OOPS 2: Constructor, Inheritance & Polymorphism

Constructors

Deep copy & Shallow copy

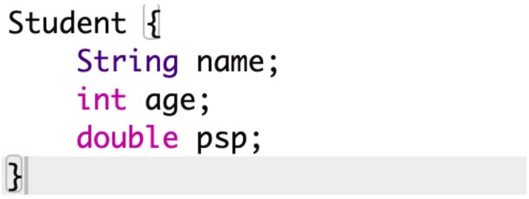
Inheritance

Polymorphism

Method Overloading & Overriding

Class → Blueprint of an entity

Object → Instance of class / Real Entity



```
Student {
    String name;
    int age;
    double psp;
}
```

constructor

Student st = new Student();

int a = 12

datatype

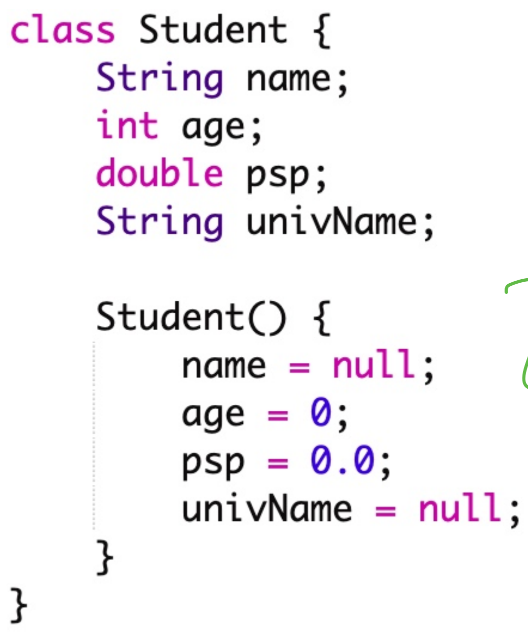
variable name

Default Constructor

Default value eg ⇒ int = 0

string = null / ""

double = 0.0 etc



```
class Student {
    String name;
    int age;
    double psp;
    String univName;

    Student() {
        name = null;
        age = 0;
        psp = 0.0;
        univName = null;
    }
}
```

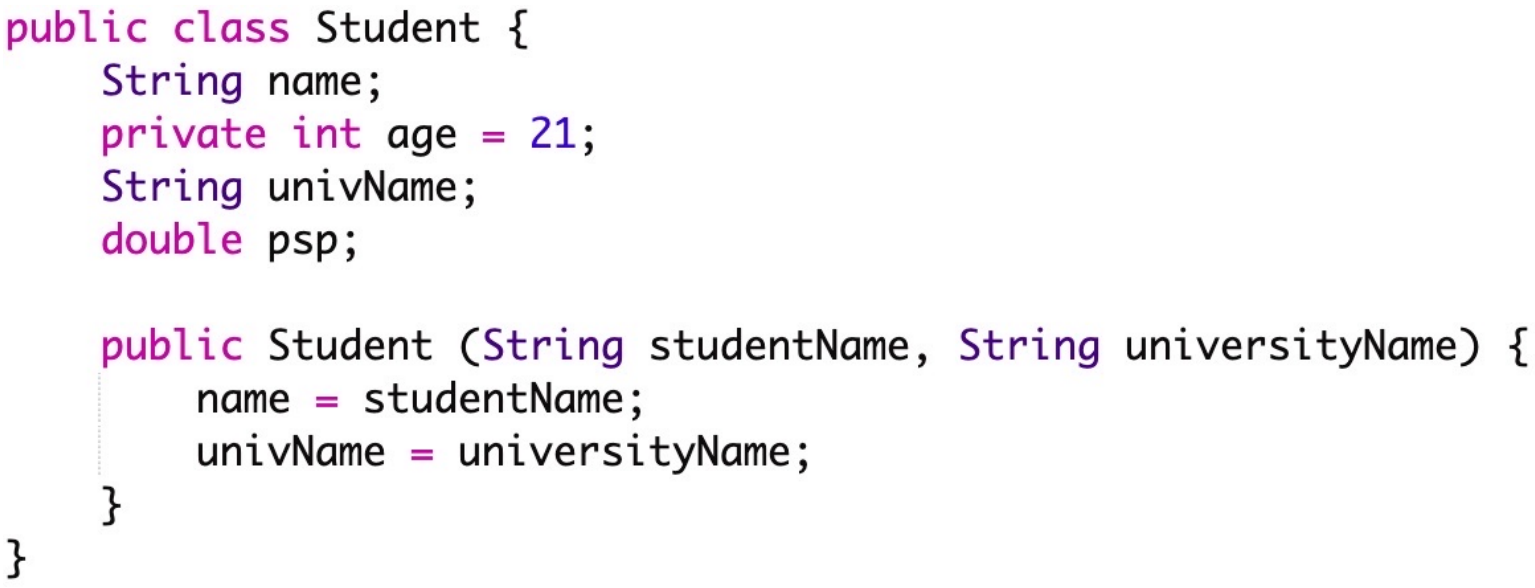
→ not written by us

Is there any case when default constructor is not created?

Yes, when we create our own constructor

Default Constructor →

1. No parameters
2. Name is same as class name
3. Datatype returned is class name



```
public class Student {
    String name;
    private int age = 21;
    String univName;
    double psp;

    public Student (String studentName, String universityName) {
        name = studentName;
        univName = universityName;
    }
}
```

```
Student st = new Student();
```

// Error → there is no default constructor now

∵ we have created our own constructor

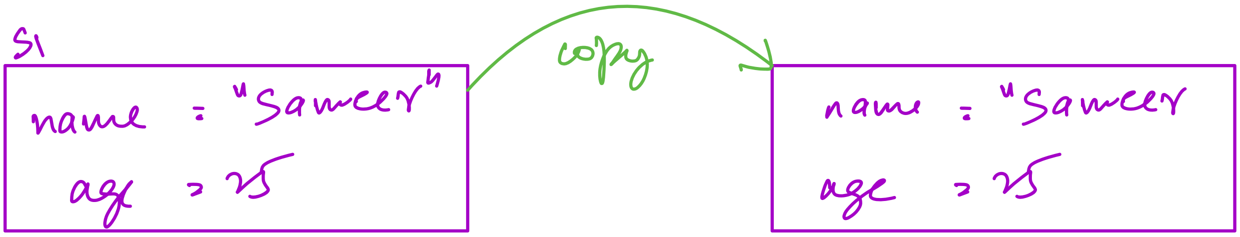
```
Student st = new Student("Nipun", "IITB"); ✓
```

Manual Constructor ⇒

for data members not passed in the constructor will get the default value.

```
int age
int id

Student( int a ){
    id = age * 5;      = 0×5=0
    age = a;           = a
}
```

→ default value is used here i.e, 0

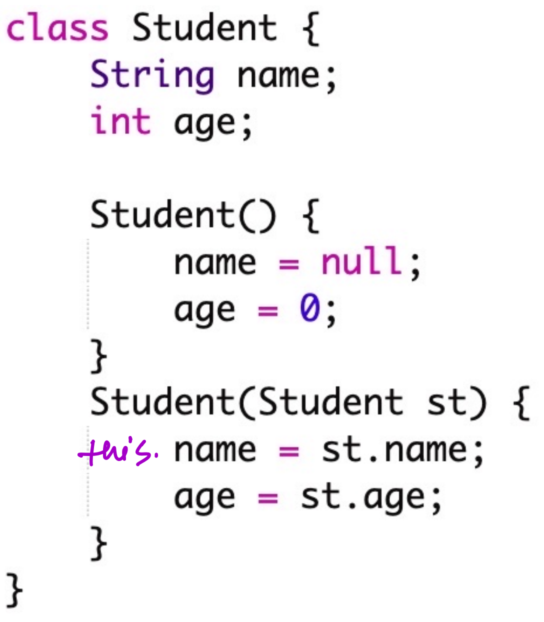


S1

name = "Sameer"
age = 25

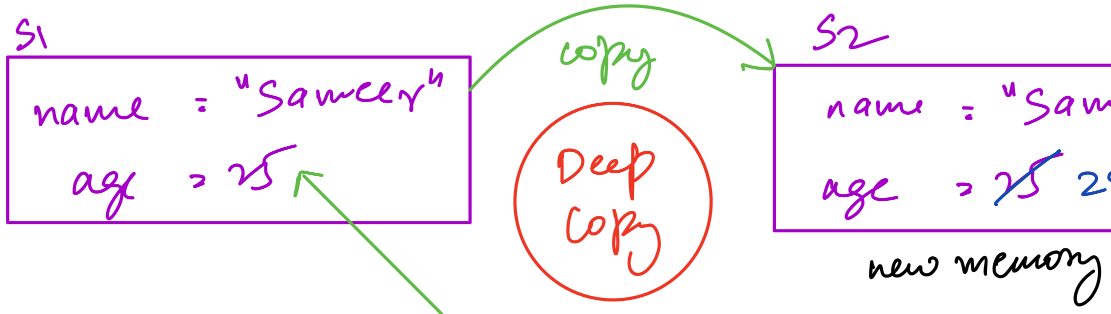
copy

name = "Sameer
age = 25



```
class Student {
    String name;
    int age;

    Student() {
        name = null;
        age = 0;
    }
    Student(Student st) {
  this. name = st.name;
        age = st.age;
    }
}
```

Student s2 = new Student(s1);

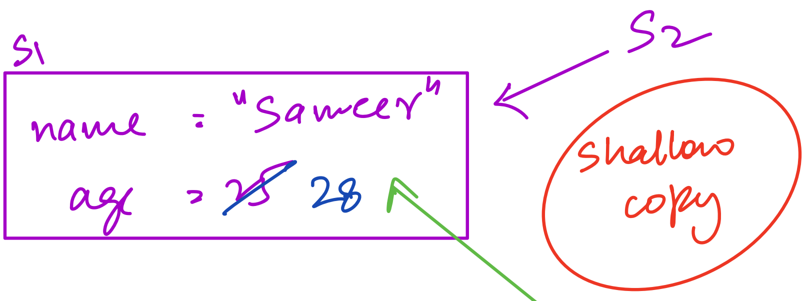


s2.age = 28

print(s1.age) // 25

Any update in copy will not reflect in original object & vice versa. Both are independent.

Student s2 = s1;



object reference point to the same memory

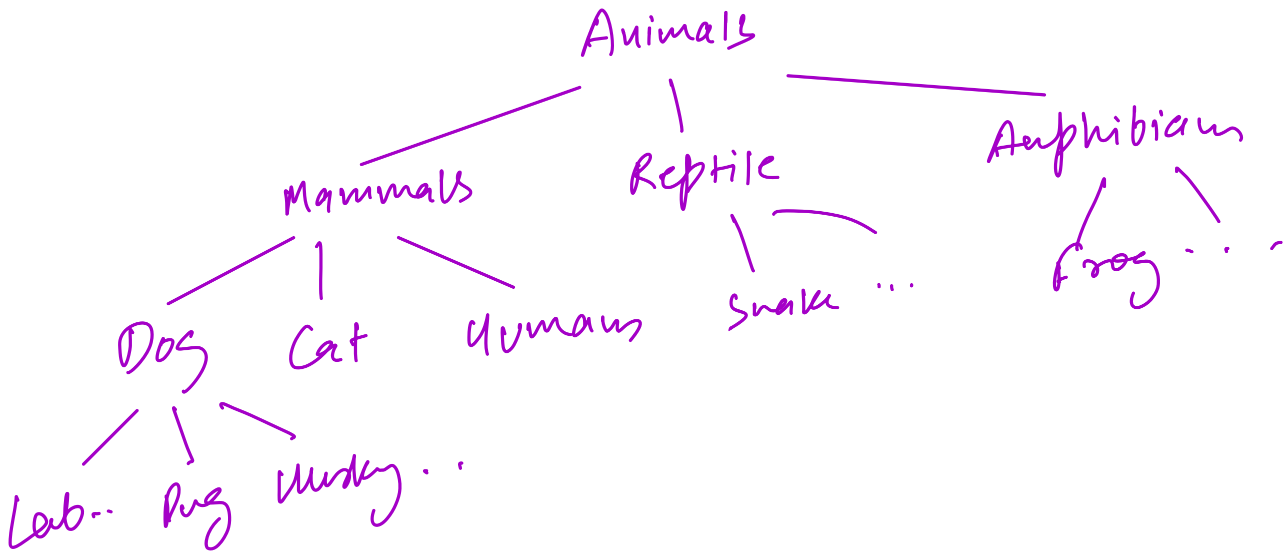
s2. age = 28

print (s1. age) // 28

Google Docs is a shallow copy example

Any update in copy will reflect in original object & vice versa. Both are not independent.

Inheritance



Animals can move ⇒ Mammals can also move

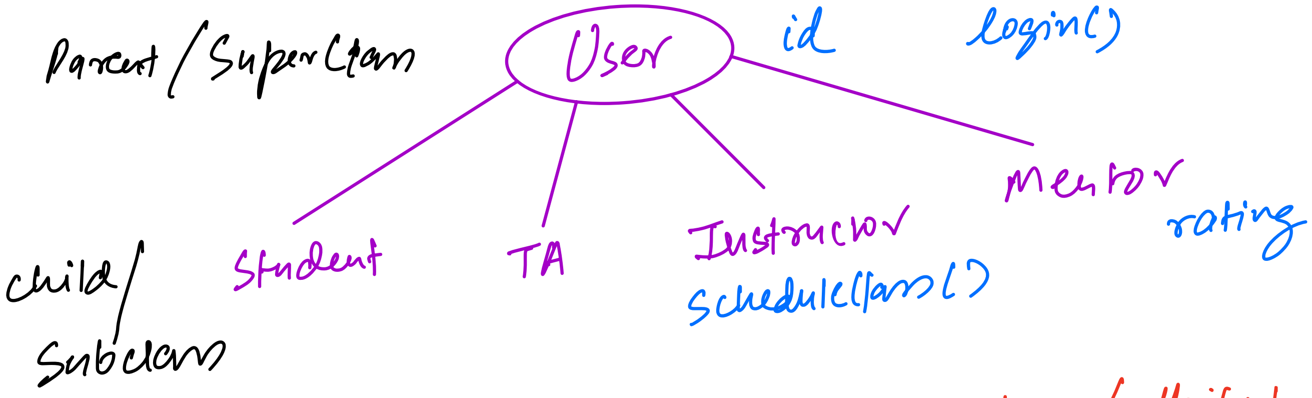
Dog can bark ⇒ Husky can bark

⇒ Snake can bark ✗

Representation of this heirarchy is known as inheritance.

↓

Parent-child relationship b/w different classes



→ They will share all members/attributes/methods etc of User class + may have some more of their own

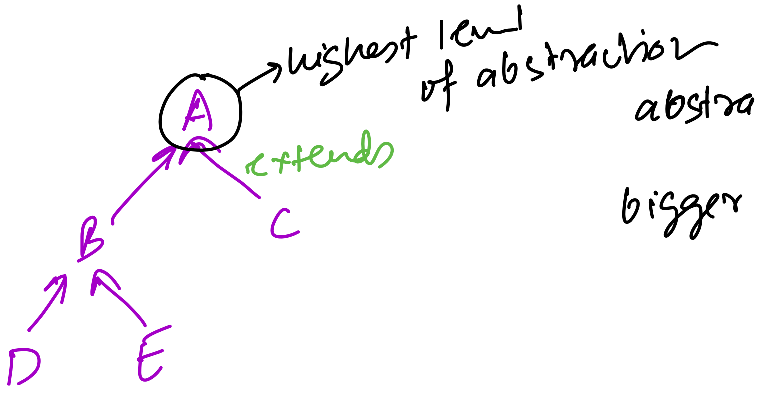
```
class User {
   string name;
   void login() {
      ...
   }
}
```

Java →
```
class Instructor extends User {
      :
   void sheduleClass() {...}
}
```

Python →
```
class Instructor ( User ):
      :
```

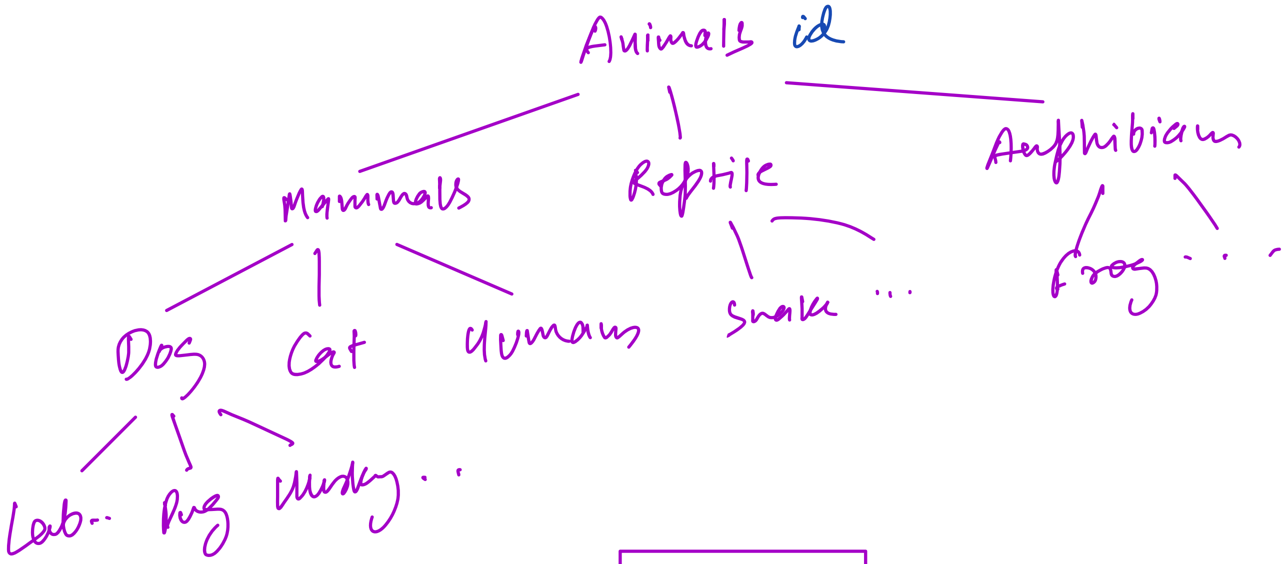
C++ →
```
class Instructor : public User {
   :
}
```

```
Instructor i = new Instructor()

i.login();
```



abstraction is idea

bigger idea → more general idea

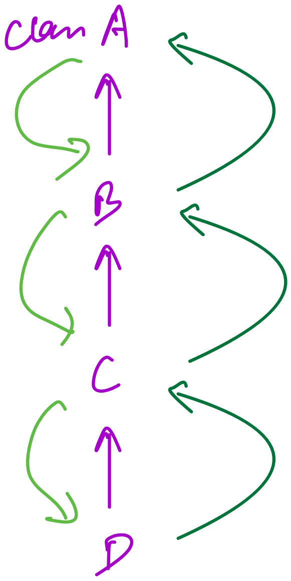


```
Husky h = new Husky();
```

constructor of Animal class can initialize id.

Can a child be born without parent → NO

⇒ first parent should be constructed then child class.

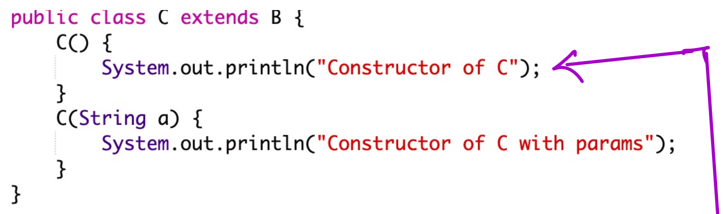


D d = new D(); // create obj. of class D

steps →

1. Constructor of D is called
2. Befor its execution, constructor of C is called.
3. Before execution of C class constructor, constructor of B is called.
4. Similarly, A constructor is called

A class constructor will be executed first then B is completed followed by C & then D.



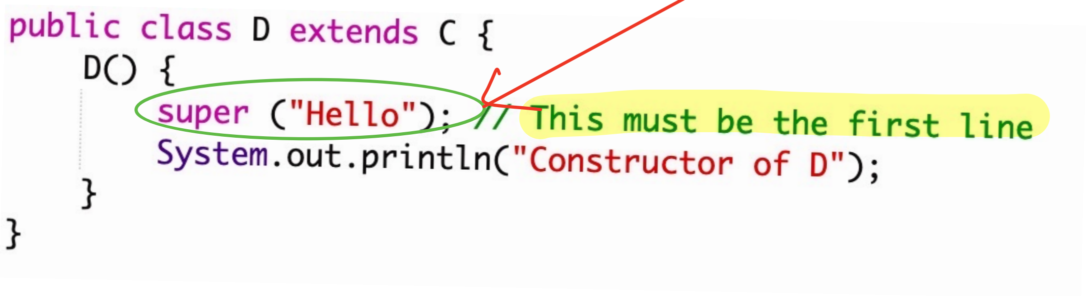
```
public class C extends B {
    C() {
        System.out.println("Constructor of C");
    }
    C(String a) {
        System.out.println("Constructor of C with params");
    }
}
```

D d = new D();

How to call manual constructor?

using "super" key word

super() refers to constructor of parent class.

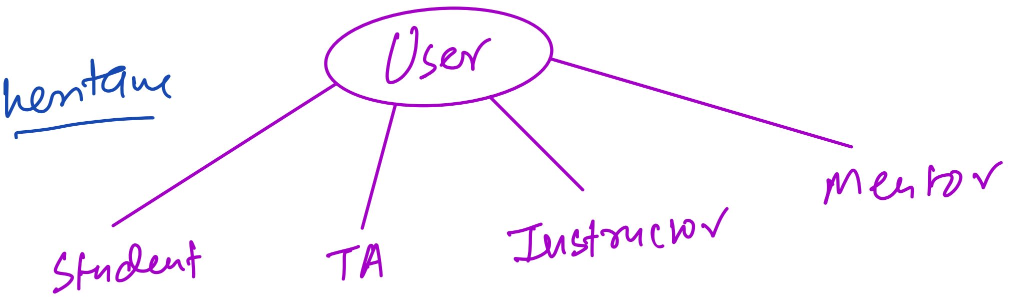


```
public class D extends C {
    D() {
        super ("Hello"); // This must be the first line
        System.out.println("Constructor of D");
    }
}
```

Polymorphism

many   forms

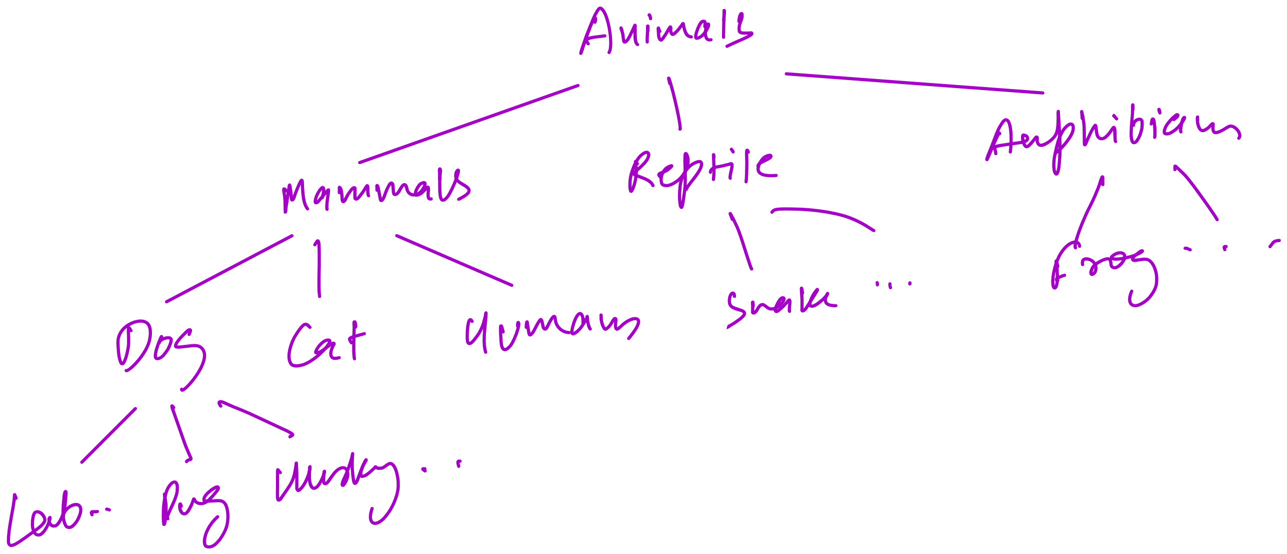
eg → Inheritance



User has many forms

User can be Student

can be TA ....



```
Animal a = new Dog(); ✓
```

all dogs are animals

```
Dog d = new Animal(); ✗
```

all animals are not dogs

We can put object of child class in reference of parent but vice versa is not true.

List of Animals → object of animal,
dog,
mammal,
human ...

```
class A {
    int age;
    string name;
}
```

```
class B extends A {
    string univName;
}

class C extends A {
    double psp
}
```

```
A a = new C(); ✓

a.psp = 50; ✗
```

'a' has a datatype of 'A' & class A doesn't have psp compile time error : compiler only allows to access members of datatype of that variable.

```
A  a = getA();
 a.age = 20; ✓
a.psp=50; ✗
A  getA(){
    int x = random() % 2;
    if (x==0)   return new B();
     else return  new C();

}
```

Method Overloading

```
void hello(){
    print ("hello");

}
void hello( String s){
     print(s);

}
        hello();
        hello("xyz");
```

many forms
→ polymorphism

⇓

The final form of execution is known to compiler. So, its called compile time polymorphism

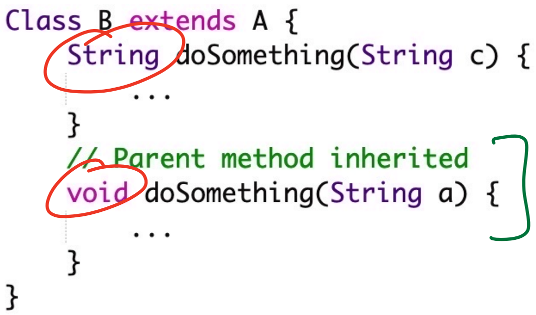
1. void hello()
   void hello(String s) ✓

2. void hello (String s)
   String hello (String s) } This is not method overloading

Method Signature → method name (datatype of parameter)

method overloading → diff. method signature

Method Overriding



```
Class B extends A {
    String doSomething(String c) {
        ...
    }
    // Parent method inherited
    void doSomething(String a) {
        ...
    }
}
```

→ method of class A

Not valid ∵ method signature is same

child class method with different signature → method overloading

If parent & child have exactly same method =>

method overriding

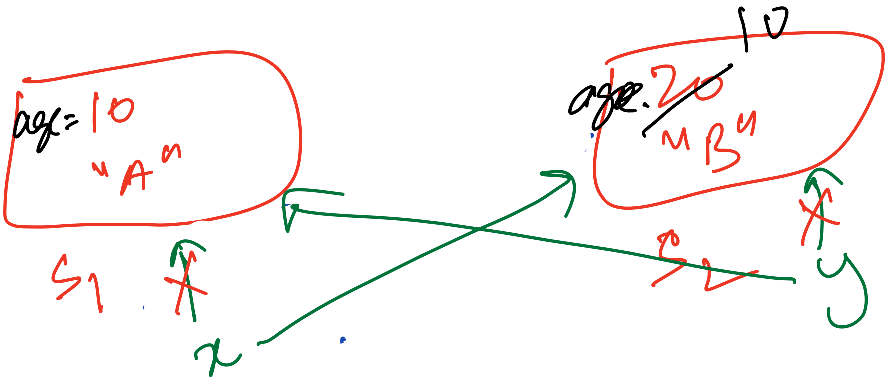
```
class A {
    void doSomething() {
        print("A")
    }
}
class B extends A {
    void doSomething() {
        print("B")
    }
}
```

```
A a = new A();
a.doSomething()  →  "A"

B b = new B();
b.doSomething()  →  "B"

A a = new B();
a.doSomething()  →  "B"
```

method execution → actual object created

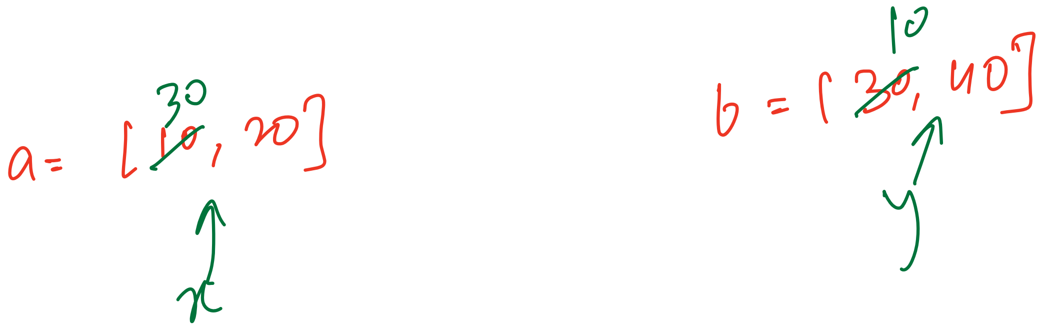
Runtime polymorphism

Doubt



```
swap(St x, St y) {
    temp = x.age
    y.age = x.age
    x = temp.age
}
```

```
S1.display();
x.display();
```

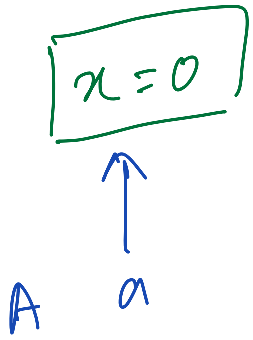


a[0] = 30

```
(x, y) {
    swap(x[0], y[0])
}
```

```
class A {
    int x
}
```

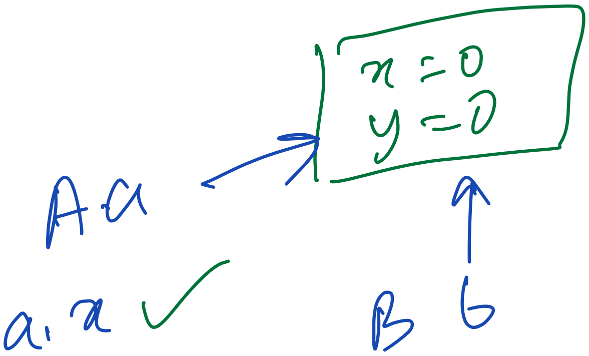
new A();



a.x = ✓

```
class B extends A {
    int y
}
```

new B();



a.x ✓

a.y ✗

b.x ✓

b.y