**Day 1 : 17-01-2022**

**C**

**Procedure language**

**Oops**

**object : object is a any real world entity**

**properties or state --🡪 have 🡪 fields / variables**

**person**

**behaviour --🡪do/does -🡪 methods / functions**

**object is a concept.**

**bank**

**animal**

**car**

**mobile**

**etc**

**int a;**

**a int;**

**a number;**

**class : blue print or template of object. class is also known as user-defined data type which help to create the object. with the help of class we can define the object. class is a implementation of object.**

**class Car {**

**int wheel = 4;**

**String color;**

**float price;**

**void start() {**

**coding ….**

**}**

**void appliedGear() {**

**}**

**void moving() {**

**}**

**void stop() {**

**}**

**}**

**Car innova = new Car(); memory created (heap memory)**

**innova.start();**

**Car santro = new Car(); another car memory created..**

**Car bmw = new Car();**

**C+ OOPs = C++ (C with classes )**

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

**Java Version 1.0 , 1.1,1.2 1.8, Java 9, 10,11,12, Java 17**

**Open the cloud lab**

**Open the terminal**

**java –version**

**syntax of class**

**class ClassName {**

**variable declaration**

**method declaration**

**}**

class Demo {

public static void main(String args[]){

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

}

}

Save program with extension .java

**Demo.java**

**Open the terminal**

**javac Demo.java (To compile the program)**

**java Demo (To run the program)**

**Day 2 : 19-01-2022**

**Eclipse IDE**

**Package is a collection classes and interfaces.**

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

**2 types**

1. **Primitive data types : it is use to store only values.**
2. **Non primitive data types or reference data types : it use to store value as well as reference of another data types.**

**Primitive : 8 types**

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

**Syntax to declare the variable**

**datatype variableName;**

**datatype variableName=value;**

**int a=10;**

**type casting : converting from one data type to another data type is known as type casting.**

1. **Implicit type casting**
2. **Explicit type casting**

**int family**

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

**byte short int long**

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

**Explicit type casting**

**datatype variablename = (type)variabeName;**

**int to float**

**implicit --🡪**

**int float**

**🡨-- explicit**

**In Java Every decimal number by default double consider.**

**Operator :**

**Arithmetic Operator : +, -, \*, /, %**

**Conditional operator : > , >=, <, <=, ==, !=**

**Logical operator : &&, ||, !**

**Assignment operator : =**

**Increment and decrement : ++ , --**

**Conditional statement**

**If statement**

**If else**

**Nested if**

**If else if**

**Switch :switch statement help to execute the set of statement base upon the user requirement.s**

**Syntax**

**switch(variableName) {**

**case label1: block1;**

**break;**

**case label2: block2;**

**break;**

**case label3: block3;**

**break;**

**default : wrong block**

**break;**

**}**

**switch, case, default and break are keywords.**

**Variable can be type of int, char or string.**

**Looping : looping is use to execute the set of statement again and again till the condition becomes false.**

**Initialization : start and end position i=1,n=10**

**Condition true i<=n**

**Do the task hello**

**Increment or decrement. I++ or I = i+1**

**While loop**

**Do while loop**

**For loop**

**For each loop or enhanced loop : This type of loop we can use with array or collection of classes.**

**Taking the value through keyboards.**

**Scanner : Scanner is a pre-defined class which help to take the value through keyboards.**

**Syntax to create the Scanner class object.**

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

**Scanner pre-defined class part of util package.**

**import java.util.Scanner;**

**reference data type or non primitive data type**

1. **array**
2. **class (it may be user-defined or pre-defined) : String is a pre-defined class. it is also known as reference data type.**
3. **interface (it may be user-defined or pre-defined)**
4. **enum**

**by default every java program imported lang package.**

**Day 3 :**

**24-01-2022**

**int a;**

**Reference data types**

**array : array is type of reference data type which is use to store more than one value same type.**

**Syntax**

**datatype arrayName[];**

**declaration of array**

**int abc[];**

**initialization of array**

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

**size of array is 6. To access the array values we can use index position. Index position start from 0.**

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

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

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

**To find the size of the array we can use pre-defined fields**

**arrayname.length;**

**syntax to declare for each loop or enhanced loop**

**for(datatype variableName : arrayName) {**

**}**

**Creating the memory size for array**

**Syntax**

**datatype arrayName[]=new datatype[size];**

**int abc[]=new int[10]; java.**

**Int leng = abc.length; size 10, memory size 40 byte.**

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

**System.out.println(abc[9]); value is zero.**

**Int family -🡪0**

**Float family 🡪 0.0**

**Char 🡪 space (white space)**

**Char name =’ ’**

**Boolean 🡪 false**

**String 🡪 null**

**referenceName = null;**

**System.gc();**

**int abc[10]; in C or C++**

**printf(“%d”,abc[0]);**

**java provide auto garbage collector.**

**object and class**

**object : object is any real world entity or instance of a class.**

**Properties or state have variables / fields.**

**Person**

**Behaviour do/does functions / methods**

**Bank**

**Animal**

**Car**

**Mobile**

**Class : class is a blue print of object or template of object**

**Or user-defined data types or reference data type which help to create the object.**

**Syntax to create the memory**

**className referenceName = new ClassName();**

**refereceName.methodName();**

**refereceneName.variableName = value;**

**Car innova = new Car();**

**Types of variables or fields.**

**In Java variable are divided into 3 types**

1. **instance variable**
   1. **The variable which declare inside a class but outside method including main method is known as instance variable.**
   2. **Instance variable hold default value with respective their data types. int family 0, float family 0.0, char space, String null, boolean false.**
   3. **If variable is type of instance we can access those variable directly with a same class but method must be part of same class and it must be non static.**
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. **Scope of the variable within that method where it declared.**
3. **static variable**

**Day 4:**

**25-01-2022**

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

**Points**

1. **Constructor have the same name as the 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.**

**If we want to perform any task in the life of the object may be fixed task or dynamic task then we have to write the code inside a constructor.**

**But in the life of the object if we want to perform any task more than one time that type of code we have to write inside a method.**

**If we not write any constructor by default jvm provide default constructor it empty constructor.**

**But if we write explicitly empty or parameter then jvm doesn’t provide default constructor.**

**cid, cname,age : instance variable**

**default constructor**

**cid 123**

**cname unknown**

**age 21**

**parameter constructor**

**cid, cname, age, address, phonumber.**

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

**By default with the help of class concept we can achieve encapsulation.**

**Example : class**

**If instance variable is private we can’t access or change the value of variable directly as well as through object from outside class but within class we can access.**

**When instance variable and local variable or parameter variable have same name then local variable hide the visibility of instance variable.**

**this keyword : if we want to refer to instance variable when instance variable and local variable have same that type we can use this keyword to refer to instance variable.**

**Inheritance : Inheritance is use to inherits 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 or derived class or child class**

**properties**

**behaviour**

**}**

**With the help of sub class object we can access it own properties and behaviour as well as super class properties and behaviour.**

**With the help of super class object we can access only its property and 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 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{ }**

**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 {} Java doesn’t support this type of inheritance.**

**This type of inheritance in java we can achieve using interface.**

**Super class must be general and sub class must be specific.**

**class Employee {**

**id,name,salary; primitive property**

**}**

**class Manager extends Employee {**

**numberOfEmp;**

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

**}**

**class Developer extends Employee {**

**tech**

**}**

**class ProjectManager extends Manager{**

**numberOfProject**

**}**

**class Address {**

**city, state**

**}**

**OOPs relationship**

**Is a relationship**

**Has a relationship**

**Manager Is a Employee**

**Employee Has a Address**

**Day 5:**

**07-02-2022**

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

**2 types**

1. **Compile time polymorphism or static binding or early binding**

**Example**

**Method Overloading : The method have same name but different parameter list. Ie type of parameter list or number of parameter list must be different.**

1. **Run time polymorphism or dynamic binding or late binding**

**Example**

**Method Overriding : The method have same name as well as same method signature.**

**To achieve method override we have to use inheritance concept.**

**abstract keyword**

**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.**

**syntax**

**abstract returnType methodName(parameterList);**

1. **abstract class : if class contains abstract method we have to declare the that class as abstract class**

**abstract class className {**

**abstract method as well normal method.**

**}**

1. **Which ever class extends abstract class that class must be provide the body for all abstract method mandatory. That class can ignore only if that class itself is a abstract class.**
2. **We can’t create the object of abstract class.**
3. **Abstract class can contains normal as well as abstract method.**
4. **Abstract class can contains zero or 1 or many abstract methods.**

**If method is abstract we have to declare the class as abstract. But class can be abstract not mandatory method must be abstract.**

**Abstraction : abstraction is use to hide the internal implementation.**

**class Bike {**

**void speed(){**

**System.out.println(“80km/hr”);**

**}**

**}**

**class Honda extends Bike {**

**void speed(){**

**System.out.println(“100km/hr”);**

**}**

**}**

**Day 6:**

**08-02-2022**

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

1. **Static variable : if variable is static we can assign the value for that variable or we can call that variable without object. with the help of class name.**

**className.variableName=value;**

**className.variableName**

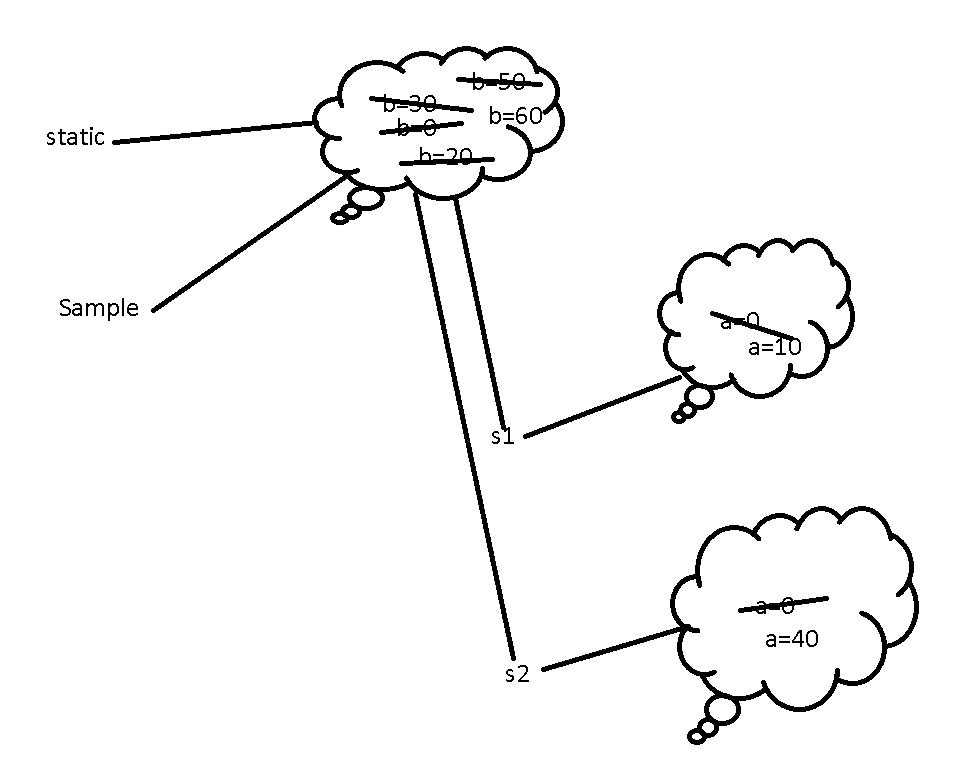
1. **If method is non static we can call that method with help of object. If method is static we can call that method with help of class name.**

**className.staticMethodName();**

1. **Static variable as well as static method we can call through object also.**
2. **Inside non static method we can access static as well as non static variable directly.**

**But inside static method we can access only static variable directly we can’t access non static variable directly.**

1. **Inside static method we can’t access non static variable directly. Because non static memory belong heap memory. If we want to call we have to create the object of that class and access through object.**

****

**Every class we get only one static memory.**

**Employee --🡪 id,name,salary**

**mgrId or clientId or projectId**

**Id,name,salary : instance variable**

**mgrId or clientId : static variable.**

**Static is global to all object if any object do any changes it will effect for all objects.**

**Has a relationship**

**Association**

**Aggregation**

**Composition**

**Day 7:**

**14-02-2022**

**Is a relationship : inheritance. Super class can be normal class or abstract class.**

**Has a relationship : Inside one class we have to create object of another class.**

**Association**

**class A {**

**B obj = new B(); // 0, 1 or many (multiplicity)**

**}**

**class B {**

**A obj1 = new A(); // 0, 1 or many (multiplicity)**

**}**

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

**class Manager {**

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

**}**

**class Address {**

**}**

**Composition it is type of association which is known as strong association.**

**class Student {**

**StudentHistory sh = new StudentHistory();**

**}**

**class StudentHistory{**

**}**

**class Product {**

**Orders od = new Orders();**

**}**

**class Orders {**

**}**

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

**final variable : if variable is final we can’t change the value. final variable is use to declare the constant variable. Final variable we can’t change the value.**

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

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

**final and abstract we can’t use together.**

**Interface : interface is type of reference data type. Which is also known as 100% pure abstract class.**

**Syntax to create the interface**

**interface interfaceName{**

**fields;**

**methods;**

**}**

**By default all variable present inside a interface are public , static and final.**

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

**interface A {**

**public static final int a=10;**

**static final int b=20;**

**final int c=30;**

**int d=40; // by default public, static and final**

**int e; // Error**

**public abstract void dis1();**

**abstract void dis2();**

**void dis3();**

**}**

**interface A {**

**int a=10;**

**void dis1();**

**}**

**interface B {**

**int b=20;**

**void dis2();**

**}**

**interface C extends A,B{**

**int c=30;**

**void dis3();**

**}**

**class D implements A,B {**

**void dis1() {}**

**void dsi2() {}**

**}**

**Like a class one interface can extends another interface. Interface can extends more than one interface. Class always implements interface. Class can implements more than one interface.**

**Whichever class implements any interface that class must be provide the body for all abstract methods belong to that interface.**

**Day 8:**

**15-02-2022**

**Access specifies while overriding the method**

**Super class / interface Sub class**

**Method**

**public public**

**protected public**

**protected**

**default (nothing) public**

**protected**

**default (nothing)**

**private we can’t override.**

**Difference between abstract class and interface**

1. **Attract class can contains abstract method. it may be zero or 1 or many. But interface contains all methods abstract by default.**
2. **Abstract class can contains normal as well as final variable (constant variable). But interface contains only final or constant variable.**
3. **Normal class can extends only one abstract class but it can implements more than one interface.**
4. **Using abstract class we can achieve partial abstraction. But using interface we can achieve fully abstraction.**

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

**We can’t create the object of interface as well as abstract class.**

**Run time polymorphism using object creation.**

**SuperClass(NormalClass/Interface) referenceName = new SubClass();**

**We can create object of sub class and super class reference. Super class can be normal class or can be abstract class or can be interface.**

**1st Example**

**package** abc;

**class** A {

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

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

}

}

**class** B **extends** A {

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

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

}

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

System.***out***.println("B class dis2() method");

}

}

**public** **class** Test {

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

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

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

obj2.dis1();

obj2.dis2();

A obj3 = **new** B();// creating sub class object and super class reference possible.

obj3.dis1(); // we can call only those method which belong to super class or overrided methods. If sub class overrrided that method we get that output else super class method output.

//obj3.dis2();

//B obj4 = new A(); // creating super class object and sub class reference not possible

}

}

**2nd Example**

**package** abc;

**abstract** **class** A {

**public** **abstract** **void** dis1();

}

**class** B **extends** A {

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

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

}

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

System.***out***.println("B class dis2() method");

}

}

**public** **class** Test {

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

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

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

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

}

}

**3rd Example**

**package** abc;

**interface** A{

**void** dis1();

}

interface C {

void dis3();

}

**class** B **implements** A,C {

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

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

}

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

System.***out***.println("B class dis2() method");

}

public void dis3() {

System.out.println(“dis3 method override in B class”);

}

}

**public** **class** Test {

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

//A obj1 = new A();

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

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

C obj4 = new B(); obj4.dis3();

// With the help of obj3 ie reference of interface. We can call only those methods.

// which belongs to interface but body provided by sub class.

}

}

**Day 9:**

**21-02-2022**

**package** com;

**interface** Abc { // abstract class Abc // you can write method with body

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

}

**interface** Xyz { // abstract class Xyz

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

}

**class** Server **implements** Abc,Xyz {

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

**return** x+y;

}

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

**return** x-y;

}

**public** **void** ownMethod() {

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

}

}

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

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

Server ss = **new** Server();

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

System.***out***.println(ss.sub(100, 50));

ss.ownMethod();

Abc obj1 = **new** Server(); // with help of obj1 we can call only Abc interface methods.

System.***out***.println(obj1.add(100, 200));

Xyz obj2 = **new** Server(); // with the help of object2 w can call only Xyz interfac methods.

System.***out***.println(obj2.sub(20,10));

}

}

**Access specifiers : Access specifiers is use to provide restriction or provide the visibility for class, methods or variables.**

**4 types**

**private**

**default (nothing)**

**protected**

**public**

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

**Scope :private scope within a same class.**

**default : we can use default access specifiers with all.**

**Scope : within a same package.**

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

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

**public : we can use public access specifiers with all except local variable.**

**Scope: same package as well as other package.**