Day 2

28-02-2022

1st questions

String

StringBuilder

StringBuffer

Scanner obj = new Scanner(System.in);

System.out.println(“Enter the name”);

String name = obj.nextLine(); // Raj Deep

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

System.out.println(“Enter the id”);

itn id = obj.nextInt();

In Java String is pre-defined class part of lang package.

By default every java program imported lang package.

String also known as reference data types or non primitive data types.

Syntax to create string class object.

String str1 = “Welcome to Java” // literal style

String str2 = new String(“Welcome to Java”); // using new keyboard.

2nd question

public

private

protected

default

if(task1)

Java

Welcome program

Data Types

Primitive data types

8 data types

Non primitive data types

4

Array

Array syntax

datatype arrayName[];

int abc[];

int []xyz;

int [] mno;

int[] mmm;

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

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

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

for(int i=2;i<4;i=i+2) {

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

}

for(int n:num) {

System.out.println(n);

}

Creating array memory in java

datatype arrayName[]=new datatype[size];

Store n number of employee details as id,name,salary

Take array value through keyboards.

Find gross salary = 10% hra, 5%da and 7%pf

Salary = salary + hra + da – pf ;

Id,name,salary(Gross Salary)

Store n number of employee details as id,name,salary, desg

Take array value through keyboards.

Find gross salary = 10% hra, 5%da and 7%pf

If desg is manager + 50000 bonus

If desg is developer + 3000 bonus

Else

1500

Salary = salary + hra + da – pf ;

Id,name,salary(Gross Salary)

Store n number of employee details as id,name,salary, desg

Take array value through keyboards.

Find gross salary = 10% hra, 5%da and 7%pf

If desg is manager + 50000 bonus

If desg is developer + 3000 bonus

Else

1500

Salary = salary + hra + da – pf ;

Id,name,salary(Gross Salary)

Display descending order salary by employee details.

Pre defined classes/ user defined class

Interface pre-defined or user-defined

Enum

Day 3

02-03-2022

OOPs :

object : any real word entity.

Properties or state -🡪 have

Person

Behaviour 🡪do/does

Bank

Car

Animal

Employee

Customer

class : blue print of object or template of object or user-defined data type which help to create the object.

className referenceName = new ClassName();

types of variable or fields.

3 types

1. Instance variable
2. The variable which declared outside a method is known as instance variable.
3. It hold default value according to their data types. int family 🡪0 float family 0.0, char space, String null, boolean false.
4. We can use instance variable directly inside all method but method must be part of same class and it must not static.
5. Local variable
6. The variable which declared inside a method is known as local variable.
7. It doesn’t hold default value.
8. Scope within that method where it declared.
9. Static variable

Encapsulation : Binding or wrapping data and code in a single unit is known as Encapsulation.

**class**

local variable and instance variable have same name then local variable hide the visibility of instance variable. So if you want to refer to instance variable we have use this.variableName;

Day 3 :

03-03-2022

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

class OldClass { super class or base class or parent class

Properties

Behaviour

}

class NewClass extends OldClass{ sub class, child class or derived class.

Properties

Behaviour

}

Types of inheritance

1. Single inheritance :

One super class and one sub class

Class A { }

Class B extends A { }

1. Multilevel inheritance : one super class and n number of sub class connected one by one

Class A { }

Class B extends A { }

Class C extends B { }

Class D extends C { }

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 this type of inheritance.

This type of inheritance we can achieve using interface.

1. Hierarchical inheritance

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

Class A { }

Class B extends A{ }

Class C extends A{ }

OOPs relationship

1. Is a relationship
2. Has a relationship

class Employee {

id,name,salary

readEmp()

disEmp()

}

class Manager extends Employee {

Address add = new Address();

numberOfEmp

}

class Programmer extends Employee{

techName;

}

class ProjectManager extends Manager{

projects

}

class Address {

city, state

}

Has a relationship

Association

class A {

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

}

class B {

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

}

Aggregation : it is a type of association which is known as weak association.

class Manager {

Address oadd = new Address();

Address ladd =new Address();

}

class Address {

}

Composition : it is a type of association which is known as strong association.

class Student {

StudentHistory sh = new StudentHistory();

}

class StudentHistory {

}

Polymorphism :one name many forms or many implementation

Compile time or static binding or early binding

Ex : Method overloading : The method have same name but different parameter list is known as method overloading.

Run time or dynamic binding or late binding

Ex :method overriding : method same and same method signature is known as method overriding.

Annotation : annotation is known meta-data. Meta data means data about data.

Java provided lot of pre-defined annotation and all annotation start with prefix @ followed by annotation name. Few annotation we can use on method level or class level or property level.

@Override : This annotation we can use on method level (only those method which are override by sub class).

abstract, final and static

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

1. Abstract method : The method without body or without curly braces or incomplete method. is known as abstract method.
2. If class contains abstract class then we have to declare the class as abstract class.
3. Which ever class extends abstract that class must be override all abstract method belong to that class. that class can ignore if this class itself is a abstract class.
4. Abstract class we can’t create the object.
5. Abstract class can contains normal as well as abstract method. it can contains 0 or 1 or many abstract method.

Final :

Final keyword we can use with variable, method and class

final variable : to declare constant variable in java we use final keyword.

final int A=10;

A=20; Error we can’t change the final variable value.

final method : if method is final we can’t override that method in sub but we can use it.

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

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

1. static variable : if variable is static we can access or assign the value for that variable using class name as well as through object.
2. static method : if method is static we can call that method using class name as well as through object.
3. inside static method we can access only static variable directly. Inside non static method we can access both static as well as non static variable directly.

Every class we will get only one static memory

Number of heap memory equal number of object creation.

Access modifiers or specifiers

It is use to show the visibility of variable, method and class.

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

Scope : within a same class

default : we can use with all.

Scope : within a same package.

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

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

public : we can use with instance variable, static variable, static as well as non static method and constructor, class but not with local variable.

Scope : same package and other package.

Package : collection of classes and interfaces.

When two classes and interface which have same name but different purpose then using package concept we can avoid naming conflicts.

Education

School college

Attendance Attendance

Interface : it is known as 100% pure abstract class.

Syntax

Interface interfaceName {

fields // public static and final by default

methods; // public and abstract

}

interface Abc {

int A=10;

void dis1();

}

interface Xyz {

int B=20;

void dis2();

}

interface Mno extends Abc,Xyz{

int C=30;

void dis3();

}

class Test implements Mno {

public void dis1() { }

public void dis2() { }

public void dis3() { }

}

07-03-2022

Types exceptions

Try and catch

Finally

Nested try

User-Defined Exception

throw and throws

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

Java program

Compile program run the program

Javac java

Compile time error run time error

Syntax error

Typo error.

Run time error

Error Exception 🡪 both are pre-defined classes in java.

Error : The error which generate at the run time which we can’t handle it.

JVM Crash, Software and hardware issue.

Exception : it is type of run time error which we can handle it . divided by zero.

Object

Throwable

Error Exception

Checked exception unchecked exception

RuntimeException

IOException ArithmeticException

FileNotFoundException ArrayIndexOutOfBoundsException

SQLException

Five keyword

Try

Catch

Finally

Throw

Throws

Try with single catch block

Try with multiple catch block

try {

code 1

try{

code 2

}catch(ArithmeticException e) {

}

Code 3

}catch(NumberFormatException e) {

}

Finally block : finally is a block which will execute 100% sure if any exception generate or not.

try {

open the file

read and write

}catch(Exception e) {

}finally {

close the file

}

throw : throw keyword is use to raise or generate pre-defined or user-defined exception with terms and condition .

syntax

throw new Exception();

or

throw new ExceptionSubClass();

throws is use to throw the exception to caller method. The exception can be checked or unchecked.

void display() throws Exception, ExceptionSubClass {

}

Collection Framework

Collection framework is use to store the collection of object or elements.

int a=10;

int abc[];

class Employee {

int id;

String name;

float salary;

}

Employee emp = new Employee();

emp.id=100;

emp.name=”Ravi”;

emp.salary = 12000;

Array object

Employee employees[]=new Employee[100]; 0 object created of employee class.

Memory created array not for Employee.

employees[0]=new Employee();

employees[1]=new Employee();

employees[99]=new Employee();

collection framework provided set of collection of classes and interfaces which help to store the collection of object or elements. By default collection of classes allow to store any object.

it provided lot of pre-defined method which help to add, remove, search, iterate the elements very easily.

Collection framework hierarchy

Collection --🡪 interface

Set List Queue Map

All four are interfaces

Set, List and Queue internally extends Collection. But Map doesn’t extends Collection.

Set :Doesn’t allow duplicate. Few API under set maintain the order, unorder and sorted.

HashSet🡪 unorder

LinkedHashSet🡪 Order

TreeSet 🡪 Sorted.

List : maintain the order. It allow duplicate

Stack : First In Last Out

ArrayList

LinkedList : it is also type of queue

Queue : First In First Out

PriorityQueue

Map :it allow to store the information in key-value pairs key is unique and value may be duplicate.

HashMap : unorder

LinkedHashMap : order

TreeMap : sorted as key

Hashtable :synchronized

Wild card with generics

username,password, typeOfUser

raj, 123, client

ravi, 567,visitor

Ramesh, 111,client

Raju, 444,visitor

09-03-2022

class Employee {

private int id=123;

private String name=”Ramesh”;

private Address add;

Employee() {

id=100;

name =”Ravi”;

add = new Address();

}

}

class Address {

private String city;

private String state;

}

10-03-2022

1. Thread and Runnable :
2. Thread using API.
3. Using multithreading

Finally and throw

1. File handling

Thread and Runnable

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

Synchronized keyword : we can use with method or block.

wait(), notify() and notifyAll()

wait() : this method is use to suspend()

notify() this method is use to resume the thread

more than one thread must be created in same memory.

Method must be synchronized.

These method belong to Object class.

Inter thread communication.

**javac -d . Demo.java**

finally and throw

try{

open the file

read and write

}catch(Exception e) {

}finally {

Close resources

}

**IO packages**

System.out.println();

System.in;

Scanner obj = new Scanner(System.in);

System is a class part of lang package it contains in, out and err three constant and static fields.

System.in InputStream class reference.

InputStream is = System.in;

InputStream always refer to standard device keywords.

PrintStream ps = System.out;

ps.println(“Welcome to Java”);

PrintStream class always refer to standard output device : console

IO operation using stream

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

Source 🡪 keyboard, file, database, network etc.

Destination 🡪 console, file, database, network, browser etc.

Stream

byte char

1 byte 2 byte

Input Output input output

InputStream OutputStream Reader Writer

DataInputStream DataOutputStream FileReader FileWriter

FileInputStream FileOutputStream BufferedReader BufferedWriter

BufferedInputStream BufferedOutputStream

PrintStream

Serialization : storing the object in external file is known as object serialization.

Property : fields : storing only fields.

Behaviour ; function not storing

Identity : reference : not storing

**14-03-2022**

Java 8 Features

From Java8 onward interface can contains method with body.

But method must be default or static.

If interface contains default method sub class can override but not mandatory.

If interface contains static method sub class can’t override and that method must be call through interfaceName.methodName();

**package** com;

**interface** A {

**void** dis1();

**default** **void** dis2() {

System.***out***.println("This is A interface default method dis2()");

}

**default** **void** dis3() {

System.***out***.println("This is A interface default method dis3()");

}

**static** **void** dis4() {

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

}

}

**class** B **implements** A {

@Override

**public** **void** dis1() {

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

}

@Override

**public** **void** dis2() {

System.***out***.println("B class overrided default method part of A interface");

}

//@Override

**public** **static** **void** dis4() {

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

}

}

**public** **class** Java8InterfaceExample {

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

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

obj1.dis1();

obj1.dis2();

obj1.dis3();

A.*dis4*();

obj1.*dis4*(); // B class its own method

B.*dis4*(); // B class itw own method

}

}

Marker : The interface contains zero method or no method is known as marker interface.

Serializable

Functional interface

The interface must be contains only one abstract method. it can contains more than one default as well as static method but only one abstract method is known as functional interface.

**package** com;

@FunctionalInterface

**interface** Sample {

**void** dis1();

//void dis2();

**default** **void** dis3() {

}

**default** **void** dis4() {

}

**static** **void** dis5() {

}

**static** **void** dis6() {

}

}

**public** **class** FunctionalInterfaceExample {

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

}

}

Inner class

Class within another class is known as inner class.

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

**package** com;

**class** Outer {

**void** dis1() {

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

Inner1 obj = **new** Inner1();

obj.dis2();

}

// number of outer class object.

**class** Inner1 {

**void** dis2() {

System.***out***.println("dis2 method part of non static inner class");

}

}

// one copy

**static** **class** Inner2 {

**void** dis3() {

System.***out***.println("dis3 method part of static inner class");

}

}

}

**public** **class** InnerClassExample {

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

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

out.dis1();

// syntax to create the non static inner class object

// one way

Outer.Inner1 in1 = out.**new** Inner1();

in1.dis2();

Outer.Inner1 in2 = **new** Outer().**new** Inner1();

in2.dis2();

// syntax to create the static inner class object

Outer.Inner2 in3 = **new** Outer.Inner2();

in3.dis3();

}

}

Lambda Expression : it is a Greek word, Using lambda expression we can do functional programming language in java.

Lambda expression replacement of anonymous function or methods.

Lambda expression we can use only on functional interface.

**package** com;

**interface** Abc {

**void** dis1();

}

**class** Xyz **implements** Abc {

**public** **void** dis1() {

System.***out***.println("dis1 method provided body by Xyz");

}

}

**public** **class** AnonyousInnerClassdemo {

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

//1st approach

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

obj1.dis1();

//2nd approach

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

**public** **void** dis1() {

System.***out***.println("This is anonymous inner class with logic 1");

}

};

obj2.dis1();

Abc obj3 = **new** Abc() {

**public** **void** dis1() {

System.***out***.println("This is anonymous inner class with logic 2");

}

};

obj3.dis1();

// 3rd approach

Abc obj4 = ()->System.***out***.println("This is Lambda expression styntax");

obj4.dis1();

}

}

Stream API

Multithreading:

Program : Set of instruction to perform a specific task is known as a program.

Processor : Processor is responsible to execute the code.

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

Thread : small execution of a code with in a process. Thread also known as light weighted process. Light weighted means it takes less memory or resources of our machine.

Generally process is heavy.

Inside a process thread will execute.

Thread Process

Processor

Some language are process base programming language

Some language are thread base programming language

Java is thread base programming language.

In Java inside a main always one default thread will execute.

If we want to find default thread details in java

Thread is a pre-defined class part of lang package. Inside Thread class pre-defined method is currentThread(). This method is static so we can call this method with help of class name.

Thread.currentThread();

This method return type is same class reference.

Thread t = Thread.currentThread();

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

Main -🡪 name of the thread

5🡪 priority of the thread

Main -> group of the thread.

By default name of the thread is main, default priority is 5 and group is main.

We can set the priority using setPriority method min priority is 1 and max is 10. We can set 1 to 10 range.

Thread.MAX\_PRIORITY ie 10

Thread.MIN\_PRIORITY ie 1

Thread.NORM\_PRIORITY ie 5

Multi tasking

We can do multi tasking using two ways

1. Using process base
2. Using thread base

Processor

2 min to process it

Task1 Task2 Task3

5 min 5 min 5 min

Multi tasking using thread is faster than multi tasking using process base.

In Java We can create more than one thread or user-defined thread using 2 ways

1. Extends Thread class
2. Implements Runnable interface

extends Thread class

1. First we have to make user-defined class and that class must be extends Thread class.
2. Now we have to create the Thread class reference. (Make the object of that class the class which extends Thread class).
3. Using thread class reference call thread class pre-defined method ie start(). At this point of time two thread are started as well as destroy.
4. start() method internally all run() which is a part of Thread class. That method doesn’t contains any logic.
5. So if you want to execute any custom logic then we have to override run method inside that class which extends Thread class and write the logic.

Implements Runnable interface

1. First we have to make user-defined class which implements Runnable interface.
2. Runnable is a interface which contains one method ie run() . Actually run() is method part of Runnable interface internally Thread class implements Runnable interface and override run() method with empty body.
3. So when class implements Runnable interface we have to override run method mandatory.
4. Now we have to create the reference of thread class. Create the thread class object and inside a constructor pass the object of that class which class implements Runnable interface reference.
5. Using the thread class reference we can call start() and start() method internally call run method.

If class extends Thread class that class can’t extends any other class for different purpose.

Because java doesn’t support multiple inheritance.

If class implements Runnable interface that class can extends any other class as well as implements more than one interface.

When class extends Thread class that class become heavy class because all method of Thread class are inherited. But when class implements Runnable interface that time class is light weighted.

Thread life cycle

sleep()

isAlive()

join

wait()

notify()

notifyAll()

Create the thread ----------🡪 Runnable State ------------🡪 Running State ---------------🡪 Destroy

obj1 or obj2 obj1.start() run() i=10 or j =10

ready to run logic

t1 or t2 t1.start() run() i=10 or j=10

ready to run logic

sleep() : This method is use to pause the thread flow. This method throw checked exception ie InterputedException. So you have to handle it.

isAlive() :This method is use to check thread is running or not. If thread is running we will get true else it return false.

join() method join method is use to join all child thread to parent thread. So parent thread will wait till child thread get destroy.

To work wait(), notify() and notifyAll() we have to create more than one thread in same memory.

Synchronization : synchronization is concept which help to lock the thread or block thread or allow to use all resources for only one thread at time.

To achieve synchronization in java we have use synchronized keyword. This keyword we can use with method or block. If we use method as a synchronized from beginning to end only one thread will execute. Inside a method if you want to execute few line as a multi threading and few line as a single thread then we can use more than one synchronized block.

wait() , notify() and notifyAll()

wait() : This method is use to make the thread to wait or suspend with some conditions.

notify() : This method is use to callback waited thread or resume waited thread. Notify can resume only one thread. notifyAll() can resume more than one thread.

1. Wait(), notify() and notifyAll() method are belong to Object they are not a part of Thread class.
2. Notify() can be work only if more than one thread created in same memory.
3. Method must be synchronized.

Wait

Notify()

15-03-2022

**Lambda Expression**

Lambda expression by default return the value without return keyword.

**package** com;

**interface** Operation {

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

}

**interface** FindLargest{

**public** **boolean** find(**int** x, **int** y);

}

**class** OperationImp **implements** Operation{

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

**return** x+y;

}

}

**public** **class** LambdaExpressonExample {

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

Operation op1 = (x,y)->x+y;

System.***out***.println(op1.add(100, 200));

Operation op2 = (a,b)->a+b;

System.***out***.println(op2.add(100, 200));

Operation op3 = (**int** x, **int** y)->x+y;

System.***out***.println(op3.add(100, 300));

Operation op4 = (**int** x, **int** y)-> {

**int** sum= x+y;

**return** sum;

};

System.***out***.println(op4.add(1, 2));

FindLargest lf1 = (**int** x, **int** y)->x>y;

System.***out***.println(lf1.find(10, 5));

FindLargest lg2 = (**int** x, **int** y)-> {

**if**(x>y) {

**return** **true**;

}**else** {

**return** **false**;

}

};

System.***out***.println(lg2.find(5, 10));

}

}

Stream API :

Stream API is part of function package and function sub package of util package.

Pre-defined functional interfaces.

javap java.util.function.Function

Function

apply(): it take T as a parameter and return R value.

Supplier

Get() : it doesn’t take any parameter but return T value.

Predicate

test() : take T as a parameter and return boolean value.

Consumer

accept() take T as parameter and no return type.

Stream API : Collection framework is memory data structure. It takes memory while doing some operation on data.

Stream API help to load the data from collection framework on demand and we can do any operation temporary on data.

Source

Array -🡪 Stream --🡪 IO1 🡪IO2🡪O3--🡪Terminal operator

Or

Collection

Intermediate operator return type is stream itself. And terminal operator return type is non stream mean may be void or any primitive value.