


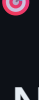

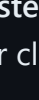


# Misc. OOP Concepts in C++

-  [Nested Classes](#)
-  [Abstract Base Class](#)
-  [Static](#)
-  [Final](#)
-  [Friends](#)
-  [Quiz!](#)

## Nested Classes

A **nested class** is a class defined within another class. It helps encapsulate helper classes that are only relevant to the outer class.

```
class Outer {
private:
    int number;
public:
    class Inner {
    public:
        void display(const Outer& outer) {
            std::cout << "Inside Inner class\n" << outer.number;
        }
    };
};
```

### Use Cases:

- Logical grouping of classes
- Encapsulation of helper functionality

### Access:

- The nested class has access to the outer class's members!

## Abstract Base Class

An **abstract base class** is a class that contains at least one **pure virtual function**. It cannot be instantiated and is used to define an interface.

```
class Animal {
public:
    virtual void speak() = 0; // Pure virtual function
};
```

### Purpose:

- Enforce a contract for derived classes
- Enable **polymorphism**

### Example:

```
class Dog : public Animal {
public:
    //derived classes must override the pure virtual function
    //if they do not, the derived class will become abstract too
    void speak() override {
        std::cout << "bark bark\n";
    }
};
```

## Static

The `static` keyword has different meanings depending on context:

### Static Variable (Inside Function)

- Retains its value between function calls

```
void counter() {
    static int count = 0;
    count++;
    std::cout << count << "\n";
}
```

### Static Member (Inside Class)

- Shared across all instances of the class

```
class MyClass {
public:
    static int count;
};
int MyClass::count = 0;
```

## Final

The `final` specifier prevents further overriding or inheritance.

### Prevent Overriding:

```
class Base {
public:
    virtual void show() final {
        std::cout << "Base show\n";
    }
};
```

### Prevent Inheritance:

```
class FinalClass final {
    // Cannot be inherited
};
```

## Friends

A `friend` function or class can access the **private** and **protected** members of another class.

### Friend Function:

```
class MyClass {
private:
    int secret = 42;
    friend void reveal(MyClass);
};

void reveal(MyClass obj) {
    std::cout << obj.secret << "\n";
}
```


### Friend Class:

```
class A {
    friend class B;
private:
    int data = 100;
};

class B {
public:
    void show(A a) {
        std::cout << a.data << "\n";
    }
};
```

## Quiz!

Here's a short quiz on the topic: [quiz](#)

 Footer Separator

## Markdown Viewer

How to view the markdown files in a browser...

- [Markdown Viewer](#)

## Lecture Practices

Here are the lecture Practices...

- [Day 7](#)
- [Day 8](#)
- [Day 9](#)

## Lecture Quizzes

Here are the lecture quizzes...

- [Day 7](#)
- [Day 8](#)
- [Day 9](#)

## Weekly Topics

Here are the topics for the week...

- [Classes](#)
- [Structs](#)
- [Fields](#)
- [Getters and Setters](#)
- [Constructors](#)
- [Instances](#)
- [Inheritance](#)
- [Polymorphism](#)
- [Pointers](#)
- [Upcasting](#)
- [Misc. Concepts](#)
- [4 Pillars of OOP](#)