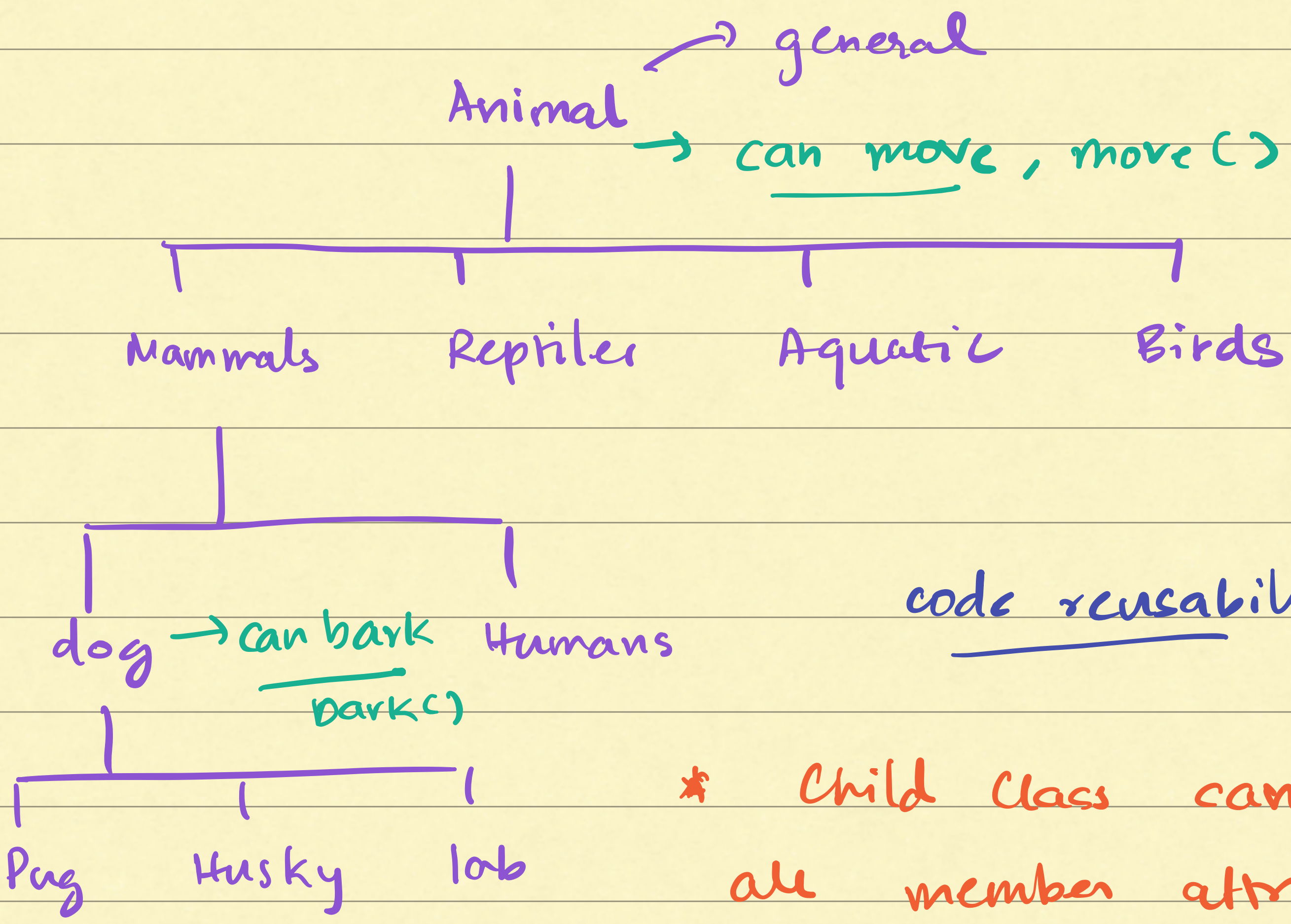
Agenda :

- Inheritence
- Polymorphism

hierarchy of entities .

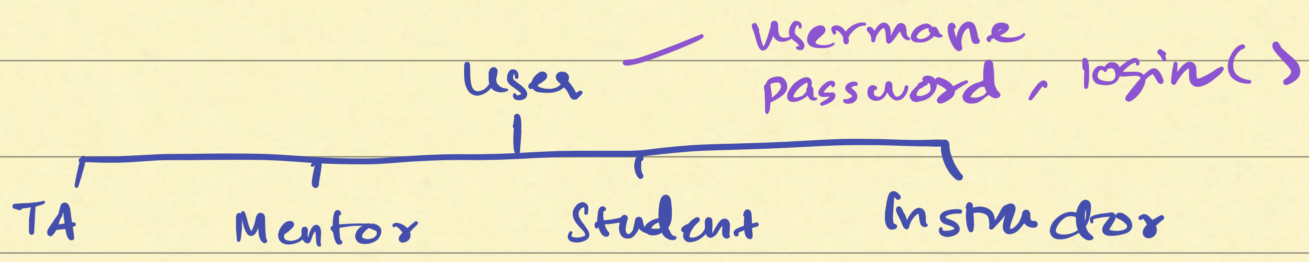


code reusability

* Child Class can inherit all member attributes and methods of the parent class .

- but not vice versa .

- Private members are exceptions , they are not inherited .



```
Class User {
  String username;
  String pass;
  boolean login(){
     =
     =
  }
}
```

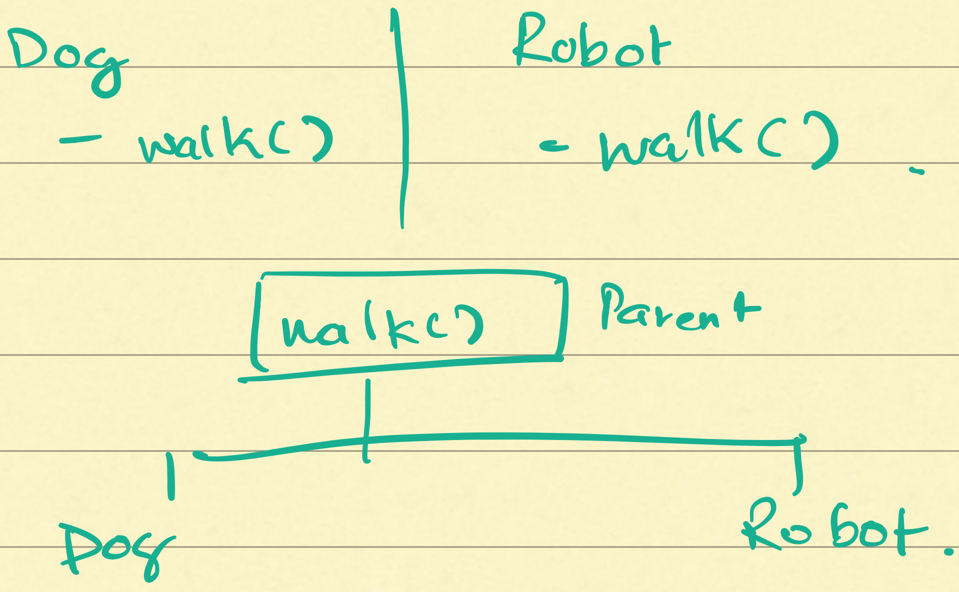
should be common for all sorts of user (TA, Mentor, Student)

```
class Student extends User {
    String username;
    String pass;
    boolean login() {
       =
       =
       =
    }

    String batch ;
    float psp ;
    void changeBatch () {
       =
    }
```

are not explicitly visible but inherited from parent.

}

Inheritence helps to relate physically or logically related entities.



Polymorphism

poly → many   morph → forms

many forms

TA is an User
Mentor is an User
student is an user

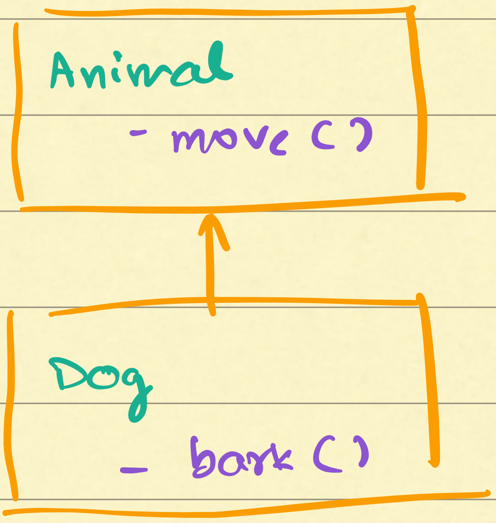
is many forms of User

```
                              user
printUserNames ( List < ? > users )
                        Datatype?
```

```
User u  = new Student();  ✓
User u1 = new Mentor();   ✓
```

Parent class Reference variable can point / hold child object. (not vice versa)

```
Student s = new User(); X
```

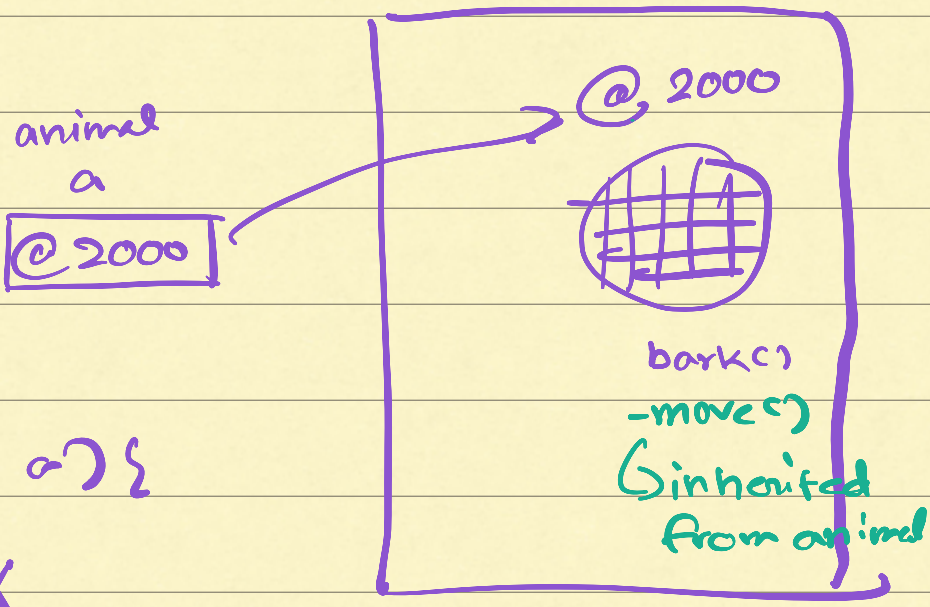


✓ ① Animal a = new Dog();

✓ ② a. move ();

✗ ③ a. bark ();

↳ compile time error



```
void fun ( Animal a){
    a. bark(); X
}
```

```
fun( new Dog());
fun( new Cat());
```

Method Overloading (compile time polymorphism)

Using the same method with different arguments ∴ method signature must be different.

Method Signature :

return type methodName (Datatype1, Datatype2 ...)

return type → doesn't matter.

Datatypes → order matters in signature

→ variable names also doesn't matter.

```
① void print (String batch, String name);
② void print (int psp);
③ void print (String batch, int psp);
④ void print (int psp, String batch);

⑤ void print (String name, int age) ✗

⑥ int print (float);
⑦ String print (float);
```

⑥, ⑦ } can't have different return type, if signature is same.

print(0.5)
print(0.7) .

⑥ string print(long); ✓

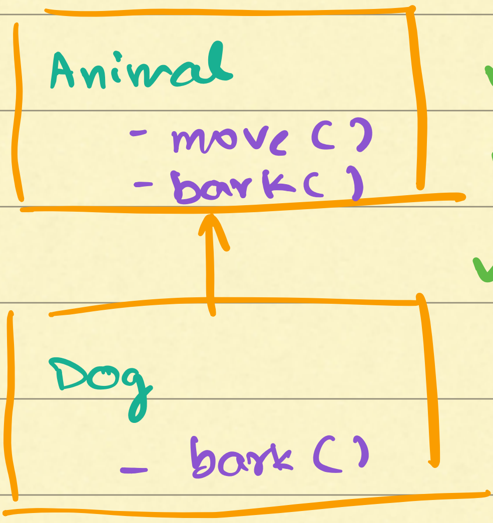
print(5l);

print("Akash", 6);
print(5, "Akash);

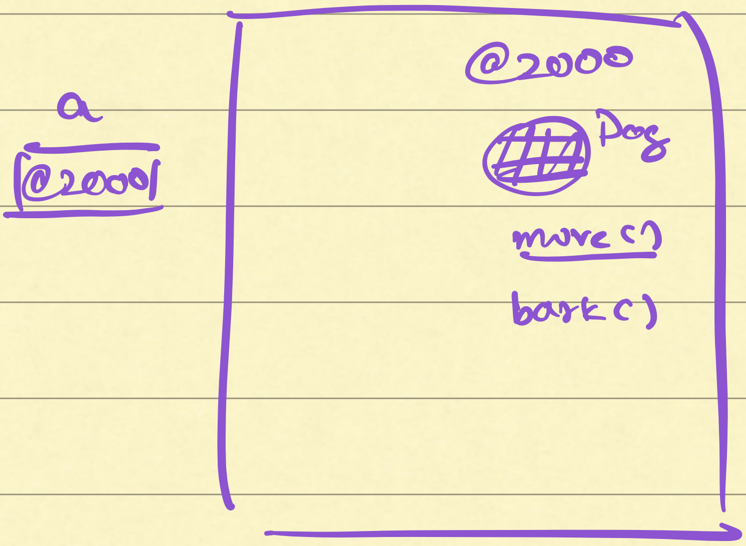
Method Overriding : (Runtime Polymorphism)

```
class Animal {
    void move(){
        ==      walk();
    }
}

class Dog extends Animal {
     void move() {      → not explicitly
        ==                 visible.
     }
     void bark() {
        ==
     }
     void move(int x) {  } method
        ==               } overloading.
        ==
     }

     void move() {
        ==
        ==          gym
        ==
     }
}
```

```
User
  - changePassword {
        ====
↓     }
Student
   - changePassword {
        extra checks
            ==
     }
```



```
✓ ① Animal a = new Dog();
✓ ②     a.move();
✓ ③     a.bark();
         ↳ output:
```

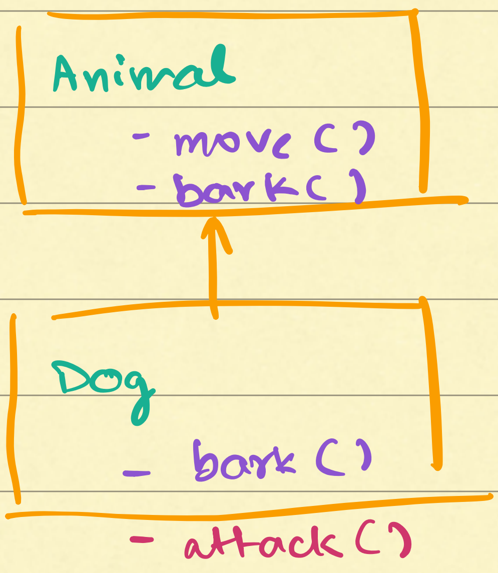


```
class Animal {
    void bark() {
      sout("Animal Barking");
    }
}
class Dog extends Animal {
    void bark() {
    sout("Dog barking");
```

}
}

compiler time — Animal Bark check
Run time — Dog Bark run.

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



Dog d = new Dog();
d. attack ();

Animal a = new Dog();

void commandToAttack
(Dog d) {

}