**DAY – 15,16**

1. **CLASS**

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

It is a logical entity. It can't be physical.

A class in Java can contain:

-Fields

-Methods

-Constructors

-Blocks

-Nested class and interface

Syntax: class <class\_name>{

field;

method;

}

1. **OBJECT**

An entity that has state and behavior is known as an object e.g., chair. It can be physical or logical.

Object has 3 characteristics:

\*State: represents the data (value) of an object.

\*Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.

\*Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user.

However, it is used internally by the JVM to identify each object uniquely.

1. **INSTANCE VARIABLE**

Instance variables in Java are non-static variables which are defined in a class outside any method, constructor or a block.

Each instantiated object of the class has a separate copy or instance of that variable. An instance variable belongs to a class.

1. **Local Variable**

A variable declared inside the body of the method is called local variable.

You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

public class A

{

static int m=100;//static variable

void method()

{

int n=90;//local variable

}

public static void main(String args[])

{

int data=50;//instance variable

}

}//end of class

1. **CONSTRUCTOR**

A constructor in Java is a special method that is used to initialize objects.

The constructor is called when an object of a class is created.

It calls a default constructor if there is no constructor available in the class.

**Java Default Constructor**

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

<class\_name>(){}

1. **GETTER & SETTER**

They are used to protect data ,particularly when creating class.

A getter method returns its value while setter method updates or sets its value.

Getter starts with 'get' setter starts with 'set'.

class Bike1{

//creating a default constructor

Bike1(){System.out.println("Bike is created");}

//main method

public static void main(String args[]){

//calling a default constructor

Bike1 b=new Bike1();

}

}

1. **toString()**

This method returns string representation of Object.

By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

class Student{

int rollno;

String name;

String city;

Student(int rollno, String name, String city){

this.rollno=rollno;

this.name=name;

this.city=city;

}

public String toString(){//overriding the toString() method

return rollno+" "+name+" "+city;

}

public static void main(String args[]){

Student s1=new Student(101,"Raj","lucknow");

Student s2=new Student(102,"Vijay","ghaziabad");

System.out.println(s1);//compiler writes here s1.toString()

System.out.println(s2);//compiler writes here s2.toString()

}

}

1. **hashcode()**

It returns a hash code value for this object, equal to the primitive int value represented by this Integer object.

Syntax:

public int hashCode()

public static int hashCode(int value)

public class IntegerHashCodeExample1 {

public static void main(String[] args)

{

//Create integer object

Integer i = new Integer("155");

//Returned hash code value for this object

int hashValue = i.hashCode();

System.out.println("Hash code Value for object is: " + hashValue);

}

}

1. **this**

There is reference variable that refers to cureent object.

public class Test{

int x;

//constructor parameter

Test(int x){

this.x=x;

}

//call constructor

public static void main(String[] args)

{

Test obj=new Test(15);

System.out.println("value of x =" + obj.x);

}

}

1. **POJO**

POJO in Java stands for Plain Old Java Object.

It is an ordinary object, which is not bound by any special restriction.

The POJO file does not require any special classpath.

It increases the readability & re-usability of a Java program.

Generally, a POJO class contains variables and their Getters and Setters.

POJO class is used to define the object entities.

**//PROPERTIES**

* The POJO class must be public.
* It must have a public default constructor.
* It may have the arguments constructor.
* All objects must have some public Getters and Setters to access the object values by other Java Programs.
* The object in the POJO Class can have any access modifies such as private, public, protected. But, all instance variables should be private for improved security of the project.
* A POJO class should not extend predefined classes.
* It should not implement pre-specified interfaces.
* It should not have any pre-specified annotation.

1. **static vs final**

The static keyword means the value is the same for every instance of the class.

The final keyword means once the variable is assigned a value it can never be changed.

The combination of static final in Java is how to create a constant value.

~Static keyword denotes that a member variable, or method, can be accessed without requiring an instantiation of the class to which it belongs.

~The final keyword denotes an entity that can only be assigned once.

(Variables)

~The static variables can be reinitialized.

~The final variables cannot be reinitialized.

(Methods)

~Can be called by other static methods and only access the static members of the class.

~The final methods cannot be overridden.

(Class)

~The static class object cannot be created. It only contains static members only.

~The final class cannot be inherited by other classes.

(Block)

~The static keyword can be used in a block.

~The final keyword is not used with a block.

1. **Inheritance**

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object .

class Subclass-name extends Superclass-name

{

//methods and fields

}

The extends keyword indicates that you are making a new class that derives from an existing class.

The meaning of "extends" is to increase the functionality.

1. **Abstraction**

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

It is a non-access modifier keyword , used for classes and methods.

FOR EX: sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

* **Abstract class in Java**

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented.

It cannot be instantiated.

Syntax: abstract class A{}

* **Abstract Method in Java**

A method which is declared as abstract and does not have implementation is known as an abstract method.

Syntax: abstract void printStatus();//no method body and abstract

abstract class Bike{

abstract void run();

}

class Honda4 extends Bike{

void run(){System.out.println("running safely");}

public static void main(String args[]){

Bike obj = new Honda4();

obj.run();

}

}

1. **Overloading**

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

class Adder{

static int add(int a,int b){return a+b;}

static int add(int a,int b,int c){return a+b+c;}

}

class TestOverloading1{

public static void main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(11,11,11));

}}

1. **Overriding**

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class,

it is known as method overriding.

class Vehicle{

void run(){System.out.println("Vehicle is running");}

}

//Creating a child class

class Bike extends Vehicle{

public static void main(String args[]){

//creating an instance of child class

Bike obj = new Bike();

//calling the method with child class instance

obj.run();

}

}