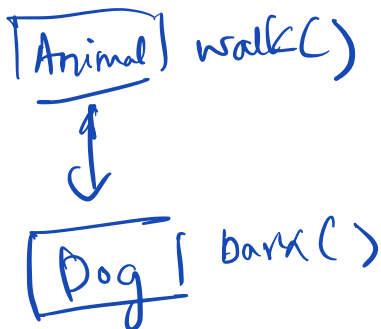


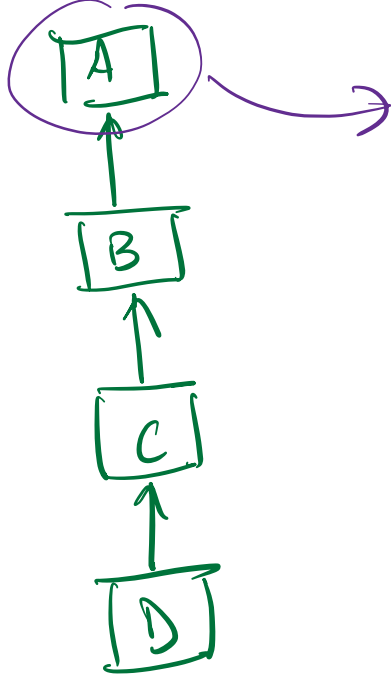
# Agenda

1. Inheritance - (Constructor Chaining)
2. Polymorphism
  - Compile Time Polymorphism (Method Overloading)
  - Runtime polymorphism (Method Overriding)
3. Interfaces
4. static (Basic) , Destructors

## Inheritance :

- child classes inherit attributes and behaviours of their parent class , also they may add their own or not.





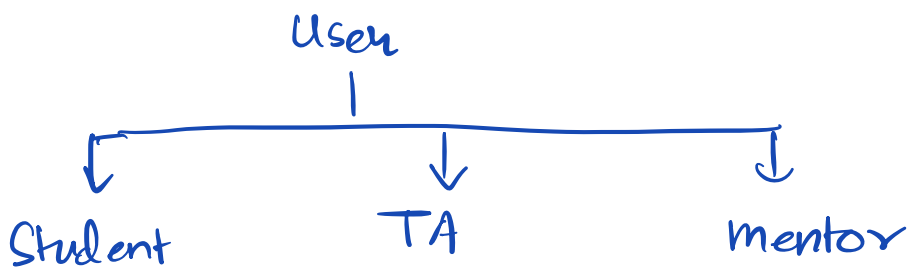
Which one has the highest abstraction over here?

A

Note: No private members are inherited in child class. (Although since attributes are created in memory it gives a feeling of inheritance with no access)

Polymorphism = Poly + morphism  
                                  ↓                                  ↓  
                                  many                                  forms

What have we seen so far with many forms?



Student	<u>IS A</u>	user
Mentor	IS A	user
TA	<u>IS A</u>	user

printUserNames( ? user )

→ type of this?

Student s = new Student();

Mentor m = new Mentor();

printUserNames(s);

printUserNames(m);

Animal walk()

↑ inheriting

Dog Bark()

Dog d = new Dog();

✓ Animal a = new Dog(); ①

✓ a.walk(); ②

Parent class reference ~~X~~ a.bark(); ③

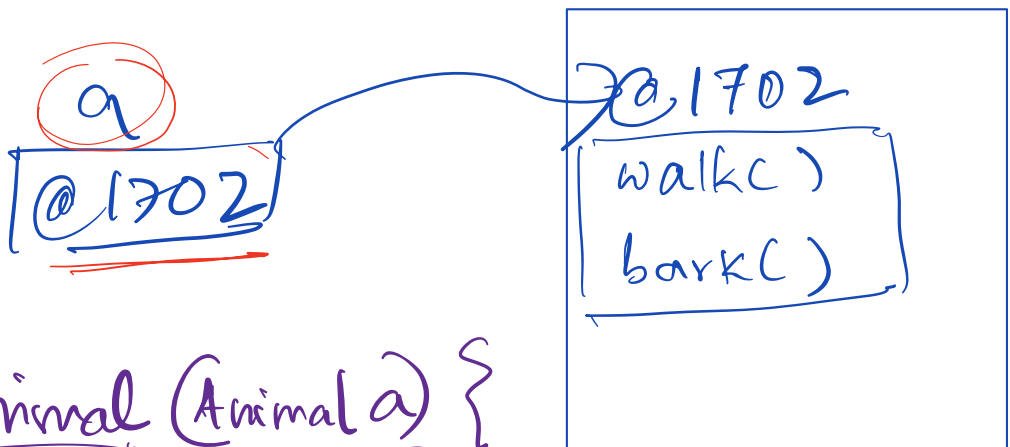
↳ compile time error

variable can contain

child class object, but not vice versa i.e.

object creation happens  
at runtime.

~~X~~ Dog d = new Animal();



specific To Animal (Animal a) {

a.bark(); X

} a.meow(); X

specific... (new Dog)  
new

Break Till 8:10

# Method Overloading

```
public Student (int A)  
public Student (String A, int A)
```

Many methods have different forms.

```
void print (int a) ; ①  
void print (int a, String a) ; ②  
void print (String a, int a) ; ③
```

compiler is able to identify  
from function call.

```
print (10) ; ①  
print (10, "akash") ; ②  
print ("akash", 10) ; ③
```

compile time polymorphism

print (int a, double d); ✓  
print (double d, int a); ✓  
print (10, 10.0);  
print (10.0, 10)

print (String a);  
print (String s);

print ("Akash");  
print ("Akash");

void  
int print (int a);  
print (int a);

What matters in overloading is:

return method Name (Parameter type & order)

type  
doesn't  
matter.

void  
int print (String a);  
print (int a);

## Method Overriding :

```
class Bird {
```

```
void makeSound () {
```

```
    --  
    --
```

```
}  
void makeSound () X
```

```
class Peacock extends Bird {
```

```
void makeSound () // inherited
```

```
void makeSound () {
```

```
    // overriding
```

```
}
```


```
void makeSound (int a) {
```

```
    // overloading
```

```
String makeSound () X
```

```
String makeSound (String s) {
```

```
    // Overloading
```



In Overriding whole method signature  
return type + method name + parameters  
have to be exactly same.



# Static

```
class Student {  
    int age;  
    static int count;  
}
```

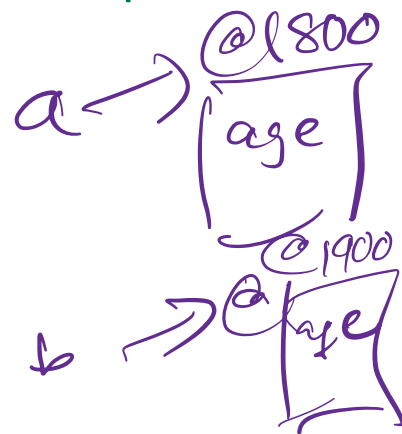
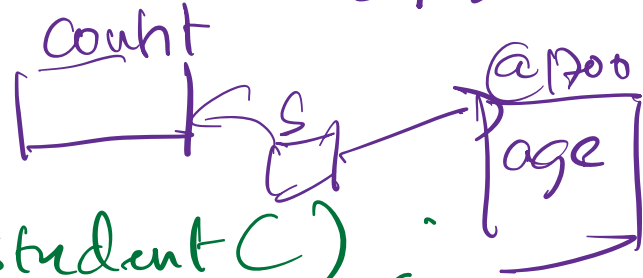
```
Student s = new Student();  
s.age;
```

class Name  
Student.count;

s.count  
a.count  
b.count

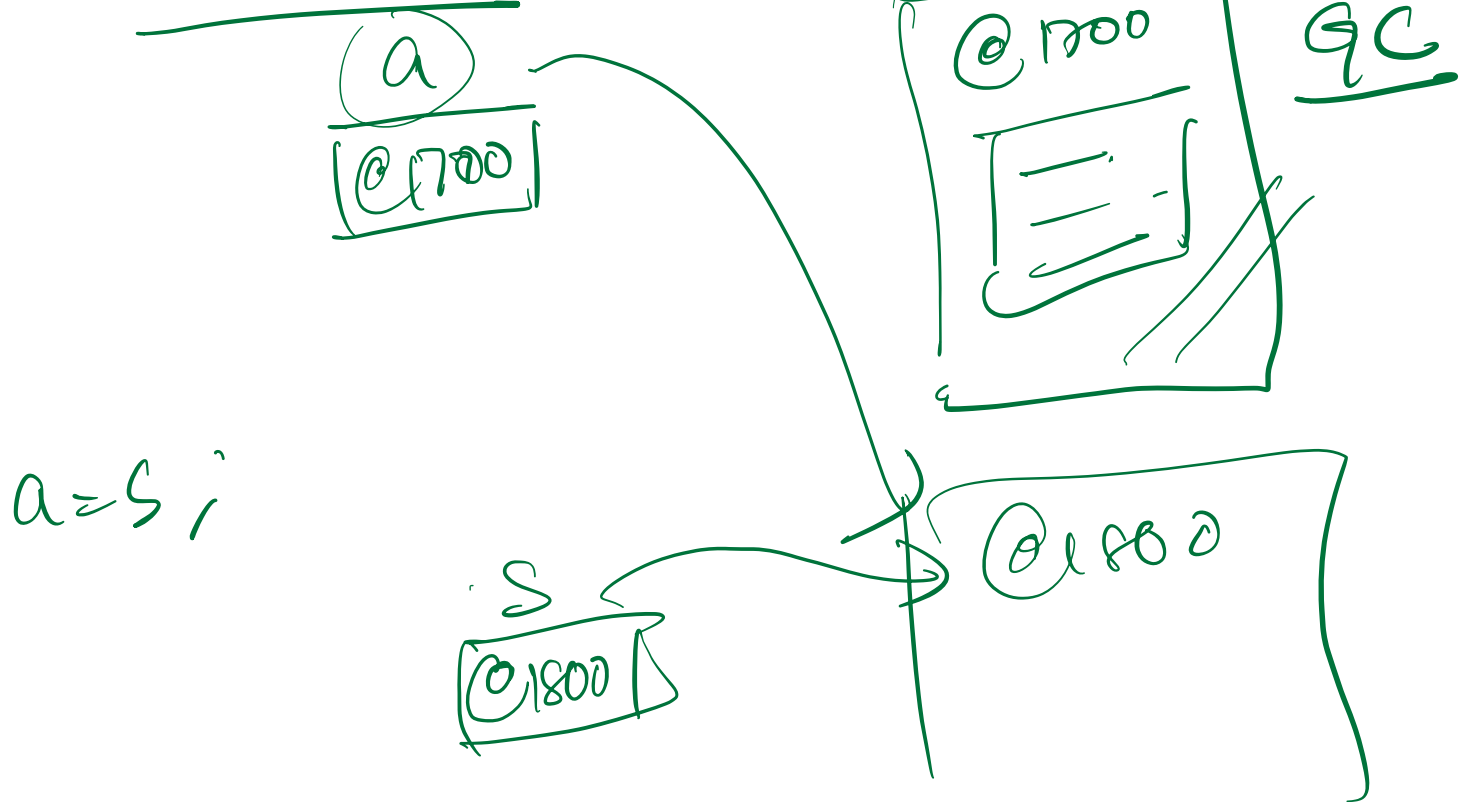
} will access  
same address/var

```
Student s = new  
Student a = "  
Student b = 14
```

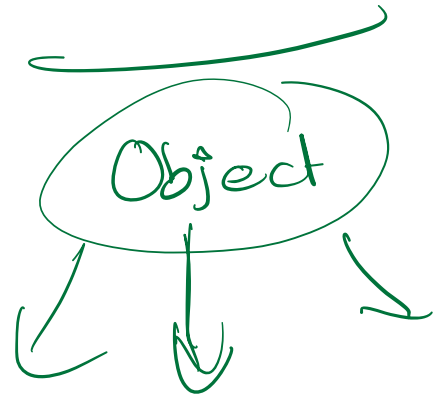


- Overloading does work for static methods.

# Destructor - JVM Garbage Collector



Object class ?  
↳ finalize();



Student s = new Student();

s.~~page~~ = 10 ;

s.finalize()