**Day 1 : 11-01-2020**

**Phase 1 :**

Agile

Git

Java : Basic Programming

Java OOPS Concept

Exception handling

Multithreading

File Handling

Collection Framework and Data Structure

Maven

**Agile Methodology**

Application or Software

**SDLC : Software Development Life cycle**

90’s

**Waterfall Model**

1. Requirement Gathering and documentation
2. System design (look and feel)
3. Implementation or functionality using any language
4. Testing
5. Run or deploy or delivery
6. Maintenance

**Agile Methodology**

Moving fast, be flexible and response the change according to client requirements.

**Agile is concept.**

**Agile manifesto 4 :**

1. Individual and interaction -- > Process and tools
2. Working product – Comprehensive documentation
3. Customer Collaboration
4. Responding to changes

12 Agile Principles

1. Customer Satisfaction
2. Welcome changes
3. Deliver frequently
4. Work together : DevOps
5. Motivated teams
6. Face – to – Face

Etc

Implementation of Agile

1. XP
2. Lean
3. Scrum
4. Crystal

**What is Scrum :**

Agile is a set of methods and practise that focuses on iterative developments.

Scrum :

1. Product Owner : Interact with the client. It maintain the **product backlogs**. (functionality of the projects).
2. Scrum Master : The scrum master helps to team and apply the scrum to obtains business. He is only responsible for the scrum process.
3. Scrum Team : They are cross platform developers.

7 to 9 or 10 members

Sprints : 2 to 4 week

Daily they conduct scrum meeting : 10 to 15 minutes

**Spring backlogs**

**Git**

**Centralized Repository**

GitHub

Code Commit : AWS

GitLab

Azure

Google cloud

First create the Folder

Then open git bash terminal

**git init**

**git add filename.txt**

or

**git add .**

When we add the file using git add then file is present in git **staging area.**

**Staging area :** before commit the file the local repository

**git commit –m “message”**

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

**The initial name of the Java is Oak. Java developed in 1991.**

**But in Nov 1995 they rename from Oak to Java.**

**It was belong to Sun Microsystem but not it is a part of Oracle.**

**Java was developed by James Gosling and Team.**

**Version of Java**

**1.0, 1.2. Java 5, 6, 7 ---🡪 Java 8, 9, 10,11,12,13, 14 Version**

**Features of Java**

1. **Java is Simple : Pointer, Memory Management, Virtual Keyword, Friend functions, destructor etc.**
2. **Pure object oriented programming language :**
3. **Platform independent programming language.**
4. **Compiler and Interpreter**
5. **Exception Handling**
6. **Multi threading**

**Open the command prompt**

**Then**

**java –version**

**javac : java compiler**

**java : java interpreter**

**class syntax**

**class className {**

**fields or variables;**

**methods;**

**main (pre-defined methods)**

**}**

In Java ClassName always follow Pascal Naming rules

1. If class contains one word then first letter of name must upper case.
2. If class contains more than one word. Then each word first letter must be upper case.

Example

Demo

Test

Employee

EmployeeDetails

ManagerInfo

To compile the Java program

**javac classname.java**

To run the program

**java className**

**Demo.java**

class Demo {

public static void main(String args[]) {

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

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

System.out.printf("Welcome Java...");

}

}

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

Data types mainly divided into two types.

1. Primitive data types
2. Non primitive data types or reference data types.

**Primitive data types:** This type of data type is use to store only value.

8 types

1. byte 1 byte
2. short 2
3. int 4
4. long 8 : it is use to store value without decimal point
5. float 4
6. double 8 : it is use to store the value with decimal point
7. char 2 : any single character
8. boolean 1bit : true or false.

**Data type example**

class DataTypeDemo {

public static void main(String args[]) {

int a=10;

double b = 20.20;

char c ='$';

boolean res = true;

System.out.println(a);

System.out.println(b);

System.out.println(c);

System.out.println(res);

}

}

**Type casting**

Converting one type to another is known as type casting.

2 types

1. Implicit : Internally convert
2. Explicit : We have to covert

Int family

----------------------🡪Implicit type casting -------------------🡪

byte short int long

🡨----------------------Explicit type casting -----------------------

Int family type casting

class TypeCasting {

public static void main(String args[]) {

byte a =10;

short b=a; //Implicit type casting

System.out.println(a);

System.out.println(b);

short c = 10;

// (type)variableName;

byte d = (byte)c; //Explicit type casting

System.out.println(c);

System.out.println(d);

}

}

**Int and float type casting**

----------------🡪 Implicit ---------------🡪

int float

🡨-------------Explicit ----------------------

By default every decimal number in java double consider. So double data type size is 8 types. Which we can’t store in float ie 4 byte.

class TypeCasting1 {

public static void main(String args[]) {

int a=10;

float b=a; //Implicit type casting

System.out.println(a);

System.out.println(b);

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

float c = (float)10.10;

int d = (int)c;

System.out.println(c);

System.out.println(d);

}

}

**Operator**

1. Arithmetic Operator : +, -, \*, /, %(find the remainder)
2. Relational Operator : >, >=, <, <=, ==, !=
3. Logical operator : &&, ||, !
4. Assignment operator : =
5. Increment and decrement : ++ it increment the value by 1 and – it decrement the value by 1
6. **instanceOf : it is use to check the reference type.**

If statement

1. simple if

if(condition) {

if true block;

}

1. if else

if(condition) {

if true block

}else {

if false block

}

1. nested if : if within another if

if(condition) {

if(condition) {

**true block**

}else {

**false block**

}

}else {

if(condition) {

**true block**

}else {

**false block**

}

}

1. if else if or if ladder

if(condition1) {

Block1

}else if(codition2) {

Block2

}else if(codition3) {

Block3

}else {

Else block

}

1. switch statement : in switch statement user can decide which block you want to execute.

Syntax

switch(variableName) {

case v1:block1;

break;

case v2:block2;

break;

case v3:block3;

break;

case v4:block4;

break;

default:defaultblock

break;

}

switch, default, case and break are keywords.

varibleName must be type of int or char or String.

class SwitchStatement {

public static void main(String args[]) {

int n =4;

switch(n) {

case 1:System.out.println("1st block");

break;

case 2:System.out.println("2nd block");

break;

case 3:System.out.println("3rd block");

break;

default:System.out.println("Wrong choice");

break;

}

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

}

}

**Looping :** It is use to iterate the records one by one

Initialization : start position and end position

Conditions

Do the task

Increment or decrement

While loop

Do while loop

For loop

For each loop or enhanced loop

**class Loop {**

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

**/\*System.out.println("While loop - Entry loop ");**

**int i=1;**

**int n=10;**

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

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

**i++;**

**}**

**\*/**

**System.out.println("Do while - Exit Loop");**

**int i=1;**

**int n=10;**

**do {**

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

**i++;**

**}while(i>=n);**

**}**

**}**

**Day 2 : 12-01-2020**

**For loop**

**Syntax**

**1 2 4**

**for(initialization;condition;increment/decrement) {**

**for body; 3**

**}**

For loop Example

class ForLoop {

public static void main(String args[]) {

System.out.println("For loop example");

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

System.out.println(i);

}

System.out.println("Finish");

}

}

**Non primitive date type or reference data types** : It is use to store value as well as reference of another data types.

4 types

1. array
2. class (pre-defined class or user-defined class)
3. interface (pre-defined interface or user-defined interface)
4. enum (per-defined or user-defined enum)

**array:** array is user-defined data or also known as reference data type which is use to store collection of elements or value of same types.

int a;

a=10;

a=20;

syntax

datatype arrayName[];

int abc[]; // array declaration

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

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

**For each loop or enhanced loop**

Syntax

for(datatype variableName : arrayName) {

}

**For loop**

class ArrayDemo {

public static void main(String args[]) {

int abc[];

int xyz[]={10,20,30,40,50,60,100,130,150,200,250};

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

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

System.out.println("Size of array "+xyz.length);

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

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

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

}

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

for(int n : xyz) {

System.out.println(n);

}

}

}

Another For each example

class ArrayDemo {

public static void main(String args[]) {

int abc[];

int xyz[]={10,20,30,40,50,60,100,130,150,200,250};

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

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

System.out.println("Size of array "+xyz.length);

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

for(int i=2;i<8;i=i+2) {

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

}

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

for(int n : xyz) {

System.out.println(n);

}

}

}

**Creating the memory size for the array**

Syntax

datatype arrayName[]=new datatype[size];

int abc[]=new int[10];

Example

class ArrayDemo {

public static void main(String args[]) {

int abc[]=new int[10];

System.out.println("Size of array is "+abc.length);

abc[0]=100;

abc[1]=200;

int flag = 123;

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

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

System.out.println(abc[2]);

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

abc[i]=flag;

flag = flag +10;

}

System.out.println("All elements are ");

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

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

}

}

}

Another Example

class ArrayDemo {

public static void main(String args[]) {

int a=4; // 4 bytes

int abc[]=new int[5];

abc[0]=100;

}

}

int abc[]; one dimensional array

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

Two dimensional array

class ArrayDemo {

public static void main(String args[]) {

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

System.out.println("Display values");

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

System.out.println(abc[0][1]);

System.out.println(abc[1][0]);

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

System.out.println(abc[2][0]);

System.out.println(abc[2][1]);\*/

for(int i=0;i<3;i++) {

for(int j=0;j<2;j++) {

System.out.print(abc[i][j]);

System.out.print(" ");

}

System.out.println();

}

}

}

Creating the memory size for two dimensional array

int abc[][]=new int[5][3];

**OOPs**

Object Oriented Programming

Procedure language

Limitation

1. They are not working on real time scenario.
2. In procedure language they given important to function rather than variable.
3. Variable are not secure then can move from one function to another function freely. (public or private keywords are not there.)
4. We can’t reusability.

**object :** any real world entity.

**State or properties have age,name,color,h,w etc**

Person

**Behaviour do/does walking, talking, sleeping,**

**teaching**

Laptop

Pen

Animal

State or properties wheel, color, price etc

Car

Behaviour start, appliedGear, moving, stop

Bank

Etc

**object is concept.**

**class :**

**class is known as blue print of the object.**

**class is also known a template of object.**

**class is a user-defined data types which help to implement or create the object.**

**array : array is use to store same type of value.**

**C/C++ structure : structure is user-defined data type which help to store different type of values.**

**class : class is user defined data type which help to store different types of value. Class can hold set function or methods.**

class Car {

int wheel;

String color;

float price;

void start() {

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

}

void appliedGear() {

System.out.println("Applied the Gear....");

}

void moving() {

System.out.println("Car is moving....");

}

void stop() {

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

}

}

class Test {

public static void main(String args[]) {

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

Car innova = new Car(); // heap memory

innova.start();

innova.appliedGear();

innova.moving();

innova.stop();

}

}

**Type of variable or fields**

3 types

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

class Car {

int wheel;

String color;

float price;

void displayInfo() {

String temp="Car Info"; // local variable

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

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

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

System.out.println("Temp "+temp);

}

}

class Test {

public static void main(String args[]) {

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

Car innova = new Car(); // heap memory

//innova.displayInfo();

innova.wheel = 4;

innova.color = "Gray";

innova.price = 1400000;

innova.displayInfo();

Car santro = new Car(); //heap memory

//santro.displayInfo();

santro.wheel = 4;

santro.color ="Black";

santro.price = 850000;

santro.displayInfo();

}

}

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

Pts

1. Constructor have same name as class itself.
2. Don’t provide return type for constructor not even void also.
3. Constructor no need to call explicitly. It will call automatically when we create the object.

In the life of object if we want to execute any task only one time that type of task we have to write inside a constructor.

In the life of object if we can to execute any task more than one time that type of task we have to write inside methods.

Constructor Example

class Employee {

int id;

String name;

float salary;

Employee() {

id =123;

name="Unknown";

salary = 8000;

}

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

this.id = id;

this.name = name;

this.salary = salary;

}

void setValue(int id, String name, float salary) {

this.id = id;

this.name = name;

this.salary = salary;

}

void display() {

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

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

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

}

}

class Test {

public static void main(String args[]) {

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

Employee emp2 = new Employee(); emp2.display();

Employee emp3 = new Employee(1,"Ravi",12000); emp3.display();

Employee emp4 = new Employee();

emp4.setValue(2,"Ramesh",14000);

emp4.setValue(3,"Rajesh",16000);

emp4.display();

}

}

Object and class

4 pillar

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

class :

class Employee {

int id;

String name;

float salary;

void display() {

}

}

class Employee {

private int id;

private String name;

private float salary;

void setValue(int id, String name, float salary) { //helper methods

this.id = id;

this.name = name;

if(salary <0 ) {

this.salary = 8000;

}else {

this.salary = salary;

}

}

void display() {

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

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

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

}

}

class Test {

public static void main(String args[]) {

Employee emp1 = new Employee();

//emp1.id=100;

//emp1.name ="Ravi";

//emp1.salary = -12000;

emp1.setValue(100,"Ravi",12000);

emp1.display();

}

}

In Java all instance variable must be private and provide helper method to set the value with term and conditions.

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

The gold of inheritance is use to do re-usability.

class OldClass { super class, base class or parent class

state

behaviour

}

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

state

behaviour

}

Simple Example of Inheritance

class A {

void dis1() {

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

}

}

class B extends A{

void dis2() {

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

}

}

class Test {

public static void main(String args[]) {

A obj1 = new A();

obj1.dis1();

B obj2 = new B();

obj2.dis2();

obj2.dis1();

}

}

**Types of Inheritance**

1. Single Inheritance : One super class and one sub class

class A { }

class B extends A { }

1. Multilevel Inheritance : One super class and n number of sub class 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 class directly connected to super class

class A {}

class B extends A {}

class C extends A{}

class D extends A {}

1. Multiple inheritance : More than one super class and one sub class

class A {}

class B {}

class C extends A, B {} : But Java doesn’t support this type of inheritance using classes. It can support indirectly using interface.

**Oops relationship**

1. Manager Is a relationship Employee
2. Employee/Manager/Programmer Has a relationship Address

class Employee { Generics

id,name,salary primitive property

}

class Manager extends Employee{ Specific

numberOfEmp

Address add = new Address();complex property

}

class Programmer extends Employee {

projectName;

}

class Address {

city, state, pincode

}

**Day 3 : 13-01-2020**

Employee Management System

Is a relationship

Employee, Manager, Programmer, Developer, Clerk, CEO etc

Banking Application

Has relationship

Customer, Bank, Account, Transaction etc.

Bank

Customer

custId

custName

age

Account

Accno

Amount

typeofAccount

Transcation

Tid

Date

Time

typeOfTran

Online Shopping

Customer, Product, Clerk, Admin etc

Single Inheritance Example

class Employee {

private int id;

private String name;

private float salary;

void setValue(int id, String name, float salary){

this.id = id;

this.name = name;

this.salary = salary;

}

void display() {

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

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

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

}

}

class Manager extends Employee{

private int numberOfEmp;

void setMgrValue(int numberOfEmp){

this.numberOfEmp = numberOfEmp;

}

void displayMgr() {

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

}

}

class InheritanceDemo {

public static void main(String args[]) {

Employee emp1 = new Employee();

emp1.setValue(1,"Ravi",12000);

emp1.display();

Manager mgr1 = new Manager();

mgr1.setValue(2,"Ramesh",45000);

mgr1.setMgrValue(10);

mgr1.display();

mgr1.displayMgr();

}

}

Hierarchical Inheritance example

class Bike {

void speed() {

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

}

}

class Pulsar extends Bike {

void color() {

System.out.println("Black");

}

}

class Honda extends Bike {

void color() {

System.out.println("Gray");

}

}

class InheritanceDemo {

public static void main(String args[]) {

Pulsar pu = new Pulsar();

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

Honda hh = new Honda();

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

}

}

**Polymorphism**

One name many forms or one concept many implementation.

2 types

Compile time polymorphism Run time polymorphism

javac java

compile interpreter

static binding dynamic binding

early binding late binding

Method Overloading Method Overriding

In C++ In C++

Function Overloading Function Overriding

Operator overloading

Java doesn’t support

Operator overloading

Compile time

Polymorphism

Method Overloading : The method have same name but different parameter list ( type of parameter list(data type) or number or parameter list must different). But return type must be same is known as Method Overloading.

**Method Overloading Example**

class Cal {

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

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

}

void add(float a, float b) {

System.out.println(a+b);

}

void add(String s1, String s2) {

System.out.println(s1+s2);

}

}

class MethodOverloading {

public static void main(String args[]) {

Cal cc = new Cal();

cc.add(1,2,3);

cc.add(10,20);

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

}

}

customerDetails(String fname,String lname,int age) {

}

customerDetails(String fname,int age) {

}

Run time polymorphism

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

To achieve the method overriding class must be in Inheritance.

(Minimum super class and sub class).

**Method Overriding Example**

class Bank {

void deposite() {

System.out.println("Generic Deposite Method Logic");

}

}

class Hdfc extends Bank {

void checkBalance() {

System.out.println("HDFC checkbalance method");

}

void deposite() {

System.out.println("Deposite Method Logic Provided by Hdfc");

}

}

class Sbi extends Bank {

void checkBalance() {

System.out.println("Sbi checkbalance method ");

}

}

class MethodOverriding {

public static void main(String args[]) {

Hdfc obj1 = new Hdfc();

obj1.deposite(); obj1.checkBalance();

Sbi obj2 = new Sbi();

obj2.deposite(); obj2.checkBalance();

}

}

Open terminal

**vim Demo.java**

**abstract**

**final**

**static**

abstract :

1. abstract is a keyword which 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 is known as abstract method.

abstract returnType methodName(parameterList);

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

abstract class className {

}

1. If any normal class extends abstract class that class must be provide the body for the all abstract methods belong to that class. That class can ignore if that class itself is a abstract class.
2. Abstract class we can’t create the object.
3. Abstract class can contains normal as well as abstract methods.

Means abstract class can contains zero or 1 or all abstract methods.

If we can to achieve partial abstraction

Hiding the internal implementation without knowing background details.

abstract class Orders {

abstract void payment();

}

class Paytm extends Orders {

void payment() {

System.out.println("Payment happend through PayTM");

}

}

class GooglePay extends Orders{

void payment() {

System.out.println("Payment happend through Google Pay");

}

}

class PhonePay extends Orders{

void payment() {

System.out.println("Payment happend through Phone Pay");

}

}

class MethodOverriding {

public static void main(String args[]) {

}

}

**final**

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

1. Final variable : if we want to declare the constant variable in java we have to use final keywords with variable.

final int A=10;

A=20; Error

1. Final method : if method is final we can’t override that methods but we can use that method.
2. Final class : if class is final we can’t extends that class or inherits that class.

static

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

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

1. Static method : if method is static we can call that method with the help of className

className.staticMethodName();

1. Static variable: if variable is a static we can assign the value for that variable with the help className

className.staticVariablename = value;

1. We can call static method through object also.
2. We can assign the value for static variable through object also.
3. Inside a static method we can access only static variable directly but inside non static method we can access both the type of variable ie static as well as non static.

Static Example

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");

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

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

}

}

class StaticDemo {

public static void main(String args[]) {

Abc.dis2();

Abc.b=10;

Abc obj1 = new Abc();

obj1.dis2();

obj1.b=20;

obj1.a=30;

obj1.dis1();

}

}

Every class contains only one static memory

Every class contains n number heap number how many object you created.

Static is like a global to all objects.

Employee

Id,name,salary -🡪 instance

MeetingTime 🡪 Static

Customer

CustId

CustName

Age

Interest 7.0 6.8 7.2 static

IFSC Code

class Abc {

final static String city=”Bangalore”;

int a;

static int b;

void dis1() {

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

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

}

}

class StaticDemo {

public static void main(String args[]) {

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

}

}

int a=10;

void abc() {

a=20;

}

void xyz() {

printf(“%d”,a);

}

void main() {

printf(“%d”,a);

abc();

printf(“%d”,a);

a=40;

xyz();

}