Constructor

Class → Blueprint of entity.
Object → Instance of class.

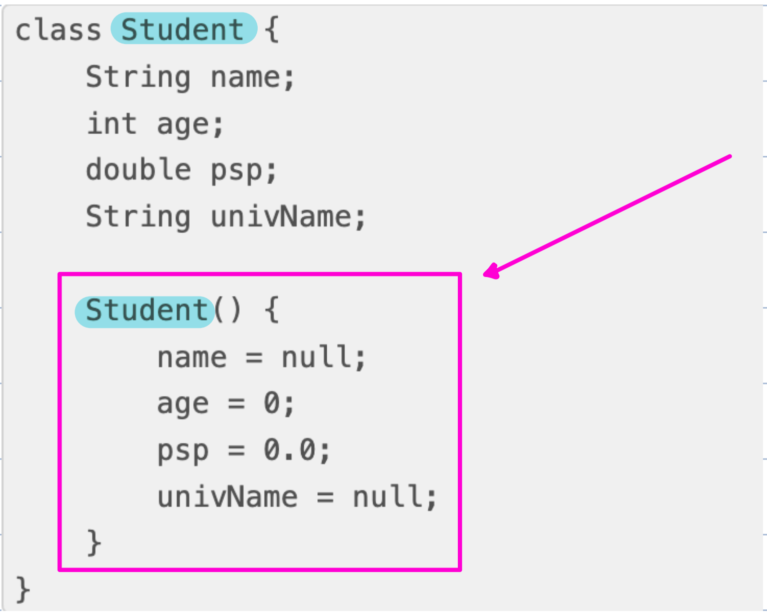
```
class Student {
    String name ;
    int age ;
    double psp ;
}
```

```
Student st = new Student ();
int a = 10 ;
```

Default constructor

creates the object of the class & set default values of the attributes.

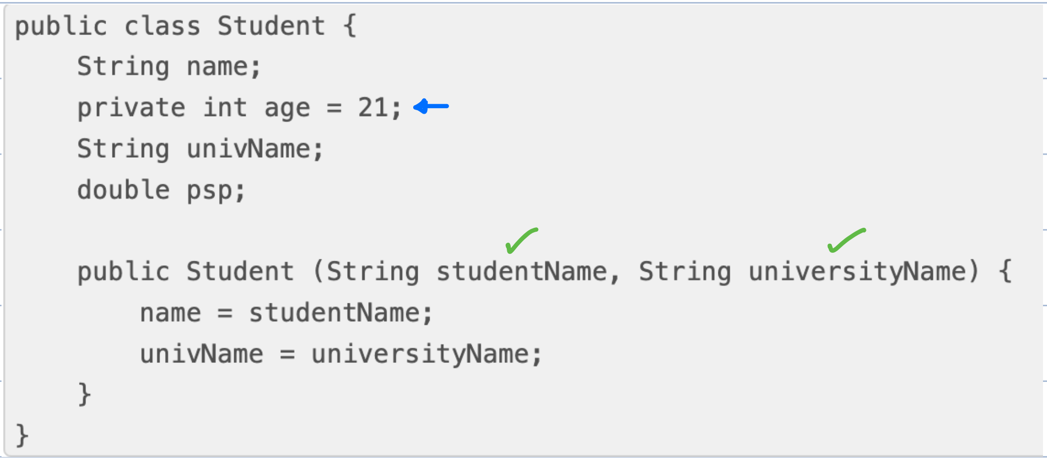
Eg → int = 0,
String = null,
double = 0.0,
boolean = false, etc.



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

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

No return type.
Same name as the class.
Its public. (default constructor)



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

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

$\boxed{psp = 0.0}$

```
public class Client {
    public static void main(String[] args) {
        Student st = new Student(); //ERROR
}
```

→ No default constructor if we have manual constructor.

```
public class Client {
public static void main(String[] args) {
    Student st = new Student("Utkarsh", "JIIT");
}
```
✓

Copy constructor

Used to create copy of an existing object.

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

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

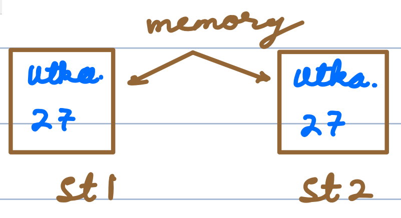
copy constructor

```
Student st1 = new Student();
st1.name = "Utkarsh";
st1.age = 27;

Student st2 = new Student(st1); // Copy Constructor
```

New object is created.

Student $\boxed{st3}$ = st1;

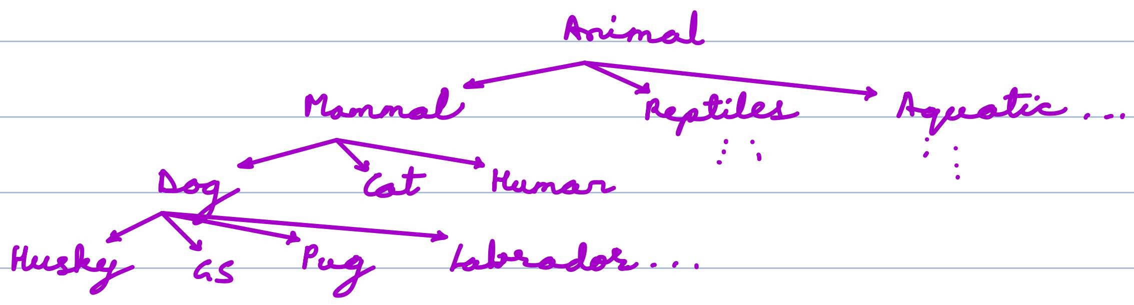


// Object reference for same object (no new object).

```
Student st2 = new Student (st1); // Deep Copy
        st2.name = "Bharath"
        print (st1.name)  // Utkarsh

Student st3 = st1;   // Shallow Copy
        st3.name = "Rakesh"
        print (st1.name)  // Rakesh
```

Inheritance



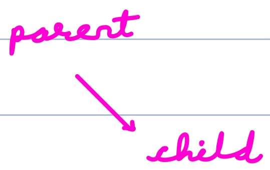
Animals can move ✓

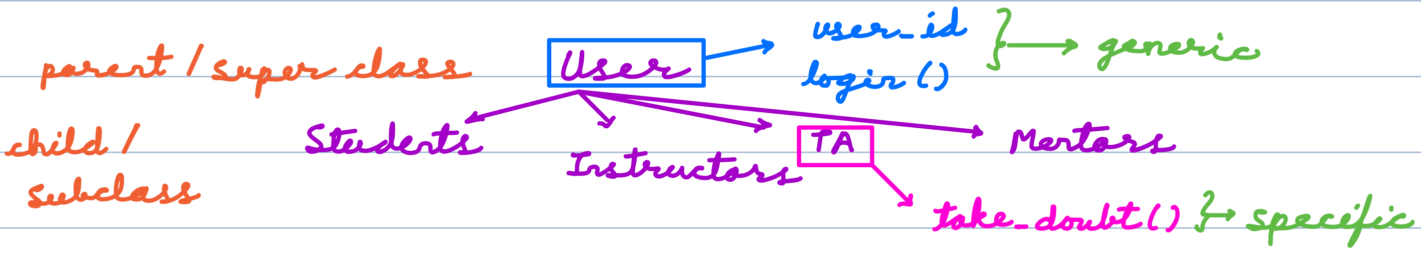
⇒ Mammal can move

⇒ Dog can move

⇒ Pug can move

Representation of hierarchies in classes is inheritance.





A child class inherits all the members of the parent class & may or may not add their own members. ✓

```
class User {
    String userName;

    void login() {
        ...
    }
}
```

Java → class Instructor extends User

Python → class Instructor (User)

C++ → class Instructor : public User

C# → class Instructor : User

⋮

child parent

```
class Instructor extends User {
    String batchName;
    double avgRating;

    void scheduleClass() {
        ...
    }
}
```

---

constructor chaining

Instructor i = new Instructor () ;

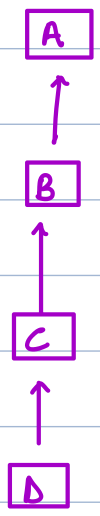
i. avgRating = 4.9 ;

i. userName = "Utkarsh" ;

How is userName initialized ?

Using constructor of parent class which is called within constructor of child class.

$$\begin{tikzcd}
 & & \boxed{A} & \text{class} \\
 & \boxed{B} \arrow[ur] & & \boxed{C} \arrow[ul] \\
\boxed{D} \arrow[ur] & \boxed{E} \arrow[u] & & \boxed{F} \arrow[u] \\
 & & & & \boxed{G} \arrow[ul]
\end{tikzcd}$$



D d = new D();

	completion
constructor of D is called	4
⇒ calls constructor of C	3
⇒ calls constructor of B	2
⇒ calls constructor of A	1

```
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();

calls parent class constructor

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

$\underbrace{\text{Poly}}_{\text{many}} - \underbrace{\text{morphism}}_{\text{forms}}$

Animal a = new Dog(); ✓

Dog d = new Animal(); ✗

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

```
class B extends A {
    String uni;
}
```

```
class C extends A {
    double psp;
}
```

```
A a = new C();
a.psp = 50.0;   // gives error
```

HW → [typecast to C & use attributes of class C.]

How to access attributes of C.

Types of Polymorphism

1) compile Time → Method overloading

(methods with same name but different parameters.)

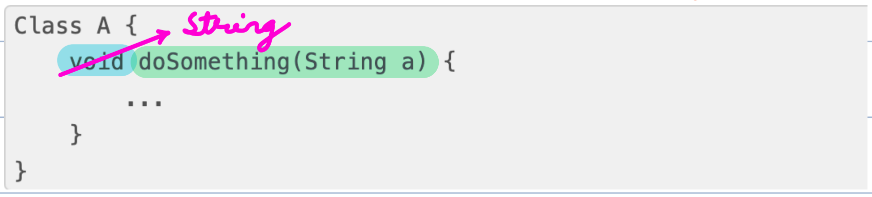
count / datatype / order ✓

```
int add (x, y) {
    return x+y;
}
```

method signature

```
int add (x, y, z) {
    return x+y+z;
}
```

2) Run Time → Method overriding



```
Class A {
    ~~void~~ String doSomething(String a) {
        ...
    }
}
```

```
Class B extends A {
    String doSomething(String c) { // overriding parent class function
        ...
    }
}
```