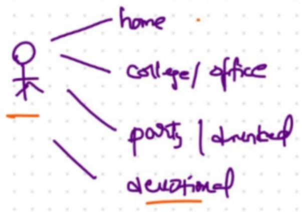
Polymorphism

– continuation of oop.

poly – many
morph – forms } Greek word $\xrightarrow{\text{Analogy}}$



(obj) = new Person ()

def: the ability of a single action or method to behave differently on the object that it is acting up on

2 types → ① Compile-time polymorphism (static) | method – overloading
② Run-time polymorphism (dynamic) | method – overriding

Notes :

(1) The prog discussed (day26 – PolyMorphProg.Java) is polymorphic
(2) We can achieve polymorphism through loose coupling
(3) We can achieve loose coupling by 1 : many relations

```
Os myOs = null;
myOs = new Windows ()
myOs = new Linux( )
myOs = new Mac ( )
```

loose Coupling
1 : many

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

tight Coupling
1 : 1

Notes :

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

→ The process of having Object & reference variable of same type is considered as tight coupling

The disadvantage of loose coupling

1, we cannot access the specialized methods of the Child Class

2, we can overcome this problem by down-casting object

i.e converting parent-type to child type

Type Casting Implicit

1, **UpCasting** : The process of converting Childtype to parent type is called as Up Casting

i.e [storing child type object in parent type reference]

$$\underbrace{\text{Os}}_{\text{parent}}\ \underset{\substack{\uparrow \\ \text{parent reference type}}}{\text{myOs}} = \text{new}\ \underbrace{\text{Windows}}_{\text{child}}();$$

2, **Down Casting** : The process of converting parent type to child type is called as Down Casting

i.e [storing parent type object in child type reference]

→ This process is to be done explicitly by the developer

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

$$\underbrace{\text{Windows}}_{\text{child}}\ \text{windows} = ((\text{windows})\ \underbrace{\text{myOs}}_{\text{parent}});$$