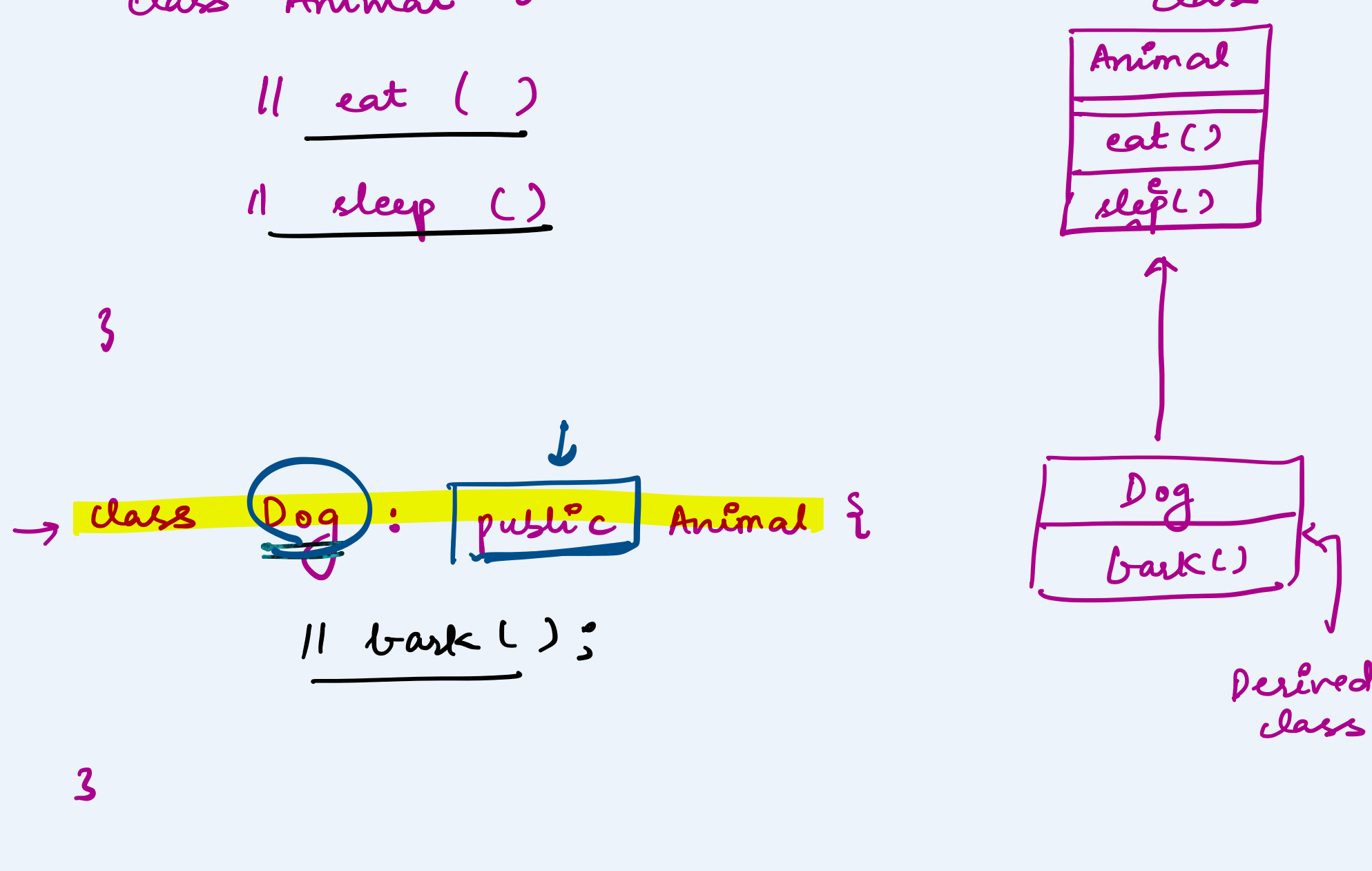


## Inheritance

→ allows us to create a new class from an existing class.

→ The derived class inherits the features from the base class.



class Dog extends Animal {

Inheritance  
is - a relationship

→ A car is a vehicle  
→ Orange is a fruit  
→ A dog is a animal

```

class Animal {
    // code
}

class Dog : private Animal {
    // code
}
    
```

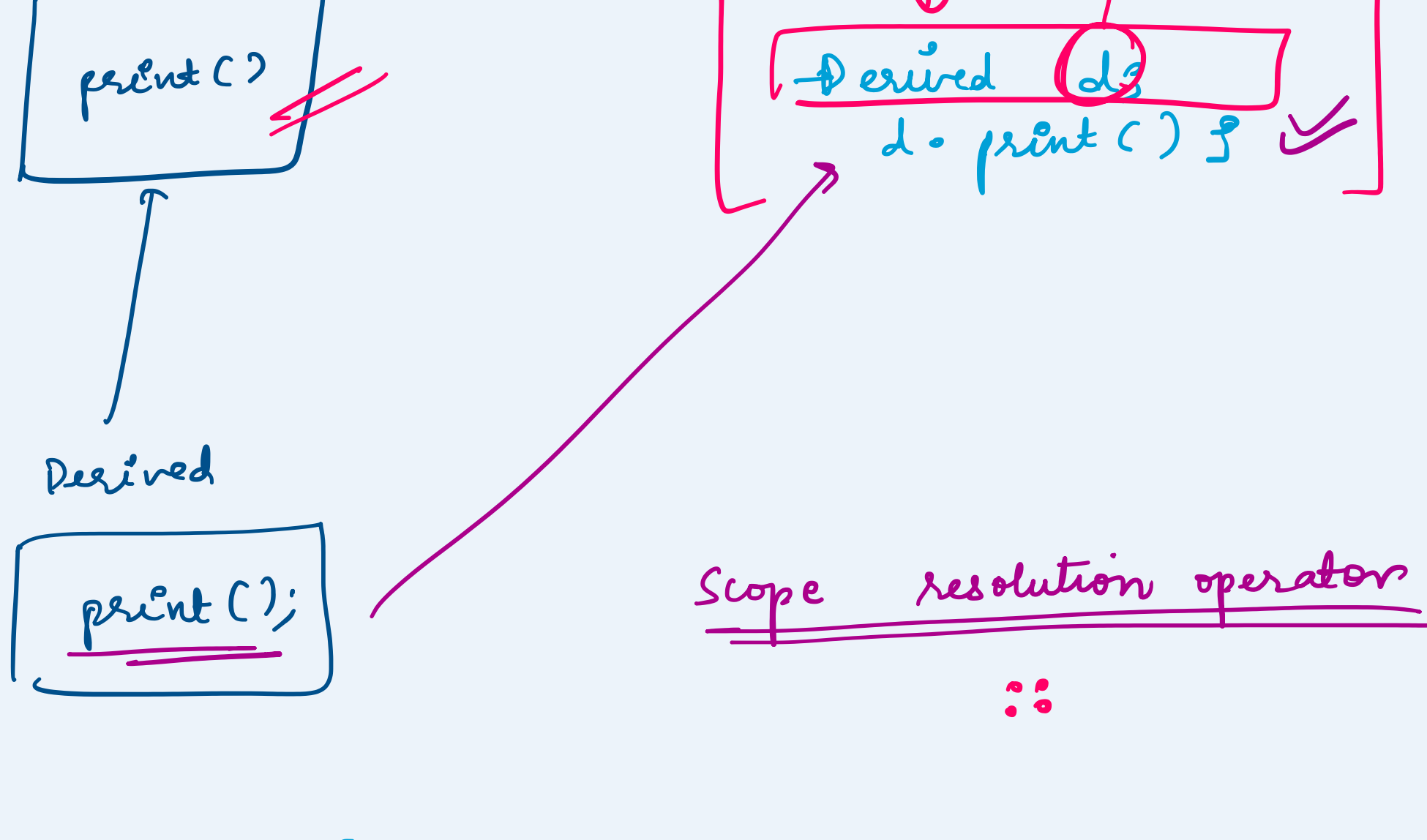
Annotations: public (pointing to Animal), protected (pointing to Animal), private (pointing to Dog's inheritance)

① public: just the way they are

② private: all members of base class becomes private in derived class

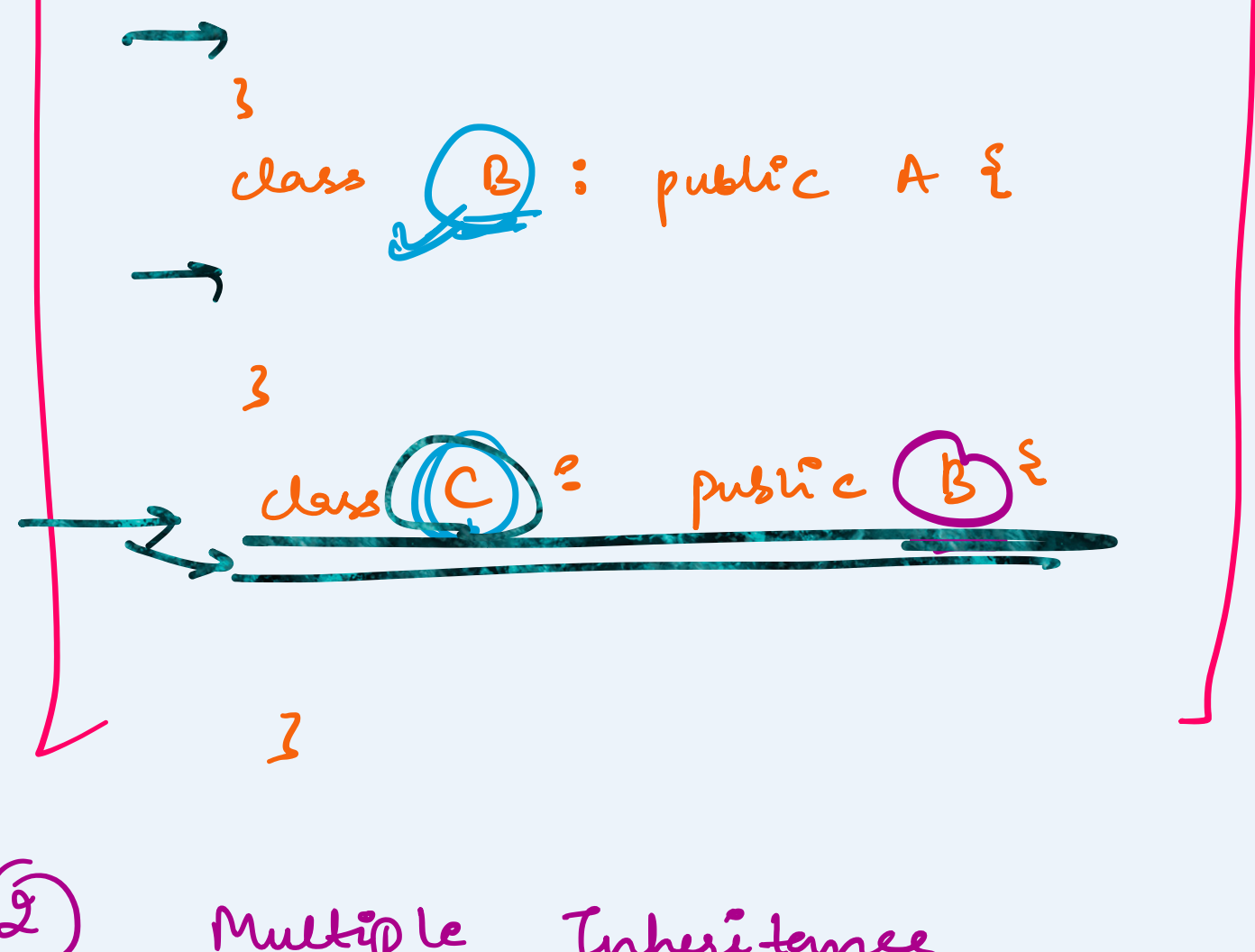
③ protected: public → protected, private → private

## function overriding

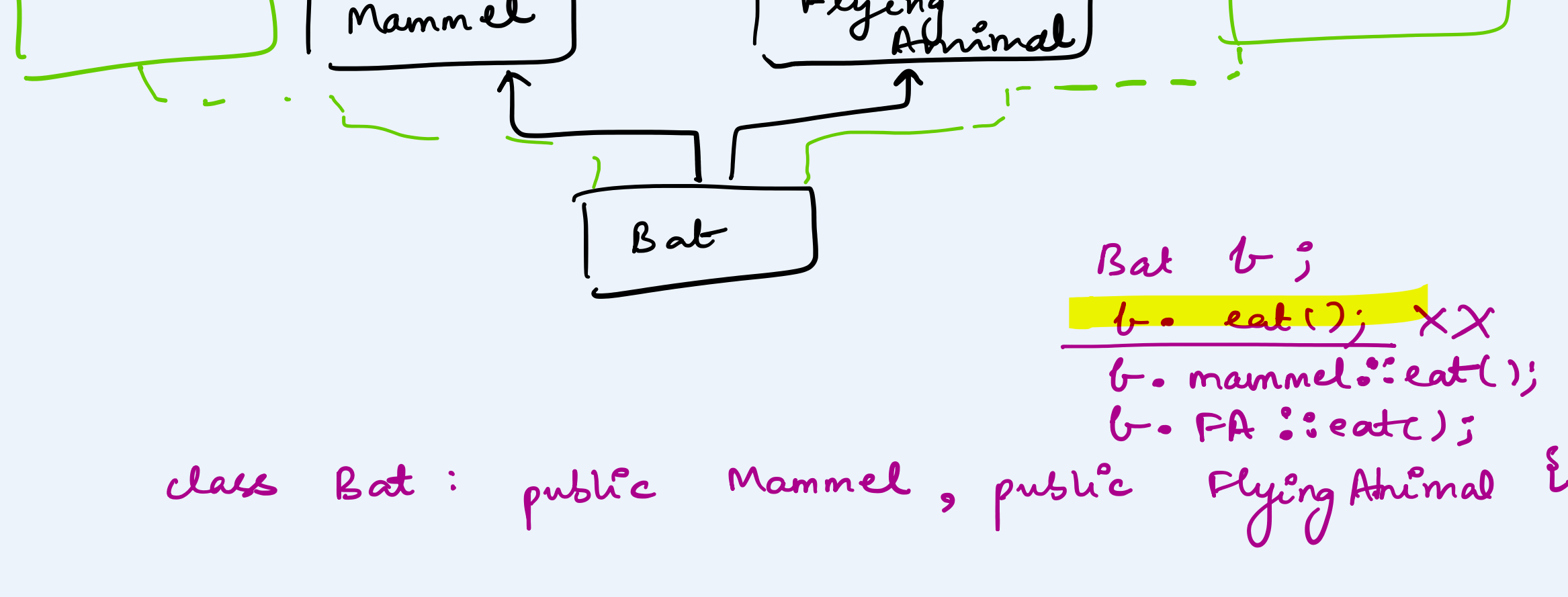


## Types of Inheritance

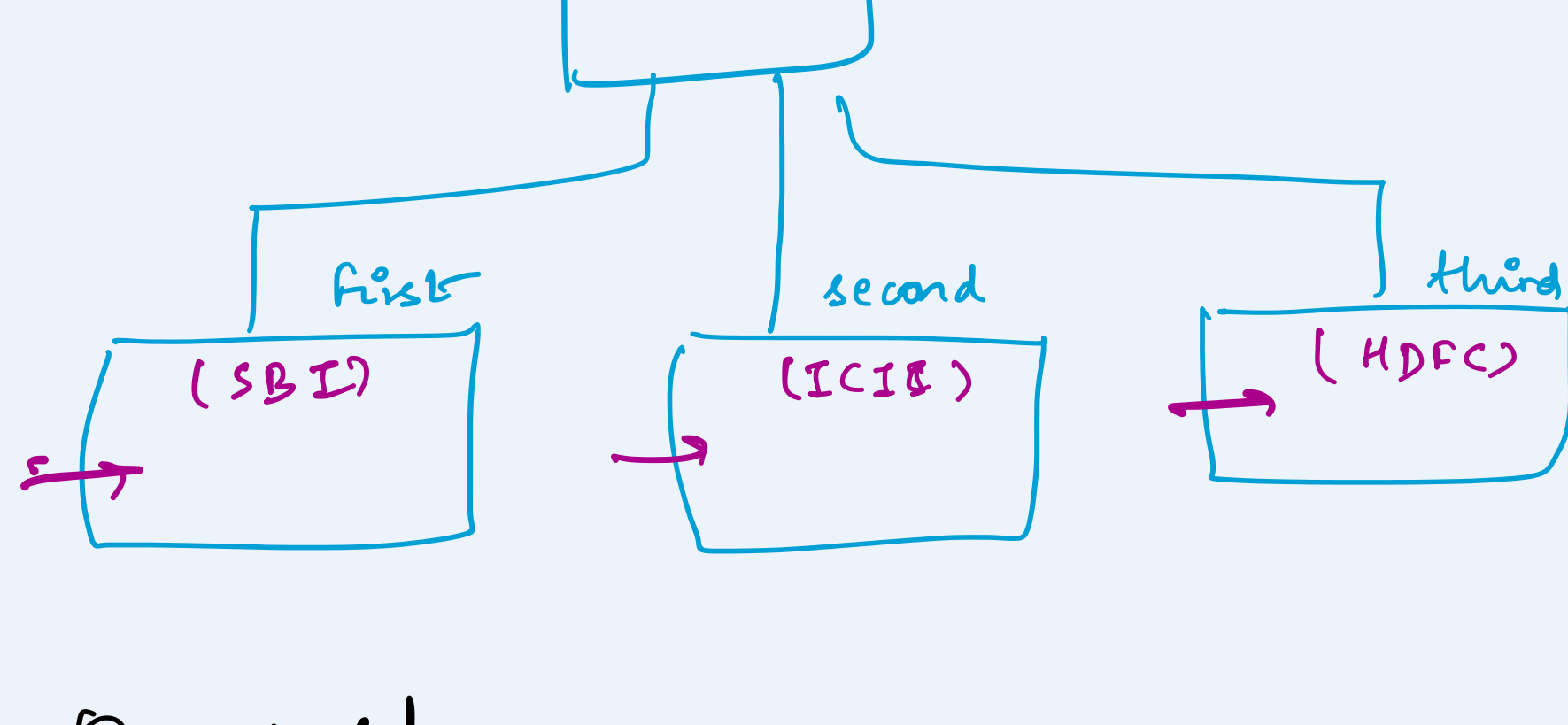
→ ① Multi level



② Multiple Inheritance



## Hierarchical Inheritance



④ Hybrid

## Friend function

→ allows to access member functions from outside the class.

```

class A {
private:
    val 1;
    friend add(A, B);
}

class B {
private:
    val 2;
    friend add(A, B);
}

add(classA objA, classB objB) {
    return objA.val1 + obj2.val2;
}
    
```

## Friend class

class friend

all the member functions of the friend class become friend functions

```

class classA {
    friend class classB;
}

class classB {
}
    
```

## Virtual Functions

→ A virtual function is a member function in the base class that we expect to redefine in derived class.

```

class Base {
public:
    virtual void display() {
    }

    void draw();
}

class Derived : public Base {
public:
    void display() {
    }
}

int main() {
    Derived d;
    Base* b = &d;
    b->display();
    b->draw();
}
    
```

Annotations: @override (pointing to Derived's display), override (pointing to Derived's display), derived function (pointing to Derived's display)