Constructor Chaining,
*** Super() method
*** Super Keyword

Unit-2

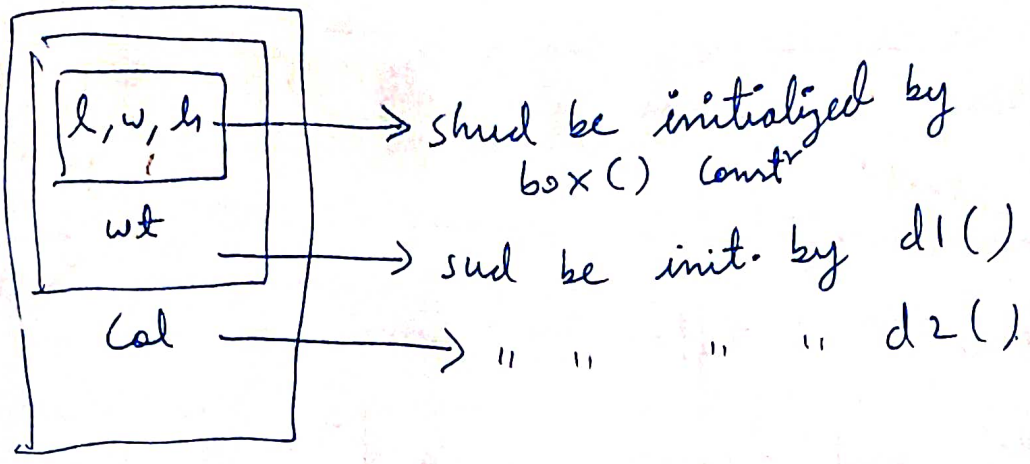
baseC → base class
derC → derived class
dC → def. constructor
PC → Parameterized Const.
NPC → Non Paramzd Const.r
CC → Const.r Chaing
Const.r → Constructor.

Points

1) A derC const.r will always call its baseC const.r. This is called Const.r chaining (CC)
2) Order of const.r calling will be Top to bottom

Why does CC occurs?
Why does a to derC const.r calls a baseC Const.r

- A const.r initializes the values of var in an obj. When a derC obj is created, it has var of baseC as well as derC
- Var of derC shud be initialized by derC const.r & var of baseC shud " " baseC Const.r.



derC obj

---

example of Constr Chaining (CC)

- In this eg, baseC of d2 is d1.
  box is not baseC of d2.
- baseC of d1 is box.
- when obj of d2 is created,
  d2() will be called.
  d2() then calls d1()
  bcoz d1() is baseC of d2.
  d1() then calls box()
  bcoz box is baseC of d1.
- So order of Constr will be
  box() → d1() → d2()

How CC occurs for diff types of Constr

There r 3 types of Constr.

1) Def. Constr - created by compiler
   (DC) - No parameters
   - Doesn't perform any task

2) Non Parameterized Constr - created by user
   (NPC) - No param.
   - Can perform some task

3) Paramzd const - created by user.
   (PC) - ~~st~~ Paramzd.
   - Can perform task.

---

1) For DC & PC,
   Constr chaining occurs automatically. wh. means derC constr auto' calls baseC constr.

2) For PC,
   - CC can't occur auto', bcoz baseC has PC
   - If baseC Constr is called, then values/param need to be specified
   - for this reason, PC in baseC can't be called auto' by derC Constr.

- To call PC of baseC & pass values/params super() method is used.

What is super() & why is it used

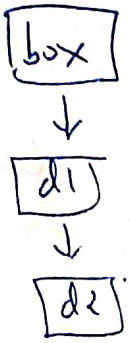
- When baseC has a PC (paramzd Constr), then derC constr must call & pass values to the baseC constr by using super()
- super() method calls the constr of baseC & passes values to it.
- super() is used bcoz it calls & passes values to the baseC constr.

When is super() method used & where is it called

**: Whenever, there is a "Paramzd Constr in baseC", then super() must called.

- super() is called inside the derC constr & not inside baseC constr.
- Inside derC constr super() calls baseC constr & passes values to it.

Example-1 : Const^r Chaining w/ default Const^r.



- In this example, no const^r r created. Compiler will b/w a dc (def. const^r) in each class.
- dc of d2 (d2()) will call dc of d1()
  then
  " " d1 " " " " box().

```
class box {
    //No const^r created
    set ( ) { - - - //defined }
    get ( ) { - - - //defined }

}
```

class d1 extends box{
    //No constr created, compiler will
    p/v a dc
    set () { // define set //}
    get () // define get
}
class ~~box~~ d2 extends d1 {
    //No constr. dc will be created.
    //define set, get.
}
C D { P S V M }.
    //create obj of d2
    d2 obj-d2 = new d2();
                ↳ d1() → box();
//There r no constr in d1, d2, box()
  So compiler will create DC
// DC of d2() will be called
then " " d1() " " "
" " box() - - - -
// Order of execution will be top to bottom
    box() → d1() → d2().
//No tasks will be performed by dc.

Example2: Constructor Chaining with Non Paramzd Constr.

- In this eg, create NPC in each class.

```
box Class box{
    //NPC - created by user, No param.
        box(){
            Sopln("Box called");
        }
}
class d1 extends box{
        //NPC of d1
          d1(){
              Sopln("d1 called");
          }
}
Class d2 extends d1 {
    //NPC of d2
      d2(){
          Sopln("d2 called");
      }
}
```

```
C d { PSVM(){
//create obj of d2
//d2() will "auto" call d1"
//d1() "    "    "    d2"
//order :- top to bottom - box() → d1 → d2
```

```
d2 obj-d2 = new d2();
                ③ ↳ d1() → box();
                    ②      ①

//o/p : box called
        d1   "
        d2   "
    }
} // demo ends.
```

Note:- In both eg, derC constr auto⁷ calls baseC constr.

Example 3: Constr Chaining w/ Parameterized Constr.

- In this eg, create a param Constr (PC) in each class wh. will initialize the values.

```
class box {
    Protected int l, w, h;
    box(int x, int y, int z) {
        l = x;
        w = y;
        h = z;
    }
}
```

PC in box class.

```
class d1 extends box {
    protected int wt; // new variable
//* * * * * * * * * * *//
```

- There is a PC in baseC, so this derC constr must call & pass values to the baseC constr.
- The baseC will not be called autoly, it will be called using super().
- BaseC constr has 3 param, so super will also have 3 param.
- super() "must" be the Ist statement inside derC constr.

```
// define d1()
d1 (int x, int y, int z, int w) {
        super(x, y, z);
        wt = w;
}
```

// BaseC constr is not called autoly. Its called using super.

// d1 will assign/initialize only the var in its own class.
// it will pass remaing var to box( )

// d1 has class has 4 var - l, w, h & inherited from base & wt. is created inside d1
// d1 will assign value to only its own var i.e wt & it will pass remaining values to the base constr thru super();
// then baseC constr will initialize var of baseC.

* what is happening here is that d1() is only initializing var of its own class (i.e wet = w) & remaining var will be initialized by base const when super() is called.

} // d1 ends.

class d2 extends ~~box~~ d1 {

protected int col;

// Then is a PC in d1, so d2() must call super.

d2 (~~int x~~, int y, int z, int w, ~~String s~~) {

super (x, y, z, w);

col = s

}

}

~~te~~ c ~~d~~ { p s v m (--) }

// create obj of d2.

// It has PC, so make sure to pass values.

d2 obj_d2 = new d2(1, 2, 3, 4, "Red");

↳ d1 (1, 2, 3, 4)

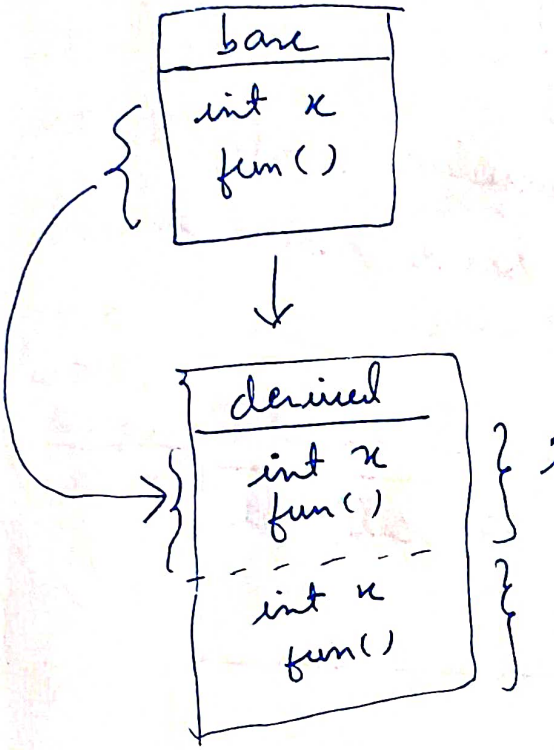
↳ box (1, 2, 3)

}

}

Super Keyword

- Super kw is used to access the members of a baseC wch have same name in derC.



- Suppose a baseC has a var x & a fun()
- The same var & fun is used/created in derC
- New derC has 2 members w/ same name.
- To access members of baseC in derC use super.membername

eg super.l ; } members of
super.fun(); } baseC inherited in derC

l ; } members created in
fun() } derClass.

Use of super kw

- Super kw is used to resolve ambiguity when any member has same name in baseC & derC

Example of super kw to call a baseC fun

```
class box{
    protected int l, w, h;
    public void set(int x, int y, int z){
        l = x;
        w = y;
        h = z;
    }
}
```

class d1 extends box{

protected int wt; // l, w, h r inherited.

// set can be defined in 2 ways here

① set initializes all the var of d1 class.

i.e l, w, h, wt

② set initializes only its own var & calls the set fun of baseC to initialize other var

public void set (int x, int y, int z, int w){

l = x;
w = y;
h = z;
wt = wt;

(1)

(or)

super.set(x, y, z);
wt = w;

(2)

initialize only own var.

}

// wt in the ② way, our effort is reduced & we have reused the set fun in base class

// Thus super keyword offers reusability.

} // super is used because both fun have same name i.e set.

```
class d2 extends d1 {
    protected String col;
// similarly, set can be defined in 2 ways here:
    public void set(int x, int y, int z, int w, String s) {

l = x;                  :   super.set(x, y, z, w);
w = y;                  or  (// call set of d1)
h = z;                  :
wt = w;                 :
col = s;                :       col = s;

    }
}

class c {  psvm(...) {
        d2 obj_d2 = new d2();
        # obj_d2.set(1,2,3,4,"Red");
    }
}.
```

Super() method	super Keyword.
• It is a fun	• It is a Keyword.
• Purpose: - is to call const^r of the base class.	- is to resolve ambiguity
• Used when: - base class has a parameterized const^r	- base & derived classes have members with same name
• accesses the base const^r	• accesses members of base class.