**Java Full Stack Training - onMobile Client**

**Day 1 - Core Java - 04-09-2019**

**C :**

**#include<stdio.h>**

**void mno() {**

**xyz();**

**}**

**void abc() {**

**mno();**

**}**

**void main() {**

**printf("Welcome to C");**

**abc();**

**}**

**OOPs :**

**object : Any real world entity.**

**Ex:**

**properties(State)----> have ---> fields/variables**

**Person**

**behaviour ------>do/does --- function/methods**

**Bank**

**Animal**

**Car**

**etc**

**class : Blue print of object or template of object or collection of object which have same properties and behaviour or use-defined data type which help to describe the objects.**

**syntax**

**class ClassName {**

**properties**

**behaviour**

**}**

**class Person {**

**String name;**

**int age;**

**void sleeping() {**

**}**

**void teaching() {}**

**}**

**Person p1 = new Person();**

**C + OOPs = C++ + PIEA**

**Java .net**

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

**Java 1.0, 1.2, 1.4, 1.5,1.6,1.7 ,**

**1.8, 1.9, 1.10, 1.11, Java 12**

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

**2 types**

**1. Primitive Data types : It is use to store only value**

**2. Non primitive data types ore reference data type: it is use to store value as well as reference of another data type.**

**Primitive data types**

**8 types**

**1. byte 1 byte**

**2. short 2 byte**

**3. int 4 byte**

**4. long 8**

**5. float 4**

**6. double 8**

**7. char 2**

**8. boolean 1 bit**

**class Demo {**

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

**int a=10;**

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

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

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

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

**}**

**}**

**Non Primitive data types**

**1. array**

**2. class (pre-defined or user-defined)**

**3. interface (pre-defined or user-defined)**

**4. enum (pre-defined or user-defined)**

**types casting : converting one data type to another data types is known as type casting.**

**2 types**

**1. Implicit type casting**

**2. explicit type casting**

**----------------------> Implicit ---------------------------------->**

**byte short int long**

**<---------------------Explicit ---------------------<-----------------**

**class Demo {**

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

**byte a = 10; // range of byte value -128 to 127**

**short b = a;**

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

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

**short c = 129;**

**byte d = (byte)c;**

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

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

**}**

**}**

**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;**

**int d = (int)c;**

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

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

**}**

**}**

**Operator**

**if statements**

**switch statement**

**looping**

**while loop**

**do while loop**

**for loop**

**for each loop or enhanced loop**

**array : it is a user-defined data types which is use to store same type of values.**

**syntax**

**dataType arrayName[];**

**int abc[];**

**int []abc2**

**int [] abc3;**

**int[] abc4**

**[]int abc5; InValid**

**int []abc2={10,20,30,40,40,50}**

**for 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=0;i<abc.length;i++) {**

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

**}**

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

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

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

**}**

**}**

**}**

**Creating memory for array in Java**

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

**int []abc = new int[10]; 10\*4 = 40 byte memory**

**int abc[10];**

**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=0;i<abc.length;i++) {**

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

**}**

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

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

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

**}**

**}**

**}**

**Taking value through keyboards in Java.**

**1. Using Scanner class**

**2. Using DataInputStream class**

**3. BufferedReader class**

**4. Command Line Arguments.**

**Scanner : Scanner is a pre-defined class part of util package. Which contains set of methods which help to scan the value through keyboards.**

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

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

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

**byte : obj.nextByte()**

**short : obj.nextShort();**

**int : obj.nextInt();**

**long: obj.nextLong();**

**float : obj.nextFloat();**

**double : obj.nextDouble();**

**boolean : obj.nextBoolean();**

**String str = obj.next(); //only one word**

**String str = obj.nextLine(); more than one word.**

**Create the Scanner class object in main method, create array id,name,salary variables and create the memory size for those variable using scanner class object. then take id,name,salary for more than one employee and find hra, da and pf where HRA is 10%on salary, DA is 5%salary and PF is 7%salary. Then display all employees id,name,salary(grosssSalary).**

**import java.util.Scanner;**

**class Demo {**

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

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

**System.out.println("How many record do you wan to store?");**

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

**int id[]=new int[n];**

**String name[]=new String[n];**

**float salary[]=new float[n];**

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

**//Receive the values**

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

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

**id[i]=obj.nextInt();**

**obj.nextLine(); //use to hold the enter key**

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

**name[i]=obj.nextLine();**

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

**salary[i]=obj.nextFloat();**

**}**

**//Business logic**

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

**float hra, da,pf;**

**hra = salary[i]\*0.10f;**

**da = salary[i]\*0.05f;**

**pf = salary[i]\*0.07f;**

**salary[i]=salary[i]+hra +da -pf;**

**}**

**//Display Records;**

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

**System.out.println("id is"+id[i]+" Name is"+name[i]+" Salary is"+salary[i]);**

**}**

**}**

**}**

**OOPs**

**object and class**

**object : any real world entity**

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

**price, wheel, color etc**

**Car :**

**start(), appliedGear(), moving(), stop()**

**className objectRefeferenceName = new className();**

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

**Access the class non static methods**

**objectReferencename.methodName();**

**santro.start();**

**objectRefernceName.variableName = value;**

**santro.wheel=4;**

**Types of variables or fields**

**In Java variables are divided into 3 types**

**1. Instance variable**

**a. The variable which declare outside a method including main method is known as Instance variable.**

**b. All instance variable hold default values according to their data types.**

**like**

**int family 0**

**float family 0.0**

**char space (white)**

**boolean false**

**String null**

**c. We can access all instance variable inside same class method directly but the method must be non static.**

**2. Local variable**

**a. The variable which declare inside a method including main method is known as local variable.**

**b. The scope of the variable within that block where it declare.**

**c. The local variable doesn't hold default value we have to initialize.**

**class Car {**

**int wheel;**

**String color;**

**float price;**

**void start() {**

**int temp=0;**

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

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

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

**}**

**void stop() {**

**String msg="Welcome";**

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

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

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

**}**

**}**

**class CarTest {**

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

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

**santro.start(); santro.stop();**

**}**

**}**

**Create two classes Employee and EmployeeTest**

**Where Employee class must be contains three instance variables id,name,salary and Scanner class objects.**

**read(), calSalary() and display() non static methods.**

**EmployeeTest class must be contains main methods then create the Employee class object and call read() which help to receive the value through keyboards. calSalary() method which help to do calculation on salary.**

**hra, da and pf (local variables).**

**hra = 10%on Salary , da = 5%on salary and pf = 7%onsalary**

**display() method to display id,name,salaray(GrosssSalary)**

**First Employee object all three methods.**

**Second Employee object only two methods (read and display)**

**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+" name is "+name+" salary is "+salary);**

**}**

**}**

**class EmployeeTest {**

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

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

**emp1.read(); emp1.calSalary(); emp1.display();**

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

**emp2.read();**

**//emp2.calSalary();**

**emp2.display();**

**}**

**}**

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

**pts**

**1. Constructor must be same name as the class itself.**

**2. Constructor doesn't contains return type not event void also.**

**3. Constructor no need to call it will call automatically when we create the object of that class.**

**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.display();**

**emp1.Employee();**

**}**

**}**

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

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

**class Employee {**

**int id;**

**Employee() {**

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

**}**

**void display() {**

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

**}**

**}**

**class EmployeeTest {**

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

**//int a;**

**Employee emp1;**

**new Employee();**

**new Employee().display();**

**new Employee().display();**

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

**emp2.display();**

**emp2.display();**

**emp2.display();**

**}**

**}**

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

**Ex :**

**class**

**class Employee {**

**private String name;**

**private float salary;**

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

**this.name =name;**

**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[]) {**

**//display();**

**//name="Raj";**

**//salary = 12000;**

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

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

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

**emp.setValue("Ravi",-12000);**

**emp.display();**

**}**

**}**

**JavaBean class : JavaBean is a type of normal class with set of rules and regulation on property and behaviours.**

**Normal class Vs JavaBeanClass**

**class Employee {**

**private int id,age;**

**private String name;**

**//helper method**

**void setValue() {}**

**}**

**public class Employee {**

**private int id;**

**private String name;**

**private int age;**

**//setter and getter**

**//set is use to set the value with term and conditions.**

**//get is use to get the value**

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

**this.id = id;**

**}**

**public int getId() {**

**return id;**

**}**

**}**

**Pure Encapsulation class ie JavaBean**

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

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

**properties**

**behaviour**

**}**

**class NewClass extends OldClass{**

**properties**

**behaviour //sub class, derived class, child class.**

**}**

**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 {}**

**2. 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{}**

**3. Hierarchical Inheritance**

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

**class A{}**

**class B extends A{}**

**class C extends A{}**

**4. Multiple Inheritance**

**more than one super class and one sub class**

**class A {}**

**class B {}**

**class C extends A,B {} Java doesn't support this type of inheritance. This type of inheritance java support indirectly using interface.**

**class Employee {**

**id,name,salary**

**Scanner obj**

**read()**

**calSalary()**

**display()**

**}**

**class Manager extends Employee{**

**numberOfEmp**

**readMgr()**

**disMgr();**

**}**

**class Programmer extends Employee{**

**projectName;**

**readPrg()**

**disPrg()**

**}**

**OOPs Relationship**

**1. Manager is a Employee**

**2. Employee/Manager has a Address**

**class Manager extends Employee{**

**}**

**class Employee {**

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

**}**

**class Address {**

**}**

**has a relationship**

**1. Association**

**2.Aggregation**

**3. Composition**

**3. static variable**

**class A {**

**//B obj1 = new B();**

**}**

**class B {**

**A obj2 = new A();**

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

**}**

**class Employee {**

**Address ladd = new Address();**

**Address padd = new Address();**

**}**

**class Address {**

**}**

**class Student {**

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

**}**

**class StudentHistory {**

**}**

**0-0**

**0-1**

**0-many**

**1-0**

**1-1**

**many-1**

**many-many**

**Day 2 - Core Java - 05-09-2019**

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

**2 types**

**Compile time Run time**

**Static binding Dynamic binding**

**Early binding late binding**

**Ex: Method Overloading Ex: Method Overriding**

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

**DriverManager.getConnection("url")**

**DriverManager.getConnection("url","userName");**

**DriverManager.getConnection("url","userName","password");**

**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(String s1, String s2) {**

**System.out.println(s1+s2);**

**}**

**}**

**class Demo {**

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

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

**op.add("Raj","Deep");**

**}**

**}**

**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).**

**Method overriding we can't achieve without inheritance.**

**Annotation : Meta-data : Data about data.**

**Java provided lot of pre-define annotation.**

**Pre-defined annotation as well as user-defined annotation.**

**all annotation start with pre-fix @ followed by annotation name**

**Few or some annotation we can write on class level or method level or constructor level and property level.**

**@Override : This annotation we will write on method level.**

**class Bike {**

**void speed() {**

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

**}**

**}**

**class Pulsar extends Bike {**

**@Override**

**void speed() {**

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

**}**

**void color() {**

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

**}**

**}**

**class Honda extends Bike {**

**void color() {**

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

**}**

**}**

**class Tvs extends Bike {**

**@Override**

**void speed() {**

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

**super.speed();**

**}**

**void color() {**

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

**}**

**}**

**class Demo {**

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

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

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

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

**}**

**}**

**abstract : abstract is keyword we can use with method and class but not with variables.**

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

**syntax**

**abstract returnType methodName(parameterList);**

**abstract void speed();**

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

**syntax**

**abstract class className {**

**}**

**abstract class Bike {**

**}**

**3. Whichever class extends abstract class that class must be provide the body for all abstract method belongs to that class that class can ignore only if that class itself is a abstract class.**

**4. abstract class we can't create the objects.**

**5. abstract class can contains normal as well as abstract methods.**

**ie zero, 1 or many abstract methods.**

**6. Abstract class can contains default constructor as well as we can write parameterized constructor.**

**abstract class Bike {**

**int wheel;**

**Bike() {**

**wheel = 4;**

**}**

**void speed(){**

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

**}**

**}**

**abstract class Pulsar extends Bike {**

**void color() {**

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

**}**

**}**

**class Honda extends Bike {**

**@Override**

**void speed() {**

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

**}**

**void color() {**

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

**}**

**}**

**class Tvs extends Bike {**

**@Override**

**void speed() {**

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

**}**

**void color() {**

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

**}**

**}**

**class Demo {**

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

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

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

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

**}**

**}**

**abstract class Bike {**

**int wheel;**

**Bike() {**

**wheel = 4;**

**}**

**void speed(){**

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

**}**

**}**

**abstract class Pulsar extends Bike {**

**void color() {**

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

**}**

**}**

**class Honda extends Bike {**

**@Override**

**void speed() {**

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

**}**

**void color() {**

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

**}**

**}**

**class Tvs extends Bike {**

**@Override**

**void speed() {**

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

**}**

**void color() {**

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

**}**

**}**

**class Demo {**

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

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

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

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

**}**

**}**

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

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

**final int A=10;**

**2. final method : If the method is final we can't override that method.**

**3.final class : if the class is a final we can't extends or inherits that class.**

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

**static : static keyword we can use with variable and method but not with class. (if the class is a inner class then we can use static keyword for inner class but not for outer class).**

**1. if the variable are static we can assign the value for those variable using className as well as objects.**

**syntax**

**className.staticVariableName= value;**

**objectReferenceName.staticVariableName = value;**

**2. If the method is static we can call those method with the help of className as well as objectReferenceName;**

**className.staticMethodName();**

**objectReferenceName.staticMethodName();**

**3. Inside non static method we can access static as well as non static variable directly for the same class. But inside static method we can access only static variable directly of the same class.**

**class Outer {**

**int x;**

**static int y;**

**void dis1() {**

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

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

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

**}**

**static void dis2() {**

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

**Outer o = new Outer();**

**System.out.println("x "+o.x);**

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

**}**

**}**

**class Demo {**

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

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

**out.x=10;**

**Outer.y=20;**

**out.y=30;**

**out.dis1();**

**Outer.dis2();**

**out.dis2();**

**}**

**}**

**class Outer {**

**int x;**

**static int y;**

**void dis1() {**

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

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

**}**

**static void dis2() {**

**Outer o= new Outer();**

**System.out.println(y+", "+o.x);**

**}**

**}**

**class Demo {**

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

**Outer.dis2();**

**/\*Outer o1 = new Outer();**

**Outer o2 = new Outer();**

**o1.x=10;**

**Outer.y=20;**

**o1.y=30;**

**o2.x=40;**

**Outer.y=50;**

**o2.y=60;**

**o1.dis1();**

**o2.dis1(); \*/**

**}**

**}**

**interface : interface is a type of reference data type which is also known as 100% pure abstract class till JDK 7.**

**syntax**

**interface interfaceName {**

**fields;**

**methods;**

**}**

**by default all fields are public final static**

**by default all methods are public and abstract.**

**interface Abc {**

**public static final int A=0;**

**static final int B=20;**

**final int C=30;**

**int D=40;**

**public abstract void dis1();**

**abstract void dis2();**

**void dis3();**

**}**

**interface Abc {**

**int A=10;**

**void dis1();**

**}**

**interface Xyz {**

**int B=20;**

**void dis2();**

**}**

**interface Mno extends Abc,Xyz{**

**int C =30;**

**void dis3();**

**}**

**Super class /interface Sub class**

**public public**

**protected public**

**protected**

**default(nothing) public**

**protected**

**default**

**private 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 Demo1 implements Abc,Xyz {**

**public void dis1() {**

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

**}**

**public void dis2() {**

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

**}**

**}**

**class Demo {**

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

**Demo1 obj1 = new Demo1();**

**obj1.dis1();**

**obj1.dis2();**

**}**

**}**

**interface A {**

**int A1=10;**

**void dis1();**

**}**

**interface B {**

**int A1=20;**

**void dis1();**

**}**

**class D {**

**public void dis1() {**

**System.out.println("D class method coding ");**

**}**

**}**

**class C extends D implements A,B{**

**public void dis1() {**

**System.out.println("A,B and D coding");**

**System.out.println(A.A1 +" - "+B.A1);**

**}**

**}**

**C obj = new C();**

**obj.dis1();**

**this(), this, super, super():**

**class A {**

**int x=10;**

**}**

**class B extends A {**

**int x=20;**

**void dis1() {**

**int x=30;**

**System.out.println("x local variable "+x);**

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

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

**}**

**}**

**class Demo {**

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

**B obj = new B();**

**obj.dis1();**

**}**

**}**

**this() : this() is use to do constructor calling or constructor chaining of the same classes.**

**class Employee {**

**int id;**

**String name;**

**float salary;**

**Employee() {**

**this.id = 123;**

**this.name = "Unknown";**

**this.salary = 8000;**

**}**

**Employee(int id) {**

**this();**

**this.id = id;**

**}**

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

**this(id);**

**this.name = name;**

**}**

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

**}**

**void dis() {**

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

**}**

**}**

**class Demo {**

**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();**

**}**

**}**

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

**class A {**

**A() {**

**System.out.println("A class constructor");**

**}**

**A(int x) {**

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

**}**

**}**

**class B extends A {**

**B() {**

**super(10);**

**System.out.println("B class constructor");**

**}**

**}**

**class Demo {**

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

**B obj1 = new B();**

**}**

**}**

**class A {**

**A() {**

**this(100);**

**System.out.println("A class constructor");**

**}**

**A(int x) {**

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

**}**

**}**

**class B extends A {**

**B() {**

**this(10);**

**System.out.println("B class constructor");**

**}**

**B(int x) {**

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

**}**

**}**

**class Demo {**

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

**B obj1 = new B();**

**}**

**}**

**abstract class Employee {**

**int id;**

**String name;**

**float salary;**

**Employee() {**

**this.id = 123;**

**this.name = "Unknown";**

**this.salary = 8000;**

**}**

**Employee(int id) {**

**this();**

**this.id = id;**

**}**

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

**this(id);**

**this.name = name;**

**}**

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

**this(id,name);**

**this.salary = salary;**

**}**

**void dis() {**

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

**}**

**}**

**class Manager extends Employee {**

**int numberOfEmp;**

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

**super(id,name,salary);**

**this.numberOfEmp = numberOfEmp;**

**}**

**@Override**

**void dis() {**

**super.dis();**

**System.out.println("Number of Emp "+this.numberOfEmp);**

**}**

**}**

**class Demo {**

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

**Manager mgr = new Manager(1,"Raj",45000,10);**

**mgr.dis();**

**}**

**}**

**init or initialization block and static block**

**class A {**

**A() {**

**System.out.println("A class constructor");**

**}**

**static {**

**System.out.println("static block A class ");**

**}**

**{**

**System.out.println("init bock");**

**}**

**void dis() {**

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

**}**

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

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

**String str[]={};**

**Demo.main(str);**

**}**

**}**

**class Demo {**

**static**

**{**

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

**}**

**{**

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

**}**

**Demo () {**

**System.out.println("Demo class constructor");**

**}**

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

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

**obj1.dis();**

**obj1.dis();**

**A obj2 = new A();**

**obj2.dis();**

**obj2.dis();**

**}**

**}**

**class A {**

**A() {**

**System.out.println("A class constructor");**

**}**

**static {**

**System.out.println("static block A class ");**

**}**

**{**

**System.out.println("init A bock");**

**}**

**void dis1() {**

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

**}**

**}**

**class B extends A {**

**B() {**

**System.out.println("B class constructor");**

**}**

**static {**

**System.out.println("static block B class ");**

**}**

**{**

**System.out.println("init B bock");**

**}**

**void dis2() {**

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

**}**

**}**

**class Demo {**

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

**new B();**

**}**

**}**

**class A {**

**void dis1() {**

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

**}**

**}**

**class B extends A {**

**void dis1() {**

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

**}**

**void dis2() {**

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

**}**

**}**

**class Demo {**

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

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

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

**A obj3 = new B(); obj3.dis1(); //obj3.dis2();**

**B obj4 = (B)obj3; obj4.dis1(); obj4.dis2();**

**//B obj4 = new A();**

**}**

**}**

**Super class is type of abstract class**

**abstract class A {**

**abstract void dis1();**

**}**

**class B extends A {**

**void dis1() {**

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

**}**

**void dis2() {**

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

**}**

**}**

**class Demo {**

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

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

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

**A obj3 = new B(); obj3.dis1(); //obj3.dis2();**

**B obj4 = (B)obj3; obj4.dis1(); obj4.dis2();**

**//B obj4 = new A();**

**}**

**}**

**interface reference**

**interface A {**

**void dis1();**

**}**

**class B implements A {**

**public void dis1() {**

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

**}**

**void dis2() {**

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

**}**

**}**

**class Demo {**

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

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

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

**A obj3 = new B(); obj3.dis1(); //obj3.dis2();**

**B obj4 = (B)obj3; obj4.dis1(); obj4.dis2();**

**//B obj4 = new A();**

**}**

**}**

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

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

**Day 3 - Core Java - 06-09-2019**

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

**package are divided into 2 types**

**1. user-defined package.**

**2. pre-defined or built-in package.**

**Education**

**School College PG**

**Attendance.java Attendance.java Attendance.java**

**sql.Date**

**util.Date**

**package com;**

**public class Demo {**

**}**

**Access Specifiers :**

**private :**

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

**Scope within a same class.**

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

**Scope within a same package.**

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

**scope within a same package other package if it is sub class**

**public : instance variable, static variable, non static method, static method, constructor, class(in one editor it may be notepad or eclipse editor only one class we can us public access spcifiers), but not with local variable.**

**scope same package as well as other package.**

**Demo.java**

**package bean;**

**public class Demo{**

**private int a;**

**int b;**

**protected int c;**

**public int d;**

**}**

**DemoTest.java**

**package com;**

**import bean.Demo;**

**public class DemoTest extends Demo{**

**void dis() {**

**c and d**

**}**

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

**DemoTest dt = new DemoTest(); td.dis();**

**}**

**}**

**Pre-defined packages**

**java javax ----> Root package**

**lang : language sql**

**io : input/output rmi**

**util : utility net**

**sql : structure query language servlet**

**awt : abstract window toolkit ejb : Enterprise Java Bean**

**net : networking jms: Java Messaging Service**

**rmi : remote method invocation**

**etc etc**

**lang package**

**Exception Handling**

**Multithreading**

**lang package**

**classes and interfaces**

**String**

**StringBuffer**

**StringBuilder**

**Math**

**Process**

**RunTime**

**Cloneable**

**Runnable**

**Object**

**passing the object as a parameter and Return the complex objects.**

**bean package**

**class Employee {**

**id,name,salary ,desg,age**

**setter and getter methods**

**}**

**service package**

**class EmployeeService {**

**public void display() {**

**}**

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

**}**

**public String getInfo() {**

**return "Welcome";**

**}**

**public void passEmployeeInfo(int id, String name,float salary,String desg) {**

**}**

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

**return x+y;**

**}**

**public void passObject(Employee emp) {**

**}**

**public Employee getEmpObject(int code) {**

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

**//emp.setId(100);**

**//emp.setName("Ravi");**

**//emp.setSalary(12000);**

**return emp;**

**}**

**}**

**Exception Handling : Exception is a object, which occurs when unexpected or abnormal conditions during the execution of a programs.**

**Java**

**Compile Run**

**javac (java compiler) java(java interpreter)**

**Compile time error Run time error**

**syntax error**

**or**

**typo error**

**class ExpDemo {**

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

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

**for(String str:abc) {**

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

**}**

**}**

**}**

**Run time Error**

**error**

**Error Exception**

**Error :It is a pre-defined class part of lang package. Error is a type of error which generate at the run time which we can't handle it. JVM Crash, Software or Hardware issue, StackOverFlow etc**

**Exception : It is pre-defined class part of lang package. Exception is a type of error which generate at the run time which we can handle it.**

**ArthmeticException, NumberFormatException etc.**

**Object**

**Throwable**

**Exception pre-defined class**

**CheckedException UnCheckedException**

**RuntimeException**

**SQLException ArithmeticException**

**IOException NumberFormatException**

**ArrayIndexOutOfBoundsException**

**etc etc**

**throw and throws**

**Custom Exception**

**finally**

**To handle both type of exception java provided 5 keywords**

**1. try**

**2. catch**

**3. finally**

**4. throw**

**5. throws**

**try and catch block**

**syntax**

**try {**

**try block**

**}catch(Exception e){**

**catch block**

**}**

**class ExpDemo {**

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

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

**int a = 10;**

**int b =0;**

**try{**

**int res = a/b;**

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

**}catch(Exception e) {**

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

**}**

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

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

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

**}**

**}**

**class Abc {**

**void dis() {}**

**}**

**class ExpDemo {**

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

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

**obj.dis();**

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

**int a = 10;**

**int b =1;**

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

**try{**

**int res = a/1;**

**obj = null;**

**obj.dis();**

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

**int res1 = 10/info[5];**

**System.out.println("Res1 "+res1);**

**}catch(ArithmeticException e) {**

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

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

**}catch(ArrayIndexOutOfBoundsException e) {**

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

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

**}catch(Exception e){**

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

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

**}**

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

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

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

**}**

**}**

**try with multiple catch blocks**

**class Abc {**

**void dis() {}**

**}**

**class ExpDemo {**

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

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

**obj.dis();**

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

**int a = 10;**

**int b =1;**

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

**try{**

**int res = a/1;**

**obj = null;**

**obj.dis();**

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

**int res1 = 10/info[5];**

**System.out.println("Res1 "+res1);**

**}catch(ArithmeticException e) {**

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

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

**}catch(ArrayIndexOutOfBoundsException e) {**

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

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

**}catch(Exception e){**

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

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

**}**

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

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

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

**}**

**}**

**try :**

**catch : catch block will execute only if the any exception generate.**

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

**try {**

**}catch(Exception e) {**

**}finally {**

**}**

**try**

**catch catch catch catch finally**

**catch finally catch**

**finally**

**class ExpDemo {**

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

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

**try{**

**int res = 10/0;**

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

**}finally {**

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

**}**

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

**}**

**}**

**try {**

**open the file /connect the database**

**read/write operation**

**}catch(Exception e){**

**close the file**

**}finally {**

**close the resources**

**}**

**class ExpDemo {**

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

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

**try{**

**int res = 10/0;**

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

**}finally {**

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

**}**

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

**}**

**}**

**throw and throws**

**throw : throw keyword is use to generate or raise pre-defined or user-defined(custom exception ) with term and conditions.**

**syntax**

**throw new Exception();**

**or**

**throw new ExceptionSubClass();**

**class MyException extends Exception {**

**MyException() {}**

**MyException(String str) {**

**super(str);**

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

**}**

**}**

**class ExpDemo {**

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

**try{**

**int a=100;**

**int b=20;**

**if(a>b) {**

**//System.out.println("a>b");**

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

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

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

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

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

**}else {**

**int res = 10/0;// new ArithmeticException("/ by Zero");**

**}**

**}catch(Exception e) {**

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

**}**

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

**}**

**}**

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

**syntax**

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

**}**

**class MyException extends Exception {**

**MyException() {}**

**MyException(String str) {**

**super(str);**

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

**}**

**}**

**class ExpDemo {**

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

**try{**

**//int abc[]={1};**

**//int res = 10/abc[3];**

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

**throw new MyException("custom msg");**

**}catch(ArrayIndexOutOfBoundsException 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[] abc) throws Exception{**

**dis2();**

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

**}**

**}**

**class ExpDemo {**

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

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

**try{**

**Thread.sleep(2000);**

**}catch(Exception e){**

**}**

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

**}**

**}**

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

**emp.id=100;**

**emp.name="Raj";**

**emp.salary = 12000;**

**Array objects**

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

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

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

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

**emp[0].display();**

**int abc[]=new int[100];**

**Create Eclipse Project. Create three packages bean, service and main. bean package must be contains Employee class with three instance variable id, name, salary and setter/getter methods.**

**service package contains EmployeeService class with business methods.**

**Create Employee array object which help to store more than one employee class objects. In EmployeeService class.**

**1 storeEmployeeInfo with Employee class object as a parameter.**

**Here employee id must be unique and salary must > 8000. return msg**

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

**coding**

**}**

**2. deleteEmployeeInfo using id. pass id , return msg.**

**public String deleteEmployeeInfoById(int id) {**

**coding**

**}**

**3. update employeeSalary using id. pass id and salary , return msg**

**4. displayEmployee using id pass id and return Employee object**

**5. displayAllEmployee details. nothing but return array objects.**

**Mainclass**

**do{**

**1: Add**

**2: Delete**

**3:Update**

**4:DisplayOneEmployee**

**5:DisplayAll**

**switch() {**

**case 1**

**case 2**

**case 3**

**case 4**

**case 5**

**}**

**do wan to continue ?**

**}while()**

**Day 4 - Core Java - 09-09-2019**

**Multithreading :**

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

**Processor : Processor process the program or task.**

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

**Thread : Small execution of code within in a process. It is also known as light weighted process.**

**By default Java is thread process programming language.**

**By default C/C++ are process programming language.**

**Thread : It is pre-defined class part of lang package. By default every java program imported lang package.**

**Thread contains one method currentThread() it is a static method.**

**Thread.currentThread();**

**currentThread() method return type is Thread class reference.**

**public static Thread currentThread() {**

**return t;**

**}**

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

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

**main --->Name of the thread**

**5---> priority of the thread**

**main---> Group of thread**

**t.setName("My Threads");**

**t.setPriortiy(0) min --0, max -10**

**t.setPriority(3);**

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

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

**class ThreadDemo {**

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

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

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

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

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

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

**t.setPriority(2);**

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

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

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

**}**

**}**

**Multitasking :**

**1. Process based**

**2. Thread based**

**processor**

**t 5 min t 5 min t 5 min**

**p1 10 p2 10 p3 10**

**Multithreading :**

**2 ways**

**1. extends Thread class**

**a. We have to create the Thread class reference.**

**b. With reference call the pre-defined method start().**

**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);**

**}**

**}**

**}**

**class ThreadDemo {**

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

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

**B obj2 = new B(); //object created...**

**obj1.start();**

**obj2.start(); // ready to start Runnable state**

**}**

**}**

**2. implements Runnable interface**

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

**javap java.lang.Runnable**

**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);**

**}**

**}**

**}**

**class ThreadDemo {**

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

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

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

**Thread t1 = new Thread(obj1); //object created...**

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

**t1.start();**

**t2.start(); // ready to start Runnable state**

**}**

**}**

**class A implements Runnable{**

**@Override**

**public void run() {**

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

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

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

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

**}**

**}**

**}**

**class B implements Runnable{**

**@Override**

**public void run() {**

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

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

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

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

**}**

**}**

**}**

**class ThreadDemo {**

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

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

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

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

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

**Thread t1 = new Thread(obj1); //object created...**

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

**t1.setPriority(Thread.MAX\_PRIORITY);**

**t2.setPriority(Thread.MIN\_PRIORITY);**

**t1.start();**

**t2.start(); // ready to start Runnable state**

**}**

**}**

**Life cycle of a thread**

**sleep()**

**isAlive()**

**join()**

**wait()**

**notify()**

**notifyAll()**

**create -------->Runnable -----------> Running -----------> destroy**

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

**t1 t1.start()**

**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);**

**}**

**}**

**}**

**class ThreadDemo {**

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

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

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

**Thread t1 = new Thread(obj1); //object created...**

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

**t1.run();**

**t2.run();**

**}**

**}**

**class A extends Thread{**

**public void start() {**

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

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

**}**

**}**

**}**

**class B extends Thread{**

**public void start() {**

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

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

**}**

**}**

**}**

**class ThreadDemo {**

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

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

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

**obj1.start();**

**obj2.start();**

**}**

**}**

**class A implements Runnable{**

**public void run() {**

**String name = Thread.currentThread().getName();**

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

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

**try{**

**Thread.sleep(1000);**

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

**}**

**}**

**}**

**class B implements Runnable{**

**public void run() {**

**String name = Thread.currentThread().getName();**

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

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

**try{**

**Thread.sleep(500);**

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

**}**

**}**

**}**

**class ThreadDemo {**

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

**//new Thread(new A(),"Raj").start();**

**//new Thread(new B(),"Seeta").start();**

**Thread t1 = new Thread(new A(),"Raj");**

**Thread t2 = new Thread(new B(),"Seeta");**

**System.out.println("Raj Thread "+t1.isAlive());**

**System.out.println("Seeta Thread "+t2.isAlive());**

**t1.start();**

**t2.start();**

**System.out.println("Raj Thread "+t1.isAlive());**

**System.out.println("Seeta Thread "+t2.isAlive());**

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

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

**Thread.sleep(1200);**

**}**

**}**

**}**

**class A implements Runnable{**

**public void run() {**

**String name = Thread.currentThread().getName();**

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

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

**try{**

**Thread.sleep(1000);**

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

**}**

**}**

**}**

**class B implements Runnable{**

**public void run() {**

**String name = Thread.currentThread().getName();**

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

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

**try{**

**Thread.sleep(500);**

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

**}**

**}**

**}**

**class ThreadDemo {**

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

**Thread t1 = new Thread(new A(),"1st Bike");**

**Thread t2 = new Thread(new B(),"2nd Bike");**

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

**Thread.sleep(1000);**

**System.out.println("1.....");**

**Thread.sleep(1000);**

**System.out.println("2.....");**

**Thread.sleep(1000);**

**System.out.println("3.....");**

**System.out.println("1st Bike "+t1.isAlive());**

**System.out.println("2nd Bike "+t2.isAlive());**

**t1.start();**

**t2.start();**

**System.out.println("1st Bike "+t1.isAlive());**

**System.out.println("2nd Bike "+t2.isAlive());**

**t1.join();**

**t2.join();**

**System.out.println("1st Bike "+t1.isAlive());**

**System.out.println("2nd Bike "+t2.isAlive());**

**System.out.println("Game Finished.....");**

**}**

**}**

**class A implements Runnable{**

**public void run() {**

**String name = Thread.currentThread().getName();**

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

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

**try{**

**Thread.sleep(1000);**

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

**}**

**}**

**}**

**class ThreadDemo {**

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

**Thread t1 = new Thread(new A(),"1st Bike");**

**Thread t2 = new Thread(new A(),"2nd Bike");**

**Thread t3 = new Thread(new A(),"3rd Bike");**

**Thread t4 = new Thread(new A(),"4th Bike");**

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

**Thread.sleep(1000);**

**System.out.println("1.....");**

**Thread.sleep(1000);**

**System.out.println("2.....");**

**Thread.sleep(1000);**

**System.out.println("3.....");**

**t1.start();**

**t2.start();**

**t3.start();**

**t4.start();**

**t1.join();**

**t2.join();**

**t3.join();**

**t4.join();**

**System.out.println("Game Finished.....");**

**}**

**}**

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

**class A implements Runnable{**

**public void run() {**

**String name = Thread.currentThread().getName();**

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

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

**try{**

**Thread.sleep(1000);**

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

**}**

**}**

**}**

**class ThreadDemo {**

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

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

**Thread t1 = new Thread(obj1,"1st Bike");**

**Thread t2 = new Thread(obj1,"2nd Bike");**

**Thread t3 = new Thread(obj1,"3rd Bike");**

**Thread t4 = new Thread(obj1,"4th Bike");**

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

**Thread.sleep(1000);**

**System.out.println("1.....");**

**Thread.sleep(1000);**

**System.out.println("2.....");**

**Thread.sleep(1000);**

**System.out.println("3.....");**

**t1.start();**

**t2.start();**

**t3.start();**

**t4.start();**

**t1.join();**

**t2.join();**

**t3.join();**

**t4.join();**

**System.out.println("Game Finished.....");**

**}**

**}**

**wait(), notify() and notifyAll() :**

**wait() method is use to make the thread to wait or suspend() the threads with conditions.**

**notify() method is use to call back waited thread or resume() waited thread**

**notifyAll() method is use to call back all waited thread or all thread resume();**

**Conditions :**

**1. We have to create more than one thread in same memory.**

**2. The method must be synchronized.**

**3. wait(), notify() and notifyAll() method are part of Object class not a part of Thread class.**

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

**To achieve synchronization we have to use synchronized keyword.**

**This keyword we can use with method as well as synchronized block within a method.**

**yield() :**

**class Booking implements Runnable{**

**int avl =1;**

**public synchronized void run() {**

**String name = Thread.currentThread().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[]) {**

**Booking b1 = new Booking();**

**Thread t1 = new Thread(b1,"Raj");**

**Thread t2 = new Thread(b1,"Seeta");**

**Thread t3 = new Thread(b1,"Reeta");**

**t3.setPriority(8);**

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

**}**

**}**

**Inter thread communication using wait(),notify() and notifyAll() methods**

**class Task implements Runnable{**

**public synchronized void run() {**

**String name = Thread.currentThread().getName();**

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

**try{**

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

**if(name.equals("Raj") && i==4) {**

**wait();**

**}**

**if(name.equals("Seeta") && i==6){**

**//notify();**

**wait();**

**}**

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

**notify();**

**//notifyAll();**

**//wait();**

**}**

**Thread.sleep(700);**

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

**}**

**}**

**}**

**class ThreadDemo {**

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

**Task tt = new Task();**

**Thread t1 = new Thread(tt,"Raj");**

**Thread t2 = new Thread(tt,"Seeta");**

**Thread t3 = new Thread(tt,"Ajay");**

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

**}**

**}**

**Consumer and Procedure example with wait(), notify()**

**class Put implements Runnable{**

**Thread t;**

**Work ww;**

**Put(Work ww) {**

**this.ww =ww;**

**t = new Thread(this);**

**t.start();**

**}**

**public void run() {**

**int i=0;**

**while(i<=10) {**

**ww.put(i);**

**i++;**

**}**

**}**

**}**

**class Get implements Runnable{**

**Thread t;**

**Work ww;**

**Get(Work ww) {**

**this.ww = ww;**

**t = new Thread(this);**

**t.start();**

**}**

**public void run() {**

**int i=0;**

**while(i<=10) {**

**ww.get();**

**i++;**

**}**

**}**

**}**

**class Work {**

**boolean result=false;**

**int n;**

**synchronized void get() {**

**if(!result)**

**try {**

**wait();**

**} catch(InterruptedException e) {**

**System.out.println("InterruptedException caught");**

**}**

**System.out.println("Got: " + n);**

**result = false;**

**notify();**

**}**

**synchronized void put(int n) {**

**if(result)**

**try {**

**wait();**

**} catch(InterruptedException e) {**

**System.out.println("InterruptedException caught");**

**}**

**this.n = n;**

**result = true;**

**System.out.println("Put: " + n);**

**notify();**

**}**

**}**

**class ConsumerAndProducerThread {**

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

**Work ww = new Work();**

**Put pp = new Put(ww);**

**Get gg = new Get(ww);**

**}**

**}**

**types of thread**

**user-defined thread**

**daemon thread (pre-defined threads).**

**class A implements Runnable {**

**public void run() {**

**String name = Thread.currentThread().getName();**

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

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

**}**

**}**

**}**

**class DaemonThreadDemo {**

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

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

**Thread t1 = new Thread(obj1,"First");**

**Thread t2 = new Thread(obj1,"Second");**

**Thread t3 = new Thread(obj1,"Third");**

**t1.start();**

**t2.setDaemon(true);**

**t2.start();**

**//t3.start();**

**}**

**}**

**Day 5 - Core Java - 10-09-2019**

**lang package and inner classes**

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

**Few pre-defined classes and interfaces belongs to lang package.**

**String class**

**StringBuffer class**

**StringBuilder class**

**Object class**

**Math class**

**Wrapper classes**

**Runtime class**

**Process class**

**System class**

**Cloneable interface**

**Exception and Error classes**

**Thread class**

**Runnable interface**

**String : In Java String is a pre-defined class part of lang package. But it behave like data types. In C/C++ string is data types.**

**Syntax to create the String class reference**

**String str1 = "Welcome to Java";**

**String str2 = new String("Welcome to Java");**

**string : combination of one or more than one character enclosed in double quotes.**

**char name='a'; C/C++/Java**

**char name[10]={"Raj Deep"}; C/C++**

**char name[]={"Raj Deep"} C/C++**

**char name\* ="Raj Deep"; C/C++**

**char name[10]; Wrong in Java**

**char name[]={'R','a','j'}; Correct in Java**

**String str1 = "Welcome to Java";**

**String str2 = new String("Welcome to Java");**

**char name[]={'R','a','j'};**

**String str3 = new String(name);**

**class StringDemo {**

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

**String str1 = "Welcome to Java Training";**

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

**System.out.println(str1);**

**System.out.println(str2);**

**System.out.println("Number of character "+str1.length());**

**System.out.println(str1.toUpperCase());**

**System.out.println(str1.toLowerCase());**

**System.out.println(str1.substring(4)); //start from 0 position**

**System.out.println(str1.substring(4,10)); //start from 0 position and end 1**

**System.out.println(str1.indexOf('e'));**

**System.out.println(str1.lastIndexOf('i'));**

**}**

**}**

**class StringDemo {**

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

**String str1 = "Raj";**

**String str2 = "Raj ";**

**String str3 = new String("Raj");**

**String str4 = new String("raj");**

**if(str1.equalsIgnoreCase(str2.trim())) {**

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

**}else {**

**System.out.println("Not Equal");**

**}**

**}**

**}**

**String class is known as immutable class. Immutable means can't change.**

**class StringDemo {**

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

**String str1 = "Raj";**

**System.out.println(str1);**

**str1 = str1+" Deep";**

**System.out.println(str1); //**

**System.out.println(str1.toUpperCase());**

**System.out.println(str1);**

**}**

**}**

**StringBuffer and StringBuilder : There are type of String classes which known as Mutable string classes.**

**class StringDemo {**

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

**String str1 = "Raj";**

**System.out.println(str1);**

**str1 = str1+" Deep";**

**System.out.println(str1); //**

**System.out.println(str1.toUpperCase());**

**System.out.println(str1);**

**StringBuffer sb = new StringBuffer("Raj"); //converting String to StringBuffer**

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

**System.out.println(sb.append(" Deep"));**

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

**sb.insert(4,"Kumar ");**

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

**sb.delete(10,12);**

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

**sb.reverse();**

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

**String str = sb.toString(); //StringBuffer to String**

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

**}**

**}**

**StringBuffer by default methods are synchronized. Work is safe but performance is slow. Where StringBuilder method are not synchronized. Work is fast for not safe.**

**Object class : By default every java class it may be pre-defined or user-defined extends Object class.**

**Super API in Java API means(Application Programming interfaces) is Object.**

**Class :**

**Void**

**Package**

**Object class methods**

**getClass()**

**hashCode()**

**equals()**

**clone()**

**toString()**

**notify()**

**notifyAll()**

**wait()**

**finalize()**

**class Employee {**

**public void receiveObject(Object obj) {**

**Class cc = obj.getClass();**

**}**

**}**

**class Manager {}**

**class StringDemo {**

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

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

**Class cc = emp.getClass();**

**if(cc.isInstance(emp)) {**

**System.out.println(" Emloyee Object Yes");**

**}**

**}**

**}**

**hashCode method**

**class Employee {**

**}**

**class StringDemo {**

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

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

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

**Employee emp3 = emp1;**

**System.out.println(emp1.hashCode());**

**System.out.println(emp2.hashCode());**

**System.out.println(emp3.hashCode());**

**}**

**}**

**class Employee {**

**int id;**

**String name;**

**float salary;**

**Employee() {}**

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

**this.id = id;**

**this.name = name;**

**this.salary = salary;**

**}**

**@Override**

**public boolean equals(Object obj) {**

**Employee emp = (Employee)obj;**

**if(emp==null){**

**return false;**

**} else if(emp==this) {**

**return true;**

**}**

**else if(this.salary== emp.salary && this.name.equals(emp.name)){**

**return true;**

**}else {**

**return false;**

**}**

**}**

**}**

**class StringDemo {**

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

**String name1=new String("Raj");**

**String name2 = new String("Raj");**

**Employee emp1 = new Employee(1,name1,14000);**

**Employee emp2 = new Employee(2,name2,14000);**

**if(emp1.equals(emp2)) {**

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

**}else {**

**System.out.println("Not Equal");**

**}**

**}**

**}**

**toString() method**

**class Employee {**

**private int id;**

**private String name;**

**private float salary;**

**Employee() {}**

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

**this.id = id;**

**this.name = name;**

**this.salary = salary;**

**}**

**@Override**

**public String toString() {**

**return "Id "+this.id+" Name is "+this.name +" Salary "+this.salary;**

**//return "Employee class object";**

**}**

**}**

**class StringDemo {**

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

**Employee emp1 = new Employee(1,"Raj",14000);**

**System.out.println(emp1.toString());**

**}**

**}**

**finalize() method**

**class Employee {**

**private int id;**

**private String name;**

**private float salary;**

**Employee() {}**

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

**this.id = id;**

**this.name = name;**

**this.salary = salary;**

**}**

**@Override**

**public String toString() {**

**return "Id "+this.id+" Name is "+this.name +" Salary "+this.salary;**

**//return "Employee class object";**

**}**

**@Override**

**public void finalize() {**

**System.out.println("Pre-GC");**

**}**

**}**

**class StringDemo {**

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

**Employee emp1 = new Employee(1,"Raj",14000);**

**System.out.println(emp1.toString());**

**emp1 = null;**

**//System.gc();**

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

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

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

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

**}**

**}**

**clone() : clone method is use to create the clone or duplicate objects.**

**class Employee implements Cloneable{**

**int id;**

**String name;**

**float salary;**

**Employee() {}**

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

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

**this.id = id;**

**this.name = name;**

**this.salary = salary;**

**}**

**@Override**

**public String toString() {**

**return "Id "+this.id+" Name is "+this.name +" Salary "+this.salary;**

**//return "Employee class object";**

**}**

**@Override**

**public void finalize() {**

**System.out.println("Pre-GC");**

**}**

**Employee emp;**

**public Employee getCloneObject(){**

**try{**

**Object obj = clone();**

**emp = (Employee)obj;**

**}catch(Exception e) {**

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

**}**

**return emp;**

**}**

**public Object clone() throws CloneNotSupportedException {**

**return super.clone();**

**}**

**}**

**class StringDemo {**

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

**Employee emp1 = new Employee(1,"Raj",14000);**

**Employee emp2 = new Employee(2,"Ravi",16000);**

**Employee emp3 =emp1;**

**System.out.println(emp1);**

**System.out.println(emp2);**

**System.out.println(emp3);**

**Employee emp4 = emp1.getCloneObject();**

**System.out.println("After clone created....");**

**System.out.println(emp1);**

**System.out.println(emp4);**

**emp1.salary = 45000;**

**System.out.println(emp1);**

**System.out.println(emp4);**

**}**

**}**

**Math class :**

**Math class provided set of methods which help to do mathematical operations.**

**Math class also known as static class.**

**Because in Math class all fields and method are static.**

**class StringDemo {**

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

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

**System.out.println(Math.sqrt(9));**

**}**

**}**

**Wrapper classes**

**Java provided set of wrapper classes which wrap the all primitive data types.**

**Use of Wrapper classes**

**int a=10;**

**Integer b =new Integer(a);**

**All wrapper classes provided more functionality on primitive data types.**

**With the help of wrapper classes we can do type casting as well as primitive to object and object to primitive conversation.**

**8 primitive data types 8 wrapper classes**

**byte Byte**

**short Short**

**int Integer**

**long Long**

**float Float**

**double Double**

**char Character**

**boolean Boolean**

**class StringDemo {**

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

**int a=10;**

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

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

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

**int c = b.intValue(); //converting Integer object to int primitive**

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

**float f = b.floatValue();**

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

**String s1 = "10";**

**String s2 = "20";**

**System.out.println(s1+s2);**

**int s3 = Integer.parseInt(s1); //converting string to int**

**int s4 = Integer.parseInt(s2);**

**System.out.println(s3+s4);**

**String s5= String.valueOf(s3); //converting int to string**

**String s6= String.valueOf(s4);**

**System.out.println(s5+s6);**

**}**

**}**

**Runtime class**

**Process class**

**class A{}**

**class StringDemo {**

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

**Runtime rr = Runtime.getRuntime();**

**/\*A obj= null;**

**for(int i=0;i<1000000;i++) {**

**obj =new A();**

**}**

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

**System.out.println(rr.totalMemory());\*/**

**Process pp = rr.exec("calc");**

**Thread.sleep(5000);**

**pp.destroy();**

**}**

**}**

**Inner classes and nested classes**

**A class within a another class is known as inner class.**

**That class may be non static or static.**

**4 types**

**1. non static inner class**

**2. static inner class**

**3. method or block local class**

**4. anonymous inner class**

**non static inner class**

**non static inner class behave like a instance member for outer class.**

**class Outer {**

**int a;**

**void dis1() {**

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

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

**//in.dis2();**

**}**

**class Inner {**

**int b;**

**void dis2() {**

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

**dis1();**

**}**

**}**

**}**

**class InnerDemo{**

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

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

**//out.dis1();**

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

**//in.dis2();**

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

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

**in.dis2();**

**}**

**}**

**class Outer1 {**

**int a=10;**

**}**

**class Outer extends Outer1{**

**void dis1() {**

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

**}**

**class Inner {**

**void dis2() {**

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

**}**

**}**

**class Inner1 extends Inner {**

**}**

**}**

**class InnerDemo{**

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

**}**

**}**

**static inner class**

**class Outer {**

**int a;**

**static int b;**

**void dis1() {**

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

**Inner.dis4();**

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

**in.c=10;**

**in.dis3();**

**Inner in1 =new Inner();**

**in1.c=20;**

**in1.dis3();**

**}**

**static void dis2() {**

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

**}**

**static class Inner {**

**int c;**

**static int d;**

**void dis3() {**

**int e;**

**System.out.println("Inner dis3() method "+c);**

**}**

**static void dis4() {**

**System.out.println("Inner dis4() static method");**

**}**

**}**

**}**

**class InnerDemo{**

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

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

**out.dis1();**

**Outer.Inner in = new Outer.Inner(); //creating the object of static inner class**

**in.dis3(); //calling static class non static method**

**Outer.Inner.dis4(); //calling static class static method**

**}**

**}**

**local or block or method scope inner class**

**class Outer {**

**void dis1() {**

**System.out.println("dis1 Method coding");**

**int a=10;**

**class Inner {**

**void dis2() {**

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

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

**}**

**}**

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

**in.dis2();**

**}**

**}**

**class InnerDemo{**

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

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

**out.dis1();**

**}**

**}**

**anonymous class :**

**abstract class A {**

**abstract void dis1();**

**}**

**class B extends A {**

**public void dis1() {**

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

**}**

**}**

**class InnerDemo{**

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

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

**obj1.dis1();**

**A obj2 = new A(){**

**public void dis1() {**

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

**}**

**};**

**obj2.dis1();**

**}**

**}**

**Anonymous class for interface**

**interface A {**

**void dis1();**

**}**

**class B implements A {**

**public void dis1() {**

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

**}**

**}**

**class InnerDemo{**

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

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

**obj1.dis1();**

**A obj2 = new A(){**

**public void dis1() {**

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

**}**

**};**

**obj2.dis1();**

**}**

**}**

**Day 6 - Core Java - 11-09-2019**

**io packages**

**input and output package.**

**stream : flow of data. It is abstraction between source and destination or target.**

**source --> may standard input device ie keyword, file, thread, database, network etc.**

**destination --> may standard output device ie. monitor or console or browser or device, file, thread, database, network etc.**

**In java we can achieve stream using 2 ways**

**1. byte wise**

**2. char wise**

**stream**

**byte char**

**Input operation output operation input operation output operation**

**InputStream OutputStream Reader Writer**

**Abstract class Abstract class Abstract class Abstract class**

**DataInputStream DataOutputStream InputStreamReader**

**OutputStreamWriter**

**FileInputStream FileOutputStream FileReader FileWriter**

**BufferedInputStream/BufferedOutputStream BufferedReader /BufferedWriter**

**ObjectInputStream /ObjectOutputStream**

**PrintStream PrintWriter**

**System.in ----> InputStream class reference. InputStream class provide the reference of standard input device ie keyword.**

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

**System.out : PrintStream class reference. PrintStream class provide the reference of standard output device ie monitor or console.**

**PrintStream ps = System.out;**

**ps.println("Welcome to Java");**

**DataInputStream and PrintStream**

**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);**

**ps.println("Enter the id");**

**int id = Integer.parseInt(dis.readLine());**

**ps.println(" Id is "+id);**

**}**

**}**

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

**class IoDemo {**

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

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

**PrintStream ps = System.out;**

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

**ps.println("Enter the data");**

**int ch;**

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

**System.out.println(ch+"--"+(char)ch);**

**fos.write(ch);**

**}**

**fos.write('\n');**

**fos.close();**

**}**

**}**

**Read and Write operation using FileInputStream and FileOutputStream classes**

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

**class IoDemo {**

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

**FileInputStream fis = new FileInputStream("CarTest.java");**

**FileOutputStream fos = new FileOutputStream("CarTest.doc");**

**int ch;**

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

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

**fos.write(ch);**

**}**

**fis.close();**

**fos.close();**

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

**}**

**}**

**BuffereInputStream and BufferedOutputStream :**

**Buffer : It is tempory memory. Which help to improve the performance for read as well as write operations.**

**Input File --------Buffer----------> Program ----------Buffer --------> Output File**

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

**class IoDemo {**

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

**FileInputStream fis = new FileInputStream("CarTest.java");**

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

**FileOutputStream fos = new FileOutputStream("CarTest.doc");**

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

**int ch;**

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

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

**bos.write(ch);**

**}**

**bos.flush();**

**fis.close();**

**fos.close();**

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

**}**

**}**

**DataInputStream and DataOutputStream with Primitive data types**

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

**class IoDemo {**

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

**/\*FileOutputStream fos = new FileOutputStream("empInfo.txt");**

**DataOutputStream dos = new DataOutputStream(fos);**

**dos.writeInt(100);**

**dos.writeUTF("Ravi");**

**dos.writeFloat(12000);**

**dos.close();**

**fos.close();**

**System.out.println("Employee details stored...");\*/**

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

**DataInputStream dis = new DataInputStream(fis);**

**System.out.println("Id is "+dis.readInt());**

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

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

**}**

**}**

**FileReader and FileWriter class Example**

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

**class IoDemo {**

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

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

**BufferedReader br =new BufferedReader(fr);**

**FileWriter fw = new FileWriter("Demo.doc");**

**BufferedWriter bw = new BufferedWriter(fw);**

**int ch;**

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

**bw.write(ch);**

**}**

**bw.flush();**

**fr.close();**

**fw.close();**

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

**}**

**}**

**RandomAccessFile : It is type of character class which help to do both type of operation ie read and write.**

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

**class IoDemo {**

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

**RandomAccessFile raf1 = new RandomAccessFile("abc.txt","r");**

**RandomAccessFile raf2 = new RandomAccessFile("xyz.txt","rw");**

**int ch;**

**while((ch=raf1.read())!=-1) {**

**raf2.write(Character.toUpperCase(ch));**

**}**

**raf1.close();**

**raf2.close();**

**}**

**}**

**Object Serialization : Storing the object itself or storing the objects properties or convert object into bytes.**

**object :**

**properties or state or variables**

**behaviour or function / methods**

**identity reference name.**

**FileOutputStream :**

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

**class Employee implements Serializable{**

**private int id;**

**private String name;**

**private transient float salary;**

**static int mgrId;**

**Employee() {}**

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

**this.id = id;**

**this.name = name;**

**this.salary = salary;**

**}**

**public String toString() {**

**return "Id is "+id+" Name is "+name+" Salary is "+salary+" Manager id "+mgrId;**

**}**

**}**

**class IoDemo {**

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

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

**Employee.mgrId=100;**

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

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

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

**oos.writeObject(emp);**

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

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

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

**Employee e = (Employee)ois.readObject();**

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

**}**

**}**

**Day 7 - Core Java - 12-09-2019**

**Collection framework : Collection framework contains set of classes of classes and interfaces which help to store collection elements or objects. Collection framework like a data structure in C/C++.**

**int a=10;**

**a=20;**

**array**

**int abc[];**

**class :**

**class Employee {**

**int id;**

**String name;**

**int age;**

**float salay;**

**}**

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

**array object**

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

**Object obj[]= new Object[100];**

**util package :**

**Collection -----> interface**

**extends extends doesn't extends**

**Set List Queue Map --->**

**It doesn't It allow FIFO key-value pairs**

**allow duplicate. duplicate. base upon the Where key is unique**

**Some classes Maintain the priority and value**

**maintain order order using may be duplicate**

**or unorder or index position**

**sorted default.**

**HashSet Stack PriorityQueue HashMap**

**LinkedHashSet ArrayList LinkedHashMap**

**TreeSet LinkedList TreeMap**

**Vector Hashtable**

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

**class A{**

**public String toString() {**

**return "A class object";**

**}**

**}**

**class B{}**

**public class CollectionDemo {**

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

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

**int a=10;**

**Integer b = new Integer(a);**

**hs.add(b);**

**hs.add(new Integer(20));**

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

**hs.add(30); //auto-boxing : converting primitive to objects of Object class**

**hs.add(10.10);**

**hs.add(true);**

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

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

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

**}**

**}**

**HashSet class Example**

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

**public class CollectionDemo {**

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

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

**System.out.println("Size "+hs.size());**

**System.out.println("Empty "+hs.isEmpty());**

**hs.add(30);**

**hs.add(10);**

**hs.add(20);**

**hs.add(10);**

**hs.add(50);**

**System.out.println("Size "+hs.size());**

**System.out.println("Empty "+hs.isEmpty());**

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

**System.out.println("Search "+hs.contains(20));**

**System.out.println("Search "+hs.contains(200));**

**System.out.println("Remove "+hs.remove(10));**

**System.out.println("Remove "+hs.remove(100));**

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

**hs.clear();**

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

**System.out.println("Size "+hs.size());**

**System.out.println("Empty "+hs.isEmpty());**

**}**

**}**

**LinkedHashSet : It is sub class of HashSet class and maintain the order of elements or objects. Where HashSet stored in unorder.**

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

**public class CollectionDemo {**

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

**LinkedHashSet hs = new LinkedHashSet();**

**System.out.println("Size "+hs.size());**

**System.out.println("Empty "+hs.isEmpty());**

**hs.add(30);**

**hs.add(10);**

**hs.add(20);**

**hs.add(10);**

**hs.add(50);**

**System.out.println("Size "+hs.size());**

**System.out.println("Empty "+hs.isEmpty());**

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

**System.out.println("Search "+hs.contains(20));**

**System.out.println("Search "+hs.contains(200));**

**System.out.println("Remove "+hs.remove(10));**

**System.out.println("Remove "+hs.remove(100));**

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

**hs.clear();**

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

**System.out.println("Size "+hs.size());**

**System.out.println("Empty "+hs.isEmpty());**

**}**

**}**

**TreeSet :**

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

**public class CollectionDemo {**

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

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

**System.out.println("Size "+hs.size());**

**System.out.println("Empty "+hs.isEmpty());**

**hs.add(30);**

**hs.add(40);**

**hs.add(20);**

**hs.add(10);**

**hs.add(50);**

**System.out.println("Size "+hs.size());**

**System.out.println("Empty "+hs.isEmpty());**

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

**System.out.println("Search "+hs.contains(20));**

**System.out.println("Search "+hs.contains(200));**

**System.out.println("Remove "+hs.remove(10));**

**System.out.println("Remove "+hs.remove(100));**

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

**hs.clear();**

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

**System.out.println("Size "+hs.size());**

**System.out.println("Empty "+hs.isEmpty());**

**}**

**}**

**Stack : Stack, First in Last Out. FILO or LIFO**

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

**public class CollectionDemo {**

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

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

**ss.push(100);**

**ss.push(200);**

**ss.push(300);**

**ss.push(400);**

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

**System.out.println("Remove "+ss.pop());**

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

**System.out.println("Top Element check "+ss.peek());**

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

**System.out.println("Search "+ss.search(200));**

**System.out.println("Search "+ss.search(3000));**

**}**

**}**

**array :**

**int abc[]={10,20,30,50,60};**

**100**

**ArrayList :**

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

**public 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(2);**

**al.remove(new Integer(30));**

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

**}**

**}**

**LinkedList : In Java LinkedList by default use double linkedList algorithms.**

|  |  |  |
| --- | --- | --- |
| **head** | **100** | **next** |

|  |  |  |
| --- | --- | --- |
| **previous** | **200** | **Tail** |

|  |  |  |
| --- | --- | --- |
| **previous** | **200** | **Tail** |

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

**public 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(2);**

**al.remove(new Integer(30));**

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

**al.addFirst(1000); //al.add(0,1000);**

**al.addLast(2000);**

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

**}**

**}**

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

**public 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(2);**

**al.remove(new Integer(30));**

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

**al.addFirst(1000); //al.add(0,1000);**

**al.addLast(2000);**

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

**}**

**}**

**Vector :Vector class is known as legacy class. Vector class all methods are by default synchronized.**

**Queue :**

**PriorityQueue :**

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

**public class CollectionDemo {**

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

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

**pq.add(3);**

**//pq.add(1);**

**//pq.add(2);**

**pq.add(4);**

**pq.add(5);**

**pq.add(6);**

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

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

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

**}**

**}**

**Map : Key-value**

**key is unique and value may be duplicate.**

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

**public class CollectionDemo {**

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

**Hashtable hm = new Hashtable();**

**hm.put(6,null);**

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

**hm.put("a","Ramesh");**

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

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

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

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

**hm.put(4,"Ravi");**

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

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

**if(hm.containsKey(100)) {**

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

**}else {**

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

**}**

**}**

**}**

**Retrieve the objects from collection of classes.**

**1. for each loop**

**2. Iterator**

**3. ListIterator**

**4. Enumeration : three 2, 3 and 4 are interfaces.**

**Set family :**

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

**public class CollectionDemo {**

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

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

**ss.add(10);**

**ss.add(20);**

**ss.add(30);**

**ss.add(40);**

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

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

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

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

**}**

**System.out.println("Using iterator");**

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

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

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

**}**

**}**

**}**

**List family :**

**ArrayList and LinkedList**

**Iterator as well as ListIterator**

**Iterator only forward direction where ListIterator forward as well as backward direction.**

**Iterator is super interface. ListIterator extends Iterator interface.**

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

**public class CollectionDemo {**

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

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

**ll.add(10);**

**ll.add(20);**

**ll.add(30);**

**ll.add(40);**

**System.out.println("Using listIterator - forward direction");**

**ListIterator li = ll.listIterator(3);**

**/\*while(li.hasNext()) {**

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

**}\*/**

**System.out.println("Using listIterator - backward direction");**

**while(li.hasPrevious()) {**

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

**}**

**}**

**}**

**Vector with Enumeration**

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

**public class CollectionDemo {**

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

**Vector vv = new Vector();**

**vv.add(10);**

**vv.add(20);**

**vv.add(30);**

**vv.add(40);**

**Enumeration en = vv.elements();**

**while(en.hasMoreElements()) {**

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

**}**

**}**

**}**

**Map with Iterator**

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

**public class CollectionDemo {**

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

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

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

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

**hm.put(4,"Raju");**

**Set ss = hm.entrySet(); //converting map to set**

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

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

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

**Map.Entry me = (Map.Entry)ii.next(); //conveting set to map**

**System.out.println(me.getKey()+" "+me.getValue());**

**}**

**}**

**}**

**File ff = new File("C:\\Demo\\abc.txt");**

**Collection Framework with Generics**

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

**public class CollectionDemo {**

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

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

**ll.add(10);**

**ll.add(10.10);**

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

**ll.add(30);**

**Object obj = ll.get(2);**

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

**int n = i.intValue();**

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

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

**//Wrapper classes**

**List<?> ll1 = new ArrayList<>(); //raw type**

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

**ll.add(10); //auto - boxing**

**ll.add(20);**

**ll.add(30);**

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

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

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

**}**

**}**

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

**class Operation<T> {**

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

**}**

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

**}**

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

**}\*/**

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

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

**}**

**void info(List<Integer> ll) {**

**}**

**void infoDetails(List<?> ll) {**

**}**

**void infoDetails1(List<? extends Number> ll) {**

**}**

**void infoDetails2(List<? super String> ll) {**

**}**

**}**

**public class CollectionDemo {**

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

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

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

**op.add(10.10,20.20);**

**op.add("Raj", "Deep");**

**}**

**}**

**Collection Framework with complex objects**

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

**class Employee {**

**private int id;**

**private String name;**

**private float salary;**

**Employee() {}**

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

**this.id = id;**

**this.name = name;**

**this.salary = salary;**

**}**

**public String toString() {**

**return "id "+id+" name "+name+" salary "+salary;**

**}**

**}**

**public class CollectionDemo {**

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

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

**ll.add(new Employee(2,"Ravi",10000));**

**ll.add(new Employee(1,"Ajay",17000));**

**ll.add(new Employee(4,"Mahesh",16000));**

**ll.add(new Employee(3,"Balaji",12000));**

**System.out.println("Number of rec "+ll.size());**

**ll.remove(1);**

**Employee e = ll.get(0);**

**System.out.println("Number of rec "+ll.size());**

**Iterator<Employee> li = ll.iterator();**

**while(li.hasNext()) {**

**Employee emp = li.next();**

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

**}**

**}**

**}**

**Set API**

**TreeSet with Employee object and hashCode and equals methods**

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

**class Employee {**

**private int id;**

**private String name;**

**private float salary;**

**Employee() {}**

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

**this.id = id;**

**this.name = name;**

**this.salary = salary;**

**}**

**public String toString() {**

**return "id "+id+" name "+name+" salary "+salary;**

**}**

**@Override**

**public int hashCode() {**

**return 10+this.id;**

**}**

**@Override**

**public boolean equals(Object obj){**

**Employee emp =(Employee)obj;**

**if(this.id ==emp.id && this.name.equals(emp.name) && this.salary==emp.salary){**

**return true;**

**}else {**

**return false;**

**}**

**}**

**}**

**public class CollectionDemo {**

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

**Set<Employee> ss = new HashSet<>();**

**Employee emp1 = new Employee(100,"Ravi",12000);**

**Employee emp2 = new Employee(100,"Ravi",12000);**

**Employee emp3 = new Employee(101,"Raj",14000);**

**Employee emp4 = new Employee(102,"Ram",16000);**

**Employee emp5 = new Employee(102,"Ram",16000);**

**ss.add(emp1);**

**ss.add(emp2);**

**ss.add(emp3);**

**ss.add(emp4);**

**ss.add(emp5);**

**System.out.println("Emp Records "+ss.size());**

**}**

**}**

**Arrays and Collections : Both of type of static classes which provided set of methods which help to do operation on primitive arrays and collection of classes respectively.**

**public class CollectionDemo {**

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

**int abc[]={2,4,1,6,9,7,5};**

**System.out.println("Before Sort");**

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

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

**}**

**Arrays.sort(abc);**

**System.out.println("After Sort");**

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

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

**}**

**}**

**}**

**Collections.sort() method**

**public class CollectionDemo {**

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

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

**ll.add("P"); ll.add("C"); ll.add("B");ll.add("A"); ll.add("D");**

**System.out.println("Before sort");**

**for(String str : ll) {**

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

**}**

**Collections.sort(ll);**

**System.out.println("After sort");**

**for(String str : ll) {**

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

**}**

**}**

**}**

**Comparable interface by Sort**

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

**class Employee implements Comparable<Employee>{**

**private int id;**

**private String name;**

**private float salary;**

**Employee() {}**

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

**//return this.id-emp.id; //asc by id**

**//return emp.id -this.id; //desc by id**

**//return this.name.compareTo(emp.name); //asc by name**

**//return emp.name.compareTo(this.name); //asc by name**

**//return (int)(this.salary - emp.salary); //asc by salary**

**return (int)(emp.salary -this.salary); //desc by salary**

**}**

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

**this.id = id;**

**this.name = name;**

**this.salary = salary;**

**}**

**public String toString() {**

**return "id "+id+" name "+name+" salary "+salary;**

**}**

**}**

**public class CollectionDemo {**

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

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

**ll.add(new Employee(2,"Ajay",12000));**

**ll.add(new Employee(1,"Vijay",16000));**

**ll.add(new Employee(3,"Mahesh",14000));**

**System.out.println("Before sort");**

**for(Employee emp : ll) {**

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

**}**

**Collections.sort(ll);**

**System.out.println("After sort");**

**for(Employee emp : ll) {**

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

**}**

**}**

**}**

**Collections sort using Comparator**

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

**class SortByIdAsc implements Comparator<Employee> {**

**public int compare(Employee emp1, Employee emp2){**

**return emp1.id - emp2.id;**

**}**

**}**

**class SortByNameAsc implements Comparator<Employee> {**

**public int compare(Employee emp1, Employee emp2){**

**return emp1.name.compareTo(emp2.name);**

**}**

**}**

**class Employee {**

**int id;**

**String name;**

**float salary;**

**Employee() {}**

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

**this.id = id;**

**this.name = name;**

**this.salary = salary;**

**}**

**public String toString() {**

**return "id "+id+" name "+name+" salary "+salary;**

**}**

**}**

**public class CollectionDemo {**

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

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

**ll.add(new Employee(2,"Ajay",12000));**

**ll.add(new Employee(1,"Vijay",16000));**

**ll.add(new Employee(3,"Mahesh",14000));**

**System.out.println("Before sort");**

**for(Employee emp : ll) {**

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

**}**

**//Collections.sort(ll, new SortByIdAsc());**

**Collections.sort(ll, new SortByNameAsc());**

**System.out.println("After sort");**

**for(Employee emp : ll) {**

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

**}**

**}**

**}**

**Day 8 - Core Java - 13-09-2019**

**Java 7 and 8 Features**

**Switch statement**

**case variableName : Where name must be type of int family or char up to Java 6 . but from Java 7 variableName may be type of String.**

**var arguments : If method takes var argument parameter we can pass 0 or 1 or n arguments as a parameter.**

**class Abc {**

**void setValue(int x) {**

**}**

**void passArray(int abc[]) {**

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

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

**}**

**}**

**void varNumber(int...a) {**

**System.out.println("Method called...");**

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

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

**}**

**}**

**void studentInfo(int stdId, String name, int age, int...marks) {**

**}**

**}**

**class VarDemo {**

**public static void main(String...args){**

**int xyz[]={10,20};**

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

**obj.passArray(xyz);**

**obj.varNumber();**

**obj.varNumber(1);**

**obj.varNumber(1,2,3,4);**

**obj.studentInfo(1,"Raj",21);**

**obj.studentInfo(1,"Raj",21,56);**

**obj.studentInfo(1,"Raj",21,56,78);**

**obj.studentInfo(1,"Raj",21,89,90,76,54);**

**}**

**}**

**try block must contains catch or finally block till java 6. But from Java 7 we can write try block without catch and finally block. ie try with resource.**

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

**class VarDemo {**

**public static void main(String...args){**

**FileInputStream fis = null;**

**FileOutputStream fos = null;**

**try {**

**int a = 10/0;**

**fis = new FileInputStream("CarTest.java");**

**fos = new FileOutputStream("CarTest.doc");**

**int ch;**

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

**fos.write(ch);**

**}**

**}catch(Exception e){**

**}finally {**

**try{**

**fis.close();**

**fos.close();**

**}catch(Exception e){**

**}**

**}**

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

**}**

**}**

**try(**

**){**

**}catch(Exception e}{**

**}**

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

**class VarDemo {**

**public static void main(String...args){**

**try(**

**FileInputStream fis = new FileInputStream("CarTest.java");**

**FileOutputStream fos = new FileOutputStream("CarTest.doc");**

**) {**

**int a = 10/0;**

**int ch;**

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

**fos.write(ch);**

**}**

**}catch(Exception e){**

**}**

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

**}**

**}**

**User - Defined resource close**

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

**class Employee implements Closeable{**

**public void close() {**

**System.out.println("Resourc Close");**

**}**

**public void display() {**

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

**}**

**}**

**class VarDemo {**

**public static void main(String...args){**

**try(**

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

**){**

**emp.display();**

**}**

**}**

**}**

**Java 8 Features**

**interfaces : Up to Java 7 interface can contains only public static and final variables and abstract methods.**

**But from Java 8 onwards interface can contains methods with body but the method must be with keywords as default or static.**

**Collections ---->**

**set methods**

**default and static methods in interface**

**interface A {**

**void dis1();**

**default void dis2() {**

**System.out.println("A interface dis2() default implementation");**

**}**

**default void dis3() {**

**System.out.println("A interface dis3()default implementation");**

**}**

**static void dis4() {**

**System.out.println("A interface dis4() static implementation");**

**}**

**}**

**class B implements A {**

**@Override**

**public void dis1() {**

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

**}**

**@Override**

**public void dis3() {**

**System.out.println("dis3 method override...");**

**}**

**//@Override**

**/\*public void dis4() {**

**System.out.println("static B class own mthod");**

**}\*/**

**}**

**class VarDemo {**

**public static void main(String...args){**

**B obj = new B();**

**obj.dis1();**

**obj.dis2();**

**obj.dis3();**

**//obj.dis4();**

**A.dis4();**

**}**

**}**

**One interface can extends another interface and can override the default method implementation.**

**interface A {**

**default void dis1() {**

**System.out.println("A dis1() interface default impelementation");**

**}**

**static void dis2() {**

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

**}**

**}**

**interface B {**

**default void dis1() {**

**System.out.println("B dis1() interface default impelementation");**

**}**

**static void dis2() {**

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

**}**

**}**

**class C implements A,B {**

**public void dis1() {**

**A.super.dis1();**

**System.out.println("A and B dis1() interface override impelementation");**

**}**

**}**

**class VarDemo {**

**public static void main(String...args){**

**C obj = new C();**

**obj.dis1();**

**A.dis2();**

**B.dis2();**

**}**

**}**

**Lambda Expression :**

**Lambda expression is also known as anonymous function or methods.**

**interfaceName interfaceReferenceName =**

**()->Systme.out.println("Welcome");**

**Functional interface : The interface which contains only one abstract methods is known as Functional interface.**

**Runnable interface**

**Comparable**

**ActionListener**

**Closeable**

**@FunctionalInterface**

**interface A {**

**void dis1();**

**static void dis2(){}**

**default void dis3() {}**

**}**

**class B implements A {**

**public void dis1() {**

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

**}**

**}**

**class VarDemo {**

**public static void main(String...args){**

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

**obj1.dis1();**

**A obj2 = new A(){**

**public void dis1() {**

**System.out.println("Anonymou class inner class implementation");**

**}**

**};**

**obj2.dis1();**

**A obj3 = ()->System.out.println("Lambda Expression implementation");**

**obj3.dis1();**

**A obj4 = ()->{**

**System.out.println("One statement");**

**System.out.println("Second statement");**

**};**

**obj4.dis1();**

**}**

**}**

**interface Operation {**

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

**}**

**class VarDemo {**

**public static void main(String...args){**

**Operation op = (a,b)->a+b;**

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

**Operation op1 = (int a, int b)->a+b;**

**System.out.println(op1.add(100,200));**

**Runnable r = ()-> System.out.println("Thread start");**

**Thread t = new Thread(r);**

**t.start();**

**}**

**}**

**Comparator with Lambda Expression**

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

**class Employee {**

**int id;**

**String name;**

**float salary;**

**Employee() {}**

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

**this.id = id;**

**this.name = name;**

**this.salary = salary;**

**}**

**public String toString() {**

**return "id "+id+" name "+name+" salary "+salary;**

**}**

**}**

**public class CollectionDemo {**

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

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

**ll.add(new Employee(2,"Ajay",12000));**

**ll.add(new Employee(1,"Vijay",16000));**

**ll.add(new Employee(3,"Mahesh",14000));**

**System.out.println("Before sort");**

**for(Employee emp : ll) {**

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

**}**

**//Collections.sort(ll, new SortByIdAsc());**

**//Collections.sort(ll, new SortByNameAsc());**

**//Collections.sort(ll,(emp1,emp2)->emp1.id-emp2.id); //By Id**

**//Collections.sort(ll,(emp1,emp2)->emp2.id-emp1.id);**

**//Collections.sort(ll,(emp1,emp2)->emp2.name.compareTo(emp1.name));**

**//Collections.sort(ll,(emp1,emp2)->emp1.name.compareTo(emp2.name));**

**//Collections.sort(ll,(emp1,emp2)->(int)(emp1.salary-emp2.salary));**

**Collections.sort(ll,(emp1,emp2)->(int)(emp2.salary-emp1.salary));**

**System.out.println("After sort");**

**for(Employee emp : ll) {**

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

**}**

**}**

**}**

**Pre-defined functional interfaces.**

**1. java.util.function.Function ----> Function interface**

**public R apply(T) : receive t parameter and return R value.**

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

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

**interface Operation {**

**public int add(int x);**

**}**

**public class DemoTest {**

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

**Operation op = (a)->a+10;**

**System.out.println(op.add(5));**

**Function<Integer,Integer> ff = (a)->a+10;**

**System.out.println(ff.apply(5));**

**}**

**}**

**2. java.util.function.Predicate : It provided test() method which takes T as a parameter and return boolean result.**

**public boolean test(T obj) {**

**}**

**3. java.util.function.Consumer : It provide accept() method which takes T as parameter and no return type.**

**public void accept(T obj) {**

**}**

**4. java.util.function.Supplier : It provide get() method which takes 0 parameter but return T value.**

**public T get(){**

**return value;**

**}**

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

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

**interface Operation {**

**public int add(int x);**

**}**

**public class DemoTest {**

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

**Operation op = (a)->a+10;**

**System.out.println(op.add(5));**

**Function<Integer,Integer> ff = (a)->a+10;**

**System.out.println(ff.apply(5));**

**Predicate<Integer> pp = (a)-> true;**

**System.out.println(pp.test(10));**

**Predicate<Integer> pp1 = (a)-> {**

**if(a>100) {**

**return true;**

**}else {**

**return false;**

**}**

**};**

**System.out.println(pp1.test(10));**

**Consumer<Integer> cc = (a)->System.out.println("Value of consumer is "+a);**

**cc.accept(10);**

**Supplier<String> ss = ()-> "Welcome to Supplier Function";**

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

**}**

**}**

**Limitation of Lambda Expression :**

**Method reference :**

**interface Operation {**

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

**}**

**class OperationImp {**

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

**//coding**

**return x+y;**

**}**

**}**

**class OperationImp1 {**

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

**return x+y;**

**}**

**}**

**public class DemoTest {**

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

**Operation op = (a,b)->a+b;**

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

**Operation op1 = OperationImp :: add;**

**System.out.println(op1.add(100,200));**

**OperationImp1 oi = new OperationImp1();**

**Operation op2 = oi::add;**

**System.out.println(op2.add(100,200));**

**}**

**}**

**Day 9 - Core Java - 14-09-2019**

**Java 8**

**Stream API :**

**java.util.\*;**

**java.util.function.\*;**

**java.util.stream.\*;**

**Collection Framework, all classes and interfaces in collection framework hold collection of elements or objects or values. When we using collection framework all collection of classes has to populate all elements from may file or database.**

**Collection framework is in-memory data structure.**

**Stream doesn't hold data. Stream hold the data for temporary purpose. Using Stream we can't do reusability.**

**For stream source is array or collection of classes. Then using intermediate as well terminal operator or methods we can apply conditions depending upon the requirements.**

**Where intermediate operator return stream itself and terminal operator return specific type of values may be primitive type or no return type.**

**Stream load the data on-demand from the array type or collection of classes base upon the intermediate operators.**

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

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

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

**public class StreamDemo {**

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

**String name="Welcome to String API";**

**Stream <String> ss = Stream.of(name);**

**ss.forEach(data->System.out.println(data));**

**//ss.forEach(data->System.out.println(data));**

**Stream.of(name).forEach(System.out::println);**

**Stream.of(name).forEach(System.out::println);**

**}**

**}**

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

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

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

**public class StreamDemo {**

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

**String name[]={"Ravi","Ramesh","Ajay","Seeta","Mahesh","Lokesh"};**

**Stream.of(name).forEach(System.out::println);**

**Stream.of(name).forEach(data->System.out.println(data));**

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

**Stream.of(name).forEach(data->**

**{**

**if(data.contains("e")) {**

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

**}**

**}**

**);**

**}**

**}**

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

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

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

**public class StreamDemo {**

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

**//String msg="Welcome to Stream API";**

**//Stream <String> ss = Stream.generate(()->msg);**

**//ss.forEach(data->System.out.println(data));**

**Stream.generate(()->new Date()).forEach(data->System.out.println(data));**

**}**

**}**

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

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

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

**public class StreamDemo {**

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

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

**ll.add(40);ll.add(10);ll.add(30);ll.add(90);ll.add(120);ll.add(5);**

**ll.add(50);ll.add(70);ll.add(70);ll.add(70);ll.add(140);ll.add(55);**

**Stream <Integer> ss = ll.stream();**

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

**ll.stream().forEach(data->System.out.println(data));**

**}**

**}**

**terminator operator (Mandatory)**

**forEach()**

**anyMatch()**

**allMatch()**

**noneMatch()**

**count()**

**intermediate operator (Optional )**

**map()**

**filter()**

**sorted()**

**distinct()**

**limit()**

**skip()**

**StreamSource ----> intermediateOperator1 ----> intermediateOperator2--> Terminator operator**

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

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

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

**public class StreamDemo {**

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

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

**for(int i=0;i<100;i++){**

**ll.add(i);**

**}**

**//ll.stream().forEach(System.out::println);**

**ll.parallelStream().forEach(System.out::println);**

**}**

**}**

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

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

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

**public class StreamDemo {**

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

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

**ll.add("Ravi");ll.add("Ramesh"); ll.add("Balaji"); ll.add("Seeta"); ll.add("Meeta");**

**ll.stream().map(data->data.toUpperCase()).forEach(System.out::println);**

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

**ll.stream().filter(data->data.contains("ee")).map(data->data.toUpperCase()).forEach**

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

**System.out.println("Separate Stream Reference");**

**Stream<String> ss1 = ll.stream();**

**Stream<String> ss2 = ss1.filter(data->data.contains("ee"));**

**Stream<String> ss3 = ss2.map(data->data.toUpperCase());**

**ss3.forEach(System.out::println);**

**}**

**}**

**Create Customer class which contains custId, custName, age, Account class as well as Address reference.**

**Account class must be contains accNumber, typeOfAccount, amount.**

**Address class must be contains city and state.**

**Business methods**

**CustomerService**

**1. Create Account :**

**conditions**

**accountNumber must be unique**

**amount must be > 1000**

**age must be > 21**

**Where type of account must be saving or current account.**

**2. Withdraw Amount**

**ask account number then withdraw but maintain min 1000.**

**3. Deposit amount**

**deposit using account number but max 49000 only. ask pan card number.**

**4. checkBalance :**

**check balance using accnonumber**

**5. profileDetails**

**using custId**

**6. transfer amount**

**transaction details business methods**

**n methods**

**history details for deposit and withdraw**

**Java 8 Dates classes**

**Date : Date class is a part of util package. This class is use to find the system date and time.**

**Calendar class : This class is use to do operation on date. Calendar class also part of util package.**

**SimpleDateFormat : This class is use to convert date to string format. This class part of text package.**

**Date : Date class is a part of sql package. This class is use to convert system data or user-defined date into database date data type format.**

**All date classes in Java 8 part of time package. all classes are immutable.**

**class** Abc {

**void** dis() {

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

}

}

**public** **class** OptionDemo {

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

Abc obj = **null**;

Optional<Abc> op1 = Optional.*of*(obj);

System.***out***.println(op1.isPresent());

Optional<Abc> op = Optional.*ofNullable*(obj);

**if**(op.isPresent()) {

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

obj.dis();

}**else** {

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

System.***out***.println("object not created...");

}

}

}

**Day 10 - Database - Using MySQL - 17-09-2019**

**Data : raw fact or unstructured data.**

**Information : meaningful data or process data.**

**Database : It is use to store data in table format or systematic format ie row and columns.**

**DBMS : Database Management System : It is software or tool which help to store the data in table format.**

**RDBMS**

**Programming :**

**Input**

**Process**

**Output**

**File base system :**

**Limitation of File base system .**

**1. We can store same records again and again. ie Data redundancy.**

**2. Data inconsistence. in form of format, txt, doc, pdf, csv, etc**

**emp.txt**

**1Raj12000**

**1 Raj 12000**

**1,Raj, 12000**

**1/Raj/12000**

**3. Data security. only read and write mode.**

**4. CRUD Operation (Create/Insert/ Read Update and Delete)**

**Data base system :**

**1. If we create the database and tables with constraints we can avoid data redundancy.**

**2. All databases normally RDBMS follow same format to store the data ie table format.**

**3. We can achive data security.**

**4. We can achieve CRUD operation very easily using SQL Language.**

**DBMS : Excel is a type of DBMS software.**

**Flat file**

**Database Model**

**Hierarchical model**

**file1**

**file 2 file3**

**file4 file5 file6 file7**

**Network model**

**file5**

**file1 file4**

**file2**

**fil3**

**Relational model**

**Two table or files can connected using common column with the help of PK and FK.**

**1970**

**Dr. EF Codd's.**

**12 Rules start from 0 to 11. If the database follow all 12 rules that database is known as RDBMS.**

**TrainerStudent**

**TID TName Tech SId SName Age**

**1 Raj Java 100 Seeta 23**

**1 Raj Java 101 Meeta 24**

**1 Raj Java 102 Veeta 25**

**2 Ram .net 105 Keeta 26**

**2 Ram .net 106 Teeta 28**

**Trainer ---> Table (Relation)**

**PK**

**Column -->Attribute**

**TId TNam Tech**

**1 Raj Java**

**2 Ram .net Row (Tuple)**

**Student**

**PK FK(TrainerTable PK)**

**SId SName Age TSId**

**100 Seeta 23 1**

**101 Meeta 24 1**

**102 Veeta 25 2**

**103 Keeta 26 2**

**Product Name Company Name**

**Oracle Oracle**

**MySQL Oracle (Sun Microsystem)**

**db2 IBM**

**SQL Server 2017 Microsoft**

**SQL : Structure Query Language**

**5 types**

**1. DRL or DQL : Data Retrieval Language or Data Query Language**

**select clause :**

**2. DDL : Data Definition Language :**

**create, drop, alter, truncate etc.**

**3. DML : Data Manipulation Language**

**insert, delete and update**

**4. TCL : transaction control Language**

**rollback, savepoint and commit DML Operation**

**5. DCL : Data control language**

**grant and revoke**

**MySQL Commands**

**by default userName : root**

**password :**

**show databases : it is use to display all databases present in your account.**

**use databaseName; This command is use to move from one database to another database.**

**show tables: This command is use to display all tables present in current database.**

**Creating new database**

**create database databaseName;**

**Oracle**

**show databases;**

**create database databaseName;**

**use databaseName;**

**show tables;**

**scott**

**tiger**

**select \* from tab;**

**DRL or DQL :**

**select \* from tableName;**

**To view all records from a tables.**

**desc tableName;**

**This command is use to find the table structure.**

**To retrieve only few columns from a table.**

**select columnName, columnName, columnName from tableName;**

**Column alias**

**select columnName as columnAliasName from tableName;**

**FullName, BasicSalary, HRA, DA,PF,GrossSalary**

**HRA ---> 10%salary**

**DA ---> 5%Salary**

**PF --->7%Salary**

**GrossSalary = Salary + HRA + DA - PF**

**Select clause with Where Clause**

**Where clause is use to filter the data**

**1. Relational Operators**

**select \* from tableName where columnName RO value;**

**RO**

**>**

**>=**

**<**

**<=**

**=**

**!= or <>**

**2. between operator : This operator is use to filter the data within a range.**

**select \* from tableName where columnName between minValue and maxValue;**

**3. In operator : This operator is use to filter the data with more than one specific values.**

**select \* from tableName where columnName in(v1,v2,v3);**

**4. like operator :**

**select \* from tableName where columnName like 'Steven'; exact name**

**select \* from tableName where columnName like 'S%'; start with S**

**select \* from tableName where columnName like '%e'; end with e**

**select \* from tableName where columnName like '%e%'; contains e**

**% zero or many**

**select \* from tableName where columnName like '\_t\_v%';**

**select \* from tableName where columnName like '\_t\_\_v%';**

**5. is null :**

**select \* from tableName where columnName is null;**

**Logical Operators**

**and : both or all condition must be true**

**or : any one condition must be true**

**not : negation or reverse of condition.**

**Order by clause : This clause is use to sort the record may be ascending or descending order.**

**select \* from tableName order by colummName asc**

**select \* from tableName order by colummName by default acending**

**select \* from tableName order by colummName desc**

**select \* from tableName where clause order by columnName asc;**

**DDL : Data Definition Language :**

**create :**

**Emp--->**

**id,name,salary**

**syntax**

**create table tableName(columnName dataType, columnName dataType......columnName dataType);**

**DML : Data Manipulation Language**

**Insert, Delete and Update**

**insert into tableName values(v1,v2,v3);**

**insert into tableName values(v1,v2,null);**

**insert into tableName(columnName1,columnName2) value(v1,v2);**

**Update Query :**

**update tableName set columnName = value;**

**update emp set salary = 25000;**

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

**update emp set name ='Raj Deep' where name like 'Raj';**

**update emp set name ='Seeta Kumari' where salary >25000 and id = 3;**

**update emp set name ='Ramesh Patil', salary = 25000 where id =4;**

**Delete Query :**

**delete from tableName; This command is use to delete all records from table but doesn't remove the table from database.**

**delete from tableName where id =1;**

**delete from tableName where name like 'Ravi' and salary > 25000;**

**drop table tableName; This command is use to remove table structure as well as all records from a databases.**

**truncate table tableName; This command is use to delete all records from table but doesn't remove the table from database.**

**select \* from tablename where columnName regexp 'Rules';**

**.**

**?**

**+**

**\*s**

**[abc];**

**Day 11 - Database Using MySQL - 18-09-2019**

**Join : Join is use to retrieve one column or more than one column from one or more than one table with or without conditions.**

**Join also known as De-Normalization.**

**3 tables**

**table1**

**empid, name,salary**

**1 Raj 12000**

**2 Seeta 14000**

**table2**

**srno,name, amount**

**100 Raj 500**

**101 Ravi 1000**

**table3**

**accno,fname, totalAmount**

**111 Raj 5000**

**222 Ramesh 10000**

**alter table tableName change oldName**

**Cartesian Product with table alias**

**mysql> select t1.empid,t1.name,t1.salary,t2.srno,t2.amount,t2.name from table1 t**

**1, table2 t2;**

**Cross Join with table alias**

**mysql> select t1.empid,t1.name,t1.salary,t2.srno,t2.amount,t2.name from table1 t**

**1 cross join table2 t2;**

**Natural Join**

**mysql> select t1.empid,t1.name,t1.salary,t2.srno,t2.amount,t2.name from table1 t**

**1 natural join table2 t2;**

**In Natural join both table must be contains common column name.**

**Equi-Join :**

**mysql> select t1.empid,t1.name,t1.salary,t3.accno,t3.fname,t3.totalAmount from t**

**able1 t1, table3 t3 where t1.name = t3.fname;**

**Inner join**

**mysql> select t1.empid,t1.name,t1.salary,t3.accno,t3.fname,t3.totalAmount from t**

**able1 t1 join table3 t3 on t1.name = t3.fname;**

**left outer join**

**right outer join**

**full join**

**Self join : Joining the same table itself is known as self join.**

**mysql> select e1.first\_name,e1.job\_id,e2.first\_name,e2.job\_id from employees e1,**

**employees e2 where e1.manager\_id = e2.employee\_id;**

**EmployeeName, job\_title, Manager\_name, job\_title**

**mysql> select e1.first\_name,j1.job\_title,e2.first\_name,j2.job\_title from employees e1,employees e2, jobs j1, jobs j2**

**where e1.manager\_id = e2.employee\_id and**

**j1.job\_id=e1.job\_id and j2.job\_id=e2.job\_id;**

**My SQL Functions :**

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

**In Database always function takes one or more parameter and return value mandatory.**

**function mainly divided into two types**

**1. pre-defined functions**

**divided into two types**

**a. single row functions**

**string functions**

**maths functions**

**number function**

**date function**

**conversation function**

**b. multiple row functions**

**sum(), max(), min(), avg() and count() :**

**select \* from dual ; oracle database**

**mysql>**

**select manager\_id, count(\*) from employees where manager\_id is not null group by manager\_id having count(\*) >5 order by manager\_id desc;**

**2. user-defined function (PL SQL)**

**Day 12 - Database Using MySQL - 19-09-2019**

**Sub Query :**

**Query within another query is known as Sub Query**

**Syntax**

**Outer Query (Inner Query)**

**Outer Query (Inner Query (Inner Inner Query))**

**Find the employee name whose salary is > avg salary of all employee working in a department 10.**

**select avg(salary) from employees where department\_id=10;**

**select first\_name from employees where salary > result;**

**Outer Query (Inner Query)**

**Rules**

**1. Sub query must be return only one column**

**2. Some sub query return more than result or only one result.**

**base up on this conditions**

**sub query divided into two types**

**a. Single row sub query**

**b. Multiple row sub query**

**Single row sub query**

**outer Query where columnName RO (Inner Query)**

**Multiple row sub query**

**outer query where columnName**

**in (Inner Query )**

**RO any (Inner Query)**

**RO all (Inner Query )**

**max --> 9000**

**min ----> 4200**

**any**

**mysql> select first\_name,salary from employees where salary >any (select salary from employees where department\_id=60) order by salary asc;**

**or**

**mysql> select first\_name,salary from employees where salary > (select min(salary) from employees where department\_id=60) order by salary asc;**

**all**

**mysql> select first\_name,salary from employees where salary >all (select salary from employees where department\_id=60) order by salary asc;**

**or**

**mysql> select first\_name,salary from employees where salary > (select max(salary) from employees where department\_id=60) order by salary asc;**

**Co-related sub query**

**Outer Query (Inner Query)**

**select e1.first\_name,e1.salary from employees e1 where 0 =(select count(e2.salary) from employees e2 where e2.salary > e1.salary);**

**9000 ---->**

**9000 1**

**10000 2**

**7000 3**

**12000 4**

**5000 5**

**1st Scan**

**2 records**

**2nd Scan**

**1 records**

**Keys :**

**Table : Organization**

**EmpId,FName,LName,Age,PhNumber, addId, City,State,AccnoNumber, typeOfAccount, ProjectId, projectName, cId, companyName etc.**

**EmpId PK**

**AccnoNumber unique**

**PhNumber unique**

**cId unique**

**EmpId,FName**

**AccnoNumber,TypeOfAccount**

**cId, companyName**

**EmpId,FName,PhNumber**

**EmpId,FName,PhNumber**

**EmpId,FName**

**EmpId**

**EmpId,PhNumber**

**PhNumber**

**etc**

**Constraints : constraints is use to restrict the use or programmer to insert the invalid data in tables.**

**1. not null : not null constraints doesn't allow null value. In Single table we can create more than one column as not null constraints.**

**not null constraints always we can to create on column level.**

**2. unique : unique constraints doesn't allow duplicate records but can allow null values. In Single table we can create more than one column as unique constraints. Unique constraints we support column level as well as table level.**

**3. default : default constraints hold the default value if user doesn't insert any records. default value can override. In Single table we can create more than one column as default constraints. Default constraints support only column level.**

**4. check : check constraints is use to make user-defined conditions while inserting the records.**

**5. enum : enum is a type of data type which allow to store fixed values.**

**create table emp5(empid int primary key,**

**name varchar(10) not null,**

**salary float check (salary>25000),**

**phNumber int unique,**

**gender enum('male','female'),**

**company varchar(10) default 'onMobile');**

**not null constraints after table created**

**alter table emp6 modify name varchar(10) not null;**

**drop primary key constraints**

**alter table tableName drop primary key**

**drop unique key constraints**

**alter table tableName drop index columnName(This column which contains unique constraints)**

**composite primary key:**

**info**

**PK**

**cid pid price**

**A X 100**

**A Y 200**

**B X 300**

**B Y 400**

**A X 500 Error**

**E-R diagram**

**Relationship**

**1-1**

**1-may**

**many-1**

**many-many**

**PK-FK**

**ACID properties**

**TCL commit, rollback and savepoint**

**Normalization**

**1,2,3 etc**

**View**

**Index**

**SQL**

**PLSQL**

**stored procedure**

**functions**

**triggers**

**if statements.**

**Day 13 - Database Using MySQL - 20-09-2019**

**primary key and foreign key**

**one - to - many :**

**Trainer --**

**PK**

**TId TName Tech**

**1 Raj Java**

**2 Raju .net**

**3 Ramu HTML**

**Student**

**PK FK**

**SId SName Age TSId**

**100 Seeta 22 1**

**101 Reeta 23 1**

**102 Meeta 24 2**

**103 Veeta 25 2**

**104 Teeta 26 null**

**create table trainer(tid int,**

**tname varchar(10) not null,**

**tech varchar(10) not null,**

**primary key(tid));**

**create table student(sid int,**

**sname varchar(10),**

**age int,**

**tsid int,**

**primary key(sid),**

**foreign key(tsid) references trainer(tid));**

**Drop foreign key**

**alter table student drop foreign key student\_ibfk\_1;**

**alter table student add foreign key(tsid) references trainer(tid)**

**on delete cascade;**

**on delete cascade;**

**on update cascade;**

**on delete set null**

**on update set null**

**alter table student drop foreign key student\_ibfk\_1;**

**alter table student add constraint ts\_fk foreign key(tsid) references trainer(tid) on update cascade;**

**one - to - one**

**Person :**

**pid pk**

**pname**

**create table person(pid int, pname varchar(10), primary key(pid));**

**Passport :**

**ptid pk**

**passorddesc**

**create table passport(psid int, pdesc varchar(10),**

**primary key(psid),**

**foreign key(psid) references person(pid));**

**many to many**

**Students**

**SId PK**

**SName**

**SS\_SId FK**

**SkillSets**

**SSId PK**

**tech**

**S\_SSId FK**

**using two tables**

**create table students(sid int, sname varchar(10), ss\_s\_id int,**

**primary key(sid));**

**create table skillset(ssid int, tech varchar(10), s\_ss\_id int,**

**primary key(ssid));**

**alter table students add constraint s\_ss\_fk**

**foreign key(ss\_s\_id) references skillset(ssid);**

**alter table skillset add constraint ss\_s\_fk**

**foreign key(s\_ss\_id) references students(sid);**

**using three tables (join tables)**

**Students : Sid(PK), SName**

**SkillSet :SSId(PK, tech**

**Student\_SkillSet : SS\_SID(FK) and S\_SSId(FK)**

**TCL : Transactional Control Language :**

**commit, rollback and savepoint**

**ACID Properties :**

**DML : insert, Delete, update**

**View : view is database object, which also known as virtual table which refer to base table.**

**create view viewName**

**as**

**select clause;**

**create view emp1**

**as**

**select \* from emp**

**create view emp2**

**as**

**select id,name from emp**

**create view emp3**

**as**

**select with join with more table.**

**select \* from emp3;**

**create view emp\_view as select \* from emp4 where salary > 25000;**

**select \* from emp\_view;**

**create view emp\_view as select \* from emp4 where salary > 25000 with check option;**

**create view emp\_view as select \* from emp4 with read only;**

**Day 14 - Database using MySQL - 21-09-2019**

**PL SQL : Procedure Language on SQL or Programming Language on SQL.**

**Oracle Database**

**They provided two type of blocks**

**anonymous block**

**named block**

**procedure or stored procedure (trigger)**

**functions**

**declare**

**begin**

**SQL and Non SQL Statements**

**exception**

**end;**

**stored procedure or procedure : Stored procedure or procedure which is a type of named block which contains set of SQL and Non SQL statement with or without conditions.**

**Oracle**

**begin**

**dbms\_output.put\_line('Welcome to PL SQL in Oracle');**

**end;**

**My SQL**

**delimiter $**

**create procedure abc() select "Welcome to Pl SQL Using MySQL";**

**call abc;**

**$**

**create procedure xyz() select \* from emp;**

**call xyz; $**

**alter procedure xyz() select \* from emp;**

**drop procedure xyz;**

**create or replace procedure abc()**

**as**

**begin**

**end;**

**Stored procedure parameters**

**3 types of parameters mode**

**1. in parameter : receive or read**

**2. out parameter : return back or write**

**3. inout parameter receive and return /read and write.**

**PL SQL**

**function : function is a type of named block which can take more than one in parameter and return one output mandatory using return keywords.**

**Trigger : Trigger is type of special stored procedure which will get fire or call automatically when we do any DML Operation on specific table.**

**DML**

**Before After**

**Insert**

**Delete**

**Update**

**Java JDBC 2 days**

**Day 15 - JDBC with Java - 23-09-2019**

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

**sql packages**

**1. java.sql.\*; javax.sql.\*;**

**2. JDBC always throw checked exception ie SQLException So we have to write JDBC code using Exception handling concept ie try-catch or throws.**

**3. Class.forName("driverClassName ")**

**Class.forName("com.Employee");**

**Driver : Driver is software.**

**Driver is a pre-defined class provided by vendor in the form of a jar file which help to connect the database through Java Applications.**

**Types of Driver**

**1. JDBC ODBC Bridge Driver till jdk 7**

**OS dependent.**

**2. JDBC native API Driver**

**Database dependent.**

**3. JDBC net protocol Driver**

**Application Server :WebLogic, JBoss, Glashfish etc**

**DataSource :**

**Enterprise Application .**

**4. JDBC thin driver or JDBC Pure driver :**

**We have to download the jar files.**

**jar file dependent or driver dependent.**

**Class.forName("driverName");**

**4. Establish the connection .**

**DriverManager is a pre-defined class part of sql package which contains getConnection() methods. Which takes three parameter 1st parameter is URL**

**2nd parameter is UserName**

**3rd parameter is password**

**DriverManager.getConnection(url,username,password);**

**getConnection() method return type is Connection interfaces references.**

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

**5. Create type of statements.**

**3 types**

**a. Statement**

**b. PreparedStatement**

**c. CallableStatement**

**a. Statement: Statement is a interface which provided set of methods which help to do operation on table through Java program.**

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

**i. DML Operation**

**int res = stmt.executeUpdate("DML Operation");**

**ii. Select clause : If we want to retrieve the records from database. When we have to call the methods as**

**stmt.executeQuery("select clause"); This method return type is ResultSet interface reference.**

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

**PreparedStatement :**

**Limitation of Statement : Using Statement reference each time when we run the Java program query will**

**a. query compile,**

**b. send to database**

**c. execute and**

**d. get the acknowledgement.**

**In PreparedStatement query compile only one time and execute n number of times.**

**Statement doesn't support parameterized query concept.**

**PreparedStatement support parameterized query concept.**

**CallableStatement : CallableStatement is use to execute Pl SQL Stored procedure and functions through Java using JDBC.**

**Day 16 - JDBC with Java - 24-09-2019**

**JDBC Transaction Management System**

**By default in JDBC all DML Operation are auto-committed.**

**TCL :**

**Connection con = DriverManager.getConnection(url,userName,password);**

**con.setAutoCommit(false); By default true**

**Do one or mote DML Operation using Statement or PreparedStatement**

**con.commit() or con.rollback();**

**Limitation of JDBC where Connection reference hold the transaction details.**

**DatabaseMetadata : It is a interface which provide the details of Database.**

**Layer architecture**

**Table -----> Emp (id,name,salary)**

**JavaBean ---->Emp(id,name,salary) CRUD Operation**

**Main Method -----> Normal classes**

**Controller create**

**delete**

**View update**

**Console retrieve**

**Service Dao (Data Access Object) Resources**

**Pure business pure database database**

**logic logic connection**

**Hibernate 3 Days**

**Day 17 - ORM Using Hibernate - 25-09-2019**

**Limitation of JDBC**

**DAO Layer (Data Access Object)**

**1. Java is OOPs, Database (RDBMS ) , when storing the records using JDBC and retrieving the records from JDBC we convert java object to query format string or vice-versa.**

**2. JDBC throw checked exception.**

**3. JDBC exception hierarchy or exception classes are vendor dependents.**

**4. JDBC use SQL query language where SQL is database dependent.**

**5. JDBC doesn't support is a and has a relationship**

**6. JDBC doesn't support collection framework like set, list and map.**

**ORM : Object Relation Mapping :**

**Object Relation**

**OOPs Programming Language RDBMS --->Table**

**mapping**

**class Employee { EMPLOYEE**

**id,name,salary ID,NAME,SALARY**

**}**

**Employee emp = new Employee()**

**emp.setId(100);**

**emp.setName("Ravi"); 1,Ravi,12000**

**emp.setSalary(12000);**

**Employee(CLASS) --- EMPLOYEEDETAILS (TABLE)**

**id(PK) EMP\_ID(PK)**

**name NAME**

**salary SALARY**

**using XML file or using annotation**

**Implementation of ORM**

**JPA Java Persistence API (Application Programming Langauge)**

**Hibernate**

**nHibernate**

**EntityFramework**

**iBatis or MyBatis**

**JDO Java Data Object**

**etc**

**JDBI : Hexaware :**

**Framework : Framework contains set of classes and interfaces which internally connected to each other to perform a specific task. Framework also known as Template or protocol but not final product. Framework is implementation of design pattern.**

**Hibernate : Hibernate is a framework. It is open source provided by JBoss.**

**Hibernate is implementation of ORM.**

**Hibernate Version 2.x 5.x**

**Table**

**class Employee EMPLOYEE**

**id,name,salary ID(pk),NAME,SALARY**

**mapping file or annotation**

**Employee.hbm.xml**

**Employee ---EMPLOYEE**

**id ID PK**

**name NAME**

**salary SALARY**

**Database Information**

**1. using hibernate configuration file (xml file)**

**2. using hibernate properites files (property file)**

**3. using java configuration**

**database information : driverName, url, userName, password**

**dialet class :**

**hibernate.cfg.xml**

**database details**

**+**

**mapping file and entity class details**

**+**

**hibernate properties (optional)**

**sql hql**

**Structure query language Hibernate Query Language**

**Data base dependent Database Independent.**

**HQL is like SQL to OOPs.**

**select \* from employee select e from Employee e**

**from Employee e**

**from Employee**

**select \* from employee select e from Employee e**

**where salary > 12000 where e.salary > 12000**

**from Employee e where**

**e.salary > 12000**

**select id from employee select e.id from Employee e**

**select name,salary from employee select e.name,e.salary from Employee e**

**Day 18 - ORM Using Hibernate - 26-09-2019**

**Hibernate Day 2 :**

**Hibernate Relationship :**

**2 types**

**1. has a relationship**

**a. collection has a relationship**

**b. association has a relationship**

**i. one - to - one**

**ii. one - to - many**

**iii. many - to - one**

**iv. many - to - many**

**c. component has a relationship**

**2. is a relationship : inheritance :**

**class Employee {**

**private int id;**

**String name;**

**float salary;**

**}**

**class Student {**

**int sid;**

**String sname;**

**int marks[];**

**}**

**Student**

**PK**

**SId Name**

**1 Raj**

**StudentMarks**

**PK**

**SS Id Marks**

**1 0 67**

**1 1 89**

**1 2 56**

**1 3 76**

**class Employee {**

**int empId;**

**String name;**

**List<String> skillSet;**

**}**

**class Projects {**

**int pid;**

**String typeOfProject;**

**List<Employee> listOfEmp;**

**}**

**class Employee {**

**int empId;**

**String name;**

**}**

**Trainer**

**tid PK**

**tname**

**tech**

**Student**

**sid PK**

**sname**

**age**

**tsid FK**

**Students**

**Courses**

**Employees**

**Projects or SkillSet or**

**class CourseEmployeeDetails {**

**private int cid;**

**private int sid;**

**}**

**@OneToMany(mappedBy=""**

**List<Courses> listOfcours**

**component has a relationship**

**Employee**

**Id Name Salary City State**

**class Employee {**

**id,name,salary ,Address**

**}**

**class Address {**

**city, state**

**}**

**Day 19 - ORM Using Hibernate - 27-09-2019**

**Hibernate Day 3 :**

**Hibernate is a relationship (Inheritance)**

**3 types**

**1. Single Table inheritance hierarchy**

**2. Per class tables inheritance join**

**3. Concrete sub class table inheritance : union**

**id,name,salary,numberofEmp,projectName,typeofEmp**

**1,Raj,12000,null,java,Programmer**

**2,Seeta,45000,10,null,Manager**

**class Employee {**

**id,name,salary**

**}**

**class Manager extends Employee {**

**numberOfEmp**

**}**

**class Programmer extends Employee {**

**projectName**

**}**

**CompanyProjectInfo**

**PK**

**cmpId projectId price**

**1 100 15000**

**2 100 20000**

**1 101 25000**

**2 101 35000**

**@Entity not possible**

**class CompanyProjectInfo {**

**@Id**

**private int cmpid;**

**@Id**

**private int projectid;**

**private float price;**

**}**

**create table CompanyProjectInfo(cmpId int primary key,projectId int primary key,price float);**

**not possible**

**1: Clerk Login/Manager Login hibernate**

**userId and password**

**1: check account pending request**

**a. approve**

**b. deny**

**2: check all customer details**

**2: Customer Login**

**userId and password**

**Customer Details**

**1: Create**

**2:withdraw**

**3: deposit**

**4: transfer**

**5: transaction details**

**3. new User**

**Criteria API : Criteria API is use to retrieve the records from table without using any query language like HQL or SQL as developer point of you. HQL is use to select and DML(Delete/Update).**

**But criteria API only for Select operations.**

**By default using Criteria API we can retrieve whole objects. But using Criteria with Projection we can retrieve partial objects.**

**Hibernate Caching :**

**Caching is a mechanism which help to improve the performance. Database caching, browser caching, java caching, hibernate caching etc.**

**Hibernate Caching it is mainly divided into 2 types**

**1. First level caching**

**First level caching belong to session objects. By default Hibernate enable First level caching. First level caching scope is session object.**

**2. Second level caching**

**Second level caching belong to SessionFactory. By default hibernate doesn't enable Second level caching. We have to enable using XML or annotation. We have to use third party jar files to enable second level caching. Second level caching also known global caching because it belong to SessionFactory API.**

**ehcache-core-2.6.10**

**Day 20 - ORM Using JPA - 30-09-2019**

**JPA Day 1 :**

**JPA : Java Persistence API (Application Programming Interface).**

**JPA is a part of Java people. JPA is a type of EJB (Enterprise Java Bean).**

**JEE (Java Enterprise Edition). Servlet, JSP and EJB. JPA implements ORM concept. JPA Vs Hibernate**

**1. JPA is a specification as well as implementation of ORM**

**Where Hibernate is a implementation of JPA.**

**2. JPA is a technologies where Hibernate is a Framework.**

**3. JPA is a type of EJB and part of Java People. where Hibernate is Framework part of JBoss.**

**EJB + JPA = 30%, 20%**

**Spring Framework + Hibernate = 80%, 90%**

**JPA**

**iBaties**

**JDO**

**Spring boot 2.x + JPA**

**Spring 1.5 + JPA/Hibernate**

**Spring boot with Spring Data**

**JPA 1.x and 2.x**

**ORM**

**Object Relation**

**Employee Entity EMPLOYEE --table**

**id,name,salary id,name,salary ----> columns**

**mapping**

**persistence.xml file META-INF folder**

**JTA (Java Transaction API)**

**Database Details**

**driverName,**

**url**

**username**

**password**

**+**

**mapping file details**

**+**

**JPA properties (optional)**

**JPA jar files manually**

**using Maven tool and Gradle tools**

**CRUD Operation :**

**Select clause :**

**sql**

**select \* from employee;**

**hql**

**select e from Employee e**

**from Employee e**

**from Employee**

**jpql**

**select e from Employee e**

**Day 21 - ORM Using JPA - 01-10-2019**

**JPA Day 2 :**

**JPA Relationship**

**2 types**

**1. has a relationship**

**a. collection has a relationship**

**b. association has a relationship**

**i. one - to - one**

**ii. one - to - many**

**iii. many - to - one**

**iv. many - to - many**

**c. component has a relationship**

**2. is a relationship : inheritance :**

**JPA is a relationship (Inheritance)**

**3 types**

**1. Single Table inheritance hierarchy**

**2. Per class tables inheritance join**

**3. Concrete sub class table inheritance : union**

**@Entity**

**class Projects{**

**@Id**

**int pid;**

**String typeOfProjects**

**@OneToMany**

**@JoinColumn(name="peid")**

**List<Employee> listOfEmp;**

**}**

**@Entity**

**class Employee {**

**@Id**

**int empId;**

**String name;**

**int peid; FK (Optional)**

**}**

**Storing the all projects and employee records independently**

**Projects.java**

**package com;**

**import java.util.List;**

**import javax.persistence.Entity;**

**import javax.persistence.ForeignKey;**

**import javax.persistence.Id;**

**import javax.persistence.JoinColumn;**

**import javax.persistence.OneToMany;**

**@Entity**

**public class Projects {**

**@Id**

**private int pid;**

**private String typeOfProject;**

**@OneToMany // Hibernate @OneToMany(mappedBy="tsid")**

**@JoinColumn(name = "peid") // Refer to the FK**

**private List<Employees> listOfEmp;**

**public int getPid() {**

**return pid;**

**}**

**public void setPid(int pid) {**

**this.pid = pid;**

**}**

**public String getTypeOfProject() {**

**return typeOfProject;**

**}**

**public void setTypeOfProject(String typeOfProject) {**

**this.typeOfProject = typeOfProject;**

**}**

**public List<Employees> getListOfEmp() {**

**return listOfEmp;**

**}**

**public void setListOfEmp(List<Employees> listOfEmp) {**

**this.listOfEmp = listOfEmp;**

**}**

**}**

**Employee.java**

**package com;**

**import javax.persistence.Entity;**

**import javax.persistence.ForeignKey;**

**import javax.persistence.Id;**

**@Entity**

**public class Employees {**

**@Id**

**private int empid;**

**private String empname;**

**private float salary;**

**public float getSalary() {**

**return salary;**

**}**

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

**this.salary = salary;**

**}**

**private int peid; //FK**

**public int getPeid() {**

**return peid;**

**}**

**public void setPeid(int peid) {**

**this.peid = peid;**

**}**

**public int getEmpid() {**

**return empid;**

**}**

**public void setEmpid(int empid) {**

**this.empid = empid;**

**}**

**public String getEmpname() {**

**return empname;**

**}**

**public void setEmpname(String empname) {**

**this.empname = empname;**

**}**

**}**

**DemoTest.java**

**package com;**

**import javax.persistence.EntityManager;**

**import javax.persistence.EntityManagerFactory;**

**import javax.persistence.EntityTransaction;**

**import javax.persistence.Persistence;**

**public class DemoTest {**

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

**// TODO Auto-generated method stub**

**EntityManagerFactory emf = Persistence.createEntityManagerFactory("OneToManyJPARelationship");**

**EntityManager manager = emf.createEntityManager();**

**EntityTransaction tran = manager.getTransaction();**

**Employees emp1 = new Employees();**

**emp1.setEmpid(1);emp1.setEmpname("Ravi"); emp1.setSalary(12000);**

**emp1.setPeid(100);**

**Employees emp2 = new Employees();**

**emp2.setEmpid(2);emp2.setEmpname("Raj"); emp2.setSalary(14000);**

**emp2.setPeid(100);**

**Projects pp = new Projects();**

**pp.setPid(100);**

**pp.setTypeOfProject("Java");**

**tran.begin();**

**manager.persist(pp);**

**manager.persist(emp1);**

**manager.persist(emp2);**

**tran.commit();**

**System.out.println("Stored successfully...");**

**}**

**}**

**Storing the all projects and employee details with project object with Cascade property ALL**

**package** com;

**import** java.util.List;

**import** javax.persistence.CascadeType;

**import** javax.persistence.Entity;

**import** javax.persistence.ForeignKey;

**import** javax.persistence.Id;

**import** javax.persistence.JoinColumn;

**import** javax.persistence.OneToMany;

@Entity

**public** **class** Projects {

@Id

**private** **int** pid;

**private** String typeOfProject;

@OneToMany(cascade = CascadeType.***ALL***) // Hibernate @OneToMany(mappedBy="tsid")

@JoinColumn(name = "peid") // Refer to the FK

**private** List<Employees> listOfEmp;

**public** **int** getPid() {

**return** pid;

}

**public** **void** setPid(**int** pid) {

**this**.pid = pid;

}

**public** String getTypeOfProject() {

**return** typeOfProject;

}

**public** **void** setTypeOfProject(String typeOfProject) {

**this**.typeOfProject = typeOfProject;

}

**public** List<Employees> getListOfEmp() {

**return** listOfEmp;

}

**public** **void** setListOfEmp(List<Employees> listOfEmp) {

**this**.listOfEmp = listOfEmp;

}

}

**Employees.java**

**package com;**

**import javax.persistence.Entity;**

**import javax.persistence.ForeignKey;**

**import javax.persistence.Id;**

**@Entity**

**public class Employees {**

**@Id**

**private int empid;**

**private String empname;**

**private float salary;**

**public float getSalary() {**

**return salary;**

**}**

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

**this.salary = salary;**

**}**

**private int peid; //FK**

**public int getPeid() {**

**return peid;**

**}**

**public void setPeid(int peid) {**

**this.peid = peid;**

**}**

**public int getEmpid() {**

**return empid;**

**}**

**public void setEmpid(int empid) {**

**this.empid = empid;**

**}**

**public String getEmpname() {**

**return empname;**

**}**

**public void setEmpname(String empname) {**

**this.empname = empname;**

**}**

**}**

**DemoTest.java**

**package com;**

**import java.util.ArrayList;**

**import java.util.List;**

**import javax.persistence.EntityManager;**

**import javax.persistence.EntityManagerFactory;**

**import javax.persistence.EntityTransaction;**

**import javax.persistence.Persistence;**

**public class DemoTest {**

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

**// TODO Auto-generated method stub**

**EntityManagerFactory emf = Persistence.createEntityManagerFactory("OneToManyJPARelationship");**

**EntityManager manager = emf.createEntityManager();**

**EntityTransaction tran = manager.getTransaction();**

**Employees emp1 = new Employees();**

**emp1.setEmpid(3);emp1.setEmpname("Ramesh"); emp1.setSalary(16000);**

**emp1.setPeid(101);**

**Employees emp2 = new Employees();**

**emp2.setEmpid(4);emp2.setEmpname("Ram"); emp2.setSalary(18000);**

**emp2.setPeid(101);**

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

**listOfEmp.add(emp1); listOfEmp.add(emp2);**

**Projects pp = new Projects();**

**pp.setPid(101);**

**pp.setTypeOfProject("Angular");**

**pp.setListOfEmp(listOfEmp);**

**tran.begin();**

**manager.persist(pp);**

**//manager.persist(emp1);**

**//manager.persist(emp2);**

**tran.commit();**

**System.out.println("Stored successfully...");**

**}**

**}**

**<property name="eclipselink.ddl-generation" value="create-tables"/>**

**<property name="eclipselink.ddl-generation" value="create-or-extends-tables"/>**

**<property name="eclipselink.ddl-generation" value="create-drop"/>**

**one to many with Auto increment project id and not setting foreign key for the employees**

**Projects.java**

**package** com;

**import** java.util.List;

**import** javax.persistence.CascadeType;

**import** javax.persistence.Entity;

**import** javax.persistence.ForeignKey;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.JoinColumn;

**import** javax.persistence.OneToMany;

@Entity

**public** **class** Projects {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** pid;

**private** String typeOfProject;

@OneToMany(cascade = CascadeType.***ALL***) // Hibernate @OneToMany(mappedBy="tsid")

@JoinColumn(name = "peid") // Refer to the FK

**private** List<Employees> listOfEmp;

**public** **int** getPid() {

**return** pid;

}

**public** **void** setPid(**int** pid) {

**this**.pid = pid;

}

**public** String getTypeOfProject() {

**return** typeOfProject;

}

**public** **void** setTypeOfProject(String typeOfProject) {

**this**.typeOfProject = typeOfProject;

}

**public** List<Employees> getListOfEmp() {

**return** listOfEmp;

}

**public** **void** setListOfEmp(List<Employees> listOfEmp) {

**this**.listOfEmp = listOfEmp;

}

}

**Employees.java**

**package com;**

**import javax.persistence.Entity;**

**import javax.persistence.ForeignKey;**

**import javax.persistence.Id;**

**@Entity**

**public class Employees {**

**@Id**

**private int empid;**

**private String empname;**

**private float salary;**

**public float getSalary() {**

**return salary;**

**}**

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

**this.salary = salary;**

**}**

**/\*private int peid; //FK**

**public int getPeid() {**

**return peid;**

**}**

**public void setPeid(int peid) {**

**this.peid = peid;**

**}\*/**

**public int getEmpid() {**

**return empid;**

**}**

**public void setEmpid(int empid) {**

**this.empid = empid;**

**}**

**public String getEmpname() {**

**return empname;**

**}**

**public void setEmpname(String empname) {**

**this.empname = empname;**

**}**

**}**

**DemoTest.java**

**package com;**

**import java.util.ArrayList;**

**import java.util.List;**

**import javax.persistence.EntityManager;**

**import javax.persistence.EntityManagerFactory;**

**import javax.persistence.EntityTransaction;**

**import javax.persistence.Persistence;**

**public class DemoTest {**

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

**// TODO Auto-generated method stub**

**EntityManagerFactory emf = Persistence.createEntityManagerFactory("OneToManyJPARelationship");**

**EntityManager manager = emf.createEntityManager();**

**EntityTransaction tran = manager.getTransaction();**

**Employees emp1 = new Employees();**

**emp1.setEmpid(1);emp1.setEmpname("Raj"); emp1.setSalary(16000);**

**//emp1.setPeid(101);**

**Employees emp2 = new Employees();**

**emp2.setEmpid(2);emp2.setEmpname("Ram"); emp2.setSalary(18000);**

**//emp2.setPeid(101);**

**Employees emp3 = new Employees();**

**emp3.setEmpid(3);emp3.setEmpname("Ramesh"); emp3.setSalary(12000);**

**//emp3.setPeid(101);**

**Employees emp4 = new Employees();**

**emp4.setEmpid(4);emp4.setEmpname("Ravi"); emp4.setSalary(18000);**

**//emp4.setPeid(101);**

**List<Employees> listOfEmp1 = new ArrayList<Employees>();**

**listOfEmp1.add(emp1); listOfEmp1.add(emp2);**

**List<Employees> listOfEmp2 = new ArrayList<Employees>();**

**listOfEmp2.add(emp3); listOfEmp2.add(emp4);**

**Projects pp1 = new Projects();**

**pp1.setTypeOfProject("Angular");**

**pp1.setListOfEmp(listOfEmp1);**

**Projects pp2 = new Projects();**

**pp2.setTypeOfProject("Java");**

**pp2.setListOfEmp(listOfEmp2);**

**tran.begin();**

**manager.persist(pp1);**

**manager.persist(pp2);**

**//manager.persist(emp1);**

**//manager.persist(emp2);**

**tran.commit();**

**System.out.println("Stored successfully...");**

**}**

**}**