Why Do We Need the Object-oriented Programming

Zhihui Yin
3359839025@qq.com

Abstract

As the amount of code continues to expand, programmers encounter more and more problems in the process of process-oriented programming, and a new programming method is needed - object-oriented programming. Object-oriented programming is to compose a set of data structures and methods to process them, classify objects with the same behavior into classes, hide internal details through encapsulation, generalize classes through inheritance, and implement dynamic assignment based on object types through polymorphism. It has the advantages of rationally arranging the accessibility of data and reducing the coupling between different parts of the program, thereby improving the efficiency of code augmentation, code modification, and code reuse.

Keywords: Data, Methods, Objects, Classes, Encapsulation, Inheritance, Polymorphism

ACM Reference Format:
Zhihui Yin. 2023. Why Do We Need the Object-oriented Programming. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Before the advent of object-oriented programming, most programmers used process-oriented programming, or structured programming. But structured programming can be difficult to understand and maintain as it scales up; not conducive to modification and expansion; Not conducive to the reuse of code. As the scale of software continues to expand, it is becoming more and more difficult for structured programming to adapt to the needs of software development.In order to solve the above problems, OOP has the following advantages: object-oriented programming tightly connects data and algorithms that manipulate data through abstraction and encapsulation. In this way, the accessibility of data can be reasonably arranged, and the coupling between different parts of the program can be reduced, thereby improving the efficiency of code augmentation, code modification, and code reuse. OOP uses "inheritance" to easily extend new functions and features.

In the second section, I'll show you the disadvantages of structured programming by describing a piece of game code in C. In the third section, you can see how C++ OOP overcomes these shortcomings.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 An instance of a game program fragment

In this section, I'm going to show you the disadvantages of structured programming in C for game development. Let's take the game "Might and Magic" as an example.

There are all kinds of monsters in the game "Might and Magic", such as knights, angels, and many more. Each monster has two attributes: vitality and attack power. Monsters are able to attack each other, and the attacked person will be injured and fight back. Monsters have corresponding actions when they actively attack, are attacked, and counterattack.

In response to these requirements, if you write in C, you will have to write a separate struct for each monster, and write a separate global function for each struct, such as attack, counterattack, and damage. In fact, if the game has n monsters, you have to write n(n-1) attack functions, n damage functions, and n(n-1) counterattack functions - even if they are basically the same. And when the game version is upgraded and new monsters are added, 2n attack functions and counterattack functions will be added. This is a huge amount of work in real life.

3 The results and methods of OOP to solve the above problems

I'm going to show you how OOP solves these problems in C++ in this section. It also introduces the tools for OOP to solve problems. Using OOP to complete the scenario in Section 2 results are as follows.

```
class CCreature {
protected:
  int lifeValue, power;
public:
  virtual void Attack(CCreature* p) {};
  virtual void Hurted(int nPower) {};
  virtual void FightBack(CCreature* p) {};
};
class CDraon :public CCreature {
public:
  virtual void Attack(CCreature* p) {
    p->Hurted(power);
```

```
    p->FightBack(this);
  }
  virtual void Hurted(int nPower) {
    lifeValue -= nPower;
  }
  virtual void FightBack(CCreature* p) {
    p->Hurted(power / 2);
  }
};
```

3.1 What tools does C++ need to make it?

In the above, I have used some of the following tools.

Class: On the basis of structs, it supports member functions, methods that represent objects. And the member functions of a class can call each other, and the member functions of the class can be overloaded and the default value of the parameters can also be set. The "class" in the sample code is the keyword that defines the class.

Inheritance and derivation: When defining a new class B, if you find that it has all the features of the written class A, and there are other features that class A does not have, you don't have to write class B from scratch, but use A as the base class, and expand and modify it to get B on top of that. This can be said to be the inheritance of class A by B. The CDragon class in the sample code is derived from CCreature.

Virtual function: A virtual member function is preceded by a declaration.

Polymorphism: The address of a derived class object can be assigned to a base class pointer. If you call a virtual function statement with the same name and the same parameters that both the base class and the derived class have through the base class pointer, it is not determined which virtual function will be executed during compilation. When the program runs to the statement, if the base class pointer points to a base class object, the virtual function of the base class is executed, and if it points to the derived class, the virtual function of the derived class is executed. This mechanism is called "polymorphism". For example, in the example, CCreture defines three virtual functions: attack, damage, and counterattack, so that different subclasses do not need to write their own functions when interacting, but can directly call the corresponding functions of the required object during compilation through the base class pointer parameters.

3.2 The advantages of object-oriented programming

We can see that in the above writing, we can clearly see that the code is more coupled, the data and the operations related to it are closely linked, the code is reused, and only the base class or a derived class can be modified when it needs to be modified. For example, when we need to add a new monster, we only need to inherit from the parent class, and we don't need to add new functions for other monsters, that is, other classes don't need to be modified at all, so of course, the scalability of the program is greatly improved. In fact, even without taking extensibility into account, the program itself is much more streamlined than the structure-oriented C language.

4 Conclusion

Through its unique tools and concepts, object-oriented programming solves the problems that cannot be solved by process-oriented programming due to the trend of larger code sizes. The code becomes easy to understand and maintain, and the extensibility of the code itself has been greatly improved, reducing the workload of programmers and greatly liberating productivity. So we really need object-oriented programming.