



Object Oriented programming

Saniya Ashraf

Prerequisites for this class

- CS-110 – Fundamentals of computer programming
 - Which assumes
 1. You know how to turn on a computer
 2. You have used word processing
 3. You can differentiate between a mouse and a keyboard
 4. You have worked on a programming language

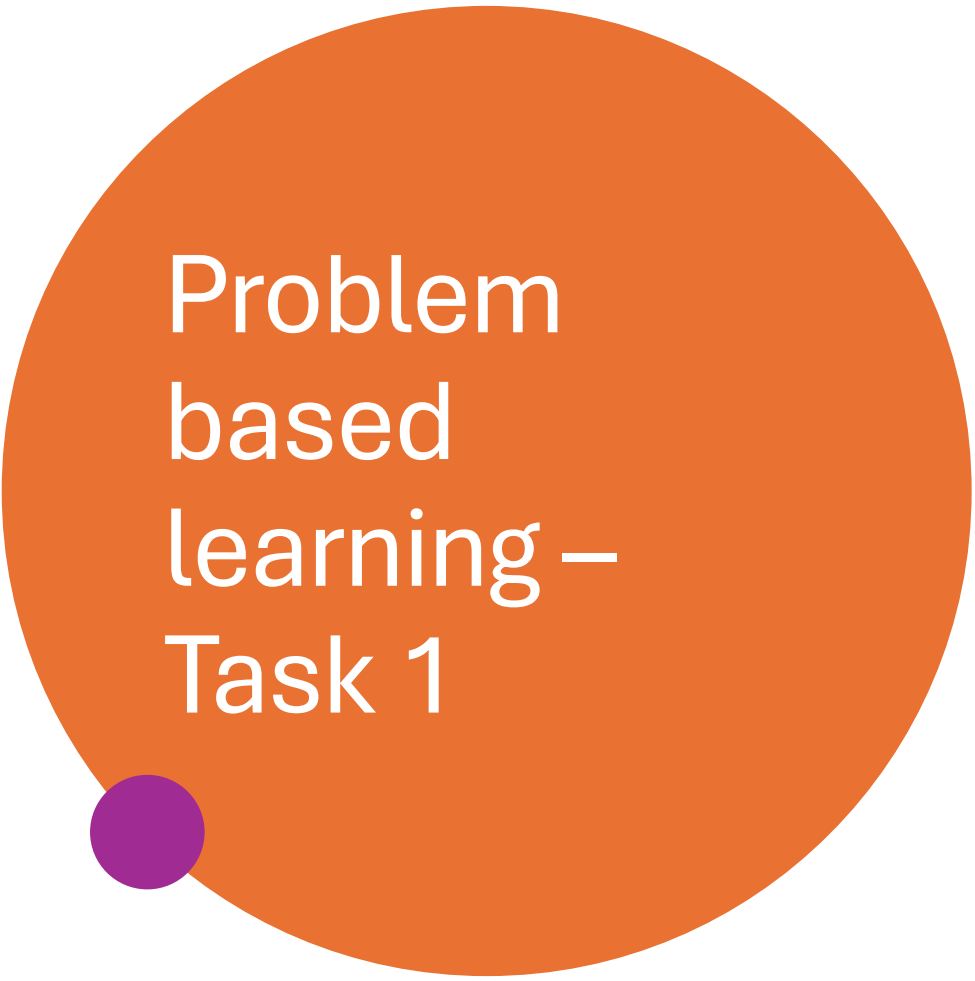


What you will learn throughout this course

- Object oriented philosophy and how it relates to software engineering
- OO in design practices and how it helps developers write good code.
- Understand the difference between procedural and OOP paradigms
- To be able to create and use OOP to map real world scenarios
- Develop programs using OOP techniques

Not all code is GOOD code

Book reference: Thinking in C++, practical programming



Problem based learning – Task 1

- Write an algorithm to bake a cake.
- 

Introduction to Object orientated Programming (OOP)

- Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.
- Why use OOP?
 - OOP is faster and easier to execute
 - OOP provides a clear structure for the programs
 - OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
 - OOP makes it possible to create full reusable applications with less code and shorter development time

Classes

- A class is a set of objects which shares common characteristics/ behavior and common properties/ attributes. It is a user-defined blueprint or prototype from which objects are created. For example, Student is a class while a particular student named Ali is an object.
- Properties of Classes
- Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
- Class does not occupy memory in Java (see JVM's method area for details). It does occupy memory in C++. This memory usage typically includes the size of member variables, any static variables or constants, and any additional overhead required by the compiler for metadata or virtual function tables.
- Class is a group of variables of different data types and a group of methods.
- A Class can contain:
 - Data member
 - Method
 - Constructor
 - Nested Class
 - Interface

```

#include <iostream>

// Class declaration
class MyClass {
public:
    // Member function declaration
    void displayMessage();

private:
    // Member variable declaration
    int myNumber;
};

// Member function definition
void MyClass::displayMessage() {
    std::cout << "Hello from MyClass!" << std::endl;
}

int main() {
    // Creating an instance of MyClass
    MyClass obj;

    // Calling a member function of MyClass
    obj.displayMessage();

    return 0;
}

```

- **MyClass** is a class declaration.
- It has a member function **displayMessage()** declared within the class.
- It has a **private member variable** myNumber.
- Inside main(), an **instance or object** of MyClass named **obj** is created, and its member function displayMessage() is called.
- Some things to note:
 - **void:** This specifies the return type of the function, which in this case is void, indicating that the function does not return any value.
 - **MyClass::** This part indicates that displayMessage() is a member function of the class MyClass.
 - In C++, the default access specifier for members of a class depends on where they are declared:
 - For members declared in a class: If you don't explicitly specify an access specifier (i.e., public:, protected:, or private:), **the default access specifier is private.**

Methods

- A method is a **block of code which only runs when it is called**.
- Methods in C++ are often referred to as **member functions** because they are members of a class. They are declared and defined within the class definition.
- **Access Specifiers:** Methods can be declared with one of three access specifiers: public, protected, or private. These access specifiers determine the visibility of the method to other parts of the program.
- **Invoking Methods:** To invoke a method, you need an instance (object) of the class. You can then use the dot . operator to access the method from the object and call it.
- **Access to Class Members:** Methods have access to all members (variables and functions) of the class, including private members, as long as they are invoked within the class's scope.
- **Passing Arguments:** Methods can accept arguments, just like regular functions. These arguments can be used within the method's body to perform operations.
- **Return Values:** Methods can return values, just like regular functions. The return type is specified before the method name. Why use methods? To reuse code: define the code once, and use it many times.

1. Modifier or access specifiers.

It defines the access type of the method i.e. from where it can be accessed in your application.

- In c++ there 3 types of **access specifiers**.
 - **public**: It is accessible in all classes in your application.
 - **protected**: It is accessible within the class in which it is defined and in its subclass/es
 - **private**: It is accessible only within the class in which it is defined.
 -

2. The return type

- The data type of the value returned by the method or void if does not return a value. It is **Mandatory** in syntax.

3. Method Name

- the rules for field names apply to method names as well, but the convention is a little different. It is **Mandatory** in syntax.

4. Parameter list

- Comma-separated list of the input parameters is defined, preceded by their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses (). It is **Optional** in syntax.

5. Exception list

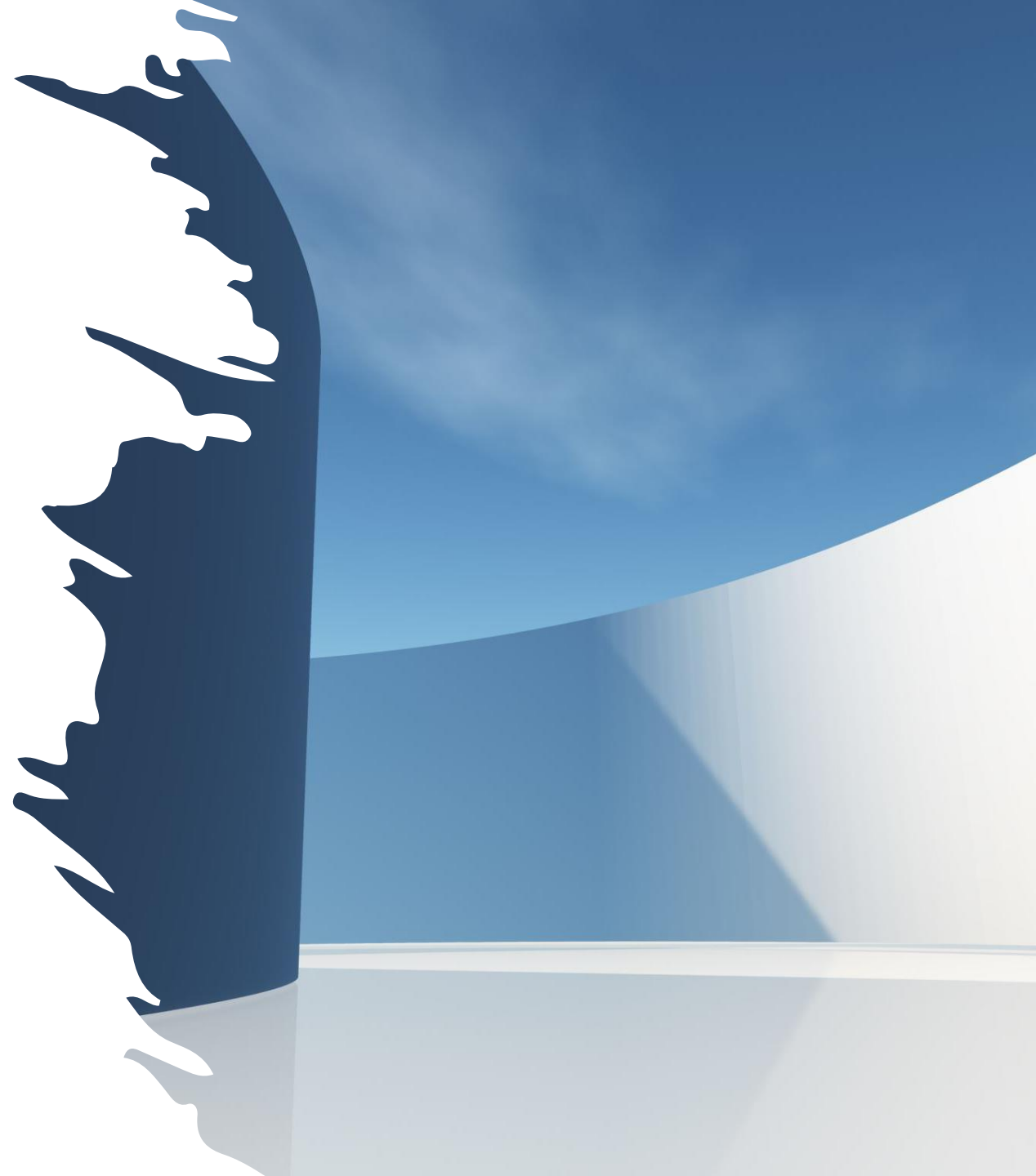
- In C++, an exception specification or exception list in a method declaration specifies the types of exceptions that the method can throw. However, it's important to note that exception specifications have been deprecated since C++11 due to various issues, and they have been removed in C++20.

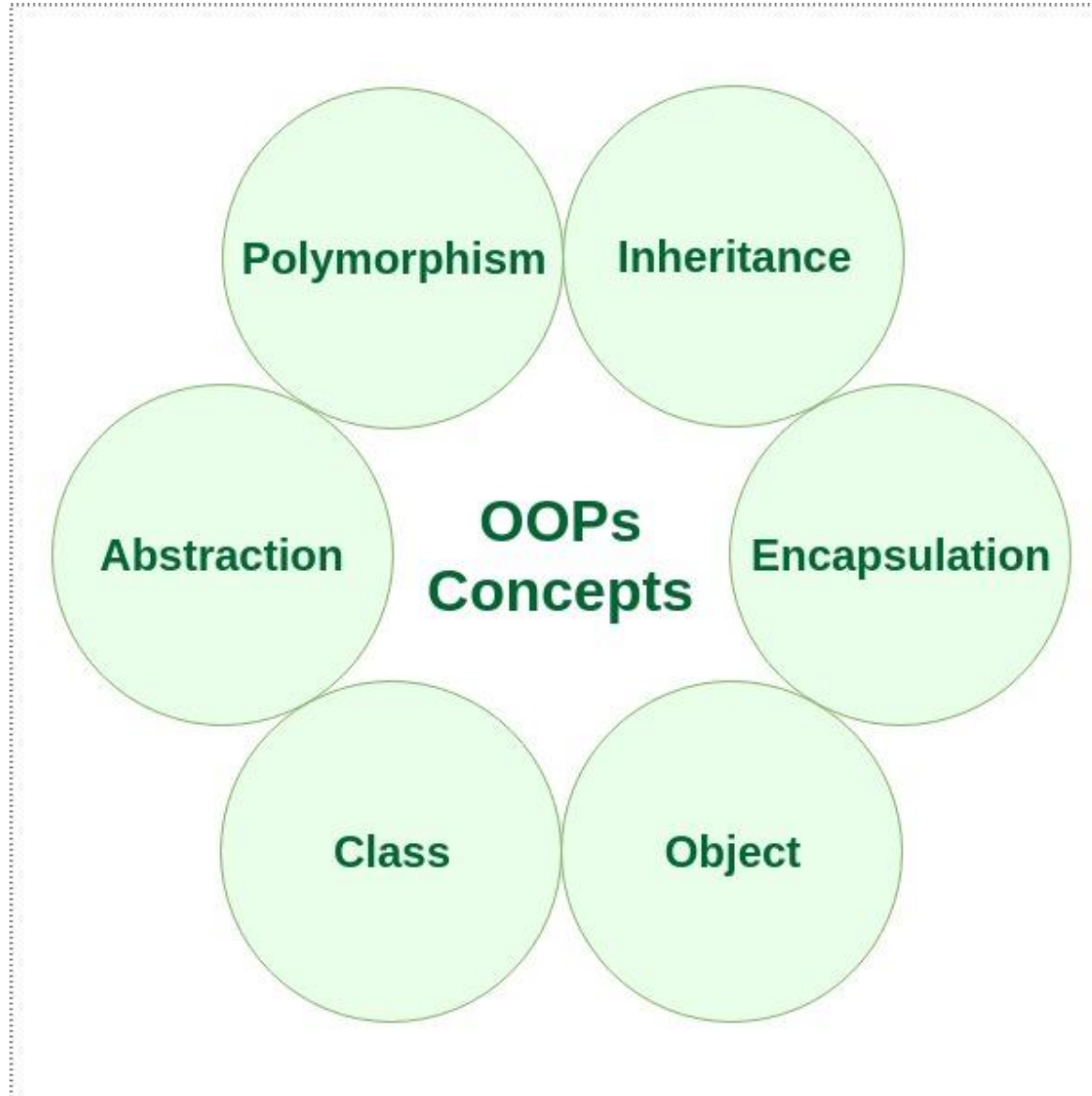
6. Method body

- it is enclosed between braces. The code you need to be executed to perform your intended operations. It is **Optional** in syntax.

Basic Concepts of OOP

Week 1- lecture 2

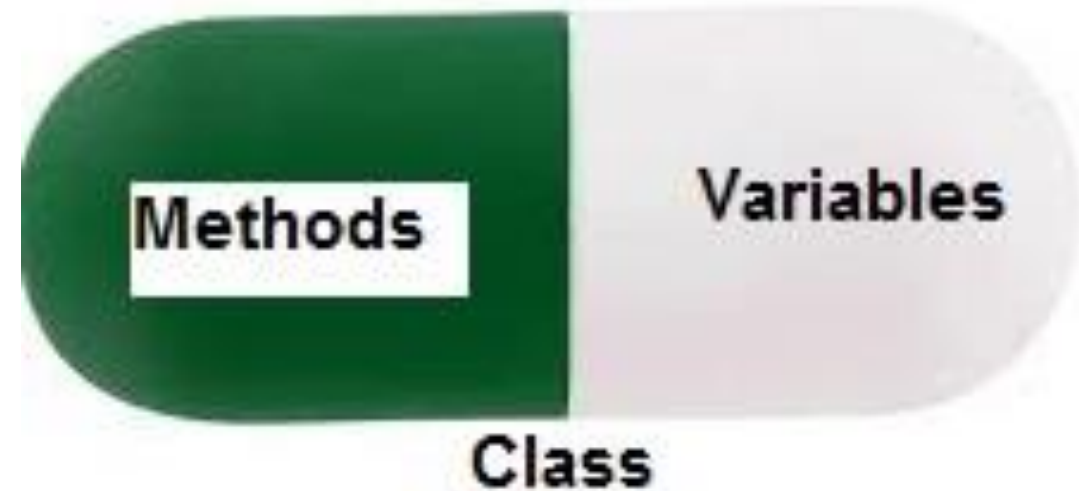




Encapsulation

- Encapsulation is defined as binding together the data and the functions that manipulate that data
- Encapsulation leads to **data abstraction** or **data hiding**

Encapsulation in C++



Abstraction

displaying only essential information
and hiding the background details or
implementation.



Types of Abstraction:

Data abstraction – This type only shows the required information about the data and hides the unnecessary data.

Control Abstraction – This type only shows the required information about the implementation and hides unnecessary information.



Abstraction



Abstraction using Classes

The class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to the outside world and which is not.



Abstraction in Header files

One more type of abstraction in C++ can be header files. Math.h, stdio.h example

Polymorphism - having many forms

- the ability of a message to be displayed in more than one form
- Example of a person
- C++ supports operator overloading and function overloading.
 - **Operator Overloading:** The process of making an operator exhibit different behaviors in different instances is known as operator overloading.
 - **Function Overloading:** Function overloading is using a single function name to perform different types of tasks.



```
int main( )
```

```
{
```

```
    sum1 = sum(20,30);
```

```
    sum2 = sum(20,30,40);
```

```
}
```



```
int sum(int a,int b)
```

```
{
```

```
    return (a+b);
```

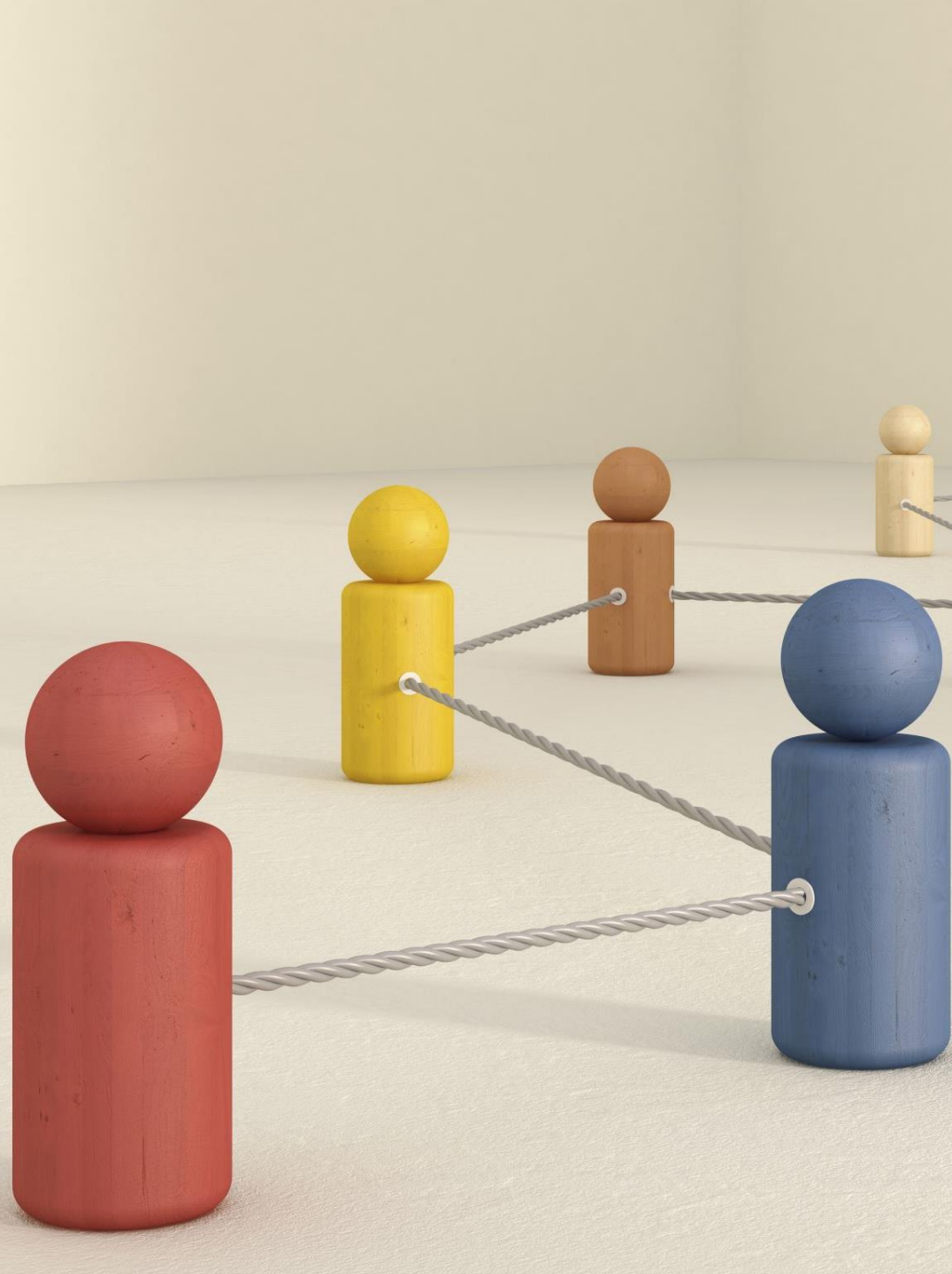
```
}
```

```
int sum(int a,int b,int c)
```

```
{
```

```
    return (a+b+c);
```

```
}
```

Inheritance

- In C++, inheritance is a mechanism where a class (called the derived class or subclass) can inherit properties and behaviors from another class (called the base class or superclass). This allows for code reuse and facilitates the creation of hierarchical relationships between classes.
- In C++, inheritance supports the **"is-a"** relationship, meaning that a derived class "is a" type of its base class.

With Inheritance

