Class running notes

Full stack = Database layer + Business layer + Presentation layer

Business layer – Developer, Presentation – UI/UX designers, Database – Data scientists.

Presentation – HTML, CSS, JS. Business layer – Java/PHP/C#/Python, Database – MySQL, MongoDB.

Mysql – https://www.youtube.com/watch?v=7S\_tz1z\_5bA

Java Full Stack – Core Java, Java Hibernate, Spring, Spring Boot.

Java History:

Java is created by James Gosling. Java is released in 1991 in the name of Oak. In 1995, java 1.0 is publicly available. Oracle acquired it in 2010.

Java Features:

Platform Independent, OOPS, Simple, Robust, Secure, Distributed, Multi-Threading, Portable, High performance, Dynamic flexibility, Sandbox execution, Write once run anywhere, Power of compilation and interpretation.

Java converts your “.java” file into a binary file “.class” file.

Here in Java objects communicate with other java objects in the program.

Oops concepts- Abstraction, Encapsulation, inheritance, Polymorphism.

Java is a high-level language.

Pointers are less secure, so we don’t use it in high level languages like java, python.

Operator overloading and multiple inheritance are additional features.

Robust aka reliable language.

Exception handling is used to handle errors which are happened unknowingly, like dividing 0/0 or arrayoutofbound etc.

Java have an inbuilt memory allocation, so developers no need to care about the allocation and deallocation of the memory.

Thread is said to be the smallest unit of dispatchable code. Multi-threading is used to minimize the CPU sleeping time. A single process is divided into multiple threads.

JIT aka High performance, Just-IN-time.

Sandbox execution helps you to run other language code such as there are some methods similar to C and this helps us to execute C code in the java language.

JDK,JRE,JVM:

Java development kit, Java Runtime Environment, Java Virtual Machine

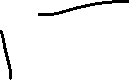
A compiler would convert .java file -> byte file aka .class file.

A .class file can be shared to any place and only requires java code.



Value

int x =5;



Variable  
identfier



Datatype

These are the keyword pre-defined in java. Keywords overview.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | continue | for | new | switch |
| assert | default | goto\* | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | import | public | throws |
| case | enum\*\*\*\* | instanceof | return | transient |
| catch | extends | int | short | try |
| char | final | interface | static | void |
| class | finally | long | strictfp\*\* | volatile |
| const\* | float | native | super | while |

double, float used for precision. pi= 3.14159265359. Accuracy and Precision is double > float.  
By default, number aka int, byte, float etc are initialized as 0. Boolean is initialized as false.

Final is used to never change the variable till the end of the program. This prevents overloading and overriding.

Finally is a part of the exception handling. Used to deal with bugs and errors. Exception handling: try, throw, throws, catch, finally.

goto is not used. goto is used to skip to the line of the program. It is widely used in fortron.

extends and implements are used to inheritance. Extends is for inheriting class to class and implements is for inheriting interface to class.

How many datatypes are there in Java:

Boolean (True/False) 1 bit > Byte (-127 to 127) 8bits > Short (-2^15 to 2^15 -1) 16bits > int (-231 to 231-1) 32bits > long (-263 to 263­-1) 64bits> float 16bits> double 32bits> char (Unicode) 4bits (“\u0000”-“\uffff”).

String belongs to object class.

Access Modifiers: Public, Private, Protected, Default.

Protected is related to packages. Private is not visible to any class even if it is inherited. Public is visible to everyone.

Static is available before the initiation. Static works on top of the runtime. Static works on the entire class and need not to be instantiated.

Synchronized is used in concept of multi-threading.

this is used as a current class instance.

com

siet

Naming conventions:

tns

Package is always small letters: com.tns.siet.

Class always start with CapitalWords like this aka camel case notation.

Variables and literals are initialized with a and small letters \_

Operators:

1. Arithmetic
2. Assignment
3. Bitwise
4. Logical
5. Relational

**Arithmetic:** +(Addition),-(Subtraction),\*(Multiplication),/(Division),%(Remainder or modulus)

**Assignment:** a=10, = is assigning a with value 10.

Likewise,

+= : b=20,a=10, b+=a // b=30

-= : b=20,a=10,b-=a // b=10

\*= : b=20,a=10,b\*=a // b=200

/= : b=20,a=10,b/=a // b=2

%= : b=20,a=10,b%=a // b=0

**Logical operator:**

&& Logical AND: If both are true, the answer is true. Or else False

|| Logical OR: If any one of the statements is true. The answer is true. If both are false, then answer is False.

! Logical NOT: True becomes False and False Becomes true.

Program, to check whether the user is above 21 and in couple, if the user is man and single, charge x amount and if the user is woman and single charge x amount.

boolean Couple=True

int age = 23

String gender = “Male”

if(age>21 && couple){

sysout(“They are in”);

}

if(age>21){sysout(“No”);)

if((gender ==”Male” || gender==”Female”) && Couple == False){sysout(“ Charge them X amount ”);}

Decision and Control Statement:

Control Statements:

If, else, elif, Switch

Example:

If: If a given condition is satisfied, the inner code block would be executed.

class DemoIf{

pvsm(s[]a){

int x=10,y=12;

if (x+y>20){

sysout(“Yahh! More than 20”);

}

}

}

Else: If the if condition is not satisfied, the code shifts to else block.

Class DemoAge{

pvsm(s[]a){

int age=20;

if (age>18){sysout(“Eligible to vote”);}

else{sysout(“Not eligible”)}

}

}

Else If: Checks other conditions if the first condition fails.

class CityDemo{

pvsm(s[] a){

String city= “Hyderabad”.

if (city== “Mumbai”) {sysout(“You can Join the company”);}

else if (city== “Hyderabad”) {sysout(“There is a work from home opportunity for 3 months.”)}

else if (city== “Bangalore”) {sysout(“We offer you work from home.”)}

else if(city == “Chennai”) {sysout(“We offer house rent allowance and petrol allowance);}

else {sysout(“Email us at [xxxxxxx@xxx.com](mailto:xxxxxxx@xxx.com), Let’s figure it out.”)}

}  
}

ALGORITHM:

Diagram

Description automatically generated

Text, letter

Description automatically generated

Subtraction of two numbers:

Step1: Start

Step2: Declare three variables such as num1, num2 and Sub

Step3: Read the value of num1 and num2

Step4: Do subtraction of num1 and num2 and assign to Sub

Sub=num1-num2

Step5: Display Sub

Step6: Stop

­­

Display Sub

Sub<-num1-num2

Read num1, num2

Declare num1 and num2 and sub

Multiplication of two numbers:

Step1: Start

Step2: Declare three variables such as num1, num2 and Mul

Step3: Read the value of num1 and num2

Step4: Do subtraction of num1 and num2 and assign to Mul

Mul=num1\*num2

Step5: Display Mul

Step6: Stop

­­

Display Mul

Mul<-num1-num2

Read num1, num2

Declare num1 and num2 and Mul

Division of two numbers:

Step1: Start

Step2: Declare three variables such as num1, num2 and Div

Step3: Read the value of num1 and num2

Step4: If num2 != 0, go to step 5, else go to step 7

Step5: Do subtraction of num1 and num2 and assign to Div

Div=num1/num2

Step6: Display Div

Step 7: Display “Error”

Step6: Stop

Display “Error”

Div<-num1/num2

If num2!=0

Display Sub

Read num1, num2

Declare num1 and num2 and Div

Modulus of two numbers:

Step1: Start

Step2: Declare three variables such as num1, num2 and Mod

Step3: Read the value of num1 and num2

Step4: Do subtraction of num1 and num2 and assign to Mod

Mod=num1%num2

Step5: Display Mod

Step6: Stop

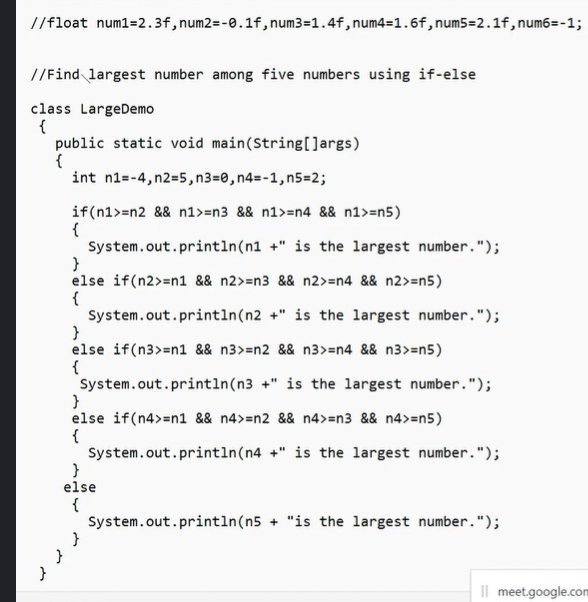
­­

Display ModD

Read num1, num2

Declare num1 and num2 and Mod

Mod<-num1-num2



Inheritance:

class A{

int a=10,b=20,c;

void add(){

c=a+b;

System.out.println("ADD"+c);

}

}

class B extends A{

void mul(){

System.out.println("Mul"+a\*b);

}

public static void main(String[] args){

B obj= new B();

obj.add();

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

obj.mul();

}

}  
  
ADD30

Mul200  
  
  
1. Single Inheritance

2. Multilevel inheritance

3. Hierarchical Inheritance.

Polymorphism:

1. Compile time: OverLoading – Method name is same but params are diff

void add(){

}

void add(int num, int num2){

}

void add(float x, float y){ }

1. Run time: Over Ridding – Method name is same and params are also same

class A{

void say(){

System.out.println(“Hello From A”);

}

}

class B extends A{

void say(){

System.out.println(“Hello From B”);

}

}

class Demo(){

pvsm(){

B obj=new B();

Sysout(obj.say());

}

}

Encapsulation: Binding the member functions and member variables together in a single form or a block

Abstraction: Hiding the functionality from the user.

Inheritance: Acquiring the properties of one class to another

Polymorphism: Having multiple forms for a single function.

Static:

Static method will never take memory, simillarly like class. If it is not static, we need to create an instance(Obj) to access the method or variable. Static methods exists only in class, which can be a static member variable or a data function.

package apr13;

class StatsTest{

    static int x=10;

    int y=100;

    public void method1() {

        System.out.println("Public method check");

    }

    static void method2(){    //Static methods exists only in class, which can be a static member variable or a data function.

        System.out.println("Static Method Check");

    }

}

public class statDemo {

    public static void main(String[] args) {

        StatsTest obj = new StatsTest();

        obj.method1();

        obj.x=100;

        obj.y=20;

        System.out.println(StatsTest.x+"     "+obj.y);

        StatsTest.method2(); // Correct way to access a static variable.

        // obj.method2(); // No need to use any object to access it.

        // statDemo.method1(); // This doesnt work like static.

    }

}

package apr13;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.\*;

/\*

 \*  To take input from the user you can do it by

 \*  1. Scanner Class. ( Predefined ) in java.util package.

 \*      - Scanner obj\_name = new Scanner(System.in);

 \*      - Methods:

 \*          nextInt(); - int

 \*          nextLine(); - String

 \*          nextFloat(); - Float

 \*          nextBoolean(); - True or False

 \*          nextDouble(); -  Double

 \*      - Use import java.util.Scanner;

 \*      - If WRONG INPUT is given in the time of fetching the value. We can get an Exception.

 \*          - if at the time of nextFloat() -> You entered a String -> ERROR!

 \*

 \*

 \*  2. BufferedReader class in java.io.BufferedReader package. and InputStreamReader in java.io.InputStreamReader package.

 \*      - BufferedReader obj\_name = new BufferedReader(new InputStreamReader(System.in));

 \*      - Methods:

 \*          readLine(); - String

 \*          read(); - int

 \*          read(char[]); - char[]

 \*      - Use import java.io.BufferedReader; and import java.io.InputStreamReader;

 \*      - If WRONG INPUT is given in the time of fetching the value. We can get an Exception.

 \*      - if at the time of read() -> You entered a String -> ERROR!

 \*/

 /\*

    For output we have 4 ways:

    1. System.out.println();

    2. System.out.print();

    3. System.out.printf();

    4. System.out.format();

    But we will use only 2 ways:

    1. System.out.println();

    2. System.out.printf();

  \*/

public class InnOut {

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

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the value of int: ");

        int a=sc.nextInt();

        System.out.println("Enter the value of Boolean: ");

        boolean b= sc.nextBoolean();

        System.out.println("Enter the value of char: ");

        char c= sc.next().charAt(0);

        System.out.println("Enter the value of double: ");

        double d =sc.nextDouble();

        System.out.println("Enter the value of float: ");

        float f = sc.nextFloat();

        sc.nextLine();

        System.out.println("Enter the value of String: ");

        String str= sc.nextLine();

        System.out.println( "Scanner");

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

        System.out.println("float: "+f);

        System.out.println("double: "+d);

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

        System.out.println("char: "+c);

        System.out.println("String: "+str);

        // BufferedReader

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        System.out.println("Enter the value of int: ");

        a=br.read();

        System.out.println("Enter the value of String: ");

        str= br.readLine();

        System.out.println("Buffered Reader");

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

        System.out.println("String: "+str);

        br.close();

        sc.close();

    }

}

**Constructor:**

The main purpose of constructor is to initialize the object.

Every java class has a constructor, even if you dont specify it.

A constructor is automatically called at the time of object crea

Example: A obj=new A(); here new A() is a constructor which is d

Constructor never return any return type including void.

\*

\*

Syntax:

<access\_modifier> class <class\_name>{

<class\_name>(){

// constructor body

} // This is a constructor

<access\_modifier> <return\_type> <method\_name>(<paramet

// method body

}

}

\*

\*/

There are two types of constructors:

1. Default constructor

2. Parameterized constructor

\*

1. Default constructor:

A constructor which is defined by default by the compiler.

It has no parameters.

It has no return type.

Syntax:

<access\_modifier> class <class\_name>{

<class\_name>(){

// constructor body

} // This is a constructor

}

\*

2. Parameterized constructor:

A constructor which is defined by the user.

It has parameters.

It has no return type.

Syntax:

<access\_modifier> class <class\_name>{

<class\_name>(<parameters>){

// constructor body

} // This is a constructor

}

\*

class A{

    public int x;

    A(int x){

        System.out.println("Value passed to constructor "+x);

        this.x=x;

    }

    A(){

        // This is a constructor. It can't return a value.

        System.out.println("I am a default constructor ");

        System.out.print("Initializing var x as 0");

        this.x=0;

        // return 10; // error: constructor cannot return a value

    }

    void show(){

        System.out.println("Value of x is "+x);

    }; // This is a normal method. It can return a value.

}

public class constructor {

    public static void main(String[] args) {

        A obj=new A();

        A obj2=new A(5);

        obj.show();

        obj2.show();

    }

}

Output:

I am a default constructor

Initializing var x as 0Value passed to constructor 5

I am a normal method

Value of x is 0

I am a normal method

Value of x is 5

**This:**

This is used to avoid confusion between the local and instance variables.

Variable inside the class outside the function is called instance variable.

Variable inside the function is called local variable.

If the local variable and instance variable have the same name then we can use "this" keyword to avoid confusion.

// example:

class A{

    int a=10;

    A(int a){

        System.out.println("Value that you want to use for a " + a);

        System.out.println("Value of instance variable a "+this.a);

    }

}

**Polymorphism**

1. **Compile time –** Executing program at the time of compilation. Method Overloading.

void show(int a){}

void show(int x, int y){}

void show(double m, float n){}

void show(int c, int d, int e){}

* By changing numbers of arguments.
* By changing the data type.

Syntax:

Returntype methodname(para1)

1. **Runtime -**

Method Overriding

Syntax:

Class A{

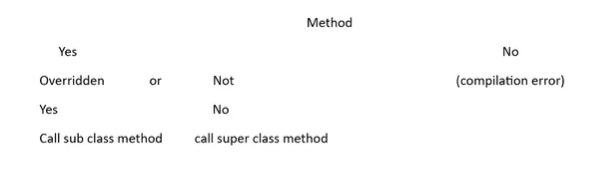
void show(){}

}

Class B extends A{

void show(int a, int b){}

}



**Abstraction: Refer apr26**

Hides implementation details from the user. Only some set of services or initiation of functions are visible to the user.

Two ways to achieve abstraction:

1. Abstract Class
2. Interface
3. Abstract Class:

* Aka Incomplete class.
* Creating a class with a keyword “Abstract” behind it.
* You cannot create an object using abstract class.
* We can create a reference variable by extending abstract class to other class.
  + abstract class Animal{}

class Dog extend Animal{}

main(){

Dog d = new Animal(); // Reference.

}

* We can define abstract or non-abstract methods can also exist aka Final, Constructor of abstract class, static methods in the Abstract Class.
* If abstract method is defined in the class, it must be overridden in the child class or else it would give us an error.
* If a class contain partial implementation, then we must define the class as an abstract class.
* Partial Implementation in the sense: If we don’t know how to implement the functionality of a function but you want to define it.
* If there is an Abstract Method then class must be an Abstract class. But Abstract Class may or maynot need to contain Abstract method, It can contain any kind of methods.
* Abstract Methods have no body.
* Abstract method must be public or protected
* Must be overridden.
* Abstract method will not have a body.

abstract class vehicle{

abstract void wheels(); // when we cannot define the properties for every sub class

}

APR 27

Interface:

1. Interface is nothing but an abstract class but every method would have no body.
2. Interface is a bridge between client and developer.
3. No need of initializing methods with abstract keyword and no access modifier.
4. We “implement” Interface block to other classes.
5. Interface only contains abstract methods.
6. Every requirement by “Client” would be mentioned in the interface and it is developers’ responsibility to build it.
7. Every method in the interface must be overridden or implemented in the “implements” class.

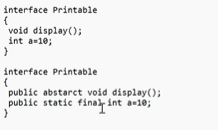
Syntax:

interface A{

void push();

void get();

}



Multiple interface and Hybrid Interface is possible, but Multiple and Hybrid Class is not possible.

Buffered Reader:

A quick and faster way to retrieve data from the console.

Works but extending reader class. Here we read InputStreamReader aka binary conversion of the input.

1. 65 – 101001
2. 97 – 111011

It reads byte stream to a character stream.

Syntax:

BufferedReader br = new BufferedReader(new InputStreamReader(new System.in));

BufferedReader br-> InputStreamReader 00101001 -> String name = A;

OOPS Revision:

1. Class
2. Objects
3. Inheritance
4. Abstraction
5. Encapsulation
6. Polymorphism
7. Class – Blue print of Object. Collection of Data variables and member methods.
8. Object - Instance of class.
9. Inheritance – A concept where a property of class is acquired by another class.
   1. Simple / Single –

Simple Inheritance is nothing but a single child class have a single parent class. This is a basic type of inheritance.

Syntax:

class child extends parent {

//code here.

}

Program:

class Employee{

int salary = 100000;

Employee(){

this.salary=100000;

}

Employee(salary) {

this.salary=salary;

}

int bonus () {

return salary\*(20/100)

}

}

class Programmer extends Employee{

Programmer(){

super();

}

Programmer(salary){

super(salary);

}

void type(){

sout(“Im a programmer”);

}

int bonus(){

return super.salary\*(30/100)

}

}

public class Demo{

pvsm(){

Programmer p = Programmer(400000);

p.type();

p.bonus();

}

}

* 1. Multilevel Inheritance

Multilevel Inheritance is nothing but a child class have a parent class and

that parent class have another parent class.

Syntax:

class child1 extends parent {

//code here.

}

class child2 extends child1 {

//code here.

}

Program:

class Pet {

void eat() {

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

}

}

class Cat extends Pet {

void sound() {

System.out.println("Meow! ");

}

}

class ScottishEarFold extends Dog {

void describe() {

System.out.println("Scottish Ear Fold have a round and cute face");

}

}

class TestInheritance2 {

public static void main(String args[]) {

ScottishEarFold s = new ScottishEarFold();

s.sound();

s.eat();

s.describe();

}

}

c. Hierarchical Inheritance –

Hierarchical Inheritance is nothing but a parent class is inherited by multiple

child classes.

Syntax:

class child1 extends parent {

//code here.

}

class child2 extends parent {

//code here.

}

Program:

class College{

College(){

System.out.println("Sreyas");

}

}

class CSE extends College{

CSE(){

System.out.println("CSE");

}

}

class ECE extends College{

ECE(){

System.out.println("ECE");

}

}

class IT extends College{

IT(){

System.out.println("IT");

}

}

class TestInheritance3{

public static void main(String args[]){

CSE c = new CSE();

ECE e = new ECE();

IT i = new IT();

}

}

Super Keyword:

Super keyword is used to access the data members and methods of parent class when both

parent and child class have same data members and methods.

Syntax:

super.data\_member\_name;

super.method\_name();

Program:

class Resturant {

String name = "Heavens";

void display() {

System.out.println("Welcome to " + name);

}

}

class ResturantBranch extends Resturant {

String name = "Heavens NYC";

void display() {

super.display();

System.out.println("Welcome to " + super.name);

}

}

class TestSuper1 {

public static void main(String args[]) {

ResturantBranch r = new ResturantBranch();

r.display();

}

}

1. Polymorphism:

Method Overloading:

Method Overloading is nothing but a method have same name but different parameters.

Syntax:

void method1(){}

void method1(args..){}

int method1(args..){}

float method1(args){}

Example:

class Addition{

int a,b;

a=10;

b=10;

int add(){

return a+b;

}

int add(int x){

return x+a+b;

}

float add(float x, float y){

return x+y;

}

}

public class TestAdd{

public static void main(String[] args){

Addition a= new Addition();

System.out.println(a.add())

System.out.println(a.add(10))

System.out.println(a.add(10.4,11.9))

}

}

Method Overridding:

Method Overridding is also known as Runtime polymorphism.

Method Overridding is nothing but a method have same name with same parameters is

overriden at the time of execution.

Syntax:

class A{

void method1{}

}

class B extends A{

@override;

void method1{}

}

Example:

class Employee{

void work(){

System.out.println(" This is a default job. ")

}

}

class Programmer extends Employee{

void work(){

System.out.println(" Im a programmer ")

}

}

public class TestOverRide{

public staitc void main(String[] args){

Programmer p = new Programmer();

p.work();

}

}

Encapsulation:

Binding the data members and data variables in a single object or an entity is called Encapsulation.

Abstraction:

Hiding the implementation of function and showing only functionality to the user.

Example: