**

1. **Web server : Tomcat (web container) : Servlet and JSP**
2. **Application Server :Glashfish, WebLogic, JBoss, Web Sphere**

**Web Container and EJB Container**

**(Servlet, JSP and EJB)**

**2 types Web Container and EJB Container**

**Servlet : Servlet is normal java program which help to create dynamic web page on server side.**

**import javax.servlet.\*;**

**servlet : servlet is a package which contains set of classes and interface.**

**Servlet: Servlet is interface which contains set of methods.**

**5 methods**

**init**

**service**

**destroy life cycle methods**

**getServletInfo**

**getServletConfig**

**GenericServlet : It is type of abstract class which internally implements Servlet interface and provided the body for four method except service methods.**

**HttpServlet :it is type of abstract class which internally extends GenericServlet and provide the body for service method as well as provided some extra method in the form of doXXX like doGet() and doPost()**

**RequestDispatcher :**

**Day 6 : 2/12/2020**

**JSP : Java Server Page : JSP is a tag based scripting language which help to create dynamic web on server side.**

**Limitation of Servlet.**

1. **Servlet is a normal java program if we do any change in servlet program we have to re-compile and re-deploy the application.**
2. **If we want write any HTML code inside a servlet it must be in pw.println(“<h1>Welcome Servlet</h1>”)**

**Ie the presentation logic is embedded inside a servlet.**

1. **Servlet is complex. If we want to display simple message through servlet we have to extends or implements type of Servlet.**

**Then we have to override method like doGet() and doPost() Then we have to create the PrintWriter class object and provide the servlet details in the dd file.**

**JSP Tags**

1. **Scripting tags** 
   1. **Scriptlet tag**

**<%**

**Java coding**

**%>**

* 1. **Declarative tag**

**<%! variable declaration %>**

* 1. **Expression tag**

**<%= %>**

1. **Implicit object** 
   1. **out it is equal to PrintWriter class reference.**
   2. **request It is equal to HttpServletRequest interface reference.**
   3. **response : It is equal to HttpServletResponse interface reference.**
2. **Directive tags**
   1. **Page directive tags**
3. **Actions tags** 
   1. **Forward**
   2. **Include**

**Limitation of a JSP**

* + 1. **JSP is not secure.**
    2. **In JSP we can’t do re-usability of any logic or coding.**
    3. **Performance wise JSP is slower than Servlet. Once you run jsp program internally it convert back to Servlet ie is page translation phase.**

**Day 7 : 2/13/2020**

**Design Pattern : Best Practice or solation for the repeating for the OOPs Application.**

**GOF: 27 Standard Design Pattern.**

**Creational Design Pattern.**

**Structural Design Pattern**

**Behavioral Design Pattern**

**public ArrayList getInfo() {**

**ArrayList al = new ArrayList();**

**return al;**

**}**

**public List getInfo() {**

**}**

**Simple Factory method Design Pattern :**

**The method is responsible to create the object.**

**Web Application :**

**Online Shopping**

**Login -🡪 Table**

**Customer /Seller**

**MVC :Model View Controller**

**Model**

**DAO : Data Access Object (Here JDBC Code)**

**Resource layer : JDBC Resource, file resources**

**MVC**

**Model Layer -🡪Normal Java classes**

**Controller -🡪 Servlet**

**View 🡪HTML/JSP/CSS/JavaScript**

**Model Layer**

**Resource layer 🡪**

**DAO : Data Access Object 🡪**

**Login**

**Customer / Staff Login**

**Customer 🡪**

**Username PK**

**password**

**Staff -🡪**

**username PK**

**password**

**Product 🡪**

**Pid PK**

**Pname**

**Price**

**bean -🡪**

**Customer**

**Staff**

**Product**

**Testing : Testing is use to find the error or defect or bugs.**

**Read a, b**

**Compute sum =a+b**

**Write a**

**2 types**

**White box**

**Input Process Output**

**Black box**

**Input Output**

**Unit Testing : it may be function, module, class, method, etc. Unit testing is a type of white box testing.**

**jUnit : it small open source framework which provided set of classes and interfaces which help to do unit testing for Java classes.**

**junit4-4.8.2**

**hamcrest-core-1.3**

**maven repository**

**Day 8 : 2/14/2020**

**Service Layer : Service layer contains pure business logic of the application.**

**These layer also normal java classes.**

**Model -🡪 resource, dao, bean, service**

**Controller -🡪 Servlet**

**View --🡪 jsp /html**

**index.jsp**

**hyperlink tags**

**customer login**

**staff login**

**customer.jsp**

**Login Page**

**UserName**

**Password**

**Sign In**

**SingUp Page 🡪hyperlink**

**Customercreate.jsp**

**username**

**password**

**create Account**

**CustomerDashboard (Servlet)**

**staff.jsp**

**Login page**

**Class A{**

**int a;**

**}**

**Class B {**

**A obj = new A();**

**Obj.a**

**}**

**Request scope**

**Session scope**

**Day 8: 2/17/2020**

**CSV :**

**Test Case : Test Case class contains more than method with @Test Annotation which help to do the testing for one or more business methods.**

**Test Suite : Test suite contains more than one Test Case classes.**

**Inner classes**

**class within another class is known as Inner class**

**Type of inner classes**

1. **Non static inner class**
2. **Static inner class**
3. **Local inner class (inside a method)**
4. **Anonymous inner class**

**class Outer {**

**int a;**

**void dis1() {**

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

**Inner in = new Inner();**

**in.dis2();**

**}**

**class Inner {**

**int b;**

**void dis2() {**

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

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

**}**

**}**

**static class StaticInner {**

**int c;**

**void dis3() {**

**System.out.println("Static inner class dis3 method");**

**}**

**}**

**}**

**class InnerClassDemo {**

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

**Outer out = new Outer();**

**out.dis1();**

**//Outer.Inner in = new Outer().new Inner();**

**Outer.Inner in = out.new Inner();**

**in.dis2();**

**Outer.StaticInner in1 =new Outer.StaticInner();**

**in1.dis3();**

**}**

**}**

**class Outer {**

**void dis2() {**

**class LocalInner {**

**void dis1() {**

**System.out.println("Local inner class");**

**}**

**}**

**LocalInner in = new LocalInner();**

**in.dis1();**

**}**

**}**

**class InnerClassDemo {**

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

**Outer out = new Outer();**

**out.dis2();**

**}**

**}**

**interface Abc {**

**void dis1();**

**}**

**class Xyz implements Abc {**

**public void dis1() {**

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

**}**

**}**

**class InnerClassDemo {**

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

**Abc obj = new Xyz();**

**obj.dis1();**

**Abc obj1 = new Abc(){**

**public void dis1() {**

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

**}**

**};**

**obj1.dis1();**

**}**

**}**

**Enum and Enumeration**

**Enumeration is a interface which help to retrieve the value from Vector API.**

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

**class VectorDemo {**

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

**Vector<Integer> vv = new Vector<Integer>();**

**vv.addElement(10);**

**vv.addElement(20);**

**vv.addElement(30);**

**vv.addElement(40);**

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

**Enumeration<Integer> e = vv.elements();**

**while(e.hasMoreElements()) {**

**int n = e.nextElement();**

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

**}**

**}**

**}**

**Enum : Enum is reference data types which help to store constant field or variables.**

**enum Color{**

**RED(10),GREEN(5),BLUE(15);**

**private int val;**

**Color(int val){**

**this.val = val;**

**}**

**public int getValue() {**

**return val;**

**}**

**}**

**class EnumDemo {**

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

**Color cc = Color.RED;**

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

**Color c[]= Color.values();**

**for(Color c1:c) {**

**System.out.println(c1+" "+c1.ordinal()+" "+c1.getValue());**

**}**

**}**

**}**

**VMware Software**

**Oracle Virtual VMBox**