OOP introduction

C++ for Developers

Content

- What is OOP?
- Classes & objects
- Access Modifiers
- Static and non-static
- Operator overloading

What is OOP?

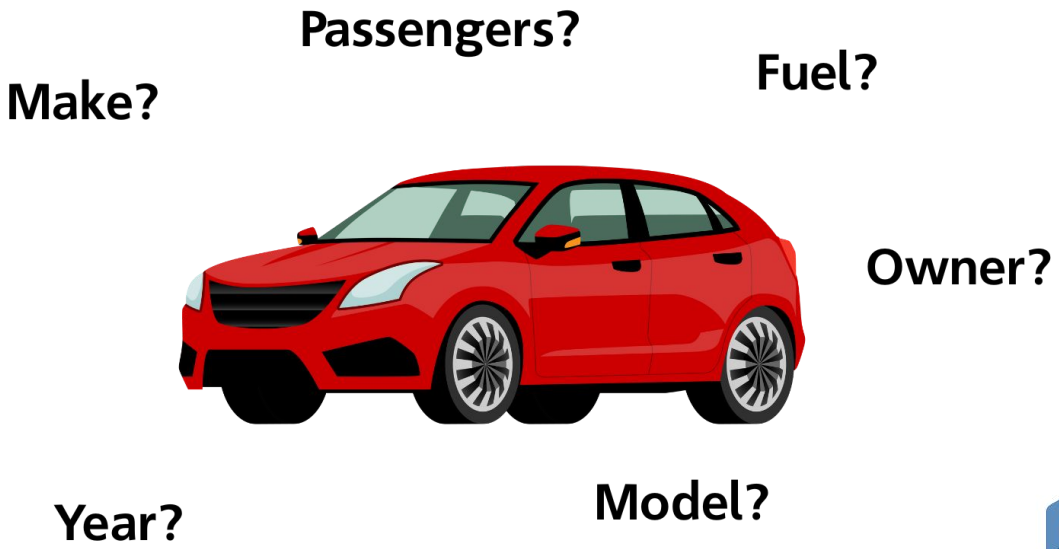
C++ for Developers

Object-oriented programming

How would you describe a car with only one variable?

Object-oriented programming

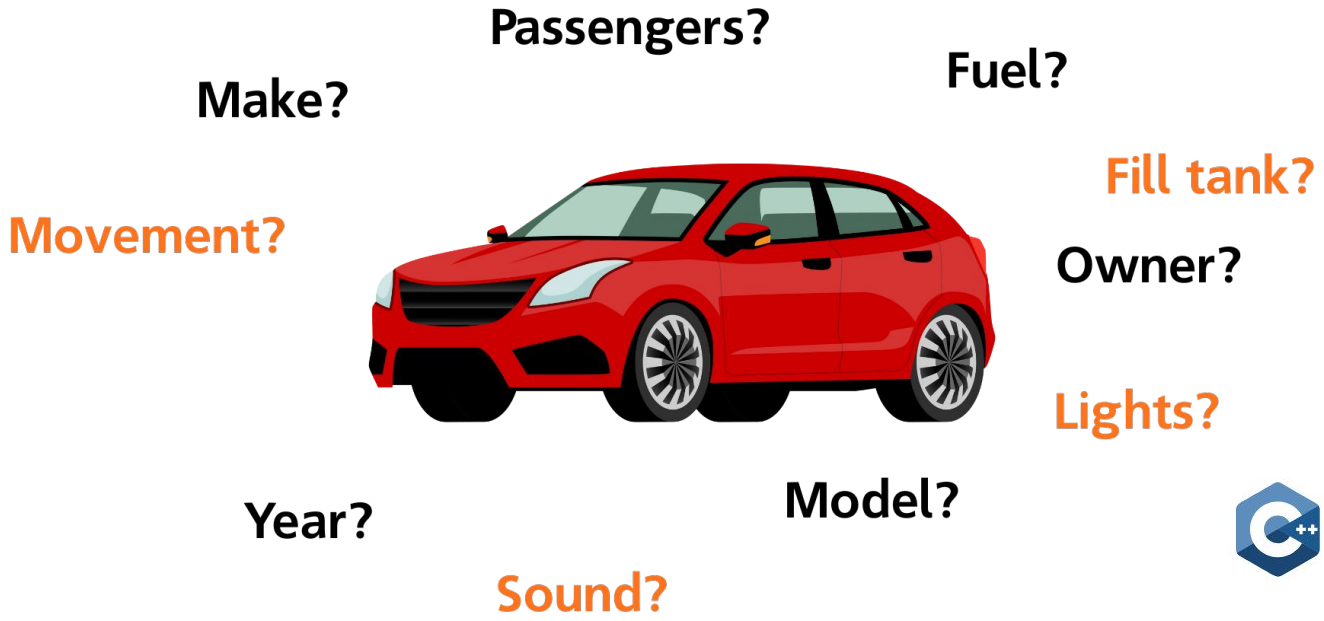
How would you describe a car with only one variable?



lafftale for developers

Object-oriented programming

How would you describe what it does and how it does it?



lafftale for developers

Object-oriented programming

Object-oriented programming is a programming paradigm that organizes code around objects. An object represents a concept or entity, modeled with:

- Attributes (fields/data): what the object is or what information it holds.
- Methods (functions/behavior): what the object can do or how it interacts with other objects.

This way of structuring code makes programs more closely resemble real-life systems. It allows us to model complex entities by describing both their state (data) and their behavior (operations) in a single unit.

User

Attributes

- First name
- Last name
- Address
- Phone number
- Email
- Customer ID
- Loyalty Points

User

Methods

- register()
- changePassword()
- logIn()
- logOut()
- updateLoyaltyPoints()
- changeInformation()

What is a class?

A class is the blueprint that defines what an object is and how it behaves. You can think of it as a detailed plan or sketch: it describes the attributes (the data it holds) and the behaviors (the actions it can perform).

For example, imagine a Car class. The class itself isn't a car you can drive — it's the design. It specifies what data a car should have (like color, model, speed) and what actions it should be able to perform (like drive(), brake(), honk()).

The class doesn't take up memory for those attributes until you create an actual object (an instance) from it. Once you instantiate the Car class into an object, that object has its own color, its own model, and its own speed, but it shares the same behavior defined by the class.

What is an object?

An object is the instantiation of a class. “Instantiate” can be simplified to mean “create,” but it involves more than just making something — it comes with overhead.

When we create an object, we don’t just bring the class into existence; we also allocate memory for its data members, initialize them, and set up any structures required for features like inheritance or polymorphism. The object has access to the methods defined in its class (and inherited classes), but the method code itself is shared by all objects — it’s not duplicated for each one.

So, to phrase it more formally: instantiating a class is the creation of, but not limited to, the object itself. It also includes setting up the necessary context for that object to function according to its class definition.

Constructor

A function that defines how you create an object of that class. This is also possible to overload for creating in different states.

The constructor is implicitly called when instantiating an object.

```
Student(std::string name, std::string course, int age, char grade);
```

```
Student::Student(std::string name, std::string course, int age, char grade) {
    this->name = name;
    this->course = course;
    this->age = age;
    this->grade = grade;
    studentCounter++;
}
```

lafftale for developers

~Destructor

A function that defines how you destroy an object of that class. This can't be overloaded and behaves differently depending on where the object is allocated:

- Stack - Destructor is called implicitly when the object runs out of scope.
- Heap - Using the 'new' keyword, the destructor must be called explicitly with 'delete' - otherwise you cause a memory leak.

```
Student::~Student() {
    studentCounter--;
}
```

```
~Student();
```

lafftale for developers

Declaring and defining a class

Student.h (Declaration)

```cpp
#ifndef OOP_INTRO_STUDENT_CLASS_H
#define OOP_INTRO_STUDENT_CLASS_H

#include <iostream>

class Student {
public:
    std::string name;
    std::string course;
    int age;
    char grade;

    static int studentCounter;

    Student(std::string name, std::string course, int age, char grade);
    ~Student();

    static void printStudentCounter();
    void printStudentInformation();
};

#endif
```

Student.cpp (Definition)

```cpp
#include "Student.h"

int Student::studentCounter = 0;

Student::Student(std::string name, std::string course, int age, char grade) {
    this->name = name;
    this->course = course;
    this->age = age;
    this->grade = grade;
    studentCounter++;
}

Student::~Student() {
    studentCounter--;
}

void Student::printStudentCounter() {
    std::cout << "Current amount of students: " << studentCounter << std::endl;
}

void Student::printStudentInformation() {
    std::cout << "-----------------------------------" << std::endl;
    std::cout << "Student: " << name << std::endl;
    std::cout << "Course: " << course << std::endl;
    std::cout << "Age: " << age << " years old" << std::endl;
    std::cout << "Grade: " << grade << std::endl;
}
```

Instantiation of an object

main.cpp

```cpp
#include <iostream>
#include <vector>

#include "Student.h"

int main(void) {

    Student student1 = Student("Mommo Sadeghi", "C++ for Developers", 23, 'A');
    Student student2 = Student("Maja Wójcik", "C++ for Developers", 27, 'A');

    student1.printStudentInformation();
    student2.printStudentInformation();

    return 0;
}
```

The ‘new’ keyword

The ‘new’ keyword allocates the instantiated object on the heap instead and the variable becomes a pointer to the object.

When allocating on the heap, we need to explicitly delete the object - otherwise we will causa a memory leak.

```
Student* studentOnHeap = new Student("Ronja Voxx", "C++ for developers", 34, 'A');
Student* anotherStudentOnHeap = studentOnHeap;

studentOnHeap->name = "Ami Moreau";

anotherStudentOnHeap->printStudentInformation();

delete studentOnHeap;
```

Access modifiers

By using access modifiers, you control the accessibility of a class's attributes (data members) and methods (member functions).

- **public:**
  Members are accessible from anywhere in the program (inside the class, outside the class, in other files, etc.).
- **protected:**
  Members are accessible within the class itself, and also in derived classes (subclasses), but not from outside.
- **private:**
  Members are accessible only within the class itself. They are not directly accessible from derived classes or outside code.

In C++ all methods and attributes are private until you explicit modify their accessibility

Static

The static keyword declares an attribute or method that belongs to the class itself, not to any individual object created from it.

- Static attributes (fields): There is only one copy of the attribute shared by all objects of the class. It is not duplicated for each object.
  Commonly used for things like counters, shared configuration, or constants.

- Static methods: A method that can be called on the class directly, without creating an object. Since it doesn't act on a specific object, it cannot access non-static attributes or methods of the class (unless it receives an object as a parameter).
  Often used for utility functions, factory methods, or managing shared state.

Comparison operator overloading

We can overload comparators in C++. By doing this we can explicitly choose how we compare the class.

```
friend bool operator==(const Student& s1, const Student& s2);
friend bool operator<(const Student& s1, const Student& s2);
```

```
bool operator==(const Student& s1, const Student& s2) {
    return s1.age == s2.age;
}

bool operator<(const Student& s1, const Student& s2) {
    return s1.age < s2.age;
}
```

```
Student student1 = Student("Mommo Sadeghi", "C++ for Developers", 23, 'A');
Student student2 = Student("Maja Wójcik", "C++ for Developers", 27, 'A');

if (student1 < student2) {
    student1.printStudentInformation();
}
```

The power of OOP

C++ for Developers

Why is OOP so powerful?

The powers of OOP

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Abstraction

C++ for Developers

Abstraction

Managing complexity by hiding intricate details.

- Breaks down the program in smaller and more manageable pieces
- Lowers threshold by keeping it on a “need-to-know” basis
- Code reusability makes the code more maintainable and scalable

Attributes

Characters *

Length

Methods

append()

length()

compare()

replace()

erase()

insert()

empty()

Good use of abstraction

- The class should only model one concept at the time.
  Example: Student is an object, but a collection of Students is also an object.

- Think about what functions should be available outside the class, and what should only be available inside.
  Example: We have no method to control the memory allocation of the string itself, but we can see how big it is.

Encapsulation

C++ for Developers

Encapsulation

Encapsulation means keeping a class's variables hidden inside the class, and controlling how they can be accessed or changed. Instead of letting other code touch the variables directly, we provide member functions (often called getters and setters) to interact with them safely.

Let's look at some code!

Declaration

- Attributes are private
- Declaring getters and setters

What is the difference in the return types of the getters?

```
class Student {
private:
    std::string name;
    std::string course;
    int age;
    char grade;

    static int studentCounter;

public:
    Student(std::string name, std::string course, int age, char grade);
    ~Student();

    static void printStudentCounter();
    void printStudentInformation();

    friend bool operator==(const Student& s1, const Student& s2);
    friend bool operator<(const Student& s1, const Student& s2);

    // GETTERS
    const std::string& getName();
    const std::string& getCourse();
    int& getAge();
    char getGrade();

    // SETTERS
    void setName(const std::string& name);
    void setCourse(const std::string& course);
    void setAge(const int& age);
    void setGrade(const char& grade);
};
```

Definition: Getters

Getters returns the attribute and controls the return type.

- const reference
- reference
- copy

```
const std::string& Student::getName() {
    return this->name;
}

const std::string& Student::getCourse() {
    return this->course;
}

int& Student::getAge() {
    return this->age;
}

char Student::getGrade() {
    return this->grade;
}
```

lafftale for developers

Definition: Setters

Setters set the attributes. Here we validate the data:

- Check the state
- Exception handling
- Standard assignment

```cpp
void Student::setName(const std::string& name) {
    int nameLength = name.length();

    if (name.empty() ||
        nameLength < STUDENT_NAME_MIN_LENGTH ||
        nameLength > STUDENT_NAME_MAX_LENGTH) {
        throw std::invalid_argument("Invalid student name!");
    }

    this->name = name;
}

void Student::setCourse(const std::string& course) {
    int courseNameLength = course.length();

    if (course.empty() ||
        courseNameLength < COURSE_NAME_MIN_LENGTH ||
        courseNameLength > COURSE_NAME_MAX_LENGTH) {
        throw std::invalid_argument("Invalid course name!");
    }

    this->course = course;
}

void Student::setAge(const int& age) {
    if (age < STUDENT_MIN_AGE || age > STUDENT_MAX_AGE) {
        throw std::invalid_argument("Invalid student age");
    }

    this->age = age;
}

void Student::setGrade(const char& grade) {
    if (grade < STUDENT_MAX_GRADE || grade > STUDENT_MIN_GRADE) {
        this->grade = 'F';
    }

    this->grade = grade;
}
```

Using in Constructors

Validate the data when creating an object.

```
Student::Student(std::string name, std::string course, int age, char grade) {
    setName(name);
    setCourse(course);
    setAge(age);
    setGrade(grade);
    studentCounter++;
}
```

Good encapsulation

- Keep all attributes private
- Only allow access through member functions
- Validate the data in the setters
- Control the return type through getters
- Create getters and setters where it makes sense

This makes your code much more protected from errors, both by yourself and your co-workers.


lafftale
for developers