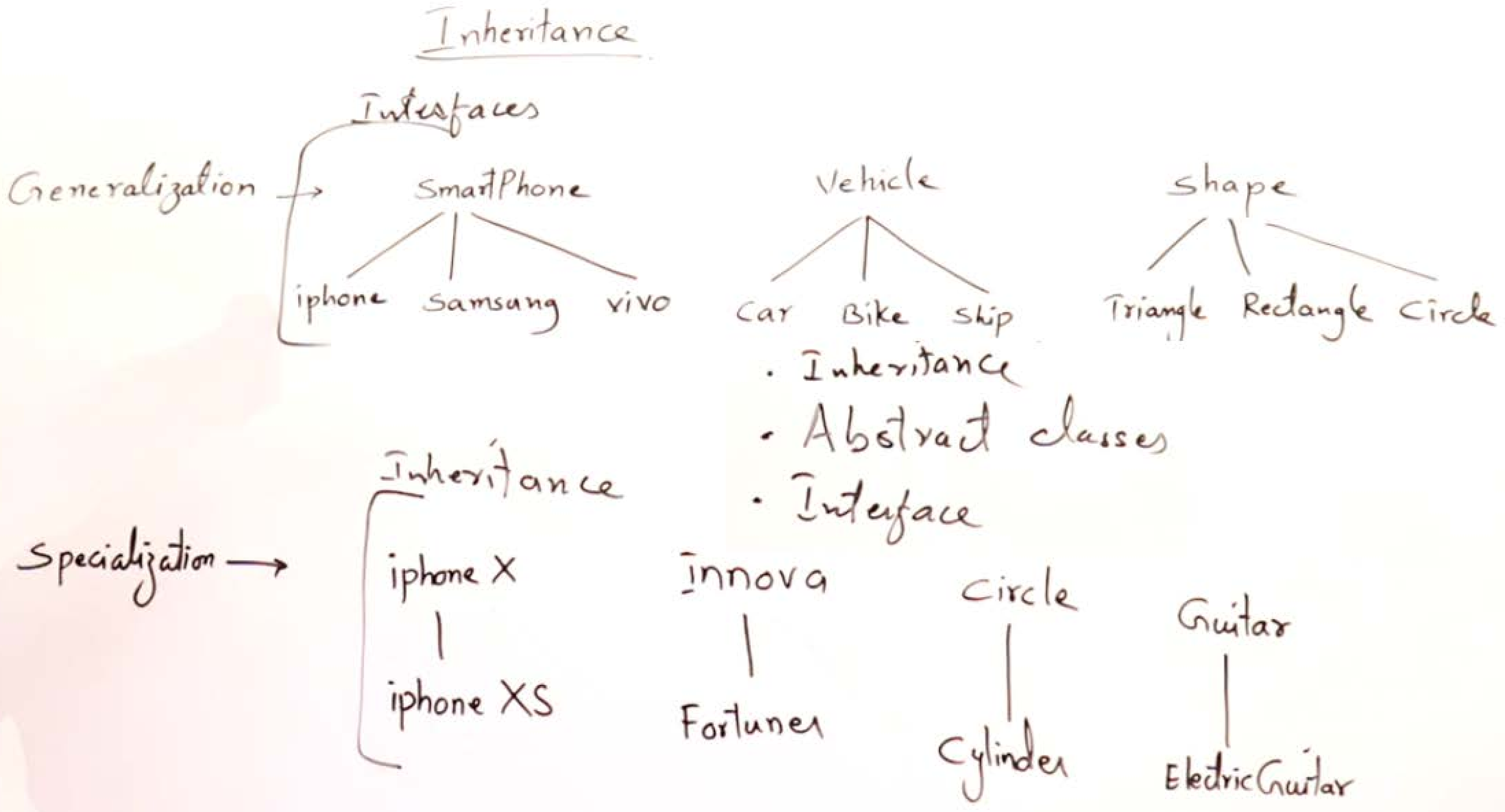


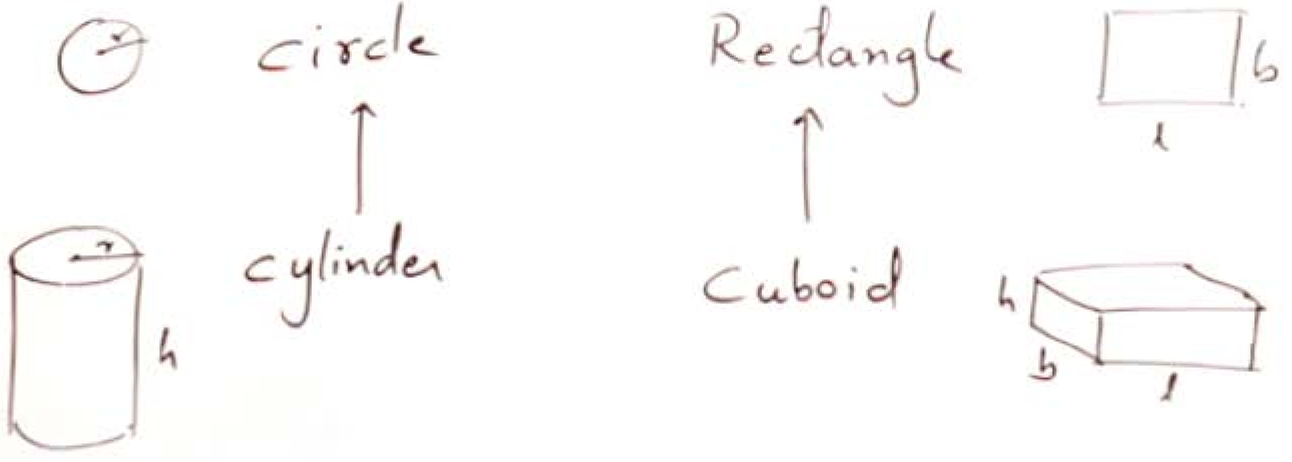
--> * Generalization mean, " <u>group of class</u> " ko kisi common name sai call karna.

* 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

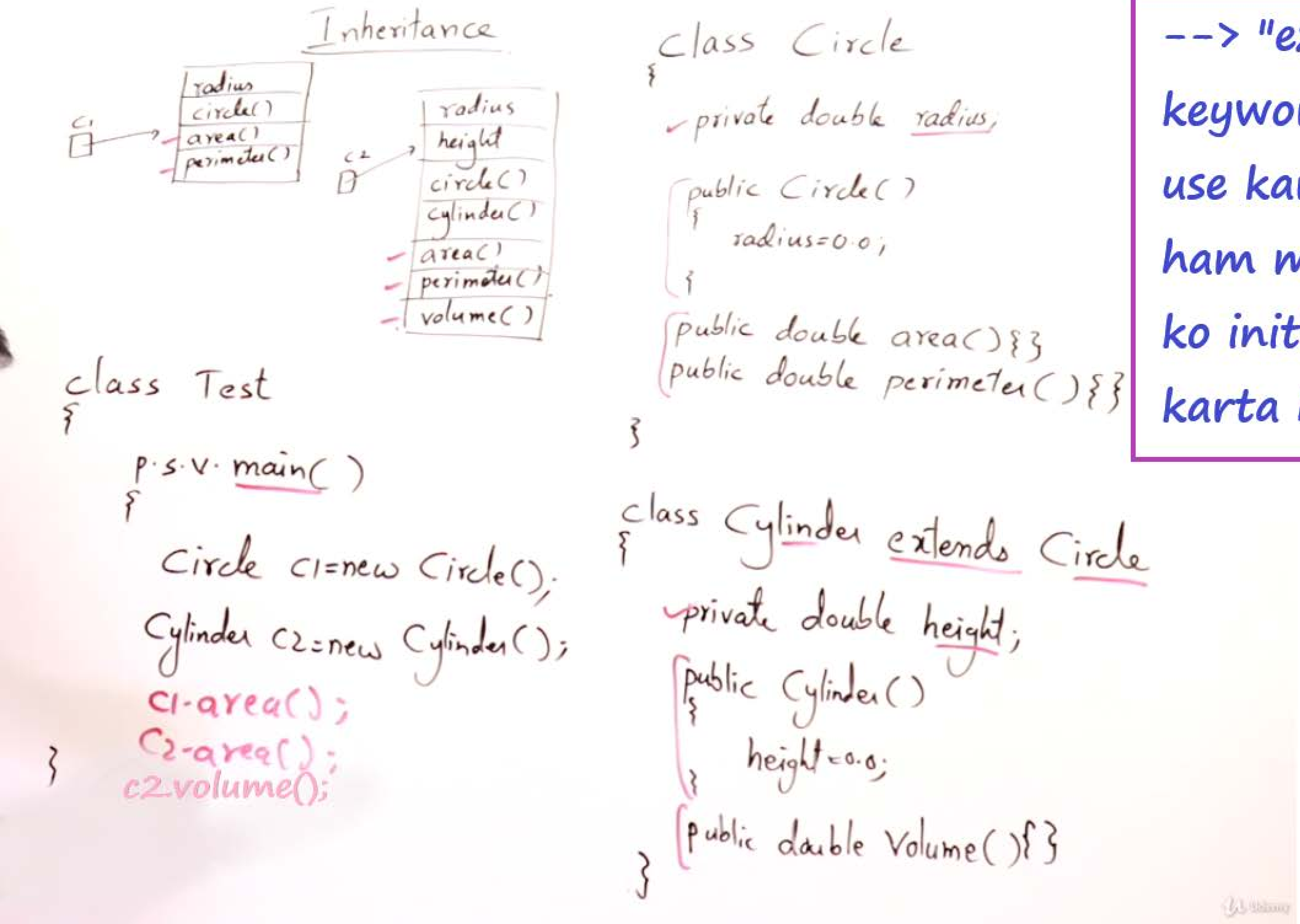


- properties
- Methods

--> 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:



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

```
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(){}
}
```

--> "extend's" 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

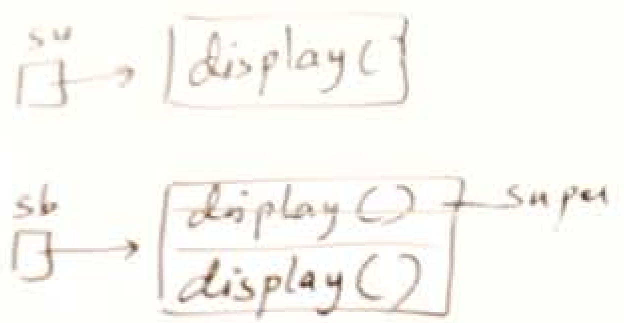
```
PS D:\Java> cd "d:\Java\6 month Java\" ; if ($?) { javac inheritancepracc.java } ; if ($?) { java inheritancepracc }
Parent Constructor.
Child Constructor.
Grand child Constructor.
PS D:\Java\6 month Java>
```

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



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

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

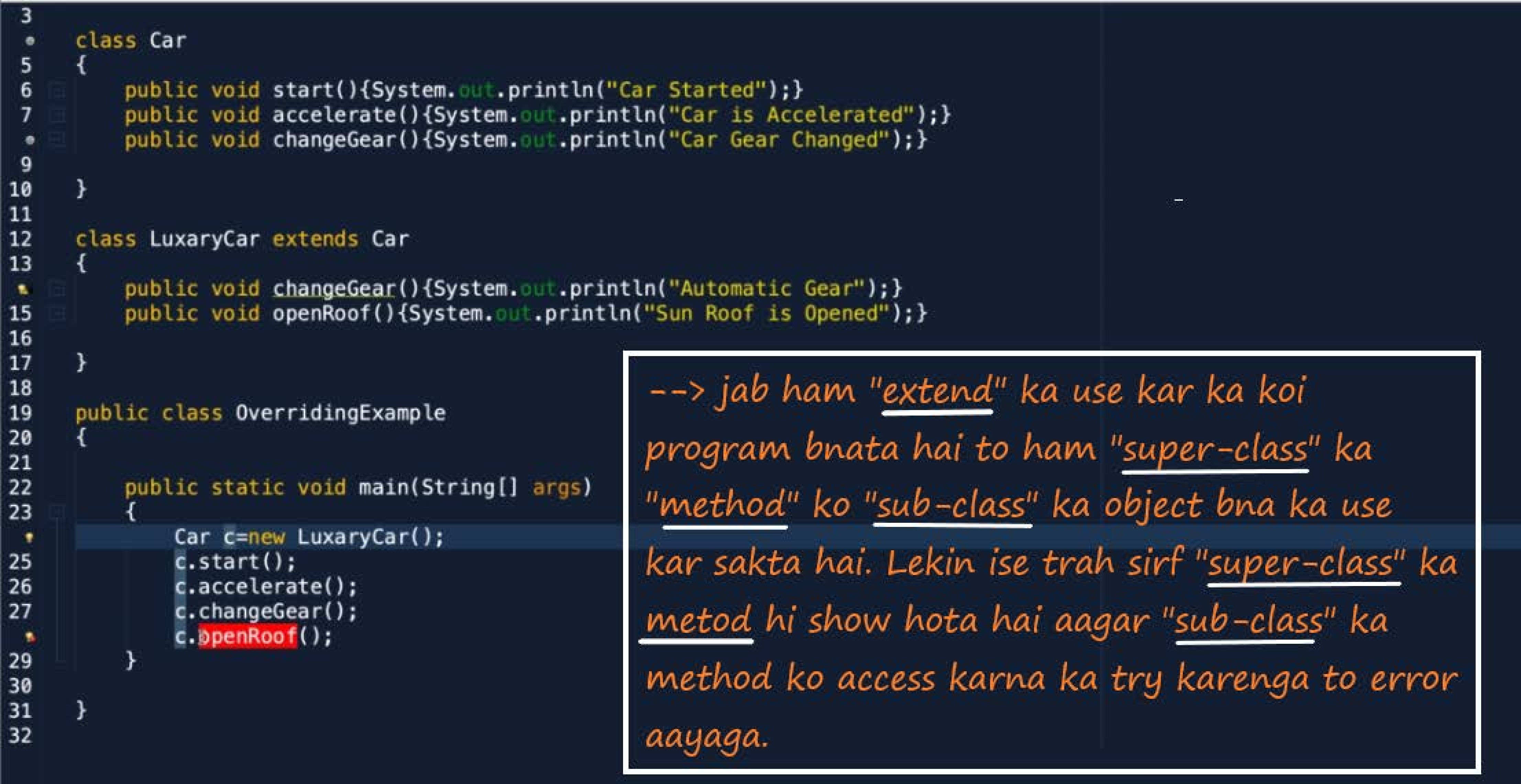
```
class Super
{
    public void display()
    {
        s.o.p("Hello");
    }
}
```

```
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.

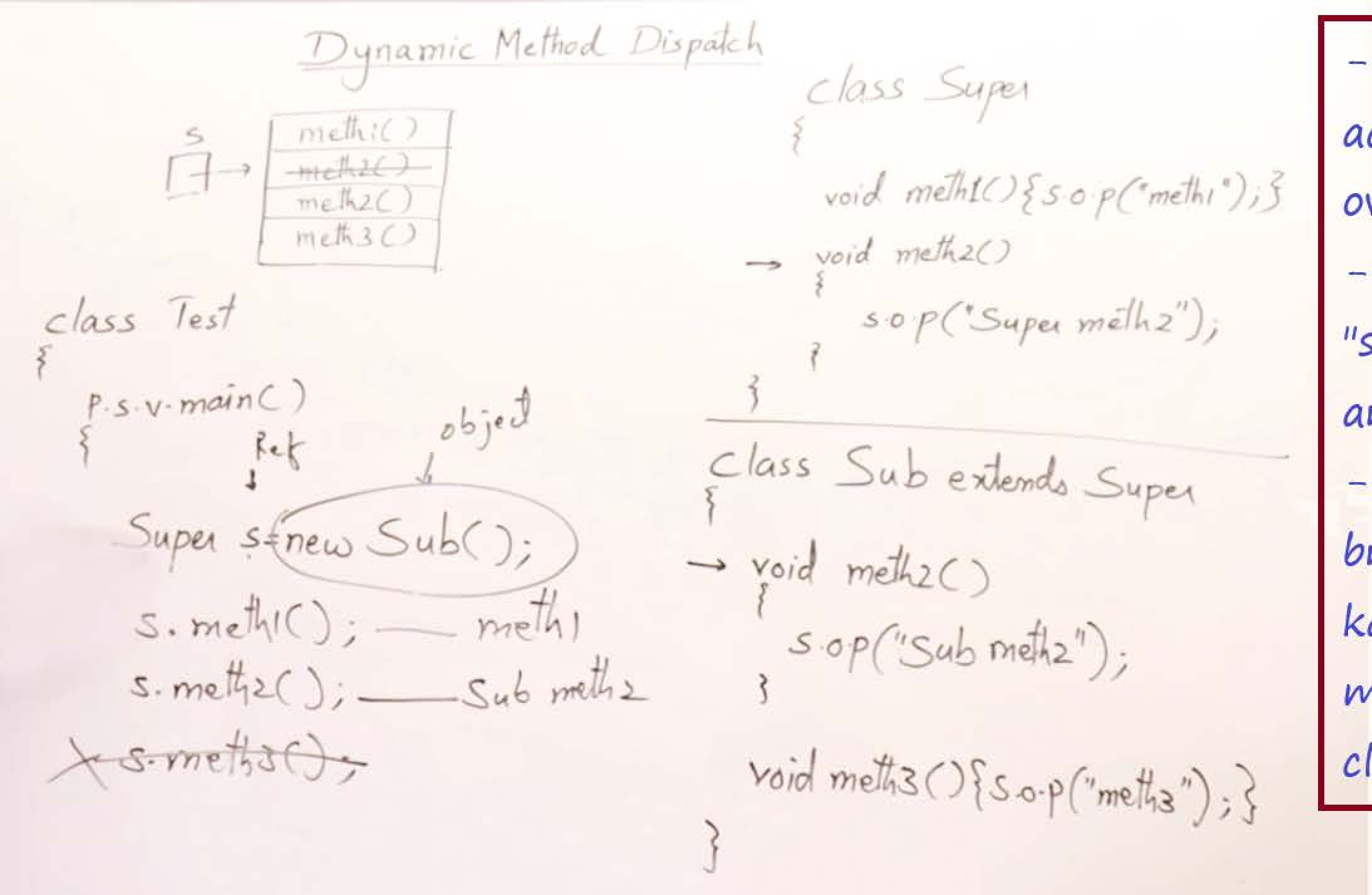
--> then same method ko same name sai sub class mai bnata hai. But ise mai uska content ko update kar deta hai.

--> 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

```
s
[ ] -> | meth1()   |
       | ~~meth2()~~ |
       | meth2()   |
       | meth3()   |

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

```
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");}
}
```

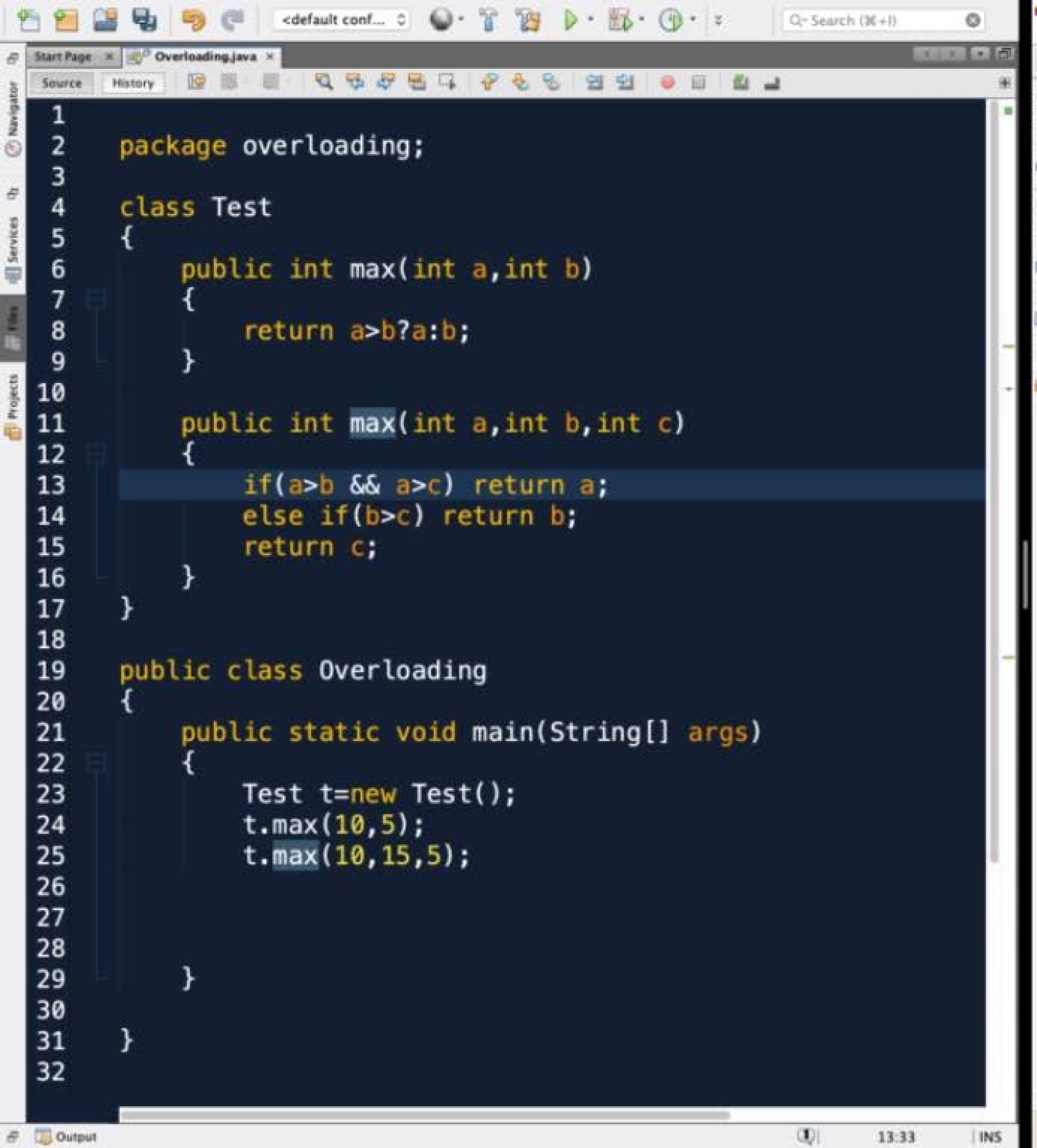
--> Dynamic method dispatch useful for achieving runtime polymorphism using method overriding.

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

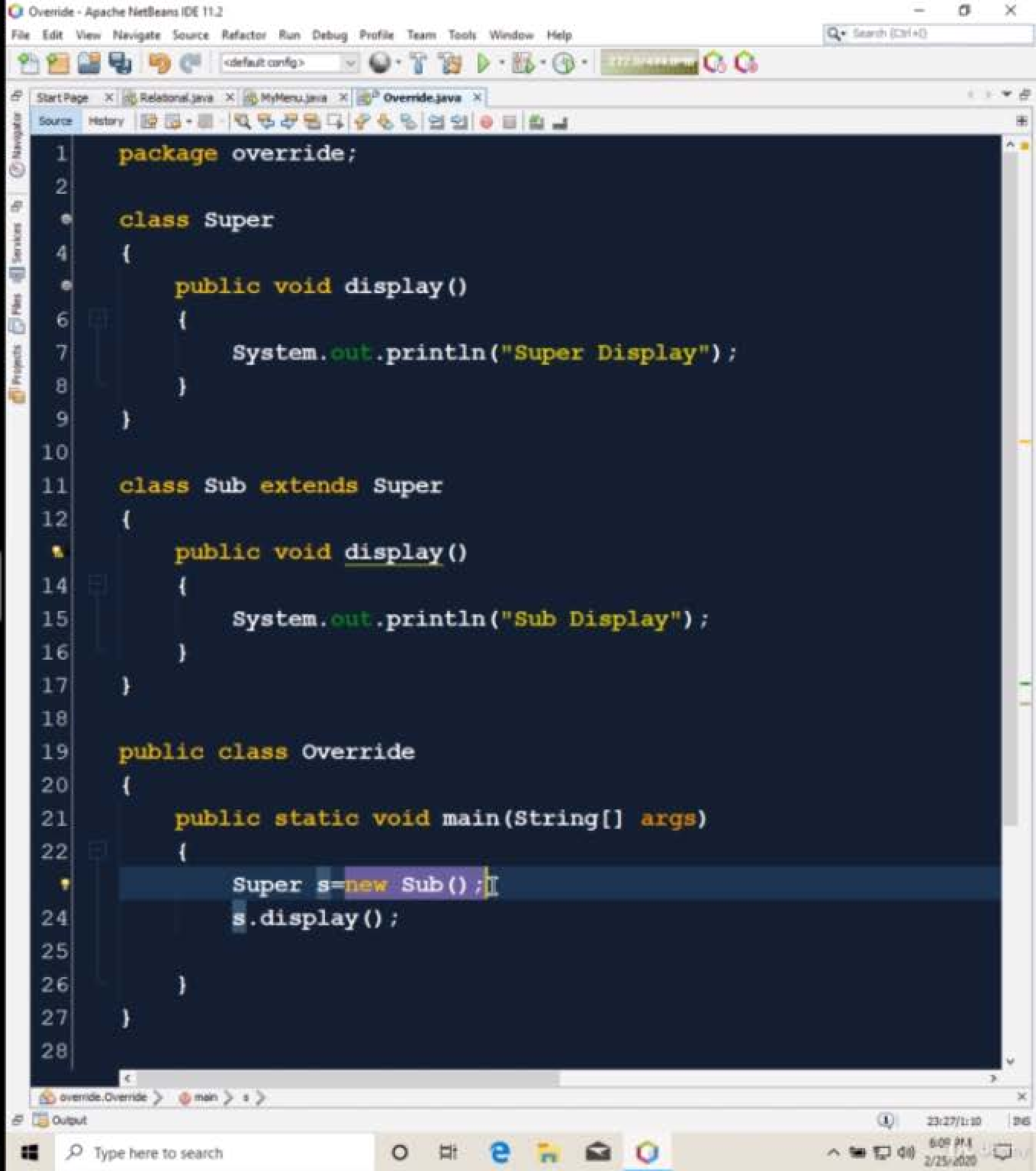
--> agar ham "extends" ka use kar kai program bnata hai and dynamic method dispatch ka use karta hai to heap mai "sub-class" ka sabhi method load ho jayenga but ham only "super-class" ka method ko use kar sakta hai.

➢ 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.**