Agenda :-

Inheritance

Polymorphism

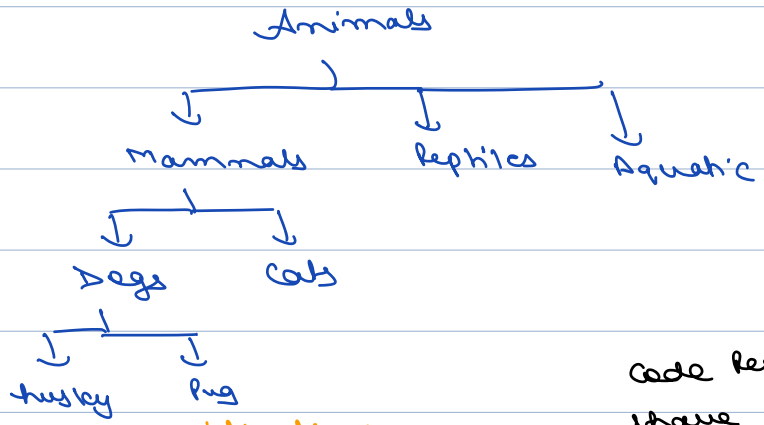
Method Overloading

Method Overriding,

Inheritance

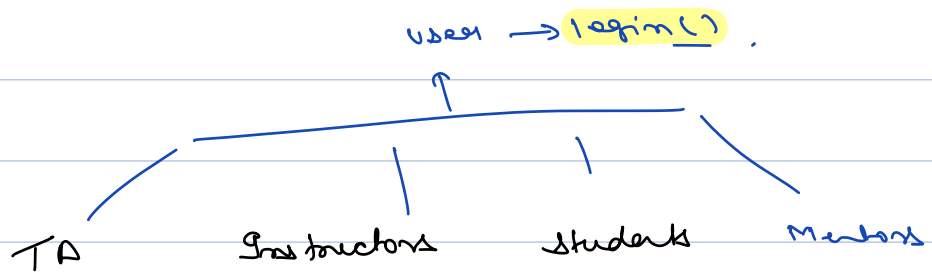
→ Hierarchy.

→ General.



→ Specifications.

Code reusability
share attributes
&
Methods



logically parent + child.

Animal
↓
canWalk()

Robot
↓
canWalk()

Inheritance

is a relationship.

Parent
↑
child

inherits all data members & member fn from parent.

+

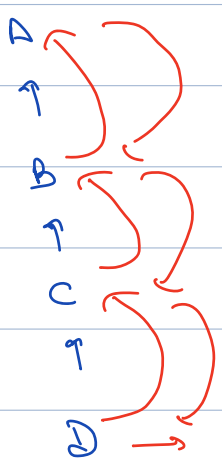
Private methods / members are not inherited.

class User
{

}

class TA extends User
{

}



child

super();

this();

```java
// Telescoping of Constructors
Student(){  2 usages  new *
    batch = "Free Batch";
    psp = 1;
}

Student(String batch){  1 usage  new *
    this();
    this.batch = batch;

}


Student(String batch, double psp){  1 usage  new *
    this(batch);
    this.psp = psp;
}

Student(String batch, double psp,int age){  no usages  new *
    this(batch,psp);
    this.age = age;
}
```

batch = Acad ~~Free Batch~~

psp = 30

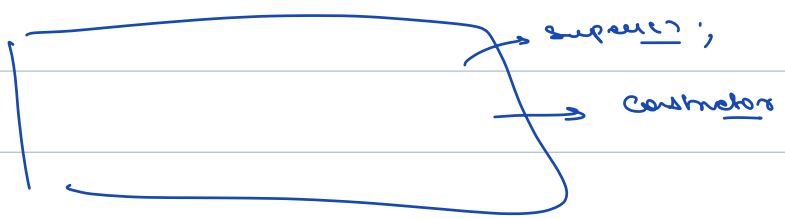
age = 25.

Telescoping of constructors

Student st = new Student ("Acad", 30, 25)

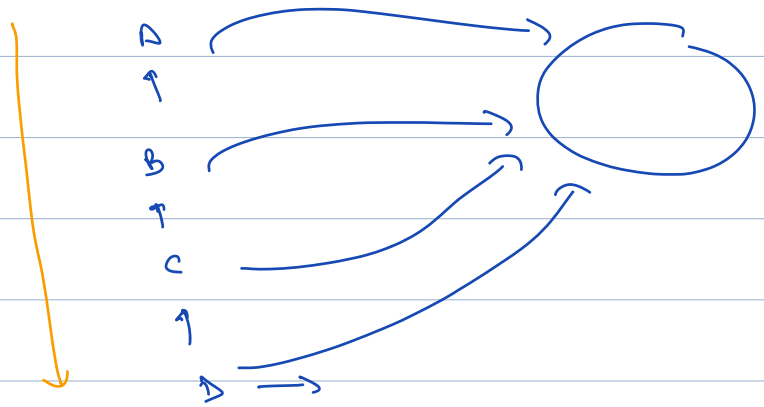
super(); → calls constructor of your parent

this() → This has to be first line;



this() & super()

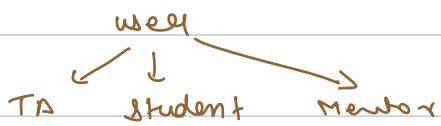
↳ can't be written together.



Polymorphism

many forms

Something / someone who has many forms.



printUserName.

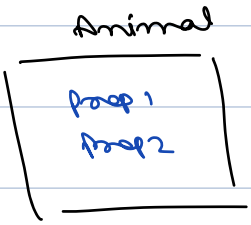
```
printUserName ( List<TA, Student, Mentor> user ) {

}
```

```
user y = new TA();
user x = new Student();
user z = new Mentor();
```

parent class ref. can point to a child class object.

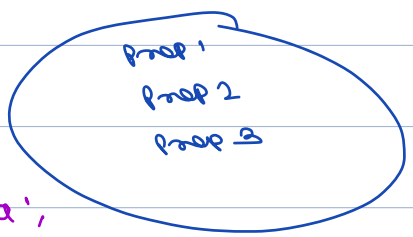
Vice versa is not true :.

```
Dog d = new Dog();
d.bark();
```




```
Animal a = new Dog();
a.prop1
a.prop2
```

↓ compiler



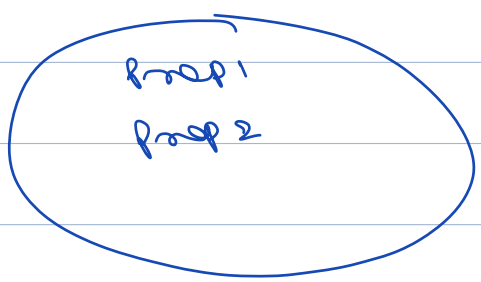

```
Dog d1 = (Dog)a;
```

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

→ not allowed

→ compiler stops it

```
d.prop1
d.prop2
d.prop3 X
```



n. anything

↳ datatype.

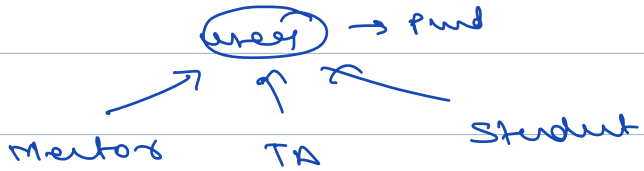
you access things based on type of your reference, not the object you are pointing to;

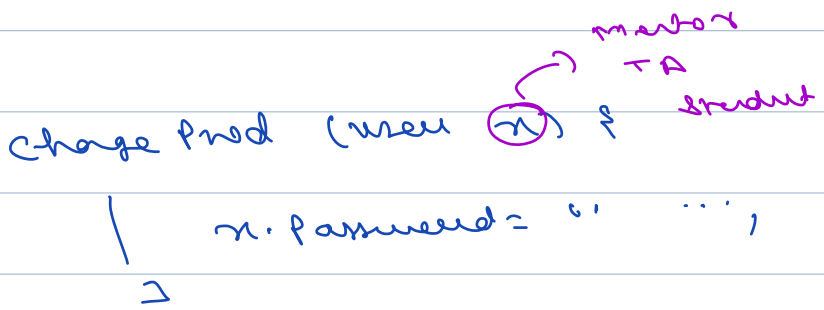
```
Dog d = new Animal(); X

Animal a = new Dog();
a.bark();
```

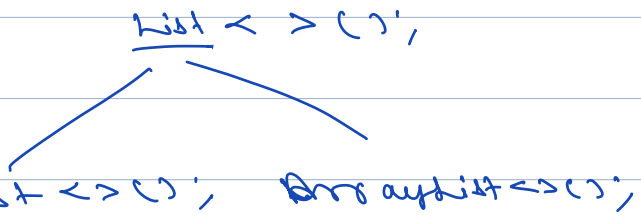

```
doSomething(Animal n) {
    n.bark(); X
}
```

Dog
cat

3



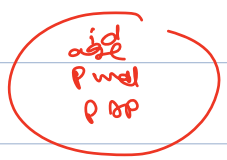




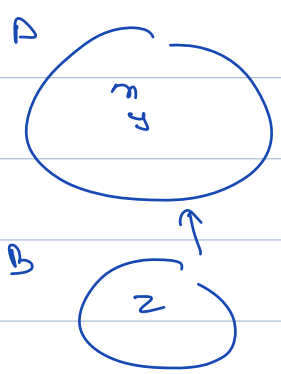

```
List <> l = new LinkedList();
List <> l = new ArrayList();
```

```
class A {
    id;
}

class B ext A {
    age;
    pwd;
}

class C ext B {
    psp;
}
```

```
A a = new B();   ✓
a.pwd            ✗

C c = new C;
c.psp   ✓
c.age   ✓
c.id    ✓

B b = new C();   ✓
b.psp   ✗
```




```
Dog d = new Animal();   ✗
Animal a = new Dog();   ✓
a.bark();               ✗
```




```
A a
a.x
a.y

B b
b.x
b.y
b.z
```

Method Overloading

```
class D {

    hello ( ) {
    }
    hello (x) {
    }
}
```

Method signature → name of func
→ Type of arg.
→ no. of arg.

↳ name (Dt1, Dt2, Dt3 - -)

```
add (strx, stry) {
}
```

```
add (int x, int y) {
}
```

```
add (x, y, z) {
}
```

✗

```
int add (x, y) {
}
```

```
String add (x, y) {
}
```

Method Overriding

```
class A {
    void doSomething (String) {        (1)
    }
}
```

```
class B extend A {        can't change return type
    void doSomething (String) {        (2)
    }
}
```

```
A a = new A();
a.doSomething();        <- (1)

B b = new B();
b.doSomething();        <- (2)
```

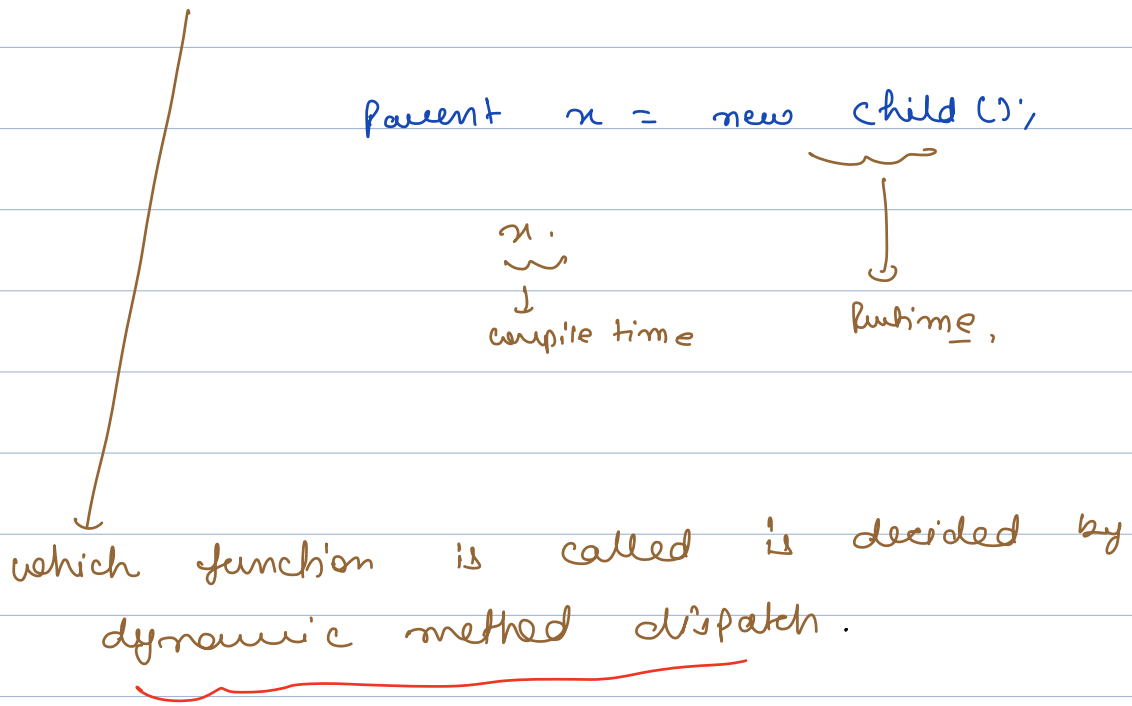
Runtime polymorphism:

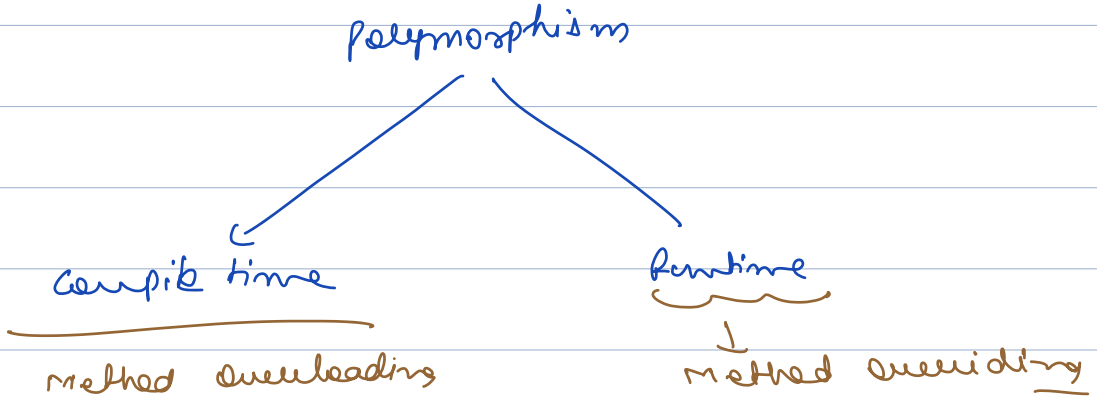
```
A a = new B();
a.doSomething();        -> (2)
```

allowed since A has doSomething available;

```
B b = new A();   → not allowed
b.doSomething();
```

in Java, when you call reference.any function, which function will be called is decided at run time and it will choose the function which is present in the object

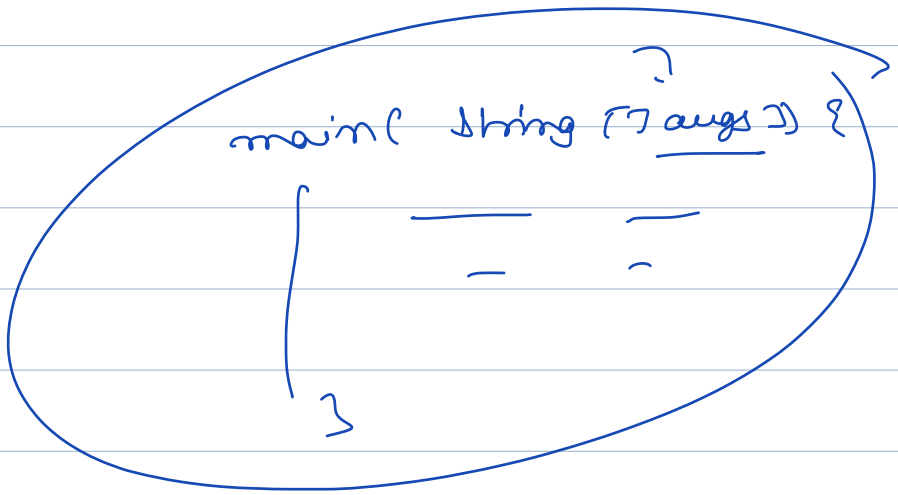



polymorphism
compile time
runtime
method overloading
method overriding

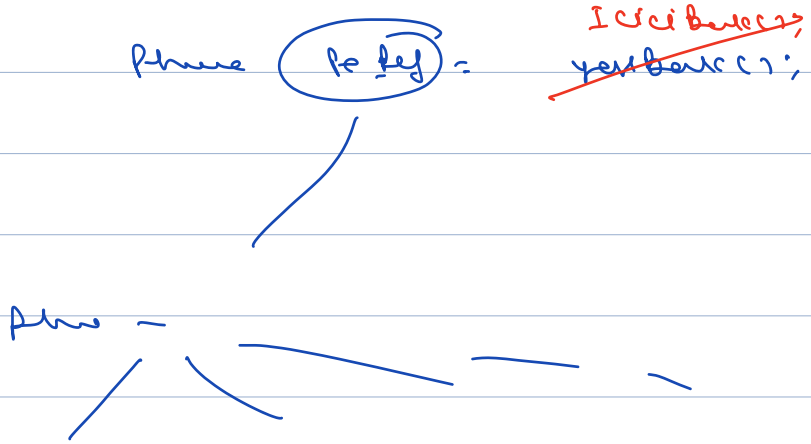
List → addLast();

LinkedList
addLast

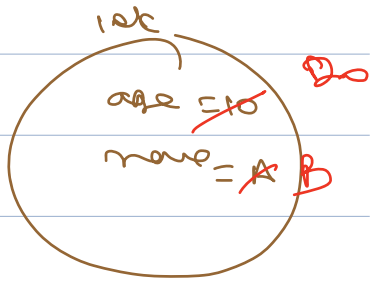
ArrayList
addLast();


main(String [] args) {
}

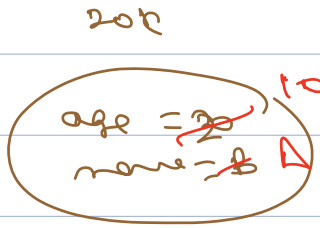

Terminal →



```
Student {
    age;
    name;

    display() {
    }

    sayHello() {
    }
}
```




```
Student S1 = new Student
Student S2 = new Student
Swap(S1,S2)
```




```
Swap (S1 . S2) {
    temp = S1.name
    S1. —
    ——
}
```