OOPS 2

Hello Everyone

Today's Content:

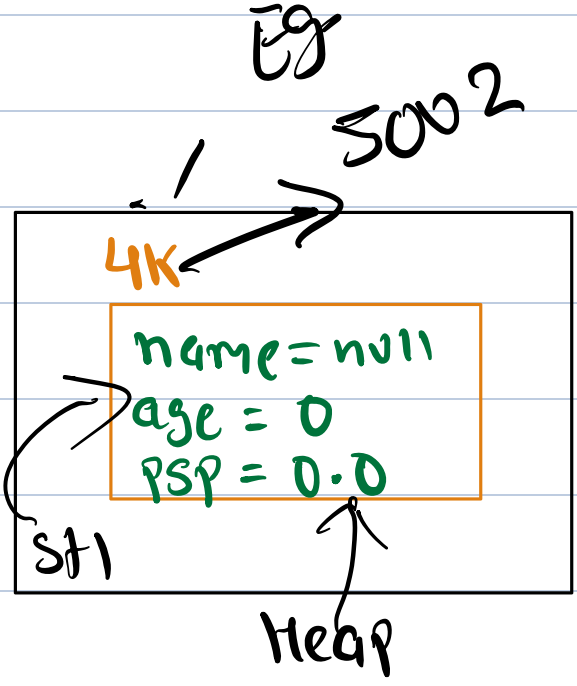
1) Constructor
2) Types of Constructor
3) Deep copy and shallow copy
4) Inheritance
5) Polymorphism
6) Method overloading and method overriding

Class: Blueprint of an entity.

Object: Actual entity / instance of a class.

Constructor :

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




```
Student St1 = new Student();
int a = 20;
```

St1 = 4K

Default Constructor:

If you don't have any constructor in your class then default constructor is created.

```
Class Student
{
    String name;
    int age;
    double PSP;

    Student(){
    }
}
```

=> 0 for integer
=> null for String
=> 0.0 for double

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

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

Properties:

1) Constructor are used to initialise attribute.

2) Name should be exactly similar to class name.

3) No return type not even void.

4) By default, it will return the address of the newly created object.

5) Set every attribute of a class to it's default value.

6) Created only if we don't write our own constructor.

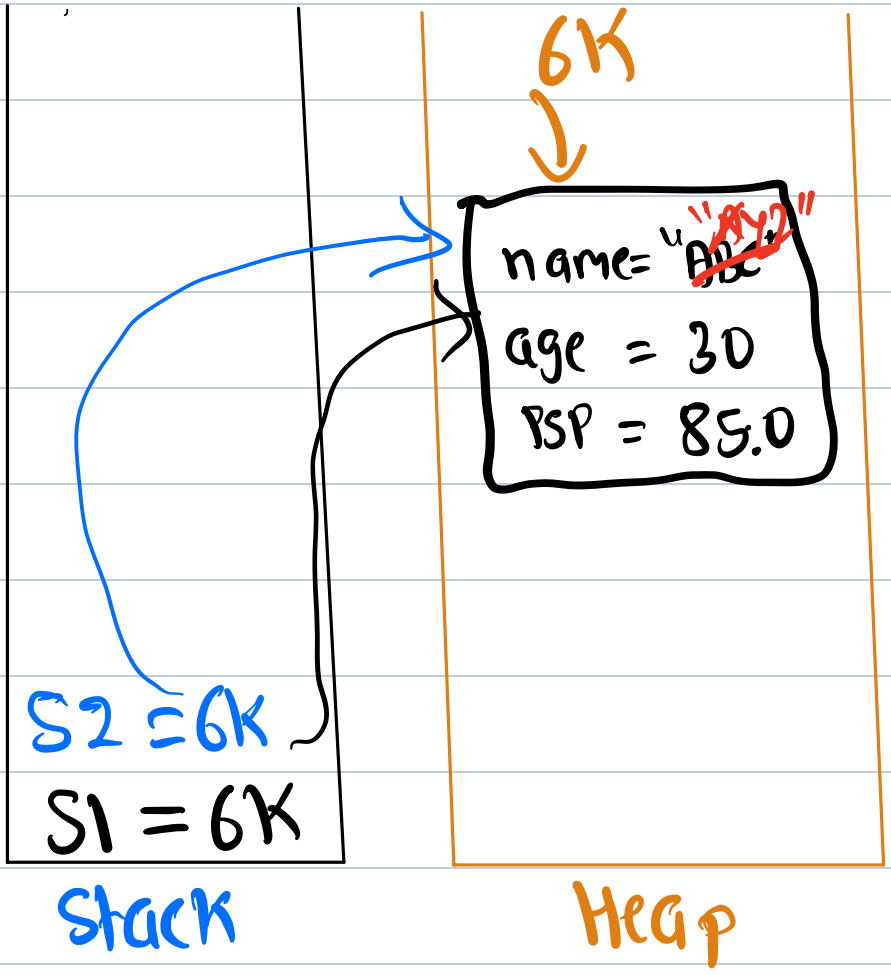
Parameterised Constructor

→ To pass value of our choice while creating object.

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

    Student(String str, int a, double p)
    {
        this.name = str
        this.age = a
        this.psp = p
    }
}
```

✗ Student s = new student()

```
Student S1 =
new Student("ABC",
    30, 85.0);

Student S2 = S1

S2.name = "XYZ";

print(S2.name)  // XYZ
print(S1.name)  // XYZ
```



Student S1 = new Student("ABC", 30, 85.0);

Student S2 = new Student("XYZ", 30, 85.0)

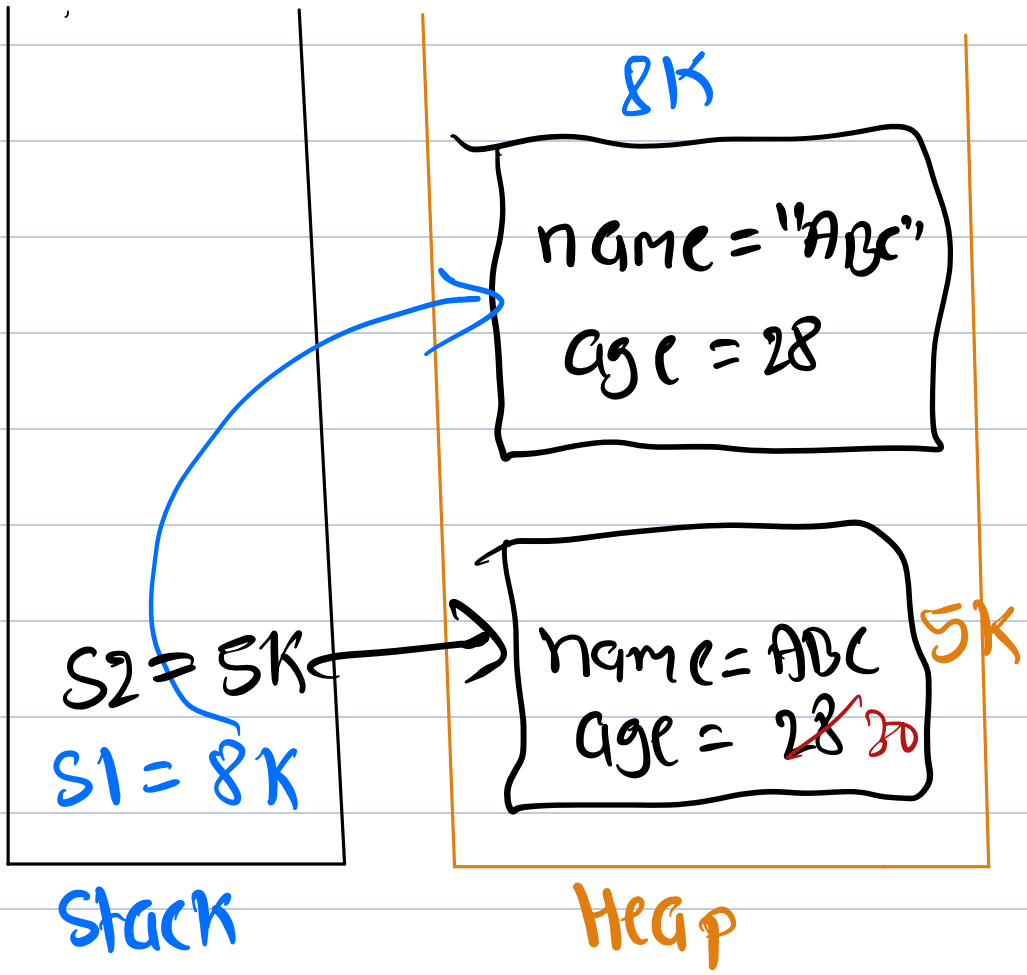
Copy Constructor: Create Deep Copy.

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

    Student(String stuName, int stuAge)
    {
        this.name = stuName;
        this.age = stuAge;
    }

    Student(Student s)
    {
        this.name = s.name;
        this.age = s.age;
    }
}
```

Copy Constructor

Student S1 =
new Student (
"ABC", 28)

Student S2 =
new Student (S1);



S2.age = 30

Print (S1.age) → 28

Shallow copy :

→ 1 instance, multiple variables referring to that instance.

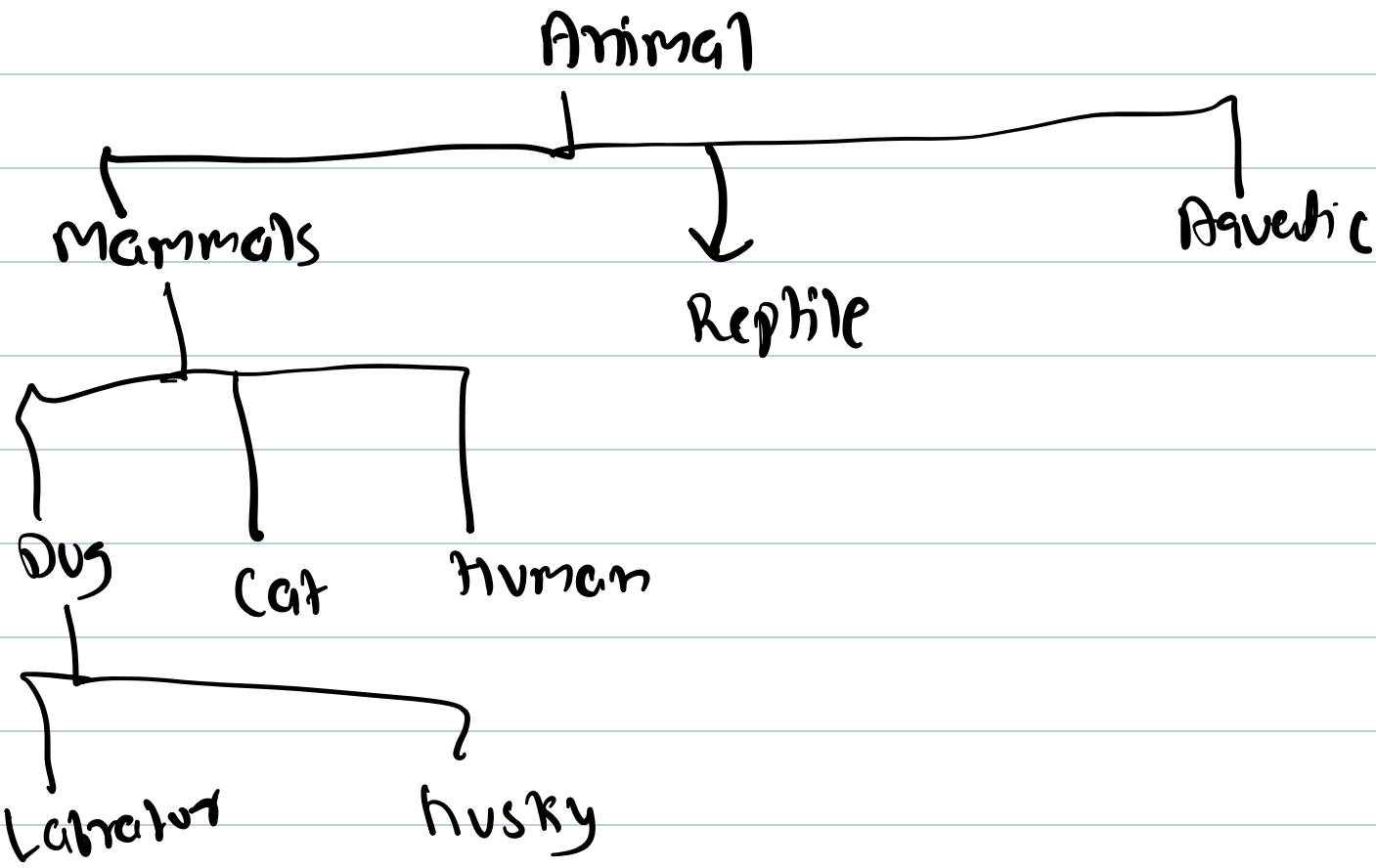
→ Whenever you create new object, new object still refers to the attribute of older object.

```
Student S2 = S1;
```

Deep copy :

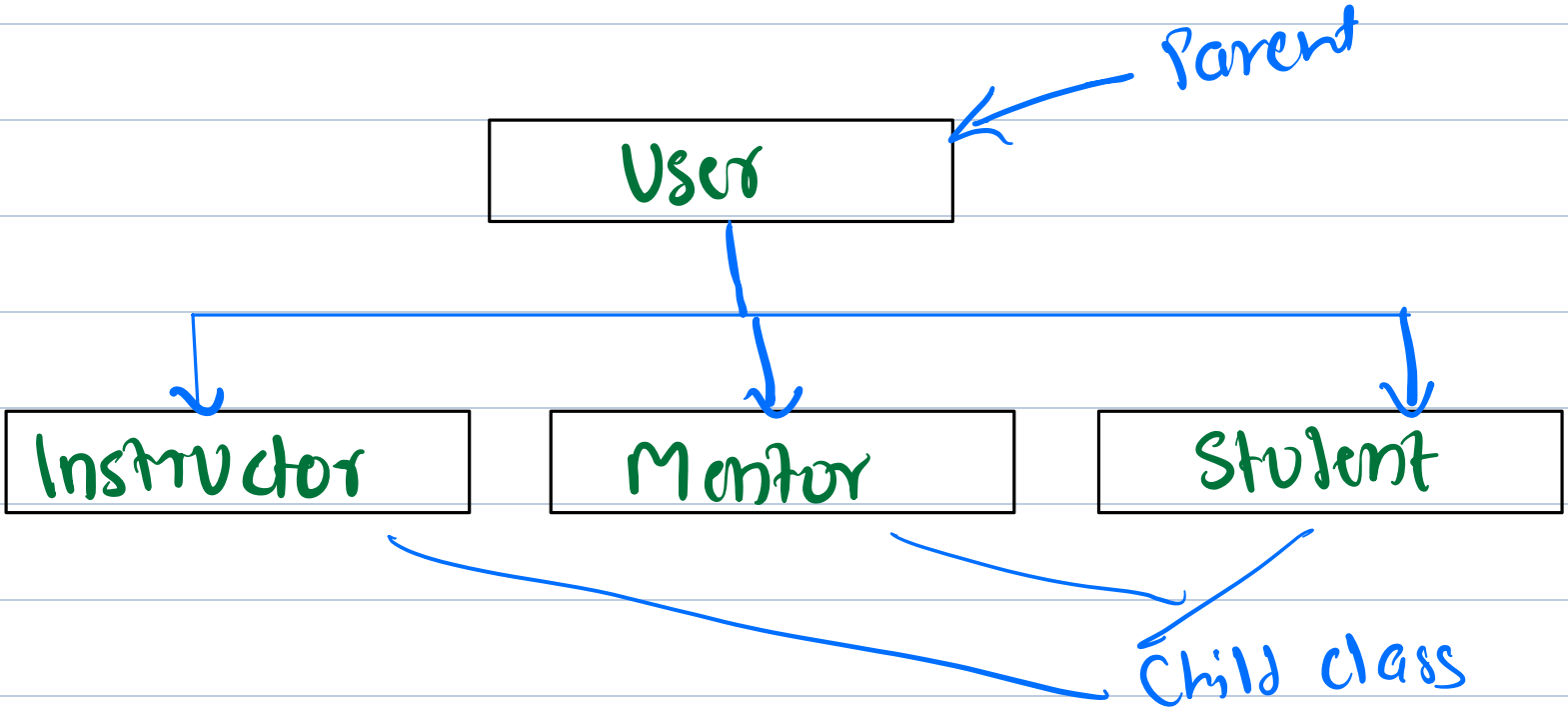
Whenever you create a new object behind the scene new object is not referring to the attribut of older object.

Inheritance: Representation of hierarchies



Representing Inheritence

Represented by parent child relationship between classes.

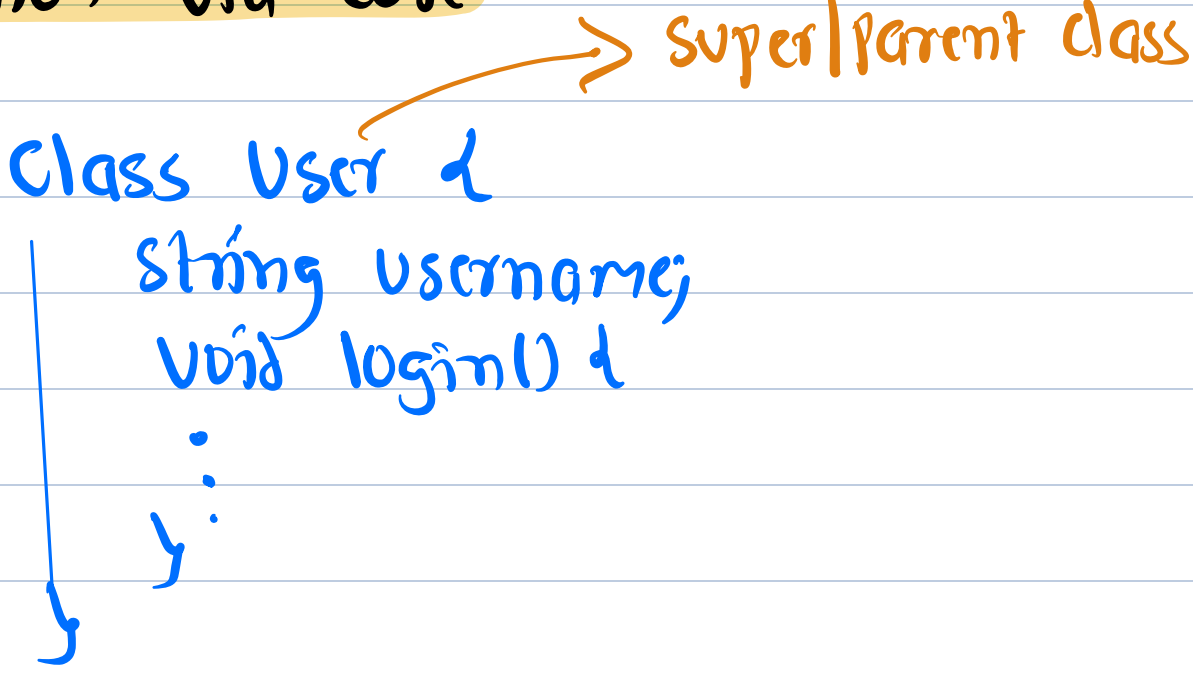



```
Class User
{
    String username;
    boolean login;
}
```

→ Child class | sub class can have specific attribute or behaviours which are not present in the parent | super class.

→ All attributes of parent class will be present in the child class.

Representation via code



Learner → Child | sub class

Python: Class Learner(User)

Java : class Learner extends User

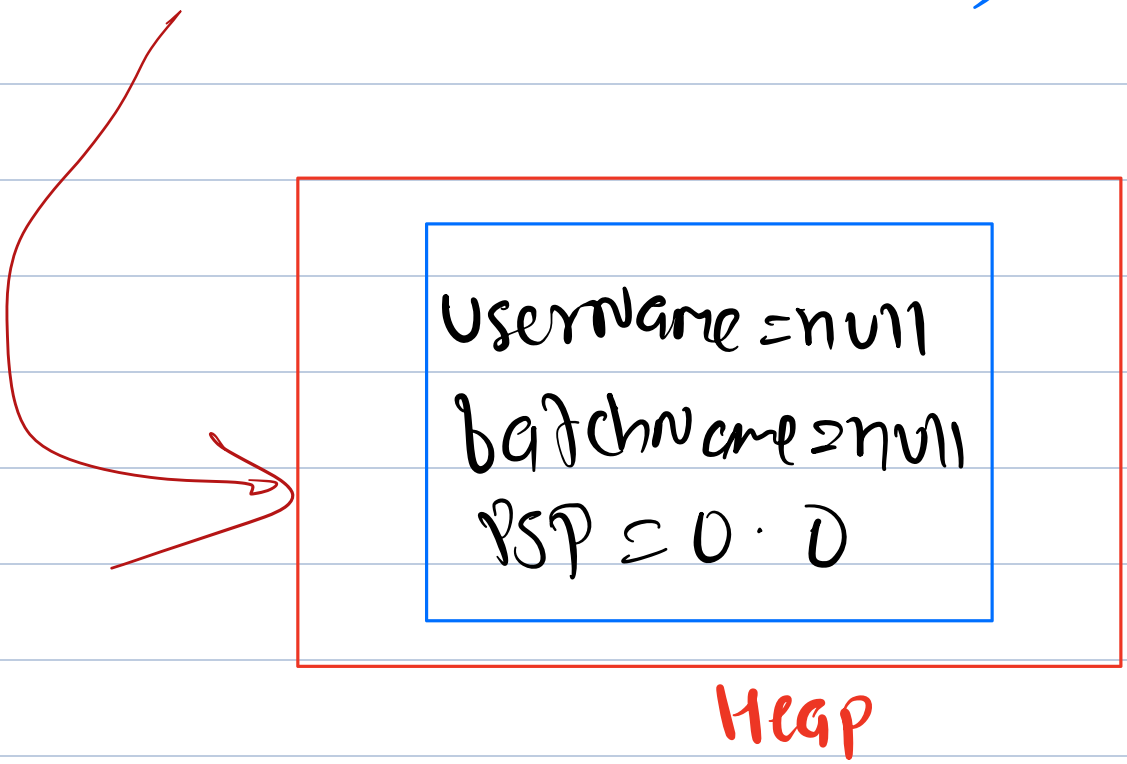
C++: class Learner : public User

C#: class Learner : User

```
class Learner extends User {

    String batchName
    double psp
    void attendClass() {

    }
}
```

Generating Object of Child class

```
Learner learner = new Learner();
```



Q. How is learner class initialising attribute of the parent class?

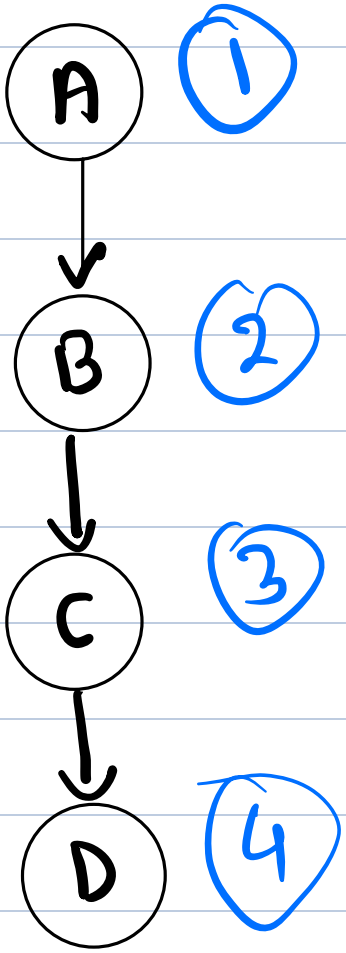
User → ① default constructor of User got called
username=null

Learner → ② initialise PSP and batchname

Constructor Chaining:

```
D d = new D();
```

A -> B -> C -> D



Conclusion: Whenever a constructor of a child class is called before its execution the default constructor of the parent class is executed.

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

Q: Creating an object of D will call the default constructor of C. How can I call the parameterised constructor of C.

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

Solution: Using super Keyword.

Break : 10.14

Polymorphism: Poly means many
morphism means form.

✓ `Animal a = new Animal()`

✓ `Dog d = new Dog()`

✓ `Animal a1 = new Dog()`

✗ `Dog d1 = new Animal()`

`User U = new User()` ✓

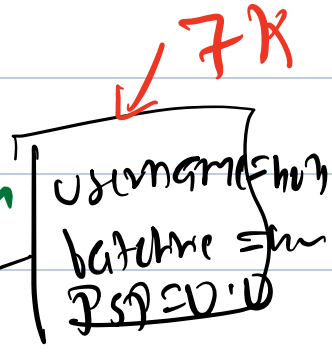
`User U1 = new Learner()` ✓

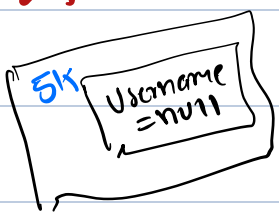
U1 = 7K

`Learner L1 = new User()` ✗

L1 = 5K

L1.PSP





```
A {
    int age
    String name
}

B extends A {
    String univ
}

C extends B {
    String company
}
```

```
A a = new A()

A a = new C()
```

age=0
name=null
univ=null
company=null

a.company = ?

age name

```
C c1 = new C()
```

age, name, univ, company.

Compile time polymorphism

Method overloading:

```
Class A
{
    void hello()
    {
        print("hello world");
    }

    void hello(string name)
    {
        print("hello" + name);
    }
}
```

```
A a = new A()
a.hello() → hello world.
a.hello("Alex") → hello Alex.
```

```
Class A
{
    void hello(String name)
    {
        :
    }

    String hello(String name)
    {
        :
    }
}
```

Compile time error

```
A a = new A()
a.hello("xyz");
```

Run time polymorphism

```
class A
{
    void doSomething(String a)
    {
        =
    }
}
```

(1)

```
class B extends A
{
    String doSomething(String c)
    {
        =
    }
}
```

(2)

```
B b = new B();
b.doSomething("xyz");
```

Compile time error.

```
class A
{
    void doSomething(String a)
    {
        ==
    }
}
```

(1)

```
class B extends A
{
    void doSomething(String c)
    {
        ==
    }
}
```

(2)

```
B b = new B();
b.doSomething("x42");
```

→ 2

```
A a = new A()
a.doSomething("x42");
```

→ 1

~~B b = new A();~~
~~b.doSomething("abc");~~ →

A a = new B();
a.doSomething("ab"); → (2)

If parent and child class have same method name, same return type, same signature. it is called as method overriding.

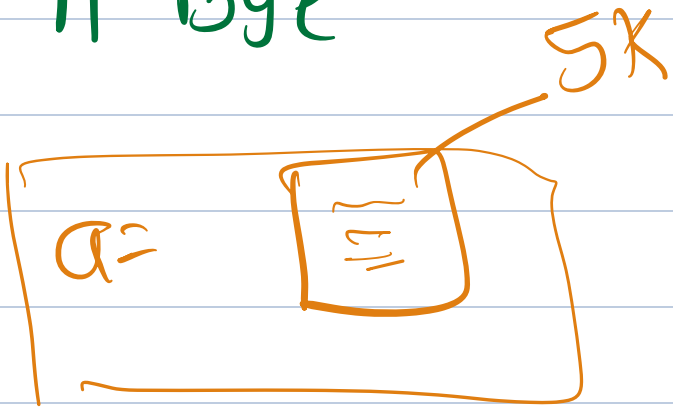
```
Class A
{
void dos()
{
 print("hello");
 }
}
```

```
class B extends A
{
 void dos()
 {
  print("Bye");
 }
}
```

Client class → main

```
A a = new A();
a.dos(); // hello
```

```
A a = new B();
a.dos(); // Bye
```



Doubt

void method()

{

}

method overloading

int method(int a)

{

}

Employee

FT Emplo

PT Emplo

Employee a = getEmployee(empId)

→ Employee

{

}

→ PartTime extend Employee

{

getEmployee

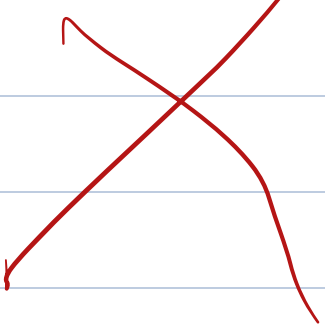
}

ParentClass

{

Emply getEmployee ()

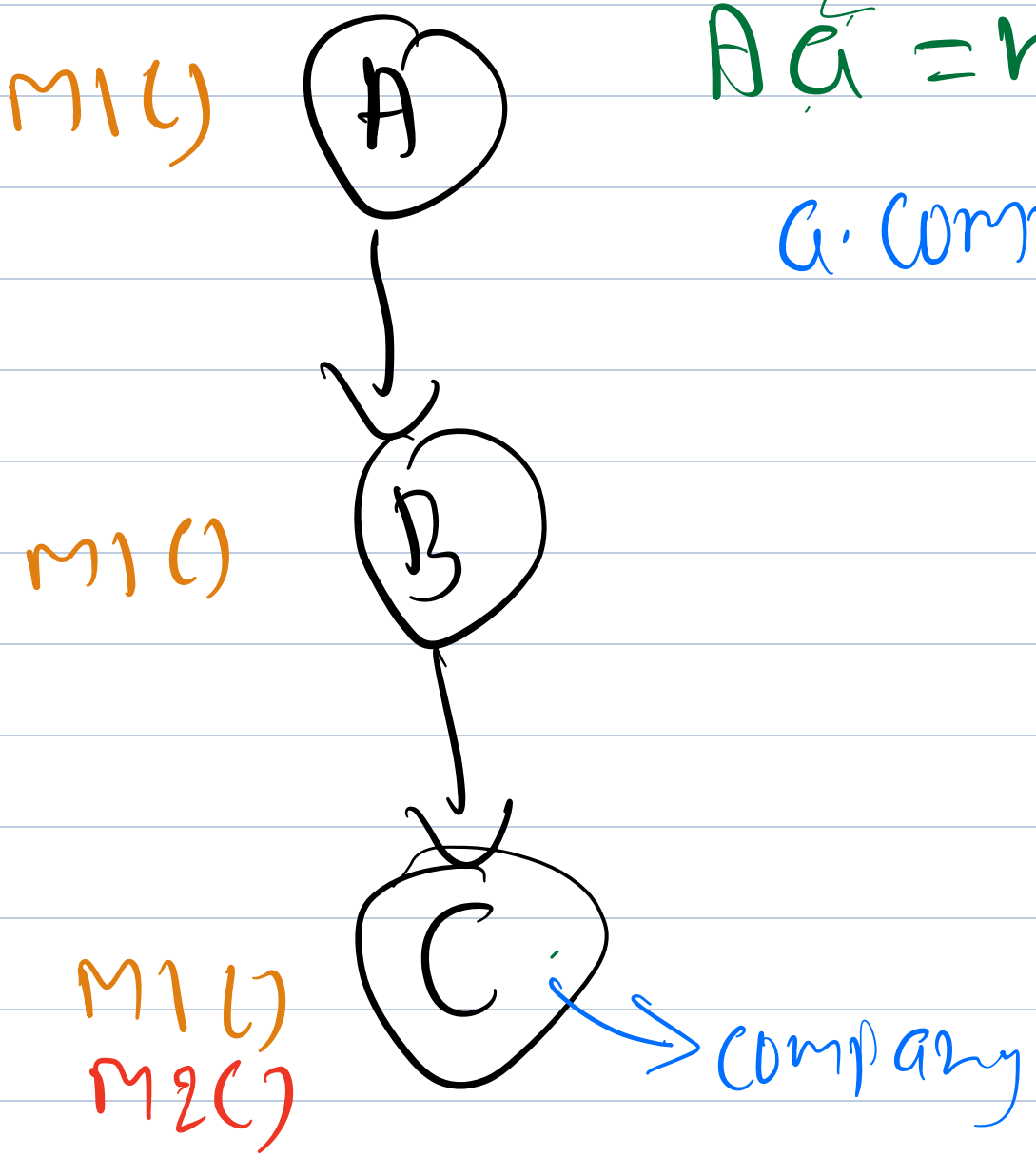
{



Child Class extends Parttime {

getEmployee()

}



A a = new C()

a.company X.