📘 Understanding Fields in C++ Classes

- 🧠 What Are Fields?
- 🧱 Syntax of Fields in a Class
- 🏏 Example: `Student` Class with Fields
- 🔐 Access Control for Fields
- 🧠 Best Practices
- ✍️ Naming Conventions for Fields
  - 🔤 Common Conventions:
  - ✅ Best Practice Example:
- 🎯 Quiz!

🧠 What Are Fields?

In C++, **fields** (also called **data members** or **member variables**) are variables declared inside a class. They represent the **state or attributes** of an object. Each object has its **own copy** of the fields.

🧱 Syntax of Fields in a Class

```
class ClassName {
  private:
    int field1;
    double field2;

  public:
    void setField1(int value);
    int getField1();
};
```

🏏 Example: `Student` Class with Fields

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

class Student {
  private:
    string name;
    int age;

  public:
    Student(string n, int a) {
        name = n;
        age = a;
    }

    void displayInfo() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Student s1("Alice", 20);
    Student s2("Bob", 22);

    s1.displayInfo();
    s2.displayInfo();

    return 0;
}
```

🔐 Access Control for Fields

Fields can be declared with different **access modifiers**:

Modifier	Description
`private`	Accessible only within the class (default for classes)
`public`	Accessible from outside the class
`protected`	Accessible within the class and derived classes

🧠 Best Practices

- Use `private` fields to protect internal state.
- Provide `public` **getter** and **setter** methods to access and modify fields.
- Initialize fields using **constructors**.

✍️ Naming Conventions for Fields

Consistent naming conventions improve **readability**, **maintainability**, and help distinguish between **fields**, **parameters**, and **local variables**.

🔤 Common Conventions:

Convention	Example	Description
`camelCase`	`studentName`	Common in many C++ codebases
`snake_case`	`student_name`	Often used in C-style or embedded systems
`m_` prefix	`m_age`	"member" prefix to distinguish fields from parameters
`_` suffix	`name_`	Common in modern C++ to avoid naming conflicts
`this->` usage	`this->name`	Explicitly refers to the field when shadowed by a parameter

✅ Best Practice Example:

```
class Student {
  private:
    string name_;
    int age_;

  public:
    Student(string name, int age) {
        name_ = name;
        age_ = age;
    }

    void displayInfo() {
        cout << "Name: " << name_ << ", Age: " << age_ << endl;
    }
};
```

Or using `this->`:

```
Student(string name, int age) {
    this->name = name;
    this->age = age;
}
```

🎯 Quiz!

Here's a short quiz on the topic: quiz

🔭 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