**Java Notes :**

**Day 1**

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

Java

J2SE J2EE J2ME

JavaSE JavaEE JavaME

JSE JEE JME

Java Standard Java Enterprise Java Micro

Edition Edition Edition

Core Java

Basic Programming

OOPs

Exception handling

Packages

Lang, io, util, awt, swing, jdbc, rmi etc

Java Software :

Version of Java 1.0, 1.1,1.2,1.4,1.5, 1.6, 1.7,Java 8 , 9,10, 11, 12

JDK : Java Development Kit

JRE : Java Run time Environment :

**Java Virtual Machine**

Class syntax

class className {

variables

methods /functions

}

class Test {

public static void main(String args[]) {

System.out.println(“Welcome to Java”);

}

}

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

2 types

1. Primitive data types
   1. byte 1 byte
   2. short 2
   3. int 4
   4. long 8
   5. float 4
   6. double 8
   7. boolean 1 bit
   8. char 2 byte
2. Non primitive data types or reference data types.

**Types Casting :**Converting from one data type to another data types

2 types

1. Implicit type casting
2. Explicit type casting

-------🡪implicit type casting -----🡪

byte short int long

1 2 4 8

🡨--------explicit type casting ---------

class Test {

public static void main(String args[]) {

byte a = 10;

short b =a; //i

System.out.println(a);

System.out.println(b);

short c = 127;

byte d = (byte)c; //E

System.out.println(c);

System.out.println(d);

}

}

int float

4 4

class Test {

public static void main(String args[]) {

int a=10;

float b = a; //Im

System.out.println(a);

System.out.println(b);

float c = 10.0f;

int d =(int)c;

System.out.println(c);

System.out.println(d);

}

}

**Reference Data types** : It is use to store value as well as reference of another data types.

4 types

1. array
2. class
3. interface
4. enum

**array :** array is use to store the same type of values.

int abc[10]; C/C++

int abc[];

int []abc;

int [] abc;

int[] abc;

looping :

1. while loop
2. do while loop
3. for loop
4. for each loop

class Test {

public static void main(String args[]) {

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

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

System.out.println(abc[5]);

System.out.println(abc.length);

}

}

while loop

initialization start and end

condition

body of the loop

class Test {

public static void main(String args[]) {

int i=0,n=10;

while(i<=n) {

System.out.println(i);

i++;

}

}

}

class Test {

public static void main(String args[]) {

int i=0,n=10;

do{

System.out.println(i);

i++;

}while(i<=n);

}

}

class Test {

public static void main(String args[]) {

int i,n;

for(i=0,n=10;i<=n;i++){

System.out.println(i);

}

}

}

For each loop

Syntax

for(dataType variableName:arrayName) {

}

class Test {

public static void main(String args[]) {

int []abc={10,20,30,40,50,60,70};

System.out.println("Normal 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);

}

int b;

//System.out.println(b);

}

}

**Creating memory size to array**

dataType []arrayName=new dataType[size];

int abc[]=new int[10];

class Test {

public static void main(String args[]) {

int abc1[]={10,20,30,40,50};

int []abc=new int[10];

System.out.println(abc.length);

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

System.out.println(abc[9]);

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

abc[i]=100+i;

}

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

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

}

}

}

**Taking value through keyboards**

1. using Scanner class
2. using DataInputStream class
3. using BufferedReader class
4. command line arguments.

Scanner is a pre-defined class which provided set of methods which help to scan the value through keyboards. Syntax to create the Scanner class object.

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

Scanner class is a part of **util** package.

**import java.util.Scanner;**

**or**

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

nextByte()

nextShort()

nextInt();

nextLong()

nextBoolean()

nextFloat()

nextDouble()

nextChar() no methods

next() or nextLine() to receive String value.

import java.util.Scanner;

class Test {

public static void main(String args[]) {

Scanner obj = new Scanner(System.in);

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

int id = obj.nextInt();

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

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

String name = obj.nextLine();

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

float salary = obj.nextFloat();

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

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

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

}

}

**OOPs**

**object and class**

object : any real world entity

state or properties,have,**variables/fields**

Person

Behavior , do/does , **function /methods**

Bank

Animal

Car

Employee

**class :** class is a blueprint of object or template of objet or user-defined data types which help to describe the objects.

class Car {

int wheel;

String color;

float price;

void start() {

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

}

void appliedGear() {

}

void moving() {

}

void stop() {

}

}

class CarTest {

public static void main(String args[]) {

System.out.println("Main Method");

Car santro =new Car();

santro.start();

}

}

**Types of variables /fields**

3 types

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

Employee

id,name,salary -🡪instance variables

Scanner object

read()

calSalary()

hra 10%, da 5% and pf 7% local variables

display()

id,name,salary(grossSalary)

EmployeeTest

Main method

Employee object create

Call read, calSalary, 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 = 0.07f\*salary;

salary = salary+hra+da-pf;

}

void display(){

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

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

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

}

}

class EmployeeTest {

public static void main(String args[]){

Employee emp1 = new Employee();

emp1.read();

emp1.calSalary();

emp1.display();

Employee emp2 = new Employee();

emp2.read();

emp2.display();

}

}

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

**Pts**

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

class Employee {

Employee() {

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

}

void display(){

System.out.println("method");

}

}

class EmployeeTest {

public static void main(String args[]){

Employee emp1 = new Employee();

emp1.display();

emp1.display();

emp1.display();

}

}

class Employee {

int id;

Employee() {

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

}

void display(){

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

}

}

class EmployeeTest {

public static void main(String args[]){

Employee emp1,emp2,emp3;

new Employee();

new Employee().display();

emp1 = new Employee();

emp1.id=100;

emp2 = new Employee();

emp2.id=200;

emp3 = emp1;

emp1.display(); emp2.display(); emp3.display();

emp1=null;

//emp1.display();

emp2.display(); emp3.display();

emp3 = null;

}

}

**Parameterized constructor with this keyword**

class Employee {

int id;

String name;

Employee() {

id = 123;

name="Unknown";

}

Employee(int id, String name) {

this.id =id;

this.name = name;

}

void setValue(int id, String name) {

this.id = id;

this.name = name;

}

void display() {

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

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

}

}

class EmployeeTest {

public static void main(String args[]){

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

Employee emp2 = new Employee(1,"Ravi"); emp2.display();

Employee emp3 = new Employee();

emp3.setValue(2,"Ajay");

emp3.setValue(3,"Seeta");

emp3.display();

}

}

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

Ex : class

**JavaBean class:**

**Normal class Vs JavaBean class**

class Person {

id,name,salary

}

public class Person {

private int id;

private String name;

private float salary;

//setter and getter

public void setId(int id) {

this.id = id;

}

public int getId() {

return id;

}

}

**Day 2**

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

class OldClass { super class, base class

properties parent class

behavior

}

class NewClass extends OldClass {

properties sub class, derived class,

behavior 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 InheritanceDemo {

public static void main(String args[]) {

//A obj1 = new A();

//obj1.dis1();

B obj2 = new B();

obj2.dis2();

obj2.dis1();

}

}

**Types of Inheritance**

1. Single Inheritance

class A {}

class B extends A {}

1. Multilevel Inheritance

class A {}

class B extends A{}

class C extends B {}

class D extends C {}

1. Hierarchical Inheritance

class A {}

class B extends A {}

class C extends A {}

1. Multiple Inheritance

class A {}

class B {}

class C extends A,B {} wrong in Java.

Java doesn’t support multiple inheritance. It support using **interface**.

**OOPs Relationship**

1. Is a relationship
2. Has a relationship

Manager **is a** Employee

Employee **has a** Address

class Employee {

id,name,salary

Address add=new Address();

}

class Manager extends Employee{

}

class Developer extends Employee{

}

class Address {

city, state

}

Has a relationship

1. Association
2. Aggregation
3. Composition

class A {

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

}

class B {

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

}

class Manager {

Address add =new Address();

}

class Developer {

Address add = new Address();

}

class Address {

}

class Student {

StudentHistory sh = new StudentHistory()

}

class StudentHistory {

}

class Employee {

id,name,salary

Scanner obj =new Scanner();

Address add = new Address();

read(), calSalary(),display()

}

class Manager extends Employee{

numberOfEmp;

readMgr()

add.readAdd();

disMgr()

add.disAdd();

}

class Programmer extends Employee {

projectName;

readPgr()

disPgr()

}

class Address {

city

state

Scanner obj =new Scanner(System.in);

readAdd()

disAdd()

}

**Inheritance Example with is a and has a relationship**

import java.util.Scanner;

class Employee {

int id;

String name;

float salary;

Scanner obj = new Scanner(System.in);

Address add = new Address();

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 = 0.07f\*salary;

salary = salary+hra+da-pf;

}

void display(){

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

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

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

}

}

class Manager extends Employee {

int numberOfEmp;

void readMgr() {

read();

System.out.println("Enter the number of Emp");

numberOfEmp = obj.nextInt();

add.readAdd();

}

void disMgr() {

display();

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

add.disAdd();

}

}

class Programmer extends Employee {

String projectName;

void readPgr() {

read();

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

projectName = obj.next();

add.readAdd();

}

void disPgr() {

display();

System.out.println("Project Name is "+projectName);

add.disAdd();

}

}

class Address {

String city;

String state;

Scanner obj = new Scanner(System.in);

void readAdd() {

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

city = obj.next();

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

state = obj.next();

}

void disAdd() {

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

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

}

}

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.display();\*/

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

Manager mgr = new Manager();

mgr.readMgr();

mgr.disMgr();

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

Programmer prg = new Programmer();

prg.readPgr();

prg.disPgr();

}

}

**Polymorphism** One name many forms or many implementations.

2 types

1. Compile time polymorphism
   1. Method Overloading (static binding or early binding)
2. Run time polymorphism
   1. Method Overriding (dynamic binding or late binding)

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

class Operation {

void add(int a, int b) {

System.out.println(a+b);

}

void add(int a, int b,int c) {

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

}

void add(String a, String b) {

System.out.println(a+b);

}

}

class PolyDemo {

public static void main(String args[]){

Operation op = new Operation();

op.add(10,20);

op.add(1,2,3);

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).

To achieve method overriding more than one class with inheritance is require.

class Bike {

void speed() {

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

}

}

class Pulsar extends Bike {

void speed() {

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

}

void color() {

System.out.println("Black");

}

}

class Honda extends Bike {

void color() {

System.out.println("Red");

}

}

class Tvs extends Bike {

void speed() {

super.speed(); //merge super class coding

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

}

void color() {

System.out.println("White");

}

}

class TvsSuper extends Tvs {

void speed() {

super.speed(); //merge super class coding

System.out.println("100km/hr");

}

void color() {

System.out.println("White");

}

}

class PolyDemo {

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();

TvsSuper ts =new TvsSuper(); ts.speed();

}

}

**Annotation :** annotation is a meta-data. Meta-data means data about data.

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

Some annotation we can use on class level, method level, property level.

int a;

**@Override :**

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("Black");

}

}

class PolyDemo {

public static void main(String args[]){

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

}

}

**abstract :** abstract is a keyword we can use with class and method 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 return type methodName(parameterList);

abstract void speed();

1. Abstract class: if class contains abstract method then we have to declare the class as a abstract class.
2. Abstract class can contains normal as well as abstract methods.(0, 1 or all).
3. Which ever class extends abstract class that class must be provide the body for all abstract method belong to that class. That class can ignore only it that class itself is abstract class.
4. Abstract class we can’t create the object.
5. Abstract class can contains default constructor as well as we can write parameterized constructor.

abstract class Bike {

abstract void speed();

}

abstract class Pulsar extends Bike {

void color() {

System.out.println("Black");

}

}

class PolyDemo {

public static void main(String args[]){

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

}

}

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

**Final variables :** to declare the constant value in java we use final keyword with variables.

**final int A=10;**

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

**final class:** If class is final we can’t extends that class.

class Employee {

final int id;

final String name;

Employee() {

id =100;

name="Ravi";

}

void dis() {

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

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

}

}

class PolyDemo {

public static void main(String args[]){

Employee emp1 = new Employee();

emp1.dis();

}

}

class Employee {

final int id;

final String name;

Employee() {

id =100;

name="Ravi";

}

void setInfo(final int abc) {

System.out.println(abc);

//abc=100;

System.out.println(abc);

}

void dis() {

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

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

}

}

class PolyDemo {

public static void main(String args[]){

Employee emp1 = new Employee();

emp1.dis();

emp1.setInfo(123);

}

}

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

class Outer {

static int a;

static void dis() {}

static class Inner {

}

}

class Abc {

int a;

static int b;

void dis1() {

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

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

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

}

static void dis2() {

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

Abc obj= new Abc();

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

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

}

}

class StaticDemo {

public static void main(String args[]) {

Abc obj1 = new Abc();

obj1.dis1();

Abc.dis2();

obj1.a=100;

Abc.b=200;

obj1.dis2();

obj1.b=300;

}

}

**Static concept**

class Abc {

int a;

static int b;

void dis1() {

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

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

}

static Abc obj;

static void dis2() {

obj = new Abc();

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

}

}

class StaticDemo {

public static void main(String args[]) {

Abc.dis2();

/\*Abc obj1 = new Abc();

Abc obj2 = new Abc();

obj1.a=10;

obj1.b=20;

Abc.b=30;

obj2.a=40;

obj2.b=50;

Abc.b=60;

obj1.dis1(); obj2.dis1();\*/

}

}

**Interface :** Interface is a type of reference data types which is use to store constants variables and abstract methods. Interface also known as 100% pure abstract class.

Syntax

interface interfaceName {

variables;

methods;

}

By default all variables inside a interface are **public** **static** and **final**

By default all methods inside a interface are **public** and **abstract**

interface Abc {

int A=10;

void dis();

}

interface Xyz {

int B=20;

void dis1();

}

interface Mno extends Abc,Xyz{

int C=40;

void dis2();

}

class Demo **implements** Abc, Xyz {

}

**Super Sub**

(Class/Abstract class

Interface)

public public

protected public

protected

default(nothing) public

protected

default

private Can’t Override

interface Abc {

int A=10;

void dis();

}

interface Xyz {

int B=20;

void dis1();

}

interface Mno extends Abc,Xyz{

int C=40;

void dis2();

}

class Demo implements Abc, Xyz {

public void dis() {

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

}

public void dis1() {

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

}

}

class InterfaceDemo {

public static void main(String args[]) {

Demo obj =new Demo();

obj.dis();

obj.dis1();

}

}

**Difference between abstract class and interfaces**

1. Interface contains only final variables but abstract class can contains normal as well as final variables.
2. Interfaces contains only abstract methods but abstract can contains normal as well as abstract methods.
3. Interface doesn’t contains default constructor as well as we can’t write parameterized constructor but abstract class can contains default as well as parameterized constructor
4. Any class(normal/abstract) can extends only one abstract/normal class but can implements more than one interfaces.

**Runtime polymorphism using classes**

Creating sub class object and super class reference.

1st Example

class A {

void dis1() {

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

}

}

class B extends A {

void dis1() {

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

}

void dis2() {

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

}

}

class InterfaceDemo {

public static void main(String args[]) {

A obj1 = new A(); obj1.dis1(); //obj1.dis2();

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

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

//B obj4 = new A();

}

}

2nd example

Sub class object and super class reference (where super class is abstract class)

abstract class A {

abstract void dis1();

}

class B extends A {

void dis1() {

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

}

void dis2() {

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

}

}

class InterfaceDemo {

public static void main(String args[]) {

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

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

//B obj4 = new A();

}

}

3rd Example

Sub class object and interface reference.

interface A {

void dis1();

}

class B implements A {

public void dis1() {

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

}

void dis2() {

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

}

}

class InterfaceDemo {

public static void main(String args[]) {

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

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

//B obj4 = new A();

}

}

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

interface ATeam {

int add(int a, int b);

}

interface BTeam {

int sub(int a, int b);

}

class Server implements ATeam,BTeam{

public int add(int a, int b){

return a+b;

}

public int sub(int a, int b){

return a-b;

}

public void ownMethod() {

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

}

}

class InterfaceDemo {

public static void main(String args[]) {

Server ss = new Server();

ATeam tt1 = new Server();

System.out.println(tt1.add(10,20));

//System.out.println(tt1.sub(10,20));

BTeam tt2 = new Server();

//System.out.println(tt2.add(10,20));

System.out.println(tt2.sub(10,20));

}

}

**this, super, this() and super()**

class A {

int n=10;

}

class B extends A {

int n=20;

void dis1() {

int n = 30;

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

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

System.out.println("super n "+super.n);

}

}

class InterfaceDemo {

public static void main(String args[]) {

B obj = new B();

obj.dis1();

}

}

**Constructor chaining using this()**

class A {

A() {

this(10);

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

}

A(int x) {

this(1,2);

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

}

A(int x, int y) {

System.out.println("A(x,y)");

}

}

class InterfaceDemo {

public static void main(String args[]) {

A obj1 = new A();

}

}

<https://github.com/Kaleakash/ciscocorejavatraining.git>

**Day 3**

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

2 types

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

Package creating syntax

package com;

Education

school college

**Attendance edance**

**Access specifiers :**

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

**Pre-defined packages**

**java javax--🡪root package**

**lang swing**

**io net**

**util rmi**

**sql sql**

**awt servlet**

**net ejb**

**rmi jms**

**etc etc**

**Exception Handling :**

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

Java

Compile time error Run time error

Syntax error

Or typo error

**Run time Error**

**Error Exception**

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

**Error :**The error which generated at run time which we can’t handle ex :JVM crash, Out of Memory, Software/Hardware issue.

**Exception** : It is a type of error which generated at the rum time which we can handle it. Aritmetic Exception (Divided by Zero)

**Exception**

Checked Exception UnCheckedException

**SQLException RunTimeException**

**IOException ArithmeticException**

NumberFormatExcepiton

NullPointerException

ArrayIndexOutOfBoundsException

To handle both type of exception in Java

They provided 5 keywords.

1. Try
2. Catch
3. Finally
4. Throw
5. Throws

Syntax for try- catch block

try {

}catch(Exception e) {

}

class ExpDemo {

public static void main(String abc[]){

System.out.println("Hi");

int a =10;

int b =0;

int res,res1;

int num[]={1,2,3,4};

try{

res = a/b;

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

res1=10/num[1];

System.out.println(res1);

}catch(Exception e){

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

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

}

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

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

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

}

}

try with multiple catch block

try {

}

catch(ArimethicException e) {

}catch(ArrayIndexOutOfBoundsException e) {

} catch(Exception e) {

}

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

class ExpDemo {

public static void main(String abc[]){

System.out.println("Hi");

int a =10;

int b =1;

int res;

try{

res = a/b;

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

}catch(Exception e){

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

}finally {

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

}

System.out.println("normal statement");

}

}

try

catch catch catch catch finally

catch finally catch

finally

file handling

try {

open

read/write

}catch(Exception e) {

}finally {

close

}

class ExpDemo {

public static void main(String abc[]){

System.out.println("Hi");

int a =10;

int b =0;

int res;

try{

res = a/b;

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

}finally {

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

}

System.out.println("normal statement");

}

}

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

Syntax

throw new Exception()

or

throw new ExceptionSubClass()

**Custom Exception Example**

class MyException extends Exception {

MyException() {}

MyException(String msg) {

super(msg);

}

}

class ExpDemo {

public static void main(String abc[]){

System.out.println("Hi");

int a=10;

int b=5;

try {

if(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;

}

}catch(Exception e){

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

}

}

}

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

void dis() throws Exception, ExceptionSubClasses {

}

class ExpDemo {

static void dis1() throws Exception{

int a=10/0;

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

}

static void dis2() throws Exception{

//try{

dis1();

//}catch(Exception e){}

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

}

public static void main(String abc[]) {

dis2();

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

}

}

class ExpDemo {

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

System.out.println("Hi");

//try{

Thread.sleep(5000);

//}catch(Exception e){}

System.out.println("Take Lunch break....");

}

}

**Multithreading :**

Programs or Task : Set of instruction of perform a specific task.

Processor :

Process and Thread :

Process :time taken to execute the code.

Thread : Small execution of a code within process.

Thread t = Thread.currentThread();

class ThreadDemo {

public static void main(String args[]) {

Thread t = Thread.currentThread();

System.out.println(t);

t.setName("My Thread");

t.setPriority(8); //min 1, max 10, norm 5

System.out.println(t);

t.setPriority(Thread.MAX\_PRIORITY);

System.out.println(t);

}

}

Creating user-defined thread

2 ways

1. extends Thread class
2. implements Runnable interface

Creating Thread using Thread class

class A extends Thread{

@Override

public void run() {

Thread t = Thread.currentThread();

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

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

}

}

}

class B extends Thread{

@Override

public void run() {

Thread t = Thread.currentThread();

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

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

}

}

}

class ThreadDemo {

public static void main(String args[]) {

A obj1 = new A(); //object created

B obj2 = new B();

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

obj2.start();

}

}

javap java.packageName.className/interfaces

class A implements Runnable{

@Override

public void run() {

Thread t = Thread.currentThread();

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

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

}

}

}

class B implements Runnable{

@Override

public void run() {

Thread t = Thread.currentThread();

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

System.out.println(t+"="+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(); // ready to run Runnable state

t2.start();

}

}

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

**synchronized**

synchronized(objectReference) {

}

class Booking implements Runnable {

int avl=1;

public synchronized void run() {

Thread t = Thread.currentThread();

String name = t.getName();

if(avl==1) {

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

avl=avl-1;

}else {

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

}

}

}

class ThreadDemo {

public static void main(String args[]) {

Booking bb = new Booking();

Thread t1 = new Thread(bb);

Thread t2 = new Thread(bb);

Thread t3 = new Thread(bb);

t1.setName("Raj");

t2.setName("Seeta");

t3.setName("Ajay");

t1.start();

t2.start();

t3.start();

}

}

**lang packages**

String

StringBuffer

StringBuilder

Object

Math

Runnable

Thread

Cloneable

**String class :**

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

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

class LangDemo {

public static void main(String args[]) {

/\*String str1 = "Ravi Kumar";

String str2 = new String("Ravi Kumar");

System.out.println(str1);

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

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

System.out.println(str1.indexOf('a'));

System.out.println(str1.substring(3));

System.out.println(str1.substring(3,8));\*/

String name1 = "Raj";

String name2 = "Raj";

String name3 = new String("Raj");

String name4 = new String("Raj");

if(name4.equals(name3)) {

System.out.println("Equal");

}else {

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

}

}

}

class LangDemo {

public static void main(String args[]) {

String str = "Raj";

System.out.println(str);

str = str + "Deep";

System.out.println(str);

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

str.lowerCase();

System.out.println(str);

StringBuffer sb = new StringBuffer("Raj");

System.out.println(sb);

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

sb.reverse();

System.out.println(sb);

}

}

**Object class**

By default every class extends Object class.

class Emp {

private int id;

private String name;

private float salary;

Emp(){}

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

this.id = id;

this.name = name;

this.salary = salary;

}

@Override

public String toString(){

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

}

@Override

public boolean equals(Object obj){

Emp e = (Emp)obj;

if(e.salary==this.salary && e.name.equals(this.name)){

return true;

}else {

return false;

}

}

}

class LangDemo {

public static void main(String args[]) {

Emp e1 = new Emp(1,"Ravi",12000);

Emp e2 = new Emp(2,"Ravi",12000);

System.out.println(e1);

System.out.println(e2);

if(e1.equals(e2)) {

System.out.println("Equals");

}else {

System.out.println("Not Equals");

}

Emp emp1 = new Emp();

Emp emp2 = new Emp();

Emp emp3 = emp1;

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

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

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

System.out.println(emp1);

}

}

**Clone Demo**

class Emp implements Cloneable{

private int id;

private String name;

private float salary;

Emp(){}

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

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

this.id = id;

this.name = name;

this.salary = salary;

}

public void setSalary(float salary) {

this.salary = salary;

}

public String toString() {

return "id="+id+"name="+name+"salary="+salary;

}

public Emp getCloneInstance() throws Exception{

Object obj = clone();

Emp e= (Emp)obj;

return e;

}

}

class LangDemo {

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

Emp e1 = new Emp(1,"Raj",12000);

Emp e2 = new Emp(2,"Ravi",14000);

Emp e3 = e1;

System.out.println(e1);

System.out.println(e2);

System.out.println(e3);

Emp e4 = e3.getCloneInstance();

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

System.out.println(e4);

e3.setSalary(18000);

System.out.println(e4);

System.out.println(e3);

System.out.println(e3.hashCode());

System.out.println(e4.hashCode());

}

}

**Day 4**

**File Handling: IO Packages**

IO means Input/Output.

**stream** : flow of data or it is a abstraction between source and destination/target.

Stream

byte char

Input Output Input Output

**InputStream OutputStream Reader Writer**

**4 classes are abstract classes.**

**DataInputStream/DataOutputStream**

**FileInputStream/FileOutputStream**

**BufferedInputStream/BufferedOutputStream**

**ObjectInputStream/ObjectOutputStream**

**PrintStream**

**Char**

**Reader Writer**

**InputStreamReader OutputStreamWriter**

**FileReader FileWriter**

**BufferedReader BufferedWriter**

**PrintWriter**

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

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

**PrintStream ps = System.out;**

**ps.println(“Welcome”);**

**byte**

**Source -🡪keyboards/keyboards /file /file**

**Target 🡪Console/file /console /file**

1st Example

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);

}

}

2nd Example

import java.io.\*;

class IODemo {

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

DataInputStream dis = new DataInputStream(System.in);

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

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

int ch;

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

fos.write(ch);

System.out.print(ch);

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

}

fos.close();

}

}

**3rd Example**

import java.io.\*;

class IODemo {

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

FileInputStream fis = new FileInputStream("StaticDemo.java");

FileOutputStream fos = new FileOutputStream("Demo.doc");

int ch;

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

fos.write(ch);

//System.out.print(ch);

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

}

fis.close();

fos.close();

}

}

**4th Example**

import java.io.\*;

class IODemo {

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

FileInputStream fis = new FileInputStream("StaticDemo.java");

BufferedInputStream bis = new BufferedInputStream(fis);

FileOutputStream fos = new FileOutputStream("Demo.doc");

BufferedOutputStream bos = new BufferedOutputStream(fos);

int ch;

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

bos.write(ch);

//System.out.print(ch);

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

}

bos.flush();

fis.close();

fos.close();

}

}

**Object Serialization : Storing the object state itself in a external file.**

**When we want to pass the object from network environment it must in serialize format.**

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

**emp.dispay();**

import java.io.\*;

class Employee implements Serializable{

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 is = "+id+" name is "+name+" salary is "+salary;

}

}

class IODemo {

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

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

System.out.println(emp1);

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

ObjectOutputStream bos = new ObjectOutputStream(fos);

bos.writeObject(emp1);

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

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

ObjectInputStream ois = new ObjectInputStream(fis);

Employee emp2= (Employee)ois.readObject();

System.out.println(emp2);

}

}

**Collection Framework (Data Structure)**

Collection of contains set of collection of classes and interfaces which help to store the collection of object or elements. It provided set of methods with the help of those methods we can add, remove, search, modify the elements.

int a;

int []abc;

**class Emp {**

**id,name,salary**

**}**

**Emp emp =new Emp();**

**Array object**

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

**e[0]=new Emp(); e[0].id=10;**

**to**

**e[99]=new Emp();**

**Collection Framework hierarchy(util package)**

**Collection --🡪 interface**

**extends extends doesn’t extends**

**Set List Queue Map 🡪interfaces**

**Set : doesn’t allow duplicate. Some element are order, unorder or sorted.**

**HashSet : unorder**

**LinkedHashSet: order**

**TreeSet: sorted**

**List : Allow duplicate and maintain the order.**

**Stack :FILO**

**ArrayList : retrieve records again and again**

**LinkedList : update and delete operation again and again**

**Vector : by default all methods are synchronized.**

**Queue : First in first out using priority**

**PriorityQueue : FIFO using priority**

**Map :It help to store key-value pairs where key is unique and value may be duplicate.**

**HashMap : unorder**

**LinkedHashMap : order**

**TreeMap : sorted using key**

**Hashtable : synchronized**

**1st Example**

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

**class Employee{}**

**class CollectionDemo {**

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

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

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

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

**hs.add(10);**

**hs.add(10.10);**

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

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

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

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

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

**hs.remove(10);**

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

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

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

**}**

**}**

**2nd Example**

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

**class Employee{}**

**class CollectionDemo {**

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

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

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

**ll.add(10.10);**

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

**ll.add(20);**

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

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

**int n = i.intValue();**

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

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

**ll.add(10);**

**ll.add(20);**

**ll.add(30);**

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

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

**}**

**}**

**JDBC:** Java Database Connectivity.

JDBC is a API(Application Programming Interfaces) which provided set of classes and interfaces which help to connect the database using Java application.

Connect MySQL Database using username and password

create database mydb;

use mydb;

show tables;

show databases;

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

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

delete from employee where id =1;

update employee set salary = salary +400 where id =2;

select \* from employee

**Steps to connect the database through Java using JDBC.**

1. **Import the package** 
   1. **java.sql or javax.sql**
2. **Write the method (main or user defined with exception handling concept).**
3. **Load the Driver. Driver is a pre-defined class provided by vendor which help to connect the database.**

**Class.forName(“driverName”);**

1. **Establish the connection.**

**DriverManager is a pre-defined class which provided getConnection() method which takes 3 parameter, url, username and password. This method return type is Connection interface reference.**

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

1. **Statement or PreparedStatement : Both are interfaces provided set of methods which help to do CRUD Operation (Create, Read, Update and Delete).**

**JDBC Application**

**package** com;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.Statement;;

**public** **class** DemoTest {

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

**try** {

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

System.***out***.println("Driver Loaded successfully...");

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

System.***out***.println("Connected successfully..");

Statement stmt = con.createStatement();

//Select Query

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

while(rs.next()) {

int id = rs.getInt(1);

String name= rs.getString(2);

float salary = rs.getFloat(3);

System.out.println(" Id is "+id +" Name is "+name+" Salary is "+salary);

}

rs.close();

\*/

//Insert Query

/\*int temp = stmt.executeUpdate("insert into employee values(3,'Ajay',16000)");

if(temp>0) {

System.out.println("Record inserted successfully....");

}\*/

//Delete Query

/\*int temp = stmt.executeUpdate("delete from employee where id =1");

if(temp>0) {

System.out.println("Record deteled successfully");

}else {

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

}\*/

//Update Query

**int** temp = stmt.executeUpdate("update employee set salary = salary+200 where id=2");

**if**(temp>0) {

System.***out***.println("Record update successfully");

}**else** {

System.***out***.println("Record not present");

}

stmt.close();

con.close();

} **catch** (Exception e) {

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

}

}

}

**Statement or PreparedStatement :**

**Statement reference each time query get compile on java side, then send database and execute and get the acknowledgement on java side.**

**PreparedStatement: Here query compile only one time and execute n number of times.**

//Insert Record through PreparedStatment using Scanner

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

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

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

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

pstmt.setInt(1, id);

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

String name = sc.next();

pstmt.setString(2, name);

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

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

pstmt.setFloat(3, salary);

**int** temp = pstmt.executeUpdate();

**if**(temp>0) {

System.***out***.println("Record inserted successfully");

}

**ORM : Object Relation Mapping**

**Hibernate / JPA**

**JDBC**

**Hibernate**

**JPA**

**Spring Data**

**Spring AOP**

**Day 5**

**Java with JSON :**

**Java**

**JSE JEE : Java Enterprise Edition**

**Servlet/JSP/EJB**

**Java Server Pages**

**Enterprise Java Bean**

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

**Client Server**

**HTML/HTML5**

**Hypertext Markup Language**

**CSS/CSS3**

**Cascading Style Sheet**

**JavaScript JEE**

**Asp.net**

**Python**

**Php**

**Node JS**

**MVC : Model View Controller**

**View 🡪 HTML/HTML5/CSS/CSS3/JavScript/JSP**

**Controller -🡪 Servlet**

**Model -🡪 EJB**

**JavaBean/Service/DAO Layer**

**Req Java Req**

**Client HDFC XML/JSON HSBC**

**Res JEE .net Res Asp.net**

**Web Service : Giving the Service for web application when two application running using different technologies . Web Service is platform independent, language independent and browser independent.**

1. **SOAP Web Service : only XML**

**WSDL file :Web Service description language**

1. **RestFull Web Service : XML/JSON as well as any other format.**

**WADL file : Web application description language**

**View 🡪HTML/HTML5/CSS/CSS3/JavaScript/TypeScript/Angular/React/Vue JS**

**Controller -🡪Servlet (RestController)**

**Model 🡪 Spring Framework**

**JSON : JavaScript Object Notation : JSON is a text based platform independent format of data which help to share the data between two technologies.**

**Json hold the data in the form of key-value pairs. Where key must be string and value may be number, Boolean, date, string, array etc.**

**Employee emp = new Employee(1,”Raj”,12000);**

**XML format**

**<Employee>**

**<id>100</id>**

**<name>Raj</name>**

**<salary>12000</salary>**

**</Employee>**

**Json format**

**{“id”:100,”name”:”Raj”,”salary”:12000}**

**SOAP Web Service : JAX\_WS**

**Axis 2, metro tool etc**

**RestFull Web Service : JAX\_RS**

**Apache Wing, Spring framework etc**

**Unit Testing :**

**Testing is use to find the bugs or defects or errors.**

**2 types**

1. **White box testing**

**Input -🡪Coding -🡪Output**

1. **Black box testing**

**Input ---🡪Output**

**Unit : Unit may be function/ methods/modules or class. Unit testing is a type of white box testing.**

**jUnit : It is opens source framework which provided set of API which help to do unit testing for Java Application.**

**jar : .java/.class**

**war : html/css/js/jsp/servlet/xml**

**ear : EJB**

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

**coding**

**return x+y;**

**}**

**junit4-4.8.2**

**hamcrest-core-1.3**

**Desing Pattern :**

**Factory method design pattern**

**class Employee {**

**private Employee() {}**

**public static Employee getInstance(int code) {**

**if(code==1){**

**return new Employee();**

**}else {**

**return null;**

**}**

**}**

**void display() {**

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

**}**

**}**

**class DesDemo {**

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

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

**Employee emp = Employee.getInstance(2);**

**emp.display();**

**}**

**}**

**Singleton Design pattern**

**class Employee {**

**private Employee() {}**

**int id;**

**private final static Employee emp = new Employee();**

**public static Employee getInstance() {**

**return emp;**

**}**

**void display() {**

**System.out.println("dis() method "+id);**

**}**

**}**

**class DesDemo {**

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

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

**Employee emp = Employee.getInstance();**

**emp.id=100;**

**emp.display();**

**Employee emp1 = Employee.getInstance();**

**emp1.display();**

**}**

**}**

**Java 8 Features**

1. **Interfaces : Java 8 interface can contains method with body but method must be default and static.**

**Interface A**

**void dis1();**

**default void dis2(){}**

**default void dis3(){}**

**Interface B**

**void dis1();**

**default void dis2() {}**

**static void dis3() {}**

**class C implements A,B {**

**public void dis1() {}**

**public void dis2() {**

**}**

**public void dis3() {**

**}**

**}**

**interface A {**

**void dis1();**

**default void dis2() {**

**System.out.println("Default implementation - dis2 ");**

**}**

**default void dis3() {**

**System.out.println("Default implementation - dis3");**

**}**

**static void dis4() {**

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

**}**

**}**

**class B implements A {**

**public void dis1() {**

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

**}**

**public void dis2() {**

**System.out.println("Default implementation - dis2- Override ");**

**}**

**public static void dis4() {**

**System.out.println("static implementation - dis4- own method ");**

**}**

**}**

**class DesDemo {**

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

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

**obj1.dis1(); obj1.dis2(); obj1.dis3(); obj1.dis4(); A.dis4();**

**}**

**}**

**Inner classes**

**class Outer {**

**int a;**

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

**void dis1() {**

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

**in.dis2();**

**}**

**class Inner {**

**int b;**

**void dis2() {**

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

**}**

**}**

**}**

**class DesDemo {**

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

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

**out.dis1();**

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

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

**in.dis2();**

**}**

**}**

**Lambda Expression**

**@FunctionalInterface**

**interface Abc {**

**void dis1();**

**}**

**interface Mno {**

**void dis2();**

**void dis3();**

**}**

**class Xyz implements Abc {**

**public void dis1() {**

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

**}**

**}**

**class DesDemo {**

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

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

**Abc obj2 = new Abc(){**

**public void dis1() {**

**System.out.println("Abc method using Anonymouse inner class");**

**}**

**};**

**obj2.dis1();**

**Abc obj3 = ()->System.out.println("Lambda Expression coding");**

**obj3.dis1();**

**}**

**}**

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

**interface Cal {**

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

**}**

**class DesDemo {**

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

**Cal cc = (a,b)->a+b;**

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

**Runnable r = ()->{**

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

**System.out.println("Thread start");**

**}**

**};**

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

**t.start();**

**}**

**}**

**Another examples**

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

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

**class Abc implements Consumer<Integer> {**

**public void accept(Integer n) {**

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

**}**

**}**

**class DesDemo {**

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

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

**ll.add(10);ll.add(20);ll.add(30);ll.add(40);**

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

**ll.forEach((v)->System.out.println(v));**

**System.out.println("Without lambda expression");**

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

**ll.forEach(obj);**

**}**

**}**