第四章 类与对象

4.1 面向对象程序设计的基本特点

4.2 类和对象

4.3 构造函数和析构函数

4.4 类的组合

面向对象的4个特征

- 抽象
- 封装
- 继承
- 多态

4.1.1 抽象

- 抽象是对具体对象（问题）进行概括，抽出这一类对象的公共性质并加以描述。
  - 注意问题的本质及描述，忽略细节。
  - 数据抽象：描述某类对象的属性或状态。
  - 代码抽象：描述某类对象的共有的行为或功能
  - 抽象的实现：通过类的声明。

抽象实例——钟表

■数据抽象：

- int **hour**,int **minute**,int **second**

■代码抽象：

- **setTime(),showTime()**

4.1.2 封装

- 将抽象出的数据成员、代码成员相结合，将它们视为一个整体。
  - 目的是增强安全性和简化编程，忽略实现细节，而只需要通过外部接口，以特定的访问权限，来使用类的成员。

4.1.2 封装（续）

- 实例：

```
class  Clock
 {
     public:
         void setTime(int h, int m, int s);
         void showTime();
     private:
          int hour, minute, second;
};
```

外部接口

特定的访问权限

4.1.3 继承

- 是C++中支持层次分类的一种机制，
  - 继承允许从现有的类（基类）建立新类
  - 派生类继承了基类的属性和行为
  - 派生类可以修改和扩充这些属性和行为
  - 派生类可以增加新的数据成员和成员函数
- 实现：派生类——见第7章

4.1.4 多态

- 多态：同一名称，不同的功能实现方式。
- 目的：达到行为标识统一，减少程序中标识符的个数。
- 实现：重载函数和虚函数——见第8章

4.2 类和对象

- 类是具有相同**属性**和**行为**的一组对象的集合，是抽象描述，包括属性和行为两个主要部分。
- 类可以实现数据的封装、隐藏、继承与派生。
- 利用类易于编写大型复杂程序，其模块化程度比C中采用函数更高。

4.2.1 类的定义

类是一种用户自定义类型，声明形式：

```
class 类名称
{
   public:
          公有成员（外部接口）
   private:
          私有成员
   protected:
          保护型成员
}
```

4.2.3 对象

- 对象是该类的某一特定实体，即类型的变量。
- 声明形式：
  - 类名 对象名；
  - 例：Clock myClock;
- 类中成员互访
  - 直接使用成员名
- 类外访问
  - “对象名.成员名”方式访问 public 属性的成员

对象和类

■对象是具体实体，实例

- 每个学生、每本书—具体的实体
- 刘顺顺、郑发是不是人？
- “人”不是具体的实体,是类,是抽象概念

■类是某些对象共同特征的抽象

- 类是对象的抽象，对象是类的实例

如何区分类和对象

■以“人”和“张一帆、张一航”为例

- “人”描述了所有人（包括“张一帆”）都具有的属性和行为，如有姓名、身高、体重等等
- 类（“人”）描述的概念是抽象的

  “人”的姓名是什么？

  “人”的身高是什么？
- 对象（“张一航”）是具体的

  性别是“男”

■还有哪些类和对象的例子

- 教室—104教室
- 国家—中国
- 学校—石家庄铁道大学
- ……

4.2.4 类的成员函数

- 成员函数是函数的一种
- 和普通函数基本一样
- 可以inline、重载、带默认形参值
- 区别：成员函数属于类，可以被限定符修饰
- 如果一个类没有成员函数，就等同于结构体
- 成员函数主要提供外部接口

4.2.4 类的成员函数

- 在类中说明原型，可以在类外给出函数体实现，并在函数名前使用类名加以限定。也可以直接在类中给出函数体，形成内联成员函数。

- 允许声明重载函数和带默认形参值的函数

内联成员函数

- 为了提高运行时的效率，对于较简单的函数可以声明为内联形式。
- 内联函数体中不要有复杂结构（如循环语句和switch语句）。
- 在类中声明内联成员函数的方式：
  - 将函数体放在类的声明中。
  - 使用inline关键字。

4.2.5 程序实例——例4-1

```
#include<iostream>
using namespace std;
class Clock{
public:
  void setTime(int h = 0, int m = 0, int s = 0);
  void showTime();
private:
  int hour, minute, second;
}
int main() {
  Clock myClock;
  myClock.setTime(8, 30, 30);
  myClock.showTime();
  return 0;
}
```

运行结果：

8:30:30

例4-1（续）——类的实现

```
void Clock::setTime(int h, int m, int s)
{
   hour = h;
   minute = m;
   second = s;
}
void Clock::showTime()
{
   cout<<hour<<":"<<minute<<":"<< second;
}
```