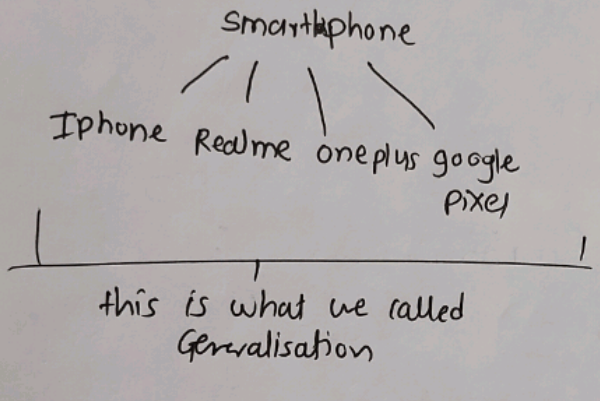
⊕ Inheritance in Java Programming language.

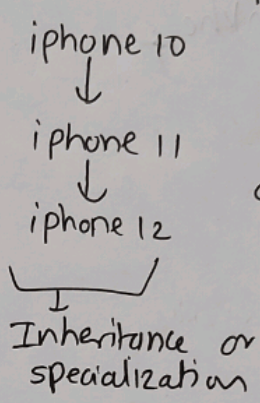
Ⓐ Generalisation & Specialization

⇒ Generalization deals with grouping multiple things together & form a generalised set of something (Overloading) & (polymorphism) mostly deal with the ~~inheritan~~ Generalization

⇒ Specialization is taking Particular things and making progress & with that and do modification & develope subsequent version such that every new version has best characteristics from it's previous once.

Ex:





Specialization will for multilevel heiarchy of things

Also

A → Based class / Parent / Superclass

B → Derived class / child / Subclass

Superclass has subsequent subclasses

⇒ Both Generalization & Specialization will form heiarchy.

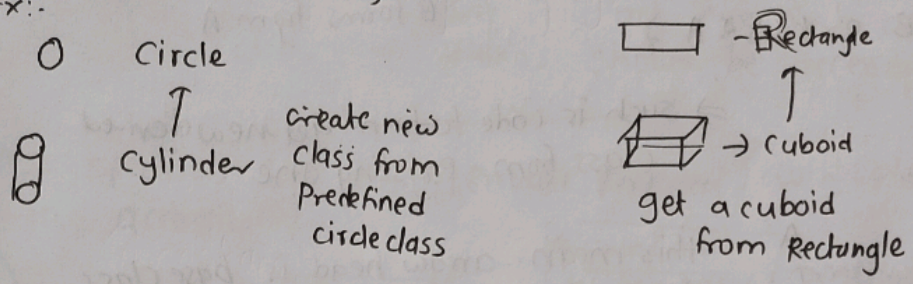
⇒ In java Interfaces are there to achive generalization & for specialication we will use ~~Interfaces~~ Inheritance

⇒ Abstract classes is used as middle ground betn Generalization & Specialization

⊛ Inheritance in java

It is a process of acquiring feature of existing class to a new class i.e (borrowing of features)

Ex:-



So here we borrow all properties of parent & we can able add new one also.

⇒ keyword used for deriving Extends

```
class Circle {
    private double radius;
    public double area(){
    ;
    public double circumference(){
        ——— Code ———
    }
}
```

Syntax

Derived class extends Base class

```
Derived class      Base class
class Cylinder extend Circle {
    private double height;
// and cylinder has differ formula for area & perimeter
    Public double area(){
        return height x
            — Code ———
    }
    ;
}
```

& `public double volume{ ——— //—`

⇒ above is example how you can able to ~~find out~~ get d parent & child classes from ~~them~~ form new derived class

```
class A{
}
class B extends A{
}
```

Extends → keyword for inheritance

A
↑
B A Derives B

B forms from A

⇒ such is code to form get new derived class from existing one

→ notation

A
↑
B this mean arrow head is base class & arrow tail is derived class

Parent class / superclass / Base class

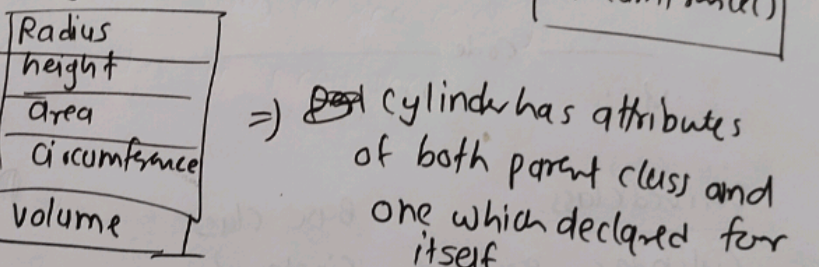
↑ ⇒

Child class / Subclass / Derived class

⇒ Reference for circle & cylinder class

Circle C1 = new Circle();

Cylinder C2 = new Cylinder();



⇒ cylinder has attributes of both parent class and one which declared for itself

(Imp)

⊕ So say you have function area in base class having some code X1 & in it's derived class you will right same function name area() having code y1 & X1 ≠ y1, so for derived class on calling aread code y1 will invoked & if we not defined area() in that class (Derived) so on calling area it will automatically call function area() from base class due to inheritance

⇒ Also you have to learn about 2 terms accessible & available.

A Parent class

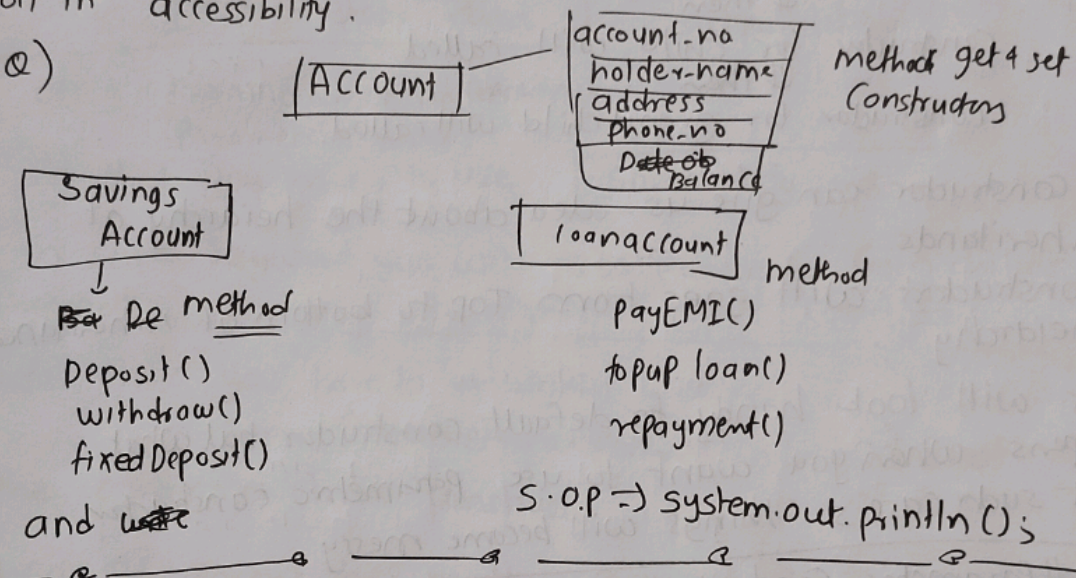
↑

B Child class

so one doing inherited derivation on all things from Parent will be available to child but it is not case that all available things should be accessible

Access specifiers (i.e public, private, protected) will play big roll in accessibility.

Q)



S.O.P => System.out.println();

⊛ Constructors in inheritance

let see a code

```
class Aa{

    Pubic Aa(){
        S.O.P.("Parent Constructor");
        }
}
class Ab extends Aa{

    public Ab(){
        S.O.P("child Constructor");
        }
}
```

so now use main function an find out how what will happen if we create a object of class Ab

Output

Parent Constructor
Child Constructor

So from this we can state that if we create a inherited class then for that first base class object is constructed & then inherited one.

@ flow Ⓐ Create object or derived(child) class
↓
Ⓑ constructor for base class invoked then
Ⓒ ↓ constructor for b child class is invoked.

so Ex: you have 3 constructors defined in
class Parent { } class child { } class grandchild { }

so constructor of parent will called
⇓ then
Constructor for child will called
⇓ then
Constructor for grand child will called.

⇒ Constructor can give us idea about the heiarchy of inheritands

⇒ constructors will goes from Top to bottom of inheritance heiarchy

@this will look handy for default constructor but what happens when you want to use parametric constructor at such cases things will become messy

---

@ #Parametric Constructor in inheritance

DO see coding part

Ex: class A {
public A() {
S.o.println("non parametric parent");
}
public A(int x) {
S.o.p("parametric parent");
}
Child B extends A {
public B() { s.o.p("nonparametric Child"); }
Public B(int x) {
s.o.p("parametric child");
}

so If you do → noparametric

⊙ B a = new B();

O/P:→ nonparametric parent
non-parametric child

⊙ what happen if you call

B a = new B(10);

O/P:→ nonparametric parent
parametric child.

€ so such things will happens & so what if we want output like

O/P ⇒ parametric child Parent
parametric child

for that you have to use [super] keyword

in super keyword you will pass parameter from child's parametric constructor to parent's parent constructor

IMP for that you have to write

super (parameters to parent) in first line of the parametri child constructor

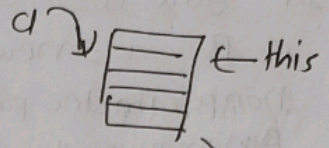
⊛ so If Parent has multiple parametric constructor so at that time type and count of parameters in child's super will play important role

→ super(); should alway at first line

→ super will acts like agent to pass values as argument from child constructor to parent

this Vs super keyword

this is reference to current object



so

Ex Class a = new class();

this will create a block

so inside class this will also used to reference same object

⇒ the keywords this & super only used inside class declaration

Ex → without this

```
class A{
int a;
int b;
A(){
a = 10;
b = 11;
}
A(int h, int g){
a = h
h = g
};
}
```

If you create something in class without access specifie it will private by default

```
class A{
int a;
int b;
A(){
this.a = 10;
this.b = 11;
}
A(int h; int g){
this.a = h;
this.b = g;
}
}.
```

→ ~~super~~ Both are same

→ this keyword used for self class substance

→ this reference to object which currently called.

→ It is not case that you always have need to go for using this every time

⇒ so if you have same Parameter name that of Properties at such case to distinguish betn data members & parameters we use this

Ex:
```
class A {
int a; int b;
A(int a, int b) {
a = a;
b = b;
}
```
↑ this will give error so in order to deal with it do

```
class A {
int a; int b;
A(int a, int b) {
this.a = a;
this.b = b;
}
```
this is how you may be able to solve your issue

⊛ Super

~~this~~ is also reference to super class that is parent class

→ this is for self | To avoid name conflict this & super used.

So If you have a variable of same name in current & Parent class so

super.variable ⇒ variable from parent class

variable ⇒ variable from child class (current class)

→ super is used to pass values to parent class and also can be used to fetch value from it's parent too.

```
class A {
int x; int y;
A(int x, int y) {
this.x = x;
this.y = y)
}

class B extends A {
int x;
B(int a, int b, int c) {
super(b, c)
this.x = a
}
```
→ To ~~not be used~~ used

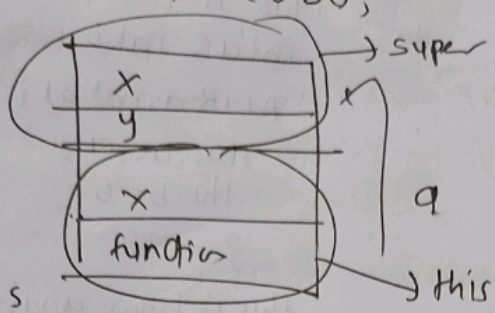
So

```
void function() {
S.O.P.(super.x)
S.O.P(x)
}
```

So super.x will point to parent

this will point to itself

So B a = new B();



super will point to it's parent into

this will point to class which invoked.

this is for self class

super is for parent class.

~~this is more even the type specific~~

Method overriding

It deal with function name, so redefining a function which is also previously declared in parent class

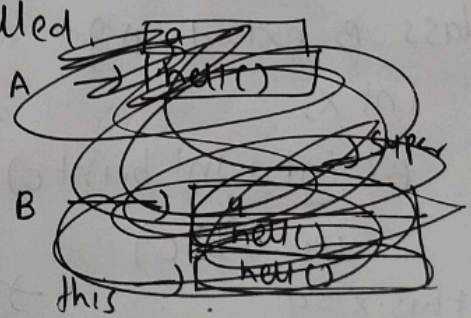
```
class A{
   int a;
public void hell(){
     s.o.p.("Help");
   }
}
class B extends A{
   int c;
public void hell(){
     s.o.p("Suffer");
  }
}
```

so we declared same function hell in child & previously in parent

This is method overiding declaring function in child class which is previously installed in parent class

→ so when object of Parent calls that function then function from parent will called

⇒ When object of child calls the function then function from child will be called



A a = new B();

So which class will allocated to a.

⇒ Here one declared after new will be alloted

So

A → is reference

B() ⇒ object

So object created first and then assigned to a reference

this is called dynamic method dispatch

the no. of parameter will also affect method overiding

→ If parameter involved then both method overloading & method overriding will come to play

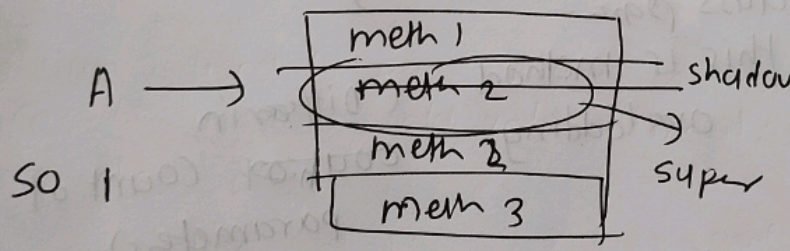
Reverse like (Child c = new Parent();) this is not allowed

Reference is parent object is child

⊕ Dynamic method dispatch

Parent A = new child();

↑ Reference ↑ object



So !

```
class Parent {
public void meth1(){}
public void meth2(){}
}
class child extends Parent {
public void meth2(){}
public void meth3(){}
}
```

meth 2 is hidden that is masked due overrided meth2 in child class

So so

```
A.meth1();
A.meth2();
```

But

```
A.meth3()
```

↑ It is in that A block but can't be able to access

Even method is there we can't access !! as Reference is of Parent class

so ~~method~~ you can Reference Parent class to a child Object but able to access that function only which are present in parent class even though functions of child class present there.

So now

S.meth2()

Which function will be called?

the meth2() from child will called.

→ so with dynamic method dispatch if you reference a parent variable to child object and if it consist a overrided function so with that reference you will access method from child (overrided method)

---

* Does & Don't in method overriding

display(); → int class Parent

display(int x); ⇒ this is method overlodding (Differ in count of parameters)

↑ In child class

display() → in Parent class

display(); in child class → display(int x);
→ display(in y);

method overiding

Static & final methods cant be overrided.

→ you have to be carefull about access Specifiers

Private
protected
Public ↓

order of Strict to loose