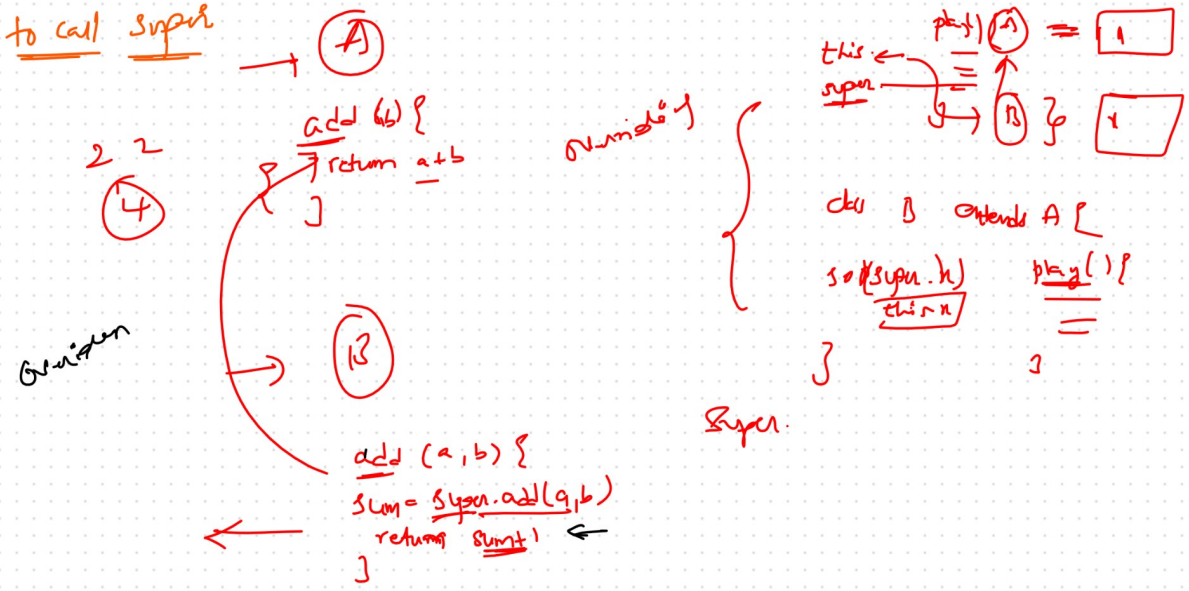
Reference to call super



Rules of Method Overriding

(*) While overriding a method we can't change the "signature" of the method [i.e name & parameters], incase if we change the signature

- return type
- method_name
- parameters

then it will be new specialized method

Parent `void play() {` ① `}`

`void add(int a, int b) {` ② `}`

child

`void play(String type) {` → new method `}`

overriding

to participate in overriding

* While overriding a method we cannot change the modifier

X

Parent `public void play() {` `}`

child `private void play() {` `}`

// Error

* while overriding the method we cannot change the return type of the method

→ We can change the modifier while overriding a method only if the accessibility to be increased in the method, then it is allowed/possible

public , private , protected

Note:

protected → public
private → public

public → private ✗

private
A {
...
}

protected
pkg

A ✗ → ✓ B
A ✓ → ✗ B

B extend A {
→ method c1s selfish
...
}
}

(*) While overriding the method we can change the return type, provided if the return-types are "Co_varient" → parent-child relation

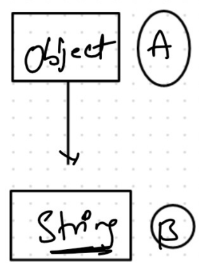
Parent
↑
Child

class → non-primitive data type

```
class Gamma {
    public Gamma createInstance() {
        return new Gamma()
    }
}

class Delta extends Gamma {
    public Delta createInstance() {
        return new Delta()
    }
}
```

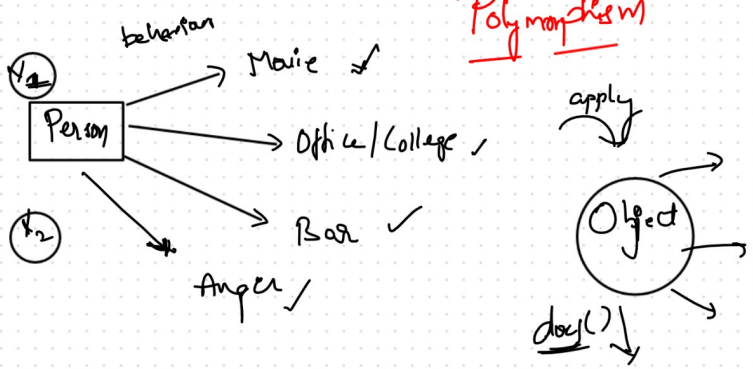
A obj = new A()
+ name
+ age
+ id
} keys



class String extends Object {
}

class ⟷
int ⟷ byte ✗

Polymorphism



Greek word — 1 : many

poly — many

morph — forms

def: the ability of a single action (or method) to behave differently depending on the object that it is acting upon.

Note:

(1) The example program is polymorphism (Os → Windows)

(2) We can achieve polymorphism through loose coupling

(3) We can achieve loose coupling by 1 : many relation

eg:

```
Os myOs = null;
```

→ Implicit

switching

```
myOs = new Window();
myOs = new Linux();
myOs = new Mac();
```

→ loose coupling

1 : many

Parent ← child object

```
Windows w = new Windows()
Linux   l = new Linux()
Mac     m = new Mac()
```

→ tight coupling

1 : 1

here we cannot change type

Type any type

Note:

→ The process of having parent type reference for the child type object is considered as "loose coupling"

convert Parent → child

→ The process of having Object & reference variable of same type then we consider it as "tight coupling"

Disadvantages of tight coupling

1. We cannot access the specialized methods of the child class
2. We can overcome this problem by down casting i.e converting parent type to child type

Type Casting

Data Type

Primitive Casting

→ Implicit : byte - short - int - long (widening)

→ Explicit : (byte)(n) (narrowing)

① Upcasting: The process of converting child type to parent type is called as upcasting [storing child type object in parent type reference]

this is an implicit conversion

eg:

```
Os myOs = new Windows();
```

Os → parent; myOs → parent reference type; Windows() → child

② Down Casting: The process of converting parent type to child type is called as Down Casting i.e [storing parent type object in the child type reference], this is to be done "explicitly" by developer

eg

```
Os myOs = new Windows();   // tight coupling
Windows myWindows = (Windows) myOs;
```

Windows → child; myOs → parent

myWindows, &