Virtual Member Functions

In this lesson, we'll be learning about a very important concept of polymorphism, i.e., Virtual member.

WE'LL COVER THE FOLLOWING

> **Virtual** means existing in appearance but not in reality.

Definition

A `virtual` function is a member function which is declared within the base class and is *overridden* by the derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for the function call. They are mainly used to achieve Runtime polymorphism. Functions are declared with a `virtual` keyword in a base class. The function resolution call is done at run-time.

Why Do We Need a Virtual Function?

Suppose you have a number of objects of different classes like in this case, we have multiple shapes classes. You want to put them all in an array and perform a particular operation on them using the same function call. Given an example, we want to access the same function `getArea()` from multiple child classes.

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


// A simple Shape interface which provides a method to get the Shape's area
class Shape {
  public:
  virtual float getArea(){}
};

// A Rectangle is a Shape with a specific width and height
class Rectangle : public Shape {   // derived form Shape class
  private:
  float width;
  float height;

  public:
  Rectangle(float wid, float heigh) {
    width = wid;
    height = heigh;
  }
  float getArea(){
    return width * height;
  }
};

// A Circle is a Shape with a specific radius
class Circle : public Shape {
  private:
  float radius;

  public:
  Circle(float rad){
    radius = rad;
  }
  float getArea(){
    return 3.14159f * radius * radius;
  }
};

int main() {
  Rectangle r(2, 6);    // Creating Rectangle object
  Shape* shape = &r;   // Referencing Shape class to Rectangle object

  cout << "Calling Rectangle from shape pointer: " <<  shape->getArea() << endl; // Calls sha

  Circle c(5);    // Creating Circle object
  shape = &c;   // Referencing Shape class to Circle object

   cout << "Calling Circle from shape pointer: " <<shape->getArea() << endl;

}
```

Explanation

we have seen earlier when we're trying to print the child class function

`getArea()` by referencing a parent class pointer, it gave us an error. Just by writing the keyword `virtual` we can reference a parent class pointer to child class object.

In the next lesson, we'll be learning about the `Pure Virtual` functions.