OOPs NOTES

What is OOPs?

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behaviour.

What is a class?

A class is a collection of objects. Classes don't consume any space in the memory.

It is a user defined data type that act as a template for creating objects of the identical type.

A large number of objects can be created using the same class. Therefore, Class is considered as the blueprint for the object.

What is an object?

An object is a real-world entity which have properties and functionalities. Object is also called an instance of class. Objects take some space in memory.

For eg .

- Fruit is class and its object s are mango, apple, banana
- Furniture is class and its objects are table, chair, desk

What is the difference between a class and an object?

Class:

1. It is a collection of objects.
2. It doesn't take up space in memory.
3. Classes are declared just once

Object:

1. It is an instance of a class.
2. It takes space in memory.
3. Objects can be declared as and when required

What is the difference between a class and a structure?

Class is a collection of objects.

Structure is a collection of variables of different data types under a single unit

Class is used to combine data and methods together.

Structure is used to grouping data.

Class's objects are created on the heap memory.

Structure's objects are created on the stack memory.

A class can inherit another class.

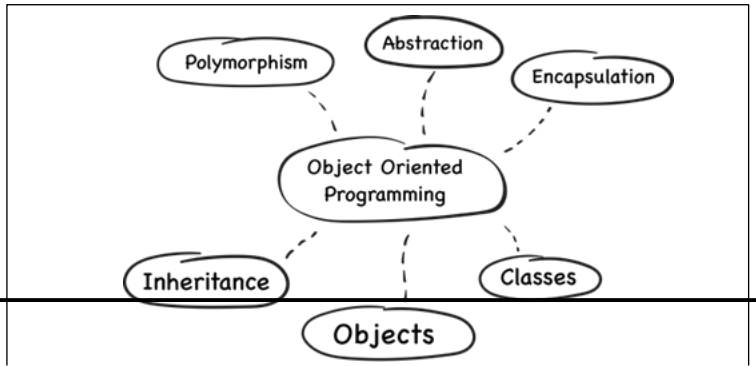
A structure can't inherit another structure.

A class has all members private by default

A structure has all members public by default

Classes are ideal for larger or complex objects

Structures are ideal for small and isolated model objects



There are typically Three Principles in object-oriented programming (OOP):

Encapsulation: This principle focuses on the bundling of related data and functions into an object, allowing access only through the object's methods. It helps in hiding the internal implementation details and protects the data from unauthorized access.

Inheritance: This principle provides the ability to create new classes based on the existing classes. It allows the derived class (subclass) to inherit the properties and behaviours of the base class (superclass). Inheritance promotes code reusability and supports the concept of "is-a" relationship.

Polymorphism: Polymorphism means the ability of an object to take on many forms. In OOP, it refers to the ability to use a single interface or method to represent different types of objects. It allows objects of different classes to be treated as objects of a common superclass through method overriding and method overloading.

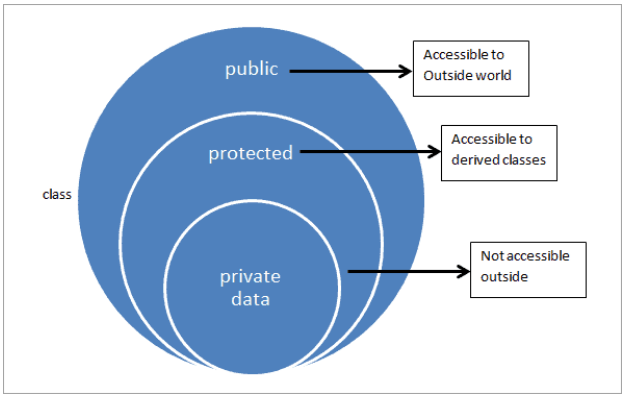
Abstraction is not considered as a separate principle in OOP, but rather a concept that is utilized in all the pillars mentioned above.

Abstraction: Abstraction is the process of hiding the internal details and complexities of an object and providing a public interface to interact with the object. It helps in achieving data security, modularity, and code maintainability.

Data Hiding: Process of hiding data from outside world.

Encapsulation:

The main advantage of encapsulation is that data is hidden and protected from randomly access by outside non-member methods of a class. Encapsulation is the process of binding data and methods in a single unit. In encapsulation, data(variables) are declared as private and methods are declared as public



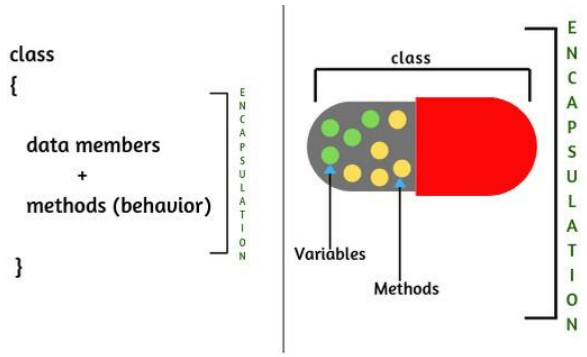


Fig: Encapsulation

```
public class CharToString {
    public static void main(String[] args) {
        char arr[] ={'a','s','h','w','a','n','i'};

        String s=new String(arr);
        System.out.println(s);

    }
}
```

Inheritance:

Inheritance is the procedure in which one class inherits the attributes and

methods of another class.

In other words, It is a mechanism of acquiring properties or behaviours of existing class to a new class

The Base Class, also known as the Parent Class is a class, from which other classes are derived.

The Derived Class, also known as Child Class, is a class that is created from an existing class

```
public class Animal {

    public String ranveer()
    {
        //System.out.println("papa papa papa papa");
        return "papa papa papa papa";
    }

    public String ranveer(String arr)
    {

        return arr;
    }

    public int ranveer(int a)
    {

        return a;
    }

    public int ranveer(int a, int b)
    {
        return a+b;
    }

    public String rashmika()
    {
       // System.out.println("---------------------");
        return "---------------";
    }
    public String rashmika(String arr)
    {
        return arr;
    }
}
```

```java
class Dad {

    public  String plot()
    {
        return "200 gaj ka plot";
    }

}

 class Khushi extends Dad{
    public void demo()
    {
        plot();
    }

     public  String plot()
     {
         return "khushi coffee point";
     }


}

public class Driver
{
    public static void main(String[] args) {
        Khushi khushi=new Khushi();
        System.out.println(khushi.plot());

    }
}
```

There are four types of inheritance in OOP:

- Single Level Inheritance
- Hierarchical Inheritance
- Multi-Level Inheritance
- Multiple Inheritance
- Hybrid inheritance

Abstraction:

Allows to hide unnecessary data from the user. This reduces program complexity efforts.

it displays only the necessary information to the user and hides all the internal background details.

If we talk about data abstraction in programming language, the code implementation is hidden from the user and only the necessary functionality is shown or provided to the user.

In other words, it deals with the outside view of an object (Interface)

Eg.

All are performing operations on the ATM machine like cash withdrawal etc.

but we can't know internal details about ATM

phone call we don’t know the internal processing

We can achieve data abstraction by using

1. Abstract class
2. Interface

Real Life Example of Abstraction

Polymorphism

Polymorphism is a programming concept that allows for the same code to be used with different data types. This is achieved by using inheritance and method overriding.

Type of polymorphism

1. Overloading
2. Overriding

Overloading

Method overloading means declaring multiple methods with same method name but having different method signature.

In method overloading while writing the method signature we have to follow following 3 Rules

• Method name must be same for all methods

• List of parameters must be different like different type of parameters, different

number of parameters, different order of parameters.

• Return type is not considered in method overloading; it means we never decide method overloading with return type

```
public class OverloadingDemo {

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

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

Overriding

If we want to achieve the run time polymorphism then we have to use method overriding.

Method overriding means declaring 2 methods with same method signature in 2 different classes which are having IS-A relation.

While Method overriding and writing the method signature, we must follow following rules.

- Method name must be same
- List of parameters must be same
- Return type must be same
- Private, final and static methods cannot be overridden.
- There must be an IS-A relationship between classes (inheritance)

```
class Shape {
    void draw()
    {
        System.out.println("Drawing Shape");
    }
}

class Circle extends Shape{
    @Override
    void draw() {
        System.out.println("Drawing circle");
    }
}

class Rectangle extends Shape{
    @Override
    void draw() {
        System.out.println("Drawing Rectangle");
    }
}


public class Test2{

    public static void main(String[] args) {
        Shape shape=new Shape();
        shape.draw();
        Circle circle=new Circle();
        circle.draw();;
        Rectangle rectangle=new Rectangle();
        rectangle.draw();

        Shape shape1=new Circle();
        Shape shape2=new Rectangle();
        shape1.draw();
```

```
        shape2.draw();

        //Circle circle1=new Shape();
    }
}
```

super keyword

In Java, super keyword is used to refer to immediate parent class of a child class. In other words, super keyword is used by a subclass whenever it need to refer to its immediate super class.

Child.java

```
1  class Parent {
2      String name;
3
4      void details() {
5          System.out.println(name.toUpperCase());
6      }
7  }
8
9  public class Child extends Parent {
10     String name;
11
12     public void details() {
13         super.name = "Parent"; // refers to parent class member
14         name = "Child";
15         System.out.println(super.name + " and " + name);
16         super.details(); // refers to parent class method
17     }
18
19     public static void main(String[] args) {
20         Child cobj = new Child();
21         cobj.details();
22     }
23 }
```

Note: When calling the parent class constructor from the child class using super keyword, super keyword should always be the first line in the method/constructor of the child class.

Q. Can we use both this () and super () in a Constructor?

NO, because both super () and this () must be first statement inside a constructor. Hence, we cannot use them together.

Final keyword

final is a keyword or modifier which can be used at variables, methods & classes.

If we declare a variable as final then we can't modify value of the variable. The variable acts like a constant. Final field must be initialized when it is declared.

If we declare a method as final then we can't override that method

If we declare a class as final then we can't extend from that class. We cannot inherit final class in Java.

FinalDemo.java

```java
public final class FinalDemo {

    final int a = 10; // valid

    a=20; // invalid

    final int c; // invalid

    final void m1() {
        System.out.println("Hello");
    }

    public static void main(String[] args) {
        FinalDemo d = new FinalDemo();
        d.m1(); // valid
    }
}

class Test extends FinalDemo { // invalid
}
```

Type casting

Type casting is the process of converting value from one type to another type.

We always do typecast between 2 different data types which are compatible.

Note: In between 2 same data types typecasting is not required.

In java we have following 2 types of type castings

1.type casting w.r.t primitive data types

2.type casting w.r.t reference types

Type casting wrt primitive data types

Type casting wrt primitive data types means converting value from one primitive data type to other primitive data type

Type casting can be done only between compatible data types

Compatible data types are byte, short, int, long, float, double, char

we have following 2 types of Type casting w.r.t primitive data types

1) Widening

2) Narrowing

Widening

Widening means converting the value from lower data type into higher data type.

syntax:

higher datatype = (higher datatype) lower datatype ;

- Here data type specified in between the pair of parentheses is called type casting.
- In widening writing the type casting is optional
- If we don't write any type casting then compiler will write the type casting automatically hence it is called as implicit type casting.

Main.java

```java
class Main {
    public static void main(String[] args) {
        // create int type variable
        int num = 10;
        System.out.println("The integer value: " + num);

        // converting int type to double type
        double data = num;
        System.out.println("The double value: " + data);
    }
}
```

Narrowing

narrowing means converting the higher data type value into smaller data type.

syntax:

lowerdatatype = (lowerdatatype) higherdatatype;

-> In narrowing writing the type casting is mandatory

-> In narrowing if we don't write any type casting then compiler won't write any typecasting

because there is a chance of loss of some data so that user has to write the typecasting explicitly hence it is called as explicit type casting

Main.java

```java
class Main {
    public static void main(String[] args) {
        // create double type variable
        double num = 10.99;
        System.out.println("The double value: " + num);

        // convert double type to int type
        int data = (int) num;
        System.out.println("The integer value: " + data);
    }
}
```

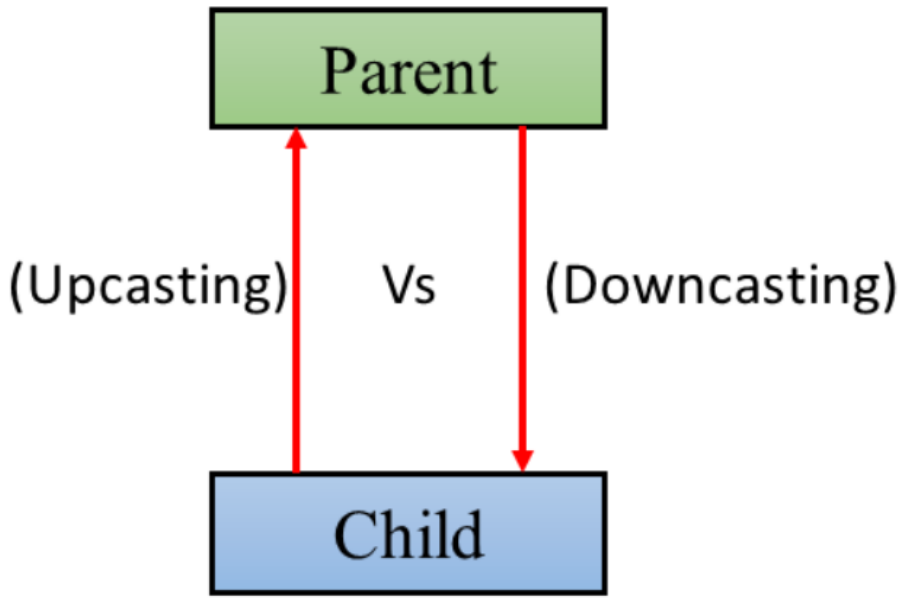
Type casting w.r.t reference types

-> Type casting w.r.t reference types means converting the object from one reference type to another reference type

-> But Type casting w.r.t references can be done only between compatible types

-> The two references said to be compatible if and only if its corresponding classes having Is-A relation

-> Type casting w.r.t references is also classified into following 2 types



Up casting

-> Up casting means storing the child class object into the parent class reference.

syntax:

parentreferencetype = (parentreferencetype) childreferencetype

Note: In up casting writing typecasting is optional Down casting

-> Down casting means storing the Parent class object into the child class reference.

syntax: childreferencetype = (childreferencetype) parentreferencetype

Note: In down casting writing typecasting is mandatory

TypeCasting.java

```java
class Parent {
    void show() {
        System.out.println("This is Parent class Method");
    }
}
class Child extends Parent {
    void show() {
        System.out.println("This is Child class Method");
    }
}

class TypeCasting {
    public static void main(String args[]) {
        Parent p = new Parent();
        p.show();

        p = (Parent) new Child(); // typecasting optional (upcasting)
        p.show();

        Child c = (Child) p; // typecasting is mandatory ( downcasting)
        c.show();
    }
}
```

Java Abstract class and methods

-> A class which is declared using abstract keyword known as abstract class.

-> An abstract class may or may not have abstract methods.

-> We cannot create object of abstract class.

-> It is used to achieve abstraction but it does not provide 100% abstraction because it can have concrete methods.

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It is used for abstraction.

Syntax :

abstract class class_name { }

Abstract method

Method that are declared without any body within an abstract class are called abstract method.

The method body will be defined by its subclass.

Abstract method can never be final and static.

Any class that extends an abstract class must implement all the abstract methods.

Syntax:

abstract return_type function_name (); //No definition

Car.java

```java
abstract class Vehicle {
    public abstract void engine();
}

public class Car extends Vehicle {

    public void engine() {
        System.out.println("Car engine");
    }

    public static void main(String[] args) {
        Vehicle v = new Car();
        v.engine();
    }
}

```

this in Java

he `this` keyword refers to the current object in a method or constructor.

```
class Animal {
    String name;
}

class Dog extends Animal{
    String name;
    void print(){
        super.name="Animal";
        name="jypsy";
        System.out.println("Parent name = "+super.name+" Child name =
"+this.name);
    }

}
```

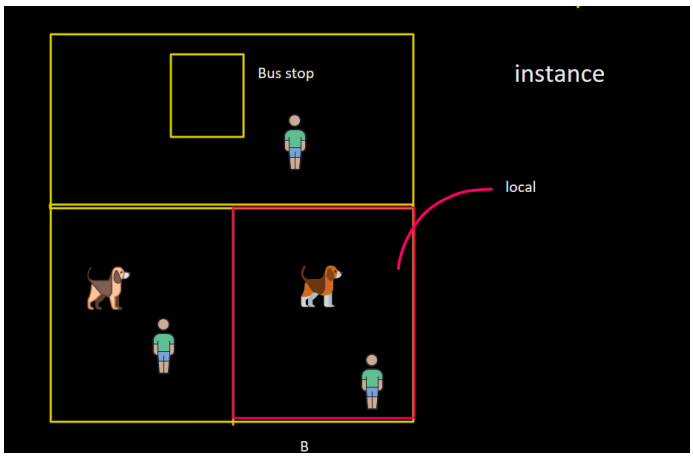
Types of variables:

1. Instance: ese variable jo class k ander or method k bahar ho
2. local: ese variable jo method k ander ho
3. static: ese variable jo static keyword se bane ho

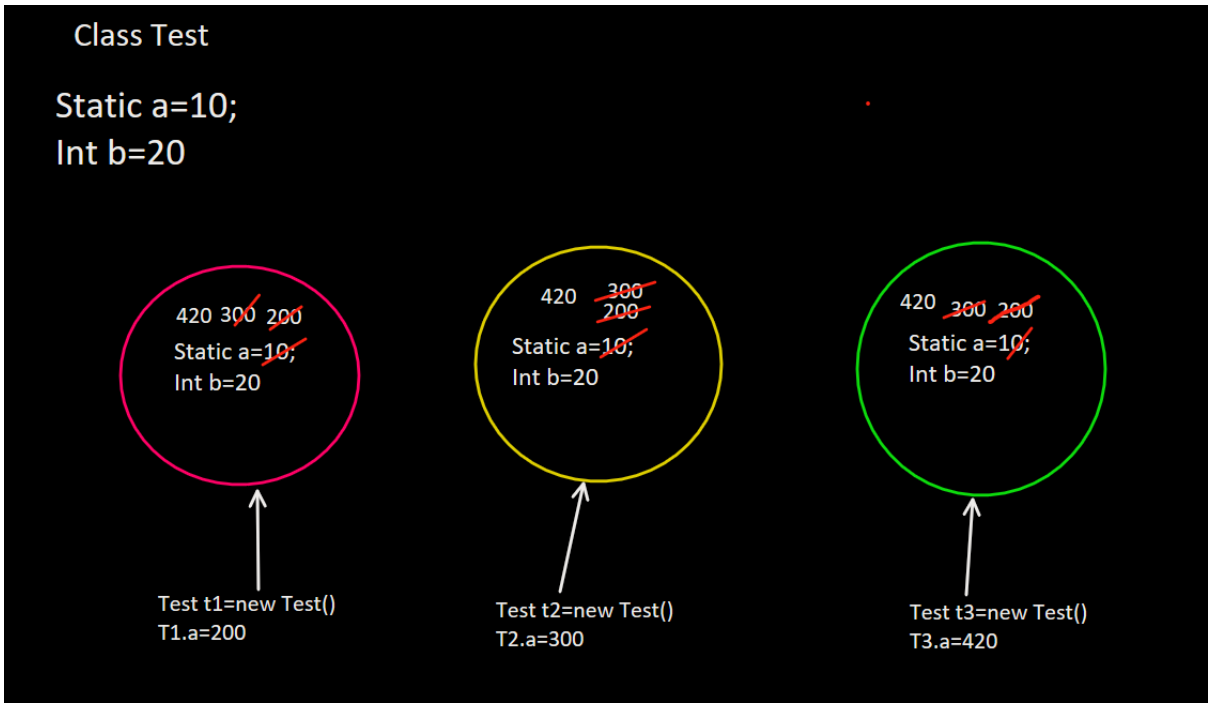
Instance Variable:

1. declared inside a class but outside any method, constructor, or block.
2. Available to all methods, constructors, and blocks in the class.
3. Each object of the class has its own copy of the instance variables.
4. Created when an object is created using the new keyword and destroyed when the object is destroyed.

```
1 usage   1 inheritor   new *
class Parent
{
    1 usage
    String name ="Dipti"; //instance variable
    no usages   1 override   new *
    public void display()
    {
        String name ="navya";  // local
        System.out.println(name);
    }
}
```



Check code for below example:



Local Variable

1. Variables declared inside a method, constructor, or block.
2. Only accessible within the method, constructor, or block in which they are declared.
3. Stored on the stack and only exists during the execution of the method, constructor, or block.
4. No default values. Must be initialized before use.
5. **Access Modifiers**: Cannot use access modifiers (private, protected, public) for local variables.

```java
public class Example {
    public void method() {
        int localVar = 10; // Local variable
        // Can access localVar here
    }
}
```

Static Variable

1. Variables declared with the static keyword inside a class, but outside any method, constructor, or block.
2. Belongs to the class rather than any object. All instances of the class share the same static variable.
3. Stored in the static memory. Only one copy exists regardless of the number of instances.

```java
class Test{
    7 usages
    static int a=10; //static variable
    6 usages
    int b=20;
    no usages  new *
    public void display()
    {
        System.out.println(a);
    }
}
```

- Class Test

Static a=10;
Int b=20