Abstraction in C++

# Abstraction is one of the fundamental concepts of Object-Oriented Programming (OOP). It allows you to hide the complexity of a system and show only the necessary features to the user. In C++, abstraction is implemented using abstract classes and pure virtual functions.

# Key Concepts of Abstraction in C++:

1. Abstract Class:  
- A class that contains at least one pure virtual function.  
- Cannot be instantiated directly.  
  
2. Pure Virtual Function:  
- A function that is declared in an abstract class but has no definition.  
- It is denoted by '= 0' at the end of the function declaration.

# Syntax of Abstract Class in C++:

class AbstractClass {  
public:  
 virtual void abstractMethod() = 0; // Pure virtual function  
 void normalMethod() {  
 // A normal method can also be part of the abstract class  
 std::cout << "Normal method" << std::endl;  
 }  
};

# Example of Abstraction in C++:

#include <iostream>  
using namespace std;  
  
// Abstract class  
class Shape {  
public:  
 // Pure virtual function  
 virtual void draw() = 0;  
  
 // Regular method  
 void info() {  
 cout << "This is a shape." << endl;  
 }  
};  
  
// Derived class: Circle  
class Circle : public Shape {  
public:  
 void draw() {  
 cout << "Drawing a circle." << endl;  
 }  
};  
  
// Derived class: Rectangle  
class Rectangle : public Shape {  
public:  
 void draw() {  
 cout << "Drawing a rectangle." << endl;  
 }  
};  
  
int main() {  
 // Shape s; // Error: cannot instantiate an abstract class  
 Circle c;  
 Rectangle r;  
  
 c.draw(); // Output: Drawing a circle.  
 r.draw(); // Output: Drawing a rectangle.  
   
 c.info(); // Output: This is a shape.  
 r.info(); // Output: This is a shape.  
  
 return 0;  
}

# Explanation of the Example:

1. Abstract Class `Shape`:  
- `Shape` contains the pure virtual function `draw()`, which means every class derived from `Shape` must provide an implementation for this function.  
- It also contains a normal function `info()` which can be inherited directly.  
  
2. Derived Classes `Circle` and `Rectangle`:  
- Both `Circle` and `Rectangle` are derived from `Shape` and implement the `draw()` method, providing specific behavior for each shape.  
  
3. Instantiation and Usage:  
- You cannot create an object of the `Shape` class directly because it is abstract (i.e., contains a pure virtual function).  
- You can create objects of the derived classes (`Circle` and `Rectangle`), which provide concrete implementations of the `draw()` method.

# Why Use Abstraction?

1. Simplifies Code: You only need to deal with the essential features and hide the complex implementation details.  
2. Code Reusability: Abstraction promotes the reusability of code by allowing common behaviors to be defined in base classes and extended in derived classes.  
3. Better Maintainability: By focusing on high-level functionality, you can more easily maintain and extend your codebase.

# Real-World Analogy:

Imagine a car:  
- Abstraction: You drive the car without needing to understand how the engine works.  
- The car’s interface (accelerator, steering wheel, etc.) provides you with a simplified way to interact with the car, while the complex internals (engine, transmission, etc.) are hidden from you.

# Conclusion:

Abstraction allows you to define the interface of a class (what it does) while hiding its implementation details (how it works).  
It is implemented in C++ using abstract classes and pure virtual functions.