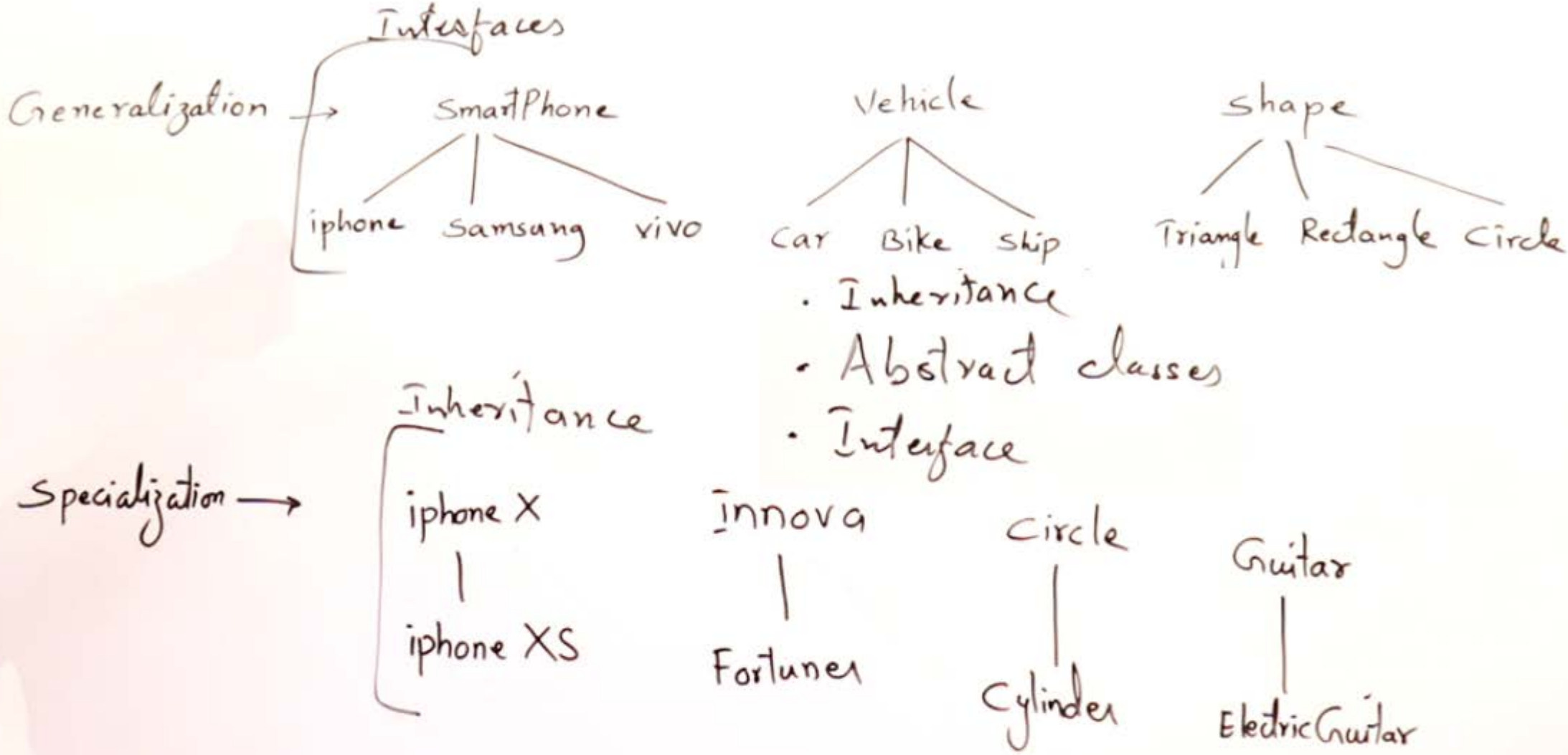
Inheritance



- Inheritance
- Abstract classes
- Interface

--> * Generalization mean, " <u>group of class</u> " ko kisi common name sai call karna.

* ya ek Bottom up (mtlb nicha sai upper ki traf) approach hota hai.
* ham interface approach Sai, generalization ko achieve kar sakta hai.

--> * Specalization mean, kisi existence class ka upgrade version jis mai kuch purana and kuch new features add ho.

* ya ek Top - down (upper sai nicha ki traf) approch hai.
* Specalization achieve using " <u>inheritance</u> ".
* new Class is derived from an existing Super Class.

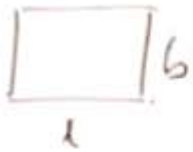
Inheritance

circle

↑

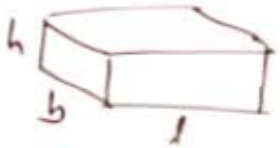
cylinder

Rectangle



↑

Cuboid



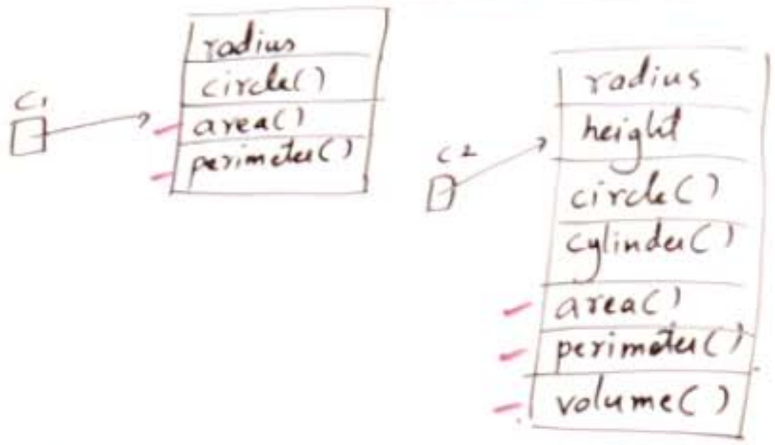
- properties
- Methods

--> Process of acquiring the features of existing class to new class called inheritance.

Ex.: jaisa hmna circle mai height add kar dia to hma cyclinder mil gya.

Method to initialize inheritance:

Inheritance




```
class Circle
{
    private double radius;

    public Circle()
    {
        radius=0.0;
    }

    public double area(){}
    public double perimeter(){}
}

class Cylinder extends Circle
{
    private double height;

    public Cylinder()
    {
        height=0.0;
    }

    public double Volume(){}
}
```

```
class Test
{
    p.s.v. main()
    {
        Circle c1=new Circle();
        Cylinder c2=new Cylinder();
        c1.area();
        c2.area();
        c2.volume();
    }
}
```

--> "extends" keyword ka use kar ka ham method ko initialize karta hai.

```java
// Constructor in heritance //

class parent
{
    public parent()
    {
        System.out.println(x:"Parent Constructor.");
    }
}

class child extends parent
{
    public child()
    {
        System.out.println(x:"Child Constructor.");
    }
}

class grandchild extends child
{
    public grandchild()
    {
        System.out.printf(format:"Grand child Constructor.\n");
    }
}

class inheritancepracc
{
    Run | Debug
    public static void main(String[] args)
    {
        grandchild gc = new grandchild();
    }
}
```

--> Jab bhi chain of constructor available ho and haam " grandchild - class " ko call karat hai to usa sai uper ka jitna hi constructor ho wo sabhi call hota hai top to bottom series mai.

Output

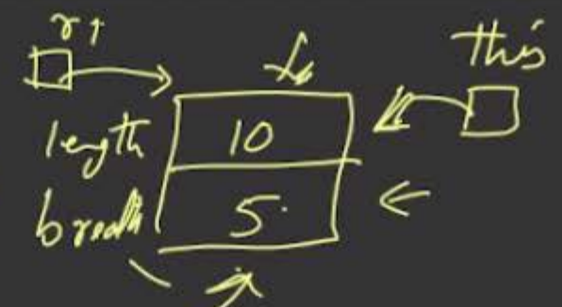
```
PS D:\Java> cd "d:\Java\6 month Java\" ; if ($?) { javac inheritancepracc.java } ; if ($?) { java inheritancepracc }
Parent Constructor.
Child Constructor.
Grand child Constructor.
PS D:\Java\6 month Java>
```

this and super

```
class Rectangle
{
  int length;
  int breadth;

  Rectangle(int l,int b)
  {
     this.length=l;
     this.breadth=b;
  }

  void display()
  {
     System.out.println("Length :"+this.length);

     System.out.println("Breadth :"+this.breadth);
  }
}
```

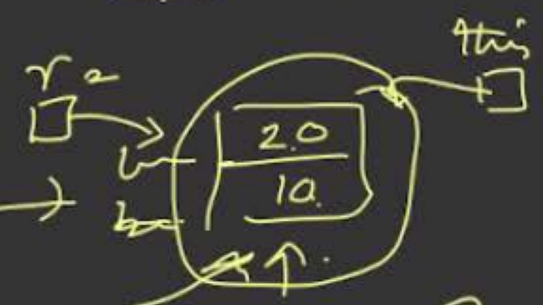
Instance variable

local variable



Rectangle r1=new Rectangle(10,5);

r1.display();



Rectangle r2 = new Rectangle(20,10);

r2.display();

--> " This " keyword ka use tab kiya jata hai jab ham within same class same name ka variable ko create karta hai and haam instance variable ko refer karna chahta hai.

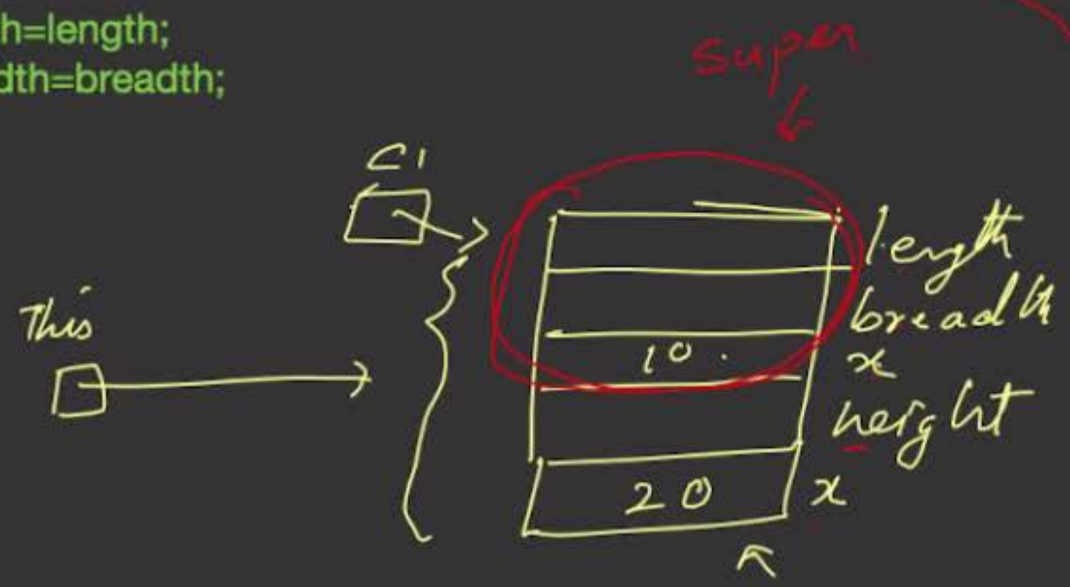
--> Instance class variable ki preference high hoti hai over local variable, iseliya haam isa instance class ka variable koi hi mostly show krana ka liya use karta hai.

--> Define: In Java, " this " keyword is used to refer to the current object inside a method or a constructor.

this and super

```
class Rectangle
{
   int length;
   int breadth;
   int x=10;

   Rectangle(int length,int breadth)
   {
      this.length=length;
      this.breadth=breadth;
   }
}
```

```
class Cuboid extends Rectangle
{
   int height;
   int x=20;

   Cuboid(int l,int b,int h)
   {
      super(l,b);
      height=h;
   }

   void display()
   {
      System.out.println(super.x);
      System.out.println(x);
   }
}
```



→ ***Accessing v/s Invoking***

In coding, "accessing" and "invoking" refer to different actions related to objects and their members (such as fields or methods) in object-oriented programming languages like Java. Here's the difference between accessing and invoking:

1. Accessing:
   - Accessing refers to the act of retrieving or reading the value of a field or property of an object.
   - It is typically used when you want to get the current value of a variable or attribute without changing it.
   - Accessing is associated with fields, properties, or attributes of an object and is often accomplished using getter methods or direct access, depending on the language and visibility modifiers.
   - Accessing does not involve executing a method or changing the state of the object; it is a read-only operation.

Example in Java:

```java
public class MyClass {
    private int myField;

    public int getMyField() {
        return myField; // Accessing the value of myField
    }
}
```

1. Invoking:
   - Invoking, on the other hand, refers to the act of calling or executing a method or function associated with an object.
   - It is used when you want to perform an action, potentially with side effects, that may change the state of the object or return a result.
   - Invoking is specific to methods and functions and is accomplished by providing the necessary arguments (if any) and then calling the method or function.
   - Invoking a method typically involves executing a series of statements within the method's body.

Example in Java:

java Copy code

```java
public class MyClass {
    public void myMethod() {
        // This is invoking myMethod
        System.out.println("This is myMethod.");
    }
}
```

In summary, accessing is about reading values (like fields or properties), while invoking is about executing methods or functions. Both are fundamental concepts in object-oriented programming and understanding the difference between them is essential for writing effective and maintainable code.

→ Super v/s this keyword

*In Java, the "**super**" keyword is used to access members (fields, methods, or constructors) of a superclass (parent class). It is often used within a sub-class (child class) to refer to the members of its immediate superclass. The "**super**" keyword can be used in the following ways:*

1. Accessing Superclass Members:

*** You can use "**super**" to access fields and methods of the superclass. This is particularly useful when the subclass has its own members with the same name as those in the superclass, and you want to distinguish between them.*

java Copy code

```java
class Parent {
    int x = 10;

    void display() {
        System.out.println("This is the parent class.");
    }
}

class Child extends Parent {
    int x = 20;

    void display() {
        System.out.println("This is the child class.");
        System.out.println("Child's x: " + x);     // Access child's x
        System.out.println("Parent's x: " + super.x); // Access parent's x
        super.display(); // Call the parent class's display method
    }
}
```

2. Invoking Superclass Constructors:

*** You can use super to invoke constructors of the superclass. This is typically done in the constructor of a subclass to initialise the inherited members.*

```java
class Parent {
    int x;

    Parent(int x) {
        this.x = x;
    }
}

class Child extends Parent {
    int y;

    Child(int x, int y) {
        super(x); // Call the constructor of the superclass
        this.y = y;
    }
}
```

3. ***Invoking Superclass Method:***

***You can also use super to call a method from the superclass, even if the subclass overrides that method.*

```java
class Parent {
    void print() {
        System.out.println("This is the parent class.");
    }
}

class Child extends Parent {
    @Override
    void print() {
        super.print(); // Calls the parent class's print method
        System.out.println("This is the child class.");
    }
}
```

*The "**super**" keyword is essential for maintaining a clear distinction between the members of the superclass and those of the subclass, especially when they have the same names. It ensures that you can access and manipulate the superclass's members when needed.*

Method Overriding

Definition:
Redefine the method of "super - class" in "sub-class" called method overriding.

```
class Super
{
    public void display( )
    {
        s.o.p("Hello");
    }
}
```

```
class Sub extends Super
{
    public void display( )
    {
        s.o.p("Hello Welcome");
    }
}
```

Method Overriding

su → [display()]

sb → [display() → super | display()]

```
class Test
{
    p.s.v.main()
    {
        Super su=new Super();
        su.display(); —— Hello

        Sub sb=new Sub();
        sb.display(); — Hello welcome
    }
}
```

```
class Super
{
    public void display()
    {
        s.o.p("Hello");
    }
}
```

```
class Sub extends Super
{
    public void display()
    {
        s.o.p("Hello Welcome");
    }
}
```

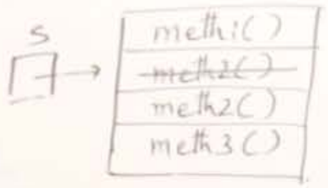
--> method overriding method ko initiate karna ka liya phela hma super class mai ek method bnana hota hai.
--> then same method ko same name sai sub class mai bnata hai. But ise mai uska content ko update kar deta hai.

--> Ham jab bhi mian class mai subclass ka function ko call karenga to memory mai dono method show karega but super class wala method over-shadow ho jayaga and update output show hoga.

```java
class Car
{
    public void start(){System.out.println("Car Started");}
    public void accelerate(){System.out.println("Car is Accelerated");}
    public void changeGear(){System.out.println("Car Gear Changed");}

}

class LuxaryCar extends Car
{
    public void changeGear(){System.out.println("Automatic Gear");}
    public void openRoof(){System.out.println("Sun Roof is Opened");}

}

public class OverridingExample
{

    public static void main(String[] args)
    {
        Car c=new LuxaryCar();
        c.start();
        c.accelerate();
        c.changeGear();
        c.openRoof();
    }

}
```

--> jab ham "extend" ka use kar ka koi program bnata hai to ham "super-class" ka "method" ko "sub-class" ka object bna ka use kar sakta hai. Lekin ise trah sirf "super-class" ka metod hi show hota hai aagar "sub-class" ka method ko access karna ka try karenga to error aayaga.

Dynamic Method Dispatch




```
class Test
{
    p.s.v.main()
    {
        Super s=new Sub();
        s.meth1();  —— meth1
        s.meth2();  —— Sub meth2
        ~~s.meth3();~~
```

Ref

object

```
class Super
{
    void meth1(){s.o.p("meth1");}
    void meth2()
    {
        s.o.p("Super meth2");
    }
}
```

```
Class Sub extends Super
{
    void meth2()
    {
        s.o.p("Sub meth2");
    }

    void meth3(){s.o.p("meth3");}
}
```

--> Dynamic method dispatch useful for achieving runtime polymorphism using method overriding.

--> Dynamic method dispatch mai ham "super - class" ka reference ka use karta hai and "sub-class" ka object create karta hai.

--> agar ham "extends" ka use kar kai program bnata hai and dynamic method dispatch ka use karta hai to heap mai "sub-class" ka sabhi method load ho jayenga but ham only "super-class" ka method ko use kar sakta hai.

➢ Do's and Don'ts of Overriding

- Signature must be same in method overriding.
- If the method name is different the method is not overridden but it is overloaded.
- Argument may be different but the parameter must be same.
- Return type must be same, if it is not same then the method is neither overridden nor overloaded.
- Final and static methods cannot be overridden.
- Method can be overridden with same or lenient (public, protected) access specifiers but the stricter(private) access specifiers cannot be used in sub class.

```java
package overloading;

class Test
{
    public int max(int a,int b)
    {
        return a>b?a:b;
    }

    public int max(int a,int b,int c)
    {
        if(a>b && a>c) return a;
        else if(b>c) return b;
        return c;
    }
}

public class Overloading
{
    public static void main(String[] args)
    {
        Test t=new Test();
        t.max(10,5);
        t.max(10,15,5);



    }

}
```

```java
package override;

class Super
{
    public void display()
    {
        System.out.println("Super Display");
    }
}

class Sub extends Super
{
    public void display()
    {
        System.out.println("Sub Display");
    }
}

public class Override
{
    public static void main(String[] args)
    {
        Super s=new Sub();
        s.display();

    }
}
```

➢ Polymorphism using Overloading and Overriding

- Polymorphism is one of the principles of Object-oriented-programming, polymorphism means one name different actions.
- Poly means 'many', morphism means 'forms'.
- Polymorphism is achieved using method overriding and overloading.
- In method overloading access specifiers, return types are same but number of parameters or type of parameters are different.
- In overloading number or type of argument will decides which method is to be called.
- Overloading is achieved in same class whereas overriding is achieved in inheritance.
- In method overriding signature is same but in overloading signatures must be different.
- Method calls are different in overriding it depends on object.
- Method overriding is used for **runtime polymorphism** and method overloading is used for **compile time polymorphism.**