

# The C++ Master Companion — Syntax, Insight & Practice

ZephyrAmmor

October 2025

## Contents

Module 5: Object-Oriented Programming (OOP) in C++	1
5.1 OOP Basics	1
5.2 Classes and Objects	1
5.3 Access Specifiers	2
5.4 Constructors and Destructors	2
5.5 Inheritance	2
5.6 Polymorphism	3
5.7 Abstraction	3
5.8 this Pointer	4
5.9 Static Members	4
5.10 Friend Function	4
5.11 Practical Summary Table	4
5.12 Visualization — Inheritance Hierarchy	4
5.13 Common Pitfalls	5
5.14 Quick Checklist	5

## Module 5: Object-Oriented Programming (OOP) in C++

Purpose: Introduce and master the principles of Object-Oriented Programming — encapsulation, inheritance, polymorphism, and abstraction — with strong syntax understanding and practical examples.

### 5.1 OOP Basics

Concept: OOP allows modular, reusable, and maintainable code by modeling real-world entities as objects.

Core Principles (EIPA):

1. Encapsulation: Binding data and functions into a single unit (class).
2. Inheritance: Acquiring properties and behaviors of another class.
3. Polymorphism: Same function behaving differently for different objects.
4. Abstraction: Hiding complex details and showing only the necessary features.

### 5.2 Classes and Objects

Syntax:

```
class ClassName {
private:
    int data;
public:
    void setData(int d) { data = d; }
    int getData() { return data; }
};

int main() {
    ClassName obj;
```

```

    obj.setData(10);
    std::cout << obj.getData();
}

```

Visual Representation:

```

+-----+
|   ClassName   |
+-----+
| - data : int   |
+-----+
| + setData(int) |
| + getData() : int |
+-----+

```

---

### 5.3 Access Specifiers

Specifier	Accessibility	Example Use
public	Accessible everywhere	Class interface
private	Within class only	Internal data members
protected	Within class + derived classes	For inheritance

---

### 5.4 Constructors and Destructors

Constructor: Automatically called when an object is created.

```

class Person {
    string name;
public:
    Person(string n) { name = n; }
};

```

Destructor: Automatically called when an object is destroyed.

```

~Person() { cout << "Object destroyed"; }

```

Types of Constructors:

- Default: Person() {}
  - Parameterized: Person(string n)
  - Copy: Person(const Person &p)
- 

### 5.5 Inheritance

Syntax:

```

class Base {
public:
    void greet() { cout << "Hello"; }
};

class Derived : public Base {
public:
    void intro() { cout << "I am derived."; }
};

```

Types of Inheritance:

- Single(class B : public A)
- Multiple(class C : public A, public B)
- Multilevel(class C : public B : public A)

- Hierarchical(class B, C : public A)
- Hybrid (combination)

Access Modifiers in Inheritance:

Inheritance Type	Base public	Base protected	Base private
Public	public	protected	—
Protected	protected	protected	—
Private	private	private	—

## 5.6 Polymorphism

Compile-time (Static)

### 1. Function Overloading

```
class Calc {
public:
    int add(int a, int b) { return a + b; }
    double add(double a, double b) { return a + b; }
};
```

### 2. Operator Overloading

```
class Complex {
    int r, i;
public:
    Complex(int a=0, int b=0): r(a), i(b) {}
    Complex operator + (Complex const &obj) {
        return Complex(r + obj.r, i + obj.i);
    }
};
```

Runtime (Dynamic)

- Achieved using virtual functions and pointers.

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

class Derived : public Base {
public:
    void show() override { cout << "Derived"; }
};

Base *ptr = new Derived();
ptr->show(); // Output: Derived
```

## 5.7 Abstraction

- Achieved using abstract classes (with pure virtual functions).

```
class Shape {
public:
    virtual void draw() = 0; // pure virtual
};

class Circle : public Shape {
public:
    void draw() override { cout << "Drawing Circle"; }
};
```

---

## 5.8 **this** Pointer

- Points to the calling object.

```
class A {  
    int x;  
public:  
    A(int x) { this->x = x; }  
};
```

---

## 5.9 Static Members

- Belong to class, not object.

```
class Counter {  
    static int count;  
public:  
    Counter() { count++; }  
    static int getCount() { return count; }  
};  
int Counter::count = 0;
```

---

## 5.10 Friend Function

- Can access private/protected data.

```
class Box {  
    int width;  
    friend void printWidth(Box b);  
};  
  
void printWidth(Box b) {  
    cout << b.width;  
}
```

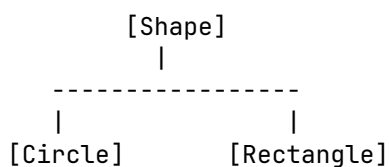
---

## 5.11 Practical Summary Table

Concept	Keyword(s)	Type	Example
Class	class	Encapsulation	class Car {}
Inheritance	:	Reusability	class Tesla : public Car {}
Polymorphism	virtual, override	Flexibility	ptr->show()
Abstraction	=0	Interface Design	virtual void draw()=0;
Friend	friend	Controlled Access	friend void f();
Static	static	Shared Resource	static int count;

---

## 5.12 Visualization — Inheritance Hierarchy



### 5.13 Common Pitfalls

- Forgetting `virtual` in base class when overriding.
  - Not initializing base class constructors in derived ones.
  - Using object slicing when assigning derived to base by value.
  - Memory leaks due to non-virtual destructors.
- 

### 5.14 Quick Checklist

- ☒ Know how to create classes & objects.
  - ☒ Understand all access modifiers.
  - ☒ Can use constructors/destructors properly.
  - ☒ Comfortable with inheritance & overriding.
  - ☒ Understand static, friend, and `this`.
- 

Next → Module 6: Advanced C++ Concepts (Templates, Exception Handling, Namespaces, File Handling)