# 组合类例题分析4-4

类的组合，创建组合类的对象的时候，一定会先创建它的内嵌对象，所以一定会先调用内嵌对象所对应的构造函数或者复制构造函数。通过初始化列表来完成。

4-4程序

设计思想

运行结果分析

//4\_4.cpp

#include <iostream>

#include <cmath>

using namespace std;

class Point { //Point类定义

public:

Point(int xx =0 , int yy = 0) {

x = xx;

y = yy;

}

Point(Point &p);

int getX() { return x; }

int getY() { return y; }

private:

int x, y;

};

Point::Point(Point &p) { //拷贝构造函数的实现

x = p.x;

y = p.y;

cout << "Calling the copy constructor of Point" << endl;

}

//类的组合

class Line { //Line类的定义

public: //外部接口

Line(Point &xp1, Point &xp2); // Line(x1,y1,x2,y2)

Line(Line &l);

double getLen() { return len; }

private: //私有数据成员

Point p1, p2; //Point类的对象p1,p2

double len;

};

//组合类的构造函数

Line::Line(Point &xp1, Point &xp2) : p1(xp1), p2(xp2)

{

cout << "Calling constructor of Line" << endl;

double x = static\_cast<double>(p1.getX() - p2.getX()); // (double)(p1.getX() - p2.getX())

double y = static\_cast<double>(p1.getY() - p2.getY());

len = sqrt(x \* x + y \* y);

}

//组合类的拷贝构造函数

Line::Line (Line &l): p1(l.p1), p2(l.p2) {

cout << "Calling the copy constructor of Line" << endl;

len = l.len;

}

//主函数

int main() {

Point myp1(1, 1), myp2(4, 5); //建立Point类的对象

Line line(myp1, myp2); //建立Line类的对象

Line line2(line); //利用拷贝构造函数建立一个新对象

cout << "The length of the line is: ";

cout << line.getLen() << endl;

cout << "The length of the line2 is: ";

cout << line2.getLen() << endl;

return 0;

}

# 类的前向引用声明

当类的定义在后面，要在前面使用的时候，需要先对类名进行声明。

注意事项：

虽然进行了前向引用声明，但是不能定义类的对象。

可以定义这个类的引用，通常作为函数的形参。但是在看到该类的完整定义之前，不能调用它的成员函数。

# ＵＭＬ简介（了解）

UML是什么的缩写？统一建模语言

UML是做什么用的？

设计语言，用来设计系统的组成和结构。

面向对象的设计思想来设计系统，要考虑系统里面有哪些类，类与类之间是什么关系。

用一些图形的方式来展现系统中的类及类与类之间的关系，可以有效的从整体体现出系统的设计思路。

UML中有哪些基本元素？

类图：表现类的信息。

类与类的关系：

1 存在各种不同的关系 **包含（组合） 继承**

2 不同的关系用不同的符号来表示

# Ｃ＋＋中struct关键字

# struct可以像class一样使用，可以用来定义类。

两者的区别：

默认的访问属性

class 私有

struct 公有

# 综合案例-银行账户管理程序

模拟银行账户，模拟账户的创建和日常的工作（存钱，取钱，计算利息）

设计一个账户类，封装必要数据成员和函数成员。

#include <iostream>

using namespace std;

class SavingsAccount{

int id;

double balance;

double rate;

int lastDate; // 记录上一次余额修改的时间

double accumulation; // 记录累计余额

double accumulate(int date){

return accumulation + (date-lastDate)\*balance;

}

void record(int date,double amount){ // 处理业务

// 1 更新 accumulation

accumulation = accumulate(date);

// 2 更新余额

balance += amount;

// 3 打印明细

cout<<"id "<<id<<"\t"<<date<<"\t"<<amount<<"\t"<<balance<<endl;

}

public:

SavingsAccount(int date,int id,double rate)

:id(id),lastDate(date),rate(rate){

balance = 0;

accumulation = 0;

}

void deposit(int date,double amount){

record(date,amount);

}

void withdraw(int date,double amount){

if(amount>balance){

cout<<"余额不足"<<endl;

}else{

record(date,-amount);

}

}

void settle(int date){

accumulation = accumulate(date);

double interest = accumulation\*rate/365;

if (interest!=0)

record(date,interest);

accumulation = 0;

}

};

int main(){

SavingsAccount sa0(1,6678678,0.015);

sa0.deposit(5,5000);

sa0.withdraw(30,1000);

sa0.settle(90);

}