```
Void m1 (int ...v){_}

Void m1 (boolean ...b){_}
```

m1();

↓

ambiguity

↓

runtime error

11/3/2024

Inheritance

→ 'Object' is Super most class of each class in java.

→ Class A{

}

Object ⟶ A

→ for inheritance, we use 'extends' keyword

Object
↓
A
↓
B

Object
↓
C

```
Class A{
    int i=1;
    void m1(){
        SOP("m1");
    }
}
Class B extends A
{
    int j=2;
    void m2(){
        SOP("M2");
    }
}
```

```
Class C{
    PSVM(_){
        A a1 = new A();
        B b1 = new B();
        SOP(b1.i);
        SOP(b1.j);
        b.m1();
        b.m2();
    }
}
```

Polymorphic References.

What relation can A & B have

```
Class A{
}
Class B extends A{
}
```

① A Superclass reference can refer object of Subclass.

```
① { A a1;
    B b1 = new B();
    a1 = b1;

② { A a1 = new B();

③ { A a1 = new A();
    B b1 = new B();
    a1 = b1;
```

All are same.

② A Subclass reference cannot refer object of Superclass.

```
A a1 = new A();
B b1 = a1;      ✗ Compile time Error
```

③ A Subclass reference cannot refer object of Super Class but it can be Casted. {Conditional}

```
A a1 = new A();      Compile ✓
B b1 = (B) a1;       Run ✗
                     ↓
                     {Class caste exception.}
```

④ A Super class Object can be casted in Subclass only if that super class reference is refering Object of Subclass.

```
A a1;
B b1 = new B();
a1 = b1;
B b2 = B(a1);
```

Compile ✓

Run ✓

{Consider example of Vehicle}

```
Class A{
    string msg = "A";
    Void m1(){
        SoP("M1");
    }
}

Class B extends A{
    String msg = "B";
    Void m1(){
        SoP("M1 of B");
    }
}
```

```
A a1;
B b1 = new B();
a1 = b1;
SoP(a1.msg); // A
SoP(b1.msg); // B
a1.m1(); // M1 of B
b1.m1(); // M1 of B
```

a1.msg → Reference to A
a1.msg → Object of B

In Super - Sub Class case.
Data members binding → Compile time (Early binding)
Methods binding → Run time (Late binding)

When there is Subclass Object in Super class then Binding of data members happens at Compile time hence data member of type of obj. reference gets called. whereas binding of reference is done at run time hence type of method of object type gets called.

⑤

```
Class A{
    int i=1;
    Void m1(){
        SoP("m1 of A");
    }
}
Class B extends A{
    Void m1(){
        SoP("m1 of B");
    }
    void m2(){
        SoP("M2 of B");
    }
}
```

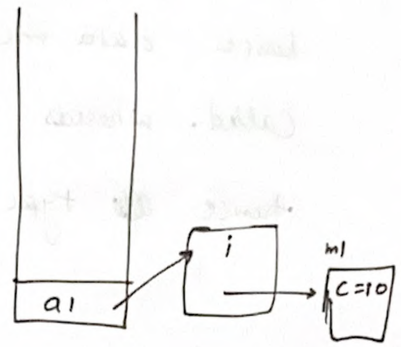
```
A a1;
B b1 = new B();
a1 = b1;
a1.m1(); //m1 of B
a1.m2(); // error.
```

By superclass,
∧ You cannot call methods of Subclass which are not declared in Superclass

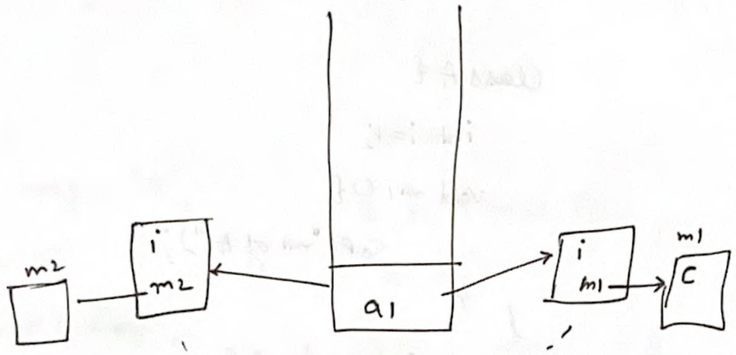
12/3/2024

```
Class A{
    int i;
    void m1(int c){
    }
}
```

```
A a1 = new A();
a1.m1(10)
```




```
Class A{
Static int i;
    int j;
Void m1(int c){
}
Static void m2(){
}
}
```



That's why we can't access non-Static member/methods in static methods.

```
Class A{
    int i=1;
    void show(){
        SoPC("Show of A");
    }
}
```

```
Class B extends A{
    int i=2;
    Void show(){
        SoPC("Show of B")
    }
    Void m1(){
    }
}
```

```
Class C{
    PSVM(_){
        A a1 = new B();
        SoP(a1.i)  // i of A
        a1.show()  // Show of B
        a1.m1()  // error
```

```
Class Box{
    double w,h,d;
}
Class BoxWeight extends Box
{
    double wt;
    BoxWeight(double w1, double h1,
              double d1, double wt1){
        w = w1;  d = d1;
        h = h1;  wt = wt1;
    }
}
```

```
BoxWeight bw = new BoxWeight
                (2,3,4,2.5);
```

⇓

→ When object of sub class is created, Subclass Constructor call Constructor of super class automatically.

→ By default no-argument / default Constructor of super class is called, if we want to call any other Constructor of super class it must be called by using 'super' as first statement in the Constructor of subclass.

```
Class A{
    int i;
    A(int i){
      this.i = i;
    }
}

class B extends A{
    int j;
    B(int i1; int j1){
        i = i1;
        j = j1;
    }
    Void show(){
        SoP(i + " " + j);
    }
}
```

```
Class C {
    PSVM(__){
        B b1 = new B(10,20);
        b1.show();
    }
}
```

Output => error

This will try to call A's Constructor as nothing is mentioned it will call default constructor which doesn't exists as we already have a constructor in A

```
Class B extends A{
    int j;
    B(int i1, int j2){
        super(i1);
        j = j1;
    }
}
```

To pass value to Super class via constructor we need to use super keyword & it must be the first statement of the constructor aswell

13/3/2024

Super

→ To call Super classes Constructor

→ To access Super Classe's members.

Example → ①

```
Class A {
    int i=1;
}
Class B extends A {
    int i = 2;
    Void show(){
        SoP(super.i);  // 1
        SoP(i);  // 2
    }
}
```

②

```
class A
{
    int i=1;
}
Class B extends A{
    int i=2;
}
Class C extends B {
    int i=3;
    Void show(){
        SoP(i);  // 3
        SoP(super.i);  // 2
        SoP(super.super.i);  X
    }
}
```

```
Class B extends A{
    int i=2;
    int getSuperI(){
        return super.i;
    }
}
```

```
Class C extends B{
    int i=3;
    Void show(){
        SoP(i);  // 3
        SoP(super.i);  // 2
        SoP(getSuperI());  // 1
    }
}
```

```
Class A{
    int i = 1;
    int a = 5;
}

Class B extends A{
    int j = 1;
}
```

```
Class C extends B{
    int i = 3;
    Void show(){
        SoP(i); // 3
        SoP(j); // 1
        SoP(super.i); // 1
        SoP(a); // 5
    }
}
```

If B doesn't have i it will automatically call its super's i.

```
Class A{
    Private int i = 1;
    int j = 2;
    Protected int k = 3;
}
```

```
Class B extends A{
    Void show(){
        SoP(i); // error
        SoP(j);
        SoP(k);
```

Private ⇒ Can be accessed within same class.

No modifier / Default ⇒ Can be accessed within same class / same package.

Protected ⇒ Can be accessed within same class, same package & subclass (in same / any other package)

Public ⇒ Can be accessed from anywhere.

Method Overriding

⇒ Re-writing superclass method in subclass with same name & same signature.

⇒ is done either for extension or for restriction.

```
Class A{
    Void m1(int a, String b){
    }
    void m2(int a){
    }
    void m3(int a, double d){
    }
}
```

```
Class B extends A{
    void m1(int a, String b){
    } // Overiding
    Void m2(int a, int b){
    } // overloading.
}
```

⇒ Only default, Protected & Public methods can be overriden

Private methods cannot be overriden.

```
class A{
    Private void m1(){
    }
}
```

```
Class B extends A{
    void m1(){
    }
}
```

// Compile time error

⇒ In Overriding accessibility can be increased not decreased

```
Class A{
    Public void m1(){
    }
}

Class B extend A{
    Void m1(){
    }
}
```

⇓ error

Public
↓
Protected
↓
Default
↓
Private

```
Class B extends A{
    Void m1(){
    }
}

Class B extends A{
    Public Void m1(){
    }
}
```

✓

⇒ Return type must be same, from JDK 5, return type can be subclass also.

```
Class A{
}
Class B extends A{
}
```

```
Class X{
    A m1(){
        return new A();
    }
}
Class Y extends X{
    B m1(){
    }
}
```

⇒ final methods cannot be overriden:

```
Class A{
    final void m1(){
    }
}

Class B extends A{
    Void m1(){
    }
}
```

// error

Final

⇒ final class cannot be extended

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

✗ error

⇒ final method cannot be overridden.

⇒ final data members means constants & discuss later}

⇒ In JDK-17 we got concept "Sealed classes" {later}

⇒ Static methods cannot be overriden, rather these are hidden.

Abstract Classes

(1) ⇒ which cannot be instanciated. { can make reference but not the object }

Example ⇒

```
abstract class A{
}
```

A a1; ✓

A a1 = new A(); ✗

(2) ⇒ We can have abstract methods in a class and if there is even a single abstract method in a class that class must be abstract.

```
abstract Class A{
    abstract int A1();
    void m1(){
    }
    void m2(){
    }
}
```

```
abstract Class A{
    abstract int A();
    abstract int B();
}
```

Final abstract void m1(); ✗ not allowed

- A Class be abstract without any abstract method.

(3) ⇒ An abstract class can be extended, in subclass all the abstract methods of superclass must be initialised otherwise the subclass will also be abstract.

```
abstract Class A{
    abstract void m1();
    abstract void m2();
}
```

```
class B extends A{
    void m1(){
    }
}
```

In this case class B has to be abstract.

```
abstract Class B extends A{
    void m1(){
    }
}
```

```
class C extends B{
    void m2(){
    }
}
```