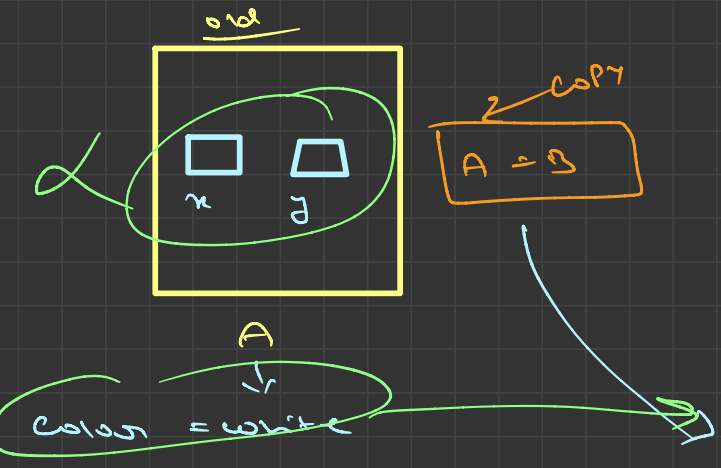
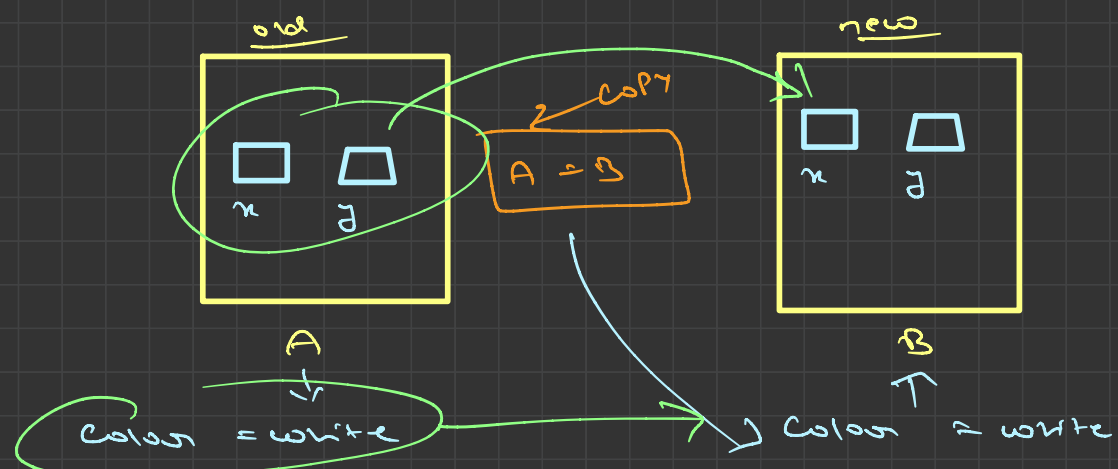
Shallow Copy → behen behen se copy karna, not deep copy → andar ka content copy nhi krega

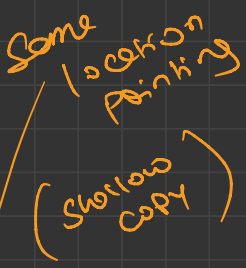


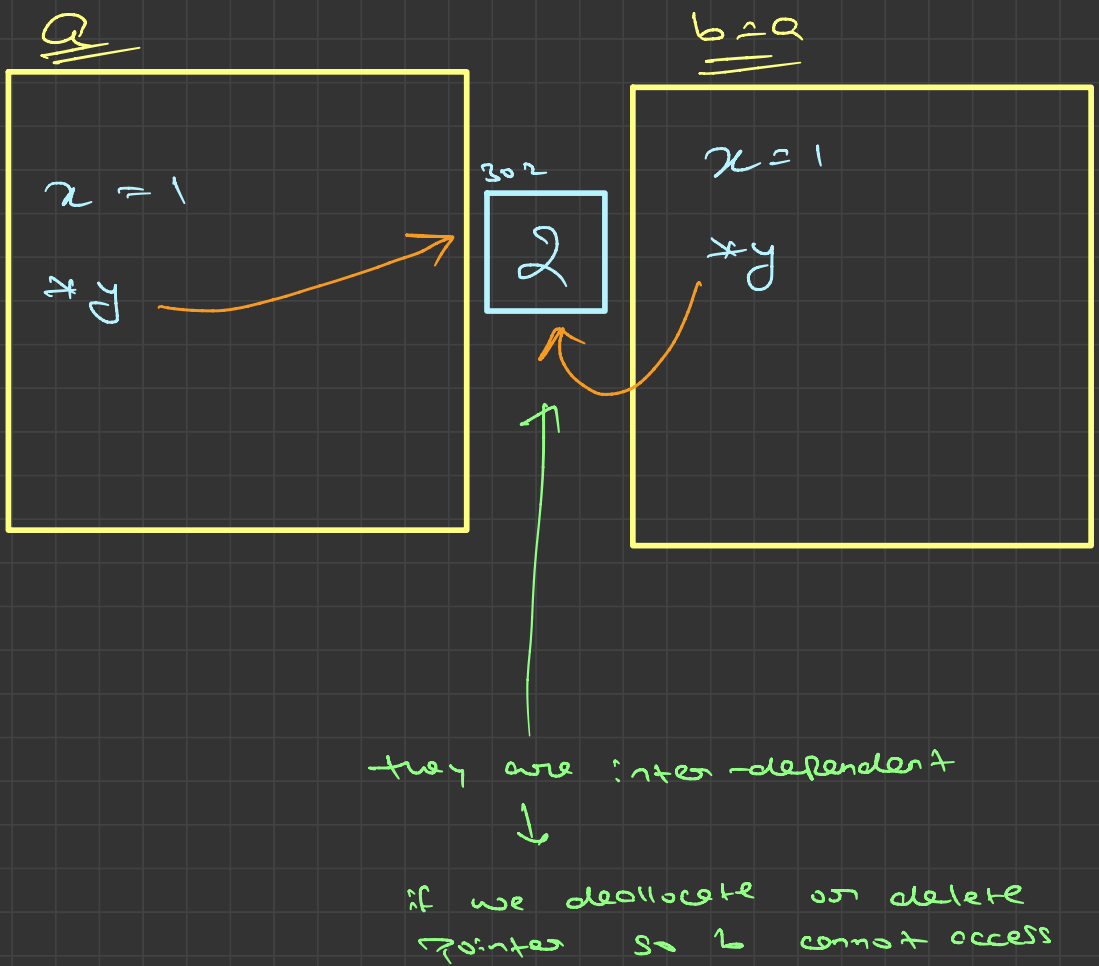
Deep Copy → Inner & outer copy



```cpp
#include<iostream>
using namespace std;

class abc{
    public:
        int x;
        int *y;

        abc(int _x, int _y) : x(_x), y(new int (_y)) {}

        // default copy constructor called automatically -> dumb in nature -> shallow copy
        // abc(const abc &obj){
        //     x = obj.x;
        //     y = obj.y;
        // }

        void print() const{
            cout << "x: " << x << endl;
            cout << "PTR y: " << y << endl;
            cout << "Content of y (*y): " << *y << endl;
        }
};

int main(){
    abc a(1,2);
    cout << "printing a" << endl;
    a.print();
    cout << endl;

    // copy
    abc b = a; // shallow copy
    // abc b(a); // deep copy
    cout << "printing b" << endl;
    b.print();
    cout << endl;

    b.x = 10;
    *b.y = 20;
    cout << "printing b" << endl;
    b.print();
    cout << endl;

    cout << "printing a" << endl;
    a.print();
}
```

```
printing a
x: 1
PTR y: 0x142605e60
Content of y (*y): 2

printing b
x: 1
PTR y: 0x142605e60
Content of y (*y): 2

printing b
x: 10
PTR y: 0x142605e60
Content of y (*y): 20

printing a
x: 1
PTR y: 0x142605e60
Content of y (*y): 20
```




a
x = 1
*y
302
2
b = a
x = 1
*y
they are inter-dependent
if we deallocate or delete pointer a, b cannot access

```cpp
#include<iostream>
using namespace std;

class abc{
    public:
        int x;
        int *y;

        abc(int _x, int _y) : x(_x), y(new int (_y)) {}

        // default copy constructor called automatically -> dumb in nature -> shallow copy
        // abc(const abc &obj){
        //     x = obj.x;
        //     y = obj.y;
        // }

        // our smart copy constructor -> deep copy
        abc(const abc &obj){
            x = obj.x;
            y = new int(*obj.y);
        }

        void print() const{
            cout << "x: " << x << endl;
            cout << "PTR y: " << y << endl;
            cout << "Content of y (*y): " << *y << endl;
        }
};

int main(){
    abc a(1,2);
    cout << "printing a" << endl;
    a.print();
    cout << endl;

    // call -> copy constructor
    abc b = a;
    // abc b(a); // same
    cout << "printing b" << endl;
    b.print();
    cout << endl;

    b.x = 10;
    *b.y = 20;
    cout << "printing b" << endl;
    b.print();
    cout << endl;

    cout << "printing a" << endl;
    a.print();
}
```

Constructor Called

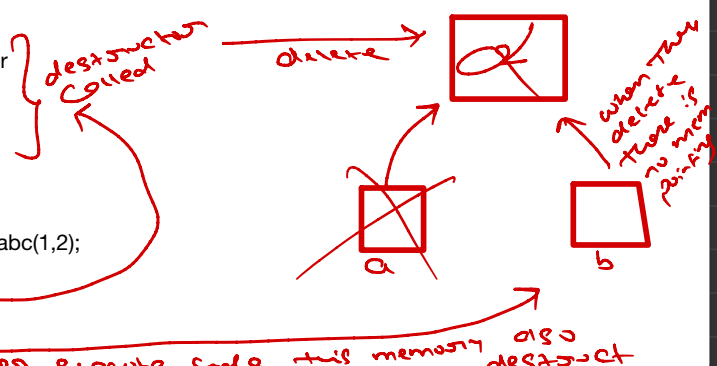
```
printing a
x: 1
PTR y: 0x157605e60
Content of y (*y): 2

printing b
x: 1
PTR y: 0x157606020
Content of y (*y): 2

printing b
x: 10
PTR y: 0x157606020
Content of y (*y): 20

printing a
x: 1
PTR y: 0x157605e60
Content of y (*y): 2
```

a has pointing to some memory

Some memory

Shallow copy destructor

```
#include<iostream>
using namespace std;

class abc{
    public:
        int x;
        int *y;

        abc(int _x, int _y) : x(_x), y(new int (_y)) {}

        // default copy constructor called automatically -> dumb in nature -> shallow copy
        // abc(const abc &obj){
        //     x = obj.x;
        //     y = obj.y;
        // }

        void print() const{
            cout << "x: " << x << endl;
            cout << "PTR y: " << y << endl;
            cout << "Content of y (*y): " << *y << endl;
        }

        // destructor
        ~abc(){
            delete y;
        }
};

int main(){
    abc *a = new abc(1,2);
    abc b = *a;
    delete a;
    b.print();
}
```




```
x: 1
PTR y: 0x153606020
Content of y (*y): 0

copy(69752,0x1e2035300) malloc: Double free of object 0x153606020
copy(69752,0x1e2035300) malloc: * set a breakpoint in malloc_error_break to debug
```

Correctly given → delete a

abort access the memory already deleted by a

But when we use deep copy constructor with pointer there is no error when destructor called