C++期末不挂科

CH3 - 面向对象1 - 抽象与封装

本章内容

- 面向对象思想
- 抽象与UML图
- 类和对象
- 封装

理解以下名词：

- 面向对象思想
- 抽象
- UML类图
- 成员变量和成员函数
- 实例化
- 构造函数与析构函数
- this指针
- 封装
- 访问控制属性

理解以下问题：

- 为什么从面向过程 -> 面向对象
- 如何对系统内的事务进行抽象分析
- 类和对象有什么区别

熟悉以下题型：

- 给定系统需求或UML类图，编写类定义
- 通过重载的构造函数创建对象
- 编写正确的析构函数以释放对象所持有的资源
- 判断私有成员和公有成员的可见性

面向对象思想

抽象与UML图

类和对象

封装

从面向过程到面向对象

编程实现计算器：

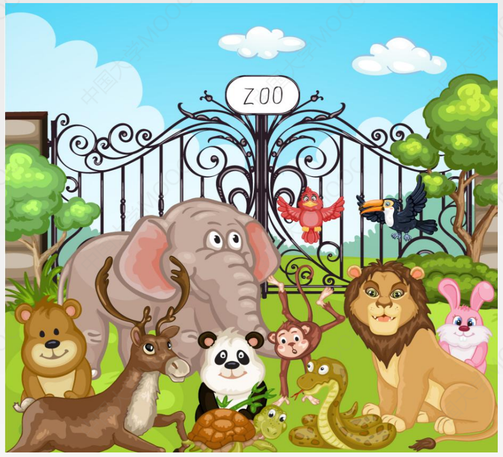
float add(float a, float b);

float sub(float a, float b);

float mul(float a, float b);

float div(float a, float b);

问题/场景/系统越来越复杂...



编程实现动物园模拟：

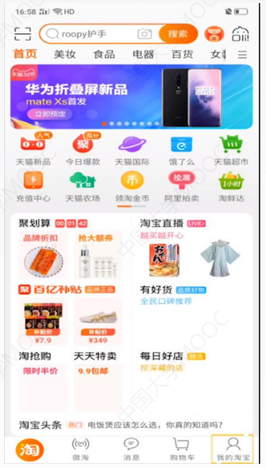
void AeatB(A a, B b);

void AeatC(A a, C c);

void BfightD(B b, D d);

void Esleep(E e);

..... （**很多很多过程**）



实现一个商城系统：

用户登录、购买商品、退换货、

商品上下架、优惠券、库存、商家、

评论..... （**很多很多过程**）

做一个游戏：

有玩家、野怪、地图等实体

玩家又有血条、蓝条等属性

每个实体还能进行攻击、放技能等行为....（**很多很多过程**）

从面向过程到面向对象

- 随着计算机发展，问题场景越来越复杂，面向过程在大型系统开发中捉襟见肘
- 面向对象思想应运而生，核心思想：

既然随着系统参与实体的增多，过程变得复杂，那就不费力描述每一个可能的过程了，转而**描述每一个实体**。如果每一个实体都被正确描述了，那么将这些实体置于系统中，系统就能正确运行。

属性 + 行为

从面向过程到面向对象

- 对于问题：求解不同图形的周长和面积
- 以**面向过程**的思路：

按流程走

(1) 确定图形是什么图形，三角形，正方形，圆形？...

(2) 获取计算所需要的信息：

for ▲： 底和高

for ■： 边长

for ●： 半径

...

(3) 根据不同的计算公式进行计算

(4) 得到结果

从面向过程到面向对象

- 对于问题：求解不同图形的周长和面积
- 以**面向对象**的思路：

属性：

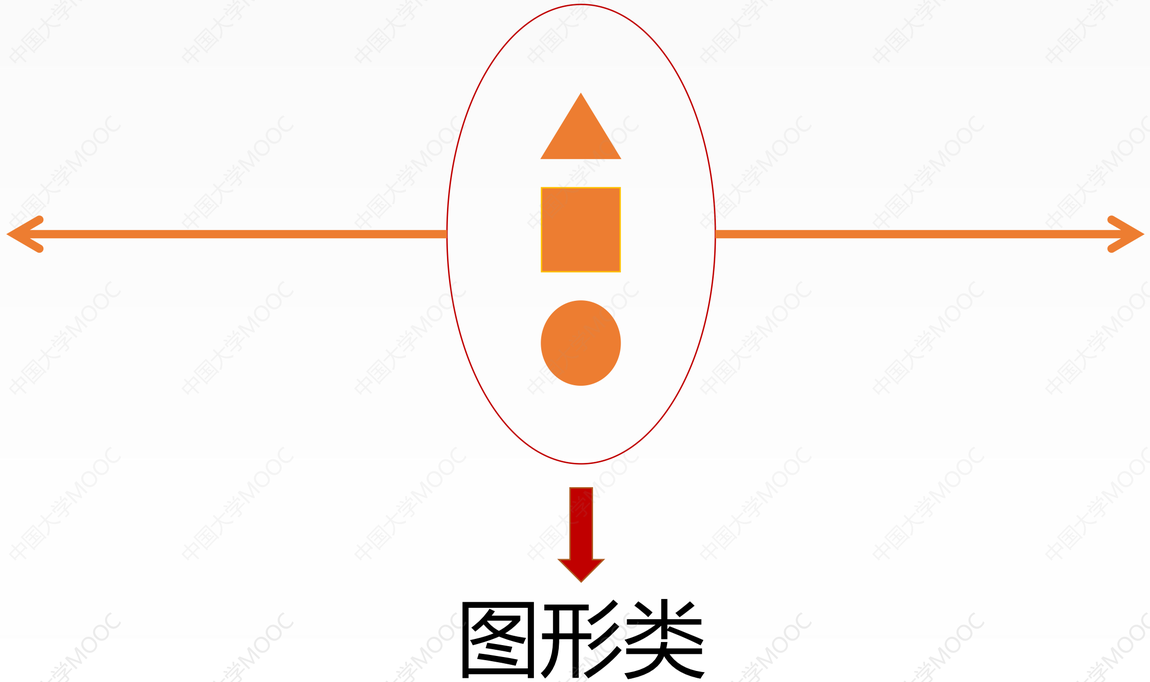
边长

半径

高

角度

…



行为：

求面积

求周长

缩放

错切

旋转

…

面向对象思想

抽象与UML图

类和对象

封装

抽象 Abstract

图书馆管理系统：

图书

-属性：名字，编号...

-方法：借出，还入...

读者

-属性：姓名，身份证号...

-方法：注册，还欠费...

学生信息管理系统：

学生

-属性：名字，学号...

-方法：查成绩，查课表...

老师

-属性：姓名，院系...

-方法：登成绩，上线课程...

游戏：

-属性：ID，血量，位置...

-方法：吃药，移动，开枪...

-属性：名称，伤害，可装配件...

-方法：拾起，丢弃，装弹...

面向对象的四个特征之1：抽象

分析问题，识别出各个实体及其属性和行为

Unitfied Modeling Language

UML类图

- 识别出问题中各个实体(属性+行为)后，需用规范的方式描述

图书馆管理系统：

图书

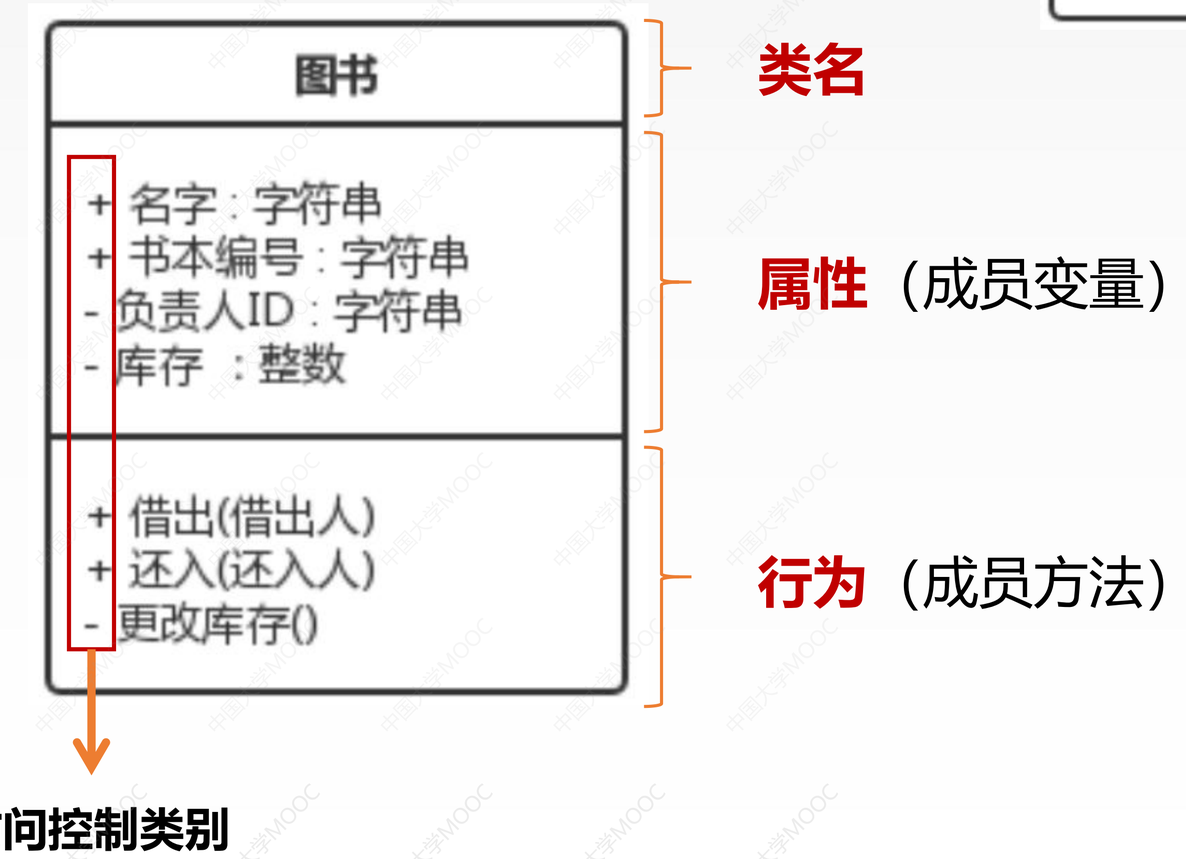
-属性：名字，编号...

-方法：借出，还入...

读者

-属性：姓名，身份证号...

-方法：注册，还欠费...



面向对象思想

抽象与UML图

类和对象

封装

定义类

- 定义一个类 = 定义它的属性(***成员变量***) + 行为(***成员函数***)

class 类名 {

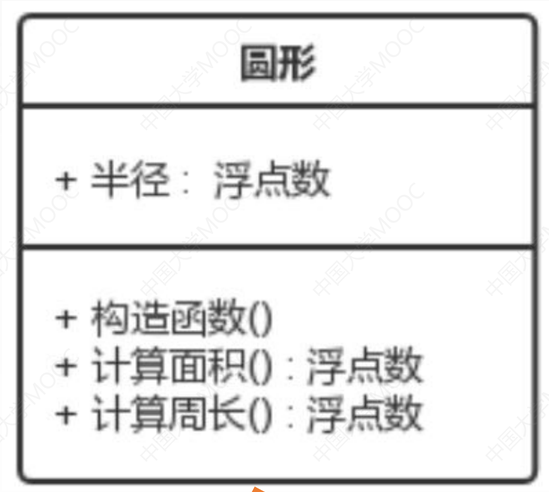
访问控制修饰符：

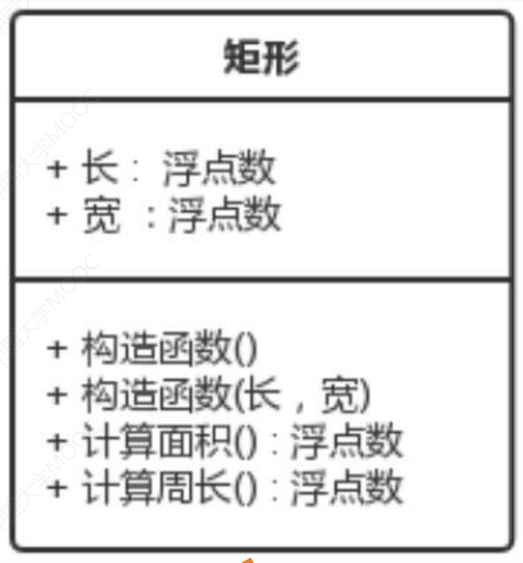
定义成员变量

定义成员函数

};

不要忘了结尾的分号





根据UML图完成类定义

结构体 vs 类

- 简单理解：结构体 + 行为(成员函数) = 类
- 事实上C++中也支持结构体定义成员方法，两者并无本质区别了
- 根据使用场景选择结构体或类：
    - *结构体：主要记录数据，极少行为（如资源配置信息、网络连接信息等）*
    - *类：既有属性也有行为（如学生类、用户类、玩家类等）*

类 vs 对象

- 类 ---*实例化*---> 对象

```
class MyClass {
public:
    int n;
    MyClass(int n) {
        this->n = n;
    }
};
```

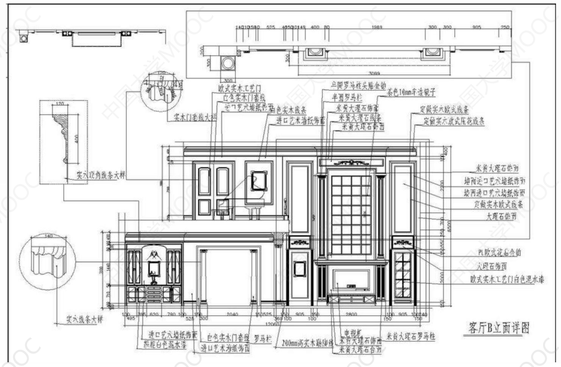
类定义

实例化

存在于内存中

```
MyClass my_object(10);
```

真实存在的对象



设计图纸

起楼

真实存在的楼

constructor　　destructor

特殊的成员函数：构造函数与析构函数

- 构造函数和析构函数是两种特殊的类成员函数
- **构造**：对象实例化时，在分配得到的空间上构造对象（如**初始化**成员变量、**分配资源**等）
  - *默认构造函数：没有参数的构造函数*
  - *有参构造函数：有参数的构造函数*
- **析构**：对象生命周期结束时，回收空间前，完成对象的清理工作（如**释放资源**等）
- 构造函数和析构函数都没有返回值！
- 析构函数没有参数！

constructor destructor

特殊的成员函数：构造函数与析构函数

- **构造**：对象实例化时，分配空间后，完成对象的构造工作（如初始化成员变量、分配资源等）
- **析构**：对象生命周期结束时，回收空间前，完成对象的清理工作（如释放资源等）

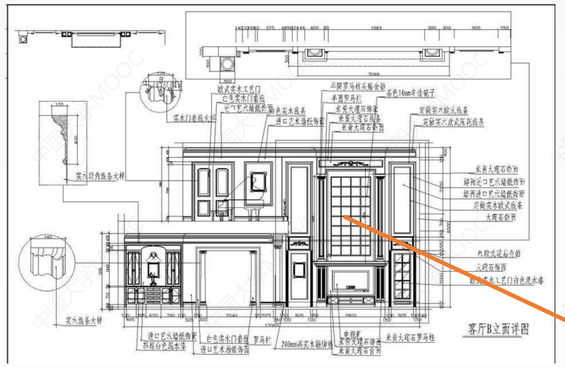
```
class A {
public:
    int n;
    char* data = nullptr;
    A(int n) {
        this->n = n;
        data = (char*)malloc(100);
    }
    ~A() {
        free(data);
    }
};
```

(有参)构造函数

析构函数

this 指针

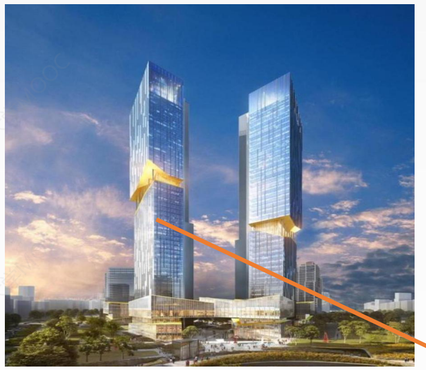
- this的中文含义：这、这个、当前这个
- this指针在类定义内部使用，指向当前对象



设计图纸

这栋楼要有100层
这栋楼是白色的
这栋楼要能发光
……

实例化



真实存在的A大厦

A大厦有100层
A大厦是白色的
A大厦能发光
……

面向对象思想

抽象与UML图

类和对象

封装

面向对象的四个特征之2：封装

封装 Encapsulate

- 封装：将类的一些成员变量或方法藏起来，不允许**外界直接**操作
- 不允许直接操作 ≠ 不允许操作，而是通过自定义的特定方法操作

访问控制属性

public 公有

访问控制属性为 public 的成员

外部可以直接通过 对象.名字 访问

```
A a;
a.xxx = 10;
a.func();
```

protected 保护

private 私有

访问控制属性为 private 的成员

外部**不可**直接通过 对象.名字 访问

```
A a;
✗ a.xxx = 10;
✗ a.func();
```

getter / setter 方法

- 为某些私有成员变量提供外部读写方法：**get_xxx(读) / set_xxx(写)**
- getter 和 setter 一般是 public 的，不然没意义

```
class Book {
private:
    string name;
    int count;
public:
    Book(){…}
    int get_count() {
        return count;
    }
};
```

getter函数的通常格式（设xxx的类型为T）：

```
T   get_xxx() const {
        return xxx;
}
```

常成员函数：不能修改类成员变量

getter / setter 方法

- 为某些私有成员变量提供外部读写方法：**get_xxx(读) / set_xxx(写)**
- getter 和 setter 一般是 public 的，不然没意义

```
class Book {
private:
    string name;
    int count;
public:
    Book(){…}
    void set_name(const string& name) {
        this->name = name;
    }
};
```

setter函数的通常格式（设xxx的类型为T）：

```
void   set_xxx(const T& xxx) {
        this->xxx = xxx;
}
```

☆ Key 1：**抽象**：识别问题/场景/系统中事物的属性与行为

☆ Key 2：UML类图可用于规范化描述一个事物的属性与行为

1. 下列用于描述一个学生类的UML类图中正确的是：B

A.

Student
+ get_name(): string + get_age(): int + grow_up(int years)
+ id: int - name: string - age: int - score: float

B.

Student
+ id: int - name: string - age: int - score: float
+ get_name(): string + get_age(): int + grow_up(int years)

C.

Student
+ id: int - name: string - age: int - score: float + get_name(): string + get_age(): int + grow_up(int years)

D.

Student
+ get_name(): string + get_age(): int + grow_up(int years)

☆ Key 3：类定义主要是定义一个类的成员变量和成员函数

☆ Key 4：使用class关键字进行类定义

1. 根据UML类图，完成C++的类定义：

```
class Student {
private:
        string name;
        int age;
        float score;
public:
        int id;
        int get_age() { return age; }
        float get_score() { return score; }
        void grow_up(int years) { age += years; }
};
```

Student
+ id: int - name: string - age: int - score: float
+ get_name(): string + get_age(): int + grow_up(int years)

感觉缺了什么？

成员变量都没初始值！

☆ Key 5：类是对某一类事物的描述，对象是该事物真实存在的一个实体

1. 下列关于类和对象的叙述中，错误的是：B

A. 类是对某一事物的抽象

B. 一个类只能有一个对象

C. 类和对象的关系类似数据类型与变量的关系

D. 对象在内存中真实存在

⭐ Key 6：**构造函数**是特殊的成员函数，名称为类名，通常目的是初始化对象

⭐ Key 7：①创建对象时自动调用 ②可以有多个重载 ③不可以有返回值！

1. 为Student类补充一个**默认构造函数**和一个**有参构造函数**：

```
Student() {
        id = 1001;
        name = "Mike";
        age = 18;
        score = 85;
}
```

```
Student(int _id, string _name, int _age, float _score) {
        id = _id;
        name = _name;
        age = _age;
        score = _score;
}
```

2. 关于构造函数的叙述正确的是：B

A. 构造函数可以有返回值

B. 构造函数的名字必须与类名完全相同

C. 构造函数必须带有参数

D. 构造函数必须定义，不能默认

⭐ Key 8：**this指针在类成员函数定义内部使用，指向当前对象**

1. 利用this指针编写Student的有参构造函数，避免变量名覆盖问题：

```
Student(int id, string name, int age, float score) {
        this->id = id;
        this->name = name;
        this->age = age;
        this->score = score;
}
```

2. 判断正误：

this指针保证每个对象拥有自己的数据成员，但共享处理这些数据成员的代码。（ √ ）

☆ Key 9：**封装**：控制类成员在外部的可见性

☆ Key 10：public标记公有成员 / private标记私有成员私有

1. 设a是类A的一个对象，则两行代码**皆不会**产生编译错误的是：C

A.
```
a.n = 10;
a.s = “hello”;
```

B.
```
cout << a.get_s();
a.increase_n();
```

C.
```
cout << a.get_n();
string t = a.get_s();
```

D.
```
a.get_n() = 10;
cout << a.get_s();
```

```
class A {
private:
        int n;
        string s;
public:
        A(int n, string s) { this->n = n; this->s = s; }
        string get_s() const { return s; }
        int get_n() const { return n; }
private:
        void increase_n() { n++; }
};
```

⭐ Key 11：getter函数通常会被设置为const函数，setter函数则通常接收const参数

1.给定类A的定义如下，则下列其私有成员变量的getter/setter合理的是：B

```
class A {
private:
        int n;
public:
        A(int n) { this->n = n; }
};
```

中国大学MOOC
搜索：C++不挂科

A.

```
int get_n(const int n) { return n; }
```

B.

```
void set_n(const int n) { this->n = n; }
```

C.

```
void get_n() const { return n; }
```

D.

```
int set_n(const int n) const { this->n = n; }
```