**Java and Spring Framework notes**

**Day 1 : 11/11/2020**

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

Initial name of the Java **Oak.**

Games Gosling and Team.

Sun Microsystem it is part Oracle.

Nov 1995 rename to Java.

Java

JavaSE JavaEE JavaME

JavaStandard Java Java

Edition Enterprise Micro

Edition Edition

Syntax

class className {

fields;

methods;

}

class Demo {

public static void main(String args[]) {

}

}

Naming Rules :

Class always follow Pascal naming rules

If class contains one world first letter upper case,

If it contains more than one world each world first letter upper rules.

Demo

Employee

EmployeeDetails

JDK Java Development Kit

JRE Java Run time environment

To compile the program

**javac className.java**

To run the program

**java className**

Demo.java

class Demo {

public static void main(String args[]) {

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

}

}

**Data Types**

2 types

1. Primitive data type : it is use to hold only value
   1. byte
   2. short
   3. int
   4. long without decimal
   5. float
   6. double

with decimal

* 1. char single char
  2. boolean true or false.

2. Non primitive data type or reference data types : it is use to hold value as well as another data.

a. array

b. class

c. interface

d. enum

**Demo.java**

class Demo {

public static void main(String args[]) {

int a=10;

System.out.println(a);

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

}

}

Operator:

1. Arithmetic operator : +, -, \*, /, %
2. Logical operator : &&, ||, !
3. Relational Operator : >, >=, <, <=, ==, !=
4. Assignment operator : =
5. Increment and decrement operator : ++, --

Increment and Decrement operator

class Demo {

public static void main(String args[]) {

int a=10;

int b=a;

System.out.println(b);

b=a++; // a= a+1 post increment

System.out.println(b);

b=++a; // pre increment

System.out.println(b);

}

}

If statement

1. Simple if

if(condition) {

}

1. If else

if(condition) {

}else {

}

1. Nested if

if(condition) {

if(condition) {

}else {

}

}else {

if(codition) {

}

}

1. If else if

if(condition) {

}else if(condition) {

}else if(condition) {

}else {

}

Switch statement

Syntax

switch(varibleName) {

case lable1: block1;

break;

case lable2: block1;

break;

case lable3: block1;

break;

default : wrong choice

break;

}

switch, case, break and default are keywords.

variableName : int family, char or String

Looping

while loop

do while loop

for loop

For each loop and enhanced loop

Syntax

Initialization

i=0

n=10

while(i<=n) {

body of loop

System.out.println(i);

i++; or n--;

}

Loop example

class Demo {

public static void main(String args[]) {

/\*int i=1;

int n=10;

System.out.println("Using while loop");

while(i<=n) {

System.out.println(i);

i++;

}\*/

/\*

int i=1;

int n=10;

do {

System.out.println(i);

i++;

}while(i<=n);\*/

/\*

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

System.out.println(i);

}\*/

}

}

array : array is collection of same data type values.

Syntax

datatype arrayName[];

int abc[];

for each or enhanced loop

syntax

for(datatype variableName : arrayName) {

}

Array with for loop

class Demo {

public static void main(String args[]) {

int abc[];

int num[]={100,200,300,400,500,45,67,34,23,56,78,90};

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

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

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

for(int i=4;i<num.length;i++) {

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

}

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

for(int n:num) {

System.out.println(n);

}

}

}

Memory creation for array

datatype arrayname[]=new datatype[size];

int num[]=new int[10];

class Demo {

public static void main(String args[]) {

int num[]=new int[10];

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

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

num[0]=100;

num[1]=110;

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

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

}

}

class Demo {

public static void main(String args[]) {

int num1[]={1,2,3,4,5};

int num[]=new int[10];

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

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

num[0]=100;

num[1]=110;

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

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

}

}

In Java we can take the value through keyboards 4 ways

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

Scanner : Scanner is pre-defined class part of util package. Package is a collection of classes and interfaces.

Syntax to create scanner class object.

Scanner obj = new Scanner(System.in);

import java.util.Scanner;

class Demo {

public static void main(String args[]) {

Scanner obj = new Scanner(System.in);

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

int n = obj.nextInt(); //pre-defined methods

System.out.println(n);

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

String name= obj.next();

System.out.println(name);

/\*

nextByte();

nextShort()

nextInt();

nextLong();

nextDouble()

nextFloat()

nextBooean()

but nextChar() method

next() for String value

\*/

}

}

Taking array value through keyborads

import java.util.Scanner;

class Demo {

public static void main(String args[]) {

Scanner obj = new Scanner(System.in);

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

int n = obj.nextInt();

int num[]=new int[n];

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

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

num[i]=obj.nextInt();

}

System.out.println("Number are ");

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

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

}

}

}

1. Through keywords receive Employee Id, Name, Salary

Then using Salary find HRA, DA and PF

HRA 10%

DA 5%

PF 7 %

Display Id, Name, and Salary(GrossSalary)

For 1 Employee

1. More than one Employee using array concept.

Id,Name,Salary,Designation

HRA 10%

DA 5%

PF 7 %

If Designation is Manager + Bonus = 5000

If Designation is Programmer + Bonus = 2000

Else = 1000

**Day 2 : 13/11/2020**

object : object is any real world entity.

Properties(state)-🡪 have 🡪 variables/fields

Name, color, h, w 🡪int,float,char,double,string

Person

Behaviour --🡪do/does -🡪 functions/methods

Teaching(), talking(), sleeping(), going()

Place

Bank

Animal

Name,wheel, price etc

Car

Start(), appliedGear(), moving(), stop()

Etc

class : class is a blue print of object or template of objects.

Class is a user-defined data type which help to create the objects.

Syntax to create the object

**ClassName objectReference = new ClassName();**

class Car {

int wheel;

float price;

String color;

void start() {

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

}

void appliedGear() {

}

void moving() {

}

void stop() {

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

}

}

class App {

public static void main(String args[]) {

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

//start();

Car bmw = new Car();

bmw.start();

bmw.stop();

}

}

**Type of variables or fields**

3 types

1. Instance variable
   1. The variable which declare outside a method including main method is known as instance variable.
   2. Instance variable hold default value according to their data types

int family –0

float family -0.0

char space white

boolean –false

String –null

* 1. Instance variable we can access within same class method but method must be non-static.

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

class Car {

int wheel;

float price;

String color;

void start() {

String msg="Welcome";

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

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

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

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

//System.out.println(msg);

}

void stop() {

int n=10;

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

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

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

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

System.out.println(n);

}

}

class App {

public static void main(String args[]) {

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

//start();

Car bmw = new Car();

bmw.start();

bmw.stop();

}

}

**Assignment**

Create two classes

Employee

id,name,salary -🡪instance variable

Scanner object (outside methods)

read() receive id,name,salary

calSalary()

local variable hra,df,pf

hra = 10%

da = 5%

pf = 7%

Gross Salary

display()

id,name,salary (grossSalary)

EmployeeTest

Main method

Create the object

Call read method using object

Then we have to call calSalary method

You have to call display methods

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

Pts

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

class Car {

Car() {

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

}

void carInfo() {

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

}

}

class App {

public static void main(String args[]) {

//Car innova = new Car();

//innova.carInfo();

}

}

Parameterized constructor

class Abc {

int a,b,sum;

Abc() {

a=10;

b=20;

}

Abc(int x, int y) {

a=x;

b=y;

}

void setValue(int x, int y) {

a=x;

b=y;

}

void add() {

sum = a+b;

}

void display(){

System.out.println("Sum = "+sum);

}

}

class App {

public static void main(String args[]) {

Abc obj1 = new Abc();

obj1.add();

obj1.display();

Abc obj2 = new Abc();

obj2.add();

obj2.display();

Abc obj3 = new Abc(1,2);

obj3.add();

obj3.display();

Abc obj4 = new Abc();

obj4.setValue(100,200); obj4.setValue(101,201); obj4.setValue(102,202);

obj4.add();

obj4.display();

}

}

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

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 App {

public static void main(String args[]) {

Employee emp = new Employee();

//emp.name= "Ravi";

//emp.salary = -20000;

emp.setValue("Ravi",-20000);

emp.display();

}

}

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

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

**Fields;**

**Methods;**

**}**

**class NewClass extends OldClass {**

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

**methods;**

**}**

**class A {**

**void dis1() {**

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

**}**

**}**

**class B extends A{**

**void dis2() {**

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

**}**

**}**

**class App {**

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

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

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

**obj1.dis1();**

**obj2.dis2();**

**obj2.dis1();**

**}**

**}**

**Types of Inheritance**

1. **Single Inheritance**

**One super class and one sub class**

**class A {}**

**class B extends A{}**

1. **Multilevel inheritance**

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

**class A {}**

**class B extends A{}**

**class C extends B {}**

**class D extends C {}**

1. **Hierarchical Inheritance**

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

**class A {}**

**class B extends A{}**

**class C extends A{}**

1. **Multiple inheritance**

**More than one super class and one sub class**

**class A{}**

**class B {}**

**class C extends A,B {} Java doesn’t support.**

**In java we can achieve multiple inheritance using interface.**

**OOPs Relationship**

1. **Manager is a Employee**
2. **Employee/Programmer/Manager has a Address**

**class Employee{**

**id,name,salary**

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

**}**

**class Manager extends Employee{**

**}**

**class Programmer extends Employee{}**

**class Address{**

**city,state**

**}**

Employee

id,name,salary -🡪instance variable

Scanner object (outside methods)

read() receive id,name,salary

calSalary()

local variable hra,df,pf

hra = 10%

da = 5%

pf = 7%

Gross Salary

display()

id,name,salary (grossSalary)

Manager extends Employee

numberOfEmp of type int

readMgr()

disMgr()

Programmer extends Employee

projectName:string

readPrg()

disPrg()

EmployeeTest

Main method

Create Manager class object

Call read(), readMgr()

calSalary()

display()

disMgr()

Scanner obj = new Scanner(System.in);

String name = obj.next(); Raj Deep

String fname = obj.nextLine(); Raj Deep Kumar

String[] name;

String [] name;

String []name;

String name[];

**Day 3 : 18/11/2020**

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

2 types

1. Compile time or static binding or early binding.

Method overloading

1. Run time or dynamic binding or late binding.

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).

Example

class Operation {

void add(int x, int y) {

System.out.println(x+y);

}

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

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

}

void add(float x, float y) {

System.out.println(x+y);

}

void add(String x, String y) {

System.out.println(x+y);

}

}

class Demo {

public static void main(String args[]) {

Operation op = new Operation();

op.add(10,20);

op.add(1,2,3);

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

op.add(10.10f,20.20f);

}

}

**Find area of triangle, rectangle, circle etc using method overloading concept.**

**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).

If you want to achieve method overriding the class must be in the inheritance concept.

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("White");

}

}

class Honda extends Bike {

void color() {

System.out.println("Red");

}

}

class Demo {

public static void main(String args[]) {

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

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

}

}

**Method overriding with super keyword for variable and methods**

class Bike {

int n=10;

void speed() {

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

}

}

class Pulsar extends Bike {

void speed() { //overriding

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

}

void color() {

System.out.println("White");

}

}

class Honda extends Bike {

void color() {

System.out.println("Red");

}

}

class Tvs extends Bike {

int n=20;

void color() {

System.out.println("Gray");

}

void speed() {

int n=30;

super.speed(); // merge the code , calling super class method

System.out.println("10km/hr");

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

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

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

}

}

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, static and final**

**abstract**

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

Syntax

abstract returnType methdoName(parameterList);

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

Syntax

abstract class className {

}

4 whichever class extends abstract class that class must be provide the body for all abstract method mandatory. Only that class can ignore if that class itself is a abstract class.

1. Abstract class we can’t create the object.
2. Abstract class can contains zero or 1 or many abstract methods.
3. Abstract class can contains default as well as paratermized constructor.

Abstract method code

abstract class Bike {

int n=10;

abstract void speed();

}

abstract class Pulsar extends Bike {

abstract void mailage();

void color() {

System.out.println("White");

}

}

class SuperPulsar extends Pulsar {

void speed(){

System.out.println("120km/hr");

}

void mailage(){

System.out.println("30/l");

}

}

class Demo {

public static void main(String args[]) {

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

SuperPulsar sp = new SuperPulsar();

sp.mailage();

sp.color();

sp.speed();

}

}

Final keyword

1. Final keyword we can use with variable, method and class
2. Final variable : to declare constant value in java we use final

Syntax

final int A=10;

1. Final method : if method is final we can’t override that method.
2. Final class : if class is final we can’t extends that class.

Final and abstract we can’t use together.

Final example

final class Bike {

int n=10;

final void speed() {

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

}

}

/\*class Pulsar extends Bike {

void speed() {

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

}

}

\*/

class Demo {

public static void main(String args[]) {

final int A=10;

System.out.println(A);

//A=20;

System.out.println(A);

//Pulsar pu = new Pulsar();

//pu.speed();

Bike bb = new Bike();

bb.speed();

}

}

static :

1. Static keyword we can use with variable and method but not with class(inner class or nested class if class is inner class or nested class then we can use static keywords).
2. Static method : If method is a static we can call that method with the help of className as well object of that class.
3. Static variable : If variable is a static we can assign the value for that variable with the help of className as well as object of that class.
4. Inside a non static method we can access static as well as non static variable directly of same class. But inside static method we can access only static variable directly.

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 Demo {

public static void main(String args[]) {

Abc obj1 = new Abc();

obj1.dis1();

Abc.dis2();

obj1.dis2();

obj1.a=10;

Abc.b=20;

obj1.b=30;

}

}

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 "+b);

}

}

class Demo {

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(); //a=10 , b=60

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

}

}

**Day 4 : 25/11/2020**

**Interface :**

Interface is 100% pure abstract class. It is a type of reference data type.

Syntax

Interface interfaceName {

fields;

methods;

}

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

By default all methods in interface are public and abstract.

interface Abc {

public static final int A=10;

public abstract void dis1();

}

interface Xyz {

int B=20;

void dis2();

}

interface Mno extends Abc,Xyz{

int C=30;

void dis3();

}

class Demo implements Abc, Xyz {

}

Class extends class : only one

Interface extends interface : more than one

Class implements interface : more than one

Interface doesn’t extends or implements to class

Method override with access specifiers

**Super class / interface Sub**

public public

protected public

protected

default public

protected

default(nothing)

private we can’t override

Interface example

interface Abc {

public static final int A=10;

public abstract void dis1();

}

interface Xyz {

int B=20;

void dis2();

}

interface Mno extends Abc,Xyz{

int C=30;

void dis3();

}

class Demo implements Abc, Xyz {

public void dis1() {

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

}

public void dis2() {

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

}

}

class InterfaceDemo {

public static void main(String args[]) {

Demo obj = new Demo();

obj.dis1();

obj.dis2();

}

}

**Exception Handling**

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

Java

javac java

compiler interpreter

compile time error run time error

syntax error

run time error

Error Exception

Both are pre-defined classes part of lang package.

Error : Error is a type of error which generated at the run time which we can’t handle it. Ex : JVM crash.

Exception : Exception is a type of error which we can handle it. Ex : ArithmeticException divided by 0.

Exception

Checked exception unchecked exception

RuntimeException

ArithmeticException

ArrayIndexOutOfBoundsException

NullPointerException

Etc

SQLException

IOException

Etc

Java provide 5 keywords

try

catch

finally

throw

throws

try and catch block

try {

}catch(Exception e) {

}

Exception Example

class ExpDemo {

public static void main(String args[]) {

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

int a=10;

int b=0;

try{

int res = a/b;

System.out.println("Result is "+res);

}catch(Exception e){

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

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

e.printStackTrace();

}

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

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

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

}

}

class ExpDemo {

public static void main(String args[]) {

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

int a=10;

int b=1;

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

try{

int res = a/b;

System.out.println("Result is "+res);

int res1 = 10/abc[5];

}catch(Exception e){

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

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

e.printStackTrace();

}

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

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

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

}

}

Try with multiple catch block

try{

}catch(ArithmeticException e) {

}catch(ArrayIndexOutOfBoundsException e) {

}

Try block : the code one line or multiple line which generate exception we have to write inside try block.

Catch block : it execute only if any exception generate.

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

File handling or database using JDBC

Open file

Read/write

Close file

Try{

Open connection

Do operation on database

}catch(Exception e) {

}finally {

Close connection

}

Finally block is use to close the resource

class ExpDemo {

public static void main(String args[]) {

try{

int res = 10/1;

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

try – catch –catch – catch

try – catch – finally

try – catch – catch – catch – finally

try – finally

class ExpDemo {

public static void main(String args[]) {

try{

int res = 10/0;

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

}

finally {

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

}

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

}

}

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

Syntax

throw new Exception()

or

throw new ExceptionSubClass()

class AgeException extends Exception {

AgeException() {}

AgeException(String msg) {

super(msg);

}

}

class ExpDemo {

public static void main(String args[]) {

int age = 25;

try{

if(age<21) {

//throw new Exception();

//throw new ArithmeticException();

//throw new ArithmeticException("age is < 21");

//throw new AgeException();

throw new AgeException("age is < 21");

}else {

System.out.println("You Can Vote...");

}

}catch(Exception e) {

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

}

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

}

}

Throws : throws keyword is with method signature.

Syntax

returnType methodName() throws Exception, ExceptionSubClass {

}

class ExpDemo {

static void dis1() throws Exception{

//try{

int a=10/0;

//}catch(Exception e){}

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

}

static void dis2() throws Exception{

//try{

dis1();

//}catch(Exception e){}

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

}

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

//try{

dis2();

//}catch(Exception e){}

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

}

}

Checked exception example

class ExpDemo {

public static void main(String args[]) {

System.out.println("Hello, Welcome to Java Training");

try{

Thread.sleep(5000);

}catch(Exception e){}

System.out.println("We will continue in next session...");

}

}

**Day 5 : 27/11/2020**

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

**2 types**

1. **User-defined packages**
2. **Pre-defined package.**

**Education**

**School college**

**Attendance Attendance**

**Date : util**

**Date : sql**

**Syntax to create the package**

**package com;**

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

**pre-defined packages**

**java javax : root package**

**lang swing**

**io sql**

**util net**

**awt servlet**

**net ejb**

**sql jms**

**etc etc**

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

**Without import lang package we can use all classes and interfaces belong that package.**

**String and System : part of language**

**Exception handling as well as Multithreading Thread class are part of lang package.**

**By default every java program extends Object class. It is a part of lang package.**

**Object is super class for every java program.**

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

**Access specifiers : four types**

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

**Scope : We can access within a class.**

**default (nothing): We can use with all.**

**Scope : within a same package.**

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

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

**public : instance variable, static variable, non static method, static method, constructor, public but we can’t use with local variable.**

**(in one editor or notepad we can use only one class as public).**

**Every where. (same package as well as other package).**

**IDE : netbean, eclipse , my eclipse,**

**Download the Eclipse for Enterprise JEE**

**More than employee**

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

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

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

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

**Structure :**

**class Employee {**

**id,name,salary**

**}**

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

**emp.id=100;**

**emp.name=”Ravi”;**

**emp.salary = 12000;**

**Array objects**

**Syntax**

**className refName[]=new className[size];**

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

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

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

**emps[0].name=”Ravi”;**

**emps[0].salary=24000;**

**Normal java class Vs JavaBean class**

1. **Class may be public or may not.**
2. **All property may be default or private not.**
3. **If property are private we have to provide setMethod to set the value. The method name may anything.**

**JavaBean class**

1. **Class must be public**
2. **All property must be private.**
3. **For every variable we have to provide setter and getter method.**
4. **The setter method name must be start with set followed by variableName. Which help to set the value.**
5. **The getter method name must be start with get followed by variableName. Which help to get the value.**
6. **Both method access specifiers are public.**

**JavaBean class is known as pure encapsulation class.**

**Annotation : it is meta-data. Data about data.**

**Java provided lot of pre-defined annotation and all annotation start with @ followed by name of the annotation.**

**@Override : This annotation we can write on method level if method override**

**Day 6 : 02/12/2020**

**Collection Framework (Like a Data Structure in C/C++).**

**int a;**

**array**

**int abc[];**

**Structure :**

**class Employee {**

**id,name,salary**

**}**

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

**Array object**

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

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

**emps[99]=new Employee();**

**Collection framework contains set of classes and interfaces which help to store the collection of object or elements of any type (primitive (Wrapper) or user-defined objects).**

**Collection provide set of pre-defined methods so we can add, remove, search, iterate element or object very easily.**

**Collection Framework Hierarchy**

**Collection framework is part of util package.**

**Collection 🡪 interface**

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

**Set List Queue Map -**🡪**interfaces**

**HashSet Stack PriorityQueue HashMap**

**LinkedHash ArrayList LinkedHashMap**

**TreeSet LinkedList TreeMap**

**Vector Hashtable**

**Set : In set we can hold collection of elements. Set doesn’t allow duplicate element. Some elements in Set order, Unorder or sorted.**

**List : It maintain the order using index position. List allow duplicate elements.**

**Queue : First In First Out.**

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

**Primitive Wrapper classes**

**byte Byte**

**short Short**

**int Integer**

**long Long**

**float Float**

**double Double**

**char Character**

**boolean Boolean**

**Wrapper classes is use to convert primitive to object and vice-versa.**

**int a=10;**

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

**int c = b.intValue();**

**float d = b.floatValue();**

Auto-boxing : Converting primitive to object.

All three classes doesn’t allow duplicate elements.

HashSet : display the elements unorder.

LinkedHashSet : This class internally extends HashSet. Maintain the order.

TreeSet : TreeSet internally implements SortedSet interface. So display the elements in asc order by default.

It provide extra methods like headset, tailset, subset()

List : It allow duplicate

Stack : First In Last Out

**ArrayList** :

Normal Array :

LinkedList :

Node

By default LinkedList class internally follow double linkedlist concept.

Null 🡨-- Ref ----10 ----🡪 Null

Null 🡨-- Ref ----10 <--🡪 Ref <---20--🡪Null

LinkedList is good if we do more Insertion or Delete operation

Arraylist is good if we retrieve elements again and again.

Vector : it is know as legacy class(old class). By default all methods in Vector are synchronized.

Queue : First In First Out

PriorityQueue : First In First Out base upon the priority (ie lesser number).

**Map :** key-value pairs . key is unique and value may be duplicate.

HashMap : display the elements unorder

LinkedHashMap : maintain the order (which order we inserted)

TreeMap : sorted by default as key in ascending order

Hashtable : it is legacy class (old class). By default all methods are synchronized.

Retrieve the elements from Collection

1. For each loop
2. Iterator
3. ListIterator
4. Enumberation : Legacy Iterator. With Vector.

**Day 7 : 04/12/2020**

Collection framework with generics.

CollectionClas<Type> obj =new CollectionClass<Type>();

Type may be Wrapper classes (Integer, Float), String class or user-defined class.

Collection Framework with Complex objects.

Product

ProductService

ProductMainClass

do {

1: Add , 2:Delete, 3, Update, 4, display

5: display by Id

switch() {

case 1 : Add Product, id must be unique

case 2 :Delete product using id

case 3 : update price using id

case 4 : displayAll product

case 5: display Product details using id

case 6: Display Price Asc

case 7: Display Price Desc

}

Do you want to continue

}while()

**Arrays and Collections**

Arrays and Collections are pre-defined class part of until package. Which contains set of static method which help to operation of primitive array and collection classes respectively

**Comparable and Comparator**

Both are interfaces.

Comparable is a part of lang

By default all wrapper classes implements Comparable interfaces.

**Comparator** is a part of util

Classes and interfaces details

javap java.packageName.className/interfaceName

**Comparator**

WE have to make separate class which implements comparator interface and override compare method which takes 2 object parameter.

And

In Collection.sort(listRef,classObjectWhichImplementsComaprator)

**Day 8 : 09/11/2020**

**Java 8 Features**

Collection Framework with Stream API

1. Interface can contains method with body but the method must be default or static.

**Default and static method**

interface A {

void dis1();

default void dis2() {

System.out.println("A interface default impl - dis2");

}

default void dis3() {

System.out.println("A interface default impl - dis3");

}

static void dis4() {

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

}

}

class B implements A {

public void dis1() {

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

}

public void dis3() {

System.out.println("dis3() method impl");

}

}

class InterfaceDemo1 {

public static void main(String args[]) {

B obj1 = new B();

obj1.dis1();

obj1.dis2();

obj1.dis3();

//obj1.dis4();

A.dis4(); //static method we can't override and we have to call through interfaceName

}

}

**Functional interface**

1. The interface contains only one method ie abstract method. It can contains more than one default as well as static method but only one abstract method.

To verify that interface is functional interface or not Java8 provide @FunctionalInterface annotation.

@FunctionalInterface

interface A {

void dis1();

default void dis2() {

System.out.println("A interface default impl - dis2");

}

default void dis3() {

System.out.println("A interface default impl - dis3");

}

static void dis4() {

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

}

}

class InterfaceDemo1 {

public static void main(String args[]) {

}

}

**Inner classes**

Class within another class is known as Inner class.

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

Non Static inner class

class Outer {

int a;

void dis1() {

System.out.println("dis1() outer class method");

Inner in =new Inner();

in.dis2();

}

class Inner {

int b;

void dis2() {

System.out.println("dis2() inter class method");

}

}

}

class InnerClassDemo {

public static void main(String args[]) {

Outer out = new Outer();

out.dis1();

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

in.dis2();

Outer.Inner in1 = new Outer().new Inner();

in1.dis2();

}

}

Static inner class

class Outer {

int a;

void dis1() {

System.out.println("dis1() outer class method");

}

static class Inner {

int b;

void dis2() {

System.out.println("dis2() inter class method");

}

}

}

class InnerClassDemo {

public static void main(String args[]) {

Outer.Inner in1 = new Outer.Inner(); //syntax to create static inner class object

in1.dis2();

}

}

AWT and Swing or Android

**Runtime polymorphism**

interface A {

void dis1();

}

class B implements A {

public void dis1() {

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

}

public void dis2() {

System.out.println("its own method");

}

}

class InnerClassDemo {

public static void main(String args[]) {

A obj1 = new B(); // creating the object of sub class and reference of super interface/class. Runtime polymorphism

obj1.dis1(); // with the help of A reference we can call only those method which belongs to

//obj1.dis2(); // A interface or overrided methods.

}

}

**Lambda Expression**

**Up Java7 Java is Procedure and Object Oriented Programming language.**

**But from Java8 onwards Java become functional programming language.**

**From Java8 onwards we can write nested method using Lambda expression like a JavaScript.**

**Lambda expression is anonymous function.**

**Syntax**

**If you want to use the lambda expression. The interface must be functional interface.**

**interfaceName referecName = ()-> statement.**

@FunctionalInterface

interface A {

void dis1();

}

class B implements A {

public void dis1() {

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

}

}

class InnerClassDemo {

public static void main(String args[]) {

A obj1 = new B();

obj1.dis1();

A obj2 = new A(){

public void dis1() {

System.out.println("A interface method override by anonymous class");

}

};

obj2.dis1();

A obj3 = ()->System.out.println("Provided body using Lambda Expression...");

obj3.dis1();

}

}

Stream API **: Lambda Expression and Collection Framework**

Date API

**Day 9 : 11/12/2020**

Stream API :

Collection Framework is like a Data Structure. A collection is an in-memory data structure to hold value(primitive or user-defined objects) and before we start using collections. All the values been populated.

Java Stream doesn’t store data. It operate on the source(collection) data structure and produce pipelined data that can be use and perform any specific operation on demand.

Stream : Flow of data. It abstraction between source and destination.

Array/Collection -🡪Source(Stream)----🡪 Op1-🡪Op2🡪Op3-🡪Opn---🡪 Destination (terminator operator)

Op 🡪 Intermediate operator it return stream reference.

Destination 🡪 It return non stream reference or any primitive value.

Top core functional interface.

java.util.function.\*;

4 interfaces

1. Predicate : test(T) takes T parameter and return Boolean value.
2. Function : apply(T) takes T parameter and return R.
3. Supplier : get() no parameter but return T value.
4. Consumer : accept(T) take T parameter but no return type.

**Comparator using Java8 Steam API with lambda Expression.**

**Date API in Java8**

**Before Java8 if we can to do any date operation**

**Date : util (Basic Methods).**

**Calendar class : util (more methods)**

**Convert custom date format**

**SimpleDateFormat : text package.**

**Date class is mutable class. (We can change the date).**

**Employee javaBean**

**Id,name,salary,dob, doj**

**Setter and getter methods.**

**Java 8**

**All date related classes are part of time package.**

**All those classes are immutable classes.**

**Day 10 : 14/12/2020**

**Multi threading :**

**Multi tasking .**

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

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

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

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

**Thread also known as light weighted process. Process more resource of your machine. Thread take less resource of your machine. Inside a process always one or more than one thread will execute.**

**By default every java program depends upon the thread.**

**C and C++ are process based programming language.**

**Where is Java is thread base programming language.**

**All multithreading API are part of lang package.**

**If we can to check default thread which run inside a every java program main method.**

**currenThread() is a pre-defined method part of Thread class. Thread class is a part or lang package.**

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

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

**t 🡪 Thread[main,5,main]**

**main🡪 by default name of the thread is main.**

**5🡪 default priority of the thread is 5**

**main🡪 group of the thread is main**

MIN\_PRIORITY🡪1

MAX\_PRIORITY->10

NORM\_PRIORITY->5

In between 1 to 10

Default thread example

class Demo {

public static void main(String args[]) {

Thread t = Thread.currentThread();

System.out.println(t);

String name = t.getName();

int i = t.getPriority();

System.out.println(name);

System.out.println(i);

t.setName("My Thread");

System.out.println(t);

t.setPriority(2);

System.out.println(t);

t.setPriority(Thread.MAX\_PRIORITY);

System.out.println(t);

}

}

**Multitasking :**

Using process base

Using thread base

T1 t2 t3

Multitasking using process base is slower than multi tasking using thread base.

In Java we can create more than one thread using 2 ways

1. Extends Thread class
2. Implements Runnable interface

Both API part of lang package.

Extends Thread class

1. Create user-defined class and extends Thread class.
2. Create the reference of thread class.
3. With the help of that reference class start() method.
4. Start() method internally call run() method which is part of Thread class.
5. If you want any custom logic so we have to override run() method.

**Using Thread class**

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 Demo {

public static void main(String args[]) {

A obj1 = new A();

B obj2 = new B(); // obj1 and obj2 are thread class reference.

obj1.start();

obj2.start(); // Runnable state : means ready to run.

}

}

**Using Runnable interface**

1. Create user-defined class and implements Runnable interface.
2. Runnable is a interface which contains one abstract method.

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 Demo {

public static void main(String args[]) {

A obj1 = new A();

B obj2 = new B();

Thread t1 = new Thread(obj1); //Thread class take Runnable interface reference as a parameter.

Thread t2 = new Thread(obj2);

//t1 and t2 are thread class reference.

t1.start();

t2.start(); // Runnable state : means ready to run.

}

}

Thread life cycle

Create the Thread Runnable Running Destroy

obj1 --🡪 obj1.start() inside run() i=10,j=10

t1 --🡪 t1.start() inside run(),i=10,j=10

sleep()

isAlive()

wait()

notify()

notifyAll()

class A implements Runnable{

@Override

public void run() {

Thread t = Thread.currentThread();

String name = t.getName();

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

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

try{

Thread.sleep(1000);

}catch(Exception e){}

}

}

}

class B implements Runnable{

@Override

public void run() {

Thread t = Thread.currentThread();

String name = t.getName();

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

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

try{

Thread.sleep(500);

}catch(Exception e){}

}

}

}

class Demo {

public static void main(String args[]) {

A obj1 = new A();

B obj2 = new B();

Thread t1 = new Thread(obj1); //Thread class take Runnable interface reference as a parameter.

Thread t2 = new Thread(obj2);

t1.setName("1st Thread");

t2.setName("2nd Thread"); //t1 and t2 are thread class reference.

System.out.println("Thread1 "+t1.isAlive());

System.out.println("Thread2 "+t2.isAlive());

t1.start();

t2.start(); // Runnable state : means ready to run.

System.out.println("Thread1 "+t1.isAlive());

System.out.println("Thread2 "+t2.isAlive());

}

}

**Multithreading program with Wrong output**

class Booking implements Runnable{

int avl=1;

@Override

public void run() {

Thread t = Thread.currentThread();

String name = t.getName();

if(avl>0) {

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

avl=avl-1;

}else {

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

}

}

}

class Demo {

public static void main(String args[]) {

Booking b1 = new Booking(); //avl=1

//Booking b2 = new Booking(); //avl=1

//Booking b3 = new Booking(); //avl=1

Thread t1 = new Thread(b1);

Thread t2 = new Thread(b1);

Thread t3 = new Thread(b1);

t1.setName("Ravi");

t2.setName("Ajay");

t3.setName("Vijay");

t1.start();

t2.start();

t3.start();

}

}

Synchronization : It is use to block or lock the thread. Synchronization is a concept which help to use all resource for only one thread at time.

To achieve synchronization in java we have to use

synchronized keyword with method or block

class Booking implements Runnable{

int avl=1;

@Override

public synchronized void run() {

Thread t = Thread.currentThread();

String name = t.getName();

if(avl>0) {

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

avl=avl-1;

}else {

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

}

}

}

class Demo {

public static void main(String args[]) {

Booking b1 = new Booking(); //avl=1

//Booking b2 = new Booking(); //avl=1

//Booking b3 = new Booking(); //avl=1

Thread t1 = new Thread(b1);

Thread t2 = new Thread(b1);

Thread t3 = new Thread(b1);

t1.setName("Ravi");

t2.setName("Ajay");

t3.setName("Vijay");

t1.start();

t2.start();

t3.start();

}

}

Wait(), notify() and notifyAll()

Wait() : this method is use to make the thread to wait()

Notify() this method is use to notify() waited thread.

If more than one thread are waiting state. We call only thread is ready to run.

But if we want to start all thread then we have to call notifyAll().

Consumer and Producer :

Con proc (wait)

Do task

Wait(notify

To producer)

Points.

1. Wait(), notify() and notifyAll() all three method are part of Object class.
2. We have to create more than one thread in same memory.
3. The methods must in synchronized.

**Inter thread communication.**

class Abc implements Runnable{

@Override

public synchronized void run() {

Thread t = Thread.currentThread();

String name = t.getName();

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

try{

Thread.sleep(500);

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

wait();

}

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

wait();

notify();

}

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

}catch(Exception e){}

}

}

}

class Demo {

public static void main(String args[]) {

Abc b1 = new Abc();

Thread t1 = new Thread(b1);

Thread t2 = new Thread(b1);

Thread t3 = new Thread(b1);

t1.setName("Ravi");

t2.setName("Ajay");

t3.setName("Vijay");

t1.start();

t2.start();

t3.start();

}

}

Dead lock : One synchronized block or method trying to call another synchronized block or method and that block calling first synchronized block or methods. That time Dead lock occurs.