Object Oriented Programming

"खुल जाएंगे सभी रास्ते, तू रुकावटों से लड़ तो सही,
सब होगा हासिल, तू अपनी ज़िद पर अड़ तो सही।"

- What is OOPs
- Classes & Objects
- Access Specifiers
- Constructor & this pointer
- Encapsulation
- Polymorphism
- Inheritance & its types
- Abstraction & Inheritance
- Static keyword
- Packages
- Final Keyword
- Practice Problems on OOPS

C → procedural

10th → Experiment

(i) Aim

(ii) Apparatus

(iii) Procedure

add()
{
}

sub()
{
}

main()
{
int x
}

add()
{
}

main()
{
}

Car

Car

Accelerator
Break
Brand

```
void add ()
{
    c = 7+8;
    printf (c);
}

void fun ()
{
    add();
}
```

✓ Compile
✓ Run

Error

- ✓ Compilation
- Error

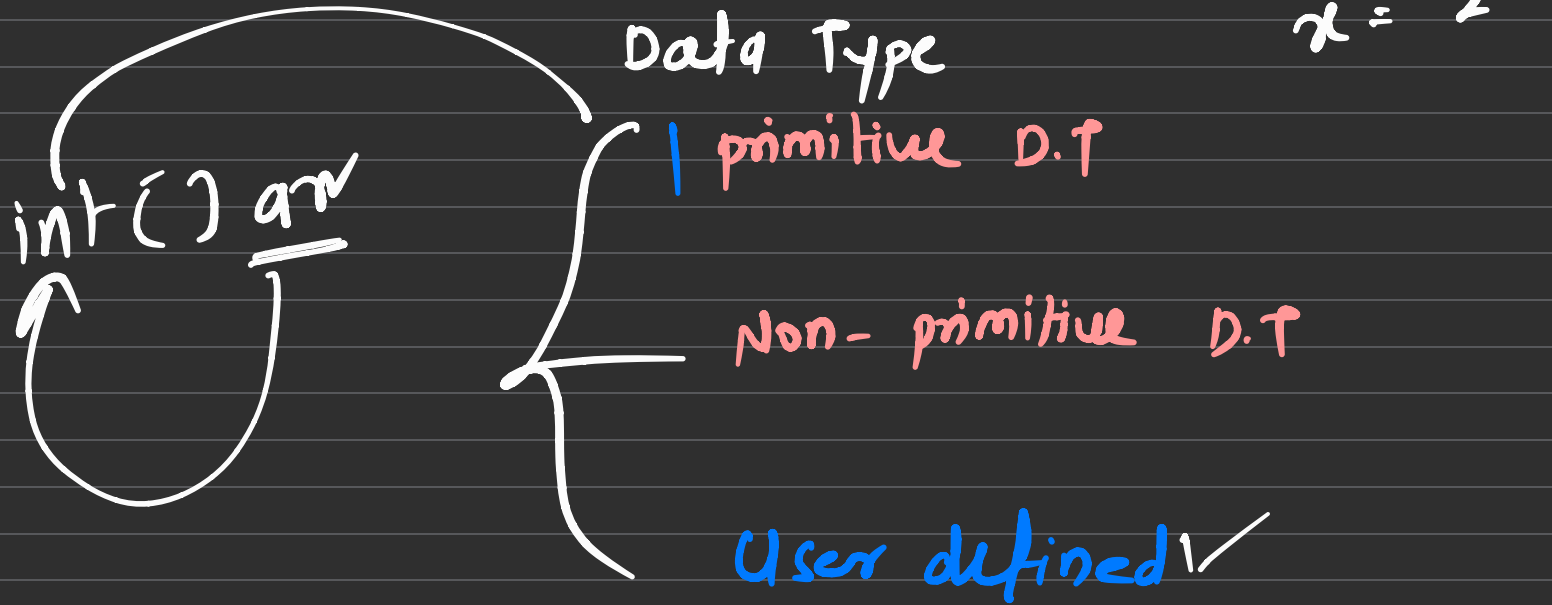
Linker

What is OOPs

OOPs is a programming technique which mainly revolve around real life objects

int x = 10

x = 'z'



Classes & Objects

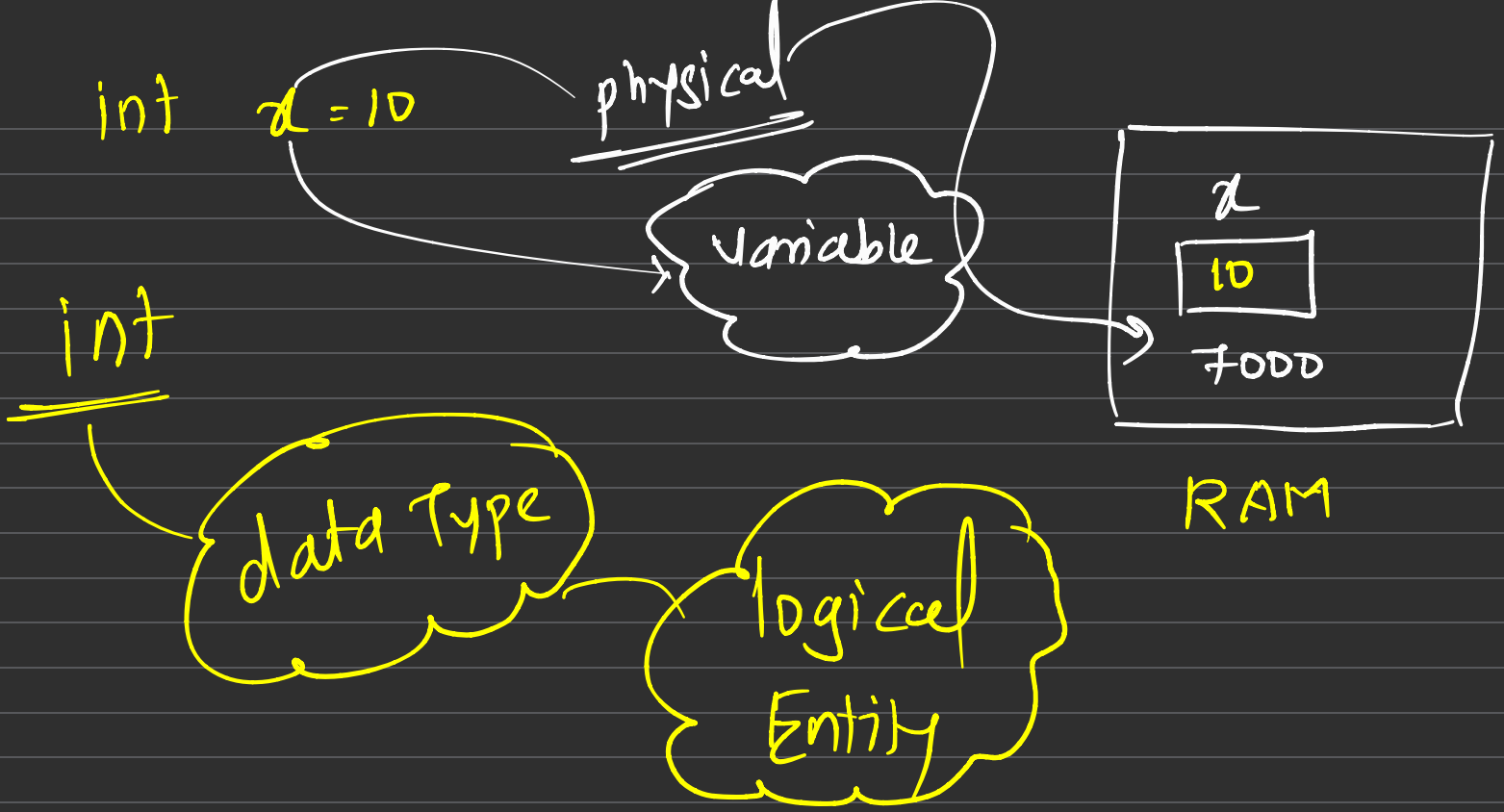
Class

It is user defined data type, which contains data members (variables) & methods (functions).

syntax:

```
class ClassName
{
    // data member

    // method

}
```

```
class Person
{
    int age
    string nm

    void show()
    {
        println(age)
        println(nm)
    }
}
```

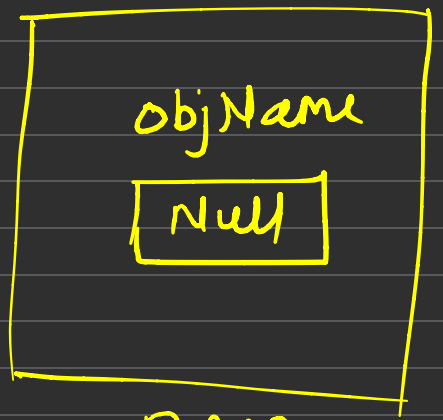


Object

It is a instance of the class – which tells what kind of operation we can perform on the class.

Syntax:

```
className objName;
objName = new className()
```




```
className obj = new className()
```

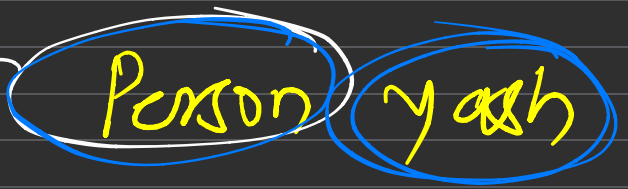
↳ dynamically

```java
class Person{  no usages
    // data members
    int age;  1 usage
    String name;  1 usage

    // member function
    public void info(){  no usages
        System.out.println("Age of the person is: " + age);
        System.out.println("Name of the person is: " + name);
    }
}
```

data Type

Person yash = new Person();

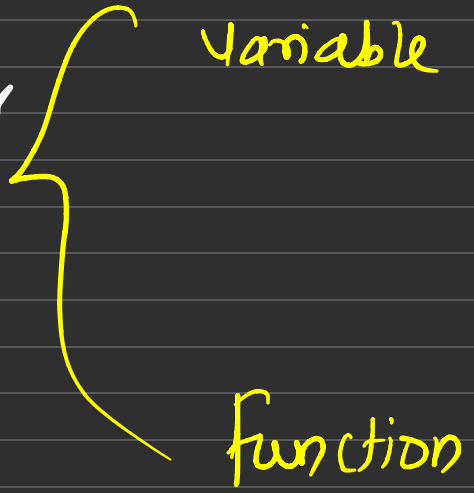
data Type



How to access members of the class

Using dot (.) operator

obj . member



Access Specifiers

Mainly decides the scope of accessibility of members of the class

- ✓ default
- public
- private
- Protected
  - inheritance (X)

* default : Can be access inside of class & outside of class but in same package

* private : Only inside the class

* public : can be accessed anywhere inside / outside package or class

Constructor & this pointer

Constructor

Mainly used to initialise the object of the class.

- name of constructor = name of class
- alway public
- called automatically
- No return type → not even void
- Diff. betn function vs Constructor

Types of Constructor

- Default Constructor
- Parameterized Constructor

this pointer

Mainly used to instialise the current object of the class.

Encapsulation

Binding of data member and member function together in the same class is called as Encapsulation.

```
class Person
{
    int age
    String name

    public void info()
    {
        println (age)
        println (name)
    }
```

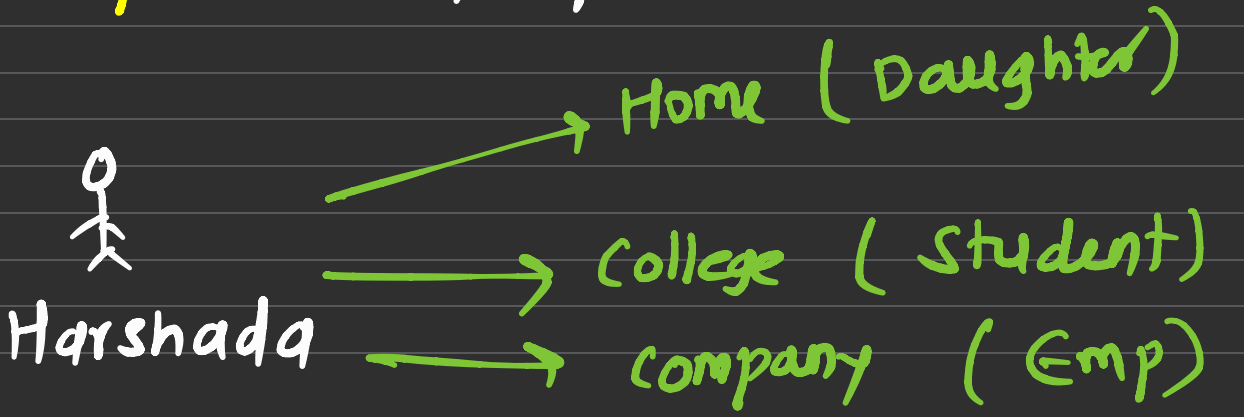
Encapsulation: Real life examples

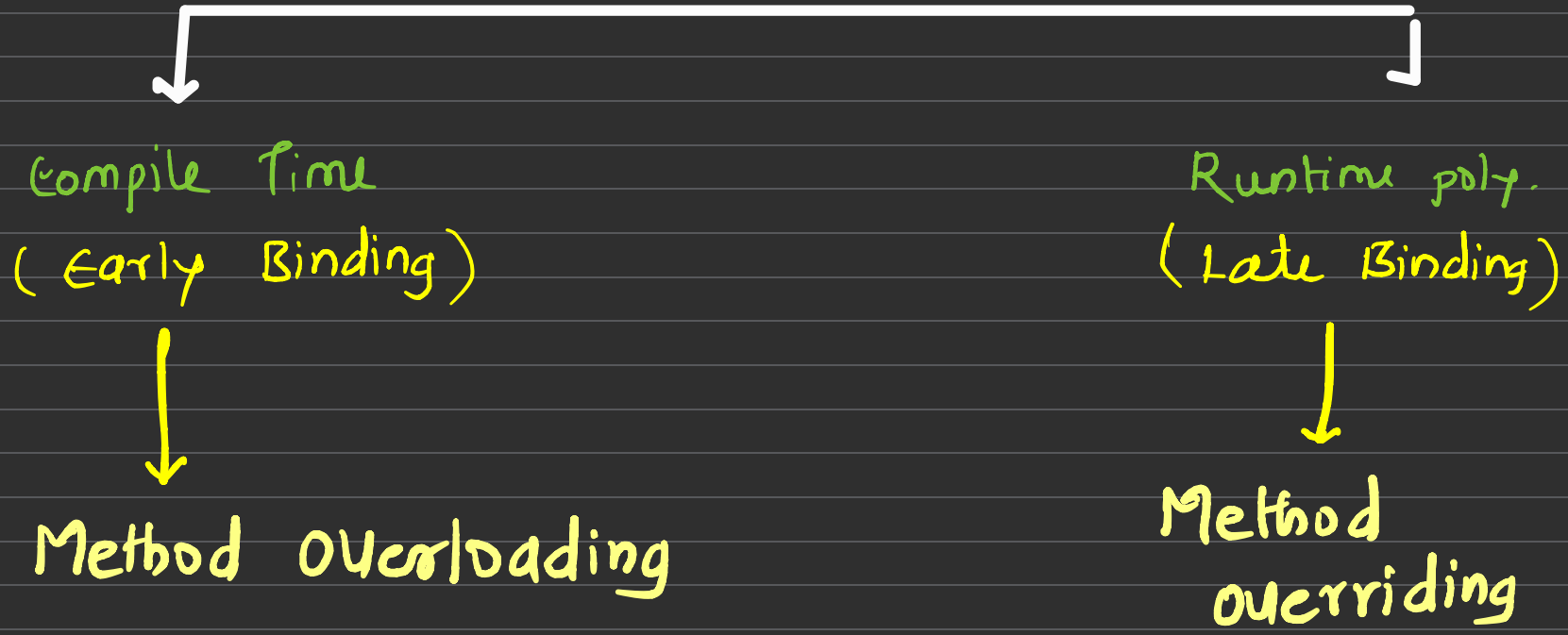
An organization consists of several departments like the production department, purchase department, sales department, and Accounts department. It combines all these departments together and formed the organization.

Polymorphism

When single entity acting differently in different scenarios is called as Polymorphism.

Greek { poly → many
morphism → forms





* Method Overloading

Two or more methods having same name but different parameters

- All method must be in same class
- Return type does not matter

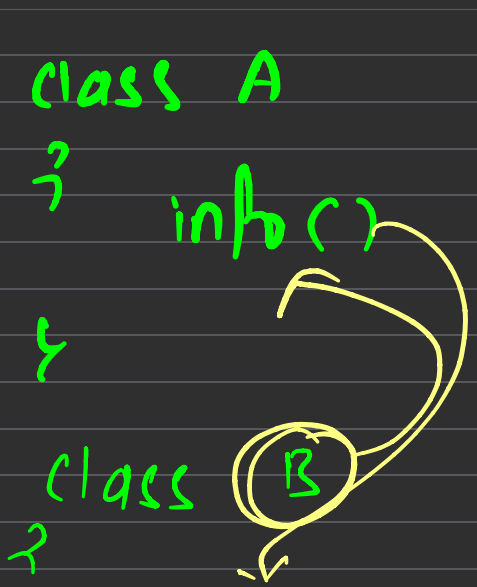
* Method Overriding

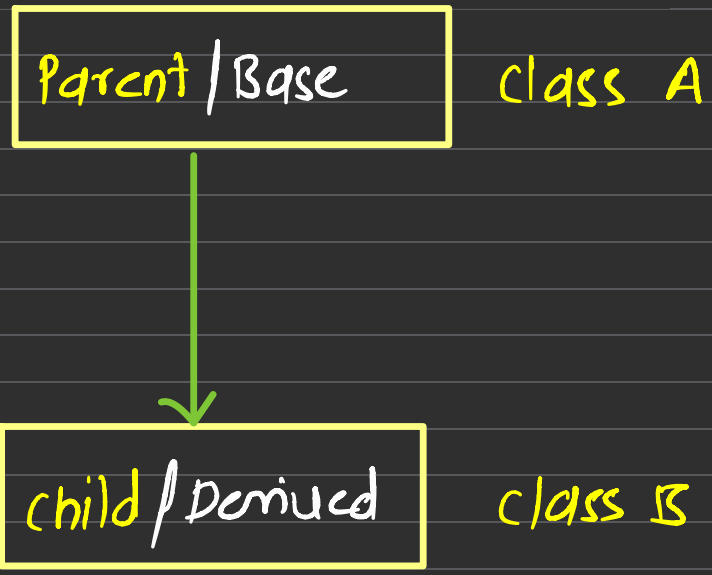
- Redefining the meaning of method from parent class to child class

- function should have
  - Same name
  - Same parameter
  - Same return type
- Should be in diff class

Inheritance

The capability of a class to derive properties and characteristics from another class is called Inheritance.





Syntax Inheritance

CPP

```cpp
// Base Class
class Person{


};

// Derived Class
class Pramod: public Person{



};
```

JAVA

```java
// Base Class
class Person{  3 usages  1 inheritor

}

// Derived Class
class Pramod extends Person{

}
```

Mode of Inheritance

Public mode: If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.

Protected mode: If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.

Private mode: If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class.

Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

```
class Person
{
    private int age
    private String name

}

class Pramod extends Person
{
    public int age
    public String name
}
```

protected > public

private > public

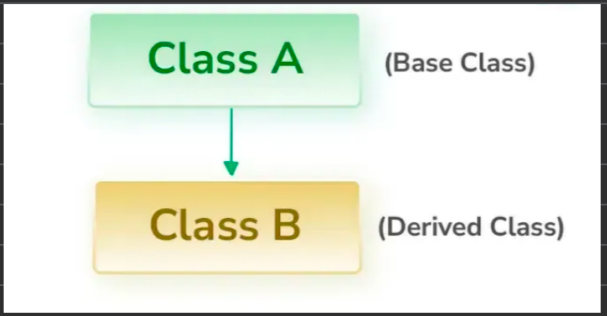
private > protected

Types of Inheritance

- Single Inheritance
- Multilevel Inheritance
- Multiple Inheritance
- Hierarchical Inheritance

Single Inheritance

In single inheritance, a class is allowed to inherit from only one class. i.e. one base class is inherited by one derived class only.



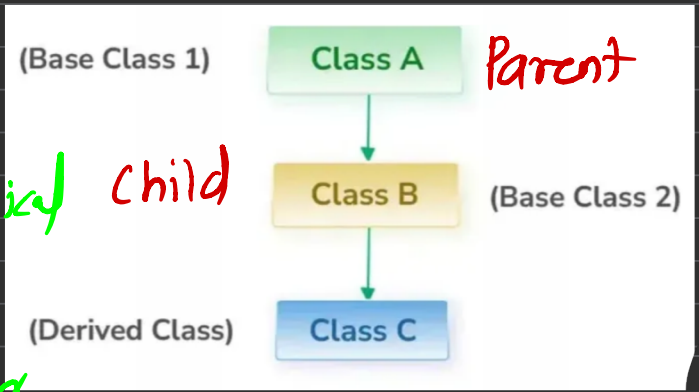
Multilevel Inheritance

A derived class is created from another derived class and that derived class can be derived from a base class or any other derived class. There can be any number of levels.

Class Vechical

Class FourWheeler extend Vechical

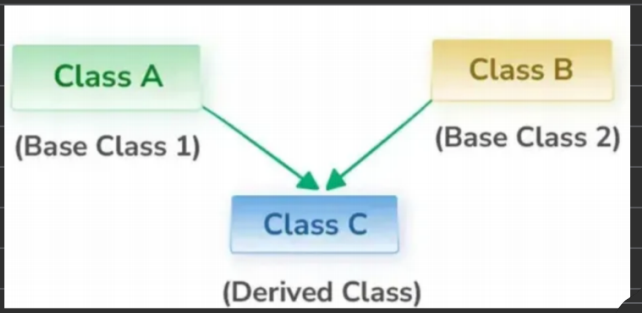
Class Car extend FourWheeler



child

Multiple Inheritance

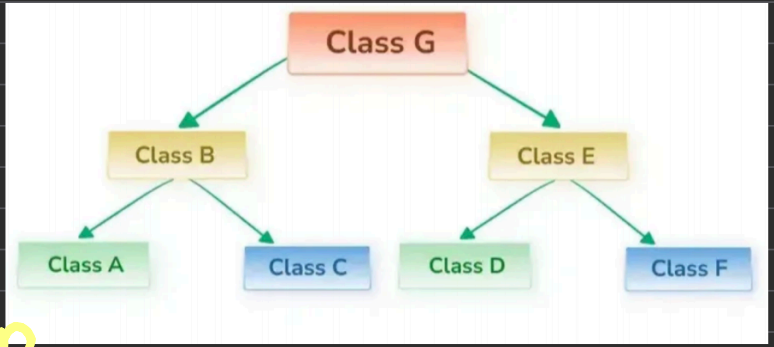
Multiple Inheritance where a class can inherit from more than one class. i.e one Derived is inherited from more than one Base Class.




```
class A      //P
{
    info()
}

class B //P
{
    info()
}

class C -> A, B   //Child
{
    info()
    info()
}

C c = new C()
c.info()
```

Hierarchical Inheritance

more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class.



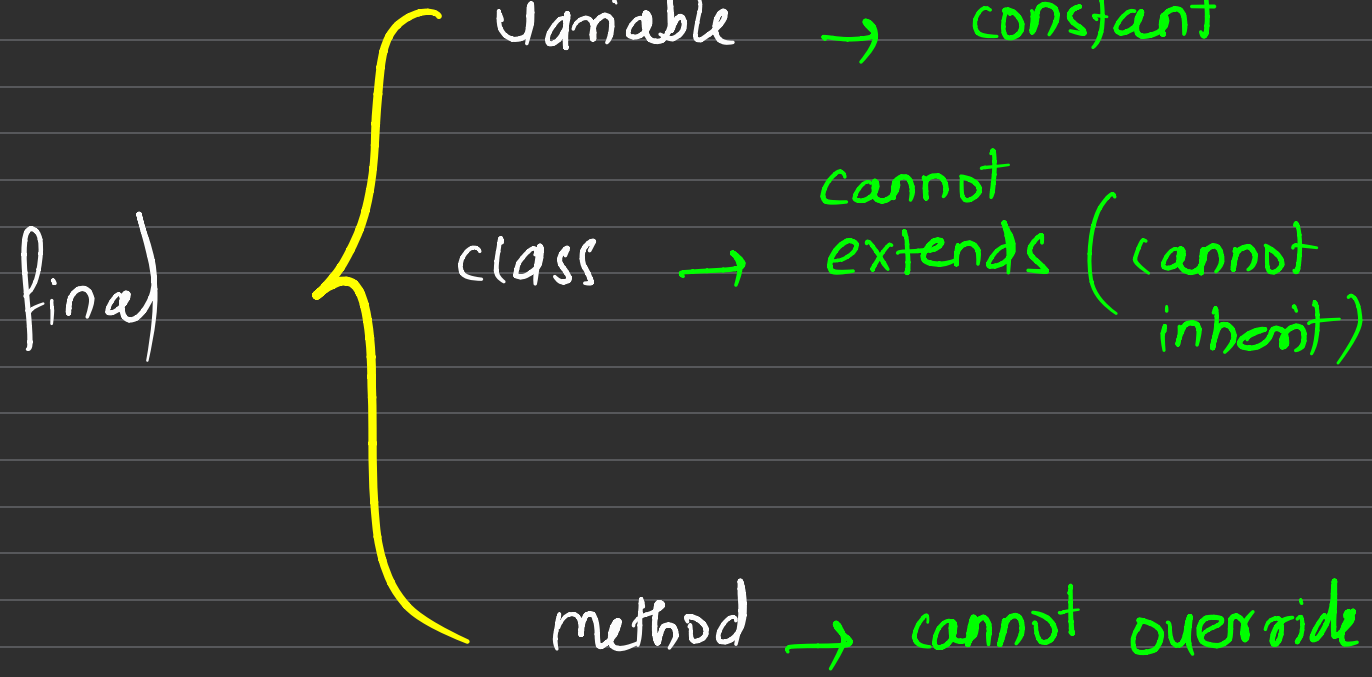
class Person (P)

class Pranid → Person

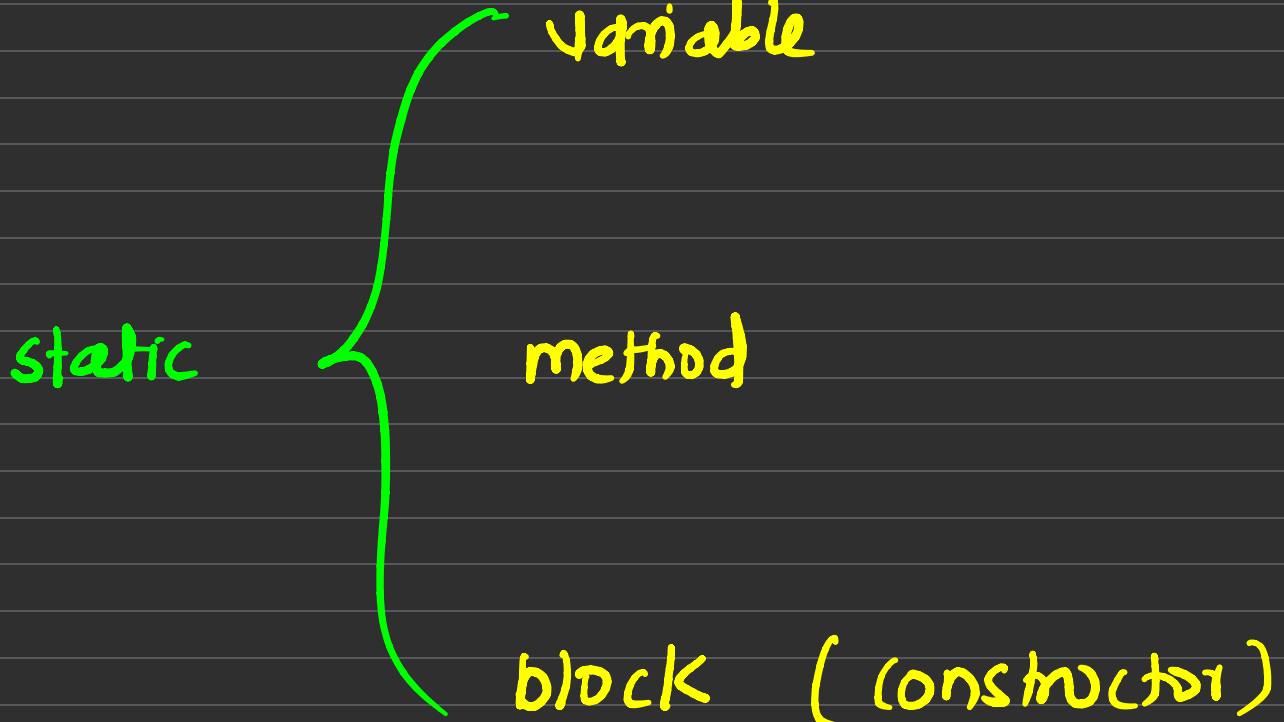
class Prachi → Person

class Yash → Person

Final → Keyword



Static (keyword)



- static members are members of class not of an object.

- We can directly call it without object

ClassName.member (static)

- Static member (Variable) cannot be present in non-static block / method

Abstraction

Abstraction in Java is the process of hiding internal implementation details and showing only essential functionality to the user. It focuses on what an object does rather than how it does it.

How to Achieve Abstraction in Java?

Java provides two ways to implement abstraction, which are listed below:

- Abstract Classes (Partial Abstraction)
- Interface (100% Abstraction)

Another real life example of Abstraction is ATM Machine; All are performing operations on the ATM machine like cash withdrawal, money transfer, retrieve mini-statement…etc. but we can't know internal details about ATM.

Real Life Example of Abstraction

Abstract class

An abstract class is a class that cannot be instantiated

Syntax :

```
abstract class ClassName
{

}
```

```
class Person
{
    int x
    final int num
    static int n

    public void info()
    {
        // statement
    }

    public void show();
```

```
abstract class Person
{
    variable { final, Non-final, static, Non static }

    public void info()
    {
        // statement
    }

    abstract void show();
```

1. An instance of an abstract class can not be created.
2. Constructors are allowed.
3. We can have an abstract class without any abstract method.
4. There can be a **final method** in abstract class but any abstract method in class(abstract class) can not be declared as final or in simpler terms final method can not be abstract itself as it will yield an error: "Illegal combination of modifiers: abstract and final"
5. We can define static methods in an abstract class
6. We can use the **abstract keyword** for declaring ***top-level classes (Outer class) as well as inner classes*** as abstract
7. If a **class** contains at least **one abstract method** then compulsory should declare a class as abstract
8. If the **Child class** is unable to provide implementation to all abstract methods of the **Parent class** then we should declare that **Child class as abstract** so that the next level Child class should provide implementation to the remaining abstract method

interface

Syntax:

```
interface   Name
{

}
```

- All the members inside the interface is public
- All the methods are by default abstract
- It can have only final or static variable

```
interface Person
{
    final string name

    void show();

}
```

```
Class Pramod extend Person
{

}
```

implements

```
class Test {
    int x;
    Test(int val) {
        x = val;
    }
    public static void main(String[] args) {
        Test t1 = new Test(10);
        Test t2 = t1;
        t2.x = 20;
        System.out.println(t1.x);
    }
}
```