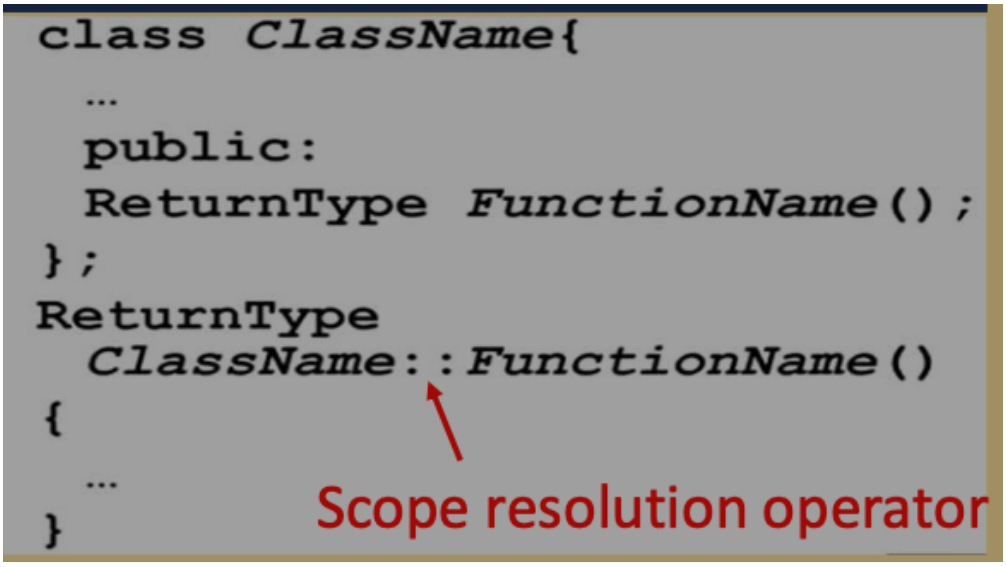
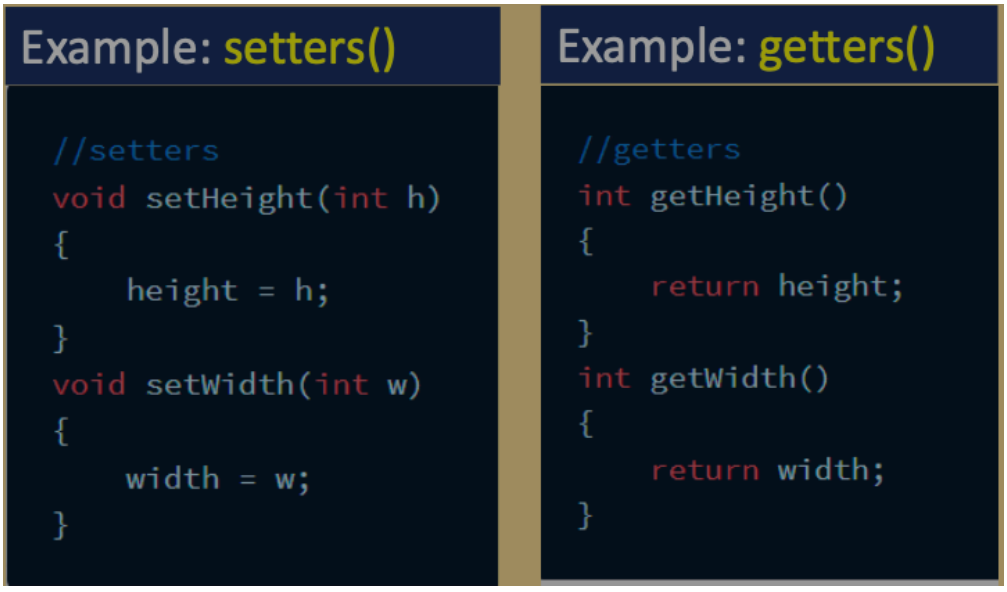
2 - Classes & Objects in OPP

A class is an entity that determines how an object will behave and what the object will contain.

- blueprint or a set of instruction to build a specific type of object.



Setter & Getter



Access Control

- Public
- Private
- Protected

Constructor

Special class functions which performs initialization of every object.

- name of a constructor function is same as name of class.
- no return type.

Destructor

A function that runs automatically when an object is destroyed.

- (~) followed by the class name.
- No arguments and no return type for a destructor.
- **Use:** deallocated memory that was allocated for the object by the constructor.

Class in different file

```
#include <iostream>
#include "reg.h"
```

Static Data Member

Static data members are class members that are declared using static keywords.

- Only one copy of that member is created for the entire class and is shared by all the objects.
- it is initialized before any object of this class is created, even before the main starts.
- only visible within class, but lifetime is the entire program.
- value must be initialized outside the class.
- the getter function must be static.

Passing Object as Parameter

🧩 1. Pass by Value

When you pass an **object itself** (not its address).
A **copy** of the object is made.

```cpp
void AvgSpeed(Ball b) {   // receives a copy of b2
    cout << (s + b.s) / 2;
}
...
b1.AvgSpeed(b2);
```

➡️ Here `b` is a **copy** of `b2`.
So if you change `b.s`, it won't affect `b2.s`.
✅ Safe, but slower (because of copying).

🧩 2. Pass by Pointer

When you pass the **address** of an object using a pointer.

```cpp
void AvgSpeed(Ball *b) {
    cout << (s + b->s) / 2;
}
...
b1.AvgSpeed(&b2);
```

➡️ Here `b` points directly to `b2`.
So any change done through `b` affects `b2`.
✅ Fast (no copy), but less safe (possible pointer errors).

🧩 3. Pass by Reference

When you pass the **reference** to an object (like an alias).

```cpp
void AvgSpeed(Ball &b) {
    cout << (s + b.s) / 2;
}
...
b1.AvgSpeed(b2);
```

➡️ Works like pointer internally but syntax is cleaner (`b.s`, not `b->s`).
✅ Fast **and** safe — this is the **most common** in modern C++.

⚡ Summary Table

Type	Example	Syntax	Copies Object?	Can Modify Original?
Pass by Value	`AvgSpeed(Ball b)`	`b.s`	✅ Yes	❌ No
Pass by Pointer	`AvgSpeed(Ball *b)`	`b->s`	❌ No	✅ Yes
Pass by Reference	`AvgSpeed(Ball &b)`	`b.s`	❌ No	✅ Yes

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

```
class Ball{
private:
    int s;
public:
    Ball(){}
    Ball(int x){
        s = x;
    }
    void AvgSpeed(Ball *b){
        cout << (s + b->s) / 2;
    }
    int main()
    {
        Ball b1(130), b2(140);
        b1.AvgSpeed(&b2);
        return 0;
    }
}
```

Object as Function Return

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

class Ball{
private:
    float s;

public:
    Ball(){}
    Ball(int x){
        s = x;
    }
    float GetSpeed(){
        return s;
    }
    Ball AvgSpeed(Ball *b){
        Ball t;
        t.s = s + b->s;
        return t;
    }
    int main(){
        Ball b1(130), b2(140);
        Ball k;
        k = b1.AvgSpeed(&b2);
```

```
        cout << k.GetSpeed() / 2;
        return 0;
    }
}
```

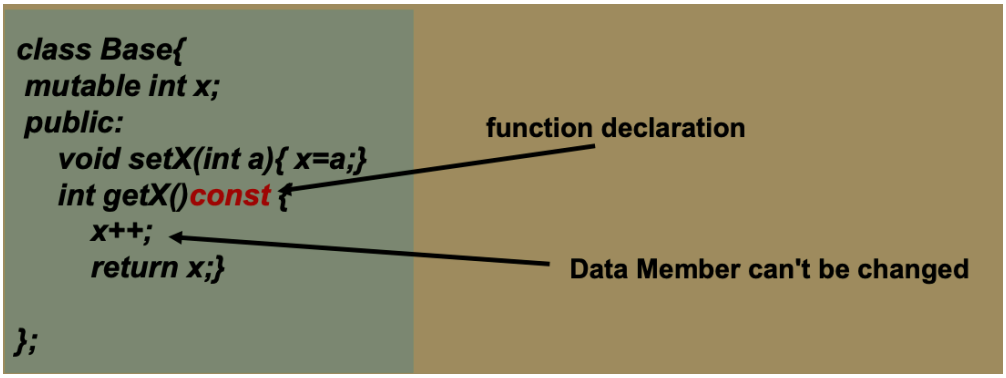
Copy Constructor

A copy constructor is a member function that initializes an object using another object of the same class.

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

class Ball{
private:
    float s;

public:
    Ball(){}
    Ball(int x){
        s = x;
    }
    Ball(Ball &b){  // copy constructor
        s = b.s;
    }
    float GetSpeed(){
        return s;
    }
};

int main(){
    Ball b1(150);
    Ball b2(b1); // copy constructor
    cout << b2.GetSpeed();
}
```

Const Member Function

- Makes no modification about the data members.



- mutable --> changeable
- inmutable --> not changeable.

Static Member Function

- can contain only static data member.
- can run using: `<classname> :: <static function name>()`

Non static member function can contain both static and non-static data member.

Friend Function

A friend function of a class defined outside that class' scope but it has the right to access all private and protected members of the class.

- friends are not member functions.

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

class test{
    private:
        int n;
    public:
    test(int x){
        n = x;
    }
    friend void show(test *t);
};

void show(test *t){
    cout << "n = " << t->n << '\n';
}

int main(){
```

```
    test t(10);
    show(&t);
}
```