C++ Programming Language Using

OOPS

Vaka Vamsi Krishna
Embedded Software Engineer

NOTE: prerequisite for learning OOPS is knowing Structures in C programming.

INTRODUCTION TO OOPS

What is OOPS?
OOPS stands for Object-Oriented Programming System.

- It's a way of writing computer programs just like we see the real world.
- It is a programming approach where we organize code using objects — real-world entities that combine data and actions together

Main Components of OOPS (Object-Oriented Programming System)

There are 4 main pillars of OOPS.
Let's imagine we're in a school to understand all of them easily! 🏫

---

1. Encapsulation – (Like a School Bag 🎒)

What it means:
Keeping all the things (data + functions) safely inside one unit — like a bag.

School example:
You put your books, pen, lunchbox in your bag.
Only you can open it and use it — others can't touch your stuff without permission!

In Programming:
We put data (like marks, name) and functions (like calculateGrade()) into a class, and protect it using access specifiers.
🌓 **It helps in data hiding and safety.**

---

2. Abstraction – (Like a Mobile Phone 📱)

What it means:
Showing only important things, hiding unnecessary details.

School example:
You use a phone to make calls or play games.
But do you know how the inside circuits work? No — and you don't need to!

In Programming:
We hide complex code inside functions and show only what's needed.

🎯 **It helps to simplify the program.**

---

3. Inheritance – (Like a Child learning from Parents 👨‍👩‍👦)

What it means:
One class inherits (takes) features from another class.

School example:
You may get your father's smile or mother's eyes.
In the same way, a class can inherit properties and behaviors from another.

In Programming:
A class Child can use code from Parent class — like getting functions or variables for free!

🔁 **It helps with reusability.**

---

4. Polymorphism – (One Thing, Many Forms 🎭)

What it means:
The same action can work in different ways.

School example:

- You say "run" 🏃 when you're in a race.
- Your computer "runs" a program 💻
  Same word — different meaning based on where it's used!

In Programming:
A function like draw() might draw a circle, or draw a rectangle, depending on the object.

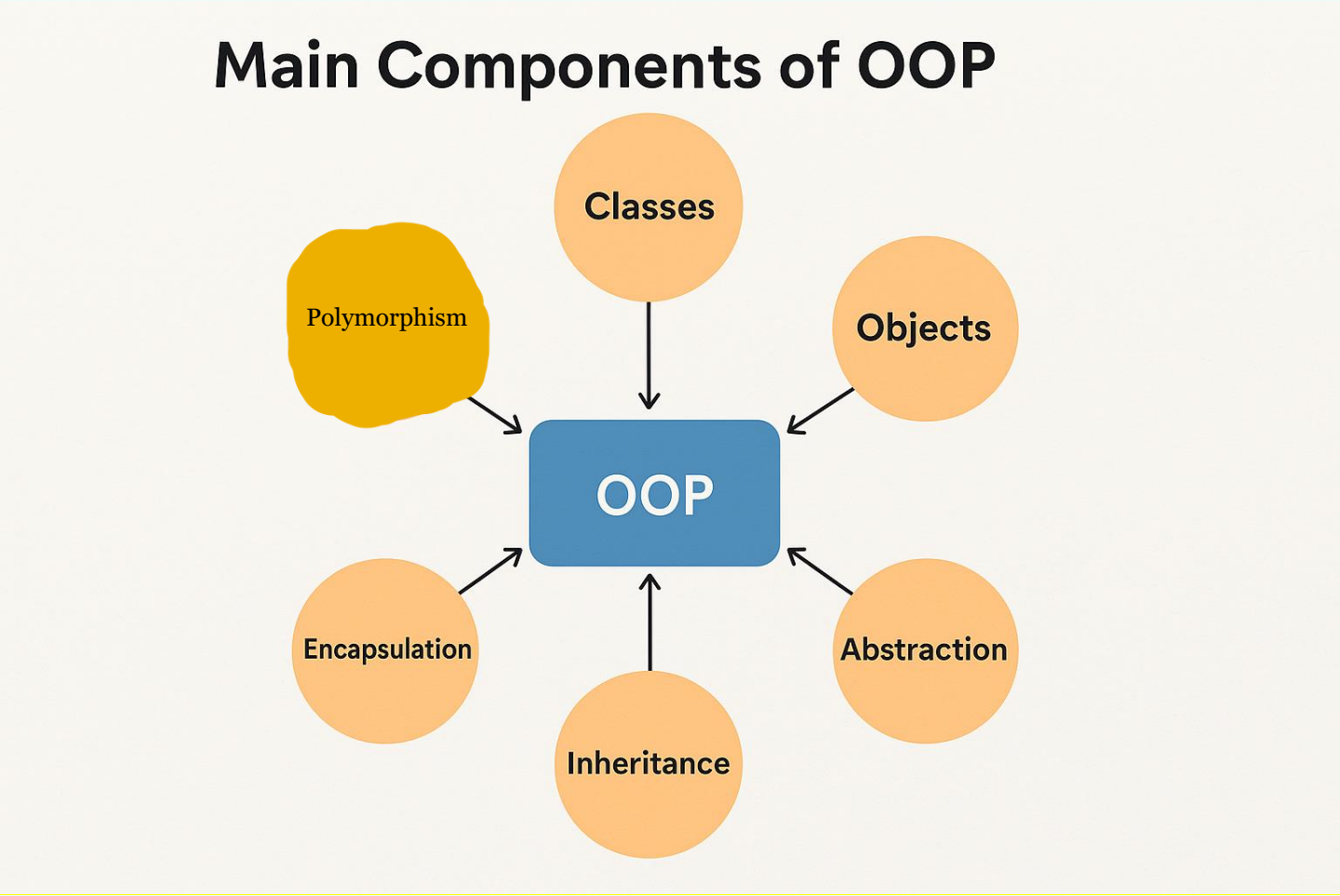
🎨 **It helps with flexibility and variety.**

📘 5. Class – (Like a Blueprint or Form)

What it means:
A class is just a template or design that tells what an object should have.

🧒 6. Object – (Real Item Made from the Class)

What it means:
An object is a real thing made using the class. It has real values and can do actions


Main Components of OOP
Classes
Polymorphism
Objects
OOP
Encapsulation
Abstraction
Inheritance

Classes and Objects

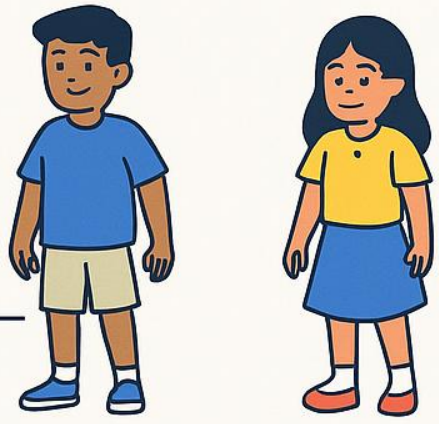
A **class** is like a blueprint or a design.

An **object** is the actual car made using that design.

Super Simple Analogy: School Student

```
class Student {
  public:
   int age;
   string name;
  }
  void study() {
    cout << name ""
    is studying."
  } endl;
  };
Sl.
```

Object = Real Students



Rahul

Priya

Student sl.age = 13;
sl.name = Rahul
sl.study();

Class	Design or Plan
Object	Real thing based on design
Function inside class	Action

What is a Class?

🔹 **A Class is like a blueprint or a design.**

Think of it like a **car design** in a showroom.
The design tells:

- How many wheels the car has
- What color it will be
- What it can do (drive, brake, honk)

BUT... it's **just a design**, not a real car yet.

🚗 **What is an Object?**

🔹 **An Object is the actual car made using that design.**

You can make many cars (objects) from the same blueprint (class).
Each car can have different values:

- Red car with speed 100
- Blue car with speed 80

Define Class: A class is a user-defined data type that allows you to group member variables (data) and member functions (behaviors) together. It supports data hiding and encapsulation using access specifiers like private, public, and protected.

A **class** is a user-defined data type that acts as a **blueprint for objects**. It can contain:

- **Data members (variables)**
- **Member functions (methods)**

📦 What is an Object?

👉 An object is an instance of a class.

That means — we create an object using the class as a reference or blueprint.

✅ **Class and Object Syntax in C++**

🔸 **1. Class Definition Syntax**

```
class ClassName
{
public:
   // Data Members (Variables)
   datatype variable1;
   datatype variable2;

   // Member Functions (Methods)
   void function1()
```

```
    {
        // code
    }

    datatype function2() {
        // code
        return value;
    }
};
```

🔹 2. Creating Objects Syntax

```
int main() {
    ClassName object1;        // Object Creation
    object1.variable1 = value; // Accessing data members
    object1.function1();      // Calling member function
    return 0;
}
```

📌 Example: Class + Object

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

class Student {
public:
    string name;
    int rollNo;

    void display() {
        cout << "Name: " << name << ", Roll No: " << rollNo << endl;
    }
};

int main() {
    Student s1;             // Object creation
    s1.name = "Vamsi";
    s1.rollNo = 101;

    s1.display();           // Method call

    return 0;
}
```

✅ Example: Simple Class and Object

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

class Car
{
public:
    string brand;
    int year;

    void display() {
        cout << "Brand: " << brand << ", Year: " << year << endl;
    }
};

int main() {
    Car car1;  // Creating an object of Car
    car1.brand = "Toyota";
    car1.year = 2020;

    car1.display();  // Accessing class method using object

    return 0;
}
```

🔸 Creating Multiple Objects

```
Car car2;
car2.brand = "BMW";
car2.year = 2023;
car2.display();
```

Practice Questions (Try these yourself!)

1. Create a class Student with data members: name, rollNo, marks. Write a method to display student info. Create 2 objects.

2. Create a class Rectangle with length and breadth. Write a method to calculate and return area. Create an object and print area.

3. Create a class Employee with data members: empID, name, and salary. Write a method to increase salary by a given amount. Create and test it.

4. Create a class BankAccount with data: accountNumber, holderName, balance. Add methods to deposit and withdraw.

5. Create a class Book with members: title, author, and price. Write a method to display book details and create 3 book objects.

Q. Why Do We Need OOPS?

🚲 Real-Life Example: Bicycle Shop

Imagine you're building a game where there are **many bicycles**. Each bicycle has:

- 2 wheels
- A bell
- A brake
- A color
- A speed

In real life, every bicycle is a bit different (color, speed), but they all have similar parts, right?

🛠️ How Procedural Programming (like in C) works

In **C**, you would write separate code for each bicycle:

```
int speed1 = 10;
char color1[10] = "Red";
void brake1() { /* code */ }

int speed2 = 15;
char color2[10] = "Blue";
void brake2() { /* code */ }
```

😩 Repeating the same thing again and again. It's like writing the same recipe for every pizza you bake — waste of time!

🎯 How OOP (like in C++) works

In **C++**, you create a **blueprint** once, and use it again and again.

```
class Bicycle {
public:
   int speed;
   string color;

   void brake()
    {
      // code to slow down
   }
};
```

Now, whenever you want a bicycle:

```
Bicycle bike1;
```

```
bike1.speed = 10;
bike1.color = "Red";

Bicycle bike2;
bike2.speed = 15;
bike2.color = "Blue";
```

✅ No need to write everything again
✅ Easy to make changes
✅ Just like making new pizzas using the same dough and oven 🍕

🧐 **Doubt: Can't we just use struct in C instead of using so many variables and functions like this?**
Let's recall the example:

```
int speed1 = 10;
char color1[10] = "Red";
void brake1() { /* code */ }

int speed2 = 15;
char color2[10] = "Blue";
void brake2() { /* code */ }
```

👆 This is like writing everything again and again for every **bicycle** 🛞.

💡 **Answer: Yes, we can use struct in C — but still, OOP is better. Let's understand with an example.**

🧯 **Real-Life Analogy: School Students**
Imagine you're a **class teacher**. You want to store information about every student:

Name	Age	Marks
Raju	14	80
Priya	13	90

In C, you can create a struct like this:

```
struct Student {
   char name[20];
   int age;
   int marks;
```

```
};
```

Cool! ✅ You've grouped **data**.
But now you want every student to do function like:

- Write Exam
- Play Sports

How will you do that?
🛑 In C: You need to write separate functions and pass the student manually.

```
void writeExam(struct Student s)
{
   // logic
}
```

Still, **functions are separate** from data. 😟 No tight bond.

Struct can support functions, especially in **C++**.
Let me explain this in a clear, layered way:

✅ **Yes — struct Can Have Functions... in C++!**

🔹 **In C:**

```
struct Bicycle
{
   int speed;
   // ❌ You CANNOT define functions inside struct in C
};
```

In **pure C**, struct is **only** for grouping data — no functions allowed **inside**.

🔹 **In C++:**

```
struct Bicycle
{
   int speed;
   void brake() {
      speed = 0;
   }
};
```

👉 In **C++**, struct and class are almost the same. Both can have:

- Data members
- Member functions

So, in **C++**, you can use both. But by convention:

- ✅ Use class for OOP design
- ✅ Use struct for **simple data-only** models (like POD = Plain Old Data)

🧐 What's the Big Deal?

🧪 **Procedural (C)**	🧠 **OOP (C++)**
Do everything step by step	Think in terms of real things (objects)
Hard to manage when program grows	Easy to manage big programs
Data and actions are separate	Data and actions are together
No real-world mapping	Feels like real life!

📌 In Simple Words:

C is like giving instructions for every single step.
C++ is like creating a machine that can follow your instructions again and again.

C (struct)	**C++ (class)**
Only holds data	Holds data **and** actions
Like a notebook	Like a student who can write, read, play
Less organized	Very neat and powerful

✅ So yes, struct is better than writing everything again...
But class is **even better** because it brings everything together (data + actions).

Our greatest glory is not in never failing, but in rising every time we fail.

- Confucius